QueryViz: Helping Users Understand SQL Queries and their Patterns

Jonathan Danaparamita
University of Washington, Seattle
jdanaparamita@gmail.com

Wolfgang Gatterbauer
University of Washington, Seattle
gatter@cs.washington.edu

ABSTRACT

We present QueryViz, a novel visualization tool for SQL queries that reduces the time needed to *read and understand existing queries*. It targets two principal audiences: (*i*) users who often issue the same or similar queries and who need to quickly browse through a repository of existing queries; and (*ii*) novices that try to familiarize themselves with the logic behind alternative patterns of SQL queries. QueryViz uses as input only two strings: the database schema and the SQL query. It can thus serve as light-weight add-on to existing database systems and is also available via an online interface at `http://queryviz.com`.

In this demonstration, we explain our visual alphabet, walk through the visualization algorithm, and let users experience the difference in understanding SQL queries from text or our graphical representation while browsing through repositories of well-known textbook SQL queries.

Categories and Subject Descriptors: H.5.2 [Information Interfaces and Presentation]: User Interfaces; H.2.3 [Database management]: Languages; H.1.2 [User/Machine Systems]: Human information processing

General Terms: Design, Theory, Human Factors

1. INTRODUCTION

Improving the usability of database systems is considered an important area of research [7]. Especially the user interfaces to databases have been long considered worthy of improvement [15], and easier application development with appropriate visual programming tools have listed repeatedly on the important database agendas (e.g. [12]). Recently, the idea of a *collaborative query management system* (CQMS) has been proposed that helps users issue queries by leveraging an existing log of queries [8, 9]. Independently, the SQL QuerIE project proposed to use personalized query recommendations to help users issue queries [4, 2]. Also a recent project, SQLshare [1] intends to combine both data and query management into one online system.

While *re-using existing solutions* is usually an effective method to speed up performance, the re-use of existing code or existing SQL queries seems to be more difficult and well characterized in the following comment of an anonymous reviewer: *"... it is very difficult, without additional documentation or explanation, to understand the meaning of complex queries written by others ... There seems to be a tension between the assumptions that the user is not sufficiently sophisticated to write his own complex query, but is sophisticated enough to parse someone else's complex query ..."* The main problem is that understanding of queries (also called query reading or *query interpretation* [13]) is often as hard as creating a new query (also called *query composition*). Hence, query interpretation is even used for testing purposes [16].

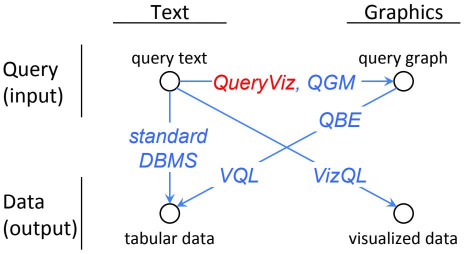
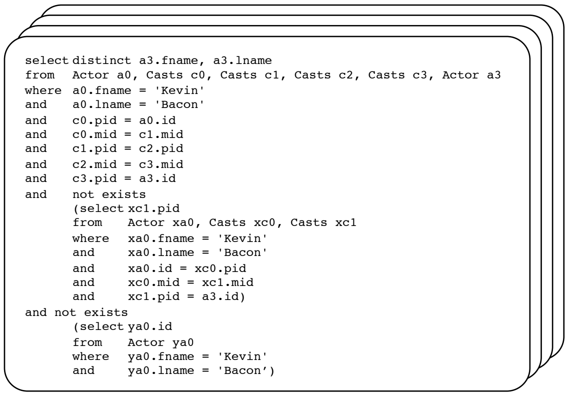


Figure 1: QueryViz takes a SQL query and creates a graph that helps users understand its meaning.

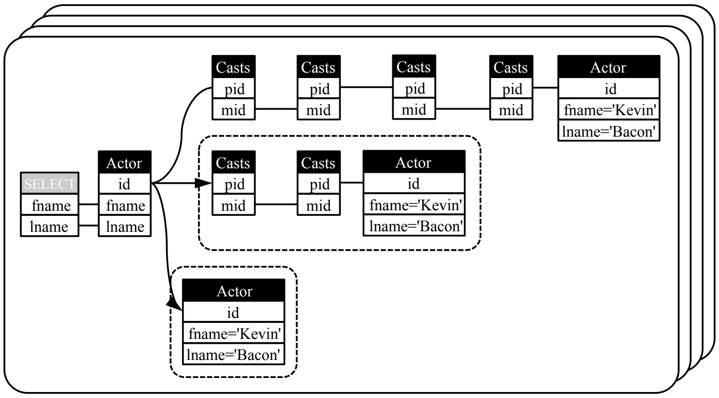
With QueryViz, we present a novel system that allows users *read and understand existing queries fast*. QueryViz thus enables effective query-reuse, a principal component in the vision of a collaborative query management system [8]. We achieve this goal by providing a concise set of visual constructs that intuitively encode the essence of a SQL query, together with a method that translates a large fragment of SQL into this representation. The key observation is that such a concise and intuitive set of visual constructs can be created by combining succinctness features of both tuple relational calculus and domain relational calculus. Thus, our motivation is quite different from an extensive body of work on developing visual query languages [3] for composing queries (e.g. QBE [18] and VQL [17], see Fig. 1). Humans are usually better in *recognizing* than *composing* visual constructs. Also, such constructs do not necessarily have to encode all information, but give only a glimpse that hint at the underlying meaning. Snippets in search engine results are a typical example. Our motivation is common with that of the Query Graph Model (QGM), developed for Star-

EDBT 2011, March 22–24, 2011, Uppsala, Sweden.
Copyright 2011 ACM 978-1-4503-0528-0/11/0003 ...$10.00



```
select distinct a3.fname, a3.lname
from   Actor a0, Casts c0, Casts c1, Casts c2, Casts c3, Actor a3
where  a0.fname = 'Kevin'
and    a0.lname = 'Bacon'
and    c0.pid = a0.id
and    c0.mid = c1.mid
and    c1.pid = c2.pid
and    c2.mid = c3.mid
and    c3.pid = a3.id
and    not exists
       (select xc1.pid
       from   Actor xa0, Casts xc0, Casts xc1
       where  xa0.fname = 'Kevin'
       and    xa0.lname = 'Bacon'
       and    xa0.id = xc0.pid
       and    xc0.mid = xc1.mid
       and    xc1.pid = a3.id)
and not exists
       (select ya0.id
       from   Actor ya0
       where  ya0.fname = 'Kevin'
       and    ya0.lname = 'Bacon')
```

(a) SQL query



(b) QueryViz representation

Figure 2: Application scenario: A user searches for a particular query by browsing through a repository of 100 previously recorded SQL queries. For each SQL query (a), she needs to *quickly understand* its meaning. QueryViz can help her understand the query by showing a *succinct representation of its logical semantics* (b).

bust [5]. However, QGM is targeted towards understanding actual query plans not the semantics of a query, and its full specification was never released to the public.

We illustrate our application scenario with an example.

EXAMPLE 1.1. *Consider a simple movie database with the schema:* Actor(*id*,*fname*,*lname*,*gender*), Casts(*pid*,*mid*,*role*), Movie(*id*,*name*,*year*,*rank*). *Now consider the SQL query depicted in Fig. 2a, or alternatively, the* QueryViz *representation in Fig. 2b. What is the meaning of this query? Answer: Find all actors with Bacon number 2.*

Our second main motivating application for QueryViz is as teaching tool. We have observed that students often have difficulties understanding the logic behind correlated and uncorrelated nested queries. Whereas coding patterns are well-studied and known, this notion is not common for SQL. QueryViz allows novice SQL users to browse through existing repositories and thus intuitively familiarize themselves with the logical patterns behind the SQL syntax.

Despite their intuitive appeal, graphical representations are not always better [11] and despite much effort and many visual languages proposed [3], visual query languages never took the important place conceived by their creators. We think there are three main points that distinguishes our motivation, scenario and solution from other approaches: (*i*) *Add-on*: Our visualization does not replace alternative encodings, but accompanies the textual representation of the query. It helps better understand existing code by getting a first intuitive glimpse of the query's meaning. This application is quite similar to text snippets in search engines. (*ii*) *No syntactic learning*: Graphical representations, especially those used in notation, are far from being intuitively obvious and need training for applying them. For us, users do not need to apply them, but just read them. They can immediately switch between text and graphics and learn while browsing. (*iii*) *Memory*: When a user issues a query often, the visual layout of the query is a strong cue for finding the same query during browsing.

Note that our work is closely related to recent work by Ioannidis et al. [14, 10] who translate SQL queries into natural language. They also use a graphical representation of a query, however only as intermediate step which is why their graphs are visually more complex (cp. [10, Fig. 3] vs. Fig. 6).

QueryViz is motivated by the CQMS project[1] at University of Washington that develops tools to help database users more effectively re-use previously executed queries.

2. QUERYVIZ OVERVIEW

QueryViz is implemented in Java. It receives two strings as input and uses a graph rendering tool (currently Graphviz[2]) to return a QueryViz image of the query (Fig. 3).

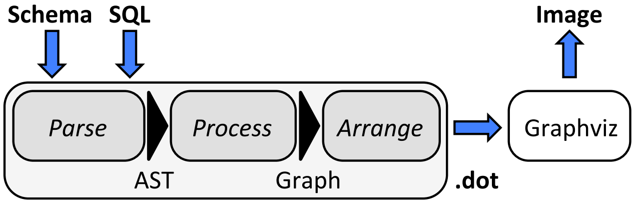


Figure 3: QueryViz consists of 3 internal modules that parse the query, process the internal data structure, and arrange the nodes, before Graphviz creates the actual figure.

First, QueryViz *parses* both the schema and the query strings into two abstract syntax trees (AST). Only a subset of SQL is currently supported and queries that use additional commands result in a syntax error. Second, it *processes* the ASTs and produces data structures representing the schema and the query. These data structures typically include information about tables to be used and their alias names, various predicates specified in the query, and columns to be projected. These data structures are then manipulated to create an underlying encoding for the graph visualization. The output of this stage is an internal representation of the QueryViz graph. The third stage *arranges* the QueryViz graph by including information about vertical and horizontal depth arrangements. QueryViz currently outputs this information as a DOT file, which is finally rendered with the help of GraphViz.

[1] http://db.cs.washington.edu/cqms

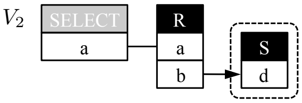
[2] http://www.graphviz.org/

R(*a*,*b*,*c*), S(*d*,*e*)

q_2
```
select  a
from    R
where   b <> ALL
        (select  d
         from    S)
```

q_3
```
select  a
from    R
where   b not IN
        (select  d
         from    S)
```

q_4
```
select  a
from    R
where   not exists
        (select  *
         from    S
         where   b=d)
```



$$q_2: \; a : \exists b.\exists c.(R(a,b,c) \wedge \forall d.\forall e.(S(d,e) \Rightarrow d \neq b))$$

$$q_3: \; a : \exists b.\exists c.(R(a,b,c) \wedge b \notin \{d \mid \exists e.S(d,e)\})$$

$$q_4: \; a : \exists b.\exists c.(R(a,b,c) \wedge \neg\exists d.\exists e.(S(d,e) \wedge d = b))$$

$$V_2: \; a : \exists b.(R(a,b,\_) \wedge \neg(\exists d.(S(d,\_) \wedge d = b)))$$

Figure 4: Schema with three equivalent queries (q_1 to q_3), their common QueryViz representation V_2 and their respective translations into FOL.

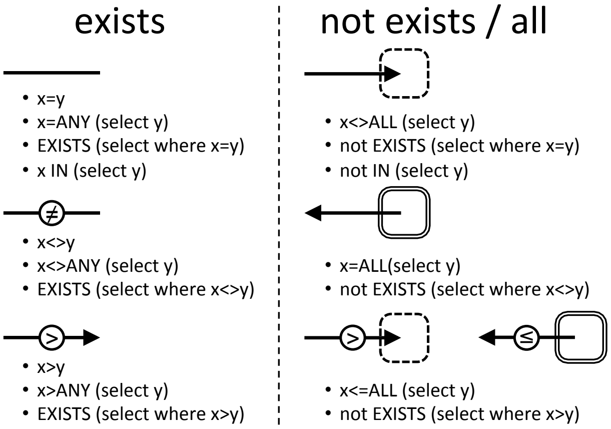
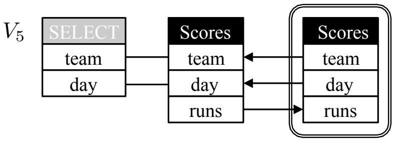


Figure 5: Lines with optional direction and comparison operators, together with bounding boxes in two line styles that group together relations suffice to express the most important syntactic constructs of nested SQL queries.

Scores(*team*,*day*,*opponent*,*runs*)

q_5
```
select  S1.team, S1.day
from    Scores S1
where   not EXISTS
        (select  *
         from    Scores S2
         where   S1.runs=S2.runs
         and     (S1.team<>S2.team OR
                  S1.day<>S2.day))
```



$$q_5: \; t,d : \exists o.\exists r.(S(t,d,o,r) \wedge \neg(\exists t_2.\exists d_2.\exists o_2.\exists r_2.$$
$$(S(t_2,d_2,o_2,r_2) \wedge r_2 = r \wedge (t_2 \neq t \vee d_2 \neq d))))$$

$$V_5: \; t,d : \exists r.(S(t,d,\_,r) \wedge \forall t_2.\forall d_2.\forall r_2.$$
$$(S(t_2,d_2,\_,r_2) \wedge r_2 = r \Rightarrow t_2 = t \wedge d_2 = d))$$

Figure 6: Query for "teams and days on which the team had a run that was neither repeated on another day nor by another team," its QueryViz representation, and their respective translations into FOL.

3. A GLIMPSE BEHIND THE SCENES

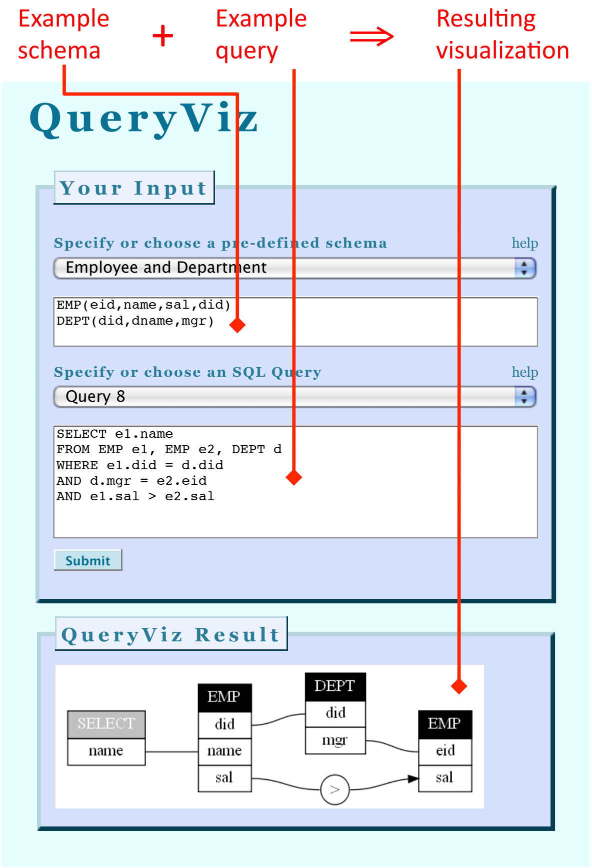
The complexity of understanding SQL queries is partly due to the fact that there are several alternative expressions that are equivalent, based on syntactic redundancy and algebraic properties of the query constructs [6]. For example, Fig. 4 shows three identical queries with different syntax.

Our observation is that a variety of syntactic constructs can be represented *with only a few core visual constructs*, namely: three types of nodes, lines with optional direction and comparison operators, and optional bounding boxes in two line styles (cp. Fig. 5). Our visual alphabet is very close to the basic operators of the First-Order Logic (FOL) representation of queries, but also combines succinctness ideas from both *tuple relational calculus* (TRC) and *domain relational calculus* (DRC): Similar to DRC (and hence Datalog), the locality between a relation and its attribute names is preserved, hence there is no need for aliases. Similar to TRC (and SQL), we do not represent anonymous variables, i.e. variables that do not participate in any selection, join or comparison predicate. Furthermore, our representation allows edges not just between attributes, but also between relation names for unions of queries (no example shown here).

Definition 3.1 (QueryViz Graph). *a QueryViz graph V is $(\mathcal{R}, S, E, \mathcal{B})$ where: $\mathcal{R}$ is a collection of sets of nodes $R_i = (r_i, A_i)$ with r_i being the relation name and A_i being an ordered subset of the relation's attributes $(a_{i1}, \ldots, a_{ik})$; $S = (SELECT, A_s)$ is set of nodes representing the selected attributes; E is a set of edges $e_j = (s_j, t_j, c_j)$ where the start node s_j and end node t_j are chosen from $\bigcup r_i \cup \bigcup a_{ij}$, i.e. from both the relation names and their attributes, and c_j is a comparison operator $c \in \{=, \neq, >, \geq, <, \leq\}$; and $\mathcal{B}$ is a possibly empty set of bounding boxes $B_i = (\mathcal{R}_i, s_i)$ where each bounding box groups together a disjoint set $\mathcal{R}_i$ of relations and has a line style $s_i \in \{dashed, double\}$.*

Example 3.2. *Consider query q_5 in Fig. 6, taken from the Demo quiz of* `gradience.com`. *Both relations participating in the query are identical but do not need aliases. Attributes which are not relevant (here* opponent*) are not shown. Also, in this particular case, visualizing disjunctions can be avoided by using universal quantification.*

Theoretical and practical limits. QueryViz focuses only on *set semantics* and does not provide visual constructs for distinct nor outer-joins. The formalism allows for unions of queries and, with the help of one additional visual construct, for arbitrary nestings of disjunctions; however, the representation can then become exponentially large in the



Specify or choose a pre-defined schema: Employee and Department

```
EMP(eid,name,sal,did)
DEPT(did,dname,mgr)
```

Specify or choose an SQL Query: Query 8

```
SELECT e1.name
FROM EMP e1, EMP e2, DEPT d
WHERE e1.did = d.did
AND d.mgr = e2.eid
AND e1.sal > e2.sal
```

Figure 7: The online interface to QueryViz will be accessible from `queryviz.com`. This simple query asks for all employees that earn more than their manager.

size of the query and, hence, less helpful for the user than the original query text (note that disjunctions do not necessarily lead to complicated representations as seen in Example 3.2). Our implementation currently does not support disjunctions, but could be extended to do so in the future. Furthermore, QueryViz focuses only on nested queries in the WHERE clause and currently lacks visual constructs for groupings and aggregations.

4. DEMONSTRATION SETUP

The QueryViz demonstration will present the visual formalism and the algorithm for translating queries. It will also let users familiarize themselves with the system across different scenarios. The setup will include a machine connected to the online version and an additional installation with various user interactions on a local machine.

The *online version* allows users to visualize existing or newly defined queries. Figure 7 shows a screenshot. The workflow and interaction are kept very simple: a user specifies or chooses an existing schema and a query, the system then renders the graphical representation of this query.

For the local setup, we will demonstrate QueryViz across three different scenarios. In each scenario, users have the option to either use only query text, only QueryViz, or both side-by-side. They can optionally measure time and compare if they are faster by using QueryViz or not in a multiple-choice test: (*i*) *browse*: Users browse through existing queries and informally familiarize themselves with the visual constructs. The goal is not to look at the formal logical foundations, but at a number of illustrating examples. We will also have several printed catalogs with example query representations that allow several users browsing in parallel. (*ii*) *search*: Users are given a natural language description of a query. The task is then to find the correct query among 10 queries in minimum time. This scenario closely matches our main motivating scenario but is difficult to measure on a large scale. (*iii*) *query interpretation* [13]: Users are given a query in the query language and a printed database with data filled in. They are asked to find the data asked for by the query. This scenario has the advantage that, while it still captures the difference in speed of understanding a query, it can be scaled: it allows to automatically generate many random questions and test understanding by the use of multiple-choice tests [16].

Acknowledgement. We like to thank YongChul Kwon, Nodira Khoussainova, Magdalena Balazinska and Dan Suciu for helpful discussions at early stages of this project. This research is supported in part by NSF grant IIS-0915054.

5. REFERENCES

[1] SQLShare. `http://escience.washington.edu/sqlshare`.
[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(1):1597–1600, 2010.
[3] T. Catarci, M. F. Costabile, S. Levialdi, and C. Batini. Visual query systems for databases: A survey. *J. Vis. Lang. Comput.*, 8(2):215–260, 1997.
[4] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. Query recommendations for interactive database exploration. In *SSDBM*, 2009.
[5] L. M. Haas, J. C. Freytag, G. M. Lohman, and H. Pirahesh. Extensible query processing in starburst. In *SIGMOD*, 1989.
[6] Y. E. Ioannidis. From databases to natural language: The unusual direction. In *NLDB*, 2008.
[7] H. V. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu. Making database systems usable. In *SIGMOD*, 2007.
[8] N. Khoussainova, M. Balazinska, W. Gatterbauer, Y. Kwon, and D. Suciu. A case for a collaborative query management system. In *CIDR*, 2009.
[9] N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu. Snipsuggest: A context-aware SQL-autocomplete system. *PVLDB*, 4(1), 2010.
[10] G. Koutrika, A. Simitsis, and Y. E. Ioannidis. Explaining structured queries in natural language. In *ICDE*, 2010.
[11] M. Petre. Why looking isn't always seeing: Readership skills and graphical programming. *Commun. ACM*, 38(6):33–44, 1995.
[12] R. Agrawal et al. The claremont report on database research. *Commun. ACM*, 52(6):56–65, 2009.
[13] P. Reisner. Human factors studies of database query languages: A survey and assessment. *ACM Comput. Surv.*, 13(1):13–31, 1981.
[14] A. Simitsis and Y. E. Ioannidis. DBMSs should talk back too. In *CIDR*, 2009.
[15] S. Spaccapietra. User interfaces; who cares? In *VLDB*, 1994.
[16] J. D. Ullman. Improving the efficiency of database-system teaching. In *SIGMOD*, 2003.
[17] K. V. Vadaparty, Y. A. Aslandogan, and G. Özsoyoglu. Towards a unified visual database access. In *SIGMOD*, 1993.
[18] M. M. Zloof. Query-by-example: A data base language. *IBM Systems Journal*, 16(4):324–343, 1977.