Databases will Visualize Queries too[*]

Wolfgang Gatterbauer
University of Washington, Seattle[†]

ABSTRACT

Visual Query Languages study ways to help users compose queries with visual metaphors. *Information Visualization* studies automatic visualization techniques to help users understand and analyze data. *Query Management* focuses on ways to help users manage and re-use existing queries. We observe that there is a related research question across those three topics which has not received much attention, namely that of *Query Visualization: How to visually represent a query to help users quickly understand its intent?* Here we argue that the involved challenges are still markedly different from those of the other three, that a solution can considerably improve the usability of DBMSs, and that the topic is thus worthy of attention. We envision, that in a few years, there will be free, modular, and lightweight tools available that allow users to visualize and interpret their queries.

1. QUERY INTERPRETATION IS HARD

Query Interpretation is the problem of reading and understanding an existing query. It is often as hard as *Query Composition*, i.e. creating a new query [22]. Just like understanding program code, query interpretation requires guessing the line of thought of the query composer, making connections between attributes of tables, while keeping in mind the schema and aliases of tables. Hence, query interpretation requires significant comprehension of SQL and is even used for testing purposes [24].

Recently, several projects have focused on building *Query Management Systems* that help users issue queries by leveraging an existing log of queries. Known systems to date include CQMS [17, 18], SQL QuerIE [3, 1], DBease [20], and SQLshare [10]. All of those are motivated by making SQL composition easier and thus databases more usable [16], especially for non-sophisticated database users.

An essential ingredient for such systems is a *query browse* facility, i.e. an interaction mode that allows the user to quickly understand and choose between several queries proposed by the system. Here, the optimal computer human interaction is similar to that for text snippets for search engines [23]: the computer searches through many and proposes a few (high recall), the human browses through a few and chooses one (high precision). As the queries are shown, their *representation* should be: (i) *representative*, so users can grasp the essence of the result from its snippet, (ii) *distinguishable*, so users can differentiate between queries with little effort, (iii) *small*, enough so users can quickly browse several query representations, while being (iv) *self-contained*, so users can understand it without other help (four goals taken from [12]). In other words, the query representation should capture the *query intent*[1] very concisely.

[†]New contact info: gatt@cmu.edu
[*]Title strongly inspired by Ioannidis and Simitsis [14].

Proceedings of the VLDB Endowment, Vol. 4, No. 12
Copyright 2011 VLDB Endowment 2150-8097/11/08... $ 10.00.

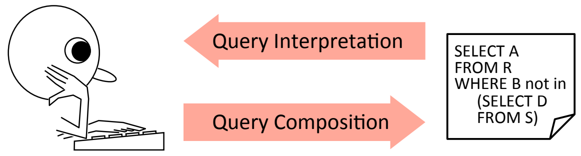

Figure 1: Interpreting an existing query is often as hard as composing a new query (red = hard).

Query representation. There are 4 principal options to help users interpret an existing query: (1) *Visual manipulation of text*: Clients to major DBMSs, such as SQL server management studio and pgAdmin for PostgreSQL, have been long highlighting different syntactic constructs or aligning query blocks and clauses. It is helpful, yet not sufficient to help users understand a query's intention fast.

(2) *Translation into natural language*: Ioannidis et al. [13, 14, 19] propose to explain queries in natural language. They convincingly argue that automatically creating effective free-flowing text from queries is difficult and the overall task quite different from previous work on creating natural language interfaces to DBMSs.

(3) *Illustration with example instances*: Olston et al. [21] study the problem of generating example intermediate data for data flow programs that quickly illustrates the semantics of the operators to users. This approach could be applied to illustrating SQL queries with example input and output.

(4) *Visualizing the query*: Query visualization creates a diagrammatic representation of an existing SQL query. It is thus the reverse of visual query languages which allow

[1]The term *query intent* captures the high-level goal of the query, independent of the actual choice of syntax.

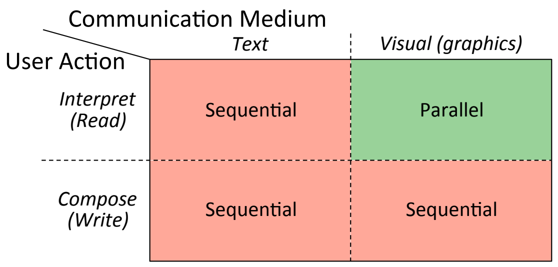

Figure 2: *Composing* a query with a visual query language is as sequential as composing it with SQL. *Interpreting* a visual (whether of information or a query) is the only modus in which a user can act on information in parallel and thus fast (green = easy).

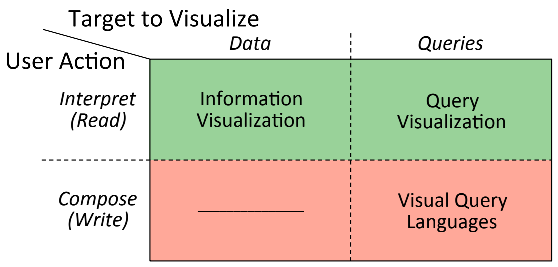

Figure 3: *Visual Query Languages* allow a user to compose queries. In contrast, *Query Visualization* helps the user understand an existing query just as *Information Visualization* helps understand data.

users to compose a query. To the best of our knowledge, there is currently no system available that allows users to quickly visualize an existing query. The two projects that come closest in spirit are the Query Graph Model (QGM) developed for Starburst [8] and Visual SQL [15] developed for teaching students the SQL syntax. The former was not targeted towards helping users understand the query intent, but rather actual query plans, and its full specification was never released.[2] The latter currently does not support the reverse transformation from SQL-code into Visual SQL.

The vision. In contrast to the heading of this section, the main thesis of this paper is that option (4) above is the one best suited for query interpretation, and that *query interpretation is actually not hard* if queries are accompanied by an appropriate visual representation of the *query intent*. The vision is that in a few years from now, major DBMSs will support query visualization and that this functionality will considerably facilitate query re-use and refinement.

2. QUERY VISUALIZATION: NOT YET ANOTHER VISUAL QUERY LANGUAGE

Humans are good at understanding complex structures from visuals, and thus the idea that a visual query language (VQL) would facilitate composition of queries is quite old (the 1997 survey of Catarci et al. [2] cites over 150 papers). So why have visual query languages not taken off at large?[3] We believe that the primary reason[4] and stumbling block for VQLs lies in Fig. 2: Humans are better in *interpreting* than *composing* visuals because visual composition is an inherently sequential process.[5] Hence, even in theory, there is no dramatic speed-up by using a visual language. In practice, the user interaction is quite cumbersome: users must be able to interactively construct and manipulate expressions in the visual language and connect graphical elements together to establish graphical relationships. In turn, the program must provide appropriate interpretations of mouse and keyboard events, and it is difficult to build formal grammars and compilers for two-dimensional drawing areas. In sum, solutions to these graphical requirements are intricate and inherently difficult to implement and use [25].

Hence, we do not suggest to create yet another VQL! We consider the reverse goal of VQLs, namely that of *helping users interpret an existing query* (Fig. 3). In programming languages, this distinction is known as the difference between *visual programming* for developing a program and *program visualization* for analyzing an existing program or software. Yet similar to translating queries into text, this explicit reverse functionality has not drawn much attention for visual query languages. Also, query visualization is related to *Information Visualization* which focuses on helping users understand and analyze data [4]. Both focus on helping users understand complex relationships through visualizations; the difference is in their respective targets (Fig. 3).

2.1 Principles of Query Visualization

Query Visualization has the potential to help users quickly understand and distinguish between a set of given queries. Not only does the interaction fit the optimal value of visualization (Fig. 2), it also enhances the user experience *without replacing* the current model of interaction for composing a query in SQL, i.e. by text. Now, the challenge of query visualization is to *find the appropriate visual metaphors which (i) allow users to quickly understand a query's intent, (ii) can be easily learned by users, and (iii) are extensible to more expressive queries.* In addition, any query visualization needs an automatic translation from SQL to the visualization, including a visually-appealing automatic arrangement of nodes of the visualization.

We believe that users can learn to interpret visualized queries by seeing examples without much active focus. This is similar to what is known in language learning theory as the difference between the active and the generally larger passive vocabulary: Actively reproducing newly learned content is generally more difficult than passively recognizing such con-

[2]Personal communication with the authors.

[3]The Picasso project by Haritsa [9] focuses on visualizing and comparing the speed of execution plan and is not directly related.

[4]A second reason is that graphs are more ambiguous than text, i.e. it is more difficult to be precise with a visual representation than with text. This is another justification for the idea in Fig. 8, namely that *composition* is better done in text (more precise), but *interpretation* with a visual (focus on intuition over precision).

[5]The argument in short: all human input methods (composition) are sequential, whether resulting in text or a graphics. Visual perception is the only human sense (interpretation) that *can* work in parallel, and it works dominantly by *spatial arrangement of information.* While reading text is also a visual activity, the spatial arrangement of the letters requires a sequential scan of the text (note that reading spatially arranged text can be partly parallel; compare option "visual manipulation of text" as query representation from before). Hence, visual interpretation of graphics is the fastest communication method to or from humans, and it only works that well in one way.

```
SELECT  A
FROM    R
WHERE   B not in
        (SELECT  D
        FROM     S)
```

```
SELECT  A
FROM    R
WHERE   not exists
        (SELECT  *
        FROM     S
        WHERE    B=D)
```

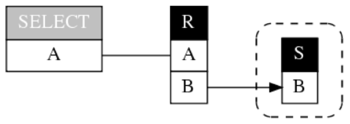

Figure 4: Two queries which are equivalent *except* if the column $S.b$ contains NULL values. Ignoring this one case, they are equivalent. Hence, the *query intent* can be shown by the same representation.

tent. Further, we propose that the development of the visual alphabet should follow three practical guiding principles:

(1) *Existing metaphors as starting point*: Most database users have seen UML diagrams before. A simple conjunctive query should not be visualized much differently from what is currently used for showing database schemas.

(2) *Minimal visual complexity*: The alphabet should contain only a minimum number of visual constructs, those which are possibly overloaded and ambiguous like in natural language. This simplicity may come at the expense of not distinguishing special cases, such as dealing with NULL values during joins (see Fig. 4 for an example[6]). Tools that help users cope with the inherent syntactic difficulty fall under the category of SQL debugging (e.g., see [7]).

(3) *Progressive complexity*: As in entropy encoding (e.g. with Fano codes), visual constructs for more common logical operators should be designed with lower visual complexity than less common ones. For example, almost all database queries use the logical AND in their first-order logic translation (e.g. joins, EXISTS, IN), but only few use OR (e.g. OR, UNION). If infrequent query constructs become increasingly complex to read, this progression does not decrease the overall usability, but rather assures that more often used constructs are simple to read, in turn.

3. OUR SUGGESTION: QUERYVIZ

When developing QueryViz [5], our own SQL visualization, we not only followed the guidelines listed before, we have also been influenced and guided by several existing ideas: We started from UML and its familiar notations for data modeling, then enhance the visual representation in a *progressive way*. *Conjunctive queries* (single query blocks) have the lowest visual complexity and are represented as shown in figure Fig. 5. In contrast to SQL, which is modeled after the relational tuple calculus, this representation is related to the relational domain calculus. The advantage is that *no aliases are needed* and each attribute of a table is

```
SELECT  F.person
FROM    Frequents F, Likes L, Serves S
WHERE   F.person = L.person
AND     F.bar = S.bar
AND     L.drink = S.drink
```

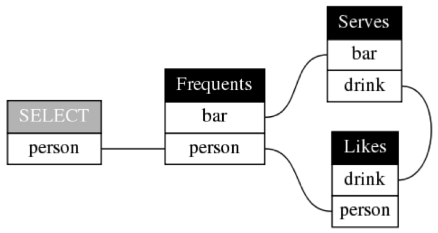

Figure 5: Visualizing a conjunctive query closely follows an all-familiar UML notation. *Q: Find persons who frequent some bar that serves some drink they like.* There is nothing really new here.

immediately recognized.

For increasing complexity of nested queries, we are inspired by a body of work on *diagrammatic reasoning systems* [11]. Diagrammatic notations are itself is inspired by the influential *existential graph notation by Charles Sanders Peirce* and exploit topological properties, such as enclosure, to represent logical expressions and set-theoretic relationships, mainly for monadic relations. QueryViz now incorporates visual metaphors from diagrammatic reasoning into relational UML graphs and adapts them where necessary or appropriate. Figure 6 shows a more complex SQL query. The natural language translations of this and the conjunctive query are equally complex and have the same length. Similarly, Fig. 6 shows an only slightly increased visual complexity (13% more visible elements) over Fig. 5. In contrast, SQL has become quite more complex (167% more words).

Since QueryViz is modeled on the first order logic query intent of SQL, it follows a *set semantics* and not the common bag semantics as in SQL. As consequence, it ignores the DISTINCT operator completely. We further simplified the visual representation by adding a universal quantifier, a construct that does not exist as such in SQL. As example, the representation from Fig. 6 can be further simplified to the one in Fig. 7. This representation now has only 7% more visible elements than the conjunctive query from Fig. 5.

Another concept borrowed from diagrammatic reasoning is a *default reading order* [6]. Note from Fig. 7 how the arrows between the relations correspond to the natural language translation. Without such, there would be no natural order placed on the existential and universal quantifiers. There are more ideas to QueryViz (such as how to handle logical OR, and how to overload the semantics of the arrow, etc.) which will be treated in a more detailed paper.

3.1 From theory to practice

All figures in this paper are drawn with an implementation of QueryViz [5]. We encourage the reader to try it. An interactive online version is available at `http://queryViz.com`. It currently supports only a limited SQL grammar (see the web page for details). Still, this online interface and its usage are proof that query visualization can have a very lightweight interaction. The user does not have to specify anything up-

[6]This tolerated ambiguity is related to the ambiguity of language. Being exhaustive in enumerating all special cases makes *explanations* cumbersome and unpleasant. Compare to the idea of default logic in AI: “Bird Tweety flies” (except if Tweety is a penguin); “Those two queries are equal” (except if an input table is empty or contains a NULL value Fig. 4). Ignoring those special cases simplifies communicating a query’s principal intent.

```
SELECT  F.person
FROM    Frequents F
WHERE   not exists
        (SELECT  *
        FROM     Serves S
        WHERE    S.bar = F.bar
        AND      not exists
                 (SELECT  L.drink
                 FROM     Likes L
                 WHERE    L.person = F.person
                 AND      S.drink = L.drink))
```

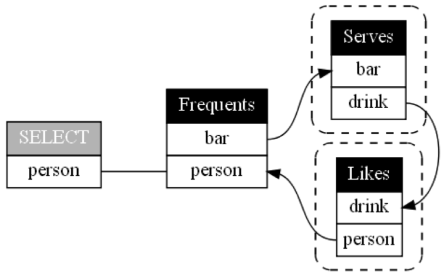

Figure 6: Visualizing a nested query still follows familiar UML notations, but now adds visual metaphors for ∄ (dashed box) and reading order (arrows). ***Q: Find persons who frequent some bar that serves only drinks they like ≡ … some bar that serves no drink that is not liked by them.***

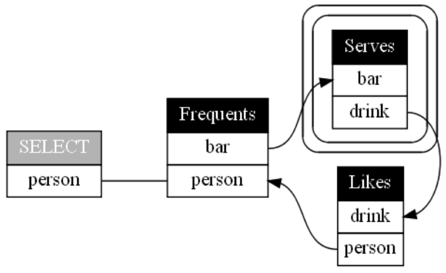

Figure 7: The visualization from Fig. 6 can be further simplified by using another visual metaphor for ∀ (double-lined box), a logical and intuitive operator that does not exist in SQL. ***Q: Find persons who frequent some bar that serves only drinks they like ≡ … some bar so that all drinks served are liked by them.***

front and can just copy the SQL query and the schema into the two available forms (note that the schema could be automatically inferred from the query). We are soon extending it with the visual metaphor for groupings and will then evaluate it with massive users studies on Amazon Turk.

Going forward to more expressive query visualizations, we have not yet solved all problems (such as for nested disjunctions of depth bigger than 2, or outer joins). All that, and even alternative outline algorithms (e.g. using a metric that captures visual homogeneity) will be future work. At the end of our project, we intend to make our code available as open-source, so people can build and extend it with improved solutions.

4. THE VISION IN A NUTSHELL

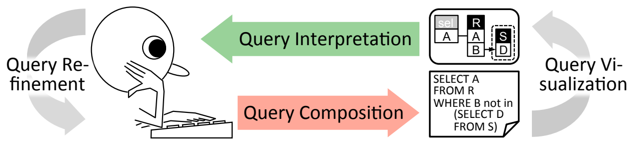

Figure 8: The vision: In the near future, *DBMSs will visualize queries too*, and not just data (as in information and scientific data visualization). This feature will allow iterative query refinement and will enhance the usability of databases.

The database community discovers that query visualization is a well-defined and worthy research area for increasing the usability of databases, quite similar to explorative information visualization. Different approaches for query visualization are developed and demo implementations are available. Focus is again on speed and effectiveness, that of transmitting a query's intent fast and correctly to a user. Those goals can easily be tested with user studies for new visual alphabets. Large sets of benchmark queries are available, and query visualization has become an integral part of major DBMSs. Hence, *databases visualize queries too* (Fig. 8).

Acknowledgement. I thank Jonathan Danaparamita for working with me on QueryViz, and YongChul Kwon for valuable discussions over the course of this research. The work is supported in part by NSF grant IIS-0915054.

5. 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(1):1597–1600, 2010.
[2] 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.
[3] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. Query recommendations for interactive database exploration. In *SSDBM*, pp. 3–18, 2009.
[4] C. Chen. *Information visualization: beyond the horizon.* Springer, New York, 2nd ed edition, 2006.
[5] J. Danaparamita and W. Gatterbauer. Queryviz: Helping users understand SQL queries and their patterns. In *EDBT*, pp. 558–561, 2011.
[6] A. Fish and J. Howse. Towards a default reading for constraint diagrams. In *Diagrams*, pp. 51–65, 2004.
[7] T. Grust, F. Kliebhan, J. Rittinger, and T. Schreiber. True language-level SQL debugging. In *EDBT*, pp. 562–565, 2011.
[8] L. M. Haas, J. C. Freytag, G. M. Lohman, and H. Pirahesh. Extensible query processing in Starburst. In *SIGMOD*, pp. 377–388, 1989.
[9] J. R. Haritsa. The Picasso database query optimizer visualizer. *PVLDB*, 3(2):1517–1520, 2010.
[10] 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.
[11] J. Howse. Diagrammatic reasoning systems. In *ICCS*, pp. 1–20, 2008.
[12] Y. Huang, Z. Liu, and Y. Chen. Query biased snippet generation in XML search. In *SIGMOD*, pp. 315–326, 2008.
[13] Y. E. Ioannidis. From databases to natural language: The unusual direction. In *NLDB*, pp. 12–16, 2008.
[14] Y. E. Ioannidis and A. Simitsis. DBMSs should talk back too. In *CIDR*, 2009.
[15] H. Jaakkola and B. Thalheim. Visual SQL – high-quality ER-based query treatment. In *ER (Workshops)*, pp. 129–139, 2003.
[16] H. V. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu. Making database systems usable. In *SIGMOD*, pp. 13–24, 2007.
[17] N. Khoussainova, M. Balazinska, W. Gatterbauer, Y. Kwon, and D. Suciu. A case for a collaborative query management system. In *CIDR*, 2009.
[18] N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu. Snipsuggest: A context-aware SQL-autocomplete system. *PVLDB*, 4(1):22–33, 2010.
[19] G. Koutrika, A. Simitsis, and Y. E. Ioannidis. Explaining structured queries in natural language. In *ICDE*, pp. 333–344, 2010.
[20] G. Li, J. Fan, H. Wu, J. Wang, and J. Feng. DBease: Making databases user-friendly and easily accessible. In *CIDR*, pp. 45–56, 2011.
[21] C. Olston, S. Chopra, and U. Srivastava. Generating example data for dataflow programs. In *SIGMOD*, pp. 245–256, 2009.
[22] P. Reisner. Human factors studies of database query languages: A survey and assessment. *ACM Comput. Surv.*, 13(1):13–31, 1981.
[23] A. Tombros and M. Sanderson. Advantages of query biased summaries in information retrieval. In *SIGIR*, pp. 2–10, 1998.
[24] J. D. Ullman. Improving the efficiency of database-system teaching. In *SIGMOD*, pp. 1–3, 2003.
[25] K. Zhang. *Visual languages and applications.* Springer, New York, 2007.