

# Visualizing Query Structure

Dennis P. Groth

Indiana University School of Informatics

Bloomington, IN 47408, USA.

Email: dgroth@indiana.edu

## Abstract

Visualization research seeks to exploit the human user's ability to interpret graphical representations of data in order to provide insight into properties of the data that may not be identifiable through other, non-graphical means. We seek to answer the following question with this research: "What can we learn from data by looking at the questions that are asked about the data?" In a database context, we view queries as the data of interest, focusing on structural properties of the queries. In this paper we describe a formalism for describing the structural equivalence of queries. We demonstrate an application of our approach by using a graph-based technique to visually represent the structure.

## 1 Introduction

Visualization research seeks to exploit the human user's ability to interpret graphical representations of data in order to provide insight into properties of the data that may not be identifiable through other, non-graphical means. Information visualization, in particular, focuses on visualizing abstract information, which often requires innovative mappings from the data space to the graphical representation.

While the goal of visualization research is often to uncover information about "data", this research seeks to discover information in an indirect fashion by visualizing queries, or questions, about the "data". In particular, we focus on structural properties of queries in order to derive an understandable measure of comparison between queries. This measure can be used to determine where and how queries are graphically represented.

We graphically represent query structure as graphs, taking advantage of the visual impact of

graph structures to enhance a user's understanding of the underlying information structure. Accordingly, a user might find themselves drawn into further, more detailed exploration of the space through visual means, which otherwise might never occur.[10, 1, 9]

Visualizing structure is certainly not new. Chemists use visualization system to view crystal structure. Biologists visualize the structure of amino acids. In the context of database systems, we have previously investigated methods for visualizing structural content of data.[7]

Somewhat related work is that of visual query specification, in which users graphically represent the syntax of queries.[2, 3, 4] Some of these approaches include graph-based presentations.[8, 6] These visual query techniques are aimed at solving a problem of specifying queries - not uncovering information about the underlying information structure.

The remainder of this paper is structured as follows. Section 2 provides basic definitions from the relational model that are used. Section 3 describes the structural components of queries that we consider, as well as providing a formalism for describing the structural equivalence of queries. Section 4 describes the visualization of queries as graph structures. Lastly, Section 5 concludes with a description of future work and applications of our technique.

## 2 Definitions

The underlying basis of our approach is the relational model.[5] For sake of clarity we identify the key constructs of the relational model that are employed. First, let  $\mathbf{R} = \{R, S, T\}$  be a database schema with a finite set of relations. For each relation  $R \in \mathbf{R}$ , we define a finite set of attributes,  $R = \{A_1, \dots, A_n\}$ . For each attribute  $A_i$  we associate a set of possible values  $\mathbf{dom}(A_i)$ , called the do-

main of  $A_i$ . When considering an instance  $\mathbf{r}$  we define  $adom(A_i)$ , the active domain of  $A_i$ , to be the set of values existing in  $\mathbf{r}$ . Note that, while  $adom(A_i)$  is finite,  $\mathbf{dom}(A_i)$  may be infinite. A tuple  $t$  of  $\mathbf{r}$  is a function  $\mathbf{dom}(A_i) \rightarrow adom(A_i)$ . Define  $\mathbf{dom}(R) = \mathbf{dom}(A_1) \cup \dots \cup \mathbf{dom}(A_n)$ .

We sometimes refer to values with the notation  $t.A$ , meaning the value of the  $A$  attribute in tuple  $t$ . For sets of attributes,  $t.X$  means the values of each attribute of  $X$  in tuple  $t$ . For a relation  $R$ , the instance  $\mathbf{r}$  is a finite set of tuples.

A query  $q$  is a function mapping from instances of relations to instances. We focus on the structural components of such queries for this research. In particular, we consider relational algebra (RA) queries involving relation symbols, attributes, and the following relational operators:

**Selection:** This operator, written  $\sigma_\theta(R)$ , selects tuples from its input that satisfies the boolean condition  $\theta$ . The  $\theta$  expression is typically of the form  $\alpha\phi\beta$ , where  $\alpha, \beta \in R \cup \mathbf{dom}(R)$  and  $\phi$  is a binary relation. For instance  $\mathbf{r}$ ,  $\sigma_\theta(\mathbf{r}) = \{t | t \in \mathbf{r} \wedge \theta\}$ .

**Projection:** This operator, written  $\pi_X(R)$ , projects attributes of  $R$ . For instance  $\mathbf{r}$ ,  $\pi_X(\mathbf{r}) = \{t.X | t \in \mathbf{r} \wedge X \subseteq R\}$ .

**Cartesian Product:** This operator, written  $R \times S$ , combines relations. For example, if  $R = \{A, B, C\}$  and  $S = \{D, E\}$ ,  $R \times S = \{A, B, C, D, E\}$ . For instances  $\mathbf{r}$  and  $\mathbf{s}$ ,  $\mathbf{r} \times \mathbf{s} = \{t | \langle t.A, t.B, t.C \rangle \in \mathbf{r} \wedge \langle t.D, t.E \rangle \in \mathbf{s}\}$ .

In addition to these operators, and since relations are considered as sets, the algebra allows for Union, Intersection and Difference operators, written as  $\cup$ ,  $\cap$ , and  $-$ .

Using the relational algebra, a query  $q$  is simply an expression composed from the basic operators. The application of each of these operators (to the proper number of relation instances) produces another relation. Thus composition of operators is automatic. As a result, the relational algebra is considered a *Procedural* language, in that the query may be implemented directly from the operations.

Sometimes included as an operator in the algebra is the Join operator, which allows for more succinct expressions. The join operator is written as  $R \times_\theta S$ , where  $\theta$  is a boolean expression, as used in the selection operator. The join,  $R \times_\theta S = \sigma_\theta(R \times S)$ . A

special type of join, called the natural join is written by  $R \bowtie S$ , in which the resultant tuples agree on the attributes common to  $R$  and  $S$ .

## 2.1 SQL - The Structured Query Language

In practice, the Structured Query Language (SQL) is used to specify queries against relational database systems. As the name implies, each query is structured with the following basic syntax:

```
select attribute_list
from table_list
where boolean condition
```

With respect to the relational algebra, the list of attributes is equivalent to the project operator, the list of tables is equivalent to the cartesian product operator, and the boolean condition is equivalent to the select operator.

## 3 Query Structure

In order to visualize query structure we must first identify the structural elements that are contained within a query. In this section we define the elements that are contained within Project-Select-Join (PSJ) queries. In particular, we focus on relations, attributes, and join conditions.

Let  $\mathbf{R}$  be a database schema, and  $q$  be a PSJ query, we define:

**Definition 3.1 Relation Schema:**  $Rel(q) = \{(R, c) | R \in \mathbf{R} \wedge R \text{ occurs } c \text{ times in } q\}$

Let  $q$  be a query and  $\pi_\theta$  be a project expression in  $q$ , we define:

**Definition 3.2 Project Schema:**  $Sch(q, \pi) = \{(R, A) | R \in \mathbf{R} \wedge A \in \theta\}$

Let  $q$  be a query,  $\beta = \{=, \neq, <, \leq, >, \geq\}$ , and  $\sigma_{A'\beta A''}$  be a select expression in  $q$ , we define:

**Definition 3.3 Select Schema:**  $Sch(q, \sigma) = \{(R, A) | R \in \mathbf{R} \wedge R \in q \wedge A \in R \wedge [A = A' \vee A = A'']\}$

Building upon the select schema, we define the join conditions in a query:

**Definition 3.4 Join Condition:**  $Join(q) = \{(R, A, \beta, S, B) | (R, A) \in Sch(q, \sigma) \wedge (S, B) \in Sch(q, \sigma)\}$

In summary, we consider these elements to be the *base* structure of a query. Consider the following example:

### Example 3.1

Consider the following relations:

$Emp = \{EmpID, EmpName, Salary, DeptID\}$   
 $Dept = \{DeptID, DeptName, ManagerID\}$

We'll use the following query: *List employee names and the names of departments that each employee works for.* The relational algebra expression for this query is:

$\pi_{EmpName, DeptName}(\sigma_{Emp.DeptID=Dept.DeptID}(Emp \times Dept))$   
 The SQL query is:

```
select E.*, D.*
from Emp E, Dept D
where E.DeptID = D.DeptID
```

The structural elements for this query are:

$Rel(q) = \{Emp, Dept\}$   
 $Sch(q, \pi) = \{(Emp, EmpName), (Dept, DeptName)\}$   
 $Sch(q, \sigma) = \{(Emp, DeptID), (Dept, DeptID)\}$   
 $Join(q) = \{(Emp, DeptID, =, Dept, DeptID)\}$

□

### 3.1 Structural Domination and Equivalence

Using the base structural elements defined in the previous section we define in this section the concept of structural equivalence. Structural equivalence provides the basis for comparing queries, which we use for visualization purposes.

Given two queries  $q_1$  and  $q_2$  we say that  $q_1$  *alpha*-dominates  $q_2$ , written  $q_1 \succeq_{\alpha} q_2$ , if  $q_1$  dominates  $q_2$  relative to some property  $\alpha$ . The properties that we consider for  $\alpha$  are the structural elements of the query, delineated below:

#### Relation Domination:

$Rel(q_1) \supseteq Rel(q_2) \rightarrow q_1 \succeq_{Rel} q_2$

#### Project Schema Domination:

$Sch(q_1, \pi) \supseteq Sch(q_2, \pi) \rightarrow q_1 \succeq_{Sch(\pi)} q_2$

#### Select Schema Domination:

$Sch(q_1, \sigma) \supseteq Sch(q_2, \sigma) \rightarrow q_1 \succeq_{Sch(\sigma)} q_2$

#### Join Domination:

$Join(q_1) \supseteq Join(q_2) \rightarrow q_1 \succeq_{Join} q_2$

Given these structural domination properties we can define the concept of structural equivalence. Again, given two queries  $q_1$  and  $q_2$  we say that  $q_1$  is *alpha*-equivalent to  $q_2$ , written  $q_1 \equiv_{\alpha} q_2$ , if  $q_1 \succeq_{\alpha} q_2$  and  $q_2 \succeq_{\alpha} q_1$ , again relative to some property  $\alpha$ . Accordingly, we define the following *alpha*-equivalent properties:

Relation Equivalent:  $q_1 \equiv_{Rel} q_2$   
 Project Schema Equivalent:  $q_1 \equiv_{Sch(\pi)} q_2$   
 Select Schema Equivalent:  $q_1 \equiv_{Sch(\sigma)} q_2$   
 Join Equivalent:  $q_1 \equiv_{Join} q_2$

In cases where queries may be *alpha*-equivalent in more than one way we extend *alpha* to be a logical expression of the properties. For example, if  $q_1$  and  $q_2$  are Join and Select Schema equivalent we denote the equivalence by  $q_1 \equiv_{Sch(\sigma), Join} q_2$ .

## 4 Visualizing Query Structure

Most visualizations rely upon the values of data to determine where to plot the graphic elements (points, lines, etc.) The order and scale of the data is depicted in the placement of graphic elements so as to inform the user of the relative difference between the plotted values. In the context of this paper, it is desirable to have some way of comparing structure of queries and then depicting the structural similarity in some graphical way. In this section we describe a visualization technique for graphically depicting query structure. Due to space considerations we focus our examples on queries that are Join equivalent.

We utilize a graph-based model for depicting query structure. A graph is represented by  $G = (V, E)$ .  $V = \{v_1, \dots, v_n\}$  is the set of vertices.  $E = \{(v_i, v_j) | v_i, v_j \in V\}$  is the set of edges in the graph. In this paper we use undirected graphs; however, future efforts will employ both directed and undirected edges.

We consider two types of vertices in the graphs we draw - relations and queries. Edges are drawn between each query and the relations contained in the query. In addition, edges are drawn between the relations if a join condition exists for some query involving those relations. Example 4.1 shows the graph generated for a single query.

### Example 4.1

Let  $q$  be the query  $\sigma_{R.A=S.A}(R \times S)$ . Figure 1 shows the graph, with the square symbol representing the query and each circle representing the relations  $R$  and  $S$ . Edges are drawn connecting the relations to the query. A single edge is drawn connecting the two relations to represent the join condition.

□

When more than one query is visualized the similarity between the queries is graphically derived from the graph structure in two ways. First, queries that are join equivalent are drawn as the same graph, with color being used to show an increased frequency for those queries. Second, relations are drawn only once, which allows for non-equivalent queries that share some (but not all) of the join conditions to become connected. Example 4.2 shows the graph generated for three queries.

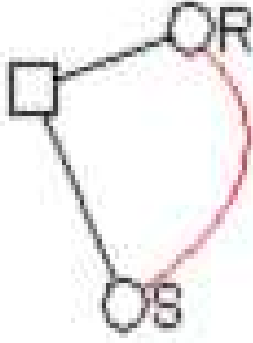


Figure 1: Graph for the query  $\sigma_{R.A=S.A}(R \times S)$ .

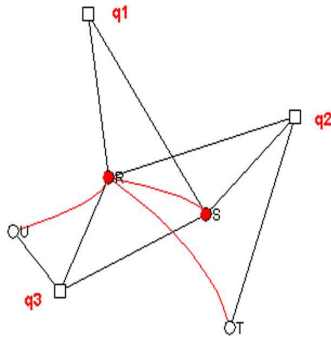


Figure 2: Graph depicting three queries.

#### Example 4.2

Let  $q_1 = \sigma_{R.A=S.A}(R \times S)$ ,  $q_2 = \sigma_{R.A=S.A \wedge R.B=T.B}(R \times S \times T)$ , and  $q_3 = \sigma_{R.A=S.A \wedge R.B=U.B}(R \times U)$ . Figure 2 shows the graph for these queries. Note that the graph demonstrates the dominance relation  $q_1 \preceq_{Join} q_2$ .

□

These two examples illustrate the basic technique we employ. However, one of the challenges in visualizing information is dealing with potentially large amounts of data. We have approached this problem architecturally in our prototype system by separating the equivalence testing from the graph drawing. This allows for the introduction of new graph drawing algorithms without impacting the underlying comparisons.

The system uses a radial placement technique for plotting the query symbols. Queries are plotted in the order they are encountered, with the first being positioned at the “12:00” position and the remainder drawn in a clockwise order. Relations are positioned using the mean of the co-

ordinates of the queries that use the relations. If a relation is used by a single query it is drawn to the outside of the radius.

The prototype system has been developed to operate in either a static mode or a dynamic mode. The static mode takes as input a collection of queries and generates the graph one time. The dynamic mode takes as input a stream of queries and grows the graph as changes occur.

The user can interact with the system by selecting queries with the mouse and then viewing those queries in a separate graph. This capability allows the user to simplify the search space as well as see the specific similarities between the selected queries. While in dynamic mode the graph is continually updated in all windows simultaneously. Figure 3 shows a medium-size graph of 100 queries. A user selection of 10 queries is shown in Figure 4.

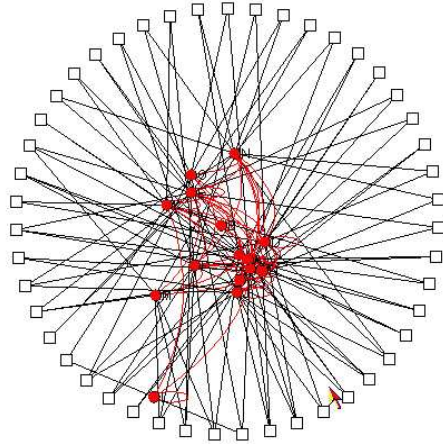


Figure 3: A graph of 100 queries.

An additional capability of the current prototype allows for a data model to be pre-loaded into the graph. The model defines the apriori assumptions of the data as defined by the designer. For example, if the model defines a relationship between two tables in the database, we would expect that queries involving those tables would exhibit the relationship with join conditions. Graphically, the model is overlaid on the graph of queries, which allows the user to verify the model, or more interestingly, see exceptions to the model. Figure 5 shows an example of this capability. Note in the figure that there are some examples of these exceptions.

## 5 Conclusion

We have presented a technique for visualizing the structural properties of queries. The technique demonstrates that graphical representations of queries can be used to ascertain properties of the underlying data. We employ a

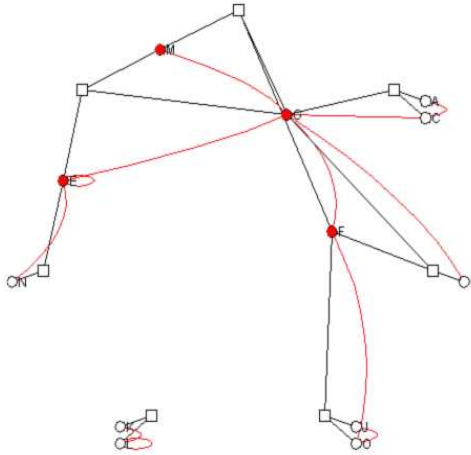


Figure 4: A subset of the 100 queries as selected by the user.

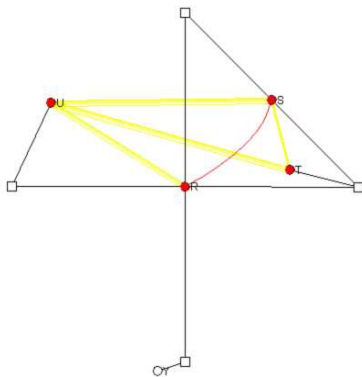


Figure 5: A query structure graph overlaid by a predefined data model.

graph-based visualization technique that mimics structure of queries as graph structures.

We envision a variety of uses for the techniques we have outlined. The following list of applications outlines future work we will undertake with this research:

**Model Discovery:** This problem involves the extraction of an unknown set of relationships based on usage of the database.

**View Selection:** Identifying useful views of the data can be used to streamline access to the data, especially for novice users.

**Schema Refinement:** Given a predefined model, what portions of the model are used with high frequency. This may suggest that denormalizing the data might be advantageous.

## References

- [1] AHLBERG, C., AND SHNEIDERMAN, B. Visual information seeking: Tight coupling of dynamic query filters with starfields displays. In *Proceedings of the ACM CHI'94 Conference* (1994), pp. 313–317.
- [2] ANGELACCIO, M., CATARCI, T., AND SANTUCCI, G. QBD\*: A fully visual query system. *Journal on Visual Languages and Computing* 1, 2 (1990), 255–273.
- [3] ANGELACCIO, M., CATARCI, T., AND SANTUCCI, G. QBD\*: A graphical query language with recursion. *IEEE Transactions on Software Engineering* 16, 10 (1990), 1150–1163.
- [4] CATARCI, T., COSTABILE, M. F., LEVIALDI, S., AND BATINI, C. Visual query systems for databases: A survey. *Journal of Visual Languages and Computing* 8, 2 (1997), 215–260.
- [5] CODD, E. F. A Relational Model for Large Shared Data Banks. *Communications of the ACM* 13, 6 (1970), 377–387.
- [6] CRUZ, I. F. DOODLE: a visual language for object-oriented databases. In *SIGMOD 1992, Proceedings ACM SIGMOD International Conference on Management of Data* (1992), pp. 71–80.
- [7] GROTH, D. P., AND ROBERTSON, E. L. An entropy-based approach for visualizing database structure. In *Proceedings of the Sixth IFIP Conference on Visual Database Systems (VDB6)* (2002).
- [8] PAREDAENS, J., DEN BUSSCHE, J. V., ANDRIES, M., GEMIS, M., GYSSENS, M., THYSSENS, I., GUCHT, D. V., SARATHY, V., AND SAXTON, L. V. An overview of GOOD. *SIGMOD Record* 21, 1 (1992), 25–31.
- [9] SHNEIDERMAN, B. Dynamic queries for visual information seeking. *IEEE Software* 11 (November 1994), 70–77.
- [10] TUFTE, E. R. *The Visual Display of Quantitative Information*. Graphics Press, 1997.