Graphical query languages for semantic database models

by BOGDAN CZEJDO, RAMEZ ELMASRI, and MAREK RUSINKIEWICZ

University of Houston
Houston, Texas

and

DAVID W. EMBLEY
Brigham Young University
Provo, Utah

ABSTRACT

Graphical representations of database schemas such as entity-relationship diagrams are commonly used to support database design. In this paper we discuss graphical representations of semantic data models and their application to interactive query languages. Operators that are appropriate for graphical query formulation are defined for several semantic data models. We also discuss a general method for implementing graphical interfaces to database systems that can be applied to a wide range of semantic data models.
INTRODUCTION

Graphic display can frequently enhance user understanding of complex objects. Since database schemas have considerable complexity, it is not surprising that several graphic representations, such as Bachman diagrams and entity-relationship (ER) diagrams, have emerged. These diagrams can conveniently represent database schemas and have been used extensively to support database design activities.

Graphic representations may also be used to support database query formulation. Rather than require a user to write a symbolic expression in a disciplined style using a formal query language, a graphical language could allow a user to formulate queries interactively, by working directly with some kind of diagram. This method of query formulation would take advantage of available graphical interfaces and pointing devices to provide a friendlier user interface to a database system and benefit both novice and experienced users. Two of the earliest query languages specifically designed for use with an interactive two-dimensional interface are QBE and CUPID. Graphical query interfaces are also discussed in several additional papers.

In this paper we present a method of query formulation for various semantic data models with natural graphical representations. For each data model, our approach is to define graphical operators that allow an end-user to manipulate a diagram until it represents a desired query. In the same way that a semantic-model diagram represents the schema of the database, a modified diagram (usually much smaller) can represent the query. From the end user's point of view the data in a stored database is manipulated in a manner consistent with changes made to a graphic representation.

We also describe the semantics for these operators and discuss ways of implementing them efficiently. We assume that an underlying database system exists that can be described by a relational schema and manipulated by relational operators. The graphical operators specified by a user are transformed into relational operators for processing.

Three data models will be discussed as an example of our approach. The entity-relationship model has a natural graphical representation in the form of ER-diagrams. The second example is based on the category relationship (ECR) model. This model constitutes an extension of the ER model by introducing the concept of a category. The third example is based on an extended relational model. In addition to relations, this model also contains connectors, which allow a natural graphical representation of a database schema.

The paper is organized as follows. First, a general approach to graphical query formulation for semantic data models is discussed. The application of our approach to interactive query formulation is illustrated for an ER model, for the ECR model, and for an extended relational model. We describe an implementation method applicable to these and other similar semantic data models and illustrate the approach by giving details for the ER model. Finally, conclusions are drawn.

GRAPHICAL QUERY FORMULATION

A graphical representation of a database using a particular semantic data model is an abstract description of the schema of the database. Views and queries can also be represented using this same abstract description, and thus, graphical query formulation can consist of manipulating a schema diagram so that it represents the query. The general approach is to discard unneeded portions of the diagram and modify the remainder of the diagram such that it defines the desired query. Diagram manipulating operators are defined to perform these actions. The final result of a query is produced when the graphically formulated query is applied to the current database instance.

A possible screen layout for a graphical interface would be to display a schema diagram together with the list of applicable operators and reserve an area on the screen for messages between the system and user. Large diagrams can be viewed using a windowing mechanism. A user manipulates a diagram by pointing to an operator and then to its operands. Depending on the operator, the user may also enter text in the message area. After each operation, a new diagram is displayed. The diagram displayed on the screen corresponds to the current view of the data and can always be interpreted as a query.

This approach to query formulation has the following advantages:

1. The query language is two dimensional. Pictorial diagrams that depict a view of the database schema are displayed and can be manipulated interactively.
2. Query formulation is flexible. A query can be formulated in many different ways since the order in which the diagram manipulating operators are invoked is often immaterial.
3. The user always has a convenient frame of reference. The current diagram reflects the current status of query formulation and is always a valid query.
4. The approach is applicable to a wide range of semantic data models.
5. An undo operator, which reverses the last operation(s), can be easily implemented by keeping a copy of the model state, corresponding to the previous step(s), on a stack.
6. Immediate feedback is provided whenever an operator is invalid in the current context. Thus, a user is assisted by being immediately informed about possible errors.
7. The intended query can be specified in several different ways corresponding to different levels of diagram reduction. The strategy to be used can be selected by the user.

We now illustrate how queries are formulated graphically for an ER model, the ECR model, and an extended relational model.

**GRAPHICAL QUERY FORMULATION FOR AN ER MODEL**

Figure 1(a) shows an ER diagram corresponding to a simple database for a university. Queries can be formulated by manipulating ER diagrams with the following operators.

*Relationship-Set Delete (R1)*: This operator removes the relationship set R1 from the diagram.

*Entity-Set Delete (E1)*: This operator removes the entity set E1 from the diagram. All relationship sets associated with E1 are also removed.

*Relationship-Set Project (R1, Z)*: This operator restricts the attributes of the relationship set R1 to the (possibly empty) set of attributes Z. Thus, attributes of relationship sets may be removed by invoking this operator.

*Entity-Set Project (E1, Z)*: Like relationship-set projection, this operator restricts the attributes of the entity set E1 to the (possibly empty) set of attributes Z. Entity sets with no attributes are useful for specifying indirect relationships, for example, the courses taught by a particular student's advisor.

*Relationship-Set Restrict (R1, e)*: This operator restricts a relationship set to the subset of relationship objects that satisfy a boolean restriction expression e.

*Entity-Set Restrict (E1, e)*: This operator restricts an entity set to the subset of entity objects that satisfies a boolean restriction expression e. This operator not only imposes a boolean restriction expression on entity set E1, but also alters the associated relationship sets to ensure that referential integrity is maintained. Thus, the operator guarantees that when an entity e1 is removed from an instance of E1 by condition e, every relationship instance involving e1 is also removed.

There are also operators for set union, set intersection, set difference, renaming attributes and entity and relationship sets, duplicating diagrams to formulate self-referencing queries, and creating new relationships among existing entities. A minimal set of operators has been identified and has been shown to have the expressive power of Codd's relational algebra.16

As an example of query formulation by manipulating ER diagrams, consider the following query: "Get names of all faculty members who are currently teaching the student whose Id# is '123456789'." We assume that the current ER diagram is the one shown in Figure 1(a).

This query can be specified in several phases. In the first phase, unnecessary entities and relationships are removed. This can be achieved by selecting the delete operator and pointing at ADVISES. This generates the following ER-algebraic operator:

Relationship-Set Delete(ADVISES)

In the second phase, the selection conditions are specified by choosing the restrict operator, pointing at the attribute Id# of STUDENT and entering the value '123456789'. This generates the following operator:

Entity-Set Restrict(STUDENT, Id# = '123456789')

In the third phase, unnecessary attributes of entities and relationships are deleted by selecting the operator project and pointing at STUDENT, CLASS, and Name (of FACULTY). This generates:

Entity-Set Project(STUDENT, φ)
Entity-Set Project(CLASS, φ)
Entity-Set Project(FACULTY, {Name})

At this point the schema diagram is as shown in Figure 1(b). This ER diagram specifies the query and can be interpreted as explained in the Implementation section.

**EXTENSIONS FOR THE ECR MODEL**

The ECR (Entity-Category-Relationship) model extends the basic ER model with the concept of category. An ECR diagram extends ER diagrams to graphically display categories. In the ECR model, there are two types of categories: subclass categories and generalization categories.

The graphical representations of categories is shown in Figure 2(a). Subclass categories are used to model a subset of entities from an entity set. In Figure 2(a), GRAD-STUDENT is a subclass category of the entity set STUDENT. STUDENT is called the defining entity set of GRAD-STUDENT. A category can have additional specific attributes that apply only to entities that are members of the category. In addition, a category can have specific relationships in which only entities that are members of the category can participate. In Figure 2(a), a specific attribute, Undergrad-School, and a specific relationship, IS-THESIS-ADVISOR, are specified for the GRAD-STUDENT category.

A subclass category specifies a restriction on the entities in the defining entity set. Hence, GRAD-STUDENT contains only the entities from STUDENT that are graduate students. The subclass category inherits all attributes and relationships of the defining entity set, since every entity of the category is also a member of the defining entity set. Hence, GRAD-STUDENT will also have all attributes of STUDENT.

For graphical query formulation, we can include additional operators to deal with subclass categories. All the operators that apply to entity sets can also apply to categories. Deleting a defining entity set E results in automatic deletion of all subclass categories of E. Deleting a subclass category, however, does not affect entities in the defining entity set. To ensure that entities in subclass categories are subsets of the...
entities in their defining entity set, restricting a defining entity set E results in automatic restriction of all subclass categories of E. Restricting a subclass category, however, does not affect entities in the defining entity set.

In addition, we can include an operator subclass-category-combine that restricts a defining entity set to those entities that are members of a category. Hence, this operator is similar to an entity-set restrict without an explicit condition—the condition is that entities must be members of the category.

**Subclass-CATEGORY-COMBINE** \((E_1, C_1)\). The category \(C_1\) is combined with its defining entity set \(E_1\) to yield a new entity set \(E_2\) with the same name as \(C_1\). The entities in \(E_2\) are restricted to those in \(C_1\), and the attributes of \(E_2\) are the union of the attributes of \(E_1\) and \(C_1\). \(E_2\) will participate in any specific relationships in which \(E_1\) or \(C_1\) participated. To maintain referential integrity, all relationship instances in which any removed entity of \(E_1\) participated are removed. All other subclass categories of \(E_1\) are removed from the ECR diagram.

The second type of category in the ECR model is the generalization category, which represents the union of entities from two or more disjoint entity sets that participate in some relationship in the same role. Figure 2(a) shows a generalization category \(\text{VEHICLE-OWNER}\) that is (a subset of) the union of the \(\text{FACULTY}\) and \(\text{STUDENT}\) entity sets. The entities in the category \(\text{VEHICLE-OWNER}\) participate in the role of owners in the \(\text{OWN}\) relationship with the \(\text{VEHICLE}\) entity set.

All the graphical operations that apply to entity sets can also be applied to generalization categories. However, a delete operation on one of the defining entity sets \(E\) automatically implies that entities in the generalization category that are members of \(E\) are automatically deleted. For example, a Entity-Set Delete(\(\text{FACULTY}\)) also deletes all \(\text{FACULTY}\) entities from the \(\text{VEHICLE-OWNER}\) category. This is necessary to maintain the category subset constraint. Similarly a restrict operation on one of the defining entity sets \(E\) implies that if entities removed from \(E\) are also in the generalization category, they are automatically removed.

A generalization category can also be combined with one of its defining entity sets to restrict members of the category to those entities in the defining entity set. This operator is called **Generalization-CATEGORY-COMBINE**.

**Generalization-CATEGORY-COMBINE** \((E_1, C_1)\): The category \(C_1\) is combined with one of its defining entity sets \(E_1\) to yield a new entity set \(E_2\). The name of \(E_2\) is the concatenation of the names of \(E_1\) and \(C_1\). The entities in \(E_2\) are restricted to those in \(C_1\) that are members of the entity set \(E_1\), and the attributes of \(E_2\) are the union of the attributes of \(E_1\) and \(C_1\). \(E_2\) will participate in any relationships in which \(E_1\) or \(C_1\) participated. To maintain referential integrity, however, all relationship instances in which any removed entity participated are removed. All defining entity sets of \(C_1\) (including \(E_1\)) are removed from the ECR diagram.

The two category combine operations are used to restrict a set of entities, as well as to cause explicit attribute inheritance
in the displayed ECR diagram. Hence, they are used for queries in which only some of the entities in an entity set or a generalization category are selected. For example, suppose we want to formulate the query to retrieve the names of all graduate students who have at least one parking ticket outstanding. Assume that the ECR diagram shown in Figure 2(a) represents the database schema.

In the first phase, we remove all unneeded entity sets, categories, and relationships. This is accomplished by pointing at the delete operator and then at FACULTY and CLASS, which generates the following operators:

Entity-Set Delete(FACULTY)
Entity-Set Delete(CLASS)

In the second phase, we graphically specify operators to combine GRAD-STUDENT with STUDENT and then to combine the resulting GRAD-STUDENT entity set with VEHICLE-OWNER. The operators generated are:

Subclass-Category-Combine(STUDENT,
GRAD-STUDENT)
Generalization-Category-Combine(GRAD-STUDENT,
VEHICLE-OWNER)

Finally, we use the graphical interface to restrict VEHICLE to those with at least one ticket outstanding, and then to project on the GRAD-STUDENT name. The operations generated are the following:

Entity-Set Restrict(VEHICLE, No-of-Tickets > 0)
Entity-Set Project(VEHICLE, {Name})
Entity-Set Project(GRAD-STUDENT-VEHICLE-OWNER, {Name})

Figure 2(b) shows the final reduced diagram. Since this is a regular ER diagram, it can be interpreted as discussed in the Implementation section.

GRAPHICAL QUERY FORMULATION FOR AN EXTENDED RELATIONAL MODEL

Figure 3(a) shows a schema diagram for an extended relational model. Each relation of the database is represented on the schema diagram by a relation descriptor consisting of the relation name and the relation attributes. We extend the relational schema diagram by adding connectors. Pairs of attributes of relation descriptors can be connected and a boolean-valued operator over the connected attributes can be specified. The model can also include connectors that relate more than two attributes. We formulate queries by manipulating diagrams for the extended relational model with the following operators.

Delete Connector (CI): This operator deletes connector C1 from the diagram.
Add Connector (R1, A1, R2, A2, 0): This operator creates a 0-comparison connector between attribute A1 of relation descriptor R1 and attribute A2 of relation descriptor R2. A1 and A2 must, of course, be 0-comparable attributes.
Delete Relation (R₁): This operator removes relation descriptor R₁ from the diagram and also removes any connectors associated with the removed relation descriptor.

Add Relation (N₁, Z, T): This operator creates a constant relation named N₁ whose attributes are given in the set Z and whose tuples are given in the set T, adds the relation to the stored database, and adds its relation descriptor to the model. We may use this operator along with add connector to restrict our query to particular constant values.

Delete Attributes (R₁, Z): This operator removes the set of attributes Z from the descriptor of relation R₁. Any connection descriptor that references a deleted attribute is also deleted.

There are also operators for combining relations using set union and set difference, explicitly reducing diagrams by joins, renaming attributes and relation descriptors, and duplicating diagrams to formulate self-referencing queries.¹⁵

As an example of query formulation for the relational model extended by connectors, we show how to manipulate the diagram to specify the sample query: “Get names of all faculty members who are currently teaching the student whose Id# is ‘123456789’.” We assume that the current diagram is the one shown in Figure 3(a).

As before, the query can be specified in several phases. In the first phase, unnecessary relations and connectors are removed. For our example a user points at the delete-connector operator and then at the connector to be removed. This generates the following algebraic operator:

\[
\text{Delete Connector}((\text{STUDENT, Advisor, FACULTY, SS\#,#}))
\]

In the second phase, the selection conditions are specified. This can be done by pointing at a graphical select operation, pointing at the Id# attribute (of STUDENT) and entering ‘123456789’. This generates the following operators:

\[
\text{Add Relation}(T₁, \{\text{Id#}\}, \{\langle\text{Id#}:'123456789'\rangle\})
\]

\[
\text{Add Connector}(\text{STUDENT, Id#, T₁, Id#,#})
\]

In the third phase, the attributes to be displayed are marked, resulting in the schema diagram of Figure 3(b) where marked attributes are underlined with stars. This diagram specifies the query and can be translated into executable code as explained next.

IMPLEMENTATION MODEL

The operators for the semantic data models can be implemented in several ways. If the underlying database management system supports a data manipulation language (DML) corresponding to the semantic data model (e.g., GORDAS¹⁷ for the ER model), our operators can be directly translated into equivalent semantic DML queries. An alternative and more general solution is to map graphical operators into equivalent algebraic operations on a corresponding relational database schema. This approach will be presented below using the ER model as an example.

A mapping from an ER diagram into an equivalent relational schema can be defined as follows. For each entity set, we assume the existence of a relation scheme whose name is the name of the entity set and whose attribute set consists of the attributes of the entity set plus a surrogate key attribute.¹⁸ For each relationship set, we assume the existence of a relation scheme whose name is the name of the relationship set and whose attribute set consists of the attributes of the relationship set plus the surrogate key attributes of the associated entity sets. For Figure 1(a), the derived relational database schema is shown below.

\[
\begin{align*}
\text{FACULTY} & (\text{eFACULTY, SS\#, Name, Office}) \\
\text{CLASS} & (\text{eCLASS, Course\#, Section\#}) \\
\text{STUDENT} & (\text{eSTUDENT, Id\#, Name, Address}) \\
\text{IS-TEACHING} & (\text{eFACULTY, eCLASS}) \\
\text{IS-TAKING} & (\text{eCLASS, eSTUDENT}) \\
\text{ADVISES} & (\text{eFACULTY, eSTUDENT})
\end{align*}
\]

The attributes prefixed with “e” are surrogate key attributes.
Each graphical operator corresponds to an operation that maps a set of relation instances into another set of relation instances. The initial instance is the current database state. For our example the initial set of relations is \{faculty, is-teaching, class, is-taking, student, advises\}. The first operator invoked for the query in the section on graphical query formulation for an ER model was relationship-set delete (ADVISES) which corresponds to removing the relation advises from the set of relations. The second operator invoked was entity-set restrict(STUDENT, Id# = '123456789') which corresponds to replacing the relation student with

\[ \sigma_{Id#='123456789'}\text{student} \]

and, to maintain referential integrity, replacing is-taking with

\[ \pi_{\text{STUDENT,CLASS}(\text{is-taking}) \sigma_{Id#='123456789'}\text{student} } \]

The last three operators invoked were entity-set project (STUDENT, \(d\)), entity-set project(COURSE, \(d\)), and entity-set project(FACULTY, [Name]). These operators each correspond to appropriate projections. The final set of relations is

\[ \pi_{\text{FACULTY,Name}(\text{faculty}, \text{is-teaching})} \]
\[ \pi_{\text{CLASS}(\text{class}, \text{is-taking})} \]
\[ \sigma_{\text{Id#='123456789'}\text{student}} \]

This set of relation instances is the database instance for the query in Figure 1(b). To produce a single table for this query, we join the relations in the final set of relation instances and project on the attributes of interest shown on the diagram.

Of course, actually manipulating the stored relation instances as the query is formulated, would be very inefficient. Instead, we can accumulate information about how to create the relation instances as the diagrams are manipulated, and thus obtain a relational algebra expression equivalent to the graphically specified query. For this example, the equivalent query is:

\[ \pi_{\text{Name}(\pi_{\text{FACULTY,Name}(\text{faculty}, \text{is-teaching})} \pi_{\text{CLASS}(\text{is-taking})} \sigma_{\text{Id#='123456789'}\text{student}) } \]

In such an implementation, the application of a graphical operator causes transformation of the diagram, and corresponding relational algebra expression(s), but does not affect the underlying stored database. This means that the proposed graphical interface can be treated as a front-end to an existing relational database. As a consequence, the relational algebra expression can be optimized before execution by the relational query processor.

This approach also has the significant advantage that the formal definition of the semantics of graphical operators can be described by providing translation rules to generate relational algebra expressions. Entity-relationship diagrams can be described formally as a pair that includes a set of entity-set descriptors and a set of relationship-set descriptors. Each operator transforms a particular ER diagram into another diagram. Most operators are partial and are valid only if certain enabling conditions are satisfied.

We enhance the above model by associating a relational algebra expression with each descriptor (entity-set descriptor and relationship-set descriptor). This relational algebra expression is referred to as the X-component and defines a set of tuples associated with the given set descriptor. Thus, when an operator is specified on an entity or relationship set, a corresponding relational-algebra operator is concatenated (using applicable syntax rules) with the X-component of the set descriptor. Hence, the state of the X-component for an entity or relationship set \(W\) defines the view generation for \(W\) as displayed on the diagram.

The semantics for a basic set of operators for our ER model are formally defined. Semantics for graphical operators for the ECR model, the extended relational model, and other semantic models can be defined in the similar manner.

**CONCLUSION**

A general approach to graphical query formulation for semantic data models has been discussed. Three data models—an ER model, the ECR model, and an extended relational model—have been used as examples. Our approach is to define graphical manipulation operators that allow queries to be specified by manipulating schema diagrams. Diagrams are transformed until they represent a desired user query. The resulting diagram (as well as all intermediate diagrams) can be interpreted in terms of the graphical representation of the data model used.

We have also explained how the operators can be defined and efficiently implemented. These graphical operators can be defined in terms of functions that operate on an abstract data model. Based on the definition of these operators, the result of formulating a query can be expressed as a relational algebra expression. Thus, it is unnecessary to manipulate the stored database while the schema diagrams are manipulated. This approach allows an efficient implementation of graphically-specified queries because the relational algebra expression can be optimized, using standard techniques, before it is executed.

Our method is general and is applicable to a wide range of semantic models. For each model, the graphical query interface provides a convenient and dynamically changing frame of reference. Immediate feedback is provided whenever an operator is invalid in the current context. Assistance in both formulating and understanding a query is provided at a higher level of abstraction, closer to the application domain of the end-user.

These graphical query interfaces can be implemented as a front-end to an existing (relational) database system. Hence, multiple interfaces can be implemented over the same underlying database system, so a user can select the interface that corresponds to his/her favorite data model.
REFERENCES
