A Visual Query Language for an ER Data Model

Bogdan Czejdo
Marek Rusinkiewicz
University of Houston

David Embley
Brigham Young University

Venugopal Reddy
Davis & Elkins College

Abstract

In this paper we discuss a graphical representation of an ER data model and its application to interactive visual query languages. We present a method of implementing an ER query interface for a relational Database Management System. The approach is based on graphical representation and interactive manipulation of ER database schemas. During the process of query formulation the information needed to generate an equivalent SQL expression for a relational database is accumulated. The sample interactions between the user and a system are given. The graphical query formulation process is formally defined. The formal definitions presented in the paper were the basis for an efficient visual query language implementation.

1. Introduction

Graphical representations of database schemas such as entity-relationship (ER) diagrams are commonly used to support database design. Graphical representations can also be used as an aid in formulating queries. A variety of interactive graphical query interfaces were described in the literature [4, 5, 6]. Several interfaces have been based on the relational [2] and the ER model [1, 3, 7]. They can provide valuable assistance in formulating queries.

In this paper we discuss a graphical representation of an ER data model and its application to interactive query languages. We present a method of implementing an ER query interface for a relational Database Management System. The approach is based on graphical representation and interactive manipulation of ER database schemas. The graphical query formulation process is formally defined. The information reflecting the current state of the query formulation process is maintained in an underlying model. The model described in [1] is modified and extended, so that the information stored in it can serve two purposes. First, it allows an extended ER diagram corresponding to the current state of the query to be displayed on the terminal screen. Second, it can be used to generate an equivalent SQL expression for a relational database. The formal model presented in this paper was the basis for an efficient graphical query interface implementation.

The paper is organized as follows. In the next section, a formal definition of an ER view and query formulation model together with the appropriate operators is given. Section 3 presents the design and implementation of an interactive graphical ER interface for view and query formulation. The process of generation of an SQL expression from the information contained in the model is described in Section 4. The Summary presents some conclusions and directions for further work.

2. View Definition and Query Formulation Model

An ER database schema can be defined by a set of entity and relationship type descriptors [1]. Each entity type descriptor contains the entity type name and the attribute names. Each relationship type descriptor contains the relationship type name and the attribute names. The model can be further enhanced to accommodate the graphical representation of the view definition and query construction which includes the position of an entity and relationship type icons on the terminal screen, the description of selected attributes and the specification of conditions.

View definition and query formulation model ERC for the ER database is a three-tuple (E, R, C), where E is the set of entity type descriptors, R is the set of relationship type descriptors and C is the set of condition descriptors.

The entity type descriptor is a 5-tuple (N, T, D, A).

Here, N denotes the original name of the entity type. T is the displayed name (alias) and initially is identical to N. D is the position of the entity type on the terminal screen. A is a set of attribute descriptors for the given entity type.

The relationship type descriptor is a 8-tuple (N, T, P, D, Ea, Eb, A, V).

Here, N denotes the name of the relationship type. T is the displayed name (alias) and initially is identical to N. P is the position of the relationship type on the terminal screen. D is a flag indicating whether the entity type is to be displayed on the screen. Ea and Eb are the positions of the entity type icons corresponding to the specified relationship on the terminal screen. A is a set of attribute descriptors for the given entity type. V is a set of condition descriptors for the given relationship type.

The attribute descriptor is a 6-tuple (N, T, D, S, O, K).

Here, N denotes the name of the attribute. T is the displayed name (alias) and initially is identical to N. D is a flag indicating whether the attribute is to be displayed on the screen. S is a flag indicating whether the attribute is selected for projection and can have two values 'Selected' and 'Not_Selected'. O denotes the ordering direction that can be either 'Ascending', 'Descending' or 'Not_Ordered'. K denotes the primary key of the entity type and can be 'Key' or 'Not_Key'.

The relationship type descriptor is a 8-tuple (N, T, P, D, Ea, Eb, A, V).

Here, N denotes the name of the relationship type. T is the displayed name (alias) and initially is identical to N. P is the position of a relationship
type icon on the terminal screen. D is a flag indicating whether the relationship type is to be displayed on the screen. Ea and Eb are distinct entity type descriptors in E. A is a set of attribute descriptors for the relationship type. The attribute descriptors for a relationship type are similar to the attribute descriptors for the entity type except that their A component is always 'Not-Key'. V indicates the origin of the relationship type. It is null when the relationship type belongs to the original ER schema. Otherwise, the relationship type is virtual and is derived from other entity and/or relationship types. In such case, V is a pair (ROP, ERCa) where Ea and Ra are sets of names of entity and relationship types respectively used in derivation of the virtual relationship type descriptor. This definition of the relationship type descriptor can be extended to allow relationships among several entity types.

The condition descriptor is a 4-tuple (ERa, Aa, OPa, Za), where ERa is an entity or relationship type descriptor, Aa is an attribute of ERa, Za is a constant from the domain Z (the same as the domain for values of the attribute Aa), OPa is an operator imposing a condition on the values of the attribute Aa based on the constant Za.

The ERC model can be represented graphically as a diagram. As an example let us consider University Database that is described by three entity types FACULTY, STUDENT, COURSE and four relationship types Advises, IsTaking, IsAuditing and Teaches. It can be represented by a diagram shown in Figure 1.

![Figure 1. ER diagram for University Database.](image-url)
the same meaning. Instances of elements of a set are denoted by concatenation of the set name and a subscript, e.g. C1 is an instance of a condition descriptor belonging to C and ER1 is an instance of an entity or relationship type descriptor belonging either to E or to R. A component of any object is denoted by dot-notation, e.g. R1.N is the name of the relationship type descriptor R1. Each operator is defined below by describing the modifications it causes in the ERC model.

2.1 Display

The Display operator displays on the terminal screen the icons corresponding to the attributes of the specified object. The object can be an entity type descriptor or a relationship type descriptor. The argument list for this operator consists of an object ER1. When this operator is applied, the D component of each attribute descriptor of ER1 is updated.

Operation: Display(ER1)
Action: for each A1 in ER1
        A1.D := 'Displayed'

2.2 Hide

The Hide operator allows the user to remove from the terminal screen the icons corresponding to the specified object. The object can be an entity type descriptor or a relationship type descriptor. The argument list for this operator consists of an object AER1. When this operator is applied, the icons corresponding to the attributes of the specified object are removed.

Operation: Hide(AER1)
Action: if AER1 is an attribute descriptor then
         AER1.D := 'Not-Displayed'
        else
         for each A1 in AER1
         A1.D := 'Not-Displayed'

2.3 Relationship Intersection, Union, and Difference

The Relationship Intersection, Union and Difference operators permit users to combine compatible relationship types between the same two entity types. Relationship types are compatible if they have the same set of attributes. They may be combined through intersection, union or set difference. The argument list for this operator consists of two relationship type descriptors R1 and R2. The new relationship type descriptor is created with the names obtained by appending the names of combined relationship type descriptors. Its attribute descriptors correspond to attribute descriptors of combined relationship type descriptors. The position of the icon corresponding to the new relationship type descriptor is the position of the icon corresponding to the first relationship type descriptor. The V component of the new relationship type descriptor is the pair (ROPa, ERCa) where ROPa is a reduction operator, equal to one of 'Relationship_Intersection', 'Relationship_Union' or 'Relationship_Difference' and ERCa is a pair (Eset, Rset). Eset is empty. Rset contains names of R1 and R2. The D component of the combined relationship type descriptors is set to 'Not_Displayed'.

Operation: Relationship_Union(R1, R2)
Action: Create a new relationship type descriptor R3
         R3.P := R1.P
         R3.T := R3.P
         R3.A := (Relationship_Union, (Eset, Rset))
         R1.D := 'Not_Displayed'
         R2.D := 'Not_Displayed'

2.4 Move

The Move operator re-positions the icon of a specified object. The object can be an entity or relationship type descriptor. The argument list for this operator consists of two elements: the object ER1 and a new position P1. When this operator is applied, the P component of ER1 is updated.

Operation: Move(ER1, P1)
Action: ER1.P := P1

2.5 Rename

The Rename operator allows a user to change the displayed name of a specified object. The object can be an attribute, an entity type or a relationship type descriptor. The argument list for this operator consists of an object AER1 and a string NewName. When this operator is applied, only the T component of the object is updated so that the original name is preserved.

Operation: Display(AER1, NewName)
Action: AER1.T := NewName

2.6 Delete

The Delete operator allows a user to remove from the terminal screen the icon corresponding to the specified object. This operator can be applied to an entity type or to a relationship type. When applied to an entity type, it removes the icon corresponding to the specified entity type descriptor E1. In addition to that, the D component of all relationship type descriptors in which E1 participates are modified.

Operation: Delete(E1)
Action: for each relationship type descriptor R1 such that R1.Ea = E1 or R1.Eb = E1
         R1.D := 'Not_Displayed'
         E1.D := 'Not_Displayed'
When applied to a relationship type, it changes the D component of the specified relationship type descriptor R1.

Operation: Delete(R1)
Action: \( R1.D := '\text{Not-Displayed}' \)

2.7 Project

The Project operator specifies the attribute of an entity or relationship type whose values are to be displayed when the query is executed. The argument list for this operator consists of three elements: an entity or relationship type descriptor ER1, an attribute A1 from ER1, and an ordering direction OT1 from the set OT. The ordering direction is optional and therefore OT1 can have 'Not_Ordered' value. If more than one attribute is selected with the ordering direction the priority could be specified by using additional values for OT1 such as 'Ascending_Secondary' etc. When the project operator is applied, the S component of the selected attribute descriptor A1 is changed and the O component of the selected attribute descriptor A1 can be modified.

Operation: Project(ER1, A1, OT1)
Action:
\[
\begin{align*}
\text{if A1.S is 'Not_Selected' then} & \\
A1.S & := 'Selected' \\
A1.O & := OT1 \\
\text{else} & \\
A1.S & := 'Not_Selected' \\
A1.O & := 'Not_Ordered'
\end{align*}
\]

2.8 Restrict Values

The Restrict Values operator allows a user to construct a condition that is used in the process of data retrieval. The argument list for this operator consists of four elements: an entity or relationship type descriptor ER1, an attribute A1 from ER1, a boolean-valued operator OP1 and a constant \( Z1 \) from the domain of A1. When this operator is applied, the new condition descriptor is created and added to the set of C.

Operation: Restrict(ER1, A1, OP1, Z1)
Action:
\[
C := C + (ER1, A1, OP1, Z1)
\]

In the next section we show examples and explain how the ERC operators can be invoked graphically.

3. Graphical Query Formulation

The basic ERC operators described in the previous section can be implemented using a graphical interface. Figure 2 shows a possible screen layout for University Database.

Listed on the left side of the screen are the operators names such as:

- **Display**: To display attribute icons on the terminal screen.
- **Hide**: To remove attribute icons from the terminal screen.

![Figure 2. Basic screen layout for University Database](image)

A user invokes operations by first using a pointing device to select an operator from the main menu described above. Then, depending on the operator, the user either points to an element of the diagram or makes a selection from the provided sub-menu or enters text in the message area.
If the operator is to be applied two or more times in succession, it need not be repeatedly selected. For example, when the Delete operator is chosen, several entity and relationship types can be deleted by just selecting the Delete operator and then pointing successively at the entity and relationship type icons to be deleted from the diagram.

For several operators the user is presented with the sub-menu. When the Project operator is chosen, the sub-menu appears that allows to select an ordering option ("ASC" or "DESC"). This selection is optional. Then the user is expected to point to the icon corresponding to a projected attribute. When the Restrict operator is chosen, after the user selects the icon corresponding to an attribute descriptor, the sub-menu containing boolean valued operators $\leq$, $\geq$, $=$, $\neq$, $><$ and $<=$ appears on the screen. After choosing the boolean valued operator, a constant value is provided.

All operators can be invoked graphically in a similar way. As an example let us consider the query: *List the names of faculty members teaching course 'cs4310'.* Assuming again that the diagram of Figure 5 is initially displayed, the query can be specified by the following sequence of user's actions:

1. Point at the operator 'Delete'.
2. Point at the icon corresponding to the entity type 'STUDENT'.
   As a result of the above actions, the display flag $D$ of the entity type descriptor with the name 'STUDENT' and the relationship type descriptors with the names 'IsTaking', 'IsAuditing' and 'Advises' are set to 'Not Displayed'. The icons corresponding to these four objects are deleted from the terminal screen.
3. Point at the operator 'Display'.
4. Point at the icon corresponding to the entity type 'FACULTY'.
   As a result of the above actions the $D$ component of the attribute descriptors of 'FACULTY' is set to 'Displayed'. Attributes of 'FACULTY' are displayed on the screen.
5. Point at the icon corresponding to the entity type 'COURSE' ("Display" is still active).
   As a result of the above actions the $D$ component of the attribute descriptors of 'COURSE' are set to 'Displayed'. Attributes of 'COURSE' are displayed on the screen.
6. Point at the operator 'Project'.
7. Point at the attribute 'name' of the entity type 'FACULTY'.
   As a result of the above action the $S$ component of the attribute descriptor with the name 'name' gets the value 'Selected'. The icon corresponding to the attribute 'name' is darkened on the terminal screen.
8. Point at the operator 'R Value'.
9. Point at the attribute 'csname' of the entity type 'COURSE'.
10. Point at the boolean valued operator '='.
11. Enter the value 'cs4310'.

As a result of the above actions the new condition descriptor (COURSE,csname, $\leq$, 'cs4310') is added to the $C$ set. The icon corresponding to the condition descriptor is displayed on the terminal screen adjacent to the attribute 'csname' icon.

The resulting diagram on the terminal screen is shown in Figure 3.

![Figure 3](image)

The diagram in Figure 3 can be described by the ERC model which components are used to generate SQL query expressions.


Each ERC schema corresponds to an underlying relational schema [1]. Here, we assume that each entity and relationship type descriptor corresponds to a single relation scheme. The relation scheme corresponding to an entity type descriptor $E_1$ has the attributes and the key of $E_1$. The relation scheme corresponding to a relationship type descriptor $R_1$ has two attributes in addition to those in $R_1$. These additional attributes are foreign keys, namely the primary keys of the relation schemes corresponding to $R_1.E_a$ and $R_1.E_b$. 

169
As an example let us consider University Database of Figure 2. The ER schema corresponding to the initial state of ERC model described in Section 2, can be mapped into the following relational schema:

- Faculty (ssno, name, salary)
- Student (idnum, sname, addr)
- Course (csnum, csname, cshrs)
- Advises (ssno, idnum)
- IsTaking (idnum, csnum)
- IsAuditing (idnum, csnum)
- Teaches (ssno, csnum)

For the given mapping of an ER schema into a relational schema, it is possible to generate SQL expressions from any state of the ERC model. The generated SQL expressions are of two types: view definition and query expression. The query expression consists of four clauses: SELECT-clause, FROM-clause, WHERE-clause, and ORDER-BY-clause. The last two clauses are optional. The view definition contains a view name and a query expression.

The clauses of the query expression are generated from the current state of the model in the following way. The SELECT-clause can be obtained by creating a list containing attribute names from all 'Selected' attribute descriptors in the 'Displayed' entity or relationship type descriptors. The FROM-clause can be obtained by creating a list containing names of all 'Displayed' entity and relationship type descriptors. The WHERE-clause can be obtained by combining all terms corresponding to join conditions and all terms specified by condition descriptors. The terms corresponding to join conditions are generated from the remaining relationship type descriptors. Each relationship type descriptor R1 is translated into two join conditions:

concatenate(R1.N, ' ', ANa, ' = ', R1.Ea.N, ' ', ANb) where ANa is the name of the 'Key' attribute descriptor in R1.Ea
where ANb is the name of the 'Key' attribute descriptor in R1.Eb

By default all terms are combined using the 'AND' connector. For terms corresponding to condition descriptors this can be changed by explicitly selecting the 'OR' modifier. The ORDER-BY-clause can be obtained by creating a list containing attribute names from all attribute descriptors with a non-null O component in the 'Displayed' entity and relationship type descriptors. If the attribute name is not unique then it is fully qualified by appending the the entity or relationship type name.

In the similar way for each virtual relationship type in an ERC diagram, the view can be defined. Let us consider the query specified by Figure 3 in Section 3. The SQL expression generated for this query is:

```sql
SELECT name
FROM FACULTY, Teaches, COURSE
WHERE FACULTY.ssno = Teaches.ssno
AND Teaches.csnum = COURSE.csnum
AND COURSE.csname = 'cs4310'
```

5. Summary

In this paper we have presented a method of designing a front-end graphical query interface to a relational Database Management System. The queries are specified by manipulating ER schema diagrams displayed on the terminal screen. Diagrams are transformed until they represent a desired user query.

During the process of query formulation the information needed to generate an equivalent SQL expression is accumulated in the defined ERC model. We have described the basic ERC operators and explained how they modify the ERC model. These operators can be efficiently implemented. The stored relational database is not affected by the manipulations involving schema diagrams until the complete SQL expression is generated. Such expression can be optimized by a standard SQL query processor.

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 the query is provided at a higher level of abstraction, closer to the application domain of the end-user.

A graphical query interface based on the described approach has been implemented using a Macintosh computer as a front-end to a relational Database Management System. The scope of the language is equivalent to a large subset of SQL. Currently we are working on removing the limitations of the system by allowing for composite keys and computations. In the next version of the system updates using an ER graphical interface will be also allowed.

References