2-1998

A visual object-relationship query language for user-database interaction

Keng SIAU
Singapore Management University, klsiau@smu.edu.sg

PERGAMON

Telematics and Informatics 15 (1998) 103–119

TELEMATICS and INFORMATICS

A visual object-relationship query language for user–database interaction

Keng Siau*

Department of Management, 209 College of Business Administration, University of Nebraska-Lincoln. Lincoln, NE 68588-0491, USA

Abstract

User–database interaction has a direct and immediate effect on the effectiveness and efficiency of database end users. Traditional query languages like SQL and QBE require end users to understand the underlying data structure in the database. This is a burden on end users, especially novice end users who have little technical knowledge or understanding of database. To alleviate the need for end users to know the logical database organization, this paper proposes the use of an object–relationship (OR) model and a formal high-level visual query language as the interface. Using this interface, end users communicate only domain knowledge with the system without the need to specify the storage structure or search strategies. The end users deal only with the conceptual model of the database and not the actual physical implementation of the database. To demonstrate the expressive power of the visual query language, this paper also provides a proof to show that it is relationally complete and hence is as powerful as traditional languages such as SQL.

Keywords: User–database interaction; Iconic interface; Visual query language; Object–relationship model

1. Introduction

To the end users, the user-database interface is the database system. The effectiveness and efficiency of information retrieval depend heavily on the user-friendliness of user–database interface. For a long time, user–database interface has been beyond the reach of many end users. One of the main reasons for this unfortunate state of affairs is the low-priority assigned to user–database interaction research. As a result, there is a lack of theories and knowledge on what factors contribute to good user–database interface.

*Tel: + 1-402-472-3078; Fax: + 1-402-472-5855; e-mail: klsiau@unlinfo.unl.edu

0306-4549/98/$—see front matter
PII: S0736-5853(98)00007-0

Another reason that contributes to the unfriendly user–database interface is the complexity of data models. Data models form the foundation of user–database interface. Simply speaking, a data model is a collection of high-level data description constructs that hide many low-level storage details (Ramakrishnan, 1998). A data model shows the way data is conceptually structured (Price Waterhouse, 1997). A complex data model will require a complex query language to manipulate and retrieve data. An abstract data model, will result in abstract and convoluted query languages. Shortcomings in the semantic expressive power of traditional data models, such as relational data model led to interest in semantic data models (Elmasri and Navathe, 1994; Ramakrishnan, 1998). The increased modeling power is derived from the ability of these newer models to handle semantically meaningful objects rather than normalized tuples or single records. With the growing popularity of object-oriented approach, object–relationship model was introduced (Embley et al., 1992; Martin and Odell, 1992). The object–relationship model is the data model for the proposed user–database interface.

This paper is organized as follows: the first part of the paper describes the OR data model and the visual query language, VORQL. A language is not useful unless it is powerful enough to support the necessary database retrieval functions. The benchmark to determine the expressive power of a database query language is relational completeness. The second part of this paper presents a proof to show that the new language is relationally complete and hence as powerful as traditional relational query languages such as SQL and QBE.

2. Object–relationship (OR) data model

The object–relationship (OR) model adopts the natural view that the real world consists of objects and relationships. It is based on the object-oriented approach which is rooted in object-oriented programming. The two main constructs in the OR model are object and relationship. An object is a uniquely identifiable entity that contains both the attributes that describe the state of a real-world object and the actions that are associated with a real-world object (Connolly et al., 1996). A relationship establishes a logical connection among objects (Embley et al., 1992). The strengths of the OR model are its high-level of abstraction, encapsulation, and inheritance properties (Yourdon, 1994). The biggest strength of the OR model, however, is the naturalness of the OR constructs—object and relationship. By having a natural model as the foundation of user–database interface, the task of designing user–friendly database interface is already half-accomplished. Norman (1986), for example, argued that many of the problems with interface could be related to the difficulties of linking our psychological goals and constructs to the physical variables of the task. In designing user–database interface, it is important to distinguish the conceptual model of the database from the actual implementation of the database [as suggested by Moran (1981) for interactive computer systems].

In this paper, we describe a visual query language designed for the object–relationship (OR) model. The main objective of this language is to provide for data

independence. Data independence is defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. Two forms of data independence can be defined—logical and physical data independence. Based on the ANSI/SPARC DBMS framework (Tsichritzis and Klug, 1978), logical data independence can be defined as the capacity to change the conceptual schema without having to change the external schema or application programs (Connolly et al., 1996; Ramakrishnan, 1998). Physical data independence is the capacity to change the internal schema without having to change the conceptual (or external) schema (Elmasri and Navathe, 1994; Hansen and Hansen, 1996). The new query language, Visual Object–Relationship Query Language (VORQL), provides for both physical and logical data independence. With this interface, end users need not worry about the physical or logical structure of the database. They communicate with the database systems using concepts that exist in the real world (i.e. objects and relationships).

3. Visual object–relationship query language

VORQL aims to provide end users both physical and logical data independence. The user interface for this query language is shown in Fig. 1. To facilitate ease of use, the interface is based on icons, menus, and visual form-structures. According to Siau and Nah (1997), "the iconic interface is especially important for novice users who use the interface on an infrequent and ad-hoc basis." A simple point-and-click interface is used for specifying queries as far as possible.

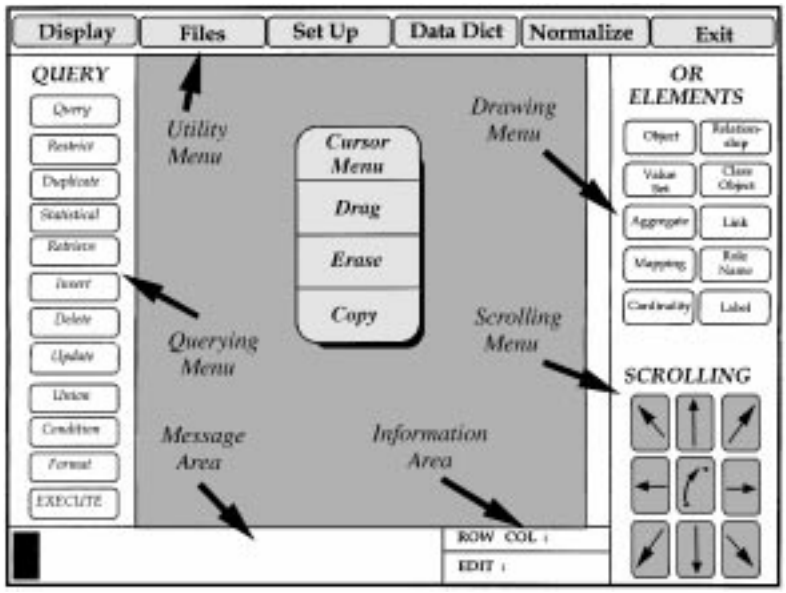

Fig. 1. User-Database Interface

The querying is based on the OR schema that is depicted on the interface. This interface is different from the traditional relational interface where the users have to remember the structure of the relational schema. Because our working memory is limited to a capacity of only 7 ± 2 chunks (Miller, 1956), it is almost impossible for novice end users to manipulate the database structure in the working memory (the database structure is not in the long term memory because these are ad-hoc and infrequent users) and at the same time trying to formulate the SQL queries. By depicting the schema on the interface and allowing users to specify queries based on the schema, the required working memory is significantly reduced. As argued by Larkin and Simon (1987), by a diagram is (sometimes) worth ten thousand words''. The querying process of VORQL is discussed below.

3.1. The querying process

The querying icons are displayed on the left of the interface. The querying icons include the QUERY, RESTRICT, DUPLICATE, STATISTICAL, RETRIEVE, INSERT, DELETE, UPDATE, UNION, CONDITION, FORMAT, and EXECUTE icons. Queries are formulated visually in VORQL by clicking within the vicinity of the objects, relationships, or attributes to be included in the query using a mouse and specifying the condition using various form structures. There are five steps in VORQL query specification: Duplicate, Restrict, Action, Condition, and Format.

3.1.1. DUPLICATE step

This is required for queries that need to make multiple distinct references to the same object or relationship. By first clicking on the DUPLICATE icon and then on an object or a relationship, a duplicate object or relationship will be created on the screen. To differentiate the various occurrences of the same object, the system will by default append a unique integer number (1,2,. . .) to the different occurrences. The user can overwrite this default by providing a new name. The purpose of this step is similar to the use of ''range variable'' in SQL or ''correlation variable'' in standard SQL.

3.1.2. RESTRICT step

The user can restrict the OR schema to only those objects and relationships of interest in the query. In other words, those objects and relationships not of interest will be removed from the screen. There are two approaches. The user can click the STAY icon on the pop-up menu of the RESTRICT icon to select the required objects and relationships. This is efficient if the number of required objects and relationships is small. Alternatively, the user can click the REMOVE icon and select those objects and relationships to be removed from the schema. This is efficient when the number of objects and relationships to be removed is small.

3.1.3. ACTION step

There are four categories of queries: Retrieve, Insert, Delete, and Update. Each of these categories can be activated by choosing the respective operator icons on the interface. The four icons are:

(a) RETRIEVE

Its function is similar to the Select-clause in SQL. The user selects the attributes to be retrieved by clicking on them. To output all the attributes of an object or a relationship, the user simply clicks within the vicinity of the object or relationship.

(b) INSERT

The user selects an object or a relationship where a new instance is to be inserted. The values for the new instance are specified using Insert-form, which will be discussed under the Condition step.

(c) DELETE

The user selects an object or a relationship where an instance or a group of instances is to be deleted. The condition for the deletion is specified using the Delete-form.

(d) UPDATE

The user selects an object or a relationship where an instance or a group of instances is to be updated. The update condition is specified using the Update-form.

3.1.4. CONDITION step

Following the Action step, the user will click the CONDITION icon to start the Condition step. Condition for the query is specified using four possible form structures: Condition-form, Insert-form, Delete-form, and Update-form. The system is intelligent enough to display the appropriate form structures to guide the user in this phase. The four possible cases are:

(a) RETRIEVE action

If the user selects the RETRIEVE icon in the Action step, a Condition-form is displayed in this step to allow the user to enter the condition for the query. The attributes in the Condition-form are generated from those objects and relationships restricted in the RESTRICT step. For example, to get the details of suppliers that supply red parts stored in London, the condition can be specified as shown in Fig. 2.

Note that the names of the objects and relationship, Supplier, Part, and Supplies, are displayed together with their attributes but on different rows. Those conditions specified on the same line (or row) will be interpreted as conjunction (i.e. AND) and those on different lines as disjunction (i.e. OR). For sophisticated and expert users, a condition line is provided for them to

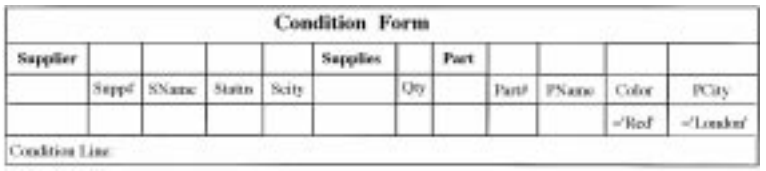

Fig. 2.

enter the condition if they choose to do so. This provides the flexibility of both visual and command-line interface.

(b) INSERT action

If the user selects the INSERT icon in the Action step, the system will automatically display an Insert-form when the user clicks the CONDITION icon. The attribute values for the new instance can then be entered into the Insert-form. The user can enter more than one instance by simply entering them on consecutive rows. For example, if the user specifies that s/he wishes to insert a new instance into Supplier entity in the Action step, the system will display the Insert-form after the user clicks the CONDITION icon. The user can then specify the values of the various attributes as shown below in Fig. 3.

(c) DELETE action

A Delete-form will be displayed for the user to enter the condition if the user selects the Delete icon in the Action step. For example, the following figure shows the specification of condition if the user wants to delete all projects in Chicago (Fig. 4).

(d) UPDATE action

For the update action specified in the Action step, an Update-form is provided for the user to enter the old value and the new value of the attribute. The Update-form for updating the color of all red parts to green is shown below in Fig. 5.

(e) FORMAT step

The user can also specify his/her own output format. Assuming that the user selects the Supplier and Part objects in the Restrict step, the following Format-form will be shown when the user clicks the FORMAT icon (Fig. 6).

The order of the instances returned for a query can be specified using the Sort By operator. The default order of sorting is in ascending order. To reverse the order, the user can enter the keyword Desc (i.e. descending) into the appropriate box. The user can also specify sorting on more than one attribute. The order of this combination is

Insert Form				
Supplier				
	Supp#	SName	Status	SCity
	=50	='Elite Distributor'	=5	='New York'

Fig. 3.

Delete Form			
Project			
	Proj#	JName	JCity
			='Chicago'
Condition Line:			

Fig. 4.

Update Form				
	Part#	PName	Color	PCity
Old			='red'	
New			='green'	

Fig. 5.

Format Form										
	Supplier					Part				
Sorted By		Supp#	SName	Status	SCity		Part#	PName	Color	PCity
Header										
Footer										
Eliminate Duplicate Records:	Yes	No								

Fig. 6.

determined by entering integer values (i.e. 1, 2, etc.) into the boxes. The functions of header and footer are self-explanatory. The last row of the Format-form is for user to specify whether redundant duplicate instances are to be eliminated in the output. The default is set to NO.

3.2. Other querying icons

Most of the querying icons have been described in the previous section. This section describes the remaining icons: QUERY, STATISTICAL, UNION, and EXECUTE icons. For every new query or subquery, the user needs to first select the QUERY icon. This is to indicate to the system that the subsequent Restrict step, Action step, and Condition step are meant for a new query. The system allows the user to specify the query and store it before execution. Those queries specified but not yet executed will be depicted as square objects on the OR schema with the query identifier enclosed. This provides VORQL the capability to accept nested queries as subsequent queries can refer to these stored queries in their specification. Each query can be named by the user or by the default numbering generated by the system.

The UNION icon is meant for connecting together two or more query specifications. The STATISTICAL icon is for the user to specify statistical operations on selected attributes. A form is provided for this purpose. For example, if the user specifies the attributes of Supplier object for output in the Action step and then clicks the STATISTICAL icon, the following form is provided (Fig. 7).

Standard operations such as COUNT, SUM, AVG, MAX, and MIN are provided in the form. In addition to these standard statistical functions, the Group By operation and the specification of Group Condition operation are also included in the Statistical-form. The COUNT function returns the number of instances or values specified in a query. The functions SUM, MAX, MIN, and AVG are applied to a set or multiset of numeric values and return respectively, the sum, the maximum

Statistical Form					
	Supplier				
		Supp#	SName	Status	SCity
Count					
Sum					
Avg					
Max					
Min					
Group By					
Group Condition:					

Fig. 7.

value, the minimum value, and the average of those values. The Group By function allows the user to apply these statistical functions to subgroups of instances based on the same value of the grouping attributes. Queries are not executed until the user selects the EXECUTE icon. During execution, subqueries will be linked to the main query and the whole query will be translated to Standard SQL query before sending it to the underlying relational system.

3.3. Features of VORQL

It would be user-friendlier and definitely more appealing to allow the user to specify the condition of the query using mouse, icons, and pictures rather than forms. This would be an ideal user-database interface for novice end users. However, this approach is limited in power and the specification is usually ambiguous. On the other hand, using the conventional command-line approach allows complex condition to be specified unambiguously. But this requires the users to learn the exact syntax of the language and to follow the syntax rigidly—a far from ideal user–database interface. A viable approach at this point in time is a visual form-based approach. The visual approach for specifying condition using forms provides the capability of specifying complex condition unambiguously and less rigidly. The form structures in VORQL are an extension of QBE-table. The form structures in VORQL extend QBE concept of tables in two important aspects. In QBE, each relation has its own QBE-table which contains the attributes of the relation whereas the various form structures in VORQL will join up all the objects and relationships specified in a query (i.e. a form contains multiple "relations"). This combination makes the JOIN operation, which has been repeatedly found to be a major problem in relational user–database interaction, implicitly depicted in the structure of the forms and frees the user from the tedious task of specifying the JOIN operation in the query. The second extension is that VORQL uses different form structures (i.e. Condition-form, Format-form, etc.) for different querying steps.

The form structure for specifying condition is chosen for this user-database interface because experimental results have shown that QBE "required about one-third the training time and appear to be about equally accurate as those using SEQUEL" (Thomas and Gould 1975). An experiment comparing QBE and SQL performed by

Greenblatt and Waxman (1978) also found that QBE involved less training time, required less time per query, resulted in more correct queries, and subjects were more confident in their answers. The use of form structure also has support from the Visual Language arena where Shu (1986) stated that forms are a natural interface between user, data, and program.

4. Relational completeness of VORQL

The expressive power of a language is an important criterion when designing user–database interface. The benchmark for measuring the expressive power of a relational query language is the relational completeness criterion. Codd (1972) first proposed relational tuple calculus as a benchmark for evaluating data manipulation languages based on the relational model. That is, a language without at least the expressive power of the safe formulae of relational calculus, or equivalently of relational algebra, was deemed inadequate. A language that can (at least) simulate safe tuple calculus, or equivalently, relational algebra or safe domain calculus, is said to be complete (Ullman, 1988).

The notion of completeness was also extended to Entity–Relationship (ER) based query languages by Atzeni and Chen (1983). They introduced two formal definitions of completeness: E–R completeness and simplified E–R completeness, the latter being a weak version of the former. However, their definition of completeness is not able to express queries involving comparisons between unrelated objects. Hence, ER complete and simplified ER complete languages are less powerful than relationally complete query languages.

In this paper, we adopt the Codd's (1972) criteria for completeness as the benchmark for measuring the expressive power of VORQL. Even researchers in the ER area uphold the superiority of relational completeness. For example, Campbell et al. (1985) concluded that "relational completeness for an ER query language is important; we can now design ER query languages that will be as powerful as traditional relational query languages".

Since relational algebra is relationally complete (Date, 1995), it follows that, to show that VORQL is also relationally complete, it is sufficient to show that VORQL includes analogs of each of the eight algebraic operators: Select, Project, Product, Union, Intersect, Difference, Join, and Divide. In fact, it is sufficient to show that VORQL includes analogs of the five primitive algebraic operations since Join, Intersection, and Divide can in turn be defined using the five primitives. The five primitive relational algebraic operations are union (U), difference (−), product (×), selection (σ), and projection (π). The proof is by induction on the number of operators.

To prove that VORQL is relationally complete, we need to have the relational representation of the OR model. The relational representation of the OR model is as follows:

An object O with key attribute k and other attributes $a_1,\ a_2,\ \ldots\ a_n$, is represented by the relation:

$O(k,\ a_1,\ a_2,\ \ldots,\ a_n)$

The name of this relation is the same as the name of the object.
A relationship R with attributes $a_1,\ a_2,\ \ldots,\ a_n$ and involving objects $O_1,\ O_2,\ \ldots,\ O_m$ with roles $role_1,\ role_2,\ \ldots,\ role_m$ respectively and keys $k_1,\ k_2,\ \ldots,\ k_m$ respectively, is represented by the relation:

$R2(O_1{-}role_1{-}k_1,\ O_2{-}role_2{-}k_2,\ \ldots,\ O_m{-}role_m{-}k_m,\ a_1,\ a_2,\ \ldots,\ a_n)$

The name of this relation is the same as the name of the relationship.
Lemma 1. *The base relation can be retrieved by VORQL.*
Proof. Consider an object O with key attribute k and attributes $a_1,\ a_2,\ \ldots,\ a_n$.
To retrieve O,

VORQL(O) =	Choose RETRIEVE icon, click object O.

Consider a relationship R involving objects O_1 to O_n with roles $role_1$ to $role_n$ and where the keys are k_i with i ranging from 1 to n. The attributes of the relationship are $a_1,\ a_2,\ \ldots,\ a_m$.
To retrieve R,

VORQL(R) =	Choose RETRIEVE icon, click relationship R. □

Lemma 2. *VORQL can create duplicate relations.*
Proof. Consider an object O with key attribute k and attributes $a_1,\ a_2,\ \ldots,\ a_n$. We can create another occurrence of object O using the DUPLICATE icon and name the new occurrence O'.

VORQL(O′) =	Choose DUPLICATE icon, click object O. Name the new occurrence O′.

Consider a relationship R involving objects O_1 to O_n with roles $role_1$ to $role_n$ and where the keys are k_i with i ranging from 1 to n. The attributes of the relationship are $a_1,\ a_2,\ \ldots,\ a_m$. We can similarly create another occurrence of relationship R using the DUPLICATE icon and name the new occurrence R'.

VORQL(R′) =	Choose DUPLICATE icon, click relationship R. □ Name the new occurrence R′.

Theorem 1. *VORQL satisfies the property of relational completeness.*
The proof of relational completeness is done as follows:

Step A.	Show that VORQL can produce relations derived by a single relational operation on the base relations.

Steps B and C.	Show that if VORQL can produce relations R_A and R_B, then VORQL can produce relations that are derived by a relational operation on R_A and/or R_B.
Step D.	Show that by induction, VORQL can produce any relation that can be derived by an arbitrary number of relational operations on the base relations.

Proof: Step A. Show that VORQL can produce relations derived by a single relational operation on the base relations.

A.1 Projection (denoted by [ ])

i. VORQL(O[x]) =	Choose RETRIEVE icon, click those attributes in set x on object O.
ii. VORQL(R[x]) =	Choose RETRIEVE icon, click those attributes in set x on relationship R.

A.2 Selection (denoted by σ_{sp} where *sp* is the selection predicate)

i. VORQL($\sigma_{sp}O$) =	Choose RETRIEVE icon, click O. Choose CONDITION icon, specify *sp* using the Condition-form.
ii. VORQL($\sigma_{sp}R$) =	Choose RETRIEVE icon, click R. Choose CONDITION icon, specify *sp* using the Condition-form.

A.3 Cartesian Product (denoted by $\times$)

i. VORQL($O_i \times O_j$) =	Choose RETRIEVE icon, click O_i and O_j.
ii. VORQL($R_i \times R_j$) =	Choose RETRIEVE icon, click R_i and R_j.
iii. VORQL($O \times R$) =	Choose RETRIEVE icon, click O and R.
iv. VORQL($R \times O$) =	Choose RETRIEVE icon, click R and O.

If $O_i = O_j$ or $R_i = R_j$, choose the DUPLICATE icon to duplicate the object or the relationship (see Lemma 2).

If O and R are directly linked on the OR schema, then duplicate either O to get O' or R to get R'. Following that, we can have either $O \times R'$ or $O' \times R$.

A.4 Union (for compatible objects/relationships)

i. VORQL(O_i U O_j) =	Choose RETRIEVE icon, click O_i. Choose UNION icon. Choose RETRIEVE icon, click O_j.
ii. VORQL(R_i U R_j) =	Choose RETRIEVE icon, click R_i.

Choose UNION icon.
Choose RETRIEVE icon, click R_j.

iii. VORQL(O U R) = Choose RETRIEVE icon, click O.
Choose UNION icon.
Choose RETRIEVE icon, click R.

iv. VORQL(R U O) =
Choose RETRIEVE icon, click R.
Choose UNION icon.
Choose RETRIEVE icon, click O.

If $O_i = O_j$ or $R_i = R_j$, choose the DUPLICATE icon to duplicate the object or the relationship (see Lemma 2).

A.5 Difference (denoted by $-$)

i. VORQL($O_i - O_j$) = Choose RETRIEVE icon, click O_i.
Choose CONDITION icon, specify O_i not exist in Q2.
Choose QUERY icon. (i.e. Q2)
Choose RETRIEVE icon, click O_j.

ii. VORQL($R_i - R_j$) = Choose RETRIEVE icon, click R_i.
Choose CONDITION icon, specify R_i not exist in Q2.
Choose QUERY icon. (i.e. Q2)
Choose RETRIEVE icon, click R_j.

iii. VORQL(O − R) = Choose RETRIEVE icon, click O.
Choose CONDITION icon, specify O not exist in Q2.
Choose QUERY icon. (i.e. Q2)
Choose RETRIEVE icon, click R.

iv. VORQL(R − O) = Choose RETRIEVE icon, click R.
Choose CONDITION icon, specify R not exist in Q2.
Choose QUERY icon. (i.e. Q2)
Choose RETRIEVE icon, click O.

If $O_i = O_j$ or $R_i = R_j$, choose the DUPLICATE icon to duplicate the object or the relationship (see Lemma 2).

Step B. In this step, we assume that two arbitrary relations R_A and R_B are derivable by VORQL queries. We then show that any relation resulting from a single relational operation on R_A and/or R_B is also derivable by a VORQL query. We assume that the VORQL query, Q_A, for R_A is of the following form:

Choose RETRIEVE icon, click a_{A1}, a_{A2}, a_{A3}, ... a_{Am}.
Choose CONDITION icon, specify condition(A) using the Condition-form.

Similarly, assume that the VORQL query, Q_B, for R_B is of the same form:
Choose RETRIEVE icon, click a_{B1}, a_{B2}, a_{B3}, ..., a_{Bn}.
Choose CONDITION icon, specify condition(B) using the Condition-form.

B.1 Projection:
VORQL(R_A[x]) = Choose RETRIEVE icon, click those attributes in set x from the list of attributes retrieved by Q_A (i.e. $\{a_{A1}, a_{A2}, a_{A3}, \ldots, a_{Am}\}$).

B.2 Selection:
VORQL($\sigma_{sp} R_A$) = Choose RETRIEVE icon, click Q_A.
Choose CONDITION icon, specify *sp* using the Condition-form.

B.3 Cartesian Product:
VORQL($R_A \times R_B$) = Choose RETRIEVE icon, click Q_A and Q_B.

B.4 Union:
VORQL(R_A U R_B) = Choose RETRIEVE icon, click Q_A.
Choose UNION icon.
Choose RETRIEVE icon, click Q_B.

B.5 Difference:
VORQL($R_A - R_B$) = Choose RETRIEVE icon, click Q_A.
Choose CONDITION icon, specify R_A not exist in Q2.
Choose QUERY icon. (i.e. Q2)
Choose RETRIEVE icon, click Q_B.

Step C. In step B, it is assumed that a relation R can be produced by a VORQL query Q of the form:
Choose RETRIEVE icon, click a_{A1}, a_{A2}, a_{A3}, ..., a_{Am}
Choose CONDITION icon, specify condition(A) using the Condition-form.
However, some relations can be produced only by a query of the form $Q_1UQ_2U\ldots UQ_n$ where each Q_i, $1 <= i <= n$, has the form assumed for Q.
It will be shown in this step that this violation of the

assumption in step B does not alter the result in step B. Assume R_A and R_B to be produced by VORQL queries of the form:

$$Q_{A1} U Q_{A2} U \ldots U Q_{An}$$

and

$$Q_{B1} U Q_{B2} U \ldots U Q_{Bm}$$

Each query Q_{Ai} or Q_{Bj} will produce a relation R_{Ai} or R_{Bj}. In other words, R_A equals $(R_{A1} U R_{A2} U \ldots U R_{An})$ and similarly for R_B.

C.1 Projection: $R_A[x]$

$$R_A[x] = R_{A1}[x] U R_{A2}[x] U \ldots U R_{An}[x]$$

Each of the sub-relations can be expressed by a VORQL query, as shown in step *B*.1, and by step *B*.4 the unions of the sub-relations can be done by a VORQL expression.

C.2 Selection: σ_A

$$\sigma_{RA} = \sigma_{RA1} U \sigma_{RA2} U \ldots U \sigma_{RAn}$$

An argument similar to *C*.1 applies here, showing that σ_{RA} can be done by a VORQL query.

C.3 Cartesian Product: $R_A \times R_B$
This is equal to

$$\begin{aligned}
&R_{A1} \times R_{B1} U R_{A1} \times R_{B2} U \ldots U R_{A1} \times R_{Bm} \\
&U \\
&R_{A2} \times R_{B1} U R_{A2} \times R_{B2} U \ldots R_{A2} \times R_{Bm} \\
&U \\
&. \\
&. \\
&. \\
&U \\
&R_{An} \times R_{B1} U R_{An} \times R_{B2} U \ldots U R_{An} \times R_{Bm}
\end{aligned}$$

Similar argument applies here.

C.4 Union: $R_A U R_B$
This is equal to

$$R_{A1} U R_{A2} U \ldots U R_{An} U R_{B1} U R_{B2} U \ldots U R_{Bm}$$

Similar argument applies here.

C.5 Set Difference: $R_A - R_B$
This is equal to

$$
\begin{aligned}
&(((R_{A1}-R_{B1})-R_{B2})-\ldots-R_{Bm})\\
&U\\
&(((R_{A2}-R_{B1})-R_{B2})-\ldots-R_{Bm})\\
&U\\
&.\\
&.\\
&.\\
&U\\
&(((R_{An}-R_{B1})-R_{B2})-\ldots-R_{Bm})
\end{aligned}
$$

Similar argument can also be applied here, showing that the long chain of unions and set differences can be produced by a VORQL query.

Step D. Show that by induction, VORQL can produce any relation that can be derived by an arbitrary number of relational operations on the base relations.

It has been shown that it is possible to use a single VORQL query to produce any of the base relations, and that a single VORQL query can produce any relation that can be derived by a single relational operation on the base relations. It has also been shown that if two relations R_A and R_B can be produced by queries Q_A and Q_B, then any relation resulting from a single relational operation on R_A and/or R_B can be produced by a VORQL query.

It therefore follows by induction that the VORQL query can derive relations involving any number of relational operations on the base relations. Hence, VORQL is relationally complete and hence has at least the same expressive power as relational query languages like SQL and QBE.

5. Conclusions and future research

User–database interaction is a critical component of database research. Better interface increases the productivity of and empowers the end users. Although database has been the subject of much research, the major part of the research focuses on designing and development of new database architectures. No doubt these are important areas of research, the adoption of database systems by end users and the satisfaction of end users depend more on the user interface than the underlying database architectures that the users do not come into direct contact. Despite the pivotal role of user–database interface, research in this area is severely lacking.

In this paper, we present the Visual Object–Relationship Query Language (VORQL) for the object–relationship (OR) model. The design of the interface employs icons, menus, and visual form-based structures to facilitate database retrieval by novice end users. The query language, VORQL, is a complete non-procedural visual query language with a syntax that is more flexible than SQL. It provides both physical and logical data independence which free the users from worrying about the physical and logical organization of the database. As it is traditional to demonstrate the expressive power of new query languages for user–database interaction, a proof is also presented in the paper to show that VORQL is relationally complete and is, therefore, at least as powerful as traditional database languages like SQL and QBE.

References

Atzeni, P. Chen, P.P., 1983. Completeness of query languages for the entity–relationship model. In: Chen, P.P. (Ed.), Entity–Relationship Approach to Information Modeling and Analysis. ER Institute, pp. 109–121.

Campbell, D.M., Embley, D.W., Czejdo, B., 1985. A relationally complete query language for an entity–relationship model. Proceedings of the Fourth International Conference on Entity–Relationship Approach, Chicago, Illinois, USA, pp. 90–97.

Card, S.K., Moran, T.P., Newell, A., 1983. The Psychology of Human–Computer Interaction. Lawrence Erlbaum Associates.

Chen, P.P., 1976. The entity–relationship model: toward a unified view of data. ACM Transactions on Database Systems 1(1), pp. 9–36.

Connolly, T., Begg, C., Strachan, A., 1996. Database Systems: A Practical Approach to Design, Implementation and Management. Addison–Wesley Wokingham, UK.

Codd, E.F., 1972. Relational completeness of database sublanguages. In: Randell R. (Ed.), Data Base Systems. Prentice Hall. Englewood Cliffs, NJ.

Date, C.J., 1995. An Introduction to Database Systems, 6th ed. Addison–Wesley. Wokingham, UK.

Elmasri, R., Navathe, S.B., 1994. Fundamentals of Database Systems, 2nd ed. Addison–Wesley. Wokingham, UK.

Embley, D.W., Kurtz, B.D., Woodfield, S.N., 1992. Object-Oriented Systems Analysis: A Model-Driven Approach. Yourdon Press.

Greenblatt, D., Waxman, J., 1978. A study of three database query languages. In: Shneiderman, B. (Ed.), Databases: Improving Usability and Representativeness. Academic Press, New York.

Halpin, T., 1995. Conceptual Schema & Relational Database Design, 2nd ed. Prentice Hall. Englewood Cliffs, NJ.

Hansen, G.W., Hansen, J.V., 1996. Database Management and Design, 2nd ed. Prentice Hall. Englewood Cliffs, NJ.

Kroenke, D.M., 1995. Database Processing: Fundamentals, Design, and Implementation, 5th ed. Prentice Hall. Englewood Cliffs, NJ.

Larkin, J.H., Simon, H.A., 1987. Why a diagram is (sometimes) worth ten thousand words. Cognitive Science 11, pp. 65–99.

Martin, J., Odell, J.J., 1992. Object-Oriented Analysis and Design. Prentice Hall. Englewood Cliffs, NJ.

Miller, G.A., 1956. The magical number seven plus or minus two: some limits on our capacity for processing information. Psychological Review 63, 81–97.

Moran, T.P., 1981. The command language grammar: a representation for the user interface of interactive computer systems. International Journal of Man–Machine Studies 15, 3–50.

Norman, D.A., 1986. Cognitive engineering. In: Norman, D.A., Draper, S. (Eds.), User-Centred Systems Design: New Perspectives on Human–Computer Interaction.

Price Waterhouse, 1977. Technology Forecast 1997. Price Waterhouse Technology Center.
Ramakrishnan, R., 1999. Database Management Systems. McGraw–Hill. New York.
Siau, K., Nah, F., 1997. An experimental study on user interpretation of icons. In: Salvendy, G., Smith, M.J., Koubek, R.J. (Eds.), Design of Computing Systems: Cognitive Considerations, Elsevier, pp. 721–724.
Shu, N.C., 1986. Visual programming languages: a perspective and a dimensional analysis. In: Chang, S.K. Ichikawa, T., Ligomenides, P.A. (Eds.), Visual Languages. Plenum Press.
Thomas, J., Gould, J., 1975. A psychological study of query by example. In: Proceedings of the 1975 National Computer Conference, Anaheim, CA. June 1975, pp. 439–445.
Tsichritzis, D., Klug, A., (Eds.), 1978. The ANSI/SPARC DBMS Framework. AFIPS Press.
Ullman, J.D., 1988. Principles of Database and Knowledge-base Systems, vol. I. Computer-Science Press.
Yourdon, E., 1994. Object-Oriented Systems Design: An Integrated Approach. Yourdon Press.