QBD*: a Graphical Query Language with Recursion[1]

IEEE Transactions on Software Engineering, *Vol.16*, No 10, pp. 1150-1163, 1990

Michele Angelaccio°, Tiziana Catarci•, Giuseppe Santucci•

° Dipartimento di Ingegneria Elettronica
Universita' degli Studi di Roma II "Tor Vergata"
via O. Raimondo 38 - 00173 Roma, Italy

• Dipartimento di Informatica e Sistemistica
Universita' degli Studi di Roma "La Sapienza"
Via Buonarroti 12 - 00185 Roma, Italy

ABSTRACT

One of the main problems in the database area is to define query languages characterized by both high expressive power and ease of use. In this paper, we propose a system to query databases, using diagrams as a standard user interface. The system, called Query by Diagram (QBD*), makes use of a conceptual data model, a query language on this model and a graphical user interface. The conceptual model is the Entity-Relationship Model; the query language, whose expressive power allows recursive queries, supports visual interaction. The main characteristics of the interface are the ease of use, and the availability of a rich set of primitives for schema selection and query formulation.*
Furthermore, we compare the expressive power of QBD and G+, which are the only languages allowing recursive queries to be expressed graphically.*

Index Terms: Database, Query Language, Visual Language, Expressive Power, Query Language Completeness

1. Introduction

[1]Research partially supported by Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo, National Research Council (CNR) Italy, and by Software AG Italia S.p.A.

One of the main developments in the database area concerns tools that provide a non expert user with a simple understanding of the database content, and a friendly extraction of information [1, 19].
The inadequacies of traditional database query languages are often a limit to the utilization of databases by non expert users. Fourth generation languages (see [16, 38, 26, 25, 32]), although non procedural, do not seem friendly enough for the casual user; in the interaction, the user must be able to model the information content of the database, and to employ a textual language. Moreover, the user's visual ability while interacting with the database is limited and the objects of interest are rarely displayed directly: rather, they are represented by formatted text, thereby not giving the user any iconic or spatial clues to help the querying process.

The new generation of Query Languages (QL) attempts to incrementally use a person's instincts and senses. This paper proposes a system, called *Query By Diagram** (*QBD**), whose goal is to achieve user friendliness for a large amount of user types, by means of a uniform graphical interface and a visual language.

Query by Diagram is based on four basic ideas: 1) to represent the intensional part of the database by means of a conceptual model (Entity-Relationship (E-R) model [8], augmented by the introduction of generalization abstractions [35]); 2) to employ a fully graphical environment as user friendly interface with the system; 3) to study the formal properties of the graphical language (e.g. completeness) by defining an isomorphic textual language; 4) to extend the graphical constructs of the language in order to easily express a significant class of recursive queries (e.g. transitive closure).

These choices allow several benefits:

a) The user is provided with a simple yet powerful formalism to understand the database schema. The concepts used in the formalism are very expressive and independent from the data representation in the logical model. Consequently, it is possible not only to query the database, but also to compare the intensional part of different databases.
b) The graphical formalism used to represent conceptual schemata simplifies the execution of typical activities involved in query formulation, as the extraction of the subschema of interest, and the expression of the query.
c) The introduction of a textual language isomorphic to the graphical one allows first comparing it with the relational algebra (obtaining the formal proof of relational completeness [9]) and moreover defining the available recursive queries.
d) The user is made able to express a set of recursive queries by means of a graphical friendly interaction with the system. Such graphical facilities are not generally provided for this kind

of queries (a notable exception is G+ [12, 13]); the systems described in [38, 34] allow to express them in textual form.

Various languages in the literature use diagrams for query formulation (see [18, 36, 37, 17, 14, 30, 15, 29, 21]). All such systems may perform first-order queries, i.e. have the same expressive power as that of the relational algebra. On the other hand, the extension of query languages to handle problems not solvable in relational algebra is an area of much current interest. Theoretical proposals and analyses have concentrated on the use of Horn clauses as a query and data definition formalism [35]. This approach seems to require a skill that typical users will not acquire easily. To get over these problems, some query interfaces have been proposed, which allow one to easily perform typical non first-order queries. Three notable examples are : G+, QBD*, and Pasta-3. It is worth noting that none of these languages allow the user to perform all possible computable queries ([2, 7]), but they are restricted to various classes of reasonable (in terms of practical use) recursive queries. In particular, G^+ queries, that are graph patterns, have been proved to contain also non-datalog queries [23]. In G^+ the database is represented by a labelled, direct multigraph. Queries are formulated in terms of graphs, such that simple paths (specified by means of regular expressions) suffice to express queries which would otherwise have to be defined recursively. Answering the query will require searching the database graph to find all subgraphs that are "similar" to the pattern.
The interaction of the user with the system is reduced to graphically build the query graph of interest and to write the regular expressions. The interaction is not fully friendly since regular expressions are not simple to manage by a novice user.
The expressive power of G^+ is shared by QBD^*, with a difference in the visual approach. In fact, QBD^* is mainly a navigational language on E-R diagrams. The operations of the relational algebra may be expressed directly by picking up symbols (entities or relationships) in the diagram and posing conditions on their attributes by means of a simple windowing mechanism. Recursive queries are also expressed by the same mechanism, the difference being the pre-selection of a particular icon, that signals the beginning of a recursive session. In this way, the textual interaction is completely eliminated, and the complexity of some query is transparent to the user.
Such friendliness seems not to be reached by Pasta-3 [22]. In fact, this language allows to graphically express a simple recursion (on a single entity and without equality or inequality conditions) using the duplication of the entity, linked to the same relationship in two different roles. However, complex recursive queries may be expressed either including Prolog rules (depicted in a separate window) or simulating with textual icons the quantifiers (existential and universal). The system is easy to use in order to express simple queries, where complex queries need tricks to be formulated, such as duplicate entities or use icons representing existential and universal quantifiers. Then, the claimed idea of a completely visual system seems to be reduced at the idea of simulating texts with boxes containing words.

Moreover, the system shown in this paper differs from the previous ones in providing the user with a rich set of strategies and types of interactions available in all phases of query formulation. In fact, the system can be used by the novice user not only to perform simple queries by navigating in a diagram, but also to acquire information about the database schema. Furthermore, the expert user is provided with a large set of facilities in order to perform complex queries and have detailed knowledge about the available data.

The paper is organized as follows. In Section 2 the general architecture of the system is presented. Section 3 gives an informal description of the graphical interface. In Section 4 we define the syntax and the semantics of the query language in terms of relational algebra expressions, and we prove its relational completeness ([9]). Finally, in Section 5, the operator CLOSURE-OF is introduced, that enables us to compute recursive queries. Furthermore, we compare the expressive power of QBD^* and G^+, which are the only languages allowing recursive queries to be expressed graphically.

2. System architecture

In this section we present a general overview of the system, whose architecture is shown in fig.1. We assume that the DBMS is based on the relational model; we will discuss in section 4 the mapping between the Entity Relationship model and the Relational model.
The general architecture of the system is based on three main modules:

1. The Graphical Interface enables the user to query the database. The query is expressed on an E-R schema by means of the graphical commands of the interface. Moreover, the user may access four libraries, containing suitable information, namely:
   a) *the E-R schema library* , containing all the E-R schemata of the applications;
   b) *the E-R schema user library*, containing user views of the schemata stored in the E-R schema library;
   c) *the E-R top-down schema library*, containing, for each schema in the E-R schema library, a set of schemata at higher abstraction level, representing the top-down development of the schema design.
   d) *the E-R user query library*, in which the user stores the graphical queries in order to reuse them afterwards.

   The result of the querying process is a query on the E-R schema. Corresponding to such a query there is also an echo formally expressed in terms of a textual language, whose syntactic constructs are isomorphic to the graphic operations performed by the user.
2. The Translator analyzes the queries and, using the target (relational) schema, translates the non-recursive queries into relational algebra expressions, and the recursive ones into suitable programs (see section 5.1).

3. The DBMS Interface translates the algebraic expression into a query formulated in terms of the target DBMS. In the case of a DBMS not allowing looping instructions, the system calls for suitable host language programs in order to express the recursive queries.

The querying process can be described as follows:

1) the user, by means of the graphical interface, builds the query on the E-R schema. The idea underlying this phase is to define a precise correspondence between the graphical commands performed by the user and the textual query language, formally described in section 4. As an informal example of such a correspondence, the graphical selection of the path entity Person, relationship Lives, entity City, is translated in the following textual query :

SELE Person FIND City TROUGH Lives ENDSELE;

2) the E-R query, produced in the former phase, is translated into a relational expression; referring to the above example, the translation of the E-R query is:

$$\Pi_{Attr(Person)}\ \sigma_{C}(Person \wedge Lives \wedge City)$$

where Att(E) is a function returning the attribute set of E, and C is a condition on the equality on the keys of the three relational tables (see Section 4 for more details about C and the mapping between the E-R schema and the relational schema);

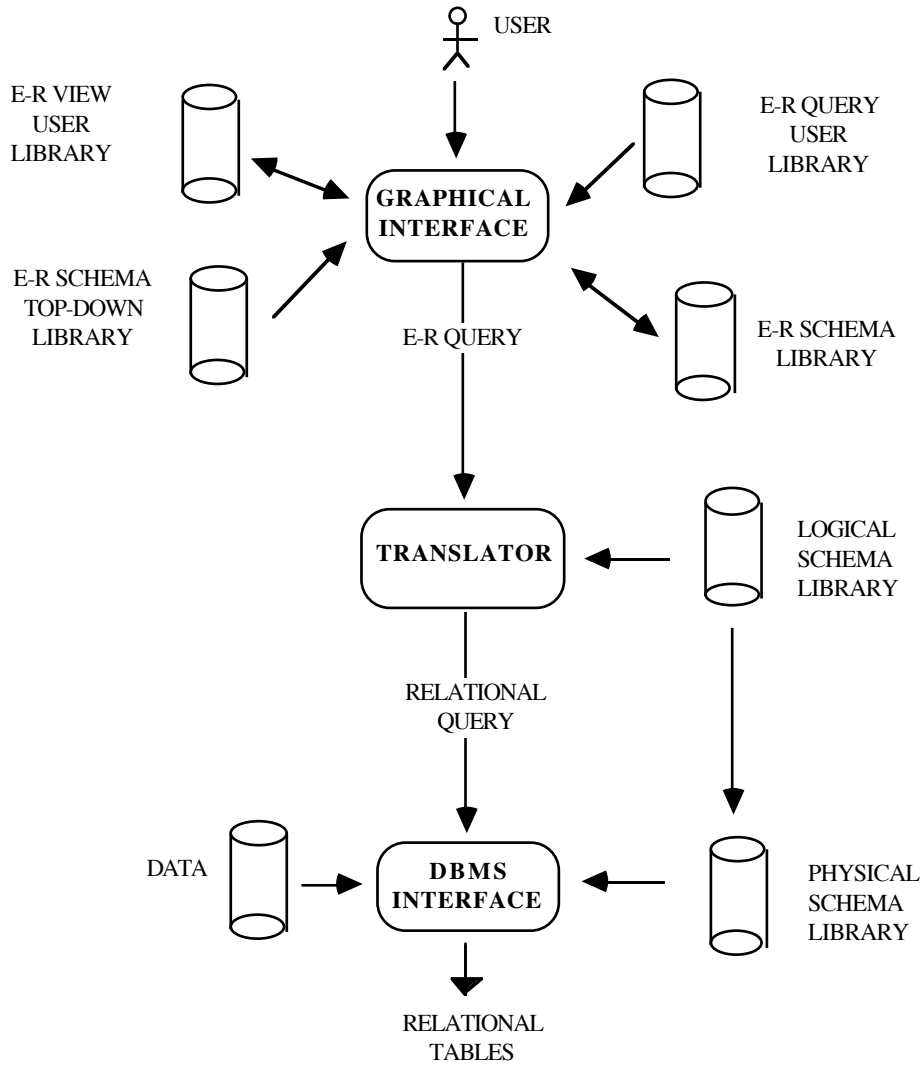


Fig.1: Architecture of the query environment

3) the DBMS interface formulates the relational query in terms of the target DBMS. The final result, expressed in SQL is :

SELECT Attr(PERSON) FROM Person,Lives,City WHERE C.

The interface not only supports expressing queries graphically, but also provides the user with a set of methodological strategies for simplifying his interaction with the system. Such an idea stems from the consideration that the user needs not only a query language. In fact, several difficulties may arise in dealing with previously unknown database schemata, such as: discover implicit information, analyze too much detailed schemata, need of restoring pre-acquired information, etc. Then, during the query formulation activity, a large set of graphical primitives is provided for helping the user in understanding the database schema and friendly extracting the required information. In principle, the user querying process can be divided in four different phases:

1) First the user may interact with the conceptual schema by means of the *top-down browsing* mechanism. The system provides for each schema both a library of top-down refinements, documenting the schema at various levels of detail, and a top-down browsing mechanism. Top-down browsing may be used for selecting either specific objects or complete schemata, and navigating up and down among the different top-down levels of the schema in order to better locate the interesting concepts and the links among them.
2) By using a set of graphical primitives (called *location primitives*), the user may extract the subschema containing the concepts involved in the query (*schema of interest*); for this purpose, she/he may adopt three different strategies: 1) *Direct extraction* , that is the most immediate method of extracting the schema of interest, and consists in picking up the interesting objects; 2) *Expression of meta-queries on the schema,* this strategy corresponds to querying the schema structure, and extracting from it meaningful information. For this purpose the user may select the concepts (entities or relationships) putting some conditions on their attributes or asking the system for all the existing paths connecting two concepts (specifying conditions on the length of the path and/or on the presence of particular concepts). Such paths are displayed to the user in order to be included in the schema of interest. Moreover, once a symbol, or a group of connected symbols, has been selected, it is possible to enlarge the sub-schema of interest by an "oil-stain" expansion; 3) *Use of a library of schemata* , the system is able to manage libraries of tailored schemata: schemata (or selected parts) can be extracted and integrated with the current schema of interest, using a "cut-and-paste" strategy. Analogously, the user may add new schemata to the library.
3) Once the schema of interest has been selected, the user may perform transformations on such a schema, by means of other primitives (called *replacing primitives*), bringing it "close to the query"; in this way, a user (temporary or permanent) view of a subset of the schema is built, that may be helpful in the subsequent activities of query formulation. The main difference between replacing and location primitives is that the former allow modifying the schema, possibly with loss of information content. In other words, the user may build a proper view of the schema, which is not isomorphic to any schema of interest resulting from applying location primitives. The essential characteristic of replacing primitives is the fact that the database schema of interest is obtained by replacing a portion of the schema by a single concept (entity or relationship). Such primitives are either *monadic* (only one concept of the schema is replaced), or *polyadic* (a portion of the schema is replaced). The monadic primitives allow adding or deleting an attribute and replacing one schema concept with another. The polyadic primitives replace (with or without loss of information content) one central concept linked to a generic number of other neighbour concepts, with a unique concept of the same kind of the neighbour concepts.
4) Finally, the user may resort to the query language in order to complete the specification of the query (we call this phase *navigation phase*, because of the navigation language). The graphical operation involved in the navigation phase are mainly navigation or selection on the

database schema (plus some icon selection); even complex queries may be expressed by means of a sequence of such elementary graphical operations.

The previous activities, so far described as logically distinct, can be interactively intermixed. For example, if during a query formulation a concept is needed, which was not previously selected, it is possible to switch to the location phase, adding the symbol, and then going back to the navigation phase for resuming the query.
In synthesis, the activity of query formulation is composed of several phases (location, replacing, and navigation), each one supported by a large set of graphical primitives. It is worth noting that such primitives should be seen as a kit of tools, that will be chosen by the user according to his needs.
In the sequel, we focus on the navigation phase (namely, on the query language), while location and replacing primitives are described in detail in [3, 4].

3. The Graphical Query Language

In this section we review the graphical query language *QBD**. The language is a visual navigation language on E-R schemata, represented by means of diagrams; the basic idea is to decompose a query into its elementary steps, expressed by a limited set of graphical operations, such as choice of icons, selection of concepts, and navigation. In order to express formally the syntax and the semantics of the language, a one-to-one correspondence between the graphical operations and the syntactic constructs of a textual query language has been defined.
The general structure of the query is based on the location of a distinguished concept, called *main concept* (an entity or a relationship), that can be seen as the entry point of one or more subqueries; these subqueries express possible navigations from the main concept to other concepts in the schema. The subqueries can be combined by means of the usual union and intersection operators.
Once the main concept has been selected, two different types of primitives are available for navigating in the schema. The first one allows the user to follow paths of concepts, the second one is used for comparing two concepts which are not directly connected to each other. Such two primitives can be arbitrarily nested.
If the previous constructs are applied to generalization hierarchies the following rules hold: if the query involves a child entity, this entity inherits all logical relationships and attributes coming from the parent entity, in addition to its own links. On the other hand, if the involved entity is the parent entity, it does not inherit the properties of the child entities.
In the following, we give the informal semantics of each graphical primitive, while the formal syntax and semantics of the graphical language will be shown, in section 4 and 5, referring to the constructs of the isomorphic textual language.

3.1 Graphical Query Primitives

The graphical query language is used in the navigation phase of the query formulation; in such a phase the user may interact with the system in the following way:

- ●) Selecting the icon corresponding to "navigation".
- ●) Choosing the main concept of the query, by picking it up with the mouse.
- ●) Specifying possible conditions on the attributes of the main entity (the same mechanism is used for comparing attributes of different entities). The list of attributes is shown in a separate window, containing also the elements involved in the comparison (i.e. constants, other attributes, etc.), and a set of icons suitable to formulate conditions on the attributes. Conditions are expressed selecting the attributes and the icon corresponding to the required operator. The system shows the result of the operation by displaying a graph, where vertices correspond to attributes, and labelled edges correspond to the operators. In fig. 3 an example of condition is shown.
- ●) Locating a path of concepts starting from the main entity. For this purpose two kinds of graphical primitives are available: 1) The first one allows joining chains of entities and relationships directly linked to each other in the schema, and expressing possible conditions on the attributes. This operation is carried out by picking up sequentially the concepts in the path. 2) The second one allows joining entities not explicitly linked in the schema. The join is performed comparing the attributes of the two entities using the window described above.

The system manages differently the attributes belonging to the main entity with respect to the others. More precisely, the attributes of the main entity are automatically considered in the final result of the query (unless explicitly deleted by the user), while the other ones are shown in the result only if requested. The reason for that is twofold: first an unsuitable growth of the result is avoided; second, since we consider the main entity as the "heart" of the query, its attributes are given higher importance with respect to the others.

Example: We suppose the user interested in the following query: "Find students taking part in a journey to Boston, which did not book any means of transport". Referring to fig. 2, representing the schema of interest concerning this query, that we suppose to be extracted from a wider database schema in the previous location and replacing phases, we select the entity TRAVEL, and we put conditions on a subset of its attributes. In particular, we want to extract all the travels whose destination (TO_CITY) is 'Boston'.

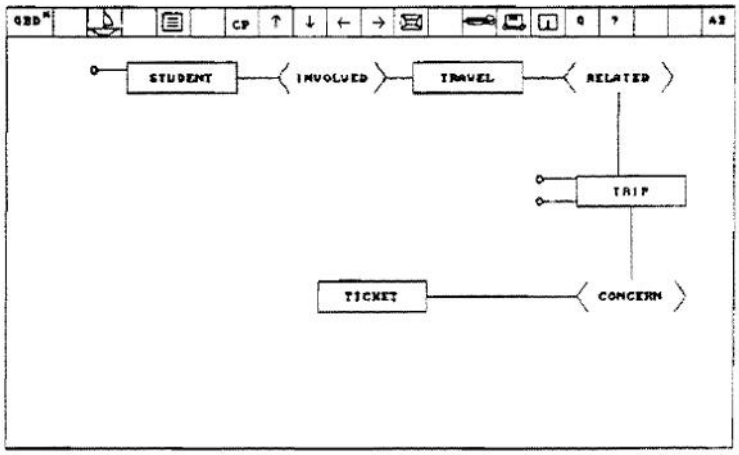


Fig.2: Schema of interest

On the left side of the window the attribute set of TRAVEL is displayed, and on the right the set of attributes of a "dummy" entity, built by the user. In order to characterize the selection, the user may specify either constants or names of attributes (in this example the constant is 'Boston'). The condition specified by the user is displayed by means of a labelled edge shown in fig. 3.

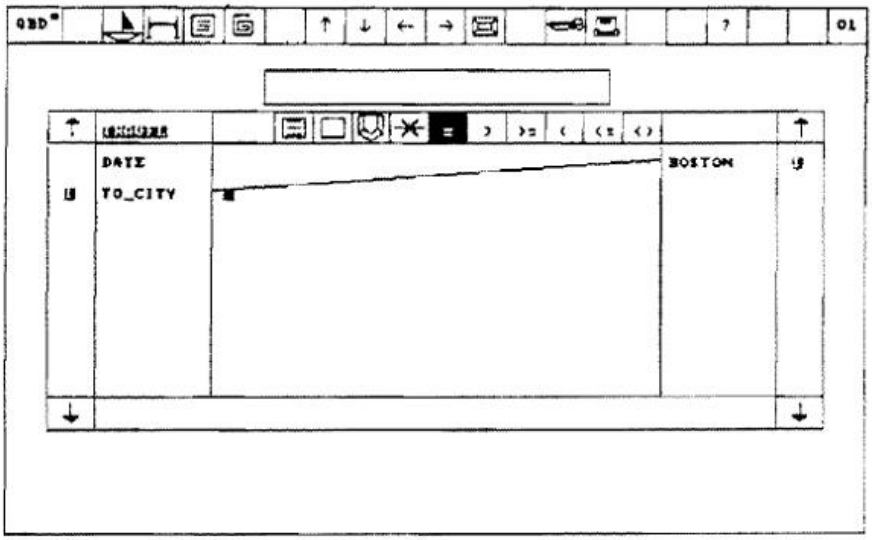


Fig. 3: Condition definition

It is worth noting that QBD* allows performing non-first order queries by means of a simple graphical mechanism, which is the same as the one allowing for joining entities. In fact, an operator computing the generalized transitive closure (GTC) [31] is available (the GTC is a transitive closure of an entity, in which the cycle conditions are extended to be boolean expressions with equality and inequality operators). Such cycle conditions are equivalent to particular joining conditions where an entity is compared with itself (see section 5.1), therefore the window used for the GTC looks like the one adopted in the joining graphical operator, the difference being the pre-selection of a particular icon, that signals the beginning of a recursive session. In such a session the user may:

●) select the desired entity;

●) specify the cycle conditions on the attributes of the entity. In the example, the cycle condition is the equality of the attributes FROM_CITY and TO_CITY (see fig. 4);
●) specify a set of conditions in order to select a subset of the result of the transitive closure.

Example: Figure 5 shows another example of the graphical solution proposed in order to compute the transitive closure. The schema contains information about flights and is based on the example presented in [2]. We wish to compute finite sequences of flights such that in each sequence:
1) the destination of each flight (except the last) is the source of the next, and
2) the arrival time of each flight (except the last) occurs before the departure time of the next.

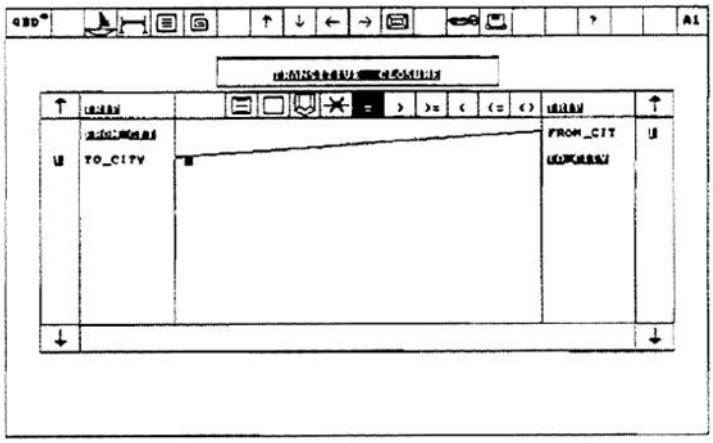


Fig. 4: Example of Transitive Closure

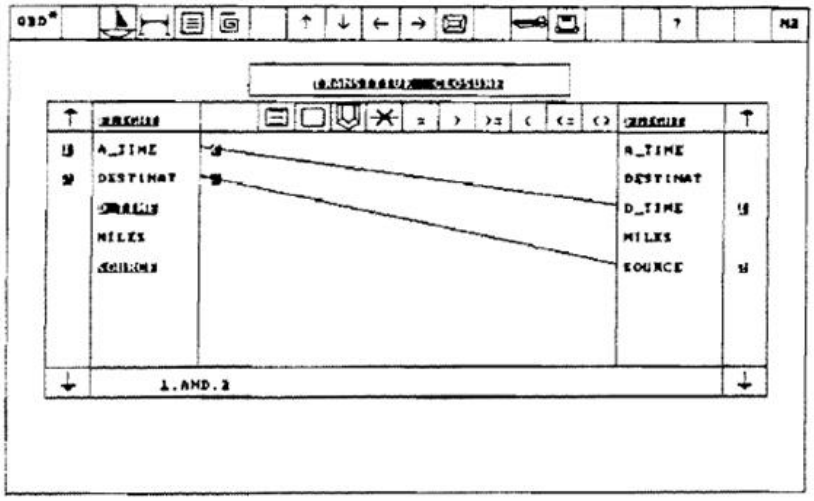


Fig. 5: Another example of Transitive Closure

The main idea is to visualize the generic step of recursion (corresponding to a cartesian product) displaying a double list of the same attributes; the user may specify the cycle conditions comparing the attributes with each other. Then, the system determines the attributes involved in the projection (shown in reverse in fig.5). In the example, the user selects the chain of flights such that DESTINATION = SOURCE AND ARRIVAL_TIME < DEPARTURE_TIME. The attributes involved in the projection are DEPARTURE_TIME and SOURCE of the previous FLIGHT, and ARRIVAL_TIME and DESTINATION of the next FLIGHT.

4. Syntax and Semantics of QBD

In this section we deal with the formal definition of the syntax and semantics of a subset of QBD*. This subset, called *QBD*, is formed by all the navigational operators of QBD*, except the transitive closure operator.
The aim of this section is twofold: first, we give the description of QBD, then we show that it is relationally complete.
We define the semantics of QBD queries by induction on the length of the connection paths in the E-R diagram, according to the graphical interpretation of the commands. In this way we are able to associate a relational algebra expression with each query, via a mapping of E-R diagrams into Relational schemata.

4.1 The Syntax

We give the syntax of the textual language corresponding to QBD by means of the usual BNF. The following metasymbols, with corresponding interpretations, are used:
[]: optional
/: alternative
{..}+: sequence of {..}

<QBD>::= <label_query> {<set_operator> <label_query>}+
<label_query>::=<query> [MEM INTO <query_name>]
<set_operator>::= MINUS / UNION / INTERSECT
<query>::=SEL<ent_rel_name>[<navigation>+]
ENDSEL[WITH<condition>+] [ADD <attr_list>] [DEL <not_key_attr_list>]
<navigation>::=<nav1> / <nav2>

<nav1>::=FIND <ent_rel_name> [THROUGH <rel_name>] [WITH <condition>$^{+}$] [ADD <attr_list>]

<nav2>::=BRIDGE <query> BUILT WITH <bridge_condition>$^{+}$ [ADD <attr_list>]

<bridge_condition>::={<bridge_exp> [<boolop> <bridge_exp>]$^{+}$} / TRUE

<attr_list>::= <attr>$^{+}$

<not_key_attr_list>::= <not_key_attr>$^{+}$

<condition>::=<attr_exp> <boolop> <attr_exp>

<attr_exp>::= <attr> <op> <attr> / <const>

<bridge_exp>::=<attr> <op> <pref_attr>

<pref_attr>::={[<pref>] <attr>}

<pref>::={BR.<alphanum> <ent_rel_name>[.<ent_rel_name>]$^{+}$

<attr>::= <attr_name> / <new_attr_name>

<new_attr_name>::= (REN OF <attr_name> OF <ent_rel_name>)

<op>::= < / > / = / <> / <= / >=

<boolop>::=AND / OR

<parameter>::= <integer>$^{+}$

<attr_name>::=<term>

<not_key_attr>::=<term>

<ent_rel_name>::= <ent_name> / <rel_name> / <query_name>

<ent_name>::=<term>

<rel_name>::=<term>

<query_name> ::=<term>

<term>::=<letter> <alphanum>

The graphical operations described in Section 3.1 correspond to the constructs of the textual language in the following way:

1) The selection of the main entity of the query (see fig. 6.a, in which the following conventions are adopted: concepts selected in the E-R diagram are denoted by dot-filled boxes; conditions expressed on the concept **X** are displayed in the first row of a window containing the list of attributes (attr(**X**))) corresponds to the construct:

SEL $\mathbf{X}_1$ ENDSEL WITH C,DEL D

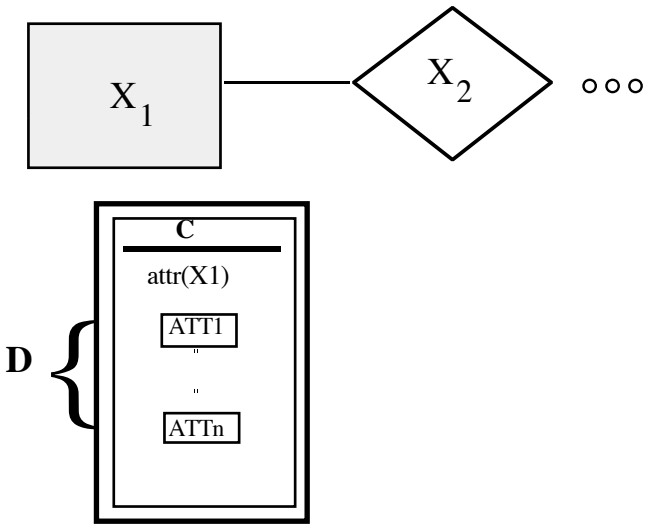


Fig.6.a: Selection of the main entity of the query

2) The location of two concepts directly linked to each other in the schema (see fig.6.b) corresponds to the construct:

SEL **X**$_1$ FIND **X**$_2$ WITH C2 ENDSEL WITH C1,DEL D ADD A

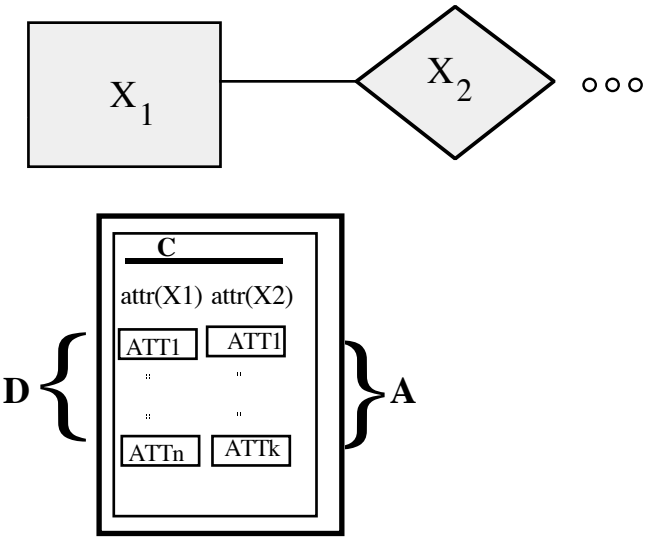


Fig.6.b: Selection of an existing path of two concepts

3) The sequential picking-up of entities not explicitly linked in the schema (see fig.6.c) corresponds to the construct:

SEL **X**$_1$ BRIDGE SEL **X**$_2$ ENDSEL BUILTWITH C2 ENDSEL WITH C1,DEL DADD A

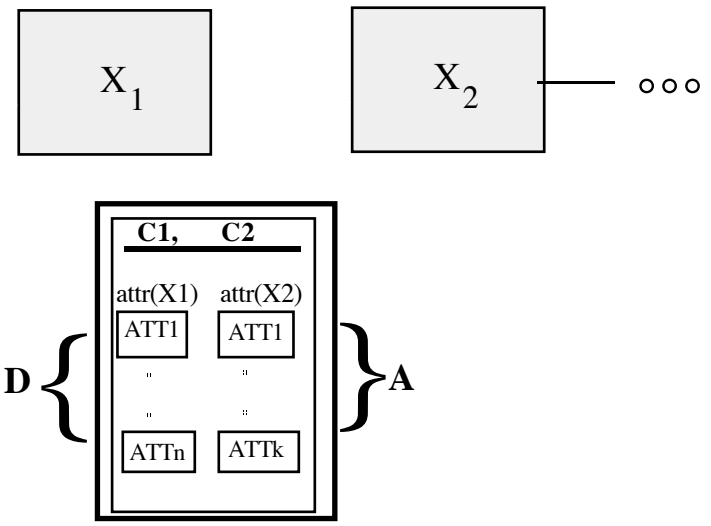


Fig.6.c: Sequential selection of two unconnected entities

4) The selection of a path of any length (see fig.6.d) is expressed as:

SEL $\mathbf{X}_1$ FIND $\mathbf{X}_3$ THROUGH $\mathbf{X}_2$ WITH $COND_{2,3}$ FIND $\mathbf{X}_5$ THROUGH $\mathbf{X}_4$ WITH $COND_{4,5}$ .. FIND $\mathbf{X}_n$ THROUGH $\mathbf{X}_{n-1}$ WITH $COND_{n-1,n}$ ENDSEL DEL D ADD A

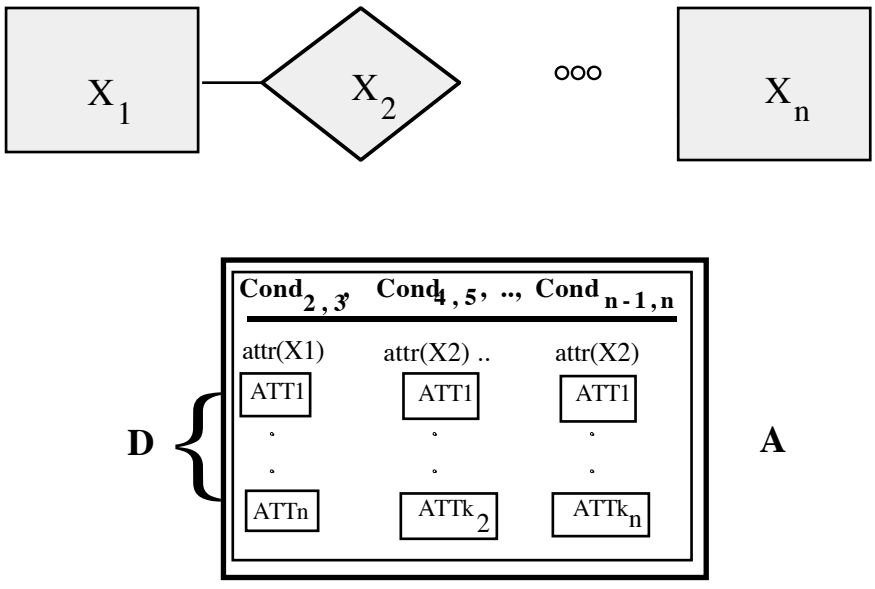


Fig.6.d: Selection of a path

4.2 The Semantics

Before defining the meaning of QBD queries, we recall the formal definition of the E-R Model and the Relational Model, and we define a mapping between them. In the following we use

several metasymbols whose meaning is as follows: $P,P_1,P_2 \in$ QBD, P_{navf} is of type <nav1>, P_{nav} is of type <navigation>, C is either of type <condition>, ADD<attr_list>, or DEL <not_key_attr_list>, **E** is an entity name, **R** is a relationship name, **X** is either an entity name or a relationship name, A, B are sets of attributes.

Let $< R_1,..,R_h >$ be a relational database schema, where each R_i is a relation schema (finite set of attributes). We write $key(R_i)$ for denoting the set of the keys of R_i. Let $B = <r_1,..,r_h>$ be a relational database (also called state) for the schema $<(R_1,..,R_h)>$ (i.e. one relational table r_i, for each R_i, with r_i satisfying the constraints expressed by $key(R_i)$). We denote by r[A] the projection of tuples of the relation r on the set of attributes A. If a is an attribute shared by two relations r and s, a.r (a.s) denotes the attribute a in the relational schema of r (of s). Let $ALG(OP)$ be the set of expressions constructed by using the set of operators OP . For instance **ALG**(-, $\cup$, $\times$, Π_A, σ_F) is the set of expressions corresponding to the relational algebra (see [35]).

Note that it is possible to consider either the state-of-mappings representation or the set-of-list representation as shown in [35]. The former will be used throughout the paper.

In the following we present the basic data modeling concepts of the E-R model. An entity set **E** is a collection of entities. We call entity-subset every entity-set **S** (son) whose set of entities is a subset of those belonging to another entity-set **F** (father). The latter is called the entity-superset of **S** in the superset/subset relationship.

A relationship among entity sets is an ordered list of entity-sets. If **R** is a relationship among entity-sets $\mathbf{E}_1$.. $\mathbf{E}_k$, then each instance of **R** is a set of k-tuples taken from $\mathbf{E}_1 \times .. \times \mathbf{E}_k$; we call such a set a relationship-set (denoted with **r**); $attr(X)$ denotes the set of attributes of the concept **X** (through the paper we denote by the word *concept* both entity-set and relationship), and $id(X)$ denotes the identifier of **X** defined as follows:

$$id(\mathbf{E}) \subseteq attr(\mathbf{E})$$
$$id(\mathbf{R}) = \cup_i id(\mathbf{E}_i)$$

where **R** is a relationship among ($\mathbf{E}_1$,.. ,$\mathbf{E}_k$).

An E-R data model is characterized by the following definitions:

the schema (E-R Schema) sc(**B**) contains the entity-sets $\mathbf{E}_i$ and the relationships $\mathbf{R}_j$ with their attribute sets and identifier sets,

the database (E-R database) **B** (associated to an E-R schema sc(**B**)) is a sequence of entity-sets $\mathbf{E}_i$ and relationship-sets $\mathbf{r}_j$, corresponding to $\mathbf{R}_j$.

We define the mapping $\mathbb{T}$

$$\mathbb{T} : \mathbf{B} \rightarrow B$$

from E-R databases to relational databases, such that:

$$\mathbb{T}(\mathbf{E}) = \{e \mid e \in \mathbf{E}\}$$
$$\mathbb{T}(\mathbf{r}) = \{(e_1,.. ,e_k) \mid (e_1,.. ,e_k) \in \mathbf{r}\}$$

Associated to $\mathbb{T}$ we have the schema mapping $\mathbb{ST}$ such that:

for each entity **E:** $\mathbb{ST}$(**E**) =attr(**E**) with key($\mathbb{ST}$(**E**))=id(**E**)

for each relationship **R** among ($\mathbf{E}_1$,.. ,$\mathbf{E}_k$):

$\mathbb{ST}$(**R**)=attr(**R**) ∪ id($\mathbf{E}_1$) ∪.. ∪ id($\mathbf{E}_h$) with key($\mathbb{ST}$(**R**))=id(**R**);

for each entity-subset **S** of the entity-superset **F:**

$\mathbb{ST}$(**S**) =attr(**S**)∪id(**F**) with key($\mathbb{ST}$(**S**))=id(**F**)

We call $\mathbb{U}$ the inverse of mapping $\mathbb{T}$:

$$\mathbb{U} = \mathbb{T}^{-1}$$

In order to define the semantics of QBD queries, we now introduce a binary operator, called navigational operator, whose meaning is given in terms of the relational algebra.

Let r, s be two relations corresponding to relational schemata R and S; let A⊆ R∪S and F,G,J be boolean conditions involving attributes of R, S, and R∪S respectively. The *navigational operator* $\nu_{A,F,G,J}$ applied to r and s is defined as

$$\nu_{A,F,G,J}\,(r,s) = \Pi_A\,(\sigma_{F \wedge G \wedge J}\,(r \times s))$$

Each query in QBD can be interpreted as an application of the navigational operator.

For instance, the BRIDGE operator, performed by picking up sequentially two entities $\mathbf{E}_1$ and $\mathbf{E}_2$, may be described in terms of the following correspondences:

- A is the attribute set obtained by taking attr($\mathbf{E}_1$), deleting the attributes occurring in the body of the DEL operator, if any, and adding the attributes in attr($\mathbf{E}_2$) occurring in the body of the ADD operator;
- F is the condition on attr($\mathbf{E}_1$);
- G is the condition on attr($\mathbf{E}_2$);
- J is the condition expressed in the body of the BRIDGE operator (BUILTWITH).

In other words, ν provides the algebraic meaning of one step of navigation: it will be used in the following to obtain a semantic description of QBD queries. We call *navigational expression* every expression belonging to **ALG**(-, ∪, ν).

Throughout the paper, the symbol "T" will denote the truth-value true.

Note that for J of the form $a_1.r = a_1.s$ *AND* .. *AND* $a_k.r = a_k.s$ (where $\{a_1..a_k\}$= R∩S), $\nu_{R,T,T,J}(r,s)$ is equivalent to the semijoin operator; in this case J will be written as Je.

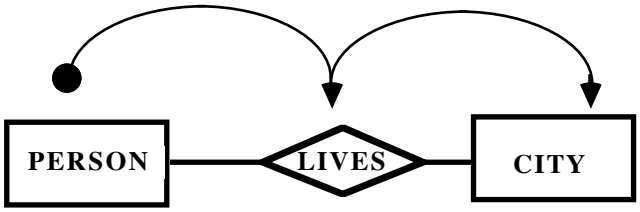


Fig.7 : An example of the navigational operator

For example, referring to the schema in fig.7, we show how the navigational operator is used for computing the attribute Living_city for the entity-set Person. In fig.7 we show the subschema that is useful for the computation. In order to derive the new attribute, the user selects the entity-set Person, the relationship Lives and the entity-set City, adding the attribute City_name. The semantics of such operations is described as a double application of the ν operator.
In particular, the first application, corresponding to the path Person-Lives, is:

$$\nu_{attr(Person),T,T,Je}\ (\mathbb{T}(Person),\mathbb{T}(Lives))$$

whereas the second application, corresponding to the path Lives-City, is:

$$\nu_{attr(Lives),T,T,Je}\ (\mathbb{T}(Lives),\mathbb{T}(City))$$

The composition of both application, is:

$$\nu_{attr(Person),T,T,Je}\ (\ \mathbb{T}(Person),\ \nu_{attr(Lives),T,T,Je}\ (\ \mathbb{T}(Lives),\mathbb{T}(City)\ )\ )$$

From a semantical point of view, every query $P \in QBD$ operates on an E-R database **B** returning a set of entities P(**B**). In order to characterize the semantics of P, we provide the definition of the relational expression **M**(P), such that $\mathbf{M}(P)(\mathbb{T}(\mathbf{B})) = \mathbb{T}(P(\mathbf{B}))$ (notice that $\mathbb{T}(P(\mathbf{B}))$ is simply the relational counterpart of the result of P). We define **M**(P) by induction on the number of concepts (entity-set or relationship) occurring in P.

Induction Basis

Let $\mathbf{X}_1,..,\mathbf{X}_n$ be all the concepts involved in P:
(n=1)

$$\mathbf{M}(\underline{SEL}\ \ \mathbf{X}_1\ \underline{ENDSEL}\ \underline{WITH}\ C,\underline{DEL}\ D) = \nu_{attr(X)\text{-}D,C,T,T}(\mathbb{T}\ \mathbf{X}_1,\mathbb{T}\ \mathbf{X}_1)$$

(n=2)

$$\mathbf{M}(\underline{SEL}\ \mathbf{X}_1\ \ \underline{FIND}\ \mathbf{X}_2\underline{WITH}\ C2\ \underline{ENDSEL}\ \underline{WITH}\ C1,\underline{DEL}\ D\ \underline{ADD}\ A) =$$
$$\nu_{attr(X_1)\text{-}D\cup A,C1,C2,Je}(\mathbb{T}\ \mathbf{X}_1,\mathbb{T}\mathbf{X}_2)$$

$$\mathbf{M}(\underline{SEL}\ \ \mathbf{X}_1\ \underline{BRIDGE\ SEL}\ \mathbf{X}_2\ \underline{ENDSEL}\ \underline{BUILTWITH}\ C2\ \underline{ENDSEL}\ \underline{WITH}\ C1,\underline{DEL}\ D\underline{ADD}\ A)=$$

$$\nu_{attr(X_1)\text{-}D\cup A,C1,T,C2}(\mathbb{T}\mathbf{X}_1,\mathbb{T}\,\mathbf{X}_2)$$

It is useful to denote the last two cases with:

$$\nu_{proj(X_1),cond(1,2)}(\mathbb{T}\mathbf{X}_1,\mathbb{T}\,\mathbf{X}_2)$$

where cond(1,2) is equivalent to the triple <C1,C2,Je> (FIND operator) or to the triple <C1,T,C2> (BRIDGE operator), and proj(X_1) is the projection (attr($\mathbf{X}_1$)-D)∪A.

(n>2)

$$\mathbf{M}(P)= \nu_{proj(X_1),cond(1,2)}(\mathbb{T}\mathbf{X}_1, \nu_{attr(X_2),cond(2,3)}(\mathbb{T}\mathbf{X}_2,..., \nu_{attr(X_{n-1}),cond(n-1,n)}(\mathbb{T}\mathbf{X}_{n-1},\mathbb{T}\mathbf{X}_n))..)$$

Finally, the semantics of the UNION and MINUS operators is defined in the obvious way:

$$\mathbf{M}(P_1 \text{ UNION } P_2) = \mathbf{M}(P_1) \cup \mathbf{M}(P_2)$$
$$\mathbf{M}(P_1 \text{ MINUS } P_2) = \mathbf{M}(P_1) - \mathbf{M}(P_2).$$

From the above definition of the QBD semantics it is easy to state the following theorem:

Theorem 1: All queries of QBD are definable in terms of navigational expressions, i.e.:

$$\mathbf{M}(QBD) \subseteq \mathbf{ALG}(-, \cup, \nu),$$

where

$$\mathbf{M}(QBD) = \{\mathbf{M}(P) \mid P \text{ is a query of QBD}\}.$$

˘

In the next section we make use of the above theorem in order to show that every query expressible in QBD is also expressible in the relational algebra. Furthermore, we will prove that the converse is also true, i.e. QBD is relationally complete.

4.3 Completeness proof

Let $\mathbf{Q}$(OP) [$\mathbf{Q}$(QBD)] be the class of queries (i.e. recursive functions defined on states of relational schemata) computed by the expressions in **ALG**(OP) [**M**(QBD)].

Let ≠ be the equivalence relation defined between expressions of relational algebra which compute the same query.

Lemma 1: The set of queries computable by relational algebra expressions coincides with the set of queries computable by navigational expressions, i.e.:

$$\mathbf{Q}(-, \cup, \times, \Pi_A, \sigma_F) = \mathbf{Q}(-, \cup, \nu)$$

Proof

$$\mathbf{Q}(-, \cup, \times, \Pi_A, \sigma_F) \subseteq \mathbf{Q}(-, \cup, \nu_C)$$

Let us consider r,s with relational schemata R,S; the following equivalences hold:

$$\sigma_F(R) \equiv \nu_{R,F,T,T}(r,r),$$

$$\Pi_A(R) \equiv \nu_{A,T,T,T}(r,r),$$

and

$$R \times S \equiv \nu_{R \cup S,T,T,T}(r,s).$$

$$\mathbf{Q}(-, \cup, \times, \Pi_A, \sigma_F) \supseteq \mathbf{Q}(-, \cup, \nu)$$

It follows immediately from the definition of ν.

˘

Theorem 2: Every navigational expression is definable in terms of a QBD query, i.e.:

$$\mathbf{ALG}(-, \cup, \nu) \subseteq \mathbf{M}(\text{QBD})$$

Proof. By induction on the definition of navigational expressions.

Induction Basis

$$r\text{-}s \equiv \mathbf{M}(\text{SEL } \mathbb{U}r \text{ ENDSEL MINUS SEL } \mathbb{U}s \text{ ENDSEL})$$

$$r \cup s \equiv \mathbf{M}(\text{SEL } \mathbb{U}r \text{ ENDSEL UNION SEL } \mathbb{U}s \text{ ENDSEL})$$

Now let us suppose that r , s have relational schema R,S such that

$$key(R) \subseteq S \text{ and } r[key(R)] = s[key(R)],$$

we have

$$\nu_{B,C1,C2,E(r,s)}(r,s) \equiv \mathbf{M}(\text{SEL } \mathbb{U}r \text{ FIND } \mathbb{U}s \text{ WITH C2 ENDSEL C1, ADD A, DEL D})$$

where A and D are such that B = R ∪A-D, otherwise we have

$$\nu_{B,C,T,J}(r,s) \equiv \mathbf{M}(\text{SEL } \mathbb{U}r \text{ BRIDGE SEL } \mathbb{U}s \text{ ENDSEL BUILT WITH J ENDSEL WITH C, ADD A, DEL D}).$$

Induction Step.

Let us consider P and Q satisfying the inductive hypothesis. We obtain:

$$\mathbf{M}(P) - \mathbf{M}(Q) \equiv \mathbf{M}(P \text{ MINUS } Q)$$

$$\mathbf{M}(P) \cup \mathbf{M}(Q) \equiv \mathbf{M}(P \text{ OR } Q)$$

$$\nu_{B,C1,C2,J}(\mathbf{M}(P), \mathbf{M}(Q)) \equiv$$

$$\mathbf{M}(P \text{ MEMINTO PL } Q \text{ MEMINTO QL SEL PL ENDSEL WITH C2 BRIDGE QL BUILT WITH J ENDSEL WITH C1, ADD A, DEL D})$$

˘

Corollary 1: The language QBD is relationally complete:

$$\mathbf{Q}(\text{QBD}) = \mathbf{Q}(-, \cup, \times, \Pi_A, \sigma_F)$$

Proof

From Theorems 1 and 2, it follows that:

$$\mathbf{Q}(\text{QBD}) = \mathbf{Q}(-, \cup, \nu),$$

Therefore, the thesis follows from Lemma 1.

5. Recursive Queries

In this section we introduce the syntax and semantics of the operator CLOSURE-OF that enables us to compute recursive queries. The semantics of the operator is defined through an extension of the navigational algebra presented in section 4, similarly to the definition of generalized transitive closure given in [31]. Moreover, we compare the class of queries obtained from the CLOSURE-OF operator with the class of recursive queries computable by G^+ [12, 13].

5.1 Syntax and Semantics

The syntax of QBD* is as follows:

<QBD*>::={<recursive_query> | <QBD>}+
<recursive_query>::= CLOSURE-OF <ent_rel_name> WITH <condition>

where <QBD>, <ent_rel_name>,<condition>, are defined in section 4.1.

In order to provide the semantics of the CLOSURE-OF operator, we now consider the structure of a typical query containing such an operator, and we characterize the result of the query by showing the query computing the answer.
The structure of the query is:

CLOSURE-OF ***E*** WITH C,ADD A,DEL D.

The corresponding program, computing relation s as answer, is:

s:= $\mathbb{T}$**E**;
t:= $\mathbb{T}$**E;**
repeat
t := **M**(SEL **E** BRIDGE $\mathbb{U}$s BUILTWITH C, ADD A,DEL D);
s:= s ∪ t
until s – t = Ø

Notice that, within the body of the loop, the relation s, initially set to $\mathbb{T}$**E**, is augmented with the tuples resulting from the application of the bridge operator to the entity-set (**E**) and to the entity-set corresponding to the current value of s ($\mathbb{U}$s).
In order to guarantee that the relational schema of t is equivalent to $\mathbb{S}\mathbb{T}$**E** it must be:

attr(**E**)∪A-D=attr(**E**)

except for the renaming of the attributes. In the following such a renaming is denoted by writing *br.att* instead of *att* when the attribute *att* is considered in any entity-set after the operator BRIDGE has been applied to (according to the syntax of QBD*).

The graphical realization of the closure operator is shown in fig.8. For the sake of simplicity, the entity E is represented with only two attributes. Note that two instances of such a graphical realization appeared in Section 3 (figg.4, 5).

Following the approach developed in section 4, we can also describe the semantics of the operator CLOSURE-OF in terms of closure of navigations.

In particular:

$$\mathbf{M}(\underline{\text{CLOSURE-OF}}\ \mathbf{E}\ \underline{\text{WITH}}\ \ C,\underline{\text{ADD}}\ A,\underline{\text{DEL}}\ D) = (\nu_{par})^{+}\ \mathbb{T}\ \mathbf{E}$$

where

$$\nu_{par}=\nu_{(attr(E)-D)\cup A,T,T,C}$$

and

$$(\nu_{par})^{+}\mathbb{T}\ \mathbf{E} = \ ''_{i<\infty}(\nu_{par})^{i}\ \mathbb{T}\ \mathbf{E}$$

with $(\nu_{par})^{1}\mathbb{T}\ \mathbf{E}= \mathbb{T}\ \mathbf{E}$ and $(\nu_{par})^{i+1}\mathbb{T}\ \mathbf{E}= \nu_{par}(\mathbb{T}\ \mathbf{E},\ (\nu_{par})^{i}\mathbb{T}\ \mathbf{E})$.

This means that $(\nu_{par})^{+}\mathbb{T}\ \mathbf{E}$ is the generalized transitive closure of $\mathbb{T}\mathbf{E}$ with respect to the navigation ν_{par}. It holds that ν_{par} plays the role of the binary operator *g* introduced in [31], without associative constraints defined over the condition C and attributes (attr(E)-D)∪A .

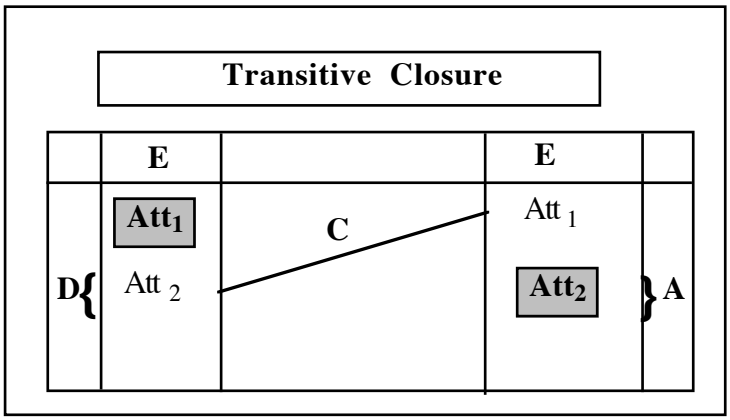


Fig. 8: Graphical realization of the operator CLOSURE-OF

5.2 Comparative Analysis

The aim of this subsection is to compare the class of queries computed by QBD* with the class of queries computed by G+. First of all, we recall some basic notion about G+ taken from [27].

Definition A Database Graph $G=(N,E,\phi,\Sigma,\lambda)$ is a directed , labelled graph, where N is a set of nodes, E is a set of edges, and ϕ is an incidence function mapping E to $N\times N$. So multiple edges between a pair of nodes are permitted in database graphs. The labels of G are drawn from the finite set of symbols Σ, called the *alphabet*, and λ is an edge labelling function mapping E to Σ.

Definition Let Σ be a finite alphabet disjoint from $\{\varepsilon, \varnothing, (,)\}$. A *regular expression e* over Σ and the language *L(e)* denoted by e are defined in the usual way. Let $G=(N,E,\phi,\Sigma,\lambda)$ be a database graph and $p=(v_1,d_1,..,v_{n-1},d_{n-1},v_n)$, where $v_i\in N$, $1\le i\le n$, and $d_i\in E$, $1\le i\le n-1$, be a path in G. We call the string $\lambda(d_1)..\lambda(d_{n-1})$ the *path label* of p, denoted by $\lambda(p)\in\Sigma^*$. Let e be a regular expression over Σ. We say that the path p *satisfies e* if $\lambda(p)\in L(e)$.
Furthermore, we say that p is *simple* if all v_i are distinct.

Definition The query Q_e on a Database Graph G is defined as the function from G to the set of pairs (x,y) such that there exists a simple path p from x to y in G satisfying e. It is useful to denote the query Q_e by the picture (query graph)

x $\xrightarrow{\quad e \quad}$ y

where x and y are variables interpreted on N.
Furthermore the set $\{Q_e \mid Q_e$ is a query on G and e is a regular expression over the edge labels of $G\}$ will be denoted with $\{Q_e\}_G$

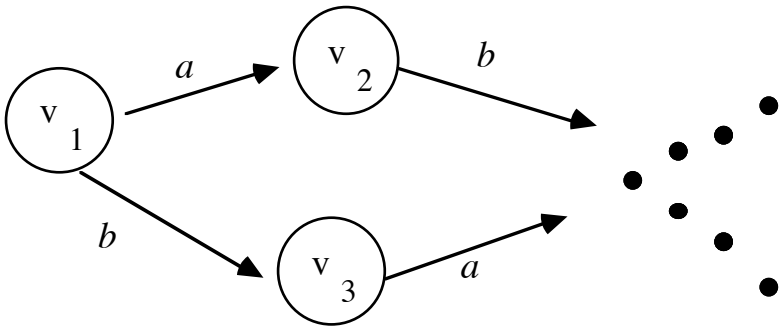


Fig.9: Database graph G with $\Sigma=\{a,b\}$

Consider the Database Graph G in fig. 9, with $\Sigma=\{a,b\}$. The corresponding E-R schema, shown in fig.10, contains the following objects:
- an entity-set **E**(a,b) whose entities are the vertices of G ;
- a key attribute of **E** corresponding to the names of vertices (Vname);

- two attributes of **E**, called Edge*a* and Edge*b*, such that for each vertex having Vname=*v* , Edge*a* (risp. Edge*b*) gives the name of the next vertex that appears along the edge *a* (risp.*b*). If there is no successor,the value is undefined.

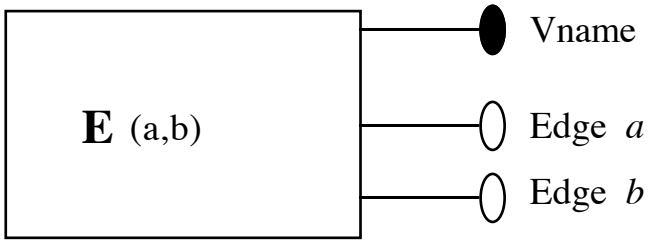


Fig.10: The E-R schema corresponding to the schema of the Database Graph*G*

Theorem 3 Every query in $\{Q_e\}_G$ is definable in QBD*:

$$\{Q_e\}_G \subseteq \mathbf{M}(\text{QBD}^*)$$

proof By induction on the structure of *e* .

[Induction Basis] We must compute all pairs of nodes satisfying the query graph (query Q_a)

x a y

It holds that:

$$Q_a \equiv \mathbf{M}(\text{ SEL } \mathbf{E}(a,b) \text{ WITH Edge}a \neq \text{Vname,DEL Edge}b)$$

Note that the condition Edge*a* ≠ Vname comes from the fact that the paths are simple.

[Induction Step]

Let *e*,*f* be regular expressions over the alphabet {*a*,*b* }. By the inductive hypothesis we have two queries P_e (risp. P_f) ∈ QBD* computing Q_e (risp.Q_f). Let $\mathbf{E}_e = P_e$ (**E**(*a*,*b*)) (risp. $\mathbf{E}_f = P_f$ (**E**(*a*,*b*))). In other words, $\mathbf{E}_e$ (risp. $\mathbf{E}_f$) is the entity-set corresponding to all pairs of vertices connected by a simple path satisfying *e* [*f*]. Note that if the path does not exist, the vertex is considered as labelled 'undef' (see fig. 11). Therefore, let us define

$$\mathbf{E}_{e,f} = \text{SEL}\mathbf{E}_e \text{ BRIDGE } \mathbf{E}_f \text{ BUILTWITH Vname=BR.Vname,ADD BR.Edge}f$$

i.e. $\mathbf{E}_{e,f}$ is the entity-set of vertices connected by paths satisfying *e* or *f* .

(*concatenation*) We must compute all pairs of nodes satisfying the query graph

(query Q_{ef})

x £ £ ¢ y.

It holds that:

Q_{ef} ≠ **M**(SEL **E**$_{e,f}$ BRIDGE **E**$_{e,f}$

BUILTWITH Edge$e$ =BR.Vname,DELEdge$f$,Edge$e$,ADD BR.Edge$f$)

(*disjunction*) We must compute all pairs of nodes satisfying the query graph
(query $Q_{e|f}$)

x £ £ £ ¢ y.

It holds that:

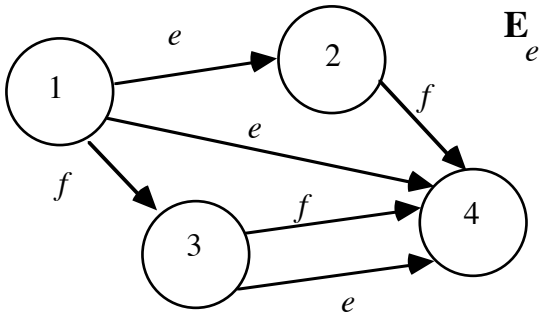
$Q_{e|f}$ ≠ **M**(SEL **E**$_e$ ENDSEL UNION SEL **E**$_f$ ENDSEL)

(*closure*) We must compute all pairs of nodes satisfying the query graph
(query Q_{e+})

x £ £ £ ¢ y.

It holds that:

Q_{e+} ≡ **M**(CLOSURE-OF **E**$_e$ WITH Edge$e$ = BR.Vname,DEL Edge$e$,ADD BR.Edge$e$)



E$_e$ = {<1, 2>, <1, 4>, <2, 'undef'>, <3, 4>, <4,'undef'>}

E$_f$ = {<1, 3>, <2, 4>, <3, 4>, <4,'undef'>}

E$_{e,f}$ = {<1, 2, 3>, <1, 4, 3>, <2, 'undef', 4>, <3, 4, 4>, <4,'undef', 'undef'>}

Fig.11: simple paths and associated Entity-sets

Two different approaches have been proposed in order to enhance the expressive power of G^+(see [13]):

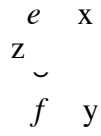
1) drawing query graphs more complex than the ones corresponding to $\{Q_e\}_G$, i.e. with more than two nodes and more than one edge;

2) introducing summarizing operators, such as the one counting the length of a path.

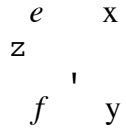
The first approach is largely the most significant one, and therefore, on the rest of this subsection, we concentrate on this.

Three basic mechanisms have been defined on G^+ in order to consider more general query graphs:

1) *common ancestor* query $CAQ_{e,f}$, that, given two regular expression e,f, results into all the pairs of vertices x,y sharing a common ancestor via two simple paths satisfying e,f respectively, as shown in the query graph:



2) *common descendant* query $CDQ_{e,f}$ that simply reverses the direction of paths in $CAQ_{e,f}$, as shown in the query graph:



3) *common* query $CQ_{e,f}$ that, given two regular expression e,f, results into all the pairs of vertices x,y connected by a simple path satisfying e, and by a simple path satisfying f, as shown in the query graph

We now show that all the above mechanisms are expressible in QBD^*.

Theorem 4 Every $CAQ_{e,f}$, $CDQ_{e,f}$ and $CQ_{e,f}$ are definable in QBD^*, i. e.

$$\{CAQ_{e,f}\}_G \cup \{CDQ_{e,f}\}_G \cup \{CQ_{e,f}\}_G \subseteq \mathbf{M}(QBD^*)$$

proof

$CAQ_{e,f} \equiv \mathbf{M}(\underline{\text{SEL}}\,\mathbf{E}_{e,f}\,\underline{\text{ENDSEL}}\,\underline{\text{WITH}}$ Edge$e \neq$'undef' $\underline{\text{AND}}$ Edge$f \neq$'undef', $\underline{\text{DEL}}$ Vname)

$CDQ_{e,f} \equiv \mathbf{M}(\underline{\text{SEL}}\,\mathbf{E}_{e,f}\,\underline{\text{BRIDGE}}\,\mathbf{E}_{e,f}\,\underline{\text{BUILTWITH}}$ Edge$e$ = $\underline{\text{BR.}}$Edge$f$, $\underline{\text{DEL}}$ Edge$f$,Edge$e$,$\underline{\text{ADD}}$ $\underline{\text{BR.}}$Vname)

and

$CQ_{e,f} \equiv \mathbf{M}(\underline{\text{SEL}}\,\mathbf{E}_{e,f}\,\underline{\text{WITH}}$ Edge$e$ =Edge$f$, $\underline{\text{DEL}}$ Edge$f$)

As an application, let us consider a database graph G whose vertices are of type person-name and whose edge labels are F,M,B (denoting "father of", "mother of" relation and "bequeather of" relation, respectively). Let us define as 'bequeather query' (Qb) the query which finds all pairs (x,y) of entities such that x is an ancestor of y and, for each descendant path, each entity bequeathers to his/her child on the path. For instance, fig.12 shows an example of database graph defined on five persons.

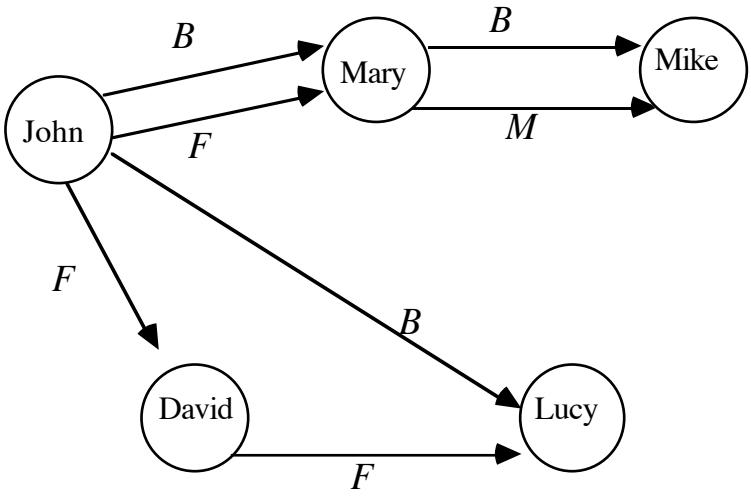


fig.12 Database graph G of persons

In this case the answer to the query is:

$$Qb(G)=\{ (John, Mary), (John, Mike), (Mary, Mike) \}$$

Strictly speaking Qb cannot be defined in G^+. However, if a composition operator is introduced, then Qb may be computed by composing the following two queries ([11]):

q_1:

$$x \xrightarrow{M|F \atop B} y \Rightarrow x \xrightarrow{M|F\_B} y$$

q_2:

$$x \xrightarrow{(M|F\_B)+} y \Rightarrow x \xrightarrow{\quad} y$$

where M|F_B is a name identifying the new edge resulting from the query q_1. Then, if we apply q_1 to G in fig.12 we obtain the graph in fig. 13. Applying q_2 on this graph we obtain the answer Qb(G).

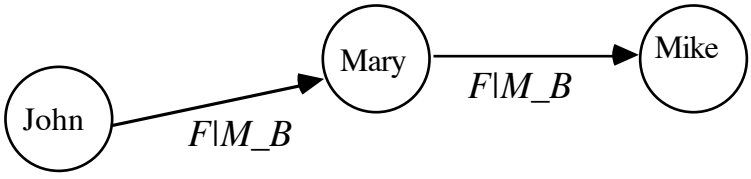


fig. 13: Intermediate Database graph

The class of queries resulting from the application of the composition operator has not yet been formally characterized in G^+.Therefore, in order to formally compare the extension of G^+ with QBD^*, we introduce the notion of visual path on Database graphs.

We call *visual path* (with respect to $CQ_{e,f}$) a sequence of (distinguished) vertices

$$p = (v_1,..,v_{n-1},v_n) \text{ in } G$$

where $v_i \in N$ and all pairs (v_i, v_{i+1}) satisfy e and f.

Definition The query $(CQ_{e,f})^+$ on a Database Graph G is defined as the function mapping G to the set of pairs (x,y) such that a visual path p from x to y in G with respect to $CQ_{e,f}$ exists. (Note that Qb is equal to $(CQ_{F/M,B})^+$)

Theorem_5 For all e, f a G exists such that $(CQ_{e,f})^+(G) \subset CQ_{e+,f+}(G)$

proof By definition it holds that for all G

$$(CQ_{e,f})^+(G) \subseteq CQ_{e+,f+}(G)$$

Let G be equal to the database graph obtained by taking two simple paths p and q such that p satisfies e^+ and q satisfies f^+. Let us suppose that their length is greater than 2, and that all vertices are different a part from the first one and the last one, in this case it holds that does not exists a pair of vertices in G satisfying e and f, so the thesis follows.

˘

Informally speaking the previous result shows that the closure of query graphs is more powerful than the closure of finite regular expressions. Since the CLOSURE-OF operator is defined independently from the edge labels we expect to compute $(CQ_{e,f})^+$in QBD^*.

The following theorem shows that this is the case. The graphical realization of the query is shown in fig.14.

Theorem_6 $(CQ_{e,f})^+$is definable in QBD^*

proof

$$(CQ_{e,f})^+ \equiv \mathbf{M}(\text{CLOSURE\_}\mathbf{E}_{e,f} \text{ WITH Edge}e = \text{BR.Vname, AND Edge}f = \text{BR.Vname,}$$
$$\text{DEL Edge}f, \text{Edge}e, \text{ ADD BR.Edge}e, \text{BR.Edge}f)$$

˘

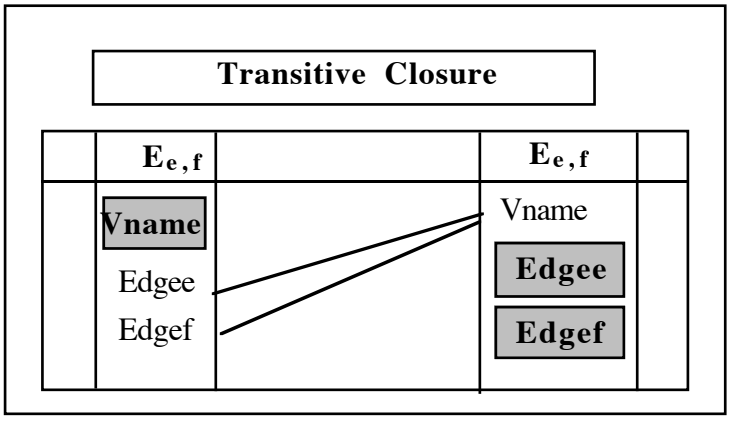


Fig.14: The graphical realization of the query of theorem 6

queries	G^+ definition	QBD^* definition
Q_e	x y	theorem 3
$CAQ_{e,f}$	x y	
$CDQ_{e,f}$	x y	theorem 4
$CQ_{e,f}$	x y	theorem 4
$(CQ_{e,f})^+$	composition	theorem 6

Table 1: Summary of the comparison between G^+ and QBD^*

In table 1, we summarize our results about the comparison between G^+ and QBD*.
It is worth noting that table 1 can be seen as a characterization of query graphs in terms of relational algebra expressions (navigational algebra expressions) plus generalized transitive closure (see Sec. 5.1).
More precisely, by theorems 3,4,6 and following the semantics of QBD*, we have:

$$\{Q_e, CAQ_{e,f}, CDQ_{e,f}, CQ_{e,f}, (CQ_{e,f})^+\}_G \subseteq \mathbf{ALG}(\cup, \nu_{A,C,T,J}, (\nu_{A,T,T,L})^+)$$

where:
- A is the set of attributes derived from attr($\mathbf{E}(a,b)$) by means of the DEL and the ADD operators used in theorems 3,4,6;
- C is a condition of type EDGEa ≠ Vname (theorem 3, induction basis)
- J is one of the bridge conditions occurring in the theorems 3,4,6;
- L is one of he closure conditions occurring in the theorems 3,6.

6. Conclusions

In this paper we have presented QBD*, a graphical query language that allows for expressing both non-recursive and recursive queries. We have characterized the expressive power of QBD* by means of a formalism suited to describe graphical query languages expressing recursive queries. In particular, we have used such a formalism for the comparison between our language and another graphical language, namely G^+ [13].

We have also described the whole query system, that uses a fully graphical interface both to specify the schema of interest and to express queries. It has been shown [24, 28] that casual users easily understand a diagrammatic representation of the conceptual schema. As a consequence, these representations may be used during the whole life cycle of the information system, not only in the conceptual design phase. This allows the user to reach a considerable autonomy in the data management activity.
We believe that both non expert and experienced users may benefit from the visual interaction dimension offered by QBD*. In principle, the rich set of strategies embedded in QBD* makes non expert users potentially able to effectively use the system, provided she/he understands the E-R model together with a simple set of mechanisms to navigate through the schema. Furthermore, expert users are provided with a useful kit of tools, aiming to simplify the execution of more complex activities, such as the definition of proper views and the execution of non first-order queries.

A prototype of QBD* has been developed at University of Rome "La Sapienza", Dipartimento di Informatica e Sistemistica. The development environment is based on the C language and the

operating system MS-DOS; the graphical interface is supported by the tool HALO according to the standard GKS, using algorithms for automatic layout described in [33].

Further research work will concern the extension of the system to manage statistical data. To this purpose, the query language allow the definition of aggregations and statistical operations and the graphical interface is based on the statistical data model defined in [5].

Acknowledgements

We would like to thank Carlo Batini and Maurizio Lenzerini for many extremely helpful suggestions, ideas and comments. Thanks are also due to Isabel Cruz for her valuable suggestions. Finally, we are grateful to the anonymous referees for their comments.

References

[1] Special Issue - The Psychology of Human-Computer Interaction - Vol 13, N.1, March 1981.
[2] A.V.Aho, J.D.Ullman - Universality of Data Retrieval Language - pp. 110-120 in Proc. of the 6th ACM SIGACT-SIPLAN Symposium on Principles of Programming Languages, 1979.
[3] M.Angelaccio, T.Catarci, G.Santucci - QBD*: A Fully Visual System for E-R Oriented Databases - in the Proc. of the Workshop on Visual Language 1989, Roma, October 1989.
[4] M.Angelaccio, T.Catarci, G.Santucci - QBD*: A Fully Visual Query System - to appear in Vol. 1, No. 2 del Journal on Visual Languages and Computing.
[5] C.Batini, G.Di Battista - Design of Statistical Databases: a Methodology for the Conceptual Step - in Information Systems, Vol.13, 4, 1988.
[6] T.Catarci, G.Santucci - QBD: A Graphic Query System - in Proc. of the 7th Entity-Relationship Conference, Rome, November 1988.
[7] A.K.Chandra, D.Harel - Computable Queries for Relational Databases - pp. 156-178 in Journal of Computer Sciences, Vol. 21, 1980.
[8] P.P.Chen - The Entity-Relationship Model toward a Unified View of Data - in ACM Transactions on Data Base Systems, 1976.
[9] E.F.Codd - Relational completeness of database sub-languages - pp. 65-98 in R.Rustin (ed.) Data Base Systems, Prentice Hall, Englewood Cliffs, 1972.
[10] I.F.Cruz - Domains of Application for the G+ Query Language - Technical Report CSRI-212, Computer Systems Research Institute, University of Toronto, September 1988.
[11] I.F.Cruz, A.O.Mendelzon, Personal Communication, October 1989.
[12] I.F.Cruz, A.O.Mendelzon, P.T.Wood - A Graphical Query Language Supporting Recursion - in Proc. ACM SIGMOD Conf. on Management of Data, 1987.

[13] I.F.Cruz, A.O.Mendelzon, P.T.Wood - G^+: Recursive Queries Without Recursion - pp. 355-368 in Proc. of the 2nd International Conference on Expert Database Systems, April 1988.
[14] B.Czejdo, D.W.Embley - An approach to computation specification for an entity-relationship query language - in S.March (ed.) Proc. of the 6th International Conference on Entity-Relationship Approach, New York, 1987, pp. 307-322, 1987.
[15] Czejdo B., Embley D., Reddy V., Rusinkiewicz M., A Visual Query Language for an E-R Data Model, to appear in Proc. of the Int. Workshop on Visual Languages, Rome, 1989.
[16] C.J.Date - An Introduction to Database Systems - Vol.I, Addison-Wesley Publishing Company, 1987.
[17] R.Elmasri, J.A.Larson - A Graphical Query Facility for ER Databases - pp. 236-245 in Proc. of the 4th International Conference on Entity-Relationship Approach, Chicago, Illinois, 1985.
[18] R.Elmasri, G.Wiederhold - GORDAS : a formal high-level query language for the entity-relationship model - pp. 49-72 in Proc. of the 2nd International Conference on Entity-Relationship Approach, Washington, D.C., 1981.
[19] H.R.Hartson, D.Hix - Human-Computer Interface Development: Concepts and Systems - ACM Computing Surveys, Vol. 21, N.1, March 1989.
[20] Y.E.Ioannidis, E.Wong - An Algebraic Approach to Recursive Inference - pp.209-223 in Proceedings of the First International Conference on Expert Systems, April 1986.
[21] H. Kangassalo - CONCEPT D: A Graphical Language for Conceptual Modelling and Data Base Use - invited paper in: IEEE 1988 Workshop on Visual Languages, Pittsburgh, USA, October 1988.
[22] Kuntz M., Melchert R., Pasta-3's Graphical Query Language: Direct Manipulation, Cooperative Queries, Full Expressive Power, in: Proc. of the Fifteenth Int. Conference on Very Large Data Bases, Amsterdam, August 1989.
[23] V.S.Lakshmanan, A.O.Mendelzon - On THe Expressive Power of DATALOG: A New Technique Based on Inductive Pebble Games - in Proc. of the 8th ACM SIGACT-SIGMOD-SIGART Symp.on Principles of Database Systems, 1989.
[24] J.A.Larson - Visual Languages for Database Users - in: Visual Languages, S.K. Chang (ed.), Plenum Publishing, New York, 1986.
[25] V.M.Markowitz - ERROL : an entity-relationship role oriented query language - in Proc. of the 3rd Conference on Entity-Relationship Approach, 1983.
[26] H.M.Markowitz, A.Malhotra, D.P.Pazel - The ER and EAS Formalism for System Modeling, and the EAS-E Language - pp. 29-47 in P.P. Chen (ed.) Entity Relationship Approach to Information Modeling and Analysis, ER Institute, 1981.
[27] A.O.Mendelzon, P.T.Wood - Finding Regular Simple Paths in Graph Databases - in Proc. of the Int. Conference on Very Large Databases, Amsterdam, 1989.

[28] F.S. Montalvo - Diagram Understanding: Associating Symbolic Descriptions with Images - in Proc. Proc. of the Int. Workshop on Visual Languages, 1984.
[29] U. Nanni - A Graphic Interface for Relational Databases - Technical Report, University of Rome, Italy, October 1987.
[30] T.R.Rogers, R.G.G.Cattell - Entity-Relationship Database User Interfaces - pp. 323-336 in S.March (ed.) Proc. of the 6th International Conference on Entity-Relationship Approach, New York 1987.
[31] S.Sippu, E.Soisalon-Soininen - A Generalized Transitive Closure for Relational Queries - in Proc. of the International Conference on Principle of Database Systems, 1988.
[32] K.Subieta, M.Missala - Semantics of query languages for the entity relationship model - in S.March (ed.) Proc. of the 6th International Conference on Entity-Relationship Approach, New York 1987.
[33] R. Tamassia, G. Di Battista, C. Batini - Automatic Graph Drawing and Readability of Diagrams - IEEE Transactions on Systems, Men and Cybernetics, 1988.
[34] J.Tillquist, F.Y.Kuo - An Approach to the Recursive Retrieval Problem in the Relational Databases - Communications of the ACM, Vol.32, N.2, pp.239-245, February 1989.
[35] J.D.Ullman - Principles of Database and Knowledge-Base Systems - vol. I, 1987.
[36] H.K.T.Wong, I.Kuo - GUIDE : Graphical User Interface for Database Exploration - pp. 22-31 in Proc. of the 8th VLDB Conference, Mexico City, 1982.
[37] Z.Q.Zhang, A.O.Mendelzon - A Graphical Query Language for Entity-Relationship Databases - in Proc. of 3rd International Conference on Entity-Relationship Approach, 1983.
[38] M.M.Zloof - Query-by-Example; a database language - IBM Syst.J., 1977.