An SQL-based Query Language For Networks of Relations

Amit Basu
College of Business and Management
University of Maryland, College Park, MD

and

Rafiul Ahad
Data Management Systems Div.
Hewlett Packard Co.
Cupertino, CA

Abstract

A set of relations can be modeled as a network through the use of *image attributes*, which are attributes defined on the domain of relation names. Such networks of relations can effectively meet many of the modeling requirements of advanced database applications such as engineering design and knowledge base systems. In this paper, we describe the features of ESQL, a novel query language that is an extension of SQL [Cham74] to exploit the added semantics of image attributes. Details of ESQL and its underlying principles can be found in [Ahad88] and [Ahad89].

1 Introduction

In recent years, widespread application of relational databases has among other things, led to the realization that the relational model as introduced by Codd does not provide some of the modeling capabilities needed for many applications, including advanced applications such as engineering design. Such capabilities include support for the following features:

1. *Vertical inhomogeniety* [Kent79]. This occurs when different instances of an attribute have different semantics and/or structure. For example, in the relation COMP [SYSID, SUBSYSorPARTID] to store a system composition hierarchy, the attribute SUBSYSorPARTID may be a subsystem identifier or a part identifier. These two identifiers are not just different in the pattern used to represent them but also in their semantics. For example, if the second attribute of a COMP tuple is a subsystem identifier then it can be used to join with other tuples of COMP on SYSID, and if it is a part identifier it is used to access another relation PART.

2. *Horizontal inhomogeniety* [Kent79]. This occurs when different tuples of a relation have different attributes. Consider a relation to store parts, as in the above example. It is obvious that not all the parts have the same set of attributes. The attribute COLOR, for instance, does not apply to a part like piston ring, whereas it is relevant to a part like steering wheel.

As an example, consider the database shown below, in which the relation PICTURE shows the primitive elements that make up a picture and their coordinates. There are three types of primitive elements, CIRCLE, ELLIPSE, and POLYGON which are represented in corresponding relations.

Example 1:

```
PICTURE(X, Y, PRIMITIVE, TID)
CIRCLE(TID, RADIUS)
ELLIPSE(TID, MAJOR, MINOR, ANGLE)
POLYGON(TID, SNO, X, Y)
```

where the domain of PRIMITIVE is the set { CIRCLE, ELLIPSE, POLYGON }. Note that TIDs are unique only within a relation. The picture shown in figure 1 can be represented using the relations shown in figure 2.

•

With a query language like SQL, a query such as "List the picture instances contained in a certain window" has to be broken down into n queries where n is the cardinality of *Dom*(PRIMITIVE). Each of these queries will select tuples of PICTURE having some value, say v, in the PRIMITIVE attribute and join the selected tuples with the relation v. The problem in this example is that the type of the tuple to be joined with each tuple of PICTURE is not known until the tuple variable of PICTURE is bound, which only occurs at run time.

Both vertical and horizontal inhomogeneity are significant in the context of this example. For instance, the semantics of the attribute TID in PICTURE depends upon the primitive type to which each individual tuple corresponds. Furthermore, the answer produced may consist of up to n different relations, which reflects horizontal inhomogeneity.

We find that the modeling capabilities of the relational model can be enhanced to support these features by allowing domains whose values are names of relations. Such domains are known as *image domains* [Smit77]; attributes defined on image domains are known as *image attributes*. In [Smit77], image attributes are used to capture generalization and specialization. Also, in his RM/T proposal [Codd79], Codd permits the uniform manipulation of schema information and the database extension through the use of two operators, NOTE and DENOTE, on the image domain RELNAME in the catalog. However, the use of image domains is restricted to the catalog only. On the other hand, ESQL exploits the semantics of image attributes more fully, while remaining an extension of SQL [Cham74]. ESQL can be used to query multirelations (a set of relations induced by an image attribute of a relation extension) without having to know the names of the individual member relations.

This modeling approach based on image domains is a simple and powerful technique to model part-subpart hierarchies, generalization/specialization hierarchies, statistical tables, proof trees in knowledge-based systems, and networks of relations. Since ESQL is a superset of SQL and the latter has become a defacto industry standard with a large population of end-users, the effort needed to understand and use ESQL should be minimal. We envision ESQL to be not only effective but also efficient compared to other approaches. ESQL preprocessors can be built relatively easily on top of a relational database management system supporting SQL. A detailed implementation technique using this approach is described in [Ahad88].

The remainder of this paper is organized as follows. In section 2, we define multirelations, describe the semantics of ESQL, and illustrate how multirelations can be queried using ESQL. In section 3, we briefly describe our implementation technique for ESQL. In section 4, we compare our approach to existing work on extensions of the relational model. Finally, section 5 contains concluding remarks.

2 Multirelations and ESQL

In this section, we introduce the concepts of *multirelations*[1] and then show how they are used in the query language ESQL.

2.1 Multirelations

Definition A multirelation is a distinguished set of relations. Two multirelations $M1$ and $M2$ are identical iff $\forall x \in M1$, $\exists y \in M2$ such that $Name(x) = Name(y)$ and $\forall x \in M2$, $\exists y \in M1$ such that $Name(x) = Name(y)$. A relation can be a member of many multirelations simultaneously.

Multirelations are defined using image attributes in relations. An important consequence of this is that existing data definition languages for the relational model can be used to define and create multirelations. For example, if R is a multirelation and A is a character-string attribute of R whose values are relation names, then $R[A]$ (the projection of R over A) is a multirelation.

Definition If R is a relation and A is an image attribute in R, then $R : A$ is a multirelation whose members are $R[A]$. If M is a multirelation and A is an image attribute, then $M : A$ is a multirelation whose members are $\{X | X \in R[A] \land R \in M \land Dom(A) \subseteq \mathcal{R}\}$.

Stated another way: if R is a relation or a multirelation, then $R : A_1 : \ldots : A_n$, $n > 0$, is a

[1] Our use of the term differs from the interpretation in [Klau85], where a multirelation is characterized as a single relation containing duplicate tuples.

multirelation where A_1 is an image attribute of R. The members of the multirelation $R : A_1 : ... : A_i$ are $\{X | X \in S[A_i] \wedge S \in R : A_1 : ... : A_{i-1} \wedge Dom(A_i) \subseteq \mathcal{R}\}$. Notice that if for all members of the multirelation $R : A_1 : ... : A_{i-1}$ the image attribute A_i is not defined then the multirelations $R : A_1 : ... : A_j, j \geq i$ will be empty. $R : A_1 : ... : A_{n-1}$ is called the *parent* of $R : A_1 : ... : A_n$. In general $R : A_1 : ... : A_i$ is called an *ancestor* of $R : A_1 : ... : A_i : ... : A_n$ where $n > i$.

The schema of a multirelation is a set of schemas, each of which is the schema of a member. The extension of a multirelation is a set of relation extensions, each of which is the extension of a member of the multirelation. Note that the schema and extension of a multirelation are determined from the extension of its parent. Since the extension of the parent varies over time, so does the membership in the multirelation. With respect to example 1, if the extension of PICTURE is as shown in figure 2, then PICTURE:PRIMITIVE is a multirelation whose members are CIRCLE, ELLIPSE and POLYGON. The parent of this multirelation is PICTURE.

2.2 Dynamically-Typed Tuple Variables

Definition A variable that is capable of representing a tuple of any relation in a multirelation is called a *Dynamically-Typed Tuple Variable* (DTTV). The type of a DTTV is a multirelation[2]. In a query, the type of a DTTV V is specified as $U : A_1 : ... : A_n$ where U is a variable (a tuple variable or another DTTV); variable U is called the *root* of variable V.

A variable V defined on a multirelation is called a DTTV because $Type(V)$ is determined by the binding of its root, and since the root variable can be bound to different tuples at run time, $Type(V)$ changes during query execution and hence is dynamic. Notice that for two variables U and V, if U is the root of V, then $Type(U)$ is an ancestor of $Type(V)$.

DTTVs are used in a query in the same way tuple variables are used. However, since a DTTV may be bound to tuples in different relations, and since the universal attribute specifier (e.g. '*' in SQL and 'ALL' in QUEL) is allowed, the response set of a query involving a DTTV may contain heterogeneous tuples. For meaningful presentation, these tuples can be organized into a set of relations, i.e., a multirelation. Furthermore, the relations must be given names for two reasons: to give a clue as to how they were obtained, and to store the result of a query. These issues are addressed in the next section in the context of a specific query language, ESQL.

[2]We will view $Type(V)$ where V is a single relation, to be a singleton set

2.3 Semantics of ESQL

In ESQL, DTTVs can be used wherever tuple variables are used in SQL. Like in SQL, ESQL permits the multirelation name to be used as its sole tuple variable. The semantics of ESQL queries that do not contain DTTVs is identical to the corresponding SQL queries. We describe the semantics of some important ESQL statements, using examples.[3]

2.3.1 Single-Variable Queries

If $Q(M)$ is a query on a multirelation M, then Q is applied to every qualifying member of M (binding rules that determine which members of a multirelation qualify for a query are described in [Ahad88]). That is, Q is applied to every member of M that has:

1. attributes needed to satisfy at least one conjunct of the disjunctive normal form of the condition of Q. (rule 2), and
2. at least one attribute to be retrieved (i.e., at least one of the attributes specified in the $SELECT$ clause of the query).

As many attributes as possible are retrieved from each qualifying member of M. For such a member, say R, qualifying tuples are displayed as a relation whose name is $\$R$. This way, the user can read the result more easily. The naming convention is discussed fully in [Ahad88].

Example 2. Suppose R is a unary relation whose sole attribute G is an image attribute, and whose current extension contains three tuples $\langle R1 \rangle$, $\langle R2 \rangle$ and $\langle R3 \rangle$. Suppose the schemas of the three relations are $R1(A,B,C), R2(A,C,D)$ and $R3(A,B,D)\}$. Consider the following query:

[3]Semantics of other ESQL statements are discussed in [Ahad88].

```
SELECT A, B, C
FROM R:G
WHERE R:G.A = 3.0;
```

Here the condition of Q is "R:G.A = 3.0". Since all the relations in $R : G$ have all the attributes to satisfy this selection condition, Q can be applied to all of them. The result of the query will be displayed as three relations, $\$R1[A,B,C]$, $\$R2[A,C]$, $\$R3[A,B]$, where $\$Ri$ is obtained from Ri by selecting the tuples whose A value is 3.0, and projecting on $Schema(R_i) \bigcap O_Q$, where O_Q denotes the output attribute set of the query Q (the attributes in the *Select* clause). These three relations form the result multirelation.

Example 3.

With the same multirelation as in example 2, consider the query Q:

```
SELECT R:G.*
FROM R:G
WHERE R:G.A = 3.0 and R:G.B = 4.0;
```

Here the condition is "R:G.A = 3.0 and R:G.B = 4.0". Now $R1$ and $R3$ can satisfy the condition and thus Q is applied to $R1$ and $R3$ giving the result $\{\$R1[A,B,C], \$R3[A,B,D]\}$. Notice that if the condition had been "R:G.A = 3.0 or R:G.B = 4.0", all three relations would have qualified.

●

2.3.2 Multi-Variable Queries

We next describe the semantics of two-variable queries. The semantics of queries involving three or more variables can be inferred from the semantics of the two-variable queries. Two-variable queries fall into two distinct classes. In the first class, the two variables are independent and in the second class one variable is an ancestor of the other.

Example 4. *Independent Variables*

Suppose the relation R is as defined in example 2 and S is a similar relation whose image attribute is H and whose tuples are $\langle R4\rangle$, $\langle R5\rangle$, $\langle R6\rangle$ where the schemas are $R4[W,X]$, $R5[W,Y]$, $R6[Z,Y]$. If Q is:

```
SELECT R:G.*, S:H.*
FROM R:G, S:H
WHERE R:G.A = S:H.W;
```

The result of this query is represented in the multirelation consisting of the relations -

$\{\$R1\$R4[A,B,C,W,X]$,
$\$R1\$R5[A,B,C,W,Y]$,
$\$R2\$R4[A,C,D,W,X]$,
$\$R2\$R5[A,C,D,W,Y]\}$

●

The following examples show different types of queries involving *dependent variables*.

Example 5. In this example, we show how a part hierarchy can be accessed. Suppose we want to retrieve all attributes of primitives whose origins are above the line $Y = 2$ in example 1, we could use the following ESQL query.

```
SELECT Q.*, P.X, P.Y
FROM P PICTURE, Q P:PRIMITIVE
WHERE P.TID = Q.TID AND P.Y > 2.0;
```

The result of this query as it will be displayed on the screen is shown in figure 3.

Example 6. This is another example of a part hierarchy, where no common attributes are present. Consider the following schema for part of a database for circuit design:

```
R[CNAME, NETLIST] ; Dom(NETLIST) =
C1, ...  , Cn
  C1 [FROM, TO]
  .
  .
  Ci [FROM, TO, TYPE]
  .
  .
  CN [FROM, TO]
```

A query to print the netlist of a given circuit name is:

```
SELECT R:NETLIST.*
FROM R, R:NETLIST
WHERE R.CNAME = given;
```

●

Notice that no specific join condition is needed in the above query because of the way binding occurs. Specifically, the tuples from R that have the given value for attribute $CNAME$ are first retrieved, say as relation S (in the above example S is a singleton relation). Then the product of S and $S[NETLIST]$ is computed and the other conditions of the WHERE clause are tested. Notice that C_i's do not contain an attribute to carry their names.

Example 7. This example shows how ESQL can be used to retrieve information from a specialization hierarchy. Consider the following schema:

```
  PERSON [SS#, NAME, PTYPE];
Dom(PTYPE) = {STU,STAFF,FAC}
  STU [SS#, MAJOR]
  STAFF [SS#, STYPE, MANAGER];
Dom(STYPE) = {FULLTS,PARTTS}
  FAC [SS#, RANK]
  FULLTS [SS#, SALARY]
  PARTTS [SS#, HRLYWAGE]
```

If we want to retrieve the name, major and salary/wage of the students who are also staff, we can use the following query:

```
  SELECT PERSON.NAME, STU.MAJOR,
STAFF:STYPE.*
  FROM PERSON, STU, STAFF:STYPE
  WHERE PERSON.SS# = STU.SS# AND
  STU.SS# = STAFF:STYPE.SS#;
```

●

The use of DTTVs provides a powerful means for referencing data in a network of relations. The binding rules for DTTVs permit the user to conveniently retrieve attributes along "paths" in the network. It should be easy to see that such retrievals are not always possible in SQL, even if image attributes are available (for instance, the different relations may not always have common attributes - example 7 illustrates such a multirelation). At the same time, the language still allows multiple-variable queries where the different variables are not hierarchically related, thus supporting the type of multiple variable queries used in SQL itself.

3 Implementation Technique

A subset of ESQL has been implemented already. The subset does not support nested queries in which the subquery contains references to multirelations. The goal of our implementation is two-fold: to implement ESQL on top of an existing SQL (any variant of SQL) system, and to study the complexity of mapping ESQL queries to SQL queries. We do not address performance optimization techniques for special cases, but rather consider optimization techniques that are applicable in general cases.

The prototype system for ESQL based query processing consists of a query preprocessor and an SQL-based DBMS (XDB [XDB88]). The user issues ESQL statements to the ESQL preprocessor. If the statement does not reference any multirelation it is passed intact to the SQL processor in XDB. The output from the SQL processor is displayed to the user. If the ESQL query references multirelations, then the ESQL preprocessor transforms it into a sequence of calls to the SQL system. The results from these calls are combined and displayed to the user by the preprocessor. The algorithm used by the preprocessor to evaluate an ESQL query is given in [Ahad88]. Here we give a summary of the functions performed by the preprocessor.

Given an ESQL query Q, the preprocessor starts by gathering all the DTTVs mentioned in Q. These include 'implicit' DTTVs (these are DTTVs that are used to process the query, but are not stated explicitly in the query statement). Once the DTTVs are determined, the preprocessor searches for a 'meta binding' for each DTTV; i.e., assigning a relation name to a DTTV. Whenever all the DTTVs are assigned relations such that the relations have all the required attributes, the preprocessor generates an SQL query. The SQL query is obtained by modifying the ESQL query as follows:

1. All unavailable attributes in the attribute list of the SELECT clause are dropped;
2. The DTTVs in the FROM clause are replaced by the relations currently bound to those DTTVs;
3. Additional selection conditions on some relations are included in the WHERE clause.

The resulting SQL query is sent to the relational DBMS for evaluation.

The above procedure requires the preprocessor to issue SQL queries to obtain the projection on image domains during the process of 'meta binding'. For each relation name thus obtained, the preprocessor checks to see if the required attributes are present in the relation. If not, the relation is discarded. This feature of the preprocessor is an extremely important one. A naive instantiation of all DTTVs in a query could generate significantly larger sets of relations when the corresponding multirelations are large (i.e. when a DTTV such as $A : B$ defines a large set of relations in $A[B]$). This in turn would generate a large number of SQL queries, most of which might fail because attributes mentioned in them are not in the referenced relations. By identifying and eliminating such relations from the query *before* generation of the SQL queries, the complexity of the ESQL query can be significantly improved.

4 Related Work

The deficiencies of the relational model in dealing with advanced applications have been argued extensively, and a number of extensions have been proposed in the literature. Kent pointed out the many shortcomings and some remedies for record-based models in [Kent79]. The approaches to extend the relational data model to deal with these problems involve extending the domain data type, and they can be divided into two categories. In the first category both the data structure of the relational data model and the operations on it are extended. Non-first-normal-form (NFNF) relations [Abit84, Sche86] and Abstract Data Type INGRES [Ston83] are the approaches in this category. These approaches require a new storage structure and query language. Thus substantial investment in both DBMS development (see [Dada86] for a system development effort for an NFNF relation approach) and database conversion is needed. Furthermore, many-to-many relationships among tuples cannot be conveniently modeled in these approaches. This implies that sharing of nested data between multiple relations is difficult. Insertion and deletion of tuples or updates are also very tedious.

In the second category are the approaches that retain the normalized relation structure of the relational data model and enhance its modeling power by introducing special domains. The query language of the relational model is extended to exploit the semantics of special domains. The approaches in this category are the image domain concept of Smith and Smith [Smit77], "Quel as a data type" [Ston84], GEM [Tsur84], and the extension to System R to support complex objects [Lori83].

In [Smit77] the image domain concept is used to model generalization/specialization hierarchies. However, a query language that exploits the semantics of the image domain is lacking. Image domains can also be used to model a network of relations in which a tuple of one relation may be related to sets of tuples of other relations or to entire relations.

The concept of complex object has been introduced into System R to manage engineering design data more effectively. A complex object is a set of tuples from different relations that should be treated as a single object. Lorie's approach [Lori83] to represent complex objects requires no change to the storage structure to the relational model. However he introduces three new domains: IDENTIFIER, COMPONENT_OF and REFERENCE all of which are tuple identifiers. Since the underlying structure of the relational model is not changed, The relational query language SQL can be used to query complex objects.

The work that is most closely related to ours is "QUEL as a data type" [Ston84], which we will refer to as QUEL+. QUEL+ extends the modeling power of the relation model by supporting QUEL queries as valid attribute values. Such an attribute will be referred to as a QUEL attribute. When tuples are inserted into a relation, the values of all QUEL attributes are QUEL statements. Conceptually, a QUEL attribute can be thought of as a set of tuples which may themselves contain QUEL attributes. The tuples referred to by a QUEL attribute value are retrieved when the attribute is referenced in a query. QUEL+ extends QUEL by introducing features in QUEL to query tuples retrieved by reference to a QUEL attribute in a relation.

We note that the image domain is a special case of QUEL+. For example, the relation R which is an element of an image domain is equivalent to the QUEL statement "range of r is R retrieve (r.all)" in QUEL+. Thus QUEL+ is more powerful in the sense that a subset of tuples of a relation can be specified as an attribute value of a tuple. The price for this added power is that QUEL+ queries can only be executed a tuple at a time and the performance improvement techniques such as those discussed in section 3.0 cannot be used. Furthermore, in many applications, the QUEL statements that appear as attribute values of tuples of relation S are of the form "range of r is R retrieve (r.all) where r.id = tid" where tid is the tuple's identifier. With image domains, the corresponding attribute value will be R and the relationship between tuples of S and R is established in a query by a simple join condition such as S.id = R.id. For such simple situations, the added complexity of QUEL+ is unnecessary. Insertion of tuples containing QUEL attribute is also very tedious [Kemp87], since with each insertion of a tuple, a QUEL statement for each of its QUEL attributes must be specified as well.

The query language for QUEL+ is an extension of the query language QUEL [Ston76]. However, this language lacks some facilities needed to effectively exploit its rich domain type. For example, consider relations $R[A, B]$, $S[C, D]$, and $T[E, F]$ where B is a QUEL attribute. Suppose that the B attribute values of some tuples of R are of the form "range of s is S retrieve (s.all) where s.C = id", and other tuples are of the form "range of t is T retrieve (t.all) where t.C = id". Now suppose we want to print the attributes of S and T relations for some R tuples such that S.D $>$ 0 for S and T.F $>$ 0 for T. The ideal query should be

```
range of r is R
retrieve (r.all)
where r.B.D $>$ 0 or r.B.F $>$ 0
```

However this query will either cause an error condition or will always return an empty set since QUEL+ will look for attributes D and F in the set of tuples obtained by executing the QUEL query in attribute B of a tuple of R. But both attributes can never be found for one tuple of R. This problem would not occur with ESQL since it requires the relations have attributes to satisfy at least one conjunct (rather than all conjuncts).

There are some additional differences between ESQL and QUEL+. For instance, image attributes in ESQL can also be treated as simple character string attributes. This allows us to pose queries in which information represented by these strings can be used directly, in addition to queries where the image attribute is used to access the member relations. Also, since ESQL query results are also multirelations (where relevant), it is easy to add such results to the database. It is not clear how this would be achieved in QUEL+. Finally, we have explicitly addressed the issue of naming and referencing multirelations, thereby enabling reference to whole multirelations as well as to any component relations within them.

We recognize that the relational model is not the only way to deal with the data modeling requirements of advanced applications. For instance, object-oriented data models provide elegant means for supporting these modeling requirements [ACM87]. However, the use of the relational model is still a desirable approach for several reasons. First, it has a well-understood formal basis that facilitates effective database design and query processing. Furthermore, relational database technology is well established, and the extensions that we have made to the relational model benefit from this *state-of-the-art*. By contrast, the theory and practice of object-oriented database technology is still in its infancy. Yet another reason for extending the relational model is that relational DBMS are today the dominant choice for most database applications, and this situation is likely to continue. Consequently, extensions such as ESQL will enable users who have invested in relational technology to attain the additional functionality needed for newer applications.

5 Conclusions

The design and implementation of ESQL is motivated by our need to pose queries against a database used for enhanced explanation support in knowledge-based systems [Basu88]. This database shares several structural properties with databases for advanced applications like CAD/CAM, and semantic databases. Thus ESQL will be useful for these applications as well. Also, preliminary studies show that ESQL can also be used to access multiple databases in a multidatabase system [Litw86], and we intend to pursue research in this direction.

As mentioned in the introduction, our approach is motivated by the inadequacies of the traditional relational data model, as well as by a desire to achieve the necessary functionalities with a minimal extension of existing relational technology. A noteworthy observation is that we are able to incorporate multirelations without *any* modifications to the storage structures and data definition language of SQL. Furthermore, we are able to support queries on multirelations in ESQL without modifying the features of SQL in any way as far as traditional data manipulation is concerned. This decision to minimize the extensions to the relational model results in some limitations. For instance, by not introducing new DDL features such as an *image* data type, we cannot support referential integrity checks on image attributes.

Our prototype system demonstrates that ESQL can be implemented using a query preprocessor on top of an existing SQL based system. Our major concern in terms of implementation has been the testing of the functionality of such a loosely coupled system (i.e. a preprocessor interacting with an SQL DBMS). In fact, it is not difficult to see that an alternative design which involves tighter coupling between the 'multirelation' and 'relation' levels has significant merits. Some significant research problems that are currently under study include these other optimization techniques of query processing in ESQL, and the implications of multirelations on the physical design of the underlying relational database. Another feature that is desirable in any query language for advanced applications is support of recursive queries. Although ESQL does not have this capability yet, we have identified several types of recursion that are relevant to multirelations, and are currently developing additional binding rules for preprocessing recursive ESQL queries.

6 References

[Abit84] Abiteboul, S. and N. Bidoit, "Non First Normal Form Relations to Represent Hierarchically Organized Data", Proc. of Symposium on Principles of Database Systems, 1984.

[ACM87] "Special Issue on Object-Oriented Systems", ACM Transactions on Office Information Systems, January 1987.

[Ahad88] Ahad, R. and A. Basu, "Modeling and Querying Networks of Relations", Information Systems Technical Report No. TR-88-39, University of Maryland, College Park, May 1988 (submitted to ACM Transactions on Database Systems).

[Ahad89] Ahad, R. and A. Basu, "ESQL: A Query Language for Networks of Relations", Information Systems Working Paper, University of Maryland, College Park, May 1989.

[Basu88] Basu, A. and R. Ahad, "Using a Relational Database to Support Explanation in a Knowledge Based System", Information Systems Technical Report No. TR-88-40, University of Maryland, College Park, August 1988 (submitted to IEEE Trans. on Knowledge and data Engineering).

[Cham74] Chamberlain, D.D. and R.F. Boyce, "SEQUEL: A Structured English Query Language", Proc. 1974 ACM Workshop on Data Description, Access and Control, May 1974.

[Codd79] Codd, E.F., "Extending the Database Relational Model to Capture More Meaning", ACM Trans. on Database Systems, vol 4, no. 4, 1979.

[Dada86] Dadam, P. et., al., " A DBMS Prototype to Support Extended NF2 Relations: An Integrated View of Flat Tables and Hierarchies", Proc. ACM SIGMOD Int. Conf. on Management of Data, May, 1986.

[Kemp87] Kemper, A. and M. Wallrath, "An Analysis of Geometric Modeling in Database Systems", ACM Computing Surveys, 19:1, March 1987.

[Kent79] Kent, W., "Limitations of Record-Based Information Models", ACM Transactions on Database Systems, 4:1, March 1979.

[Klau85] Klausner, A. and N. Goodman, "Multirelations: Semantics and Languages", Proc. of the International Conference on Very Large Data Bases, Stockholm, Aug. 1985.

[Litw86] Litwin, W. and A. Abdellatif, "Multidatabase Interoperability", IEEE Computer, 19:2, December 1986.

[Lori83] Lorie, R. and Plouffe, W., "Complex Objects and Their Use in Design Transactions", Proceedings of the 1983 Database Week: Engineering Design Applications, 1983.

[Sche86] Schek, H. and Scholl, M. "The Relational Model with Relation-Valued Attributes", Information Systems, 1986.

[Smit77] Smith, J.M. and D.C.P. Smith, "Database Abstractions: Aggregation and Generalization", ACM Transactions on Database Systems, 2:2, June 1977.

[Ston76] Stonebraker, M., et al, "The Design and Implementation of INGRES", ACM Transactions on Database Systems, 1:3, September 1976.

[Ston83] Stonebraker, M., et. al., "Application of Abstract Data Types and Abstract Indices to CAD Databases", Proceedings of the 1983 Database Week: Engineering Design Applications, 1983.

[Ston84] Stonebraker, M., et al, "QUEL as a datatype", Proc. of ACM SIGMOD Int. Conf. on Management of Data, Boston, June 1984.

[Tsur84] Tsur, S., and C. Zaniolo, "An Implementation of GEM - Supporting a Semantic Data Model on a Relational Back-End", Proc. of ACM SIGMOD Int. Conf. on Management of Data, Boston, June 1984.

[XDB88] XDB User Manual, XDB Systems, Inc., College Park, MD, 1988

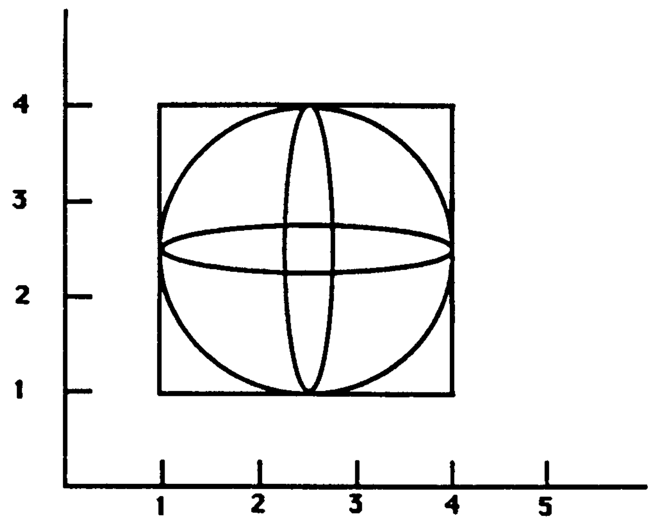

Figure 1. An Example Picture

PICTURE

X	Y	PRIMITIVE	TID
2.5	2.5	CIRCLE	1
2.5	2.5	ELLIPSE	1
2.5	2.5	ELLIPSE	2
1.0	1.0	POLYGON	1

POLYGON

TID	SNO	X	Y
1	1	1.0	1.0
1	2	4.0	1.0
1	3	4.0	4.0
1	4	1.0	4.0
1	5	1.0	1.0

CIRCLE

TID	RADIUS
1	1.5

ELLIPSE

TID	MAJOR	MINOR	ANGLE
1	1.5	0.4	0
2	1.5	0.4	90

Figure 2. Relation Extensions for Figure 1.

$PICTURE$CIRCLE

TID	RADIUS	X	Y
1	1.5	2.5	2.5

$PICTURE$ELLIPSE

TID	MAJOR	MINOR	ANGLE	X	Y
1	1.5	0.4	0	2.5	2.5
2	1.5	0.4	90	2.5	2.5

Figure 3. Display of Result of ESQL Query in Example 5.