SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 18(3), 169–203 (MARCH 1988)

PICASSO: A Graphical Query Language

HYOUNG-JOO KIM†, HENRY F. KORTH† AND AVI SILBERSCHATZ‡
Department of Computer Sciences, The University of Texas at Austin, Austin, Texas 78712, U.S.A.

SUMMARY

PICASSO (*PICture Aided Sophisticated Sketch Of* database queries) is a graphics-based database query language designed for use with a universal relation database system. The primary objective of PICASSO is ease of use. Graphics are used to provide a simple method of expressing queries and to provide visual feedback to the user about the system's interpretation of the query. Inexperienced users can use the graphical feedback to aid them in formulating queries whereas experienced users can ignore the feedback. Inexperienced users can pose queries without knowing the details of underlying database schema and without learning the formal syntax of SQL-like query language. This paper presents the syntax of PICASSO queries and compares PICASSO queries with similar queries in standard relational query languages. Comparisons are also made with System/U, a non-graphical universal relation system on which PICASSO is based. The hypergraph semantics of the universal relation are used as the foundation for PICASSO and their integration with a graphical workstation enhances the usability of database systems.

KEY WORDS Graphical query languages Graphical interfaces Universal relation data model Hypergraphs Database management

INTRODUCTION

Recent research on database systems has concentrated on making database systems easier to use. Among the major data models, network, relational and hierarchical, the relational data model, and in particular *the universal relation model*,[1] is considered to be the most user-friendly. This is due mainly to the following reasons:

1. In a relational database system, users do not have to know about the physical structure of the database (physical data independence).
2. Most query languages for database systems based on the relational data model are non-procedural. In general, non-procedural query languages are easier to use and learn than procedural query languages.
3. In a universal relation database system, users have to be concerned neither about the physical structure of the database, nor about the logical structure of the database (logical data independence).

† Research partially supported by NSF Grant DCR–8507724 and IBM Faculty Development Award.
‡ Research partially supported by NSF Grant DCR–8507724 and ONR grant N00014–80–K–0987.

0038–0644/88/030169–35$17.50

Received 21 November 1985
Revised 3 November 1986

Thus, a universal relational database system can be thought of as taking a step beyond conventional relational database systems towards user friendliness. Since users see only one relation (the universal relation), they do not have to be concerned about which attributes are in which relations. The database management system determines which relations of the underlying database need to be accessed in order to respond to a given query (i.e. the universal relation provides a convenient abstraction, a virtual universal view). Since tuple variables are assumed to be bound to the universal relation, it is not necessary to declare relations to which tuple variables are bound. More specifically, since the system determines which relations of the underlying database need to be joined in order to respond to a given query, many join operations can be hidden by a universal relation. Thus, the syntactic improvement of a universal relational query language comes from eliminating the clause for binding tuple variables to relations such as the *range-of* clause in QUEL[2] and the *from* clause in SQL[3] and also the elimination of the need to specify the *join* predicates.

Recently, workstations and intelligent terminals have become prevalent. Workstations with bit-mapped displays, pointing devices (such as a mouse or a light pen), icons, multiple windows, pop-up menus and electronic forms allow users to point directly to objects on the screen and manipulate them in a two-dimensional graphical manner (we call screen-oriented interactions between a user and a computer 'two-dimensional'). The availability of advanced graphics facilities has stimulated research on graphical interfaces for various computer applications.

The motivation behind research on graphical interfaces for database systems is as follows. Existing high level query languages (e.g. SQL, QUEL, etc.) are difficult for naïve users to use and understand.[4–6] Furthermore, naïve end users and non-expert users usually have a long learning period with conventional query languages. They also have to remember many details, such as the names of relations and attributes in a database schema. Further, they are not familiar with mathematical concepts such as predicate calculus, relational algebra and set theory. Naïve users find it difficult to understand the notions of multiple tuple variables, join operations, and nested-type queries. The above problems become worse if the database has a large and complex schema.

Many new graphical interfaces and query languages have been invented to bridge the gap between naïve users and database systems. The first successful example was IBM's QBE (Query By Example).[7] QBE displays table skeletons that are the column headings of relations. QBE users express queries by inserting 'examples' in these skeletons. The resulting language is considered more user friendly than most other relational query languages, including SQL and QUEL. In Reference 8, we surveyed several graphical query languages and interfaces: QBE,[7] OBE,[9] PBE,[10] DBE,[11] ABE,[12] CUPID,[13] GQL/ER,[14] IQL,[4] SDMS,[15] GUIDE,[6] HPS.[16]

In that survey, we made several important observations. The first observation is that many existing graphical interfaces work only for relatively simple queries and simple database schemas. When the underlying database schema is complex and consists of many attributes, most existing systems could not suggest reasonable solutions. Since the size of the screen is limited, the above problem is common to most interfaces. The second observation is the lack of a user-friendly relational join operator to connect relations in most systems. Typically, it is the join operator that leads to most of the complexity in a query language. The third observation is that naïve users find it difficult to understand the notions of multiple tuple variables and nested-type queries. In most graphical interfaces, such queries are avoided or allowed only in a restricted form. The fourth observation is that since there is no graphical feedback during the query process, it is very difficult for naïve and non-expert users to formulate a complex query correctly on the first try.

The above observations suggest several essential features for the graphical interfaces of database systems. Other researchers have suggested similar ideas.[4–6, 14]

1. The graphical interface should be able to provide information to the user about the schema of the underlying database.
2. There should be a facility in which the user can formulate queries incrementally. Building a complex query incrementally means that users can pose complicated queries by exploiting the result of previous queries for building predicates of a new query.
3. There should be a facility that allows the user to browse the database freely.
4. Graphical feedback should be provided during query processing to guide the user in the formulation of correct queries.

In accordance with the above guidelines from prior graphical interfaces, we designed and implemented a new graphical query language, PICASSO, on top of System/U, a universal relation database system. Since the System/U query language itself is a QUEL derivative, knowledge about a substantial amount of the formal syntax of QUEL is required in System/U. Using graphics, we can solve many generic problems with relational query languages, including System/U. We investigated how the graphical definitions of the semantics of the universal relation and the availability of a graphical workstation can be combined to enhance the usability of database systems.

The remainder of this paper describes our approach to the design and implementation of PICASSO.

UNIVERSAL RELATION THEORY AND System/U

The System/U database management system[17] is based on the universal relation data model and theory. Universal relation theory has a rigorous mathematical foundation. Because of space limitations, we omit the details of the formal theory and give only some intuitive ideas here. Interested readers can refer to References 1 and 18–21.

The user of a universal relation database system sees the database as a one relation database. This relation is defined as the set of all tuples such that some predicate, P, is true: $\{t|P(t)\}$. The predicate P must pertain to individual tuples if the database is to be a relation. For example, we could have a bank database of banks, accounts and customers represented by a universal relation where the scheme is (bank,account, customer). Then $P(t)$ would be 't.customer has t.account' and 't.account is with t.bank'. Let us write $P(t)$ in conjunctive normal form:

$$P(t) = P_1(t.a_1^1, \ldots, t.a_{n_1}^1) \wedge \ldots \wedge P_k(t.a_1^k, \ldots, t.a_{n_k}^k)$$

where $t.a_j^i$ denotes an attribute of t. Further, we treat each set of attributes $\{t.a_1^i, \ldots, t.a_n^i\}$ as a fundamental relationship, called an *object*. The predicate P may now be written as

$$P(t) = P_1(O_1) \wedge \ldots \wedge P_k(O_k)$$

where the O_is are the objects.

The above appears to be a highly intuitive definition of the semantics of the universal relation. However, the definition turns out to have significant theoretical implications.† Fagin, Mendelzon and Ullman[18] showed that for any universal relation U defined as we have defined it above, the JD (join dependency) $*(O_1, \ldots, O_k)$ holds. Furthermore, they characterized the MVDs (multivalued dependencies) that the above JDs imply. The characterization is best understood by considering the database to be a hypergraph whose nodes are attributes U and whose hypergraphs are the objects. An MVD of the form $X \rightarrow\rightarrow Y$ (where X and Y are disjoint) holds on U if and only if Y is the union of connected components of the hypergraph with nodes in X deleted.

With the above fundamental assumptions, the semantics of the database are expressed using a hypergraph in which nodes are the attributes of the universal relation, and the hyperedges are fundamental relationships (*objects*). A second hypergraph is then formed whose nodes are the objects, and whose hyperedges, called *maximal objects*, represent maximal sets of objects in which queries 'make sense'.[1, 21] The intuitive notion of a *maximal object* is a group of related attributes in which the user can pose System/U queries in a straightforward manner. The approach is to bind each tuple variable over a set of semantically strongly connected objects. This approach avoids cyclic database schemes. Reference 19 describes cyclic and acyclic databases and their consequences. Thus, each maximal object has an acyclic structure and, according to universal database theory, has a join dependency (i.e. the objects in each maximal object have the lossless join property). Given the specification of objects and dependency information of a database, the system is responsible for determining maximal objects. There are several algorithms for creating maximal objects,[19, 21] which are incorporated into the universal relational database system.

System/U consists of two parts: the DDL (Data Definition Language) step and the DML (Data Manipulation Language) step. The database administrator (DBA) or user prepares input for the System/U DDL. The input for the DDL step is a specification of the attributes and their data types, the relation schemes and the names of relations, the objects and any functional dependencies that hold. Given a DDL input, the system creates the list of maximal objects and the set of objects that belong to each maximal object. The database designer can change the contents of a maximal object to impose different semantics on the maximal object. With the list of maximal objects, the System/U DML processor determines which relations of the underlying database need to be accessed in order to respond to a given query and processes the query. A complete description of the System/U query interpretation is given in Reference 17. Using the notion of maximal objects, the intuitive version of System/U query interpretation is as follows: Run the query in each maximal object that contains all attributes mentioned in the query, then take the union of the results.

SYSTEM STRUCTURE OF PICASSO ENVIRONMENT

PICASSO is a graphical database manipulation language which is being used in a graphical user interface for System/U, called ROGUE (*RO*si's *G*raphical *U*ser *E*nvironment). PICASSO and ROGUE have been implemented within the ROSI (*R*elational

† We assume that the reader has the fundamental concepts of the relational database theory such as functional dependency, multivalued dependency, join dependency, etc. Reference 22 is a good introduction to those concepts.

*O*perating *S*ystem *I*nterface) project at the University of Texas at Austin.[23]

Figure 1 shows the PICASSO/ROGUE environment. ERIS is an experimental database system that was implemented at Brown University. ERIS is used as an underlying database system for System/U. ERIS and System/U are written in C under UNIX. We implemented ROGUE and ANSWERTOOL in C, using the Sun Window graphics system on the Sun workstation under UNIX. The major function of ROGUE is translation of a PICASSO query to a System/U query. Also, ROGUE supports facilities to assist the user in formulating graphical queries. The details of ROGUE are presented in Reference 24. ANSWERTOOL is a browser for the results of previous queries. The user can browse the result of a query using ANSWERTOOL and use the constant values of a previous query result for formulating the predicates of another query. Thus, the user can build complex queries incrementally.

In the PICASSO/ROGUE environment, maximal objects are represented as hyperedges in a hypergraph. We believe that the graphical representation of a database schema using hypergraphs provides the user with valuable information regarding the semantics of the database. As an example, consider a bank database consisting of four relations: rofficer(L_officer, Bank), rteller(Teller, Bank), rloan(Customer, Bank, Loan_no, Amount), racct(Customer, Bank, Account_no, Balance). Using the definition of maximal objects of Reference 1, the bank database consists of two maximal objects: MaxObj1 = (Bank, Loan_no, Amount, L_officer, Customer), MaxObj2 = (Bank, Account_No, Balance, Teller, Customer). The system draws a hypergraph representing the maximal objects on the screen as shown in Figure 2.

Query formulation is done directly on hypergraphs using the mouse. This is accomplished by clicking mouse buttons and choosing from a pop-up menu. ROGUE

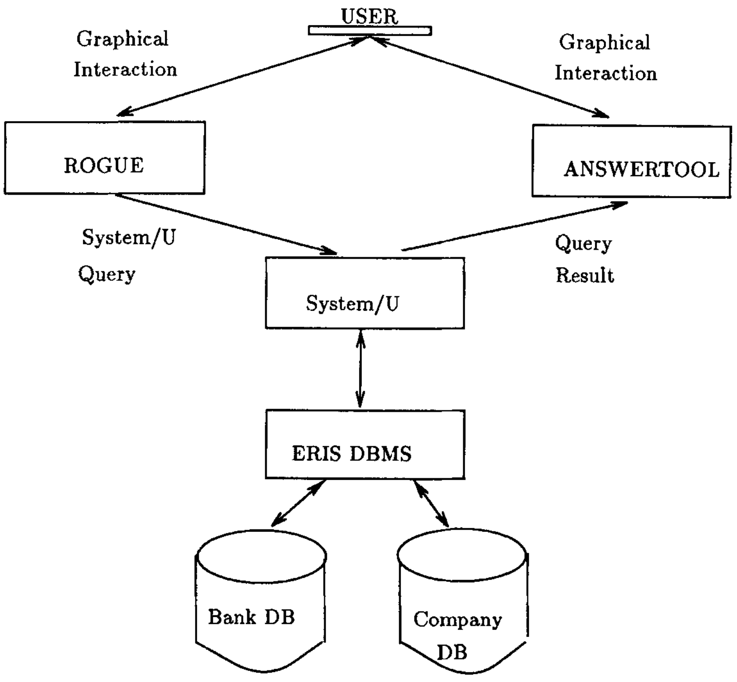


Figure 1. The system diagram of the PICASSO/ROGUE environment

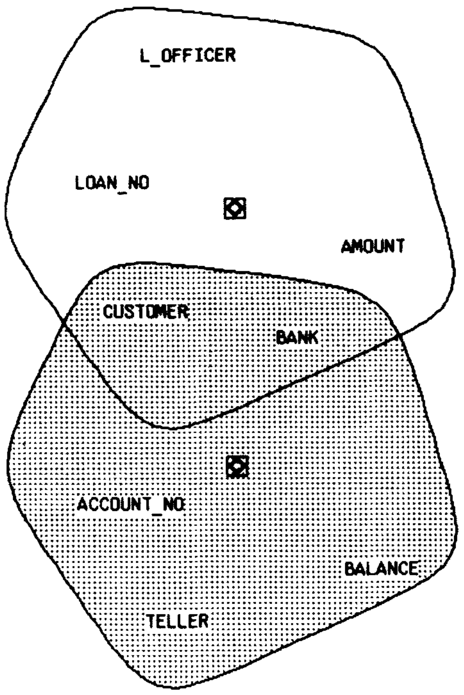


Figure 2. Hypergraph representation of bank database schema

provides HELP messages and other facilities for assisting users. Even though the underlying DBMS is System/U, the user does not have to learn the formal syntax of the System/U query language.

A query session of the PICASSO environment goes roughly as follows. First, the user specifies a database of interest, and the system draws the hypergraphs of that database. Upon the hypergraphs, the user formulates a query using mouse and pop-up menus. After the user formulates a PICASSO query, the user asks the system to run the query by selecting the 'run' command from a pop-up menu. Then ROGUE translates the graphical query into a System/U query, and sends it to System/U. After executing the query, System/U sends the result of the query to ANSWERTOOL. The user can browse the result relation and use the constant values of the result relation for formulating predicates in future queries. Several result relations can be maintained within ANSWERTOOL at one time. The user can even move into another database and pose queries using the result relations of the previous database.

We show the expressibility of PICASSO queries through various examples in the next sections. The example queries of PICASSO queries demonstrate the power of graphics to provide a simple method of stating what would be a complex query if a traditional query language were used.

OVERALL STRUCTURE OF ROGUE

ROGUE translates PICASSO queries to the System/U query language and provides facilities to help users in posing PICASSO queries. ROGUE is implemented as a *graphic tool* in the SunWindow system. Figure 3 shows a ROGUE window that consists

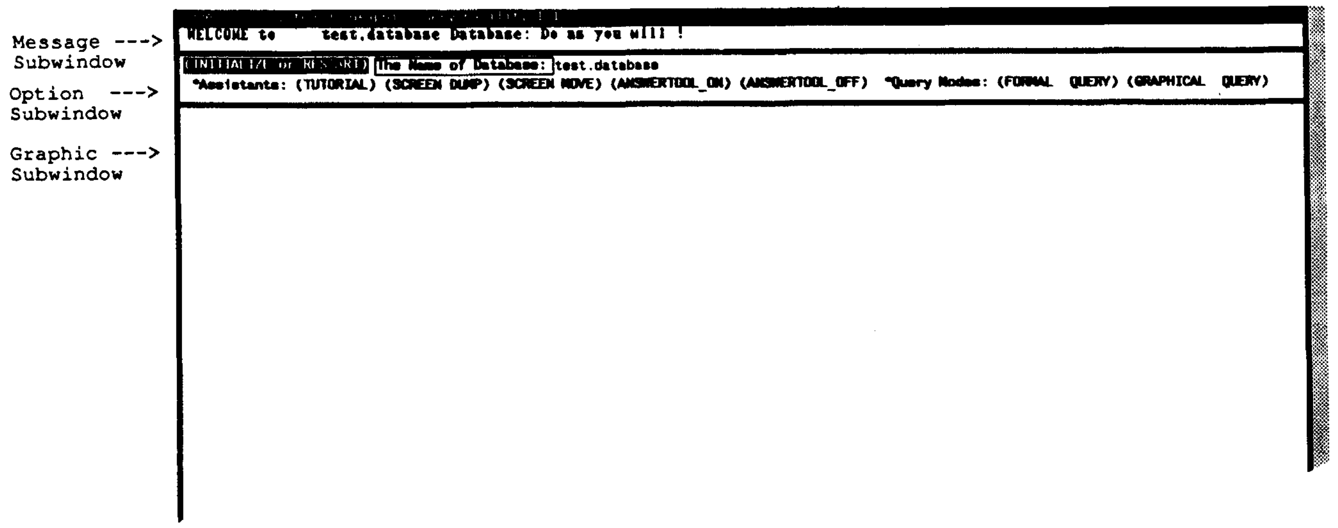

Message ---> Subwindow
Option ---> Subwindow
Graphic ---> Subwindow
WELCOME to test.database Database: Do as you will !
(INITIALIZE or RESTART)
The Name of Database: test.database
*Assistants: (TUTORIAL) (SCREEN DUMP) (SCREEN MOVE) (ANSWERTOOL_ON) (ANSWERTOOL_OFF)
*Query Modes: (FORMAL QUERY) (GRAPHICAL QUERY)

Figure 3. Overall structure of a ROGUE window

of three subwindows: a message subwindow, an option subwindow and a graphic subwindow.

Message subwindow

The message subwindow indicates the current mode (there are several modes in query processing such as select-attribute mode, predicate-formulation mode, undo mode, formal-query mode, etc). A series of messages appear according to the various modes of query processing. Each message is a warning/error message or a guidance to the user to formulate correct queries.

Option subwindow

There are two types of commands in a graphic tool of the SunWindow system: options in the option window and menu items of pop-up menus. Options usually handle changes of context (mode). A pop-up menu displays a set of commands that are available in a given context. The contents of pop-up menus can differ based on selection options. ROGUE supports eight option items in the option subwindow:

1. Initialize reads a database schema and constructs the hypergraph representation.
2. Tutorial provides the user with a brief introduction to ROGUE, some examples of PICASSO queries, and explains the functions of mouse buttons, pop-up menus, and options.
3. Screendump prints a copy of the screen contents on a laser printer.
4. Screenmove allows the user to navigate through a database schema. If the database schema has so many attributes that the hypergraphs for the database schema do not fit on the screen, the user can move the hypergraphs on the screen using the SCREENMOVE command.
5. Answertool_on invokes ANSWERTOOL (creates ANSWERTOOL as a child process). The communication channel between ANSWERTOOL and ROGUE is set up.
6. Answertool_off kills the child process ANSWERTOOL. The ANSWERTOOL window disappears.

The system supports two query modes:

1. *Graphical query*: is the default mode when the user starts the session. Queries are expressed using PICASSO.
2. *Formal query*: allows the user to pose queries using the System/U query language. Inexperienced users would prefer the graphical query mode, whereas experienced users might want to pose formal (System/U) queries. The speed of data retrieval in the formal query mode is faster than that in the graphical mode.

Graphic subwindow

The graphic subwindow is for the hypergraph representation of a database schema and the graphical representation of a PICASSO query.

QUERY FACILITIES

When users formulate queries, they must specify the attributes in which they are interested, and must give the predicates for the selected attributes. The select clause contains the attributes and the where clause contains the predicates. We assume a three-button mouse for our graphic interface. The *left button* of the mouse is used for the select clause, the *middle button* is used for the where clause and the *right button* is for the *basic* menu of query processing. No constraints are imposed on the order in which parts of the query are entered.

We believe that the human thought process about query formulation consists basically of the following three steps:

Step 1: select on or more attributes in which the user is interested.
Step 2: describe the predicates for the selected attributes.
Step 3: if one wishes to know about other attributes then go to step 1; else return;

These steps are emulated with the three mouse buttons. Selection attributes may be added after some predicates have been specified, etc. PICASSO supports an undo mechanism for modifying a query. Using the undo mechanism, predicates can be removed and selected attributes unselected. The undo mechanism for modifying graphical queries is explained below.

Consider our bank database scheme, and a query 'Display all bank names'. The user clicks the left button of the mouse on the bank attribute. Then the question mark '?' is postfixed after the selected attribute. Figure 4(a) is the graphical query for the above query. The equivalent QBE query is shown in Figure 4(b), and Figure 4(c) shows the equivalent SQL query.

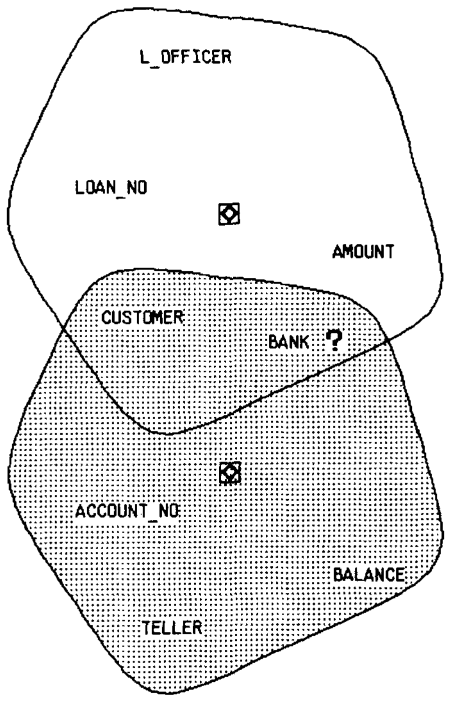


Figure 4(a). PICASSO representation of the query 'Display all bank names'

rloan	Customer	Bank	Loan-No	Amount
		P.		

racct	Customer	Bank	Acct-No	Balance
		P.		

Figure 4(b). QBE representation of the query 'Display all bank names'

```
(SELECT   T.Bank
 FROM     rloan  T
 WHERE * )
UNION
(SELECT   S.Bank
 FROM     racct  S
 WHERE * )
```

Figure 4(c). SQL representation of the query 'Display all bank names'

If the user wants to prefix some aggregate operators, or perform set operations between subqueries, then this can be achieved by a second mouse click on the selected attribute using the left button. The menu for aggregate operators and set operators will pop up. We have chosen to embed the pop-up menu for aggregate operators and set operators in this manner because these operators should be applied to already selected attributes. The menu for aggregate operators and set operators will pop up as in Figure 5. PICASSO offers the following aggregate operators: AVG (average), MIN (minimum), MAX (maximum), CNT (count) and SUM (summation) as System/U does. PICASSO allows set operators such as DIFFERENCE, INTERSECTION and UNION, which are covered below. The user may choose the IGNORE option to make the pop-up menu go away.

After selecting attributes for the select clause, the user presses the middle button of the mouse for describing predicates on desired attributes. Following Fagin *et al.*,[18] we treat the predicate P of the where clause as a conjunction of predicates $P_1, P_2, \ldots, P_n$. Each P_i is expressed separately by the user. We assume that each P_i is one of two forms:

(i) ⟨*attribute*⟩ ⟨*comparison operator*⟩ ⟨*value*⟩
(ii) ⟨*attribute*⟩ ⟨*comparison operator*⟩ ⟨*attribute*⟩

The user formulates predicates by selecting an attribute using the middle button. This attribute is the left operand of the operator. The menu of comparison operators as shown in Figure 6 will pop up. The first six items are comparison operators. The seventh item is for the group by operator and the eighth and ninth items are for set operations. The seventh, eighth and ninth items are explained below. After the user

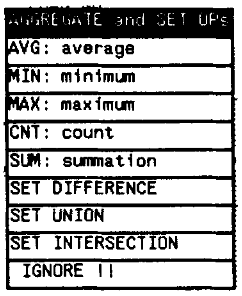


Figure 5. Pop-up menu for aggregate operators and set operators

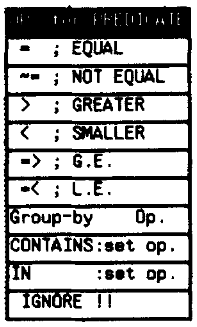


Figure 6. Menu for predicate formulation

selects the comparison operator, a template like the example in Figure 7 pops up. At this point, the right operand could be a constant value or an attribute. If the user clicks an attribute, it indicates comparison with that attribute. If the user types, it indicates that he wants to enter a constant value.

First, consider the case where the right operand is a constant value. The query 'Find the minimum amount on loans at the BOA bank' can be presented in PICASSO as depicted in Figure 8. The select clause MIN: AMOUNT? is formed in the manner described above. In order to formulate the predicate for the where clause, BANK = BOA, the user needs to type the constant value, BOA. In the next section, we show how constant values can be selected from the ANSWERTOOL window, using the mouse rather than the keyboard.

Next, consider the case in which the right operand is an attribute. The middle button is used for selecting this attribute. To illustrate this, consider the query 'Find customers having a loan for an amount greater than that of one of HFK's loans'. As shown in Figure 9, the PICASSO query involves an arrow connection between two AMOUNT attributes. Arrows are used to indicate which attribute is the left or right operand in a predicate. This is necessary for asymmetric comparison operators such as < or >. Intuitively the right part of Figure 9 means 'HFK's loan amount' and the left part is 'Find customers having a loan such that . . .'. The arrow connection asserts

Figure 7. Template for keyboard typing

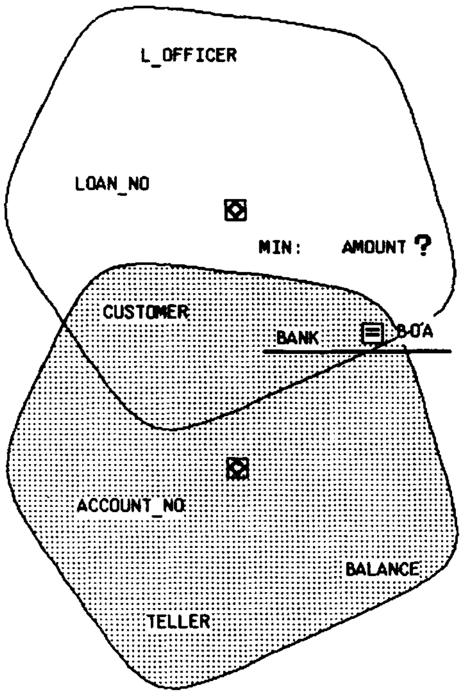


Figure 8. PICASSO representation of the query 'Find the minimum amount on loans at BOA bank'

'the left part is greater than the right part'. Combining the three parts we have the desired query. This query requires the use of two tuple variables as explained below.

Basic menu for query processing

Besides the previously mentioned pop-up menus, there is a basic pop-up menu for query processing as shown in Figure 10. The menu is triggered by clicking the right button of the mouse. The following menu items are available in the basic pop-up menu:

1. LET's GET STARTED draws hypergraphs that represent the underlying universal relation scheme for a database named by the user. Internally, the system executes a hypergraph layout procedure with a given database schema and draws hypergraphs properly. (There are many theoretical problems in representing hypergraphs graphically. Reference 25 illustrates efficient algorithms and heuristics for hypergraph representation problems.)
2. RUN your QUERY means that the user wants to see the result relation after processing a capital query. ROGUE translates the PICASSO query into a System/U query. After System/U processes the query, ROGUE pops up the message in Figure 11. When the user is ready to look at the query result, ANSWERTOOL is invoked.
3. UNDO LAST_ACTION undoes the last action. This undo is idempotent.
4. UNDO FORMULATION can be used when the user wants to edit the query being formulated. As the left button is used for the select clause and the middle button

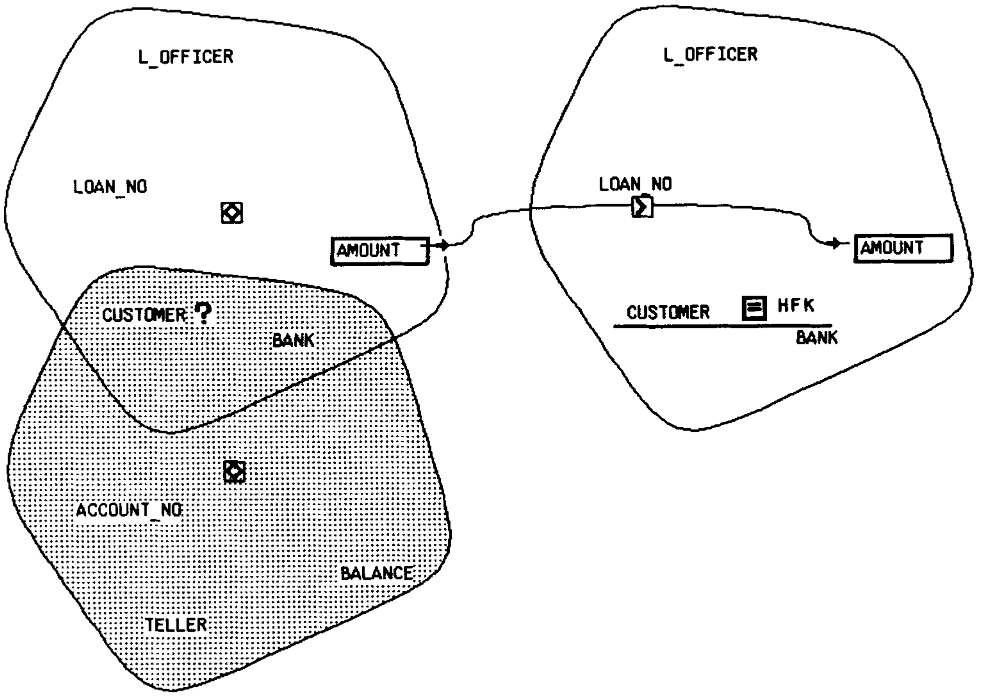


Figure 9. PICASSO representation of the query 'Find a customer having a loan for an amount greater than that of one of HFK's loans'

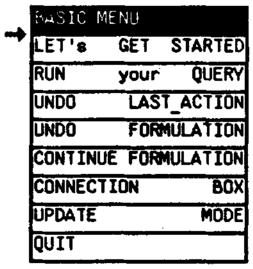
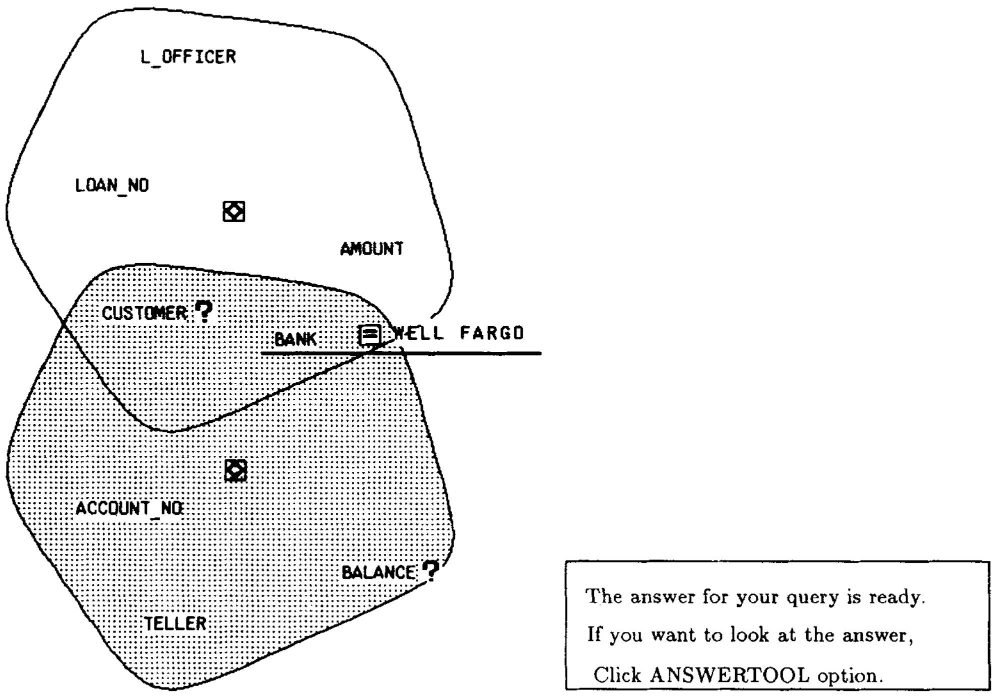


Figure 10. Pop-up menu for basic query processing

is used for the where clause, so the left button is again used for removing the selected attribute and the middle button is used for erasing the formulated predicates in UNDO mode.

5. CONTINUE FORMULATION resumes formulation of queries after partial modification in the UNDO mode. The user can add more attributes or predicates into an existing query.
6. UPDATE MODE: the current implementation of System/U does not allow the user to update a database. Therefore, this part has been left as future work.
7. CONNECTION BOX is used for building up complex predicates (see below).
8. QUIT is for quitting a query session.



The answer for your query is ready.
If you want to look at the answer,
Click ANSWERTOOL option.

Figure 11. The message that informs the user that the query result is ready

Group_By operation

A relation can be partitioned into groups according to the values of some attributes and then predicates can be applied to select only certain groups. This built-in function is called Group_By. Since the Group_By operation returns tuples, not true or false, the function of Group-By operation is different form that of predicates. Despite this, we have chosen to embed Group_Byin the pop-up menu for the middle mouse button. Consider the PICASSO query for 'What is the average balance of each bank?' in Figure 12.

A Group_By operator may be followed by a Having clause in SQL. We also allow use of a Having clause. The Having clause expresses constraints that apply to partitions, rather than tuples. We need a syntax to distinguish a Having clause from a where clause. This is done by a simple dialogue. The dialogue is necessary only in cases in which the query includes a Group_By operator. To illustrate this, consider the query: 'Find the average loan in each bank having the customer HFK'. When the user specifies the Having clause, the dialogue in Figure 13(a) for distinguishing a Having clause pops up. If the user specifies that the predicate is for a Having clause, the predicate is marked as such. The finalized formulation of the query is in figure 13(b).

Multiple constraints upon a single attribute

There are circumstances in which multiple constraints on a single attribute need to be specified. For instance, the predicate 'LOAN_NO between L100 and L10020' requires

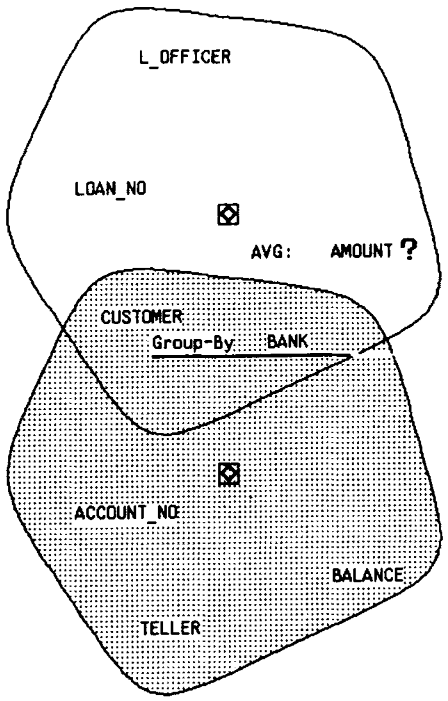
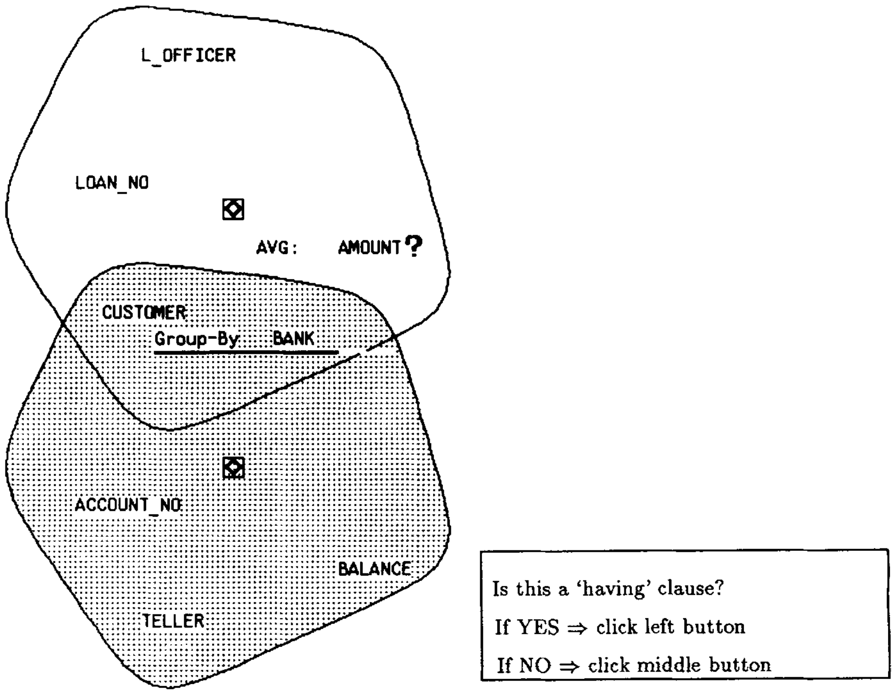


Figure 12. PICASSO representation of the query 'What is the average balance of each bank?'



Is this a 'having' clause?
If YES ⇒ click left button
If NO ⇒ click middle button

Figure 13(a). A simple dialogue for distinguishing a Having *clause*

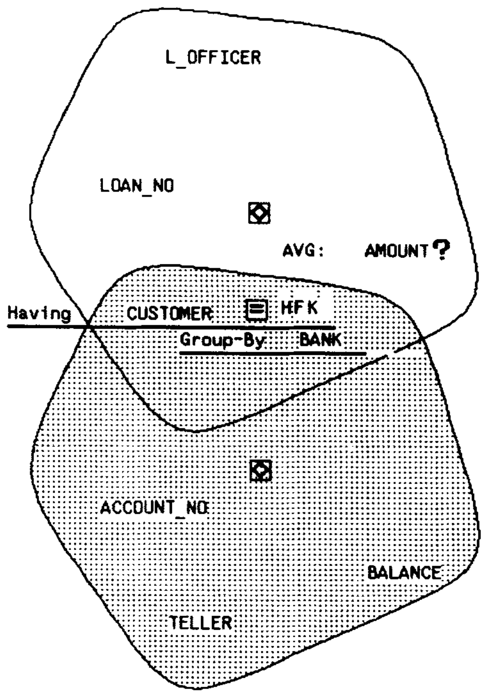


Figure 13(b). PICASSO representation of the query 'Find the average loan in each bank having the customer HFK'

the two constraints 'LOAN_NO > L100' and 'LOAN_NO < L10020'. In PICASSO this predicate is expressed as LOAN_NO = (>L100) (<L10020) as shown in Figure 14. After formulating the first constraint LOAN_NO > L100, the user can click the LOAN_NO attribute again using the middle button and just type the second constraint. The system displays this as shown in Figure 14.

Implicit conjunctive form of predicates

We give preference to conjunctive normal form in our definition of the query language. By default, the specified predicates in the screen are assumed to be connected by AND. If users want to pose disjunctive type queries, they must specify which predicates are connected by OR using the CONNECTION BOX option in the basic menu. The idea of the CONNECTION BOX is from the 'condition box' of QBE. As soon as the user clicks CONNECTION BOX from the basic menu, predicate numbers (p_1, p_2, . . .) appear beside the box of the comparison operator. CONNECTION BOX pops up as shown in Figure 15. Negation of predicates can be entered using the CONNECTION BOX as shown in Figure 16.

Tuple variable creation

In PICASSO, each hyperedge represents a tuple variable. A new tuple variable can be created by opening *the diamond box*, which is located at the centre of each original

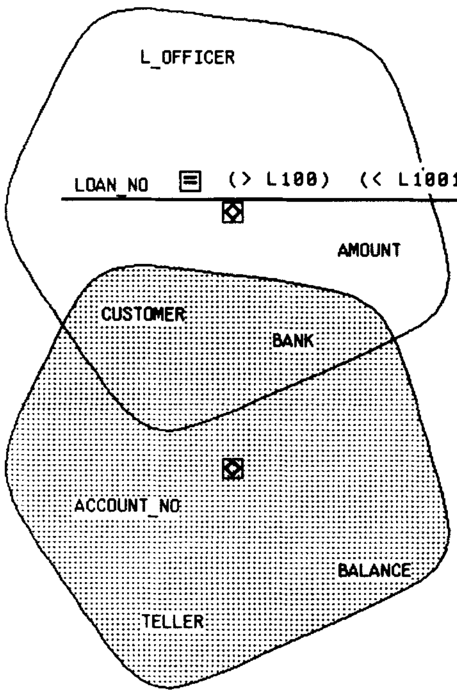


Figure 14. Multiple constraints upon a single attribute

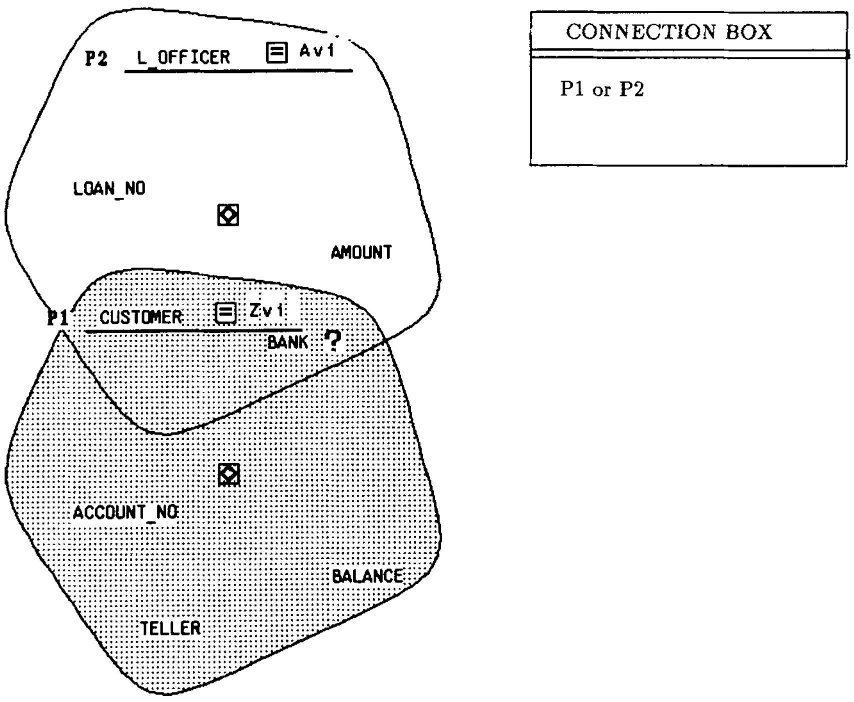


Figure 15. Disjunctive type predicates and the CONNECTION BOX

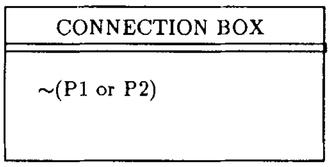


Figure 16. Negation of predicates

hyperedge. This is done by clicking with the middle button of the mouse (clicking the diamond box using the left button results in the selection of all attributes in the hyperedge). Figure 17 shows an example of the creation of a maximal object.

Character-type tuple variables as required in System/U, SQL and QUEL tend to be meaningless from the user's point of view. In general, queries involving multiple tuple variables are difficult to formulate and understand. Notice that this approach (i.e. drawing another hyperedge) eliminates character-type tuple variables completely. System/U accomplished only a partial elimination of tuple variables.

Consider the query in Figure 9 again: 'Find those customers having a loan for an amount greater than that of one of HFK's loans'. Figure 9 demonstrates how PICASSO handles the join-like operator involving two tuple variables. By representing tuple variables as hyperedges, the concept of a tuple variable is reduced to a more intuitive graphical notion.

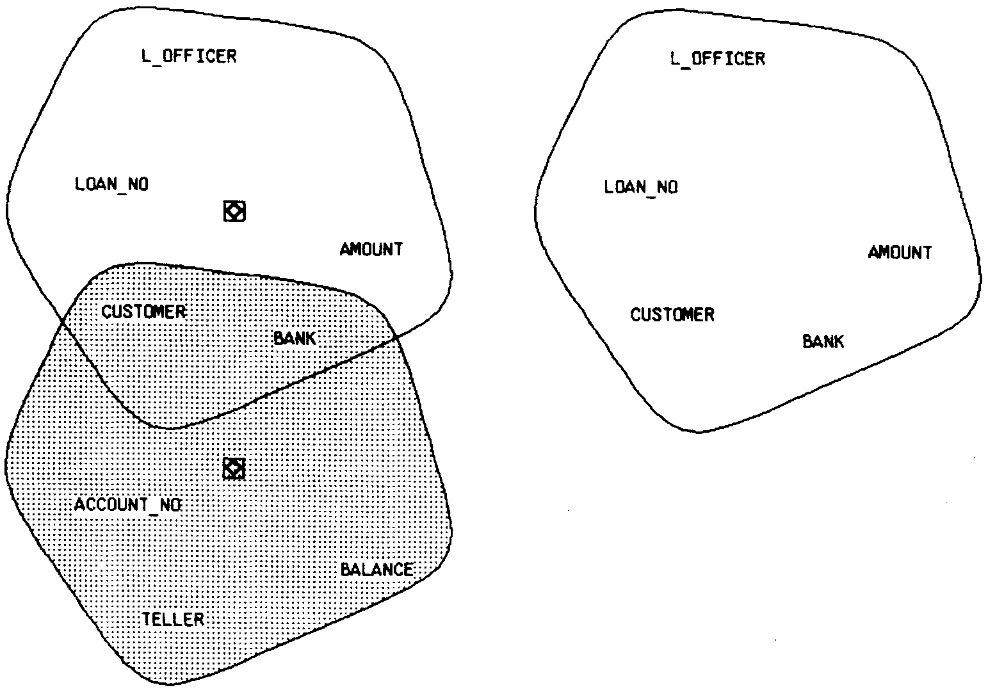


Figure 17. Creating another tuple variable

Set operations

PICASSO includes the set operators such as IN, CONTAINS, SET DIFFERENCE, SET INTERSECTION and SET UNION. Because the operators IN and CONTAINS return a boolean, they are embedded in the pop-up menu for predicate formulation. The three operators, SET DIFFERENCE, SET INTERSECTION and SET UNION, return a set of tuples. They are embedded in the pop-up menu for the select clause. They are represented by a box containing the operator (−, ∪ or ∩) and lines connecting the box to the operands.

Consider the query 'Find customers of BOA who are not customers of FCU'. Figure 18 illustrates how this query is expressed in PICASSO and QBE. We believe the PICASSO query in Figure 18(a) is more intuitive than the QBE query in Figure 18(b).

Consider the *nested type query* 'Find those banks with customers whose balance is more than $1000'. There is no tuple variable (blank tuple variable only) in the System/U-like query for the above query. The nested query is as follows:

```
SELECT Bank
WHERE Customer IN (SELECT Customer
                        WHERE Balance > 1000)
```

However, there are really two tuple variables in the above query, since the inner and outer sub-queries have distinct instances of the 'blank' tuple variables. As shown

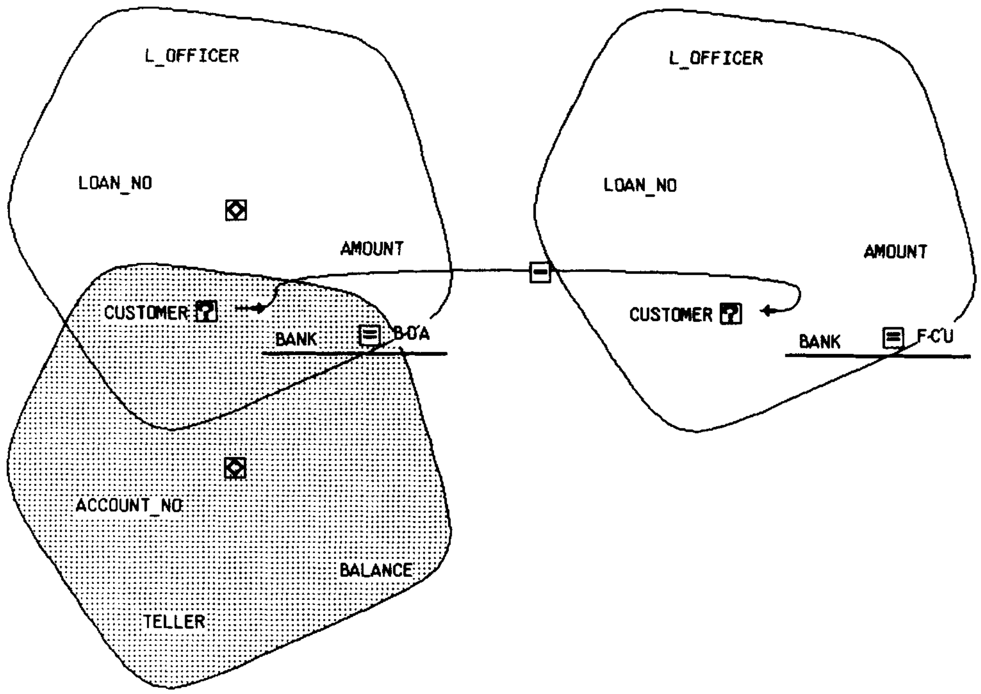


Figure 18(a). PICASSO representation of the query 'Find customers of BOA who are not customers of FCU'

rloan	Customer	Bank	Loan-No	Amount
	C1	BOA		
	C2	FCU		
	P.C3			

racct	Customer	Bank	Acct-No	Balance
	C4	BOA		
	C5	FCU		
	P.C6			

CONDITIONS
C3 = C1 - C2
C6 = C4 - C5

Figure 18(b). QBE representation of the query 'Find customers of BOA who are not customers of FCU'

in Figure 19, the PICASSO query involves a new hyperedge in order to distinguish the inner and outer subqueries.

Generally, most queries involving set operations and nested type queries must have single tuple variables in traditional relational query languages and extra hyperedges in PICASSO. However, as shown in the next section, ANSWERTOOL can be successfully used for building queries involving set operations and nested type queries without the use of extra hyperedges.

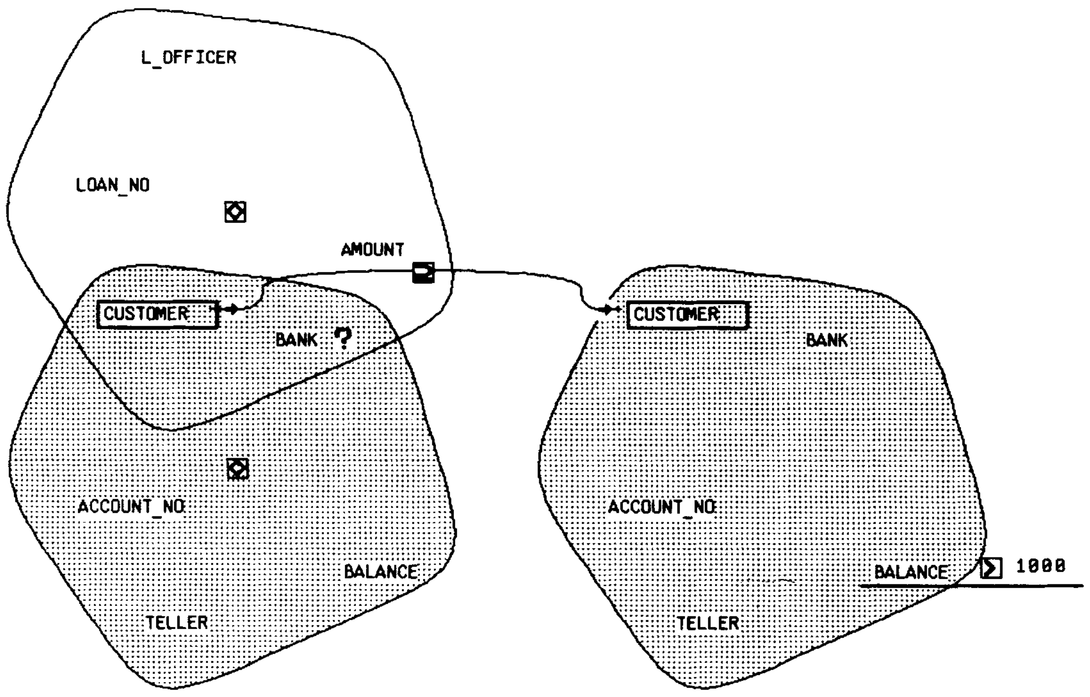


Figure 19. PICASSO representation of the query 'Find the bank containing customers whose balance is more than $1000'

Cartesian product type queries

If the user clicks the diamond box with the left button of mouse, this selects all attributes in that maximal object. The ? icon is postfixed after all attributes in the maximal object. Clicking two or more diamond boxes indicates interest in the content of all attributes that come from the Cartesian product of selected maximal objects. If the selected maximal objects have one or more attributes in common, the system generates the natural join of the maximal objects. Consider the query 'Show the entire Bank universal relation'. The graphical query in Figure 20 is entered by clicking the diamond box in each hyperedge. A straightforward translation of this query to the System/U query language is not possible since no maximal object contains the mention set (the union of the attributes appearing in a select and a where clause). However, ROGUE regards the Cartesian product of maximal objects as a special case. ROGUE generates the correct System/U expression (which requires one tuple variable for each maximal object.

ANSWERTOOL

In previous sections, we introduced the expression of nested queries and queries involving joins and set operations. Even though our approach works well for simple queries, there are some problems in expressing complexing queries involving joins and nested queries graphically (e.g. sometimes the screen becomes cluttered). In this section, we introduce a tool to assist with complex queries, called ANSWERTOOL.

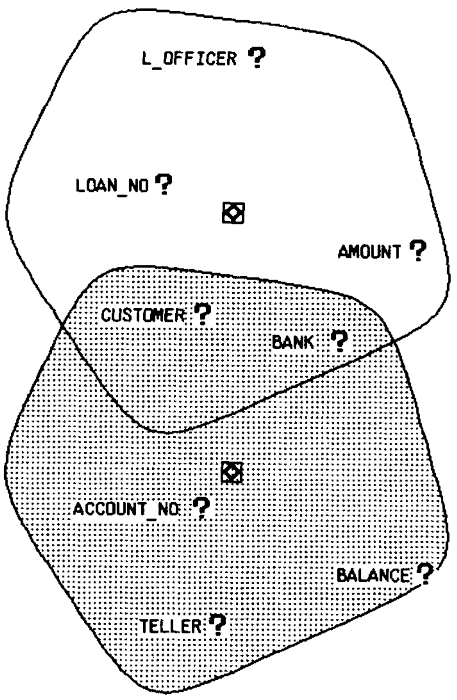


Figure 20. PICASSO representation of the query 'Show all attributes in Bank database'

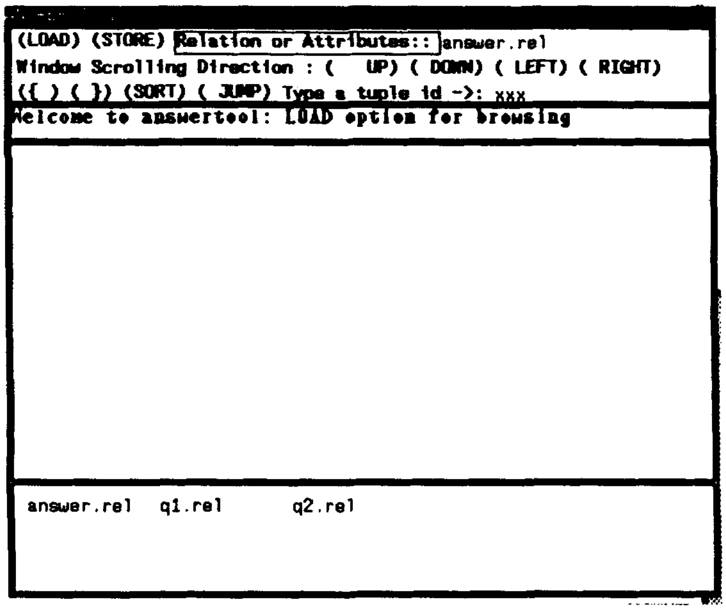
The answer to a query is a relation. The answer relation is displayed in a new window, the ANSWERTOOL window. ANSWERTOOL allows the user to browse result of a query.

Overall structure

The ANSWERTOOL window consists of four subwindows, as shown in Figure 21. The top subwindow is an *option* window. The next subwindow is a *message* window for delivering simple information to the user. The third one is a window for browsing the query result. The bottom one is a directory window for showing the list of temporary relations.

ANSWERTOOL supports several options:

1. Store and Load: we can create temporary relations through ANSWERTOOL. The Store option is used to save the result of a query in a temporary relation. Temporary relations can be accessed using Load option.
2. Scroll up, Scroll down, Scroll left and Scroll right: are for screen rolling. Scroll left and Scroll right are necessary if a relation has tuples too wide to fit in the window. Whenever the user clicks the scroll up (down) option, the next ten tuples are to scrolled up (down). Whenever the user clicks the scroll left (right) option, the window is shifted to the left (right) by one attribute.
3. Open (Closed)_bracket: sends the open (closed) bracket to ROGUE for formulation of a predicate in case that the right hand side of a predicate is a set.
4. Sort by a certain attribute: sorts all tuples in a temporary relation on the specified attribute.
5. Jump to a certain tuple_id: jumps to the part of the relation that starts with the tuple_id entered by the user.


```
(LOAD) (STORE) Relation or Attributes:: answer.rel
Window Scrolling Direction : (   UP) ( DOWN) ( LEFT) ( RIGHT)
({ ) ( }) (SORT) ( JUMP) Type a tuple id ->: xxx
Welcome to answertool: LOAD option for browsing

answer.rel   q1.rel       q2.rel
```

Figure 21. ANSWERTOOL window

ANSWERTOOL does not support any relational operators and is simply a relational browser. The bottom window of ANSWERTOOL is used for showing names of temporary relations. Several answer relations may be maintained at one time as shown in Figure 21.

Towards a joinless, tuple-variableless, nestingless query language

It is difficult to represent queries involving join or set expressions or nested type queries graphically. For example, if there is not enough space for drawing another hyperedge in Figure 18(a), a new hypergraph may be overlapped with original hypergraphs. Thus, if there are several join expressions involving several tuple variables, the screen would be cluttered. Also, nested type queries like Figure 19 lead to complicated graphical queries.

ANSWERTOOL permits the construction of complex queries involving join or set expressions without the user having to understand the notion of new tuple variables, and allows the user to pose the complex queries incrementally.

The idea of ANSWERTOOL originated from the following simple example. Assume there are two relations FACULTY(NAME, OFFICE HOUR, CLASS) and REGISTER(CLASS, NO_STUDENT). Suppose there is a query 'Find faculty members who teach classes in which more than 30 students are registered'. There are two ways of writing an SQL query for this. The normal SQL query is

```
SELECT F.NAME
FROM FACULTY F, REGISTER R
WHERE F.CLASS = R.CLASS and R.NO_STUDENT > 30.
```

However, suppose we already have a constant set C whose elements are the classes in which more than 30 students registered. The SQL query would be as follows:

```
SELECT F.NAME
FROM FACULTY F
WHERE F.CLASS IN C
```

Note that only the FACULTY relation is used making predicates in the above query. Since we have the constant set C, we do not have to join the FACULTY and REGISTER relations.

This above approach can be applied in a universal relational database. Since the univeral relation has all attributes in a database, if a right operand of a join is a constant value or set, the query can be formulated without the direct use of join expressions. The function of ANSWERTOOL is similar to that of the constant set C in the above example.

Suppose the user asks 'Among the customers of BOA bank, who has saved more money than the average balance of the WELL FARGO bank'. This query can divided into two local (sometimes called 'partial') queries q_1, q_2 and the session for the query goes as follows: (1) pose q_1: 'Find all customer's balances of the WELL FARGO bank', (2) keep the query result in the temporary relation answer.rel, (3) look at the query result using ANSWERTOOL and (4) pose q_2: 'Find the customers, at BOA bank, whose balance is more than the average balance of the answer.rel relation'. Since we already have the data of the WELL FARGO bank using ANSWERTOOL, the PICASSO query is simple, as illustrated in Figure 22. The step-by-step formulation of this query can be found in Reference 24.

We can fill in the value and right operand of a predicate using the mouse as shown in Figure 22. Normally, it is easier to click, rather than type, a constant value for the value field or a variable for the attribute field of a predicate. However, we cannot use a pop-up menu for this purpose. The number of choices for the value field is generally too large for a single pop-up menu to be able to show all choices at once. Instead, ANSWERTOOL shows a small subset of the choices. This window (ANSWERTOOL) is conceptually a window on the entire set of choices. The user can scroll up or down the window as we described above. Thus, we allow the user click values in ANSWERTOOL, and the selected value or attribute appears as the right operand of the predicate.

Similarly, some nested type queries can be simplified using ANSWERTOOL. The user first formulates the innermost part of the query and uses the ANSWERTOOL for building predicates piecemeal. Consider the query in Figure 19 again: 'Find the banks containing customers whose balance is more than \$1000'. Using ANSWERTOOL, the query is as shown in Figure 23. Most of the query is formulated with only mouse clicking.

As shown in previous sections, we allow the user to create new hyperedges and draw join expressions graphically. Thus, in PICASSO, the user can either draw a complex graphical query or resort to ANSWERTOOL. We believe that, in general, using ANSWERTOOL is more natural. We believe that, at first naïve users would decompose the complex queries into several simple subqueries and glue them together later. ANSWERTOOL accommodates those needs.

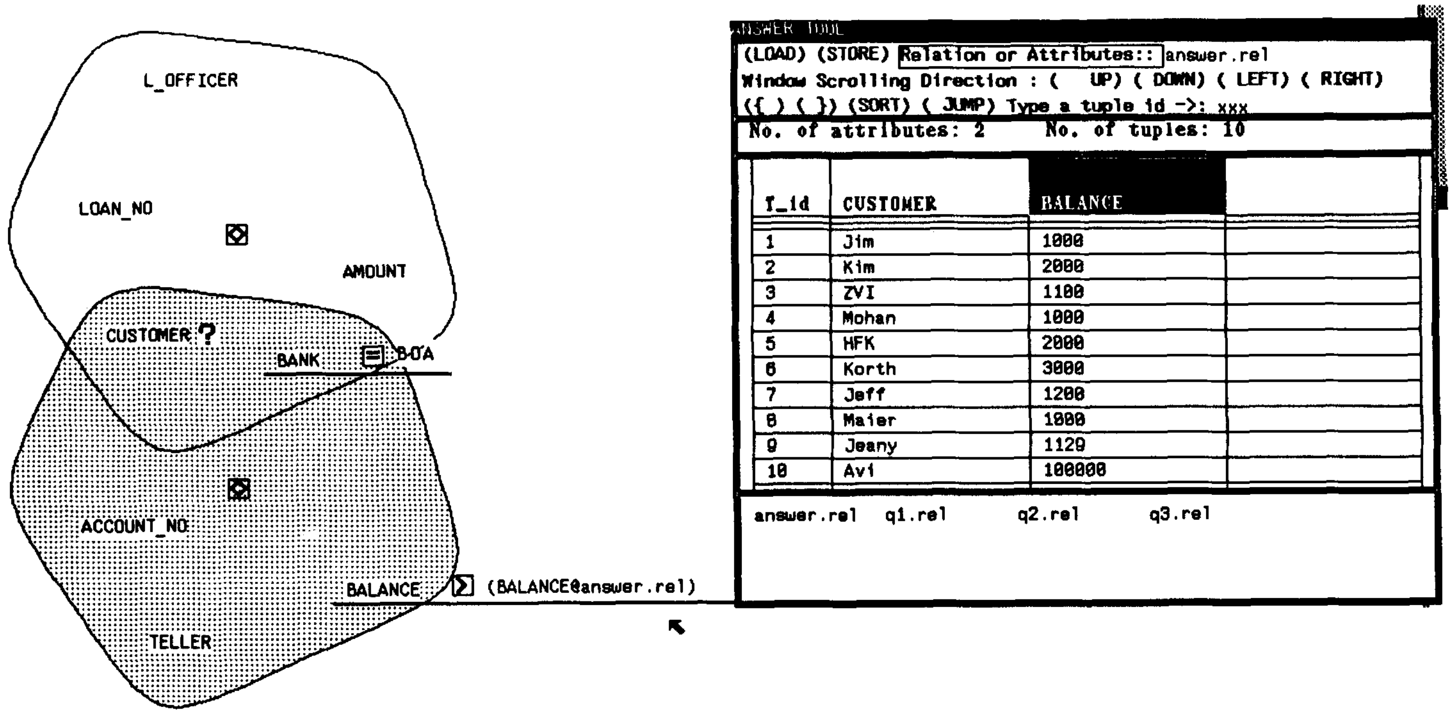
GRAPHICAL FEEDBACK

In the case of ambiguous queries, if the system uses a lengthy dialogue for disambiguation, the user would become impatient or may not understand what is required. In the case of incorrect queries, if they are simply rejected and the user gets only error messages, the user might feel frustrated and may not want to try again. Through graphical feedback, ambiguous queries can be disambiguated and erroneous queries can be corrected easily.[26]

Ambiguous queries

At any point during the specification of a query, the user has mentioned a set of attributes (the union of those attributes appearing in the select clause and those attributes appearing in the predicates of where clauses). It is this *mention set* that is used to determine the maximal objects to be used in answering the query. Thus, if the user clicks the attributes that are shared by more than two maximal objects, the system (query processor) cannot decide which maximal objects should be used for the selected attributes. The reason for the semantic problems is that the underlying hypergraph of objects is cyclic. There are several possible interpretations of a query on a cyclic scheme.

In System/U, the union of all possible answers is given as the answer to an ambiguous query. Consider the query in Figure 24(a), 'Find the customers of BOA bank'. The

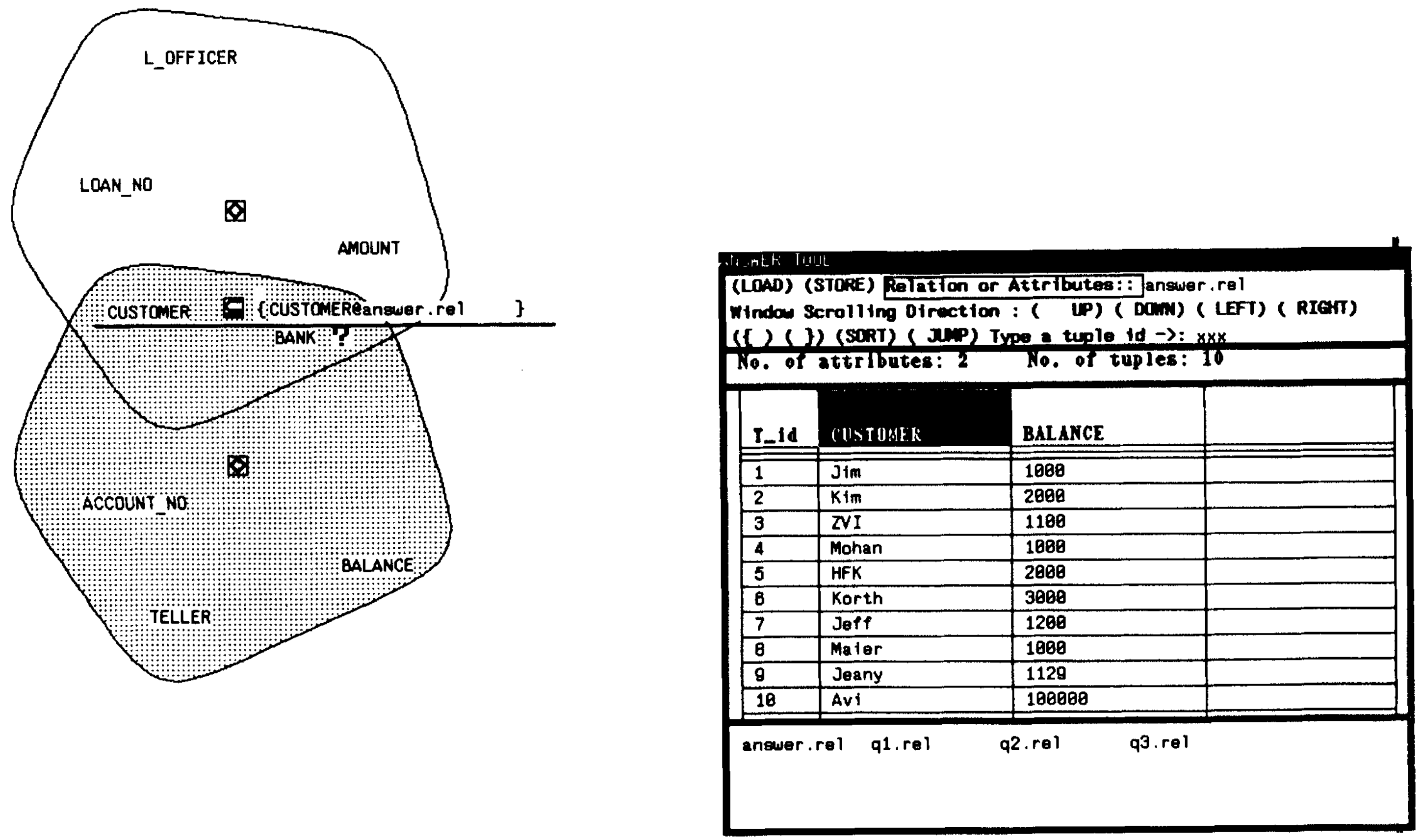


```
(LOAD) (STORE) Relation or Attributes:: answer.rel
Window Scrolling Direction : (   UP) ( DOWN) ( LEFT) ( RIGHT)
({ ) ( }) (SORT) ( JUMP) Type a tuple id ->: xxx
No. of attributes: 2      No. of tuples: 10
```

T_id	CUSTOMER	BALANCE	
1	Jim	1000	
2	Kim	2000	
3	ZVI	1100	
4	Mohan	1000	
5	HFK	2000	
6	Korth	3000	
7	Jeff	1200	
8	Maier	1000	
9	Jeany	1120	
10	Avi	100000	

```
answer.rel   q1.rel       q2.rel       q3.rel
```

Figure 22. Predicate formulation using ANSWERTOOL



```
(LOAD) (STORE) Relation or Attributes:: answer.rel
Window Scrolling Direction : (  UP) ( DOWN) ( LEFT) ( RIGHT)
({ ) ( }) (SORT) ( JUMP) Type a tuple id ->: xxx
No. of attributes: 2     No. of tuples: 10
```

T_id	CUSTOMER	BALANCE	
1	Jim	1000	
2	Kim	2000	
3	ZVI	1100	
4	Mohan	1000	
5	HFK	2000	
6	Korth	3000	
7	Jeff	1200	
8	Maier	1000	
9	Jeany	1129	
10	Avi	100000	

```
answer.rel   q1.rel     q2.rel     q3.rel
```

Figure 23. Nested-type query formulation using ANSWERTOOL

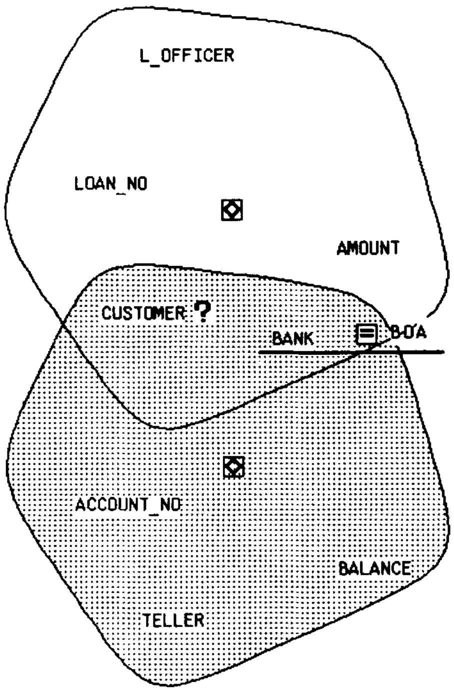
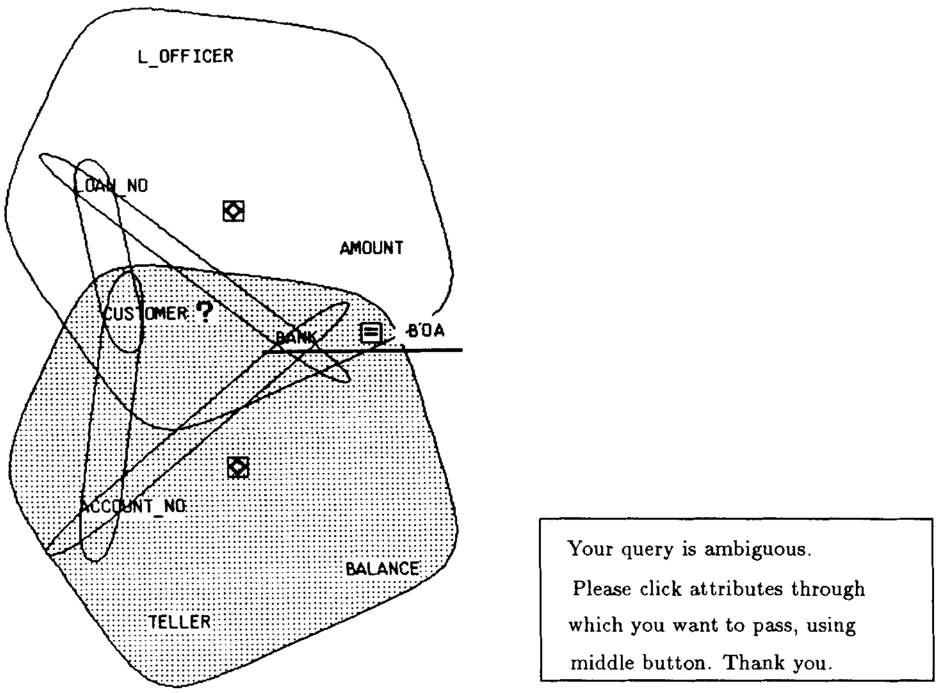


Figure 24(a). PICASSO representation of the query 'Find the customer at BOA bank'

System/U interpretation is 'Find all customers who have either an account or a loan at BOA bank'. If the user desires this interpretation of the query, he would formulate the PICASSO query as shown in Figure 24(a). However, this is not the only reasonable interpretation. For example, if a loan officer asks this query, it is highly possible that he is only interested in all borrowers.

In order to resolve this kind of semantic problem, the PICASSO/ROGUE system provides graphical feedback that requests users to clarify their intentions. After formulating the PICASSO query in Figure 24(a), if the user asks the system 'RUN your Query' using the basic menu, the graphical feedback as shown in Figure 24(b) pops up and the system waits for the user's response. The cyclic structure (which consists of four objects: each hyperedge is one object) among CUSTOMER, BANK, LOAN_NO and ACCOUNT is displayed and the system asks the user to choose the desired path. A path is selected by clicking attributes that must appear in the path.

The loan officer simply clicks the loan attribute (thereby adding it to the mention set). Then a preposition Through is prefixed at the clicked attribute as shown in Figure 24(c). Now, the meaning of the query in the screen is 'Find customers who have a loan at the BOA bank'. In the translated version of the System/U query for the PICASSO query in Figure 24(c), the tautology Loan = Loan is attached automatically. The function of this tautology predicate is to disambiguate the System/U query. This idea is called *name dropping*.[20]



Your query is ambiguous.
Please click attributes through which you want to pass, using middle button. Thank you.

Figure 24(b). A graphical feedback for the ambiguous query

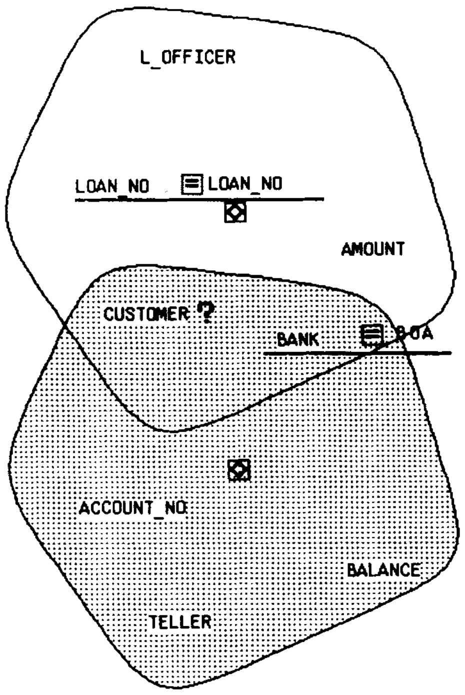


Figure 24(c). Name dropping technique in PICASSO

Erroneous queries

There are circumstances when it is difficult to formulate System/U queries correctly. In conventional database systems as well as System/U, the erroneous queries are simply rejected and the textual error messages (which may be too technical to understand) are shown to the user.

Suppose the user wants to pose the query: 'Find the customers who have saved more money than they have borrowed'. It is quite possible that if the naïve user does not understand the notion of tuple variable and join operation, he would formulate a System/U query such as retrieve CUSTOMER where BALANCE > AMOUNT. The difficulty is that System/U requires that the user be aware of the attributes' membership in maximal objects. The System/U response to the erroneous query is 'No maximal object contains all the attributes you mentioned in the query' and the query is rejected. The correct System/U query is retrieve CUSTOMER where CUSTOMER = T.CUSTOMER and BALANCE > T.AMOUNT. With only the error message as assistance, the naïve user is unlikely to be able to correct the System/U query.

Consider the query 'Find a teller whose customer's loan_no is L100'. Suppose the user poses the PICASSO query in Figure 25(a). The PICASSO query is rejected under universal relation semantics, because the System/U query for this PICASSO query is retrieve TELLER where T.LOAN_No = "L100". The mention set for this query, {TELLER, LOAN_NO}, is not contained in a single maximal object. If the user is skilful enough to pose the query retrieve TELLER where CUSTOMER = T. CUSTOMER and T.LOAN_NO = 'L100', the System/U would accept the query since the mention set of each tuple variable is contained in a maximal object. If the goal of a universal relation

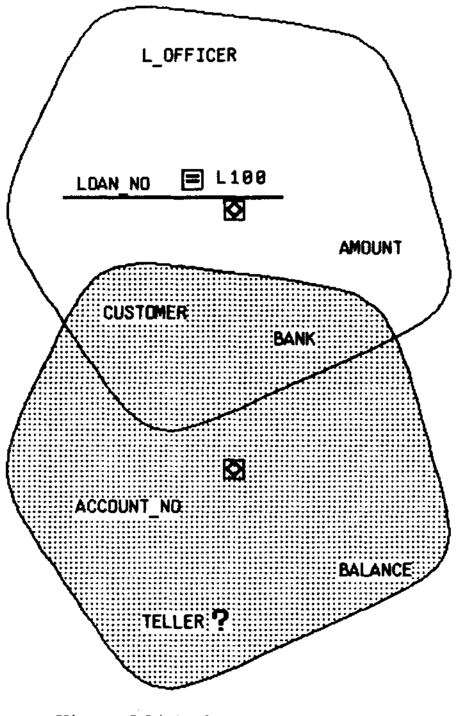
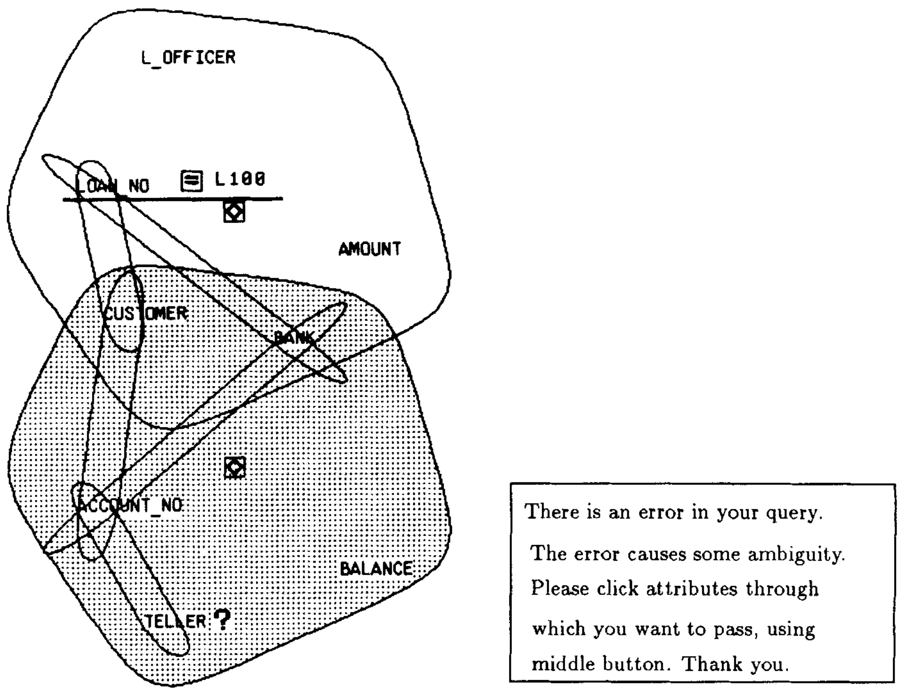


Figure 25(a). An erroneous query

system is to allow the user to pose queries with knowledge of attributes' names only, then the system should find the connection paths and inform the user of the possible ways of connecting the attributes that are located in different maximal objects, rather than rejecting the query in Figure 25(a). In our system, ROGUE gives graphical feedback for this query and explains there are two ways of connecting TELLER and LOAN_NO, as illustrated in Figure 25(b). The user would choose one or more possible paths. If the user clicks the CUSTOMER attribute, the preposition Through is prefixed. However, the representation of this Through construct is not a tautology like CUSTOMER = CUSTOMER as in the example in the beginning of this section. Its internal representation of Through in this situation is CUSTOMER = T.CUSTOMER because the objective of this predicate is connecting two different maximal objects. The details of System/U query formulation from a PICASSO query are described in Reference 24. Figure 25(c) shows a corrected query.

Consider the query 'Find the customers who have an account and a loan at BOA bank'. The two possible ways to express this query in System/U query are complicated, as shown in the following:

```
retrieve CUSTOMER
where BANK = "BOA" and
    ACCOUNT_NO = ACCOUNT_NO and
    T.CUSTOMER = CUSTOMER and
    T.BANK = "BOA" and
    T.LOAN_NO = T.LOAN_NO
```



There is an error in your query.
The error causes some ambiguity.
Please click attributes through which you want to pass, using middle button. Thank you.

Figure 25(b). A graphical feedback for the erroneous query

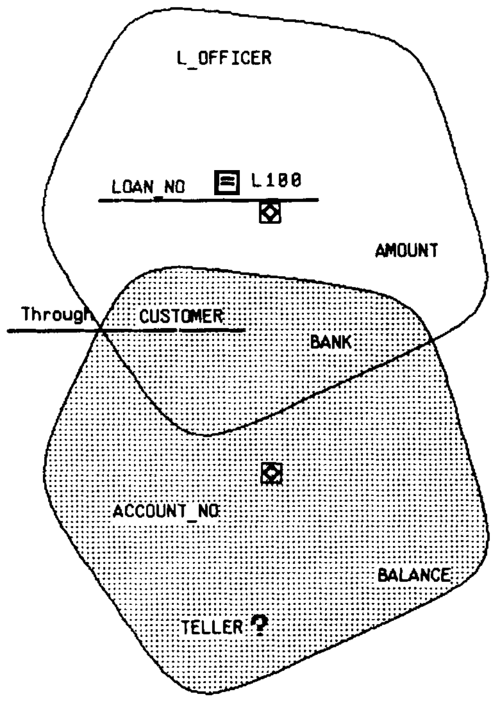


Figure 25(c). PICASSO representation of the query 'Find a teller whose customer's loan_no is L100'

or

```
(retrieve CUSTOMER
  where BANK = "BOA" and
    ACCOUNT_NO = ACCOUNT_NO)
intersect
(retrieve CUSTOMER
  where BANK = "BOA" and
    LOAN_NO = LOAN_NO)
```

However, the user can build the PICASSO query with help of ANSWERTOOL and graphical feedback.

Help messages

There are two types of help messages in ROGUE. One is from the message subwindow, which is located in the upper part of the ROGUE window as shown in Figure 3 and the other type is the pop-up message that emerges around the current location of the mouse. Our strategy for using the above two types of help message is as follows: if the message is routine and unimportant, such as the information of system status, greetings, etc., we use the message subwindow for the message. Whereas, if the message is urgent in query formulation, we use the pop/up type of help messages.

The reason for selecting the above strategy is due to the position of the user's eyes. Since the message subwindow may be located far from the part of a hypergraph on which the PICASSO query is formulated, the user might not see the help message from the message subwindow. However, the user usually concentrates on the position of mouse cursor. The pop-up type messages can be used to make sure the user is alerted.

SUMMARY and DISCUSSION

We defined and implemented a graphical query language PICASSO that is integrated into our graphical interface ROGUE. Through various examples queries, we showed the natural aspects of the hypergraph data model and the expressiveness of PICASSO queries.

The major contribution of PICASSO and ROGUE is that the user can pose complex queries using a mouse without knowing the details of the underlying database schema and the details of first-order predicate calculus or algebra. Eliminating join expressions and tuple variables in queries using ANSWERTOOL eases the task of the user posing graphical queries. Nested type queries can be formulated easily with help of ANSWERTOOL.

At the moment, the semantics of PICASSO are based on the System/U query language. It is possible to extend the features of PICASSO for other database query languages such as SQL. Furthermore, we believe that PICASSO can be extended to accommodate non-first normal form relational database languages such as those described in References 31 and 32.

Completeness

PICASSO is relationally complete because the five basic operations of relational algebra, selection, projection, Cartesian product, set union and set difference are all provided.

What's new?

In general, the relational model attempts to free the user from concern about the physical organization of the data. The universal relational model goes one step further than the relational model because the user does not have to be concerned about some of the logical organization. Using PICASSO, we remove some artificial constraints in query formulation and help users to represent their thought processes naturally:

1. Showing the hypergraph representation of the database schema to users helps them to formulate correct queries and pose complex queries in a natural way.
2. Eliminating character-type tuple variables is another notable feature in PICASSO. Naïve users do not have to learn the concept of tuple variables. PICASSO can support multiple tuple variables for complex queries by drawing hyperedges.
3. A small, but useful tool, ANSWERTOOL helps formulate queries involving joins or nesting. ANSWERTOOL can be used as a facility for constructing a complex query incrementally.
4. In PICASSO, the semantics of *Point and Click* change depending on the context. However we avoid nested pop-up menus which can confuse users. No pop-up

menu has a nested menu in PICASSO.

5. As for dealing with large database schemas, refer to Reference 24. The basic idea is that the user can organize the screen freely in the SCREEN MOVE mode, supported by ROGUE. By scrolling the cursor, the user can look at only the relevant part of a database schema. The user can make empty space for creating a new tuple variable or turn off the visibility of irrelevant maximal objects.

What's wrong?

There are several difficulties in the design and implementation of PICASSO. Most of them are due to the hypergraph data model and the characteristics of the System/U DBMS. The rest are generic problems of graphical interfaces.

1. There has been a substantial discussion about difficulties concerning the universal relation and its hypergraph data model.[27–29] In fact,some assumptions behind the universal relation data model, such as the *lossless join assumption* and the *relationship uniqueness assumption*, are still controversial. The lossless join assumption means that for any universal relation U, the join dependency over the objects in U holds. The relationship uniqueness assumption is that at most one relationship can hold among any set of attributes (i.e. one relationship for one object). A consensus has not been reached as to whether the universal relation is a practical approach for real world databases.
2. The application area of the universal relation appears to be restricted to conventional business data processing type databases. New application areas, such as design databases, user interface modelling and multimedia databases, require augmentation of the universal relation framework. PICASSO depends on System/U to a large extent. Thus, PICASSO inherits the limitations of System/U.
3. The System/U query language is a retrieval-only language. Consequently, PICASSO cannot support update operations.
4. The graphical representation is difficult to implement because it involves a splining and a polygon filling algorithm. As we mentioned, if the structures of maximal objects are complicated, figuring our proper graphical representation is a difficult task that must be accomplished in real time.

Despite the technical difficulties, however, we believe that many features of PICASSO are useful enough to be applied to future graphical interfaces for database systems.

Feature analysis

McDonald and McNally[30] suggest a taxonomy of features and environments that influence the usability of query languages. In their paper, they did a comparative analysis of eighteen query languages using the taxonomy. Their analysis methodology involves seven broad categories: data models, functions (available to a user), expression complexity (expressive power of a query language), language form (syntactic forms of query language constructs), language environment (query language's procedurality and interaction mode), user expertise (user skill level) and interface facilities (various services for the user). Following their taxonomy, we analyse PICASSO as follows:

(a) data model: universal relation data model

(b) functions: retrieval (query) only
(c) expression complexity: relationally complete
(d) language form: pictorial (two dimensional)
(e) language environment: non-procedural
(f) user expertise: casual (naïve end users)
(g) interface facilities: ANSWERTOOL (scratch paper-like tool), graphical feedback, various HELP functions.

ACKNOWLEDGEMENTS

We gratefully acknowledge the referees' helpful suggestions and thank Mark Roth for his comments on an earlier draft. We also thank Dave Hanson for numerous editorial revisions on this paper.

REFERENCES

1. D. Maier and J. D. Ullman, 'Maximal objects and the semantics of universal relational databases', *ACM TODS,* **8**, (1), 1–14 (1983).
2. M. Stonebraker, E. Wong, P. Kreps and G. Held, 'The design and implementation of INGRES', *ACM TODS,* **1**, (3), 189–222 (1976).
3. M. M. Astrahan and D. D. Chamberlin, 'Implementation of a structured English query language', *Comm ACM,* **18**, (10), 580–587 (1976).
4. F. H. Lochovsky and D. C. Tsichritzis, 'An interactive query language for external databases', *Proceedings of the International Conference on Very Large Data Bases*, 1982.
5. G. A. Wilson and C. F. Herot, 'SEMANTICS vs. GRAPHICS — to show or not show', *Proceedings of the International Conference on Very Large Data Bases*, 1980.
6. H. K. T. Wong and I. Kuo, 'GUIDE: graphical user interface for database exploration', *Proceedings of the International Conference on Very Large Data Bases*, 1982.
7. M. M. Zloof, 'Query-by-example: a data base language', *IBM System Journal,* **16**, (4), 324–343 (1977).
8. H. J. Kim, 'Graphical interfaces for database systems: a survey', *Proceedings of The 1986 Mountain Regional ACM Conference*, Santa Fe, New Mexico, April 1986.
9. M. M. Zloof, 'QBE/OBE: a language for office and business automation', *IEEE Computer*, May, 13–22 (1981).
10. N. S. Chang, 'Picture query language for pictorial database system', *IEEE Computer*, November, 22–33 (1981).
11. C. M. O. Moura and M. A. Casanova, 'Design-by-example', Department of Informatics, Pontificia Universidade Catolica Rio de Janeiro, Brasil, April 1981.
12. A. Klug, 'ABE: a query language for constructing aggregates by example', *Proceedings of Workshop on Statistical Database Management*, 1982.
13. N. McDonald and M. Stonebraker, 'CUPID: the friendly query language', *Technical Report*, The University of California, Berkeley, October, 1974.
14. Z. Zhang and A. O. Mendelzon, 'A graphical query language for entity relationship databases', in *An E–R approach to Software Engineering*, North Holland, 1983.
15. C. F. Herot, 'Spatial management of data', *ACM TODS,* **5**, (4), 493–614 (1980).
16. J. A. Larson and J. B. Wallick, 'An interface for novice and infrequent database management system users', *Proceedings of National Computer Conference*, 1984.
17. H. F. Korth, G. Kuper, J. Feigenbaum, A. Van Gelder and J. D. Ullman, 'System/U: a database system based on the universal relation assumption', *ACM TODS,* **9**, (3), 331–347 (1984).
18. R. Fagin, A. O. Mendelzon and J. D. Ullman, 'A simpified universal relation assumption and its properties', *ACM TODS,* **7**, (3), 343–360 (1982).
19. J. D. Ullman, *Principles of Database Systems*, Computer Science Press, Rockville, Maryland, 1982.
20. D. Maier and D. S. Warren, 'Specifying connections for a universal relation scheme database', *Proceedings of ACM International Conference on Management of Data*, June 1982.

21. D. Maier, D. Rozenstein and D. S. Warren, 'Windows on the world', *Proceedings of ACM International Conference on Management of Data*, May 1983.
22. H. F. Korth and A. Silberschatz, *Database System Concepts*, McGraw-Hill, New York, 1986.
23. H. F. Korth and A. Silberschatz, 'A user-friendly operating system interface based on the relational data model', *Proceedings of International Symposium on New Directions in Computing*, August 1984.
24. H. J. Kim, 'Graphical environments for query processing', *Master's Thesis*, The University of Texas at Austin, August 1985.
25. K. H. Lou, 'Efficient algorithms for graph-theoretical problems in hypergraph', *Master's Thesis*, The University of Texas at Austin, 1985.
26. H. F. Korth, 'Graphical query languages for universal relation database systems', *Internal Memo*, The University of Texas at Austin, 1984.
27. W. Kent, 'Consequences of assuming a universal relation', *ACM TODS*, **6**, (4), 539–556 (1981).
28. W. Kent, 'The universal relation revisited', *ACM TODS*, **8**, (4), 562–564 (1983).
29. P. Atzeni and D. S. Parker, 'Assumptions in relational database theory', *Proceedings of ACM Symposium on Principles of Database Systems*, 1982.
30. N. H. McDonald and J. P. McNally, 'Query language feature analysis by usability', *Computer Languages*, **7**, 103–204 (1982).
31. M. Roth, H. F. Korth and A. Silberschatz, 'Theory of non-1NF relational database', *TR-84-36*, The University of Texas at Austin, 1984.
32. M. Roth, H. F. Korth and D. S. Batory, 'SQL/NF query language', The University of Texas at Austin, TR-85-26, August 1985 (also to appear in Information Systems, 1987).