# PICASSO: a graphical query language for naive end users

*H. J. Kim, H. F. Korth, and A. Silberschatz*

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712

TR-85-30, November 1985

# PICASSO: a graphical query language for naive end users

*Hyoung-Joo Kim*†
*Henry F. Korth*†
*Avi Silberschatz*‡

**Department of Computer Sciences**
**The University of Texas at Austin**
**Austin, Texas 78712**

## ABSTRACT

PICASSO is a graphics-based database query language designed for use with a universal relational 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 while experienced users can ignore the feedback if they so choose. Inexperienced users can pose queries without knowing the details of underlying database schema and without learning the formal syntax of a SQL-like query language.

This paper presents the syntax of PICASSO queries and compares PICASSO queries with similar queries expressed 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 work station enhances the usability of database systems.

† 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

## 1. Introduction

Recent research in database systems has concentrated on making database systems easier to use. Among the major data models: network, relational, and hierarchical data model, the relational data model, and in particular *the universal relation model* [MU83], is considered to be the most user-friendly data model, mainly due to the following three reasons:

(1) Users do not have to know about the physical structure of the database (so called "physical data independence").

(2) Most query languages of database systems based on the relational data model are non-procedural. In general, non-procedural query languages are easier to use and learn.

(3) In a universal relation database system, the user does not have to be concerned either about physical structure of the database, or about the logical structure of the database (so called "logical data independence").

Thus, a universal relational database system can be thought 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). The gain in the syntactic side of a universal relational query language is the elimination of the clause for binding tuple variables to relations such as *range-of* clause in QUEL [ST76], *from* clause in SQL [AC75].

Recently, work stations and intelligent terminals have become prevalent in both industry and academia. The user can interact with a computer in a two-dimensional fashion (We call screen-oriented interactions between a user and a computer "two-dimensional"). Furthermore, work stations with facilities such as bit-mapped displays, pointing devices (mouse, light pen, ...), icons, multi-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.

In this paper, we investigate how the graphical definitions of the semantics of the universal relation and the availability of a graphical work station can be combined to enhance the usability of database systems. The universal relation database system we use is System/U [KKU84]. The motivation behind research on graphical interfaces is as follows. Existing high level languages (e.g., SQL, QUEL, etc) are not easy for naive users to use and understand [LT82, WI80, WK82]. Furthermore naive end users and non-expert users (such as statisticians, accountants, data analysts, etc) usually have a long and uncomfortable learning period with conventional query languages. They also have to remember many details such as the names of the record types and attributes in a database schema. Further, they are not familiar with mathematical concepts such as predicate calculus or relational algebra or set theory. In fact, since the System/U query language itself is a QUEL derivative, knowledge about a substantial amount of the formal syntax of QUEL is still required.

Specifically, naive users find difficult to understand the notions of multiple tuple variables, join operations, nested-type queries. In conventional interfaces, since there is no feedback during the query process, it is very difficult for the naive or non-expert user to formulate a complex query correctly on the first try.

The above problems become worse if the data base has a large and complex schema. As a way of freeing naive and infrequent users from a long, uncomfortable learning period with formal query languages, many new graphical interfaces and query languages [CA80, CH81, Fo84, HE80, KL82, LP77, LT82, LW84, MC81, McA74, McB75, ST82, WK82, WI80, ZM83, Zl81] have been invented. The first successful example was

IBM's Query-By-Example (QBE) [Zl77]. Since it displays table skeletons - the column heading of relations, QBE could be more user friendly than other relational query languages such as SQL or QUEL. Several researchers explain why a graphical interface is an efficient and friendly means of accessing to the database and suggest essential features which should be included in graphical interfaces:

(a) The graphical interface should be able to provide information about the schema of the underlying database to the user,

(b) There should be a facility in which the user can formulate queries in a piecemeal manner. Building a complex query in piecemeal manner means that users can apply predicates of the *where* clause incrementally or use the result of the previous query for building predicates of new query,

(c) There should be a facility which allows the user to browse the database freely,

(d) The interaction with a graphical interface should be easy,

(e) Graphical feedback should be provided during query processing to guide the user in the formulation of correct queries.

The purpose of this paper is to describe our approach to the design and implementation of a graphical database query language. We first introduce the System/U database system on which we have based our work. After giving an overview of the system structure we present each of the main components of our system:

- the ROGUE graphical interface
- the PICASSO query language
- the ANSWERTOOL facility for relation browsing

## 2. An Overview of our System

Since our graphical query language is heavily dependent on System/U, we will start by presenting a brief introduction to System/U. Korth, et al. [KKU84] implemented the System/U database management system which is based on a universal relational data model [FMU82, MU83, Ull82]. In a universal relation database system, the user sees only one relation (the universal relation). The gain in ease-of-use from the universal relation model is the elimination of the *from* clause from the query language. Since tuple variables are bound to a universal relation, it is not necessary to declare relations to which tuple variables are bound. Join operations are hidden by a universal relation. The system determines which relations of the underlying database need to be joined in order to respond to a given query.

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 (called *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" [MU83, MRW83]. The intuitive notion of a *maximal object* is a group of related attributes in which the user can pose System/U queries in a straight forward manner. This approach is to bind each tuple variable over a set of semantically strongly connected objects. In fact, this is an approach to avoid cyclic database schemes. [Ull82] is a good reference for 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).

The complete version of the System/U query interpretation is in [KKU84]. Under the notion of maximal object, the intuitive version of System/U query interpretation is as follows.

1. Run the query in each maximal object that contains all attributes mentioned in the query.

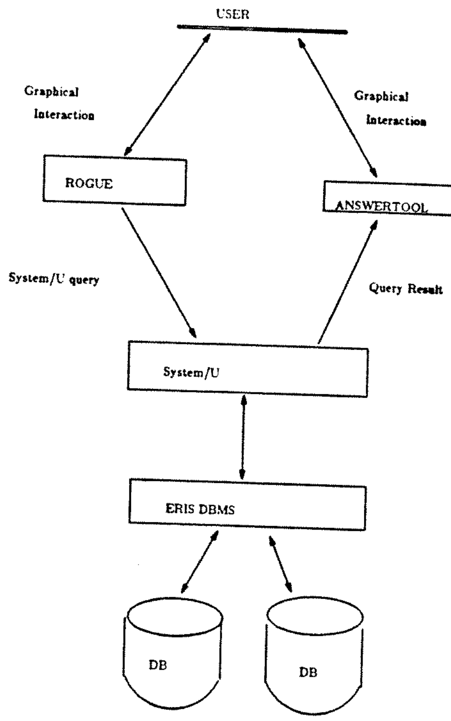


Figure 1: The system diagram of PICASSO/ROGUE environment

2. Take the union of the results.

System/U consists of two parts: DDL (Data Definition Language) step and DML (Data Manipulation Language) step. The input for the DDL step is the specification of the attributes and their data types, the relation schemes and the names of relations, the objects, and any functional dependencies that hold. The data base administrator (DBA) or user prepares the System/U DDL input. Given a DDL input, the system creates the list of maximal objects and sets of objects which belong to each maximal object. The database designer can change the contents of maximal objects.

## 3. System Structure of PICASSO Environment

PICASSO (PICture Aided Sophisticated Sketch Of database queries) is a graphical database manipulation language which is being used in a graphical user interface for System/U called ROGUE (ROsi's Graphical User Environment). The details about ROGUE are introduced in [Kim85]. The major function of ROGUE

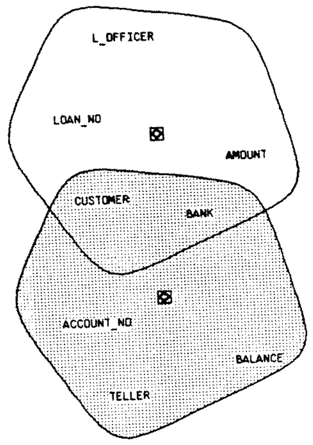


Figure 2: Hypergraph representation of Bank database schema

is translation of PICASSO queries to System/U queries. Also ROGUE supports various facilities to assist the user in formulating graphical queries. PICASSO and ROGUE have been implemented within the ROSI (*R*elational *O*perating *S*ystem *I*nterface) project at The University of Texas at Austin [KS84].

PICASSO is a high level, user friendly query language which is easy to use and learn. Figure 1 shows the diagram of PICASSO/ROGUE environment. Eris is an experimental database system which 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 the UNIX operating system. We implemented ROGUE and ANSWERTOOL in C, using the SunWindow graphics system on the Sun work station under the UNIX operating system. After the user formulates a PICASSO query, ROGUE generates and sends a System/U query to System/U. Then System/U sends the result of the query to ANSWERTOOL. The user can browse the query result from ANSWERTOOL and use the constant values of the previous query result for formulating the predicates of another query. Thus, the user can build the complex query in a piecemeal or incremental manner.

Suppose a bank database consists of 4 relations: rofficer(L_officer, Bank), rteller(Teller, Bank), rloan(Customer, Bank, Loan_no, Amount), racct(Customer, Bank, Account_no, Balance). According to universal relation theory, the bank database consists of two maximal objects: MaxObj1 = (Bank, Loan_No, Amount, L_officer, Customer), MaxObj2 = (Bank, Account_No, Balance, Teller, Customer). ROGUE draws a hypergraph representing the maximal objects on the screen and the PICASSO queries are formulated on this hypergraph by the user. Figure 2 shows the hypergraph representation of the bank database. The graphical representation of a database schema using hypergraphs provides the user with valuable information regarding the semantics of the database.

Query formulation is done directly on hypergraphs using the mouse. This is accomplished by clicking buttons of mouse and choosing from a pop-up menu. The HELP messages and other facilities for assisting users are developed in ROGUE. Even though the underlying DBMS is System/U, the user does not have to learn the formal syntax of the System/U query language.

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 etc. Furthermore, we believe that PICASSO can extended to accomodate non first normal form relational database languages such as those of [RoK85, RKS84].

We show the expressibility of PICASSO query through various examples. The example queries of PICASSO 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. At the end of this paper, we show that PICASSO is relationally complete.

## 4. Overall Structure of ROGUE

ROGUE supports various facilities for helping users to pose queries in a graphical manner. After the user formulates the PICASSO query, ROGUE converts the PICASSO query into System/U query and sends the translated query (the details of System/U query translation are in [Kim85]) to System/U. In this section, we briefly introduce the structure of ROGUE which is implemented as a tool which is a standard technique for implementing a graphic software in SunWindow system. Figure 3 shows the overall structure of ROGUE. It consists of 3 subwindows; a message subwindow, an option subwindow, and a graphic subwindow.

### *4.1 Message subwindow*

The message subwindow keeps the user informed about the current mode (there are several modes in query processing such as select-attribute mode, predicate-formulation mode, undo mode, formal-query mode, etc). The 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.

### *4.2 Option subwindow*

There are 2 types of commands in a graphic tool of the SunWindow system : options and pop-up menus. Options usually handle the change of context (mode). A pop-up menu displays a set of commands which are available in a given context. In some sense, an option window deals with a level one higher than a pop-up menu in the command hierarchy. Thus, the contents of pop-up menus can differ based selected options. We currently support eight option items in the option subwindow.

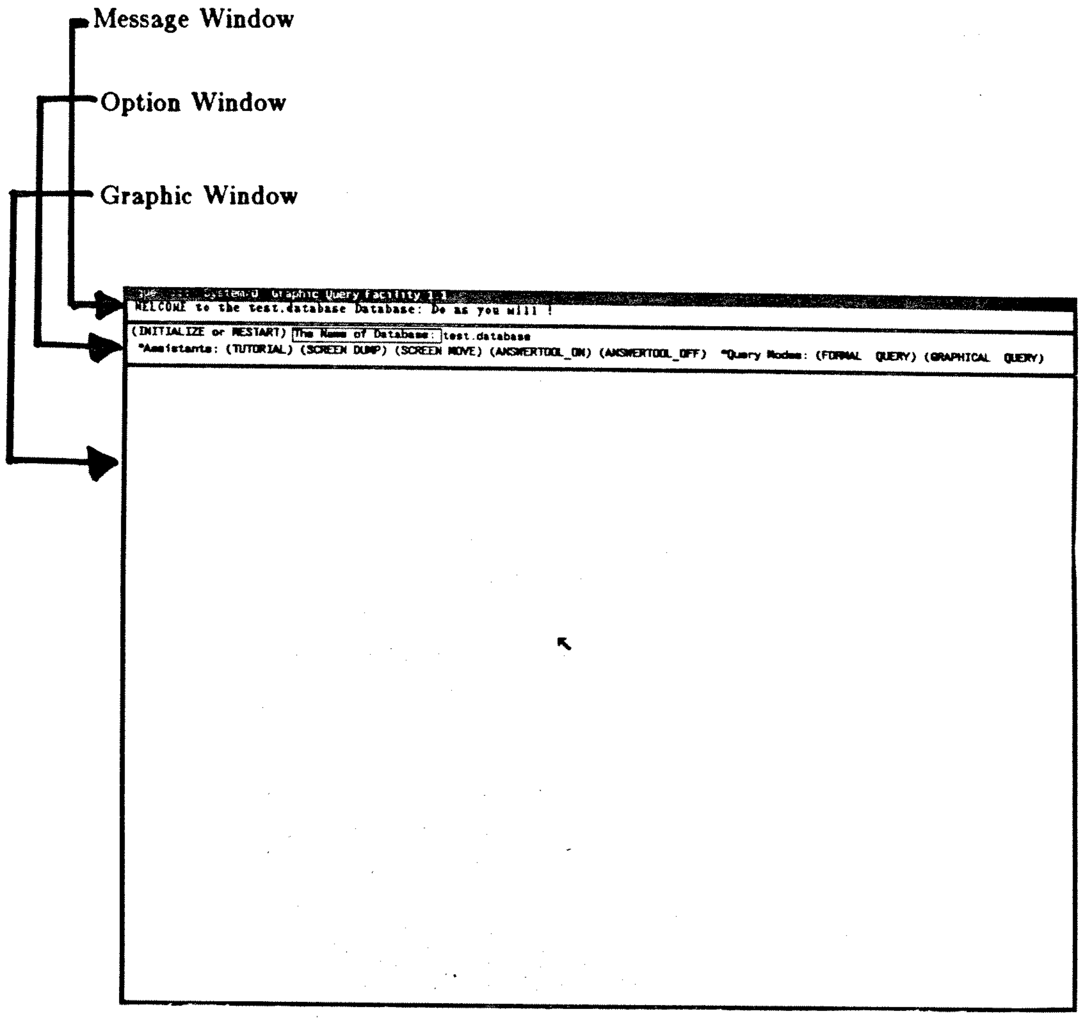
(1) INITIALIZE: reads the data of database schema which will be used in hypergraph representation.

(2) TUTORIAL: provides the user with a brief introduction to ROGUE, some examples of PICASSO queries, and explains to the user the functions of mouse buttons, pop-up menus, and options.

(3) SCREENDUMP: provides the user with a hard copy of a screen dump using a laser printer.

(4) SCREENMOVE: allows the user to navigate through a database schema. The details are covered in subsection 4.3. In this mode, the user can relocate the positions of hypergraphs or reserve some space for multiple tuple variables.

(5) ANSWERTOOL_ON: invokes ANSWERTOOL (creates ANSWERTOOL as a child process). The communication channel (pipe) between ANSWERTOOL and ROGUE will be set up at this time


Message Window
Option Window
Graphic Window
System/U Graphic Query Facility 1.1
(INITIALIZE or RESTART)
The Name of Database:
*Assistants: (TUTORIAL) (SCREEN DUMP) (SCREEN MOVE) (ANSWERTOOL_ON) (ANSWERTOOL_OFF)
*Query Modes: (FORMAL QUERY) (GRAPHICAL QUERY)

```
WELCOME to the test.database Database: Do as you will !
The Name of Database: test.database
```

Figure 3: Overall structure of ROGUE

for interprocess communication. The details are in subsection 4.8. Using this tool, the user can formulate complex queries in a piecemeal manner.

(6) ANSWERTOOL_OFF: kills the child process ANSWERTOOL.

(7) GRAPHICAL QUERY: is the default mode when the user starts the session. The user can return to this graphical query mode from the formal query mode.

(8) FORMAL QUERY: allows the user to go into the formal query mode if he does not want to pose graphical queries. In this mode, the tabular representations of relations are given to the user. The system assumes that the users pose queries using the System/U query language. The DML (data manipulation language) step of System/U is invoked automatically.

## *4.3 Graphic subwindow*

The graphic subwindow is for the hypergraph representation of a database schema and the user's drawing of

PICASSO query. The system draws the hypergraphs for the database schema in the graphical subwindow. The purpose of displaying the maximal object hypergraph is to provide the user with the semantic information about the database schema. Query formulation is done directly on hypergraphs using a mouse. This is accomplished by clicking the buttons of a mouse and choosing the contents of pop-up menus.

## 5. Query Facilities

When a user formulates a query, he must specify the attributes in which he is 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 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 one 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 goto step 1; else return;

The above steps of the human thought process about query processing can be successfully emulated with three buttons of a mouse which have separate functions. Thus, selection attributes may be added after some predicates have been specified, etc. PICASSO supports an undo mechanism for modifying a query. Using the undo mechanisms, predicates can be removed and selected attributes unselected. The undo mechanism for modifying graphical queries will be explained later in this chapter.

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

If the user wants to prefix some aggregate operators, or pose set operators between two selected attributes, 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 **SET DIFFERENCE, SET INTERSECTION,** and **SET UNION** which will be covered later in this section. The user chooses IGNORE option to make the pop-up menu go away.

After selecting attributes for the *select* clause, the user presses the middle button of mouse for describing some conditions (i.e. predicates) on desired attributes. Following Fagin, et al.[FMU82], we treat the predicate P of the *where* clause as a conjunction of predicates $P_1$, $P_2$, ... ,$P_n$. Each $P_i$ is expressed separately by the user. We assume that each $P_i$ is one of two forms:

- $\langle$*attribute*$\rangle\langle$*comparison operator*$\rangle\langle$*value*$\rangle$
- $\langle$*attribute*$\rangle\langle$*comparison operator*$\rangle\langle$*attribute*$\rangle$

Using the middle button of mouse, the user starts to formulate predicates. The first selected attribute using the middle button is the lhs (from now on, we want to use *rhs(lhs)* instead of right(left) hand side) of predicate. Then the menu of comparison operators will pop up. See Figure 6. The first six items are

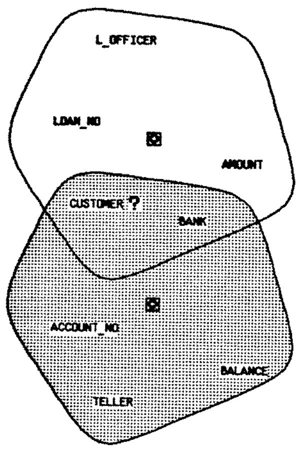


Figure 4a: "Display all bank names" by PICASSO

| rloan | Customer | Bank | Loan-No | Amount |
|---|---|---|---|---|
| | P. | | | |

| racct | Customer | Bank | Accnt-No | Balance |
|---|---|---|---|---|
| | P. | | | |

Figure 4b: "Display all bank names" by QBE

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

Figure 4c: "Display all bank names" by SQL

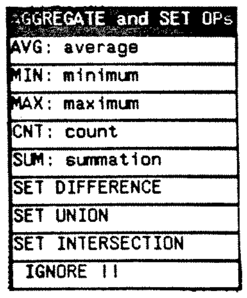


Figure 5 : Pop-up menu for Aggregate operators and Set operators

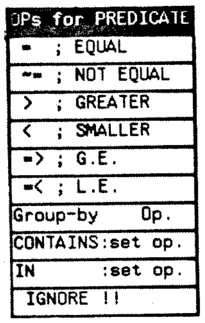


Figure 6: Menu for predicate formulation

Figure 7 : Template for keyboard typing

comparison operators. The seventh item is for the group by operator and the eighth and nineth items are for set operations. The seventh, eighth and nineth items will be explained later in this chapter. After selecting the comparison operator, the template of Figure 7 will pop up automatically. If the user clicks an attribute it means he wants to compare with that attribute. If the user types he wants to type a constant value.

First consider the case that the *rhs* 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. In section 6, we introduce the tool called ANSWERTOOL. Using ANSWERTOOL, the user can formulate a query without typing. Clicking the mouse can be used even to enter a constant value.

The middle button is used for selecting the right hand side attribute of predicates. To illustrate this, consider the query "Find a customer having a loan for an amount greater than that of one of HFK's loan".

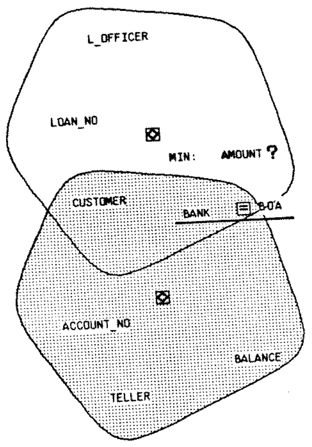


**Figure 8: "Find the minimum amount on loans at BOA bank" by PICASSO**

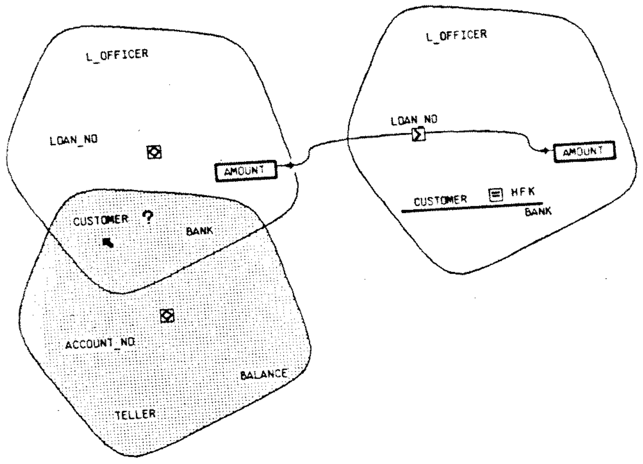


**Figure 9: "Find a customer having a loan for an amount greater than that of one of HFK's loan" by PICASSO**

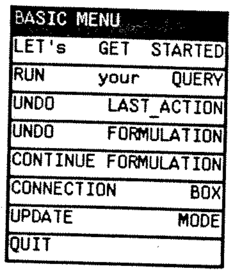


**Figure 10: Pop-up menu for basic query processing**

As shown in Figure 9, the PICASSO query involves new tuple variable creation. The details of new tuple variable creation are covered in subsection 5.6.

Arrows are used to indicate which attribute is the left or right component in a predicate. We believe that this representation method using arrows assists naive users in understanding the meaning of query. This is necessary for asymmetric comparison operators such as < or >.

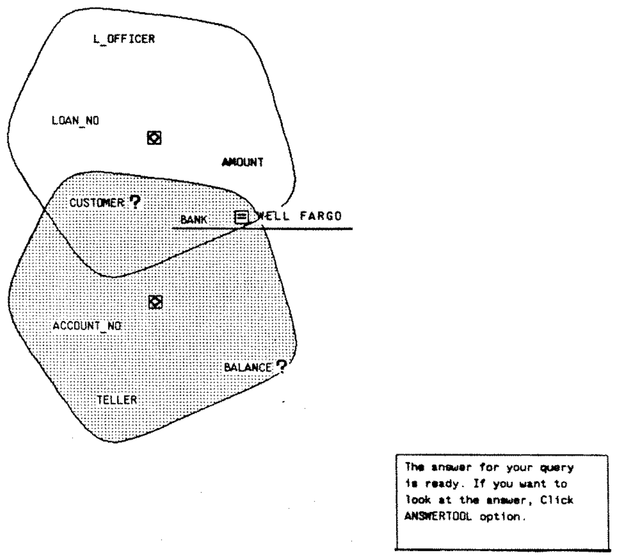
## 5.1 Basic menu for query processing

Besides the previously mentioned pop-up menus, there is also a basic menu which can be triggered by the right button of mouse. If the user clicks the right button of mouse, the menu of basic operations for query processing will pop up as in Figure 10. The following menu items are available.

- *LET's GET STARTED:* draws hypergraphs which represented the underlying universal relation scheme. The purpose of displaying the maximal object hypergraphs is to suggest to the user what the semantics of database define as the canonical connections among sets of attributes. In fact, there are many theoretical problems in representing hypergraphs graphically. [KoL85] will illustrate efficient algorithms and heuristics for hypergraph representation problems which include the planarity and colorability problem in a hypergraph.
- *RUN your QUERY:* means that the user wants to see the result relation after processing graphical query. The ROGUE system will translate the PICASSO query into System/U queries and send the translated query to System/U. Then, after processing the query the System/U will give back the query result. After receiving the query result from System/U, ROGUE pops up the message in Figure 11. As shown in Figure 11, if the user wants to look at the query result, he can call ANSWERTOOL. The details of ANSWERTOOL appear in section 6.
- *UNDO LAST_ACTION:* undoes the last action. This undo is idempotent.
- *UNDO FORMULATION:* can be used when user want to change the content of query. This mode is for partial modification a graphical query. As the left button is used for the *select* clause and the middle button 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.
- *CONTINUE FORMULATION:* resumes formulation of queries after partial modification in UNDO mode. The user can add more attributes or predicates into a existing query.
- *UPDATE MODE:* assumes that the user is in update mode. Since the current implementation of System/U does not allow the user to update a database and there are many semantic problems in graphical languages that allow updating of a database, this part has been left for future research. The recent work by Keller [KeU84, Kel82, KeW84] might be applied in our system.
- *CONNECTION BOX:* is described later in section 5.5.
- *QUIT:* is for quitting the session.

## 5.2 Group_by operation

A relation can be divided 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 from that of predicates. We embedded Group-By item in the pop-up menu for the middle button of the mouse. Consider the query: "What is the average balance of each bank?" (See Figure 12).



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

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

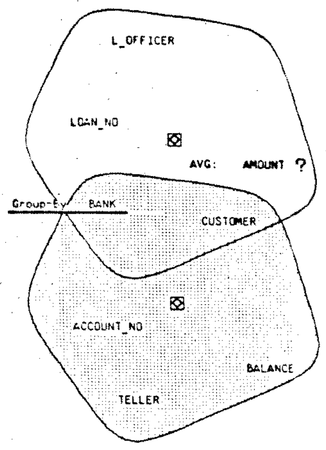


Figure 12 : "What is the average balance of each bank?" by PICASSO

A Group_by operator may be followed by a *having* clause in SQL. We also allow use of a *having* clause. After using a Group_by operator, if the user formulates any predicate, that predicate could be either a *having* clause or a part of a *where* clause. There could be a semantic problem: How do we distinguish a *having* clause from a *where* clause? In order to prevent such a semantic problem, a simple dialogue is used. The dialogue is necessary only in cases in which the user has already formulated a Group_by operator. To illustrate this, consider query: "Find the average loan in each bank having the customer HFK". When the user specifies the *having* clause, the simple dialogue in Figure 13a for distinguishing a *having* clause would be popped up. The finalized formulation of this query is in Figure 13b.

## *5.3 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 like "LOAN_NO between L100 and L10020" would need the two constraints such as "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 clicks the LOAN_NO attribute again using the middle button and just types the second constraint. The system automatically arranges the syntax as shown in Figure 14.

## *5.4 Implicit conjunctive form of predicates*

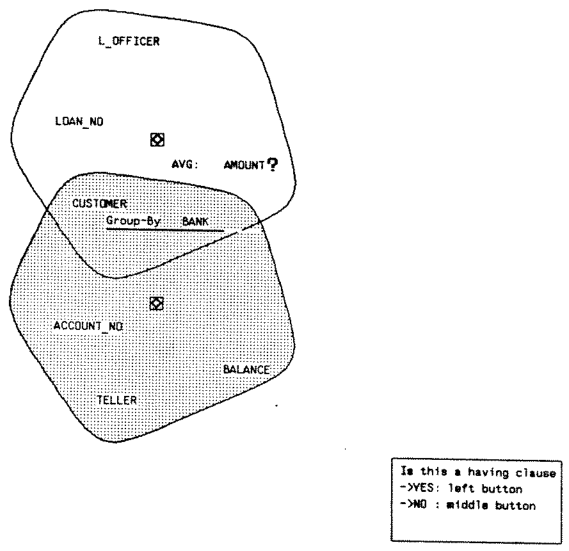
We give preference to conjunctive normal form in our definition of the query language (by default, the specified predicates in the screen are assumed to be connected by AND operators). If the user wants to pose disjunctive type queries, it is his responsibility to specify which predicates are combined with the OR operator using the CONNECTION BOX option in the basic menu. The idea of CONNECTION BOX is from a 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 comparison operator. CONNECTION BOX pops up as shown in Figure 15. Negation of predicates can be entered using the CONNECTION BOX (see Figure 16).

## *5.5 Tuple variable creation*

We assume that each hyperedge has its own tuple variable. The user can draw maximal objects for the purpose of creating new tuple variables instead of using character tuple variables as SQL does. A new tuple variable can be created by by opening *the diamond box* which is located at the center of each original hyperedge. This is done by clicking with the middle button of mouse (If the user clicks this diamond box using the left button that means the user wants to select all attributes in the hyperedge). Figure 17 shows an example of creation of maximal object.

We believe that users should not be forced to write character-type tuple variables because the character-type tuple variables, like t and s, are meaningless from the user's point of view. In general, it is considered that queries involving multiple tuple variables are difficult to formulate and understand. The reader should notice that this approach (i.e., drawing another hyperedge) eliminates of 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



Is this a having clause
->YES: left button
->NO : middle button

Figure 13a: A simple dialogue for distinguishing a *having* clause

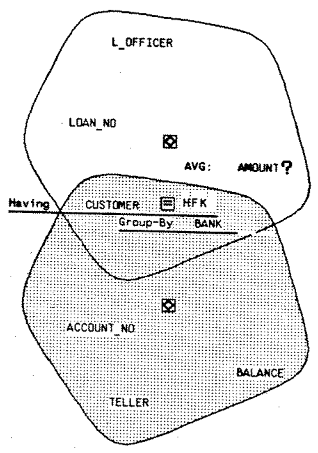


Figure 13b: "Find the average loan in each bank having the customer HFK" by PICASSO

two tuple variables in a natural way without requiring that the user learn the concept of tuple variable. Our representation of queries is natural and easy to understand because creations of a new maximal object can make users understand how the join operation is performed on two copies of same maximal objects.

## *5.6 Set operations*

We allow the use of set operators such as **IN,CONTAINS, SET DIFFERENCE, SET INTERSEC-**

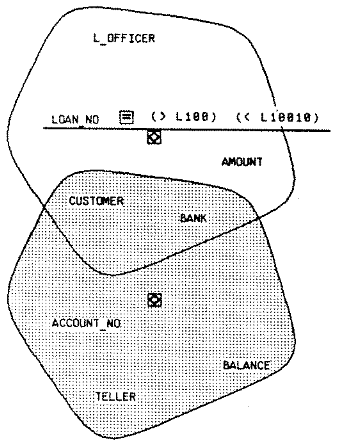


Figure 14: Multiple constraints upon a single attribute

**TION**, 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. For the five set operators, we devise five icons which can imply the meaning of set operations. Note that the operators **IN** and **CONTAINS** can be used only in predicates and the operators **SET DIFFERENCE, SET INTERSECTION, and SET UNION** are for returning a set of tuples.

Consider the following query: "Find customers of BOA who are not customers of FCU ?". Figure 18 illustrate how queries involving set operations are formulated in PICASSO and QBE.

Consider the following *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 subquery and outer query have distinct instances of the "blank" tuple variables. As shown in Figure 19 the PICASSO query involves a new tuple variable in order to distinguish the inner and outer sub query. In chapter 6, we show that a facility called ANSWERTOOL can be used for building nested type queries in a simple and easy manner without the notion of the tuple variable.

## *5.7 Cartesian product type queries*

The diamond box of maximal objects is used to enter cartesian products. We already explained the function

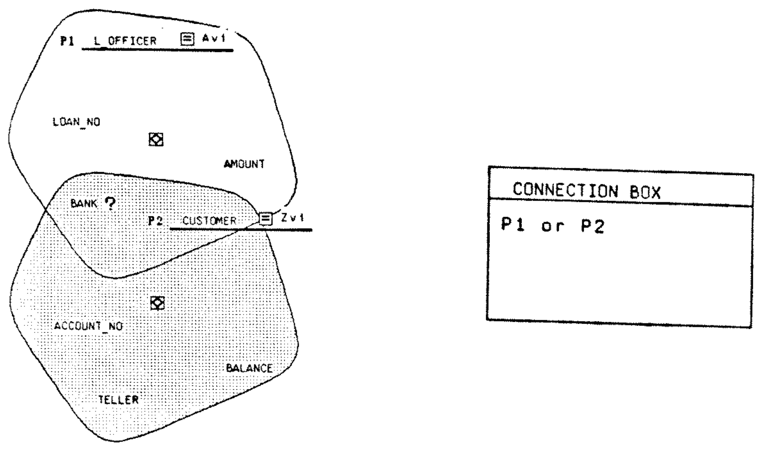
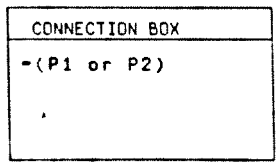


Figure 15: Disjunctive type predicates and the CONNECTION BOX


CONNECTION BOX

¬(P1 or P2)

Figure 16: Negation of predicates

of the diamond box. When we want to create another tuple variable, we click the diamond box with the middle button of mouse. If user clicks the diamond box with the left button of mouse, this means that he wants to look at contents of all attributes in that maximal object. The icon of "?" is postfixed at all attributes in the maximal object. If user clicks two or more diamond boxes, this means that he is interested in the content of all attributes which 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". See Figure 20. The graphical query in Figure 20 is entered by clicking two diamond boxes in each hyperedge. The problem in the translated System/U query of the above PICASSO query is that no maximal object contains the mention set (the union of the attributes appearing in a *select* and *where* clause). However, the ROGUE system regards the cartesian product of maximal objects as a special case. Thus, the cartesian product of maximal objects

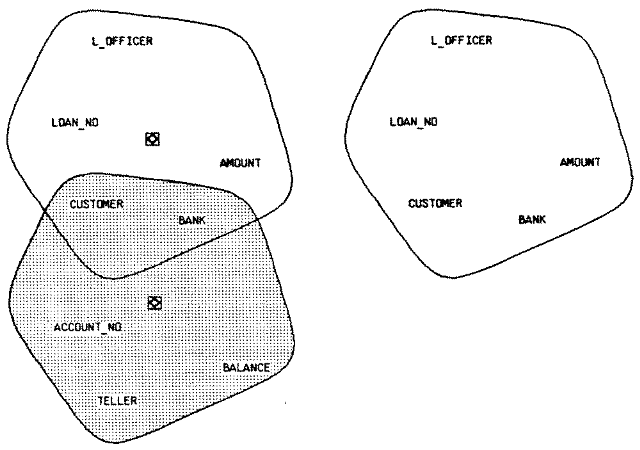


Figure 17: Creating another tuple variable

is not the feature of System/U, but of ROGUE.

## 6. ANSWERTOOL

In previous subsections, we have introduced the expression of nested queries and queries involving joins. Even though our approach is natural and easy to understand, there are some problems in expressing join operations and nested queries graphically. In this section, we propose a new way of handling join expressions easily and efficiently using a new tool, ANSWERTOOL.

The answer to a query is a relation. The answer relation is displayed in a new window which is called ANSWERTOOL on the screen. The major function of ANSWERTOOL is browsing the query result. Whenever ANSWERTOOL is invoked, the communication channel between ANSWERTOOL and ROGUE is set up. ROGUE and ANSWERTOOL communicate with each other in building PICASSO queries by sending messages.

### *6.1 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 temporarily created relations.

The ANSWERTOOL supports some options such as *store, load, scroll up, scroll down, scroll left, scroll right, open_bracket, closed_bracket, sort by a certain attribute,* and *jump to a certain tuple_id.*

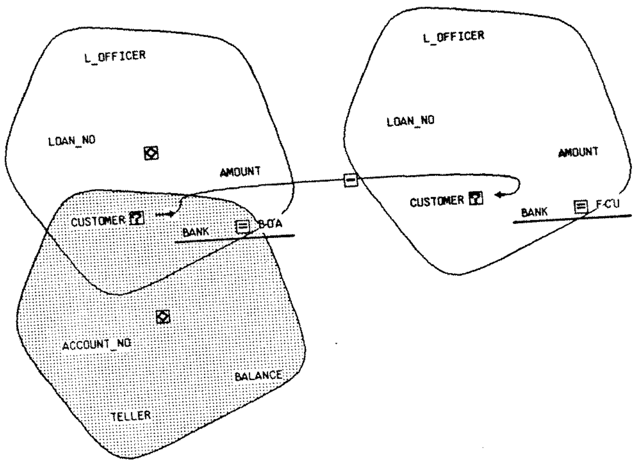
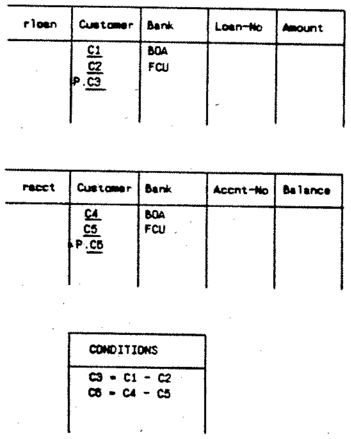


Figure 18a: "Find customers of BOA who are not customers of FCU" by PICASSO



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

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

| CONDITIONS |
|---|
| C3 = C1 - C2 |
| C6 = C4 - C5 |

Figure 18b: "Find customers of BOA who are not customers of FCU" by QBE

• *Store and Load*: We can create temporary relations through ANSWERTOOL. Normally this operation would be issued after a retrieval query. If we wish to assign the retrieved tuples to a new relation, we just type the name of relation and click the *STORE* option. The newly assigned relations can be accessed through the answer tool whenever the user wants to see them (*LOAD* option is for this).

• *Scroll up, Scroll down, Scroll left,* and *Scroll right*: are for screen scrolling. *Scroll left* and *Scroll right* would be necessary if the result relation from the query has many attributes. Whenever the user clicks the scroll up (down) option, new 10 tuples are to scroll up (down). Whenever the user clicks the scroll left (right)

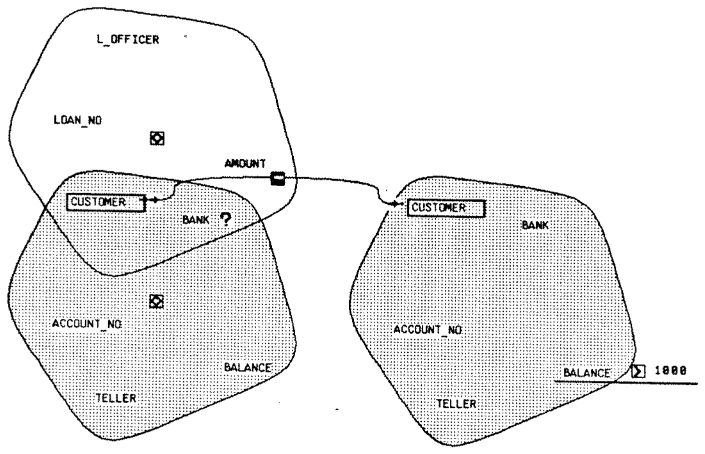


Figure 19: "Find the bank containing customers whose balance is more than $1000" by PICASSO

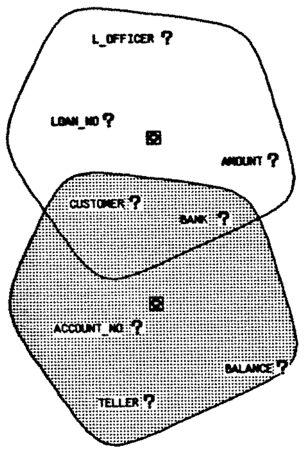
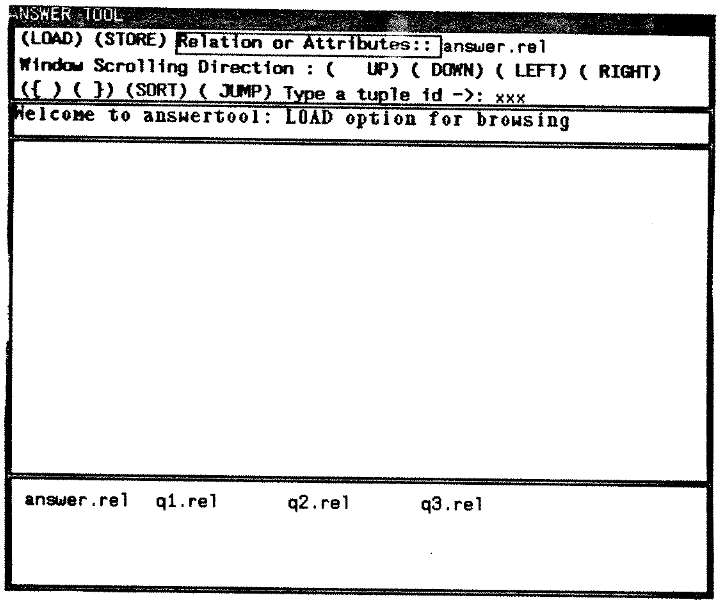


Figure 20: "Show all attributes in Bank database" by PICASSO



```
ANSWER TOOL
(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      q3.rel
```

Figure 21: ANSWERTOOL window for answer relation

option, the window is shifted to the left (right) by one attribute.

• *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.

• *Sort by a certain attribute*: sorts all tuples in a query result by a attribute on which the user clicks by a left button.

• *Jump by a certain tuple_id*: jumps to a portion of a query result which starts with a tuple_id typed by the user.

The ANSWERTOOL does not support any relational operators and is simply a relation browser. The bottom of ANSWERTOOL is used for showing the names of temporary relations. Thus, the temporarily created relations can be accessed at any time. Several answer relations may be maintained at one time. See Figure 21.

## *6.2 Towards a join-less, tuple_variable-less, nesting-less query language*

As shown in the section 5.6, it is generally difficult to represent join expressions graphically. In Figure 10, if there is not enough space for drawing another hyperedge (new tuple variable creation), a new hypergraph may be overlapped with original hypergraphs. If there are several join expressions involving several tuple variables, the screen would be cluttered. Also, in the case that the user wants to draw nested type queries (see Figure 19), the screen would be cluttered.

ANSWERTOOL permits the construction of complex queries involving join expressions without the user having to understand the notions of new tuple variables and join expressions.

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 the constant set C = {CS386, CS372} which 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 the REGISTER relation was used only for making predicates in the above query. In fact, if we have the constant set C available, we do not have to join FACULTY and REGISTER relations.

This approach can be applied in a universal relational database. Since the universal relation has all attributes in a database, if we have a right hand side of join expression as a constant value or set, the query can be formulated without join expressions. A graphical PICASSO query for a System/U join-less query is easy to build and easy to understand. The major objective of ANSWERTOOL is for guiding the user to pose the complex queries as join-less queries in a piecemeal fashion.

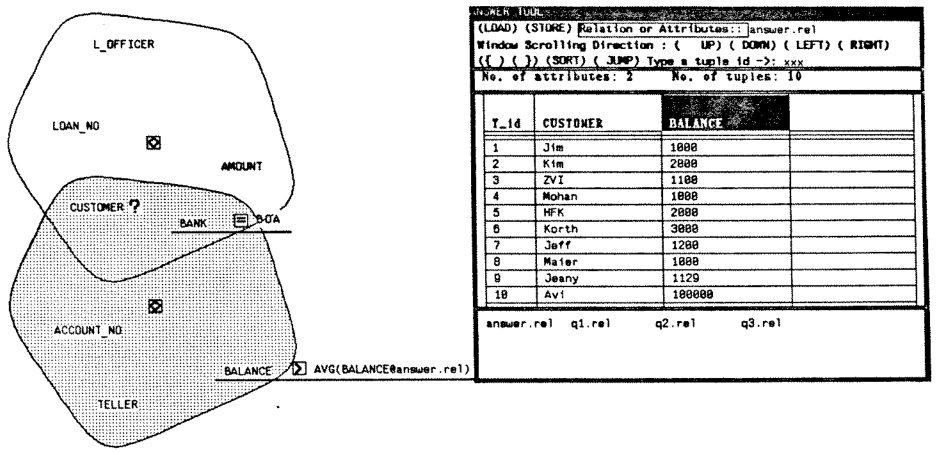
As we discussed in section one, one of important features which a graphical interface should have is a facility in which the user can formulate a complex query in a piecemeal manner. It is difficult for the naive user to formulate a complex query correctly at the first try.

Suppose the user asks: "Among the customers of BOA bank, who has saved more money than the average balance of WELL FARGO bank". This query can divided into two local (sometimes called "partial") queries $q_1$, $q_2$ such as $q_1$: "Find all customer's balance of WELL FARGO bank, keep the query result in the temporary relation answer.rel and look at the query result using ANSWERTOOL" and $q_2$: "Find the customers, at BOA bank, whose balance is more than the average of balance of answer.rel relation". See Figure 22. Since we already have the data of WELL FARGO bank using ANSWERTOOL, the PICASSO query is simple. See the appendix for step-by-step formulation of this query.

Another important function of ANSWERTOOL is that we can fill out the value and attribute field (rhs) of a predicate using the mouse. Normally, the user wants to click, rather than type, a constant value for the value field or a variable for the attribute field of a predicate. However, we can not use a pop-up menu for this purpose. The number of choices for the value field is generally too large for a 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 in the above paragraph. Thus, if we just click values in the ANSWERTOOL, the selected value or attribute will appear as the rhs (value field) of the predicate.

Similarly some nested type queries can be simplified using ANSWERTOOL. The user first formulates for the inner most part of query and uses the ANSWERTOOL for building predicates in a piecemeal manner. Consider the query in Figure 23 again: "Find the banks containing customers whose balance is more than \$1000". Compare Figure 23 with Figure 19. Note that set operations can be written between ROGUE and ANSWERTOOL.

As shown in section 5.5, we allow the user to create new tuple variables and draw join expressions. Thus,

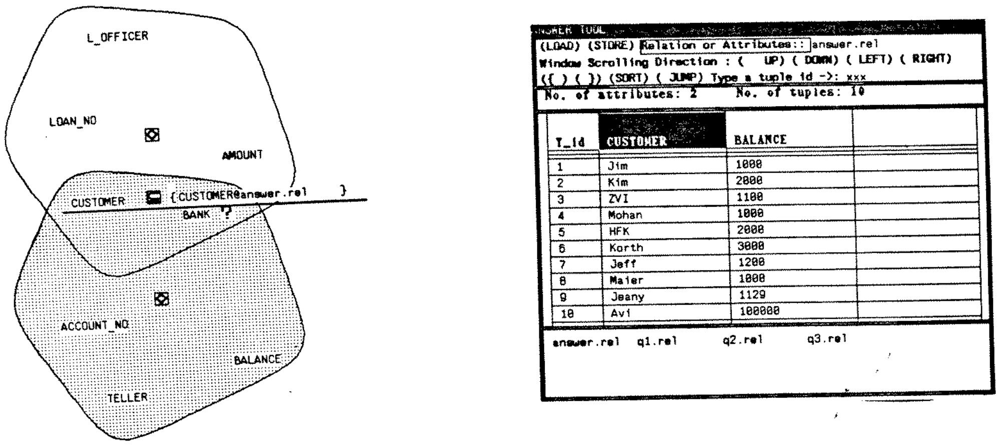


```
ANSWER TOOL
(LOAD) (STORE) Relation or Attributes:: answer.rel
Window Scrolling Direction : (   UP) ( DOWN) ( LEFT) ( RIGHT)
({ ) ( }) (SORT) ( JUMP) Type a tuple id ->: xxx
No. of attributes: 2     No. of tuples: 10
```

| 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 22: Predicate formulation using ANSWERTOOL



```
ANSWER TOOL
(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: A nest type query formulation using ANSWERTOOL

in PICASSO, the user can either draw a complex graphical query or resort to ANSWERTOOL. We believe that using ANSWERTOOL rather than drawing a complex query is more natural and easy to understand from user's point of view.

## 7. Graphical Feedback

There are circumstances when it is diffcult to formulate System/U queries. To illustrate this, suppose the user wants to pose the following queries under the bank database.

($N_1$) Find the customers who have either an account or a loan at BOA bank.

($N_2$) Find the customers who have saved more money than they have borrowed.

($N_3$) Find the teller whose customer's loan-no is L100.

($N_4$) Find the customers who have an account and a loan at BOA bank.

It is quite possible that the naive user will formulate the System/U queries for ($N_2$), ($N_3$) and ($N_4$) as the following incorrect System/U queries ($U_2$), ($U_3$) and ($U_4$) or other incorrect System/U queries. If the user is a non-technical person, we believe that only ($U_1$) can be formulated easily. For ($U_2$), ($U_3$) and ($U_4$) the user will get the error message like "No maximal object contains all the attributes you mentioned in the query".

($U_1$) *retrieve* CUSTOMER *where* BANK = BOA

($U_2$) *retrieve* CUSTOMER *where* BALANCE > AMOUNT

($U_3$) *retrieve* TELLER *where* LOAN_NO = "L100" and ... *the user may not proceed anymore*

($U_4$) *retrieve* CUSTOMER *where* BANK = "BOA" and ... *the user may not proceed anymore*

The correct System/U queries for ($N_1$), ($N_2$), ($N_3$) and ($N_4$) are ($C_1$),($C_2$), ($C_3$) and ($C_4$). Since the System/U queries ($C_2$), ($C_3$) and ($C_4$) involve multiple tuple variables and several joins, the naive user is unlikely to succeed in formulating them correctly on his first try. After receiving the error message for ($U_2$), ($U_3$) and ($U_4$) most of naive users may not catch their mistakes in their incorrect System/U queries. The difficulty is that System/U requires that the user be aware of the attributes' membership in maximal objects.

```
(C1) retrieve CUSTOMER
     where BANK = "BOA"
(C2) retrieve CUSTOMER
     where CUSTOMER = T.CUSTOMER and
                BALANCE > T.AMOUNT
(C3) retrieve TELLER
     where CUSTOMER = T.CUSTOMER and
            T.LOAN_NO = "L100"
(C4) 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
OR
    (retrieve CUSTOMER
     where BANK = "BOA" and
           ACCOUNT_NO = ACCOUNT_NO)
    INTERSECT
    (retrieve CUSTOMER
     where BANK = "BOA" and
           LOAN_NO = LOAN_NO)
```

In this section, we suggest several improved ways to grasp user's intention through graphical feedback and user's responses to them. In case of incorrect queries, if the incorrect queries are simply rejected and

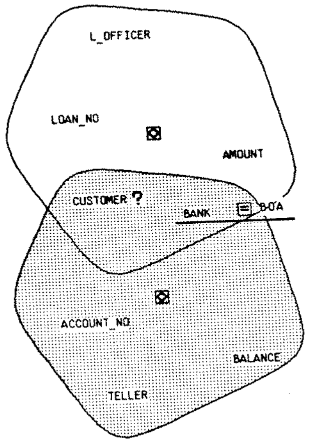
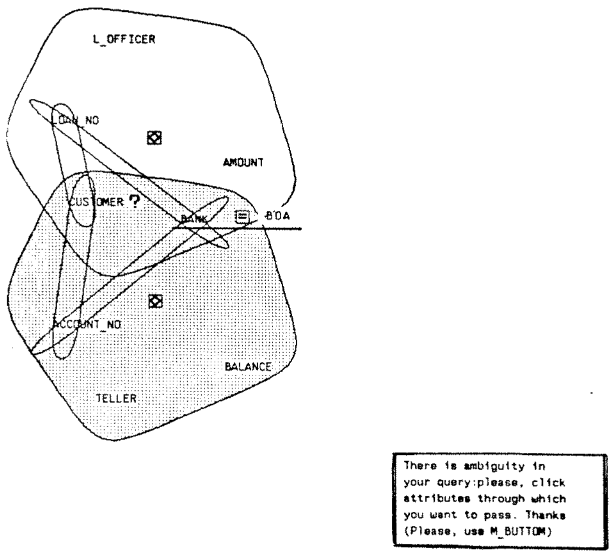


Figure 24: "Find the customer at BOA bank" by PICASSO



There is ambiguity in
your query:please, click
attributes through which
you want to pass. Thanks
(Please, use M_BUTTON)

Figure 25: A graphical feedback for an ambiguous query

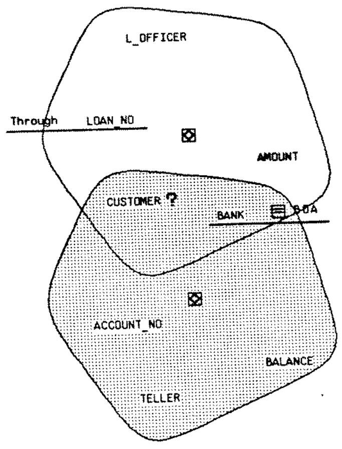


Figure 26: *Name dropping* in PICASSO

the user gets only error messages, the user would feel frustration and may not like to try again. Through graphical feedback, the ambiguous queries can be disambiguated and the erroneous queries can be corrected easily.

At any point during the specification of 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* clause). 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, we can not decide which maximal objects should be applied for selected attributes.

Consider the query in Figure 24: "Find the customers of BOA bank". The System/U interpretation is "Find all customers who have either an account or a loan at BOA bank" like $(N_1)$. In fact, it is not always correct. If the user has the intention of $(N_1)$, the PICASSO query would be formulated as shown in Figure 24. However, 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 problems, the system shows graphical feedback which is helpful in clarifying user's intention. After formulating the PICASSO query in Figure 24, if the user asks the system "RUN your Query" using the basic menu, the graphical feedback as shown in Figure 25 pops up and the system waits for the user's response. The cyclic structure (consists of 4 objects: each eclipse is one object) among CUSTOMER, BANK, LOAN_NO, and ACCOUNT is displayed and the system asks the user choose the path through which the user wants to pass. It is users' responsibility to decide which way to navigate.

Under our graphical language, PICASSO, the user (a loan officer) simply clicks the loan attribute for adding it into the mention set. Then a preposition "Through" is prefixed at the clicked attribute as shown in Figure 26. Internally, the tautology "Loan = Loan" would be formulated automatically. Now, the meaning of the query in the screen is "Find customers who have a loan at the BOA bank". The function of this predicate is to avoid the ambiguity of a graphical query with specifying access path. The idea is called *name dropping* [MW82].

Graphical feedback is useful in case of erroneous queries [Kor84]. Consider again the query ($N_3$) "Find a teller whose customer's loan_no is L100". Suppose the user poses the PICASSO query for the query ($N_3$) in Figure 27. This PICASSO query is supposed to be rejected under the universal relation semantics (since the System/U query for this PICASSO query is ($U_3$)). In fact, this query is not incorrect because a teller can have a customer and the loan_no for the customer can be L100. If the user is skillful enough to pose the query ($C_3$), the System/U would accept the query.

We believe that if the philosophy of the universal relation is really allowing the user to pose queries with only the knowledge of attributes' names, rather than rejecting the query in Figure 27, the system should find the connection paths and inform the user the possible ways of connecting the attributes which are located in different maximal objects.

In our system, ROGUE gives graphical feedback for this query and explains there are two ways of connecting TELLER and LOAN_NO. See Figure 28. 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 an tautology like "CUSTOMER = CUSTOMER". Its internal representation is "CUSTOMER = T.CUSTOMER". Since the TELLER and the LOAN_NO are located in different maximal objects, the TELLER attribute is prefixed by blank tuple variable and the LOAN_NO attribute is prefixed by tuple variable T. The details of System/U query formulation from a PICASSO query are described in [Kim85]. Figure 29 shows a correct query for ($U_3$). The final translated System/U query becomes ($C_3$).

As for the query ($N_4$), the user can build the PICASSO query of ($N_4$) with help of ANSWERTOOL.

There are two types of help messages in ROGUE. One is from the message subwindow (in the section 2 and the other type is the pop_up message which emerges around the current location of mouse. Our strategy for using the above two types of help message is as follows: (1) If the message is routine and unimportant such as the information of system status, greetings, etc, we use the message subwindow for the message. (2) 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 ususally concentrates on the position of mouse cursor. The pop-up type messages can make sure the user to be alert.

## 8. Summary

### *8.1 Completeness of PICASSO*

PICASSO is relationally complete. To see this, note that we can save temporary relations using the *store* option of ANSWERTOOL as shown in Figure 28. Each of the basic algebra operations are represented as follows. We can refer to subsection 5.7 to show how to compute a cartesian product. The two set operations union and set difference are well illustrated in section 5.8 and Figure 18. The selection and projection operators are obviously supported because PICASSO's target language is the System/U query language whose basic construct is SELECT-WHERE type. Hence PICASSO is relationally complete.

### *8.2 What's new in PICASSO*

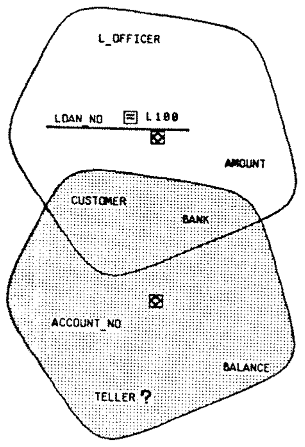
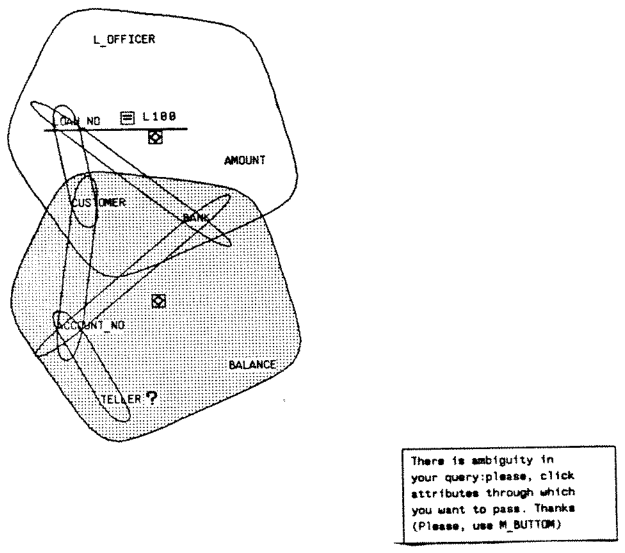


Figure 27: An erroneous query



There is ambiguity in your query:please, click attributes through which you want to pass. Thanks (Please, use M_BUTTON)

Figure 28: A graphical feedback for an erroneous query

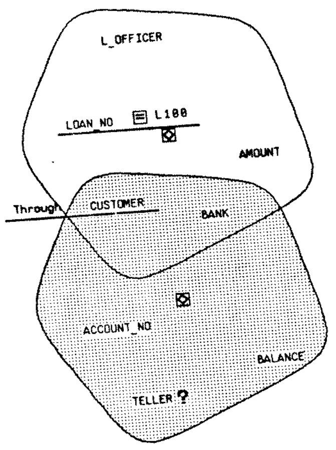


Figure 29: A corrected query for ($U_3$)

In general, we say that 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. Through the new graphical language PICASSO, we remove some artificial constraints in query formulation and help users to represent their thought naturally and graphically.

• Showing the hypergraph representation of database schema to users will help them to formulate correct queries and pose complex queries in a very natural way which can reflect users' thought processes.

• Eliminating character-type tuple variables is another notable feature in PICASSO. Naive users do not have to learn the concept of tuple variables. PICASSO can support multiple tuple variables for complex queries.

• A small, but useful tool ANSWERTOOL was introduced. Queries involving joins or nesting can be formulated easily with help of ANSWERTOOL. ANSWERTOOL can be used successfully as a facility for constructing a complex query in a piecemeal fashion.

• 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.

• As for the problem of dealing with large schemes, see [Kim85]. The basic idea is that the user can organize the screen freely in the SCREEN MOVE mode. SCREEN MOVE is one of options 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.

## 9. Conclusion

We defined a graphical query language PICASSO which is integrated into our graphical interface ROGUE. Through various examples queries, we showed the natural aspects of hypergraph data model and the powerful expressive power 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* lets the user easily pose their graphical queries. Also nested type queries can be easily formulated with help of ANSWERTOOL.

We believe that our graphical query language is easier to use and learn for the following four reasons.

a. Users can formulate their queries using a pointing device pop-up menus which appear in a syntax directed way.
b. The notions of tuple variable, join operation, and nested-type query are of less important for typical users due to the power of ANSWERTOOL.
c. Human factor issues are considered carefully. We relax artificial constraints on the order in which parts of query are entered, eliminating usage of meaningless tuple variables. We provide undo mechanisms for graphical query editing and avoid nested pop-up menus which might confuse the user.
d. The target language into which graphical query is transformed is System/U query language which is easier to use and learn than conventional relational query languages.

Since professional workstations have become popular in academia and industry, the graphical approach in data manipulation like PICASSO would enrich the applications on conventional database systems.

## 10. Bibliography

[AC75] Astrahan, M.M. and D.D. Chamberlin, "Implementation of a structured English query language", *Comm. ACM 18:10* 1976.

[CA80] Catell, R.G.G., "An Entity-based Database User Interface", *ACM, Proceedings of SIGMOD 80*, 1981.

[CH81] Chang, N.S., "Picture Query Language for Pictorial Database system", *IEEE Computer* November, 1981.

[FMU82] Fagin, R., A.O. Mendelzon, and J.D. Ullman, "A Simplified Universal Relational Assumption and Its properties," *ACM Transactions on Database Systems, 7:3* (September 1982), 343-360.

[FO84] Fogg, D., "Lessons from 'Living In a Database' graphical query", *ACM, Proceedings of SIGMOD 84*, 1984.

[HE80] Herot, C.F., "Spatial Management of Data", *ACM Transactions on Database Systems*, December, 1980

[Kel82] Keller, A., "Updates to Relational Databases through Views involving Joins," *Improving Usability and Responsiveness* , June 1982.

[KeU84] Keller, A.M. and J.D. Ullman, "On Complementary and Independent Mappings on Databases", *ACM, Proceedings of SIGMOD 84*, 1984.

[KeW84] Keller, A. and M.W. Willkins, "Approaches for Updating Databases with Incomplete Information and Nulls," *IEEE CS COMPDEC*, April 1984.

[Kim85] Kim, H.J. "GRAPHICAL ENVIRONMENTS FOR QUERY PROCESSING," *Master Thesis*, The University of Texas at Austin, August 1985.

[KKU84] Korth, H., G. Kuper, J. Feigenbaum, A. Van Gelder and J.D. Ullman, "System/U: A Database System Based on the Universal Relation Assumption," *ACM Transactions on Database Systems*, Vol. 9, No. 3, ACM, 1984.

[KL82] Klug, A., "ABE: a query language for constructing aggregates by example", *Proceedings of Workshop on statistical database management*, 1982.

[KoL85] Korth, H. and K.H. Lou, "Efficent algorithms for graph-theoretical problems in Hypergraph," *In preparation*, The University of Texas at Austin.

[Kor84] Korth, H.F., "Graphical Query Languages for Universal Relation Database Systems", *Internal Memo, The University of Texas at Austin*, 1984.

[KS84] Korth, H.F. 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.

[LP77] Lapczak, O.L., "An interactive Graphical Facility for Interacting with LSL", *MSc thesis, University of Toronto*, 1977.

[LT82] Lochovsky, F.H. and D.C. Tsichritzis, "An Interactive Query Language for External Databases", *Proceedings of International Conference on Very Large Data Base*, 1982.

[LW84] Larson, J.A. and J.B. Wallick, "An interface for novice and infrequent database management system users", *Proceedings of National Computer Conference*, 1984.

[MC81] Moura, C.M.O. and M.A. Casanova, "Design-By-Example", *Department of Informatics, PONTIFICIA UNIVERSIDADE CATOLICA RIO DE JANEIRO, BRASIL*, April, 1981.

[McA74] McDonald, N. and M. Stonebraker, "CUPID: The Friendly Query Language", *Technical Report: The University of California, Berkeley*, October, 1974.

[McB75] McDonald, N., "CUPID: A Graphic Oriented facility for support of non programmer interactions with a database", *UC Berkeley, ERL-M563*, November, 1975.

[MRW83] Maier, D., D. Rozenstein, and D.S. Warren, "Windows on the world", *ACM, Proceedings of SIGMOD 83*, May, 1983.

[MU83] Maier, D. and J.D. Ullman, "Maximal Objects and the Semantics of Universal Relational Databases", *ACM Transactions on Database Systems*, March, 1983.

[MW82] Maier, D. and D.S. Warren, "Specifying connections for a universal relation scheme database", *ACM, Proceedings of SIGMOD 82*, June, 1982.

[RoK85] Roth, M., Korth, H.F. and D.S. Batory, "SQL/NF query language", *The University of Texas at Austin*, August, 1985.

[RKS84] Roth, M.,H. Korth and A. Silberschatz, "Theory of non-1NF relational database," *TR-84-36* The University of Texas at Austin.

[ST76] Stonebraker, M. and et al., "The design and implementation of INGRES", *ACM Transactions on Database Systems, 1:3* 1976.

[ST82] Stonebraker, M. and G. Kalash, "TIMBER: A Sophisticated Relation Browser", *UC Berkeley, ERL M81/94*, May, 1982.

[Ull82] Ullman, J.D., "Principles of Database Systems," *Computer Science Press*, Rockville, MD., 1982.

[WI80] Wilson, G.A. and C.F. Herot, "SEMANTICS vs. GRAPHICS - To show or not show", *Proceedings of International Conference on Very Large Data Base*, 1980.

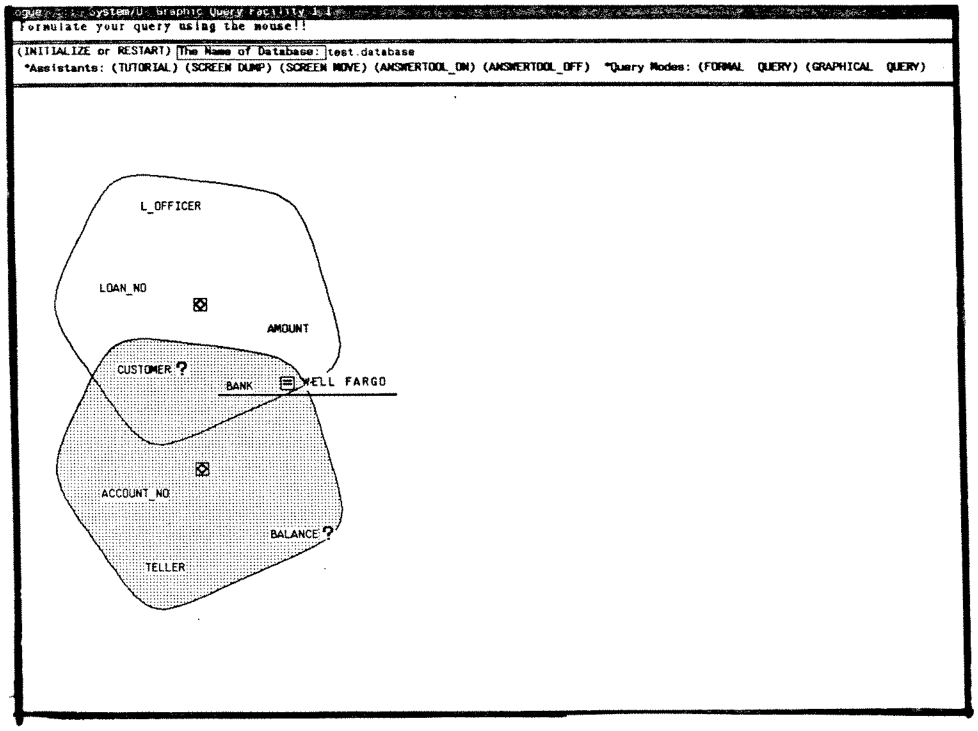
[WK82] Wong, H.K.T. and I. Kuo, "GUIDE: Graphical User Interface for Database Exploration", *International Conference on Very Large Data Base*, 1982.

[ZM83] Zhang, Z. and A.O. Mendelzon., "A Graphical Query Language for Entity Relationship Databases", *An E-R approach to Software Engineering, North Hollands*, 1983.

[Zl81] Zloof, M.M., "QBE/OBE: a language for office and business automation", *IEEE Computer*, May, 1981.

[Zl77] Zloof, M.M., "Query-by-Example: a data base language," *IBM system journal*, April 1977,IBM.

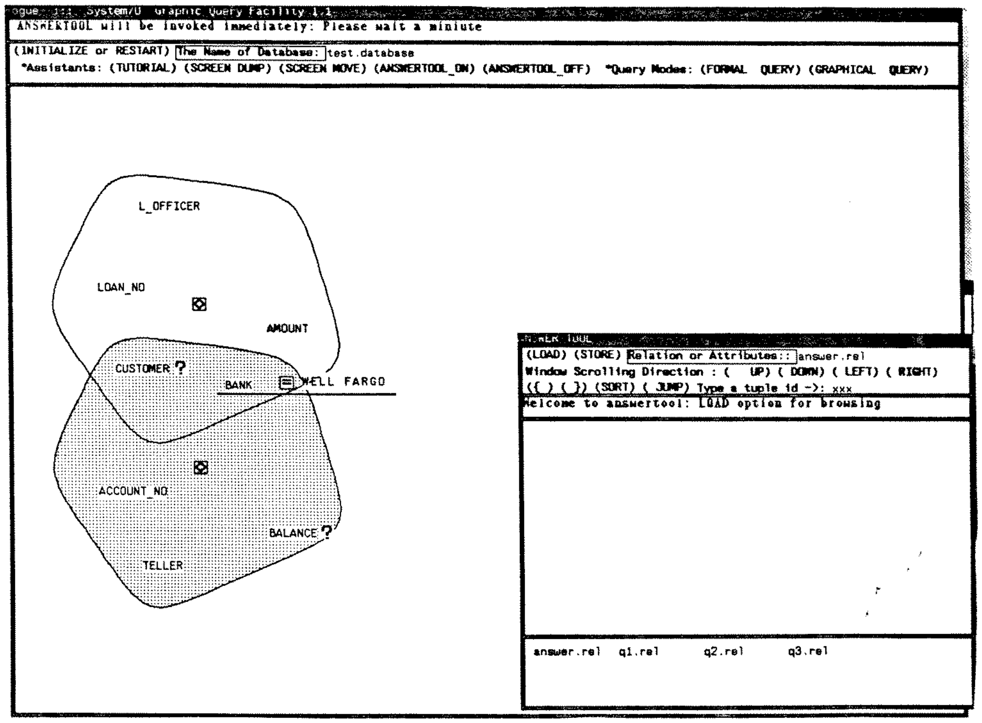
## 11. Appendix: an example of piecemeal query formulation



```
ogue System/U Graphic Query Facility 1.1
Formulate your query using the mouse!!
(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)
L_OFFICER
LOAN_NO
AMOUNT
CUSTOMER ?
BANK
WELL FARGO
ACCOUNT_NO
BALANCE ?
TELLER

Figure A.1: Sub query q1 "Find customers of WELL FARGO bank and their balances"



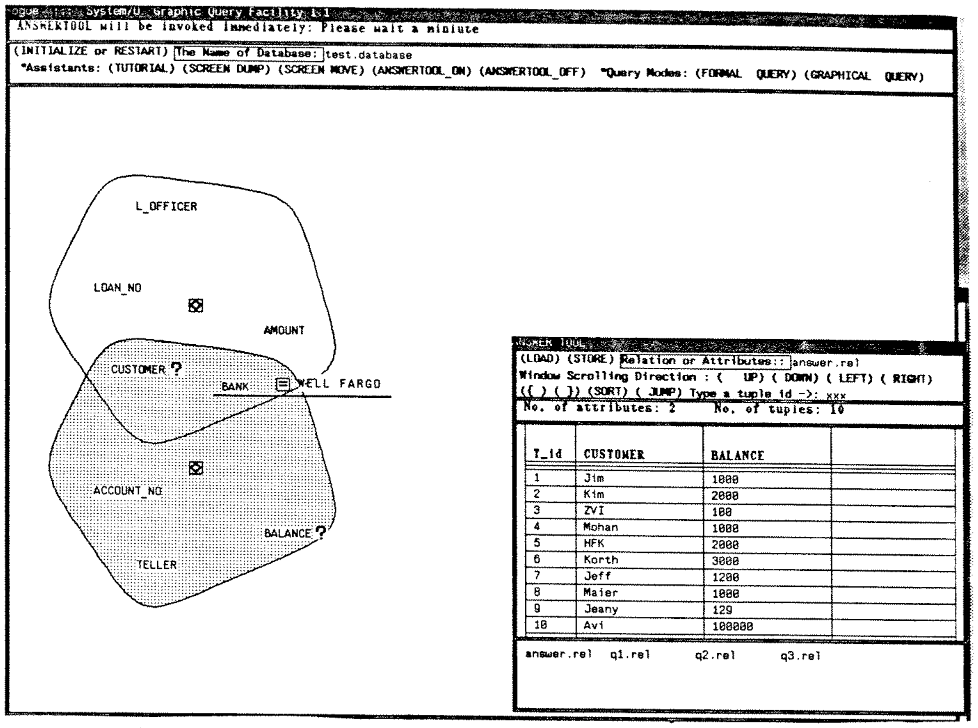
```
ogue System/U Graphic Query Facility 1.1
ANSWERTOOL will be invoked immediately: Please wait a miniute
(INITIALIZE or RESTART) The Name of Database: test.database

(LOAD) (STORE) Relation or Attributes:: answer.rel
Type a tuple id ->: xxx
Welcome to answertool: LOAD option for browsing

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

*Assistants: (TUTORIAL) (SCREEN DUMP) (SCREEN MOVE) (ANSWERTOOL_ON) (ANSWERTOOL_OFF)
*Query Modes: (FORMAL QUERY) (GRAPHICAL QUERY)
L_OFFICER
LOAN_NO
AMOUNT
CUSTOMER ?
BANK
WELL FARGO
ACCOUNT_NO
BALANCE ?
TELLER
Window Scrolling Direction : ( UP) ( DOWN) ( LEFT) ( RIGHT)
({ ) ( }) (SORT) ( JUMP)

Figure A.2: The user invokes ANSWERTOOL by clicking "ANSWERTOOL_ON" option in the option subwindow of ROGUE



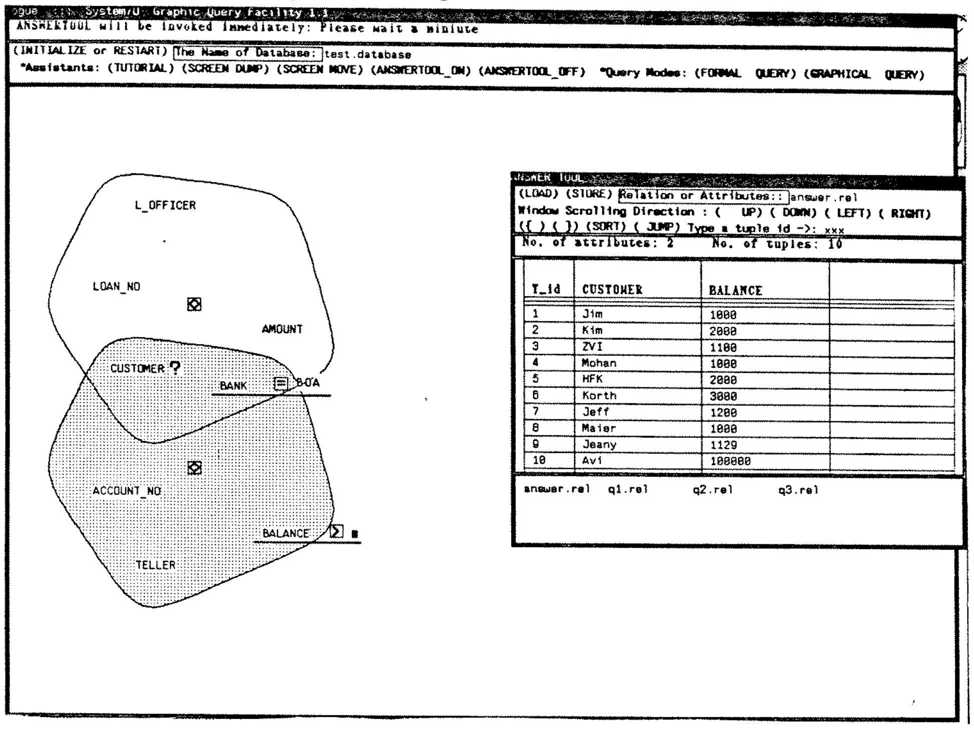
```
ogue System/U  Graphic Query Facility 1.1
ANSWERTOOL will be invoked Immediately: Please wait a miniute
(INITIALIZE or RESTART) The Name of Database: test.database
*Assistants: (TUTORIAL) (SCREEN DUMP) (SCREEN MOVE) (ANSWERTOOL_ON) (ANSWERTOOL_OFF)  *Query Modes: (FORMAL  QUERY) (GRAPHICAL  QUERY)

ANSWER TOOL
(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 | 100 | |
| 4 | Mohan | 1000 | |
| 5 | HFK | 2000 | |
| 6 | Korth | 3000 | |
| 7 | Jeff | 1200 | |
| 8 | Maier | 1000 | |
| 9 | Jeany | 129 | |
| 10 | Avi | 100000 | |

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

Figure A.3: The user looks at the contents of the query result by "LOAD" option in ANSWERTOOL



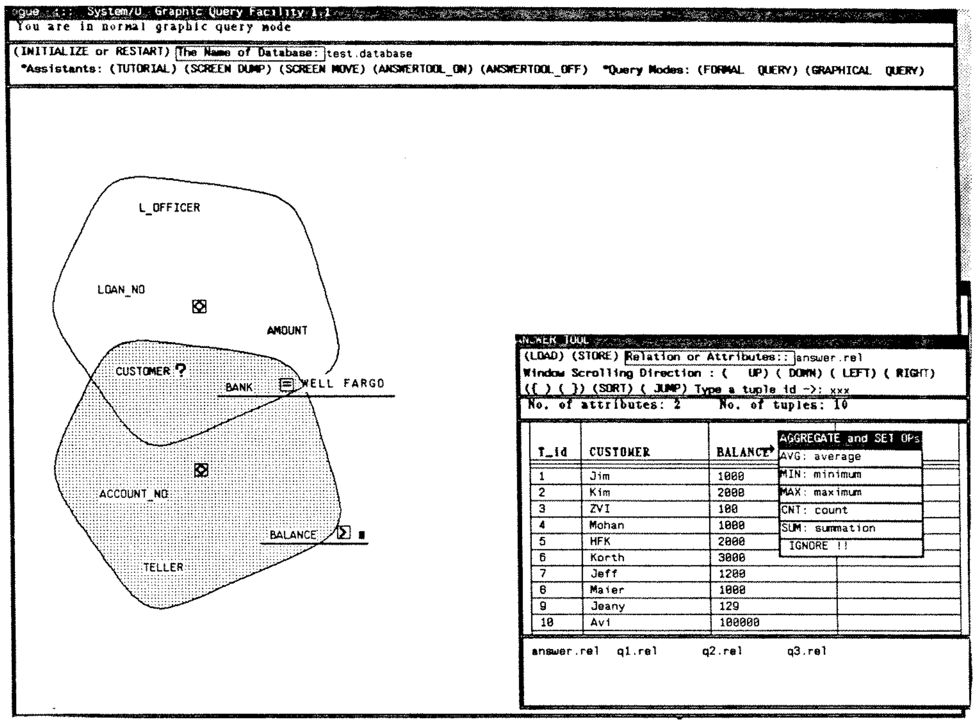
```
ogue System/U  Graphic Query Facility 1.1
ANSWERTOOL will be invoked Immediately: Please wait a miniute
(INITIALIZE or RESTART) The Name of Database: test.database
*Assistants: (TUTORIAL) (SCREEN DUMP) (SCREEN MOVE) (ANSWERTOOL_ON) (ANSWERTOOL_OFF)  *Query Modes: (FORMAL  QUERY) (GRAPHICAL  QUERY)

ANSWER TOOL
(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 A.4: After cleaning up the screen, the user selects Customer attribute and makes the predicate "BANK = BOA" and the template for predicate "Balance is more than the average balance of WELL FARGO bank"

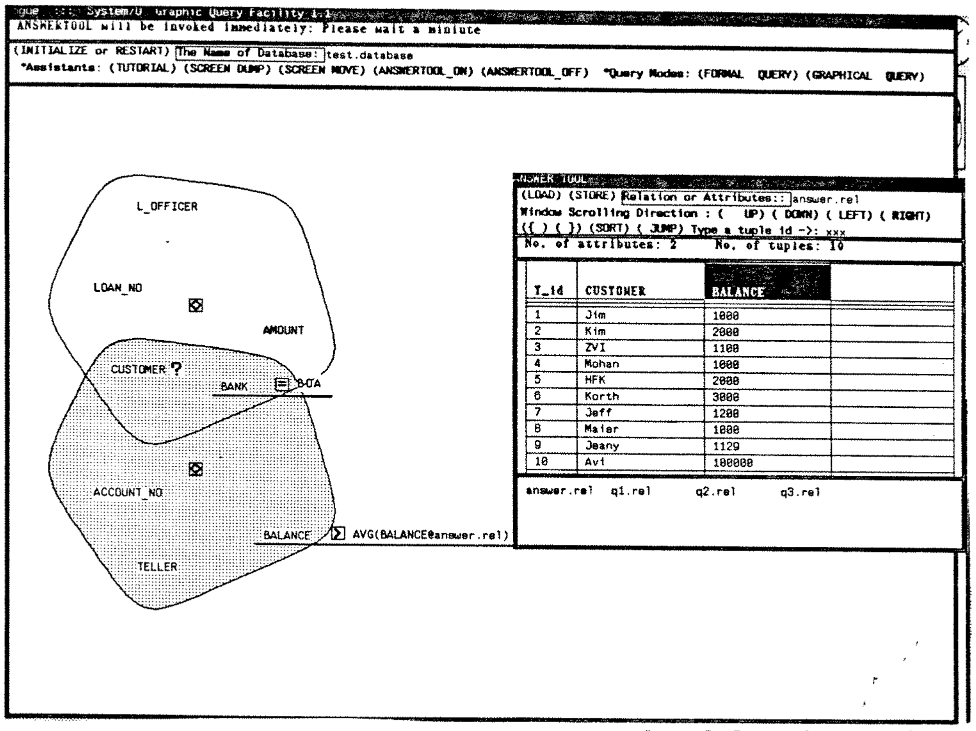


```
ogue ::: System/U Graphic Query Facility 1.1
You are in normal graphic query mode
(INITIALIZE or RESTART) The Name of Database: test.database
 *Assistants: (TUTORIAL) (SCREEN DUMP) (SCREEN MOVE) (ANSWERTOOL_ON) (ANSWERTOOL_OFF)  *Query Modes: (FORMAL  QUERY) (GRAPHICAL  QUERY)
```

```
ANSWER TOOL
(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 | 100 |
| 4 | Mohan | 1000 |
| 5 | HFK | 2000 |
| 6 | Korth | 3000 |
| 7 | Jeff | 1200 |
| 8 | Maier | 1000 |
| 9 | Jeany | 129 |
| 10 | Avi | 100000 |

Figure A.5: In order to fill in the rhs of the predicate, the user clicks clicks the Balance attribute in ANSWERTOOL and menu for aggregate functions pops up and the user chooses "AVG" item



```
ogue ::: System/U Graphic Query Facility 1.1
ANSWERTOOL will be invoked immediately: Please wait a minute
(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)
```

```
ANSWER TOOL
(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 |

Figure A.6: Sub query q2 "Find the customers whose balance is more than the average of balance of the relation in ANSWERTOOL" is formulated