RTMS: Toward Close Integration between Database and Application

Craig Thompson
Steve Corey
M. Rajinikanth
Prasanta Bose
Steve Martin
Rajiv Enand
Rick Roberts

Computer Science Lab.
Texas Instruments Inc.
P. O. Box 226015 M/S 238
Dallas, TX 75266

Abstract

This paper describes RTMS, a relational database management system, designed and implemented to realize the objective of long term management of data in a Lisp environment. Our goal was to add a database capability for MIT-derived Lisp Machines so that users could save, restore, and manipulate their persistent data objects. RTMS has a strong foundation upon which it has expanded to satisfy the needs of widely varying applications such as real-time databases, spatial databases and information retrieval. Its usefulness is enhanced by the advanced level of integration with systems such as a graphics window system and a natural language menu system. Additional noteworthy features of RTMS include a cooperative response system, an interactive interface, and a knowledge-based query processing front-end. Currently, RTMS runs on TI Explorer Lisp machines.

Introduction

Early on in the TI Explorer software development, we recognized that one of our important problems would involve long term management of data in a lisp environment. The two conventional ways to provide long-term, persistent storage for a user's data structures are to map those data structures to files or to store them in a database. File systems present on lisp machines provide only low level primitives for manipulating file content. Our goal became to add a database capability for the lisp machines so that users could save, restore, and manipulate their persistent data objects. RTMS, a Relational Table Management System, was implemented to realize this goal.

Functionally, RTMS implements most standard relational database capabilities. Data definition functions define, rename, or destroy databases, relations, views, attributes or secondary indices. There is no restriction on relation degree or cardinality. A relational algebra includes operators RETRIEVE (select and project), JOIN, UNION, DIFFERENCE, and INTERSECTION as well as operators INSERT, DELETE, MODIFY, GRANT and REVOKE. Certain basic features of most relational databases are not present in today's RTMS: full secure protection is not yet supported on data resident on lisp machines: no locking scheme protects concurrent access by multiple users or processes: finally, tables must be in virtual space to be operated on. These restrictions will be lifted as RTMS matures and, while the system is not at present a full DBMS, it is nevertheless useful.

Conventional relational databases [STON76] provide the same sort of functionality as described for RTMS but usually in an environment where the only primitive

data types are fixed length character strings, numbers, and dates, where the only functionality available is contained in the data manipulation language, where interfaces to host languages are awkward to use, where secure programs can guarantee protection, and where much attention is spent in matching record and index structure to page layout to minimize disk access emphasizing efficiency at the expense of functionality.

Abstract Data Types and User Defined Data Types

Lisp users routinely build complicated data structures from several primitives including atoms, lists, arrays, flavors, etc. In a lisp environment, users would find a relational database less useful if it failed to provide support for lisp data types. Hence, RTMS allows users to use any domains they want to define. Users can define domain dependent functions for validating the data type of domain elements inserted into relations, for domain-dependent partial ordering functions (eg GREATER-THAN-IN-LENGTH), and for flattening and restructuring complex structures so that they can be saved to disk. Default domains are also supported to allow users to ignore the details of domain definition in simple applications. Any type of data object may be stored in a relation including pointers, lists, flavor instances, arrays, variables, file names, relation names, graphical objects and large amounts of text. In the pure relational model, components of relations are atomic, but RTMS recognizes that atomicity is application dependent and does not enforce artificial restrictions requiring flattening complex data structures.

One Language Environment

The lisp environment is a one language environment. The relational algebra of RTMS is implemented with lisp functions so it is easy to embed in lisp programs. The ability to embed database calls in programs in a one language environment makes passing parameters between the database operators and other lisp functions straightforward, requiring no specialized mechanism. Within RTMS, names of databases, relations, and attributes follow the same rules as do names of Lisp variables. In addition, it is natural to allow lisp predicates in the WHERE clause of RETRIEVE statements and as the predicate in theta JOINs. For instance, if a user defines a predicate KEEP-TUPLEP which displays a tuple and allows him to mouse KEEP TUPLE or REJECT TUPLE choices, then he can include KEEP-TUPLEP at the end of a WHERE clause s-expression, allowing him to select or reject most tuples automatically based on the first part of the WHERE clause and manually select from the remaining tuples as in:

```
(RETRIEVE from 'EMPLOYEES into 'MY-EMPLOYEES
          WHERE (AND (GT age 30) (OR (GT salary 50K) (KEEP-TUPLEP))))
```

Accessor Functions

A user with other favorite data structures than those presently provided by RTMS can define them to RTMS by defining a set of accessor functions. These functions include RETRIEVE-[IMP]-[STO], INSERT-[IMP]-[STO], and several others. These functions serve to isolate the implementation level of RTMS from the functionality the user sees. Not much work is involved in defining these accessors since their basic job is to provide a means of registering a foreign data structure's database equivalent operations for traversing and modifying the structure. One example of the use of this facility is the ability to describe to RTMS the correspondence of [loaded function, location in source file] that is already maintained in a system data structure on lisp machines to facilitate the development of a code management system.

Attach and Detach Data

RTMS comes with a predefined collection of useful tuple implementations including tuples implemented as linked lists, structures, and flavor instances. In addition, RTMS provides a predefined collection of storage structures including balanced tree, hash, and heap useful for building indices. A user can specify the tuple implementation and storage structure when defining a relation.

Traditional databases maintain an "inside-outside" distinction. Either data is managed by the database or it is outside of database control. If an application maintains its own data but needs some database operations on it, in traditional databases, the data must be copied into the database, operated on and recopied to the application. RTMS provides an operations called ATTACH-RELATION, which allows a user-defined data structure that matches one of the tuple implementation-storage structure implementations to be grafted into RTMS as a database relation without recopying thus allowing a user easy access to RTMS query functionality including the ability to build other indices on the data structure to provide alternative access paths. A corresponding DETACH-RELATION operation allows a user to remove a database structure from the database without destroying it.

Optimization

The added flexibility offered by RTMS must be accompanied by a concern for efficiency. Optimization problems occur when users are given ready access to lisp. In operators like RETRIEVE and JOIN, allowing user-defined functions can hide from RTMS the opportunity to use defined indices making it necessary to loop through all tuples in a relation executing the user-defined function. For instance, in the query

```
(RETRIEVE   from   'R
            WHERE  (EQU-STRING-LENGTH  A  10)).
```

if the EQU-STRING-LENGTH predicate is not registered with RTMS, the EQU-STRING-LENGTH predicate must be executed for each tuple in R. But a framework for admitting optimization hints to RTMS can allow the system to retain the ability to optimize while allowing extra flexibility. RTMS handles optimization AND, OR, EQ, GT, LT, and other commonly used functions are supported. Thus, in the WHERE clause,

```
(AND (LT age 40) (OR (GT age 30) (KEEP-TUPLEP)))).
```

if an ordered index on AGE exists, only tuples corresponding to employees under 40 years old need be considered, all tuples between 30 and 40 must be returned, and KEEP-TUPLEP must be executed on all remaining tuples. The optimization capability is extensible; users can register a definition for OPT-[STO]-[OPT-FN] with RTMS to inform RTMS of the opportunity to use a supported storage structure in implementing a new function, as when one defines OPT-TREE-STRING-LENGTH to tell RTMS to use a TREE structured index ordered by string length to to locate strings of length 10 in the example above.

Interactive Interface

While it is expected that RTMS will be utilized mostly by the experienced user from an application program, RTMS still provides an interactive interface to aid the (novice or expert) interactive user. The RTMS interactive interface, shown in figure 3, consists of two windows. The bottom of the these windows is an enhanced Lisp listener. Here, the user may perform any Lisp or RTMS interaction desired and have access to all of the capabilities of this system feature, including command

completion and function help. The larger of the these windows is the output window, in which all of the interaction initiated while in the interface is maintained. This window is scrollable allowing the user to review previous interactions. Also, all database object names, in this window are mouse selectable. This means that the mouse may be made to point at one of these names at which time the left button may be pressed. This will cause help to be requested on the object.

The user may type any RTMS or Lisp function in the input window or the RTMS command menu may be used to enter an RTMS function. This menu contains all of the RTMS commands (see figure 2). These commands may be selected by pointing to the function desired and clicking the left mouse button. This will cause another selection window to be displayed which contains all of the keywords and parameters associated with the selected function.

An example of this is provided for the RETRIEVE function in figure 3. The user may enter the values desired and then execute the function call without having to know the syntax. This provides a method for a novice user to execute all RTMS functions without having to understand the syntax. However, the more experienced user will more likely use the Lisp Listener part of the interface, both may be accommodated in the interface while still providing the session preserving features of the output window.

Menu-based Natural Language Interface

Recently, a new approach for implementing natural language interfaces has emerged. Menu-based natural language interfaces ([TENN83a], [TENN83b], [THOM83]) guide a user in the completion of a query and constrain him to a limited but expressive natural language subset. The basic idea is that instead of typing natural language in an unconstrained way, a user selects words and phrases from a constellation of menus to complete a sentence. A semantically-constrained grammar drives the menu display so that a user's input is guaranteed to be both grammatically understood and semantically meaningful to the system. The approach has the advantage that all inputs to the system are understood, thus giving a zero failure rate. We have implemented an NLMENU interface to RTMS.

Figure 4 shows a sample NLMenu screen interfacing to a small University database stored in RTMS. The user constructs a natural language query (in this case, in English) in the sentence window, the central empty window in the figure, from constituents that he selects from the active menus above. Menus with highlighted backgrounds are active. Other menus are temporarily inactive. The grammar of the COURSES interface controls which lexical item are reachable from the current sentence fragment and paints windows "active" if they contain legal next choices

(narrowing the choices accordingly) making other windows "inactive". The NL-Menu guided approach restricts the user to the defined language in just the right way, so that he becomes aware of the features of the language but cannot go beyond it. Learning time is greatly reduced since it takes no experience to begin to state legal sentences in a new limited language subset and relatively little experience to formulate queries in interfaces with regular coverage.

Cooperative Responses

Several extensions have been made to RTMS to incorporate cooperative response. Cooperative response is an attempt to provide the user with an answer that is more useful than the literal result of a query. Three types of cooperative response have or will be added to RTMS:

1. **Corrective Indirect Response**: *Identify the point of failure in the query and describe it to the user.*

```
(RETRIEVE 'part WHERE '(and (= color red) (> weight 50)))

Traditional response: nil
Cooperative response: There are no parts whose weight is
                      greater than 50.
```

2. **Supportive Indirect Response**: *Return all attribute values mentioned in the query.*

```
(RETRIEVE 'part WHERE '(and (= color red)
                            (in-shipments WHERE (> quantity 200))

Traditional response:

    Part#  Color Weight
    -------------------
    P#1      Red     40
    P#4      Red     20

Cooperative response:

    Part#  Color Weight Shipment# Quantity
    -------------------------------------
```

```
P#1     Red     40      S#4     320
P#4     Red     20      S#2     210
```

3. **Suggestive Indirect Response:** *Suggest an alternative query based on the content (not yet implemented) of the database.*

```
(RETRIEVE 'part WHERE (and (= color red) (> weight 100))

Traditional Response: nil

Cooperative Response: The heaviest red part has a weight of 80,
                      would you like to see it?
```

With an NLMenu interface to an RTMS database, the addition of cooperative response improves the user interface. Eventually one would like to get the same kind of response from such a system as one would expect from a person answering questions. The addition of Cooperative response to RTMS is a step in that direction.

Real-time Data Management using RTMS

Presently, the emphasis in real-time systems is focused on the control aspect. Data has been treated in an ad hoc manner and specialized software has had to be written to accommodate the needs of every application. As these systems become more sophisticated, there is an increasing demand for the creation of standard hardware and software tools to operate in this environment. A database management system which has extended real-time capabilities would be such a desired tool. If standard database packages can be made to suit real-time requirements, sophisticated systems will be created with much less software investment.

The relational model, with it's flexibility, provides many of the desired features of relational systems such as high-level queries, fast and more standard code development, high portability, and simpler views of the data. However, it lacks several of the requirements for real-time systems. The following extensions are identified in ENAN85 as needed to move towards a more standard real-time data model.

- user-defined data types and operators

- o support for block updates and attach/detach
- o sharing of database process memory
- o a triggering mechanism to support an event-driven system
- o a caching system to save non-volatile intermediate results when queries are re-executed
- o levels of query compilation
- o complex object queries

We have demonstrated RTMS support for the first three requirements. In one of our applications, RTMS was used to store "real-time" data (the location of a plane) along with more static data. RTMS data structures were shared with the application so that the application could update "real-time" fields directly.

Spatial Data Management with RTMS

In another application of abstract data types in RTMS, a spatial database capability was demonstrated using a database that contained graphics objects as well as other thematic data [THOM85a]. General purpose relational databases have not been used to manipulate graphical information with the same ease as numbers and character strings. Adding this capability to RTMS leads to a powerful pictorial database capability which permitted queries like:

EX Map boundary and label of cities intersected by interstate I35 and whose population is greater than 100,000 (Figure 5)

EX Map boundary and label of highways that pass through Dallas and that pass through Houston (Figure 6)

The Explorer graphics editor GED [EXPL85] supports a primitive data structure called a polyline. GED polylines were used to represent features like road and city boundaries on a map. Operators on polylines were written to compute whether two polylines intersect, whether one is contained in another, the area of closed polylines, the perimeter of polylines, polylines west of a point, and so on. Finally, a complex map was decomposed into sets of polylines representing cities, states, roads, ships, and air routes, and corresponding RTMS tables were built. Now, a user wanting to *"find the interstate highways in Texas that go through Dallas"* could join the relations HIGHWAY (*name, type, polyline, ...*) and CITY (*name, population, polyline*) using the graphics operator POLYLINE-INTERSECTP as the JOIN

predicate. Extensions were made to the menu-based natural language Database Interface Generator to allow NLMenu interfaces to be easily generated to spatial database applications.

Information Retrieval using RTMS

In many applications, formatted database fields contain complicated textual descriptions. Although DBMS and IR systems are very similar structurally, no systems integrate both sorts of capabilities. We prototyped an integrated DBMS and IR system so that both kinds of capabilities are available in a uniform query language [THOM85b]. The capability was implemented in RTMS by first defining a new data type TEXT (a variable-length string). Then, a new predicate MATCH was defined on TEXT objects such that

```
(MATCH boolean-query text1 ... textN) --> boolean
```

MATCH returns true if the boolean-query matches any of the remaining text arguments. This extension was enough so that a user could then use mixed DBMS and IR qualifications in queries, as in:

```
(RETRIEVE FROM 'ship
    WHERE (AND (EQ type 'submarine)
       (MATCH "nuclear OR missile" description)))
```

which finds submarines in the ship table that have the words "nuclear" or "missile" in their descriptions. The next step was to build a KWIC index capability for RTMS so that users could build secondary IR indexes to support IR queries more efficiently. The KWIC index implemented was a hash structured index that associates a list of [record, field, offset] triples with each distinct word appearing in the "attributes" fields of tuples in the table. Finally, the query optimization component of RTMS had to be informed of when to use the IR index if it were present, by defining function OPT-KWIC-HASH. Using almost the same approach, graphical features of picture objects could be computed and picture data could be indexed by feature.

Knowledge-based Query Processing

KARMA [BOSE85] is an expert database system that uses RTMS as a repository of data. Currently, an intelligent user-interface to RTMS and knowledge-based

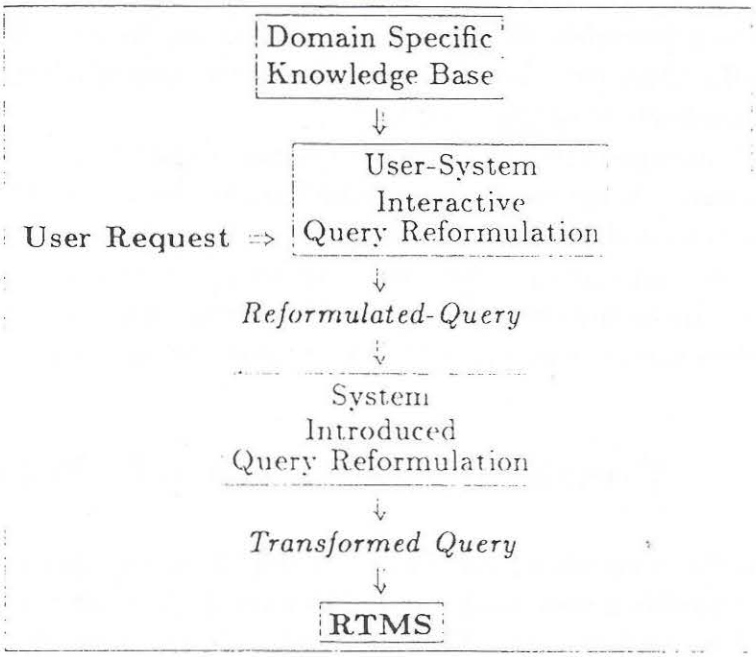


Figure 1: *KARMA Architecture*

query optimization are implemented. The objectives of KARMA include extending the optimization, providing deductive capabilities on top of RTMS, caching query results, and handling incomplete information in knowledge and data bases. The functional architecture of KARMA is shown in figure 1.

The intelligent interface is designed to aid non-expert users in formulating queries at a conceptual level and guiding them towards the relevant information through a reformulation process. It is based on the paradigm of retrieval by reformulation [TOU82] It can be of great use to users of complex databases as well as the naive users. The basic idea behind this paradigm is that, often the user makes a query to retrieve data from a database by starting with a very vague notion of what is it that he wants and even more vague idea of what kind of query is needed to get that data. Hence the system can help the user in this retrieval process by letting him incrementally construct a partial description of the data he is searching for.

To assist users in retrieving the appropriate data from RTMS, KARMA maintains a representation of the current partial description of the query and an example individual that satisfies the query. Information from these representations as well as the knowledge-base representing the concepts particular to the query domain, can

be utilized to reformulate the query in an interactive fashion. By successive reformulations, the users, who may have a limited knowledge of the underlying database schema, can retrieve what they want.

KARMA interprets the reformulated queries based on the knowledge about the database domain. It represents intensional data in the form of integrity constraints and general rules and uses them to maintain the consistency of the data, to improve the query execution and to provide deductive reasoning capabilities. The query interpretation process involves transforming the query into an executable and efficient form and then using the database system to retrieve the data [RAJI85].

Conclusions and Future Plans

In sum, the interesting contribution of RTMS is to make the functionality of a relational database more easily extendable by the user and to be well integrated with the lisp environment. In RTMS, the ability to add user-defined domains, search predicates and indices makes RTMS more useful in a lisp environment. A user seeking efficiency can restrict himself to supported data types, search predicates, and access paths. A user seeking flexibility can more easily drop into lisp to build more complicated search strategies. If those search strategies are important for his application, he has a route to add a more efficient implementation to RTMS.

Our goal is to make RTMS more useful: more general, more powerful, more intelligent, and more efficient. Spatial database capability will be extended by introducing caching and triggering mechanisms. Future work will concentrate on allowing complex databases to be modeled, providing inference and reasoning capabilities through integration of RTMS with an assumption-based reason maintenance system. We are considering a scheme for supporting a wider variety of persistent data types beyond only supporting files and tables. A general scheme would make user-defined data structures persistent, removing the current need to force long-lived data structures into tables or files. Such a scheme would remove, in a general way, the restriction that RTMS relations would need to be copied and flattened to be saved.

References

[BOSE85] P. K. Bose and M. Rajinikanth, "KARMA : Knowledge-based Assistant to a Database System", To Appear in the Proceedings of IEEE Conference on Artificial Intelligence Applications, December, 1985, Miami.

[ENAN85] R. Enand and C. Thompson, "Toward a Real-Time Data Model", International Industrial Controls Conference and Exposition/Controls West, Long Beach, CA, September 16-18, 1985.

[EXPL85] Explorer Graphics Editor Manual, Texas Instruments.

[RAJI85] M. Rajinikanth and P. K. Bose, "Knowledge-Based Approach To Database Query Processing", Proceedings of the AISB Conference, 1985.

[STON76] M. Stonebraker, E. Wong, P. Kreps, and G. Held, "The Design and Implementation of INGRES." ACM Transactions on Database Systems 1:3, pp:189-222.

[TENN83a] H. R. Tennant, K. M. Ross, R. M. Saenz, C. W. Thompson and J. R. Miller, " Menu-based Natural Language Understanding", Proc. of 21st ACL, MIT, June, 1983.

[TENN83b] H. R. Tennant, K. M. Ross, R. M. Saenz, "Usable Natural Language Interfaces Through Menu-Based Natural Language Understanding", Proceedings of the Conference on Human Factors in Computing Systems, Boston, Mass, December, 1983.

[THOM83] C. W. Thompson, H. Tennant, K. Ross and R. Saenz, "Building Usable Menu-Based Natural Language Interfaces to Databases", Proceedings of the 9th VLDB Conference, Florence, Italy, October, 1983.

[THOM85a] C. W. Thompson and S. Martin, "Asking Map- and Graph-Valued Queries Using a Menu-Based Natural Language Interface", 1985 ACM Annual Conference, Denver, Colorado, October 14-16, 1985.

[THOM85b] C. W. Thompson and S. Martin, "Using Menu-Based Natural Language to Query an Integrated Database Management and Information Retrieval System", TI-CSL Technical Report, February, 1985.

[TOU82] F. N. Tou, M. D. Williams, R. Fikes, A. Henderson, and T. Malone, "RABBIT: An Intelligent Database assistant," Proc. of AAAI-82, Pittsburgh, PA., pp: 314-318.

More Above

Relation : COURSE Database : UNIVERSITY-CATALOG Cardinality : 69.

DEPARTMENT	COURSE#	TITLE	CREDITS
CS	5970.	'INDEPENDENT STUDY IN CS'	1.
CS	5950.	'ADVANCED SMALL COMP -SYSTEMS'	3.
CS	5940.	'ADVANCED SMALL COMP -SYSTEMS'	3.
CS	5930.	'SPECIAL TOPICS IN CS'	1.
CS	5920.	'SPECIAL TOPICS IN CS'	1.
CS	5910.	'SPECIAL TOPICS IN CS'	1.
CS	5850.	'PATTERN RECOG'	3.
CS	5840.	'PATTERN RECOG'	3.
CS	5810.	'INFORMATION ORGANIZATION & RETRIEVAL'	3.
CS	5750.	'THEORY OF FORMAL LANGUAGES'	3.
CS	5730.	'COMPUTABILITY & COMPUTATIONAL COMPLEX'	3.
CS	5710.	'FINITE AUTOMATA THEORY'	3.
CS	5680.	'ADVANCED OPERATING SYSTEMS'	3.
CS	5675.	'NUMERICAL MATHEMATICS'	3.
CS	5670.	'ADVANCED OPERATING SYSTEMS'	3.
CS	5665.	'NUMERICAL MATHEMATICS'	3.
CS	5655.	'NUMERICAL MATHEMATICS'	3.
CS	5570.	'DATA BASE MANAGEMENT SYSTEMS'	3.
CS	5475.	'ADVANCED TOPICS IN NUM PART DIFF EQU'	3.
CS	5465.	'FINITE ELEMENT METHODS'	3.
CS	5455.	'FINITE DIFF METHODS FOR PART DIFF EQU'	3.
CS	5430.	'THEORY OF COMPILERS'	3.
CS	5250.	'MEDICAL COMPUTING'	3.
CS	5210.	'ARTIFICIAL INTELLIGENCE'	3.
CS	5050.	'COMP MODEL & SIMUL OF PHYSICAL -SYSTEMS'	3.
CS	5010.	'COMP ASSISTED INSTRUCTION'	3.
CS	5002.	'NON-THESIS GRAD COMPLETION'	3.
CS	5000.	'THESIS'	3.
CS	4990.	'SPECIAL TOPICS IN CS'	3.
CS	4980.	'SPECIAL TOPICS IN CS'	3.
CS			
CS			
CS			
CS			

OUTPUT

Definition	*Manipulation*	*Operators*	*Other Features*
Define Database	Delete Tuples	Retrieve Tuples	Print Relation
Define Relation	Modify Tuples	Join	Save Database
Define View	Modify Database	Union	Save Relation
Define Attribute	Modify Relation	Intersection	Save Environment
Define Environment	Modify Attribute	Difference	Save Transaction
Define Domain	Modify Domain	Select	Maptuple
Define Transaction	Modify Transaction	Project	Mapt
Define Implementation	Modify View	Commit Transaction	Active Database
Define Storage Structure	Destroy Database	Average	Environment Status
Attach Relation	Destroy Relation	Sum	Rename Attribute
Detach Relation	Destroy Attribute	Size	Rename Relation
Load Database	Destroy Domain	Count	Rename Database
Load Relation	Destroy Implementation	Maximum	Abort Transaction
Load Environment	Destroy Storage Structure	Minimum	Begin Transaction
Insert Tuples	Destroy View		End Transaction

(retrieve
ormat '(1
>

urse# desc) 'f

Rtms Inter

Display

Retrieve some tuples from a relation satisying a where clause. (RETRIEVE RELATION-NAME &REST KEYWORD-LIST &KEY &OPTIONAL DIRECTORY

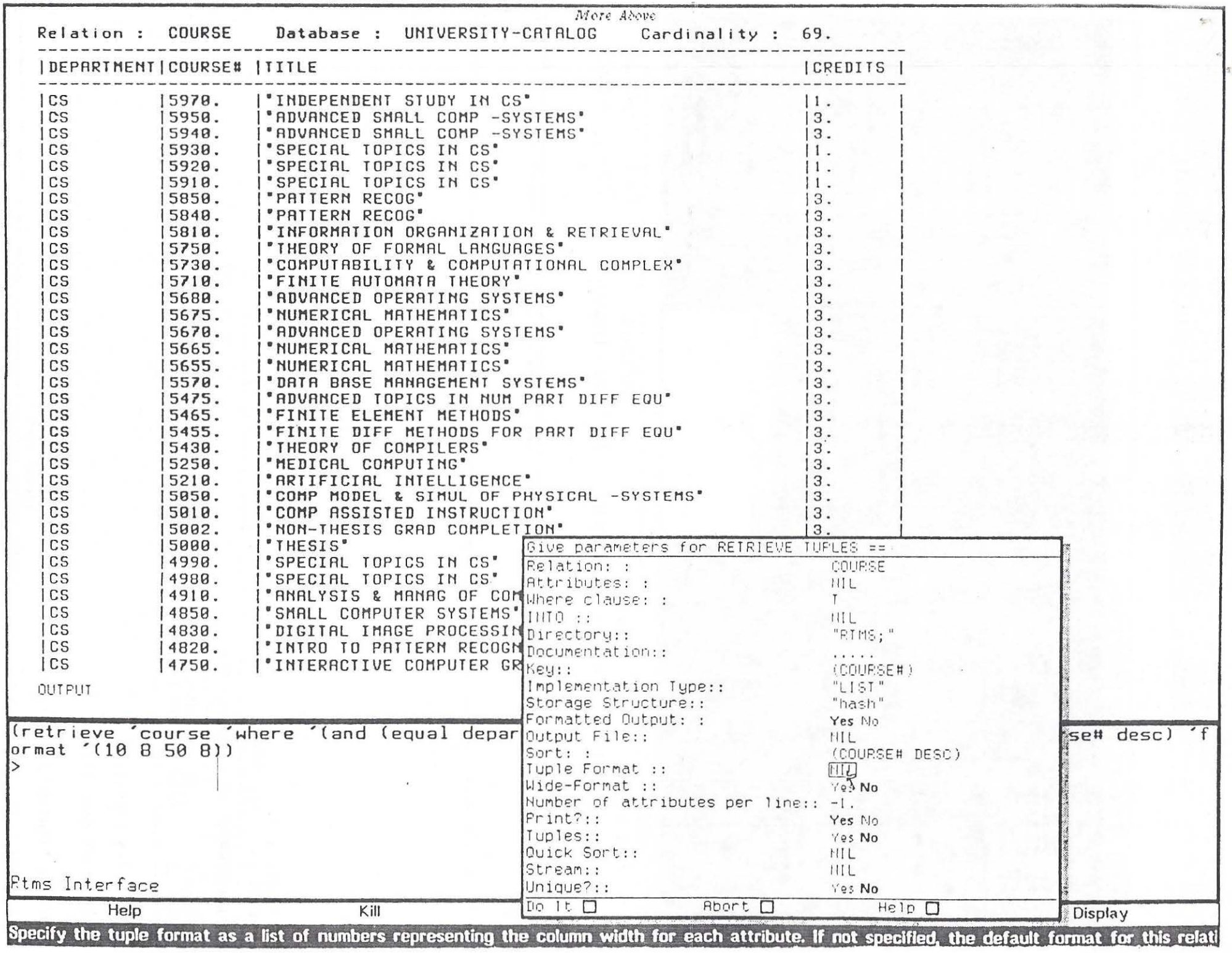
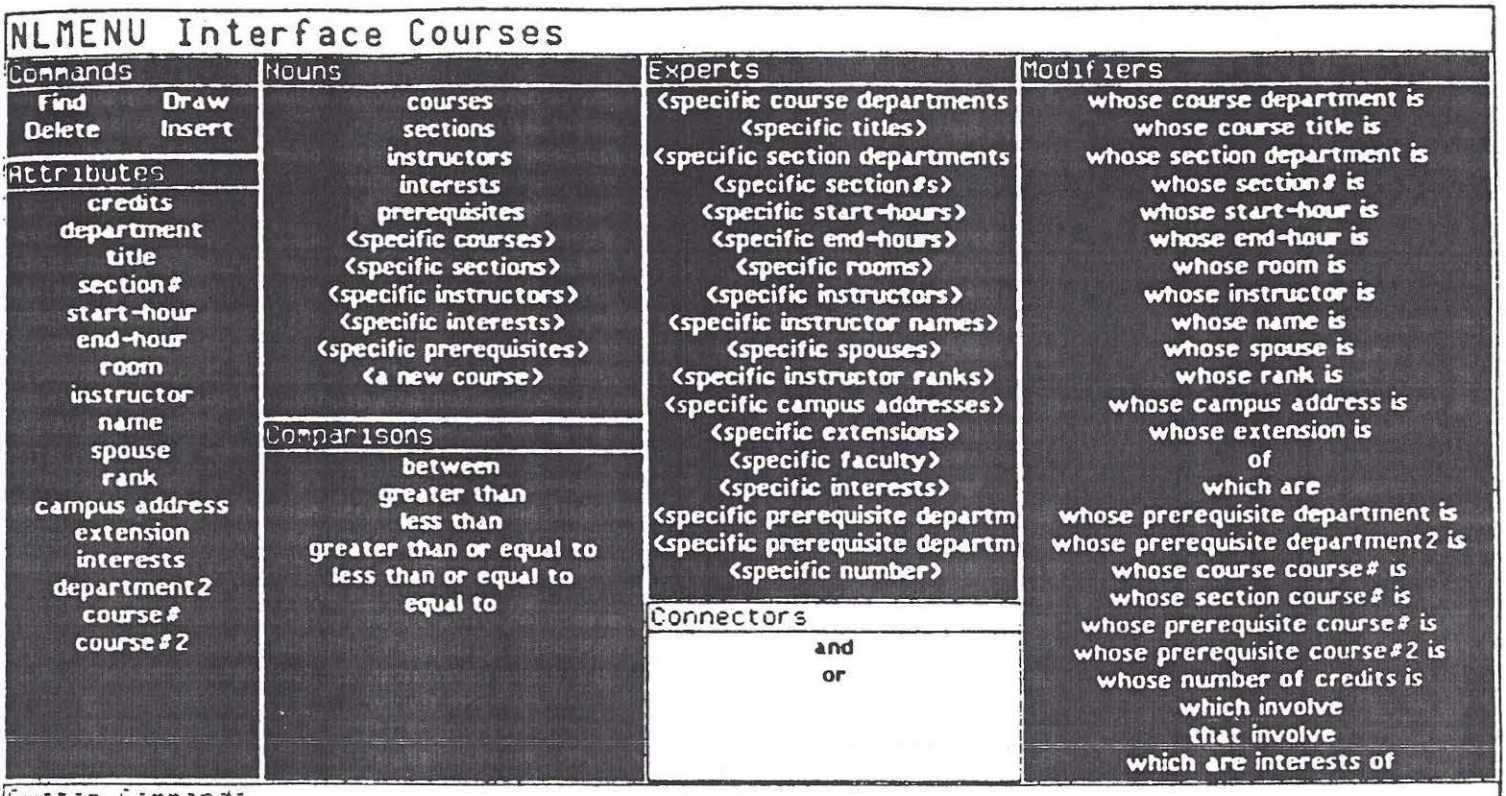
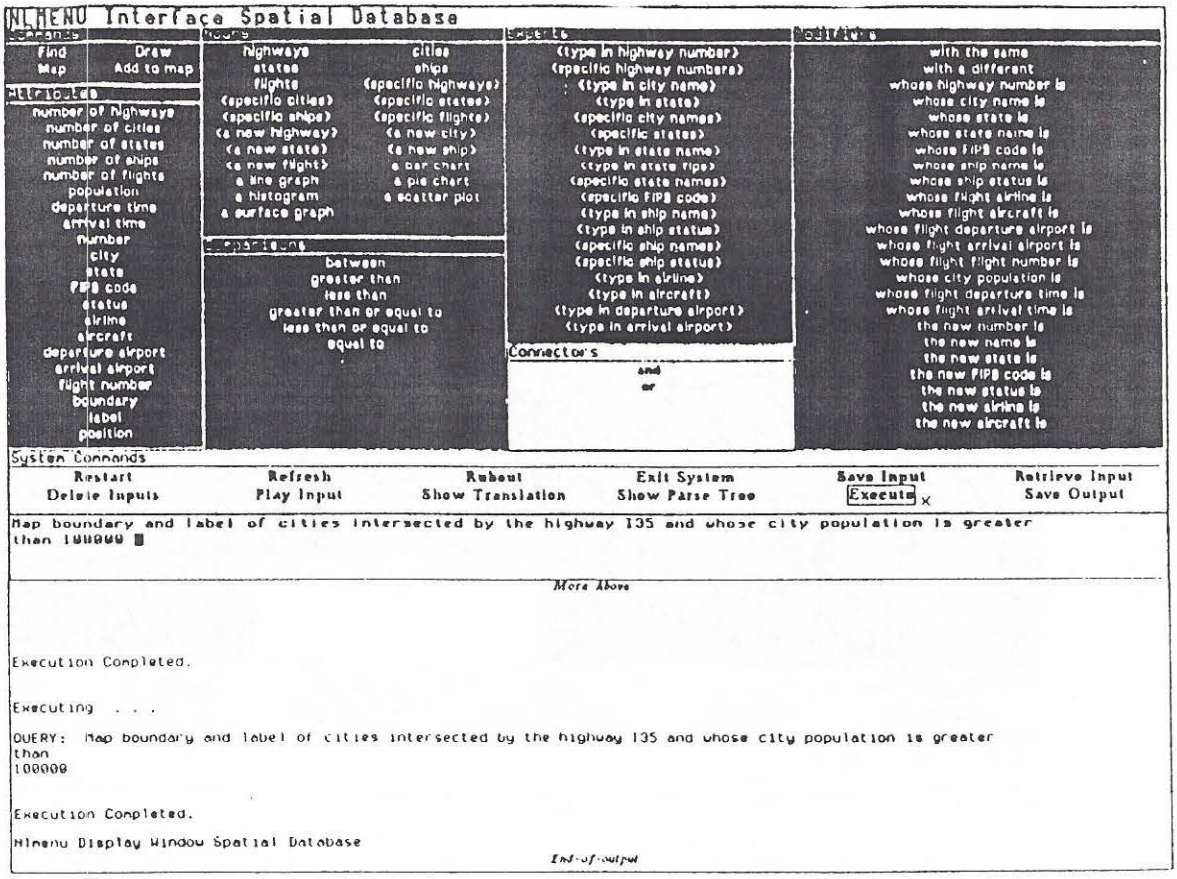
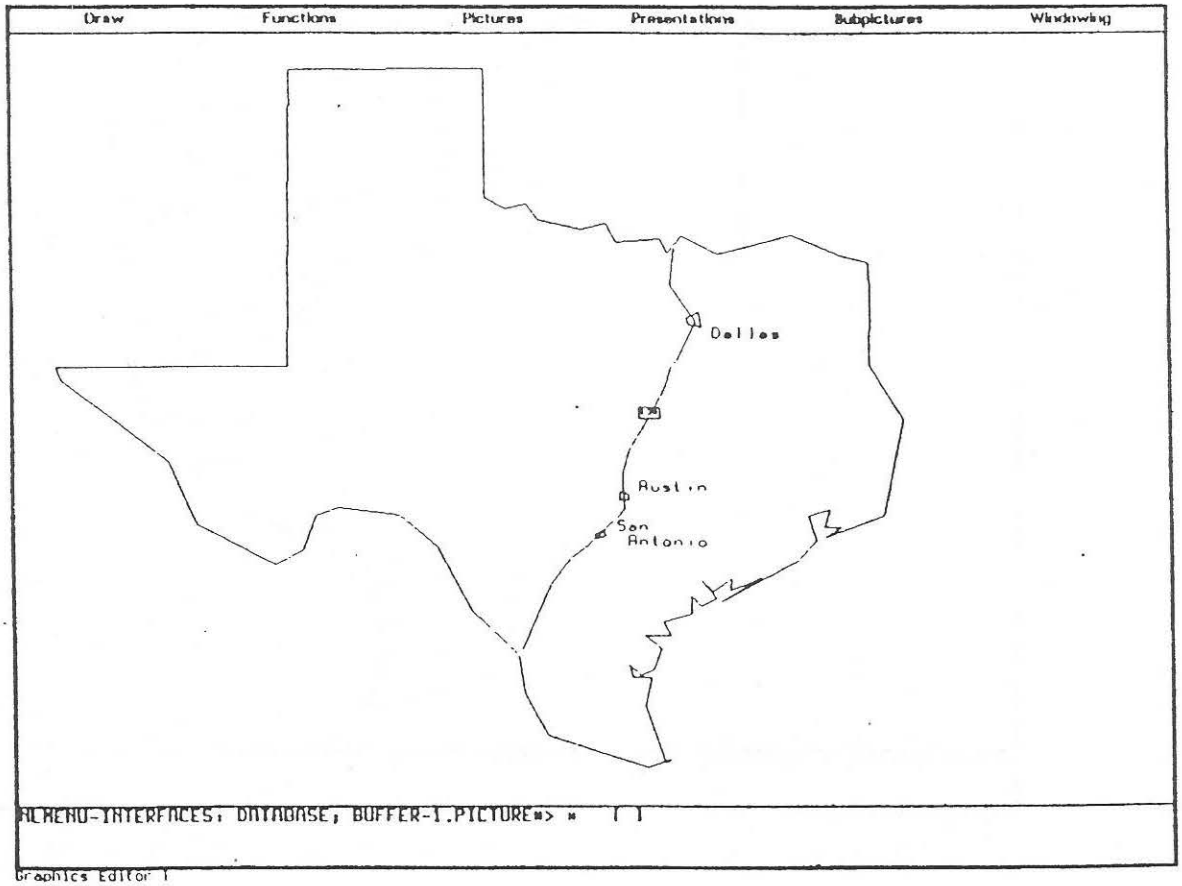
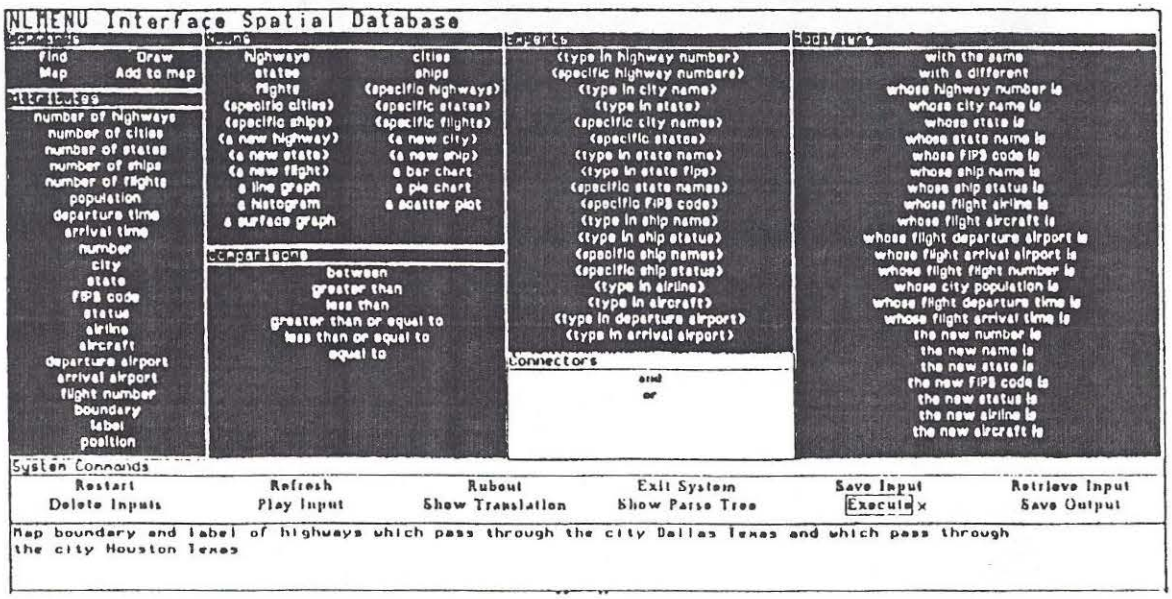

More Above
Relation : COURSE Database : UNIVERSITY-CATALOG Cardinality : 69.
DEPARTMENT	COURSE#	TITLE	CREDITS
CS	5970.	'INDEPENDENT STUDY IN CS'	1.
CS	5950.	'ADVANCED SMALL COMP -SYSTEMS'	3.
CS	5940.	'ADVANCED SMALL COMP -SYSTEMS'	3.
CS	5930.	'SPECIAL TOPICS IN CS'	1.
CS	5920.	'SPECIAL TOPICS IN CS'	1.
CS	5910.	'SPECIAL TOPICS IN CS'	1.
CS	5850.	'PATTERN RECOG'	3.
CS	5840.	'PATTERN RECOG'	3.
CS	5810.	'INFORMATION ORGANIZATION & RETRIEVAL'	3.
CS	5750.	'THEORY OF FORMAL LANGUAGES'	3.
CS	5730.	'COMPUTABILITY & COMPUTATIONAL COMPLEX'	3.
CS	5710.	'FINITE AUTOMATA THEORY'	3.
CS	5680.	'ADVANCED OPERATING SYSTEMS'	3.
CS	5675.	'NUMERICAL MATHEMATICS'	3.
CS	5670.	'ADVANCED OPERATING SYSTEMS'	3.
CS	5665.	'NUMERICAL MATHEMATICS'	3.
CS	5655.	'NUMERICAL MATHEMATICS'	3.
CS	5570.	'DATA BASE MANAGEMENT SYSTEMS'	3.
CS	5475.	'ADVANCED TOPICS IN NUM PART DIFF EQU'	3.
CS	5465.	'FINITE ELEMENT METHODS'	3.
CS	5455.	'FINITE DIFF METHODS FOR PART DIFF EQU'	3.
CS	5430.	'THEORY OF COMPILERS'	3.
CS	5250.	'MEDICAL COMPUTING'	3.
CS	5210.	'ARTIFICIAL INTELLIGENCE'	3.
CS	5050.	'COMP MODEL & SIMUL OF PHYSICAL -SYSTEMS'	3.
CS	5010.	'COMP ASSISTED INSTRUCTION'	3.
CS	5002.	'NON-THESIS GRAD COMPLETION'	3.
CS	5000.	'THESIS'
CS	4990.	'SPECIAL TOPICS IN CS'
CS	4980.	'SPECIAL TOPICS IN CS'
CS	4910.	'ANALYSIS & MANAG OF COM
CS	4850.	'SMALL COMPUTER SYSTEMS'
CS	4830.	'DIGITAL IMAGE PROCESSIN
CS	4820.	'INTRO TO PATTERN RECOGN
CS	4750.	'INTERACTIVE COMPUTER GR
OUTPUT
Give parameters for RETRIEVE TUPLES ==
Relation: : COURSE
Attributes: : NIL
Where clause: : T
INTO :: NIL
Directory:: "RTMS;"
Documentation:: .....
Key:: (COURSE#)
Implementation Type:: "LIST"
Storage Structure:: "hash"
Formatted Output: : Yes No
Output File:: NIL
Sort: : (COURSE# DESC)
Tuple Format :: NIL
Wide-Format :: Yes No
Number of attributes per line:: -1.
Print?:: Yes No
Tuples:: Yes No
Quick Sort:: NIL
Stream:: NIL
Unique?:: Yes No
Do It
Abort
Help
(retrieve 'course 'where '(and (equal depar
ormat '(10 8 50 8))
>
se# desc) 'f
Rtms Interface
Help
Kill
Display
Specify the tuple format as a list of numbers representing the column width for each attribute. If not specified, the default format for this relati

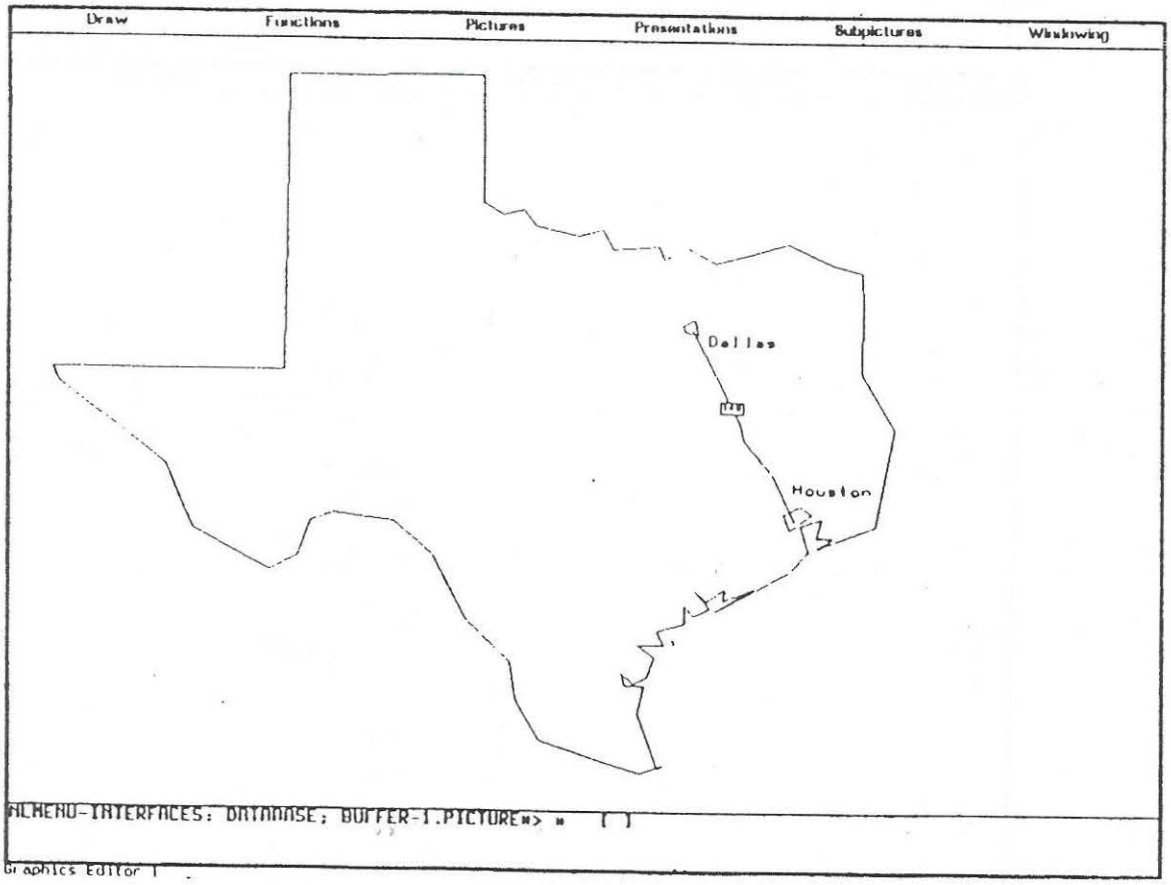
FIGURE 4: An NLMenu Interface to a University Database




NLMENU Interface Spatial Database
Commands
Find
Map
Draw
Add to map
Attributes
number of highways
number of cities
number of states
number of ships
number of flights
population
departure time
arrival time
number
city
state
FIPS code
status
airline
aircraft
departure airport
arrival airport
flight number
boundary
label
position
Nouns
highways
states
flights
(specific cities)
(specific ships)
(a new highway)
(a new state)
(a new flight)
a line graph
a histogram
a surface graph
cities
ships
(specific highways)
(specific states)
(specific flights)
(a new city)
(a new ship)
a bar chart
a pie chart
a scatter plot
Operations
between
greater than
less than
greater than or equal to
less than or equal to
equal to
Experts
(type in highway number)
(specific highway numbers)
(type in city name)
(type in state)
(specific city names)
(specific states)
(type in state name)
(type in state fips)
(specific state names)
(specific FIPS code)
(type in ship name)
(type in ship status)
(specific ship names)
(specific ship status)
(type in airline)
(type in aircraft)
(type in departure airport)
(type in arrival airport)
Connectors
and
or
Modifiers
with the same
with a different
whose highway number is
whose city name is
whose state is
whose state name is
whose FIPS code is
whose ship name is
whose ship status is
whose flight airline is
whose flight aircraft is
whose flight departure airport is
whose flight arrival airport is
whose flight flight number is
whose city population is
whose flight departure time is
whose flight arrival time is
the new number is
the new name is
the new state is
the new FIPS code is
the new status is
the new airline is
the new aircraft is
System Commands
Restart
Delete Inputs
Refresh
Play Input
Reboot
Show Translation
Exit System
Show Parse Tree
Save Input
Execute
Retrieve Input
Save Output
Map boundary and label of cities intersected by the highway 135 and whose city population is greater
than 100000
More Above
Execution Completed.
Executing . . .
QUERY: Map boundary and label of cities intersected by the highway 135 and whose city population is greater
than
100000
Execution Completed.
Nlmenu Display Window Spatial Database
End-of-output


Draw
Functions
Pictures
Presentations
Subpictures
Windowing
Dallas
135
Austin
San
Antonio
NLMENU-INTERFACES; DATABASE; BUFFER-1.PICTURE#> #
Graphics Editor 1


NLMENU Interface Spatial Database
Commands
find
Draw
Map
Add to map
Attributes
number of highways
number of cities
number of states
number of ships
number of flights
population
departure time
arrival time
number
city
state
FIPS code
status
airline
aircraft
departure airport
arrival airport
flight number
boundary
label
position
Nouns
highways
cities
states
ships
flights
(specific highways)
(specific cities)
(specific states)
(specific ships)
(specific flights)
(a new highway)
(a new city)
(a new state)
(a new ship)
(a new flight)
a bar chart
a line graph
a pie chart
a histogram
a scatter plot
a surface graph
Comparisons
between
greater than
less than
greater than or equal to
less than or equal to
equal to
Experts
(type in highway number)
(specific highway numbers)
(type in city name)
(type in state)
(specific city names)
(specific states)
(type in state name)
(type in state fips)
(specific state names)
(specific FIPS code)
(type in ship name)
(type in ship status)
(specific ship names)
(specific ship status)
(type in airline)
(type in aircraft)
(type in departure airport)
(type in arrival airport)
Connectors
and
or
Modifiers
with the same
with a different
whose highway number is
whose city name is
whose state is
whose state name is
whose FIPS code is
whose ship name is
whose ship status is
whose flight airline is
whose flight aircraft is
whose flight departure airport is
whose flight arrival airport is
whose flight flight number is
whose city population is
whose flight departure time is
whose flight arrival time is
the new number is
the new name is
the new state is
the new FIPS code is
the new status is
the new airline is
the new aircraft is
System Commands
Restart
Refresh
Rubout
Exit System
Save Input
Retrieve Input
Delete Inputs
Play Input
Show Translation
Show Parse Tree
Execute
Save Output
Map boundary and label of highways which pass through the city Dallas Texas and which pass through the city Houston Texas


Draw
Functions
Pictures
Presentations
Subpictures
Windowing
Dallas
Houston
NLMENU-INTERFACES; DATABASE; BUFFER-1.PICTURE#> #
Graphics Editor 1