SRI International

IDA: AN INTELLIGENT DATA ACCESS PROGRAM

Technical Note 145

June 1977

By: Daniel Sagalowicz
Computer Scientist
Artificial Intelligence Center

SRI International®

333 Ravenswood Ave. • Menlo Park, CA 94025
(415) 326-6200 • TWX: 910-373-2046 • Telex: 334-486

ABSTRACT

IDA was developed at SRI to allow a casual user to retrieve information from a data base, knowing the fields present in the data base, but not the structure of the data base itself. IDA is part of a system that allows the user to express queries in a restricted subset of English, about a data base of fourteen files stored on CCA's Datacomputer. IDA's input is a very simple, formal query language which is essentially a list of restrictions on fields and queries about fields, with no mention of the structure of the data base. It produces a series of DBMS queries, which are transmitted over the ARPA network. The results of these queries are combined by IDA to provide the answer to the user's query. In this paper, we will define the input language, and give examples of IDA's behavior. We will also present our representation of the "structural schema", which is the information needed by IDA to know how the data base is actually organised. We will give an idea of some of the heuristics which are used to produce a program in the language of the DBMS. Finally, we will discuss the limitations of this approach, as well as future research areas.

ACKNOWLEDGMENTS

The work described in this paper benefited from discussions with various members of the Artificial Intelligence Center at SRI International. Special mention should be given to Koichi Furukawa (now at ETL), Earl Sacerdoti, Jonathan Slocum, amd Michael Wilber. The research reported herein was supported by the Advanced Research Projects Agency of the Department of Defense under contract DAAG29-76-C-0012 with the U. S. Army Research Office.

IDA: AN INTELLIGENT DATA ACCESS PROGRAM

Daniel Sagalowicz
SRI International
Menlo Park, California

A. INTRODUCTION

This paper is concerned with one of the components of LADDER (for Language Access to Distributed Data with Error Recovery), a data base access system currently being developed at SRI [1]. The ultimate goal of this system is to provide decision makers with easy access to information stored in multiple computers, under various data base management systems (DBMSs). The need for such a system has been amply discussed in the literature--see, for example, the discussions in [2, 3, and 4]--and will not be elaborated here. The particular application for LADDER is as an aid to Navy decision makers, but the techniques being developed are likely to be applicable to a wide range of decision making activities.

The components of LADDER are shown in Figure 1. The first one INLAND (for Informal Natural Language Access to Navy Data) [5] allows the user to ask questions in English about information contained in data bases similar to those currently used by the Navy. Although the particular Navy

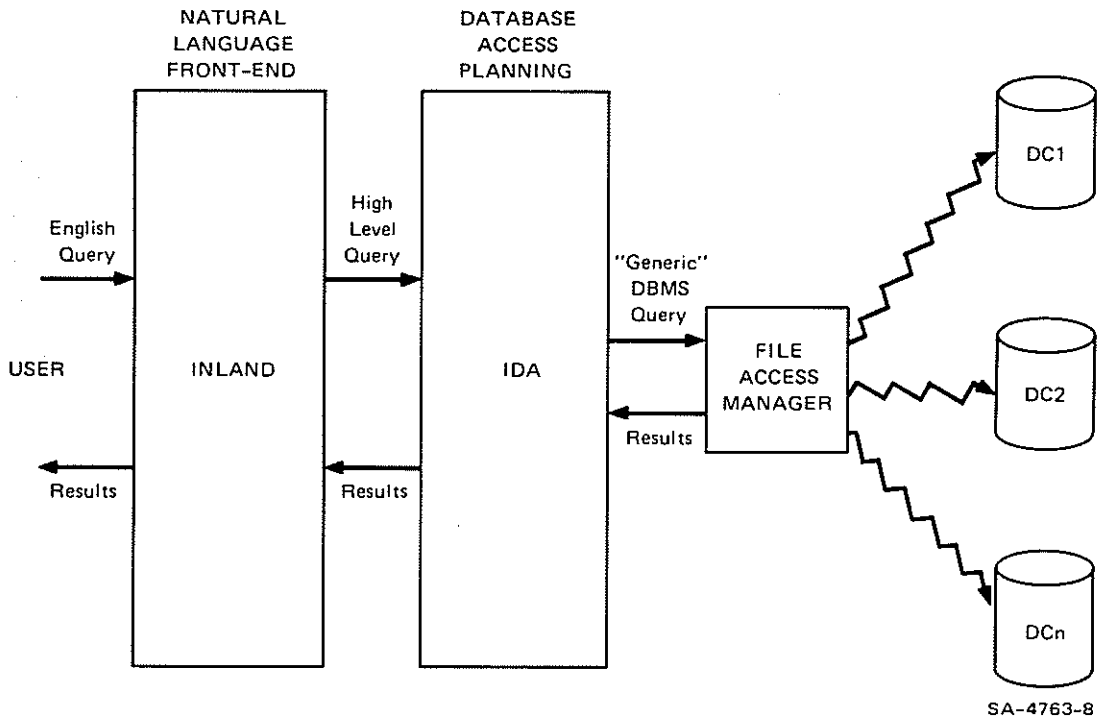
NATURAL
LANGUAGE
FRONT-END
DATABASE
ACCESS
PLANNING
English
Query
USER
INLAND
Results
High
Level
Query
Results
IDA
"Generic"
DBMS
Query
Results
FILE
ACCESS
MANAGER
DC1
DC2
DCn
SA-4763-8

FIGURE 1

terminology is encoded within INLAND, it is not provided with knowledge about how the information is organized in the data base. It knows nothing about the data base structure except the names of the fields. In particular, it does not know how the data base is subdivided into files and records, or where, within a computer network, those files are currently located. It therefore translates the query into a formal high level query that contains no mention of the data base structure, and this query is then passed to the next component of the system, IDA.

IDA (for Intelligent Data Access) contains a model of the data base structure in what we call a "structural schema." The structural schema describes the organization of the data base into records and files. IDA is used to generate a series of file-oriented queries from the single data base-oriented query provided by INLAND. However, IDA is unaware of the precise computer(s) on which the selected files are actually located. It therefore issues queries to the next component, called FAM (for File Access Manager), without specifying the computers or the directories where the files may be found. An additional problem arises when a file duplicated on several computers appears under different names on each machine. Therefore, in queries to FAM, the files selected by IDA are represented by a generic file name which stands for any of those duplicate files.

This final component, FAM [6], is responsible for establishing the connections over the ARPA network to the various DBMSs on the various computers, for finding the most up-to-date versions of the various files, for issuing the actual DBMS queries in which the actual file names replace the generic names used by IDA, and for recovering in case of certain types of errors.

In our current implementation, we use only one data base management system, the Datacomputer developed by the Computer Corporation of America [7,8], on several computers. During the coming months, we plan to extend IDA and FAM to be able to access a second DBMS.

Although IDA was developed as part of the LADDER system, it has also been used independently of the natural language front-end. In this paper, we will present IDA as if it were to be used without any language front-end. The main goal of IDA is to provide the user with a "structure free" interface to the data base. The user of IDA needs know only field names and, of course, the IDA's query language, and is not required to know the data base structure or to employ any other complex structure. The goals of IDA are similar to those of the APPLE system [4], and we refer the reader to that paper for a very clear and complete discussion of the need for such a system. Two other systems that are also intended to free the user from the need to know the real data base structure are INGRES [9] and SYSTEM-R [10]. However, neither of these systems provides an interface to the data base that is structure-free. Both allow their users to introduce a new

"logical view" of the data base which is different from the actual data base. Since the user can use only the predefined relations that belong to his logical view, he must have available a large set of predefined logical views if he is to access the data in a truly flexible manner. This means that, to use SYSTEM-R or INGRES, he must still acquire knowledge of a complex structure--in this case, the set of various logical views. When using IDA, on the other hand, the user's point of view is that, for each query, the whole data base is configured into only one relation, namely, the relation needed to satisfy his query--no matter what the query is--as long as the information is contained in the data base. To obtain the same result using the logical view technique of INGRES, the data base administrator would have to define all logical relations which can be obtained by taking the joins of two relations, the joins of three relations, and so on. Then, the DBMS would have to match the query with all those virtual relations to decide which one applies, and finally execute the query accordingly. In the particular data base that we have been using, which is composed of fourteen relations, some queries required the join of five relations. Even with such a small data base, the task of the data base administrator in setting up the logical views that would include all the joins of up to five relations would be tremendous. By contrast, IDA requires the data base administrator to define only the joins between any two relations in the data base--whenever such joins exist. Then, IDA decides dynamically which joins need to be performed to

satisfy the query. In effect, IDA decides dynamically what "logical view" corresponds to the query, and decides dynamically how to satisfy the query in that particular logical view.

As already mentioned, APPLE has goals which are very similar to those of IDA. APPLE attempts to use "paths" to describe all the possible joins and projections allowed in the data base. In APPLE, all the paths have been prestored statically and, at run time, only a choice between those prestored paths is allowed. In IDA, however, the path to be followed is decided entirely at run time, and therefore, there is no need for the data base administrator to define all possible paths through the data base.

In Section B, we present a simple data base which will be used for the examples. Section C gives IDA's input language and Section D gives IDA's main characteristics. Finally, in Section E, we present IDA's main limitations and possible areas for future research.

B. EXAMPLE DATA BASE

For the purposes of this paper, we will take our examples from the so-called "presidential data base" which was used in the special issue of Computing Surveys [11] devoted to data base management systems. Although we have not actually done it, writing the schema to handle this

presidential data base would be very easy--the generation of the data base would of course, take much longer, which is why we have not done it!

In our examples, we will assume that this data base is relational, with the following relations:

PRESIDENTS: (PRESIDENT, HOME-STATE, PARTY)

ELECTIONS : (ELECTION-YEAR, PRESIDENT, WINNER-VOTES, OPPONENT, LOSER-VOTES)

ELECTION-STATE: (ELECTION-YEAR, STATE, CANDIDATE, VOTENUM)

where PRESIDENT is the name of a president, HOME-STATE is his home state, PARTY is the party to which he belongs, ELECTION-YEAR is the year in which a presidential election occurred, WINNER-VOTES is the total number of votes obtained by the president-to-be, OPPONENT is the name of his opponent , LOSER-VOTES is the number of votes this opponent obtained, STATE is the name of a state, CANDIDATE is the name of a presidential candidate in that state, and VOTENUM is the number of votes this candidate obtained in that state.

This simple subset of the presidential data base will be sufficient to demonstrate IDA's characteristics.

C. IDA'S INPUT LANGUAGE

As we have mentioned, IDA is used in our system with a natural language front-end, and therefore a great emphasis was put into developing a very simple input language for IDA. Moreover, this simple format makes it very easy to interface

any front-end to IDA, whether it is a natural language front-end, a graphic front-end, or any formal query language front-end.

As already mentioned, the main goal of IDA was to relieve the user from knowing the structure of the data base when issuing an IDA query. A few simple examples will illustrate the format of the input to IDA.

Let us first consider the request:

Give the names of all the presidents.

For this request, the query to IDA would be: (? PRESIDENT). In general, to request the value of some field, the user simply precedes the field name by the symbol '?'.

Let us now consider the query:

Which president was elected in 1968?

In this case, the query to IDA would be:

```
(? PRESIDENT)(ELECTION-YEAR EQ 1968).
```

And for the request:

List all presidents elected between 1900 and 1948

the query to IDA is:

```
(? PRESIDENT)((ELECTION-YEAR GE 1900) AND
              (ELECTION-YEAR LE 1948))
```

In general, we may specify some restrictions on field values, using any boolean expression. The comparison operators accepted by the system are: EQ, NE, GT, GE, LT, or LE.

In the two questions above, two examples of such boolean restrictions appear: (ELECTION-YEAR EQ 1968) and

((ELECTION-YEAR GE 1900)AND(ELECTION-YEAR LE 1948)).

Finally, the query:

Which president obtained the most votes?

would correspond to the IDA query:

(? PRESIDENT)(* MAX WINNER-VOTES).

In general, to specify that he is interested in some set of fields in the data base which corresponds to the maximum (or minimum) value of some field, the user simply precedes the corresponding field name by * MAX (or * MIN). In the same way, to find the answer to a "how-many?" question such as: How many presidents were elected since 1948? the user would query IDA with: (ELECTION-YEAR GE 1948)(* COUNT PRESIDENT). This "* feature" was introduced to handle all computations which require a iterative program to be executed by the DBMS. Such is the case with the computations of the maximum, minimum, and count, which are currently implemented. It could easily be extended to other cases such as computations of sums or averages.

Having seen these examples, we can now formally specify the input to IDA: it is a series of lists, each of which may take any one of three different formats:

. (? fieldname)

. A complex boolean expression

. (* <*OP> fieldname) where <*OP> is one of MAX, MIN, or COUNT.

From the above description, it is clear that IDA does not require the user to know the structure of the data base in issuing his query.

D. FUNCTION AND STRUCTURE OF IDA

In this section, we explain the features of IDA, using examples whenever appropriate. To help keep these examples simple and self-explanatory, we will assume that IDA generates calls to a relational data base whose query language is exactly the same as IDA's, except for the explicit presence of relation names. In the system actually developed, the results of IDA are handed to the Datacomputer (via FAM) and therefore, the queries generated by IDA are in Datalanguage, the name of the query language of the Datacomputer. For the interested reader, an actual transcript of a session with our system is included in the Appendix.

D.1 IDA's Structural Schema

As indicated above, each query to IDA may be considered to be issued against a single relation, the fields of which are all those which appear in the query. IDA must then solve the problem of building this relation dynamically from the actual relations in the data base, using the classical relational operators: projections, restrictions, and joins. To do so, IDA uses a structural model of the data base, called the "structural schema." It is composed of two types of information: "relation frames" and "field frames." These "frames" are similar to those discussed by Minsky [12], and

Winograd [13], for example. Each frame is a list of property-value pairs, also called "slots," which provide some specific information about the entity the frame models.

Each relation frame corresponds to an actual relation in the data base, and gives the possible links with all the other relations. In other words, the relation frames define all the permissible joins of two relations. In the case where a direct join is not possible between two specific relations, the two relation frames would instead include the name of a third relation which must be included in the join: in the presidential data base example, the direct join between the PRESIDENTS and ELECTION-STATE relations may not be allowed, in which case the join would have to be done among all three relations. As an example, the relation frame for PRESIDENTS would be: [ELECTIONS (PRESIDENT)

ELECTION-STATE §ELECTIONS†]

where each slot has the name of a relation and the link with that relation, or the name of a third relation in case of indirect link. In this example, the first slot in the PRESIDENTS relation frame indicates that to join any projections and/or restrictions of the PRESIDENTS and ELECTIONS relations the join must be taken over the PRESIDENT field. The second slot indicates that one is not allowed to take a direct join between the PRESIDENTS and ELECTION-STATE relations; one must take the join of the three relations--this is necessary, of course, to get the correct election year.

Each field frame corresponds to a field name which could appear in a user query. The main information contained in a field frame is the list of all the relations to which this field belongs. In many cases, a field belongs to a single relation: such is the case of VOTENUM which only belongs to the ELECTION-STATE relation in the presidential data base. In many other cases, a field may belong to several relations: such is the case of PRESIDENT which belongs to all the three relations of the presidential data base. Note that in the ELECTION-STATE relation, PRESIDENT appears under the field name CANDIDATE.

These two types of information, the links in the relation frames and the relation names in the field frames, are used by IDA's "covering algorithm" to determine at run time the logical view corresponding to any given user query. More precisely, the role of the covering algorithm is to find the smallest set of relations that cover all the fields in the query--a set that provides for any two relations, either a legal direct join or an indirect join using only other relations in the set. In other words, the relation frames are analogous to a graph in which each node is a relation, and the edge between two nodes is the link between the two relations. Then, the role of the covering algorithm is to find the smallest (by the number of nodes) connected subgraph which covers every field in the query. We are currently using a heuristic algorithm which works reasonably well, and eliminates the need to do an exhaustive search in order to

find the minimum cover. Although such a search would always be successful, it would be time-consuming, particularly when the number of relations in the data base and the number of fields in the query are large. At each iteration of the algorithm, we have a list of already chosen relations (empty for the first iteration), and a list of fields in the query not yet covered by the chosen relations. Then, we pick at random one not-yet-covered field, and try to find a relation that

- Covers it;
- Has a direct link with one already chosen relation, if such a relation exists; and
- Covers as many not-yet-covered fields as possible, if there is a choice.

The strategy of IDA's covering algorithm is to minimize some cost function. In our current implementation, this cost function is just the number of relations that need to be accessed--a strategy which is optimal in many, but not all, cases. The strategy is particularly relevant when the relations are on various computers, and may need to all be copied on a single computer (this would occur if the DBMS allows, and IDA generates, multi-file queries.) However, the strategy would be suboptimal if, for example, the relations were all on the same computer, and if the indexing characteristics of these relations were very different. In this case, a "query cost estimator," of the type developed by

Hammer [14] would be needed, and it would indicate the cost that IDA would try to minimize. This may be a worthwhile later addition to the system.

To see how the covering algorithm performs, let us consider the following thre examples:

D.1.1 Example A: Which president was elected in 1968?

The corresponding query to IDA is:

(? PRESIDENT)(ELECTION-YEAR EQ 1968)

The generated program is:

IN ELECTIONS RELATION: (? PRESIDENT)

(ELECTION-YEAR EQ 1968)

Clearly, in this case, not much work needs to be done: IDA finds in the structural schema that both PRESIDENT and ELECTION-YEAR are in the ELECTIONS relation, with those very same field names, and issues the corresponding query.

D.1.2 Example B: How many votes did McGovern obtain in Ohio?

The corresponding IDA query is:

(CANDIDATE EQ 'MCGOVERN')(STATE EQ 'OHIO') (? VOTENUM)

The generated program is:

IN ELECTION-STATE RELATION:

(CANDIDATE EQ 'MCGOVERN')(STATE EQ 'OHIO')(? VOTENUM)

In this case, we have to go only to the ELECTION-STATE relation, and not at all to the ELECTIONS relation. The reason is of course, that all the fields in the query are covered by the ELECTION-STATE relation--even though some are also covered by the ELECTIONS relation.

D.1.3 Example C: When was the last president from California elected?

The query to IDA is:

```
(? ELECTION-YEAR)
     (* MAX ELECTION-YEAR)(HOME-STATE EQ 'CALIFORNIA')
```

The query program generated is:

```
IN PRESIDENTS RELATION:
                (HOME-STATE EQ 'CALIFORNIA')(? PRESIDENT)

IN ELECTIONS RELATION:
     (* MAX ELECTION-YEAR)(? ELECTION-YEAR)
                              ((PRESIDENT EQ ...) OR ...)
```

Here, ((PRESIDENT EQ ...) OR...) would be filled by the response to the first query. This is a case in which an actual link between relations is required since IDA must perform the join between the PRESIDENTS and ELECTIONS relations. Clearly, several kinds of information are needed to build the correct DBMS queries. First, IDA must determine in which relations the fields are located: this information is found in the field frames, as already indicated. Second, after IDA has decided that two relations need to be accessed, namely, PRESIDENTS and ELECTIONS, it has to determine the "link" between them--in other words, what kind of information is needed from the first relation to limit the search inside the second relation. In this case, this link is the value of the field PRESIDENT. This linkage information is found in the relation frame for either relation. For example, in the relation frame corresponding to ELECTIONS, one would find a slot corresponding to the PRESIDENTS relation, where the linkage would be indicated: in this case, the field name PRESIDENT would be in this slot.

Besides the frame slots used by the covering algorithm, other slots are defined for the field frames. For example, the frame for PRESIDENT is:

```
[RELATIONS (PRESIDENTS ELECTIONS ELECTION-STATE)
 ALIAS-IN-ELECTION-STATE CANDIDATE]
```

where the first slot corresponds to the list of relations that cover the field, and where the second slot gives the actual name of the field in the ELECTION-STATE relation. The use of this second slot is illustrated in the following example:

D.1.4 Example D: How many votes did Kennedy obtain in Illinois?

The corresponding query to IDA is:

```
(PRESIDENT EQ 'KENNEDY')(STATE EQ 'ILLINOIS')(? VOTENUM)
```

The generated program is:

```
IN ELECTION-STATE RELATION:
     (CANDIDATE EQ 'KENNEDY')
          (STATE EQ 'ILLINOIS')(? VOTENUM)
```

Although PRESIDENT was mentioned in the query to IDA, IDA generated a call using CANDIDATE instead, which is correct since PRESIDENT does not appear in the STATE-ELECTION relation, and may be replaced by CANDIDATE. IDA made the replacement because CANDIDATE was the value in the ALIAS-IN-ELECTION-STATE slot of the PRESIDENT field frame. Note that a different field frame for CANDIDATE may or may not exist, depending on the intended use of the particular data base. It may therefore, be possible to make use of several field frames corresponding to various aliases of the same field. This would allow IDA to handle a query in different ways, depending on which alias of the field name is actually used in the query. In particular, this may be a way

to avoid the multi-path problem mentioned by Carlson [4]. Take, for example, a multi-path case that arises even in the simple presidential data base. Previously, we assumed that no direct join could be taken between the PRESIDENTS and ELECTION-STATE relations. However, the question "List the number of votes obtained by every president in his home state," requires that we take this direct join, using the link between the pair PRESIDENT HOME-STATE, and the pair CANDIDATE STATE. The query to IDA would be:

(? CANDIDATE)(? HOME-STATE)(? VOTENUM)

Since this query would not mention PRESIDENT, IDA should not assume that the question was restricted to the year in which the candidate was elected president, and therefore, should not access the ELECTIONS relation. Note that, in this case, we have two field frames which correspond to elected candidates, namely CANDIDATE and PRESIDENT, and two which correspond to their home states: STATE and HOME-STATE. Then, there are four ways of combining those together in an IDA query. Two of those--namely the CANDIDATE STATE and PRESIDENT HOME-STATE pairs--do not require any join at all. The other two pairs correspond to the two possible ways of handling the join between the PRESIDENTS and ELECTION-STATE relations. Therefore, depending on which pair is actually used in the IDA query, the multi-path ambiguity can be eliminated.

Although many other slots exist in both the field and the relation frames, and are used for specific purposes by IDA, we will mention here only one final one: the "procedural slot." We will explain its use in the following example.

D.1.5 Example E: How many votes did Nixon obtain in 1960?

The query to IDA could be:

```
(? VOTES) (NAME EQ 'NIXON')
          (ELECTION-YEAR EQ 1960)
```

The main problem that is raised by this query is that there is no VOTES field in the data base. IDA must determine if it should ask:

```
IN ELECTIONS RELATION:
     (ELECTION-YEAR EQ 1960)(? WINNER-VOTES)
```

or

```
IN ELECTIONS RELATION:
     (ELECTION-YEAR EQ 1960)(? LOSER-VOTES)
```

There are several ways to decide, of course: IDA could check whether Nixon was the president or not in 1960, and issue the appropriate query to the ELECTIONS relation. In fact, in this case, the best way may be as follows:

```
IN ELECTIONS RELATION:
     (ELECTION-YEAR EQ 1960)(? PRESIDENT)(? WINNER-VOTES)
                                          (? LOSER-VOTES)
```

Then, depending on the response, we would return the value of either LOSER-VOTES or WINNER-VOTES as the value of VOTES. This is an example of what we call the "conditional case." Somewhere a condition on the value of a field has to be tested, and, depending on the result of the tests different queries to the data base may be issued, or, as in this example, different responses may be given to the user, or both.

IDA handles this situation by having in the field frame that corresponds to the VOTES field a special program, a procedural attachment, in Winograd's terms [13]; this program is executed by IDA and does exactly what is needed to handle this case. First, it replaces in the query (? VOTES) by (? WINNER-VOTES)(? LOSER-VOTES)(? PRESIDENT). Then, when the answer is returned, it builds it back to the user, using the term VOTES as he would expect it. Procedural attachments of this type may be used not only in the conditional case, but also in cases of complex redundancies. For example, if the total number of votes obtained in an election was not stored explicitly in the data base, a special procedural attachment to a TOTAL-VOTES field frame could be used to call IDA recursively to find the number of votes obtained in each state, and add them up.

These five examples have given the reader a reasonably complete picture of the use of the frame slots by IDA. In the next section, we briefly explain how IDA orders the accessing of relations.

D.2 Relation Ordering by IDA.

IDA currently decides on the ordering of the relations to access by using two very simple rules:

. IDA tries to defer retrieving the values of the fields requested by the user as long as possible. The intuitive reason for this rule is that acquiring values for fields queried by the user does not provide any additional

constraints on any subsequent data base query. All other things being equal, it is best to try to constrain the data base queries as soon as possible by obtaining early those field values to be used in subsequent queries. Since other things are not always equal, this rule is only a "soft" constraint on IDA compared to the next one.

. IDA should not issue a query of the type (* <OP> fieldname) until all other boolean restrictions have been used. This is a strict constraint, since, if not used, IDA's results would be incorrect as Example G below will make clear.

We now present two examples of the use of these rules:

D.2.1 Example F: What is the home state of the last President?

The query to IDA is: (? HOME-STATE)(* MAX ELECTION-YEAR)
The generated program is:

IN ELECTIONS RELATION: (* MAX ELECTION-YEAR)(? PRESIDENT)

IN PRESIDENTS RELATION: (? HOME-STATE)(PRESIDENT EQ ...)

This example differes from Example C only in the order of access to the two relations. In Example C, the PRESIDENTS relation was accessed first since it was the most constrained relation, while in this case the reverse is true.

D.2.5 Example G. Which president from Ohio obtained the most presidenial votes?

The query to IDA is:

(? PRESIDENT)(HOME-STATE EQ 'OHIO')(* MAX PRES-VOTES)

The generated program is:

```
IN PRESIDENTS RELATION:
     (? PRESIDENT)(HOME-STATE EQ 'OHIO')

IN ELECTIONS RELATION:
     (* MAX WINNER-VOTES)(? PRESIDENT)
                    (PRESIDENT EQ ... OR PRESIDENT EQ ...)
```

Clearly, in this case, we must not ask to find the president who had the most votes until we have found which presidents were from Ohio.

Therefore, the second rule above is used systematically, while the first is used only when there is a choice. Although, in our actual trials, these simple rules have been surprisingly efficient, we recognize the need for additional research in this area. In particular, the "query cost estimator" of the type being developed by Hammer [14] would provide some additional information which could profitably be used to complement the first rule. Moreover, a better understanding of the "semantics" of both the data base and the query would probably be useful, as clearly demonstrated by Carlson [4].

Having described the purpose and function of IDA, we will now briefly present its architecture.

D.3 IDA's Architecture

Figure 2 presents the flow diagram of IDA. IDA is basically a loop which is executed once per relation accessed. In the next few paragraphs, we explain the main functions of each of the "black boxes" of Figure 2.

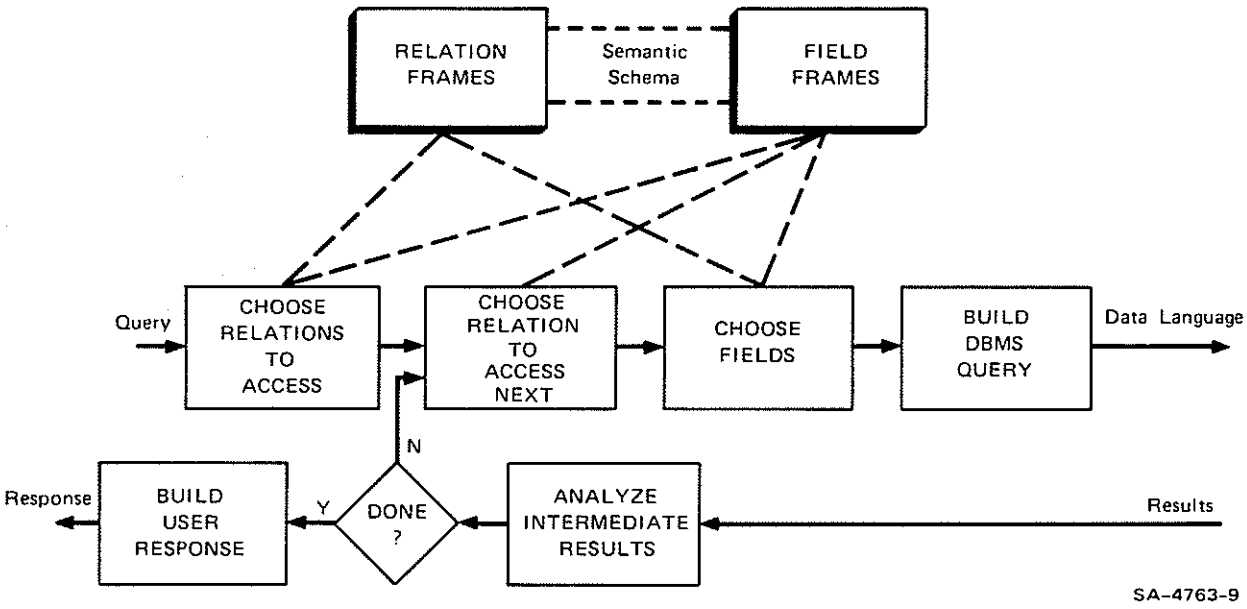
RELATION FRAMES
Semantic Schema
FIELD FRAMES
Query
CHOOSE RELATIONS TO ACCESS
CHOOSE RELATION TO ACCESS NEXT
CHOOSE FIELDS
BUILD DBMS QUERY
Data Language
N
Response
BUILD USER RESPONSE
Y
DONE ?
ANALYZE INTERMEDIATE RESULTS
Results
SA-4763-9

FIGURE 2

D.3.1 The first operation that occurs is parsing the query. This parser is extremely simple, since, as we have explained, the input language of IDA was chosen in order to guarantee the simplicity of the parser, and to allow for an easy-to-use interface to IDA from a natural language front-end, a menu selection, a graphics package or any other user communication front-end.

Immediately after parsing, IDA decides which relations will be accessed. To do so, IDA's covering algorithm not only uses the field frames of the fields appearing in the query, but also the relation frames. The relation frames must be used since relations may need to be accessed because of indirect linkages, even though no field in the query is covered by those relations. As already explained, in choosing the relations to be accessed, the covering algorithm tries to minimize a cost function, which currently is just the number of relations to be accessed.

D.3.2 Once IDA has chosen which relations it will access, it enters its basic loop. First, it decides which relation it will access next; if there are none left, the complete answer has been built and is given to the user. Otherwise, the decision as to which relation to access next is taken according to the rules mentioned in Section D.2. IDA examines what fields have their values specified in the user query, and tries to access first a relation where some of these

fields appear. In all cases, it tries to delay as long as possible taking the maximum, or minimum, of some field values (the "* <OP> cases").

Another important heuristic involves indirect links. Let us suppose that, having applied the above rules, IDA is still left with a choice between two relations, say A and B; and that C is a relation still to be accessed. Then, IDA checks whether both A and B have a direct link with C. If one of them does not, it will be accessed first. Intuitively, if B and C have a direct link, we want to restrict the records retrieved from B as much as possible before joining them with C, and therefore we should access A first. More generally, if we have a choice between several relations, we choose to access first the one which has the most indirect links with relations not yet accessed.

D.3.3 Once a relation is selected, IDA decides which restrictions to send. First, it will send any boolean restriction that applies to any field in the relation. Then, it decides which values it is going to retrieve from that relation; it will ask for values of links needed for accessing later relations, of fields needed to link with already known results (from previously accessed relations), and, if they do not appear in later relations, of the fields whose values are requested by the user. IDA will send the "* <OP> field-name" query only after all other restrictions imposed by the user query have been used.

D.3.4 Finally, the query to the DBMS is programmed and issued. Essentially, the query portions concerning fields whose values are needed, and the boolean expressions which apply, are prepared dynamically by IDA and are incorporated into one of several prestored query templates.

D.3.5 When the DBMS response comes back, IDA "joins" it with the previous results, and loops back to the second step. In essence, at the end of each loop, a relation has been built in IDA's local memory, which is the combination of all the information already obtained. In that sense, one may consider that IDA performs during each loop the three classical relational operations of restriction, projection, and joining. IDA performs this automatically without explicit help from the user.

Moreover, as explained in Example E, IDA will also execute the "procedural attachments" during any of the steps above, as required by the structural schema.

To summarize this analysis of IDA's flow, IDA operates on a "query at a time" basis. It does not build a complete data base access program in advance, and this is one reason why IDA is reasonably time efficient. Roughly, IDA takes 100 milliseconds per relation accessed, using INTERLISP as the programming language, on a DEC KL-10 computer, under the TOPS-20 operating system.

In addition to being efficient in terms of run time, IDA is, as we have seen, quite easy to use. This is mainly due to its simplicity. However, it is obvious that such efficiency has to lead to some limitations, and we explain those in Section E.

E. IDA'S LIMITATIONS AND FUTURE RESEARCH AREAS

IDA has several limitations which we will briefly mention and analyze. The first limitation is the multi-path problem mentioned by Carlson [4] and Roussopoulos & al. [3]. A simple example of that problem arises when two relations have more than one link between them. In the simplest cases, they might have two direct links; in more complex cases, they may have several links, some or all of them being indirect. The complete solution to this problem probably requires some understanding of the semantics of the data, as explained in Carlson [4], for example. However, we feel this may not be absolutely true in all cases: sometimes, it may be possible to "guess" from the query which link the user is interested in. This would occur if the structural schema indicated that some of the fields can only be associated with some of the links and not others. Then, in some cases, it could be possible to operate in essentially the same way as IDA does currently. In other cases, the query to IDA would still be ambiguous on which links to follow and the user's intervention would still be required. For reasons of simplicity and efficiency this may be an attractive route to

explore. So far, we have not pursued it, however, since we are more interested in examining how to disambiguate multi-path queries by using semantic knowledge of the data base, as suggested in Carlson [4].

Another area for future research is to extend IDA's input language to include queries not currently covered, to see whether the same basic techniques and heuristics would still apply. An example of an English question which cannot be translated into IDA query language is the following: Was there a Democratic president for whom all congresses were Republican? The problem, here, is that the current format for the IDA query does not admit any explicit scoping of quantifiers. It would be interesting to study such query situations, and to see whether the simple techniques used in IDA would still apply with limited modifications.

Another important research area is for IDA to access different data base management systems, which have different query languages, and even different data models. It is our contention that it would not be hard to rewrite IDA so that it would access, say, a CODASYL data base management system [15]. More challenging, and more interesting, would be to rewrite IDA so that it would access both a relational and a CODASYL data base. In other words, we would like to be able to model the data base management systems and their query languages, and use these models to build query programs in the appropriate access language. Some research has begun in this area and has been reported by Nahouraii et al. [16].

However, these authors assume the existence of the DIAM-II architecture in all the data base management systems to be used. We would like to free ourselves from such a requirement, and to assume that the user is interested in accessing data bases which are not prepared to cooperate with each other.

As we have mentioned, IDA operates on a heuristic step-by-step basis. In some cases, this may lead to some suboptimal query programs being generated, and may possibly even fail to produce an answer when one exists. It should be possible to use automatic program generation techniques to build a complete, optimal program of file accesses before issuing any query. Research in this area has been pursued at SRI by Furukawa [17] and has shown some promise.

Finally, we would like to point out that in all the examples, we have assumed that IDA was making the joins, instead of requesting the DBMS to do them. In fact, we have developed the routines to generate the DBMS queries asking for the joins to be performed. However, this creates some interesting difficulties which have to be studied further. For example, IDA should be made aware of the locations of the files, so that it will not ask for two relations to be joined that are located on different machines--which would be very expensive in time. Also, it already appears that, in some cases, the DBMS is less efficient than IDA at performing some joins; in other cases, some limit exists on the number of relations the DBMS may join as part of one single request.

Consequently, a model of the DBMS behavior will be needed to decide whether to ask for the join to be performed by the DBMS, or for IDA to do it itself.

F. CONCLUSION

We have presented the capabilities and characteristics of a data base access system that employs simple Artificial Intelligence techniques to free users from knowing the structure of the data base. IDA frees its users from having to know many of the peculiarities of the data base that they are using--conditional cases, redundancies, subdivisions into relations. In order to obtain this result, IDA decides automatically and dynamically which restrictions, projections and joins to perform, and in what order. This may make IDA very useful in case of large, complex data bases, where it would not be possible to build all the possible query programs in advance. In essence, IDA performs a tedious automatic programming job with reasonable simplicity and efficiency. This, in some cases, results in a suboptimal access strategy; however, from our experience with various users, it appears that it is well within acceptable limits. Additional research is needed to extend the scope of the system in the areas we mentioned; our goal is to extend it in some of the suggested areas, while still keeping its overall simplicity and efficiency.

REFERENCES

1. E. D. Sacerdoti, "Language Access to Distributed Data with Error Recovery," Proceedings of the Fifth International Joint Conference on Artificial Intelligence, Cambridge, Mass., August 1977.

2. E. F. Codd, "Seven Steps to Rendezvous with the Casual User," in Data Base Management, J. W. Klimbie and K. I. Koffeman ed., pp. 179-200 (North-Holland, 1974)

3. N. Roussopoulos and J. Mylopoulos, "Using Semantic Networks for Data Base Management," Proceedings of the First International Conference on Very Large Data Bases, Framingham, Mass., September 1975, pp. 144-172.

4. C. R. Carlson and R. S. Kaplan, "A Generalized Access Path Model and its Application to a Relational Data Base System," Proceedings of the International Conference on Management of Data, Washington, D.C., 1976, pp.143-154

5. G. G. Hendrix, E. D. Sacerdoti, D. Sagalowicz and J. Slocum, "Developing a Natural Language Interface to Complex Data," paper submitted to the Third International Conference on Very Large Data Bases.

6. P. Morris and D. Sagalowicz, "Managing Network Access to a Distributed Data base," Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley, California, May 1977.

7. T. Marill and D. Stern, "The Datacomputer--A Network Data Utility," AFIPS Conference Proceedings, Vol. 44, May 1975, pp.389-395.

8. J. Farrell, "The Datacomputer--a Network Data Utility," Proceedings of the Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley, California, May 1976, pp.352-364.

9. M. Stonebraker, "Implementation of Integrity Constraints and Views by Query Modification," Proceedings of the International Conference on Management of Data, San Jose, California, May 1975, pp.65-78.

10. D. D. Chamberlin, J. N. Gray and I. L. Traiger, "Views, Authorization, and Locking in a Relational Data Base System," Proceedings AFIPS National Computer Conference, AFIPS Press, Vol. 44, 1975.

11. ACM Computing Surveys, "Special issue: Data-Base Management Systems," March 1976

12. M. Minsky, "A Framework for Representing Knowledge," MIT Artificial Intelligence Laboratory, Memo No. 306, Cambridge, Mass., June 1974.

13. T. Winograd, "Five Lectures on Artificial Intelligence," Stanford Artificial Intelligence Laboratory, Memo No. AIM-246, Stanford, CA, Sept. 1974

14. M. Hammer and A. Chan, "Index Selection in a Self-Adaptive Data Base Management System," Proceedings of the International Conference on Management of Data, Washington, D.C., June 1976, pp.1-8

15. CODASYL, "Data Base Task Group Report," ACM, New York City, N.Y., Oct. 1969.

16. E. Nahouraii, L. O. Brooks and A. F. Cardenas, "An Approach to Data Communications between Generalized Data Base Management Systems," Proceedings of the Second International Conference on Very Large Data Bases, Brussels, Belgium, September 1976, pp. 117-142.

17. K. Furukawa, "A Deductive Question Answering System on Relational Data Bases," paper to be presented to the the Fifth International Joint Conference on Artificial Intelligence, Cambridge, Mass., August 1977.

APPENDIX

What follows is a trace, demonstrating the capabilities of the LADDER system developed at SRI. The system allows a casual user to access data about ships that is located in two remote computers, located at CCA (Computer Corporation of America) and at NOSC (Naval Ocean Systems Center), via ARPANET links that are established as needed by the program itself.

The trace included below shows the user's input, the calls to IDA (including recursive calls generated by IDA itself), the Datalanguage generated by FAM, and the response by the system. Some explanatory comments have been added.

```
@LADDER.EXE
PLEASE TYPE IN YOUR NAME: demonstration
DO YOU WANT INSTRUCTIONS? (TYPE Y OR N) NO
33_Where is the Kennedy?
PARSED!
PARSE TIME: .104 SECONDS
IDA:
QUERYLST = ((NAM EQ 'JOHN F.KENNEDY') (? PTP) (? PTD))
                ;*For this query, we ask for the present track position
                ;*(PTP) and date when it was reported (PTD).

                ;*First FAM will connect to the Datacomputer (DC).
                ;*What is preceded by >> corresponds to
                ;*synchronization messages from the DC.
 CONNECTING TO DATACOMPUTER AT CCA1:
>> ;0031 770519003427   IONETI: CONNECTED TO SRI-KL-16700010
>> ;J150 770519003429   FCRUN: V='DC-3/50.00.3' J=3 DT='WEDNESDAY, MAY 18,
**1977 20:34:29-EDT' S='CCA'
>> ;J200 770519003429   RHRUN: READY FOR REQUEST
*> SET PARAMETERS
                ;*We now ask the local program which interfaces with the DC
                ;*not to show those messages from the DC--except for errors.
*< V    VERBOSITY (-1 TO 5):   1
*< P    PROCEED WITH DATALANGUAGE  [CONFIRM WITH <CR>]
                ;*Now, FAM logs on the DC, then opens files and ports. A
                ;*PORT on the DC is both a logical view of the files and
                ;*an access path over the ARPA network. The next query
                ;*requests that data be sent from the SHIP file into the
                ;*NSTDPORT1 port, i.e. that they be sent over the ARPA net.
                ;*Whatever is preceded by CCA1: is a Datalanguage query
                ;*being sent to the DC located on the CCA1 computer.

CCA1:LOGIN %TOP.ACCAT.GUEST ;
CCA1:OPEN %TOP.ACCAT.SAGALOWICZ.NSTDPORT1 WRITE;
CCA1:OPEN %TOP.ACCAT.SHIP READ;
CCA1:FOR NSTDPORT1 , SHIP WITH (NAM EQ 'JOHN F.KENNEDY') BEGIN STRING1 =
CCA1:UIC STRING2 = VCN END;
```

```
*> TOTAL BYTES TRANSFERRED: 13
CCA1:OPEN %TOP.ACCAT.SAGALOWICZ.NSTDPORT2 WRITE;
CCA1:OPEN %TOP.ACCAT.TRACKHIST READ;
CCA1:FOR NSTDPORT2 , TRACKHIST WITH (UIC EQ 'N00002') AND (VCN EQ '0')
CCA1:BEGIN STRING1 = PTP STRING2 = PTD END;
*> TOTAL BYTES TRANSFERRED: 30
IDA = ((PTP '6000N03000W' PTD 7601171200))
                ;*This is the result: note that we needed to access two
                ;*files to obtain it. Now, the natural language front-end
                ;*presents the results to the user in a better format:
COMPUTATION TIME FOR QUERY: 2.701 SECONDS
REAL TIME FOR QUERY: 46.57 SECONDS

     (POSITION 6000N03000W DATE 7601171200)

34_What is the assigned home port of the Biddle?
PARSED!
PARSE TIME: .231 SECONDS
IDA:
QUERYLST = ((? PDEP) (NAM EQ 'BIDDLE'))
CCA1:OPEN %TOP.ACCAT.UNIT READ;
                ;*Note here the use of the alias: NAM is replaced by ANAME
CCA1:FOR NSTDPORT1 , UNIT WITH (ANAME EQ 'BIDDLE') BEGIN STRING1 = HOGEO
CCA1:END;
*> TOTAL BYTES TRANSFERRED: 10
CCA1:OPEN %TOP.ACCAT.PORT READ;
CCA1:FOR NSTDPORT1 , PORT WITH (HOGEO EQ 'CHAR') BEGIN STRING1 = DEP END ;
*> TOTAL BYTES TRANSFERRED: 46
IDA = ((PDEP 'CHARLESTON'))
COMPUTATION TIME FOR QUERY: .813 SECONDS
REAL TIME FOR QUERY: 16.732 SECONDS

     PORT = CHARLESTON

                ;*Note in the next query that INLAND remembers the context
                ;*and will understand correctly the incomplete question.
35_What is the commanding officer's name?
PARSED!
PARSE TIME: .114 SECONDS
IDA:
QUERYLST = ((? RANK) (? CONAM) (? NAM) (ANAME EQ 'BIDDLE'))
CCA1:FOR NSTDPORT1 , UNIT WITH (ANAME EQ 'BIDDLE') BEGIN STRING1 = RANK
CCA1:STRING2 = CONAM STRING3 = ANAME END;
*> TOTAL BYTES TRANSFERRED: 62
IDA = ((RANK 'CAPT' CONAM 'J.TOWNES' NAM 'BIDDLE'))
COMPUTATION TIME FOR QUERY: .262 SECONDS
REAL TIME FOR QUERY: 4.889 SECONDS

     (RANK CAPT NAME J. TOWNES SHIP BIDDLE)

                ;*The next query shows one example of the * feature.
36_Where is the fastest american nuclear submarine?
PARSED!         PARSED!
PARSE TIME: .344 SECONDS
IDA:
QUERYLST = ((? NAM) (* MAX MCSF) (NAT EQ 'US') (FTP2 EQ 'N')
            (TYPE1 EQ 'S') (TYPE2 EQ 'S') (? PTP) (? PTD))
                ;*We have reached the maximum number of files/ports we can
                ;*keep opened, FAM starts closing them as needed.
```

```
CCA1:CLOSE NSTDPORT1;
CCA1:OPEN %TOP.ACCAT.SAGALOWICZ NSTDPORT WRITE;
CCA1:BEGIN DECLARE Z STRING (,100) ,D='}' DECLARE X STRING (,100) ,D='}'
CCA1:DECLARE Y STRING (,100) ,D='}' Y = '*' DECLARE Y1 STRING (,100) ,D='}'
CCA1:Y1 = '*' DECLARE Y2 STRING (,100) ,D='}' Y2 = '*' DECLARE Y3 STRING
CCA1:(,100) , D='}' Y3 = '*' X = '00.0' FOR SHIP WITH (NAT EQ 'US') AND
CCA1:(FTP2 EQ 'N') AND (TYPE1 EQ 'S') AND (TYPE2 EQ 'S') BEGIN Z = MCSF IF
CCA1:Z LT '99.9' AND X LT Z THEN BEGIN Y = NAM X = Z Y1 = UIC Y2 = VCN END
CCA1:END NSTDPORT.STRING1 = Y NSTDPORT.STRING2 = X NSTDPORT.STRING3 = Y1
CCA1:NSTDPORT.STRING4 = Y2 END;
*> TOTAL BYTES TRANSFERRED: 52
CCA1:FOR NSTDPORT2 , TRACKHIST WITH (UIC EQ 'N00007') AND (VCN EQ '0')
CCA1:BEGIN STRING1 = PTP STRING2 = PTD END;
*> TOTAL BYTES TRANSFERRED: 30
   IDA = ((NAM 'LOS ANGELES' PTP '0000N04500E' PTD 7601171200 MCSF '30.0'))
COMPUTATION TIME FOR QUERY: 1.969 SECONDS
REAL TIME FOR QUERY: 160.169 SECONDS
(SHIP LOS ANGELES POSITION 0000N04500E DATE 7601171200 MXSPD 30.0)
                ;*The next query is an example of IDA's ability to navigate
                ;*in the data base. It will take the joins of 4 files.
37_Where are the Sturgeon class submarines?
PARSED!
PARSE TIME: .266 SECONDS
IDA:
QUERYLST = ((? NAM) (SHIPCLAS EQ 'STURGEON') (TYPE1 EQ 'S')
                (TYPE2 EQ 'S') (? PTP) (? PTD))
CCA1:CLOSE NSTDPORT;
CCA1:OPEN %TOP.ACCAT.SAGALOWICZ.NSTDPORT1 WRITE;
CCA1:CLOSE PORT;
CCA1:OPEN %TOP.ACCAT.SHIPCLASCHAR READ;
CCA1:FOR NSTDPORT1 , SHIPCLASCHAR WITH (SHIPCLAS EQ 'STURGEON') AND
CCA1:(TYPE1 EQ 'S') AND (TYPE2 EQ 'S') BEGIN STRING1 = SHIPCLAS END;
*> TOTAL BYTES TRANSFERRED: 30
CCA1:CLOSE UNIT;
CCA1:OPEN %TOP.ACCAT.SHIPCLASDIR READ;
CCA1:FOR NSTDPORT1 , SHIPCLASDIR WITH (SHIPCLAS EQ 'STURGEON') BEGIN
CCA1:STRING1 = UIC STRING2 = VCN END;
*> TOTAL BYTES TRANSFERRED: 91
CCA1:FOR NSTDPORT2 , TRACKHIST WITH( UIC EQ 'N00016' OR UIC EQ 'N00015'
CCA1:OR UIC EQ 'N00014' OR UIC EQ 'N00013' OR UIC EQ 'N00012' OR UIC
CCA1:EQ 'N00011' OR UIC EQ 'N00010') AND (VCN EQ '0') BEGIN STRING1 =
CCA1:PTP STRING2 = PTD STRING3 = UIC END;
*> TOTAL BYTES TRANSFERRED: 252
CCA1:FOR NSTDPORT1 , SHIP WITH( UIC EQ 'N00013' OR UIC EQ 'N00012' OR
CCA1:UIC EQ 'N00011' OR UIC EQ 'N00010' OR UIC EQ 'N00016' OR UIC EQ
CCA1:'N00015' OR UIC EQ 'N00014') AND (VCN EQ '0') BEGIN STRING1 = NAM
CCA1:STRING2 = UIC END;
*> TOTAL BYTES TRANSFERRED: 266
IDA = ((NAM 'STURGEON' PTP '3700N07600W' PTD 7601171200) (NAM 'WHALE' PTP
'3700N07600W' PTD 7601171200) (NAM 'TAUTOG' PTP '3700N07600W' PTD
7601171200) (NAM 'GRAYLING' PTP '3700N07600W' PTD 7601171200) (NAM
'POGY' PTP '3500N01000E' PTD 7601171200) (NAM 'ASPRO' PTP '3000N03000W'
PTD 7601171200) (NAM 'SUNFISH' PTP '3000N06000W' PTD 7601171200))
COMPUTATION TIME FOR QUERY: 2.578 SECONDS
REAL TIME FOR QUERY: 126.39 SECONDS
```

```
     SHIP     POSITION    DATE
     STURGEON 3700N07600W 7601171200
     WHALE    3700N07600W 7601171200
     TAUTOG   3700N07600W 7601171200
     GRAYLING 3700N07600W 7601171200
     POGY     3500N01000E 7601171200
     ASPRO    3000N03000W 7601171200
     SUNFISH  3000N06000W 7601171200

38_!SET(MAXNUMOFFILES 4)
                ;*We redo the same question: IDA now builds programs
                ;*to join up to 4 files--which is what request 38 means.
39_Where are the Sturgeon class submarines?
PARSED!
PARSE TIME: .246 SECONDS
IDA:
QUERYLST = ((? NAM) (SHIPCLAS EQ 'STURGEON') (TYPE1 EQ 'S')
                (TYPE2 EQ 'S') (? PTP) (? PTD))
CCA1:FOR R1 IN SHIPCLASCHAR WITH (SHIPCLAS EQ 'STURGEON') AND (TYPE1 EQ
CCA1:'S') AND (TYPE2 EQ 'S') FOR R2 IN SHIPCLASDIR WITH R2.SHIPCLAS EQ
CCA1:R1.SHIPCLAS FOR R3 IN SHIP WITH R3.UIC EQ R2.UIC AND R3.VCN EQ
CCA1:R2.VCN FOR NSTDPORT1, R4 IN TRACKHIST WITH R4.UIC EQ R3.UIC AND
CCA1:R4.VCN EQ R3.VCN BEGIN STRING1 = R3.NAM STRING2 = R4.PTP STRING3 =
CCA1:R4.PTD END;
*>TOTAL BYTES TRANSFERRED: 371
IDA = ((NAM 'STURGEON' PTP '3700N07600W' PTD 7601171200) (NAM 'WHALE'
PTP '3700N07600W' PTD 7601171200) (NAM 'TAUTOG' PTP '3700N07600W' PTD
7601171200) (NAM 'GRAYLING' PTP '3700N07600W' PTD 7601171200) (NAM 'POGY'
PTP '3500N01000E' PTD 7601171200) (NAM 'ASPRO' PTP '3000N03000W' PTD
601171200) (NAM 'SUNFISH' PTP '3000N06000W' PTD 7601171200))
COMPUTATION TIME FOR QUERY: 2.903 SECONDS
REAL TIME FOR QUERY: 224.155 SECONDS
     SHIP     POSITION    DATE
     STURGEON 3700N07600W 7601171200
     WHALE    3700N07600W 7601171200
     TAUTOG   3700N07600W 7601171200
     GRAYLING 3700N07600W 7601171200
     POGY     3500N01000E 7601171200
     ASPRO    3000N03000W 7601171200
     SUNFISH  3000N06000W 7601171200
40_done
PARSED!
Thank you
```