THE INTELLIGENT DATA OBJECT AND ITS DATA BASE INTERFACE

BY

NANCY LONG BUSACK

B. A., M. S., Ohio University, 1976, 1979

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1985

Approved by:

Major Professor Dr. E. A. Unger

CONTENTS

1. CHAPTER ONE: INTRODUCTION..... 1

1.1 ABSTRACT DATA DEFINED..... 1

1.2 ABSTRACT DATA TYPE OPERATIONS..... 3

1.3 FORMS AS INTELLIGENT DATA OBJECTS..... 5

1.4 ADVANTAGES IN USING FORMS..... 6

1.5 PREVIEW OF FOLLOWING CHAPTERS..... 8

2. CHAPTER TWO - THE IDO IN AN AUTOMATED OFFICE SYSTEM..... 10

2.1 FORMS IN THE OFFICE..... 10

2.2 ADVANTAGES..... 11

2.3 ASPECTS OF USING FORMS..... 12

2.3.1 ACCESS RIGHTS..... 12

2.3.2 FIELD SPECIFICATION..... 13

2.3.3 ABSTRACTION..... 15

2.4 OPERATIONS..... 15

2.5 FORM DEFINITION LANGUAGE..... 16

2.5.1 ADVANTAGES..... 17

2.5.2 PROTOTYPE..... 17

2.6 OFS..... 18

2.7 OFFICETALK-D..... 22

2.8 SYNOPSIS..... 23

3. CHAPTER THREE - DATA BASE AND MESSAGE SYSTEMS..... 24

THIS BOOK CONTAINS NUMEROUS PAGES WITH THE ORIGINAL PRINTING BEING SKEWED DIFFERENTLY FROM THE TOP OF THE PAGE TO THE BOTTOM.

THIS IS AS RECEIVED FROM THE CUSTOMER.

4. CHAPTER FOUR: INGRES - A RELATIONAL DATA BASE MANAGEMENT SYSTEM..... 29

4.1 STRUCTURING DATA IN A DATA BASE..... 29

4.2 INGRES - HISTORICAL..... 32

4.3 INGRES - DESIGN AND IMPLEMENTATION..... 33

4.3.1 DIRECT ACCESS..... 34

4.3.2 UTILITY COMMANDS..... 38

4.3.3 STORAGE STRUCTURE..... 41

5. CHAPTER FIVE: PROBLEM TO BE SOLVED..... 44

6. CHAPTER SIX: DESIGN OF IMPLEMENTATION PROJECT..... 48

6.1 DATA BASE..... 48

6.1.1 DESIGN OF SAMPLE DATA BASE..... 48

6.1.2 INGRES..... 49

6.2 USER INTERFACE WITH DATA BASE FILES..... 51

7. CHAPTER SEVEN: CONCLUDING REMARKS..... 53

7.1 PROJECT..... 53

7.2 EXTENSIONS..... 54

8. REFERENCES..... 55

9. APPENDIX ONE: BERN2..... 58

10. APPENDIX TWO: DATA DICTIONARY..... 59

11. APPENDIX THREE: APPEND CODE..... 60

12. APPENDIX FOUR: DELETE CODE..... 61

13. APPENDIX FIVE: REPLACE CODE..... 62

14. APPENDIX SIX: RETRIEVE CODE..... 63

15. APPENDIX SEVEN: JOIN CODE..... 64

LIST OF FIGURES

Figure 1. INTERFACE..... 4

Figure 2. STACK DEFINED AS A CLUSTER..... 4

Figure 3. OPERATIONS..... 16

Figure 4. FORM DEFINITION LANGUAGE ISSUES..... 18

Figure 5. PROCESS STRUCTURE..... 34

Figure 6. SAMPLE RELATION..... 36

Figure 7. INGRES - EXAMPLE 1..... 36

Figure 8. INGRES EXAMPLE 2..... 37

Figure 9. INGRES EXAMPLE 3..... 37

Figure 10. INGRES EXAMPLE 4..... 38

Figure 11. UTILITY COMMANDS..... 39

Figure 12. DATA BASE INTERFACE..... 44

Figure 13. SAMPLE DATA BASE..... 49

1. *CHAPTER ONE: INTRODUCTION*

An intelligent data object is loosely defined as an instance of an intelligent abstract data type. An abstract data type has been defined by several researchers as a data structure and the set of operations defined on the object. An intelligent abstract data type extends the set of operations to those defined from a class which may cause a "triggered" action, retrievals from an external source, and routing information within a potentially distributed system. Thus, the intelligent data object has its intelligence internal to itself.

1.1 *ABSTRACT DATA DEFINED*

This paper is concerned generally with the concept of the intelligent data object. This chapter will discuss what an abstract data type is and why the intelligent data object is a form of the abstract data type. An intelligent form will be defined and shown to be a superset of the intelligent data object. The purpose of the project is to investigate further the idea of the intelligent form in the context of structured electronic mail systems. The specific portion of the project addressed here is the database interface for the intelligent form in an electronic mail system. Chapters two through six will describe in more detail the IDO and the database issues involved.

There have been several authors who have done research on the concept of an abstract data type. Their literature and other authors' works will be the basis for the concept of an intelligent data object (IDO).

Originally, in defining abstract data types, the primary motivation was similar to structured programming philosophy whereby the programmer would be relieved of worrying about the details not relevant to the problem being solved. For example, when a programmer uses a procedure call such as those found in PASCAL, the concern is only with what the procedure does and not so much as how it is done. So, one need not worry about the algorithms that would be executed by the procedure. The language PASCAL, along with several other high level languages, "attempts to present the user with abstractions (operations, data structures, and control structures) useful to the application area." [LIS/ZIL]

This concept of the abstract data type follows directly once the idea of abstraction is understood. An abstract data type "defines a class of abstract objects which is completely characterized by the operations available on those objects." [LIS/ZIL] Once a definition of the appropriate operations for that type has been given the abstract data type can be defined. Going back to what was stated previously, when a programmer is working with an abstract data object the concern is only with what that

object does and how it behaves versus how that behavior is actually realized by an implementation.

The idea of an abstract data type is similar to those types that are built in by a programming language such as an integer or integer-array. When one uses these types they are concerned with only creating items of that type and the operations that can be performed on them. The interest is usually not in how the data objects are represented, such as in integer objects represented as a bit string, but rather the arithmetic operations that usually are associated with those integers.

1.2 *ABSTRACT DATA TYPE OPERATIONS*

When one defines an abstract data type it is necessary to include the set of operations that the data type requires for implementation. Zilles and Liskov [LIS/ZIL] call this the "operation cluster" or "cluster" for short. As an example they showed how a stack would be defined as a cluster. (See Figures 1 and 2.) The first part of the cluster provides a brief description of the interface; this defines the name of the cluster, the parameters required to create an instance of the cluster, and a list of the respective operations.

```
Example:  stack: cluster (element_type: type)
            is push,pop,erasetop,empty;
```

where 'is' is a reserved word used to reinforce the idea of the data type and its respective operations.

Figure 1. INTERFACE

The rest of the cluster definition has three parts to it: the object representation, the code to create objects and the operation definitions.

```
rep (type_param: type) = (tp: integer;
                          e_type: type;
                          stk: array[1..]
                               of type_param;

create
     s:rep(element_type);
     s.tp:= 0;
     s.e_type;= element_type;
     return s;
     end

push: operation(s:rep, v:s.1_type);
     s.tp:= s.tp + 1;
     s.stk[s.tp]:= v;
     return;
     end

etc.
```

Figure 2. STACK DEFINED AS A CLUSTER

Gehani and Gries [GRI/GEH77] hold a similar view of what an abstract data type is although it is defined a little differently. As they view a data type as a set of values

plus basic operations on them, they are approaching the idea of a "procedure operating on a parameter of any data type for which certain basic operations have been defined." This gives support to the idea of letting types of parameters be parameters.

1.3 *FORMS AS INTELLIGENT DATA OBJECTS*

The intelligent data object can be viewed as an abstract data type since it is a set of data objects and the operations performed on them. Intelligent data objects (IDO's) are forms that will be routed around a distributed office environment. The form will have certain intelligence internal to it to tell itself, for example, where it will be routed to, when updates should be made to the database and who has certain access rights to it.

McBride and Unger were even more explicit about what an intelligent form should contain. They stated that the intelligent form should contain information fields, a routing list, a list of instructions for the individuals to whom the form is routed, and a check-off list to keep track of the people that have processed the form. [McB/UNG83] The original concept of what constituted an intelligent form came from Gehani. [GEH82]

A form defined by Webster's New Collegiate Dictionary is a printed or typed document with blank spaces for insertion of

required or requested information. An electronic form is a computer representation of a paper form. Hence forth, this paper will concern itself only with electronic forms since they are the intelligent data object that will be discussed.

1.4 *ADVANTAGES IN USING FORMS*

There are several advantages to electronic forms in an office automation environment.

1. They help move from a manual office system to an automated one.
2. User interface can be monitored to ensure accuracy of the information.
3. Relatively simple to use and easy to understand.
4. Paper copies can be easily generated if needed.
5. They provide random access to fields for input and display purposes.
6. Some of the fields can be filled automatically by the system.
7. The form can prompt the user for input data.

In an office environment forms can be used as the basis for

office automation systems. They easily lend themselves to be treated as an entity where the forms can be viewed as one entity or subdivided into 'sibling' forms and later rejoined into the original parent form. Forms in the automated office can also be managed in the same or similar ways that paper forms can be handled. Thus, they wouldn't be totally foreign to someone using them for the first time.

Forms can also be routed and delivered automatically to those persons on a routing list. While this is occurring a means of tracing the form can also be accomplished. Internal to the routing specification can also be certain access privileges for those using the form so that certain fields can be "locked" or "unlocked" depending on the individual using the form. Finally, automatic error checking is another important feature of an electronic form use in an automated office.

The fields of a form are blank spaces that require filling. They can be more complex than those found on a paper form and they can be of several kinds. This complexity arises from the fact that the forms are designed for ease of use and that correctness checks can be performed automatically. The designer is given the responsibility to specify the attributes and properties of the form's fields. This will

concurrently ensure the user limited prespecified operations when using the form.

1.5 *PREVIEW OF FOLLOWING CHAPTERS*

In the following chapters more attention will be given to the electronic form. Chapter two will deal with the electronic form in an office information system. Many authors have supplied their views on this including Gehani[81], Gehani[83], Tsichritzis[80], Bernal[82], Shu[82] and Ellis[80]. Chapter three will look closer at the problem at hand of the data base and message systems. Supporting literature by Tsichritzis[81] and Schek, Pistor[SCH/PIS] is reviewed. Chapter four will be a discussion on Ingres, the relational database management system that is used for the implementation portion of this project. The relational database management system is discussed by Allman and Stonebraker[ALL/STO82], [STO/WON/KRE76] with examples provided by Epstein[EPS77] and Relational Technologies, Inc.[RTI84] This discussion will lead directly to chapter five, the review of the two different approaches to the implementation project. Chapter six discribes the design of the implementation project chosen using the IDO concept. Chapter seven will be concluding remarks on what was learned from the project

implementation and extensions that might be considered at some time in the future for additional study.

2. *CHAPTER TWO - THE IDO IN AN AUTOMATED OFFICE SYSTEM*

2.1 *FORMS IN THE OFFICE*

Forms, as defined in chapter one are printed or typed documents with blank spaces for insertion of required or requested information. The electronic form is a computer representation of a paper form. Most offices use a variety of paper forms. In fact, it would be a rare occurrence if no paper forms were present in a contemporary office.

With the emergence of the computer and the variety of uses that computer systems lend themselves to, forms are now starting to be used in the design of office automation systems. They are an important aspect of information retrieval systems. Forms have been used in the design of at least three office automation systems. Two of these, OfficeTalk-D (a subset of OfficeTalk) and OFS will be described later in this chapter. The third office system that uses electronic forms is Odyssey.

Some authors limit their view of the office system to that part of a business that handles the information dealing with operations such as accounting, payroll and billing. [ELL/NUT80] Examples of office work in this situation might be text editing, forms editing, filing documents, performing

simple computations, verification of information and communication between and among offices.

However, a much broader definition of an office system will be considered in this paper to include functions that are a step beyond those which a typical office worker might perform. For example, included in an office system might be a process where an engineer is required to complete a form on a cost reduction case, submit it to accounting and then after it has been approved there, route it to all levels of management for additional approvals. This would be an especially important function if the issue of extra funding for a project was present. These types of activities should be handled effectively in an automated office system.

2.2 *ADVANTAGES*

Several advantages of a form-based office automation system over a non-form-based system are as follows: [GEH82]

- Forms allow logically related data to be treated as an entity.
- Electronic forms retain many of the properties of paper forms (they can be copied, signed, mailed, stacked and check for correctness.)

- Forms can be traced.
- Specific users' access rights can be identified.
- Routing information can be internalized to the form.
- Forms can serve as a high-level protocol for information communication. Using extracted data from a form, a transmission can be made to another office with instructions that the data is of a particular form type. This results in a savings in transmission costs.

2.3 *ASPECTS OF USING FORMS*

Important aspects of using forms surface as one notes the advantages of a form-based office automation system versus a non-form-based system. Among these are user access rights, field specification and abstraction. [GEH82]

2.3.1 *ACCESS RIGHTS* Access rights of a form specify which users can change which fields. Personnel with proper authorization should be permitted to perform certain operations on forms. For example, in one company, managers might be the only ones authorized to approve purchase orders of $100,000 or more. However, at two levels lower, the department chief may be allowed to approve an order for up to $50,000 with no additional signatures.

Granting access rights is a matter of office policy. It may be done on an individual's rank or perhaps on the basis of the workstation involved. Granting by individual user's rank appears to be the more flexible of these two alternatives. With workstations having access rights it might be necessary to physically guard a station whereas with the individual, rights will be associated with the forms themselves.

Since offices are dynamic entities it would be desirable to be able to change access rights dynamically. To be able to do this, selected users should be given the access rights to change access rights of other users.

2.3.2 *FIELD SPECIFICATION* Field specification is another important aspect of using forms. Fields of a form are the blank spaces that require information to be inserted. In a paper form one must manually, at some point in time, verify each field to check for the accuracy of the information placed there. Electronic forms can audit this automatically and immediately by placing certain restriction on the data entered in each field. For example, a field may be designated to accept an integer between the values of 1 and 20. If a value of 25 or 7.5 were entered it would be rejected as not satisfying the constraints placed on that

field.

In additions to the type attribute such as integer, real or character a field may have additional attributes. These attributes, which may be assigned individually or used in sensible combinations, are as follows:

1. Required - data must be entered by the user for the form to be considered complete.

2. Virtual - this field is filled in automatically according to some computation specified using some expression supplied by the user.

3. Unchangeable - specifies that the field cannot be changed after it has been filled.

4. Tag and invariant - the value of this field determines which list of fields called a variant can be filled in by the user.

5. Ordered - can be filled in only after some other field has been completed.

6. Lock - specifies that certain fields cannot be changed after this field has been filled.

2.3.3 *ABSTRACTION* Abstraction is the third important issue mentioned with electronic forms. Chapter one described how intelligent data objects are very similar to abstract data types where they can be viewed as a set of objects and the operations performed on them. Electronic forms, being our case of the IDO, can be used as a tool for abstraction. The data contained on the form can be encapsulated so that the information contained therein can be relevant to performing one or more tasks. A user can then reason abstractly about the information in a particular type of form.

When one uses a form they should be allowed to interact with it only according to the prespecified operations suplied by the form designer. The user should not be permitted to work with the form by knowing the underlying implementation details. These should be hidden from the user.

2.4 *OPERATIONS*

Operations that one can use to manipulate forms instances in an automated office system come with every form definition. [BER82] Users, based on their access rights, will be allowed to perform the operations. See Figure 3.

- Create - used to create a new instance of a form type.
- Retrieval and filing - fundamental operations in office activities.
- Display - specified form is displayed.
- Filling - some fields must be filled automatically.
- Update - may be applied to the fields of the record for that form or to the constraints defining actions for that form.
- Mailing - mails the specified form to another user.
- Making copies - copies of the same form may be sent to multiple users.
- Manipulation - easy manipulation techniques are essential to the electronic form; included in this category is good text editing facilities.
- Destroy - the specified form is "destroyed" or made inaccessible to the user.

Figure 3. OPERATIONS

2.5 *FORM DEFINITION LANGUAGE*

A form definition language has been proposed as one way to quickly develop a form in an automated office environment. [GEH83] Of the few systems that have been developed using the concept of forms only SBA comes close to providing a high level form definition facility. SBA provides the user a means of defining forms using Query-By-Example. [ZLO77]

2.5.1 *ADVANTAGES* There would be several advantages to having a high-level form definition language available. Among these are the following:

1. Ease of defining new forms.

2. All pertinent information to design form stored in one location.

3. Form modification accomplished by changing high level definition instead of modifying code.

4. The form definition mechanism can be incorporated into a programming language to provide for convenient form processing. Forms would become an object or type such as integers or queues.

2.5.2 *PROTOTYPE* Gehani has developed a prototype electronic form system to study the feasibility of a form definition language and to evaluate what should be contained in this language. [GEH81] He recommends the issues displayed in Figure 4 be addressed by the form definition language. See Figure 4.

- Specification of the textual information that is to be displayed.
- Specification of the types and other attributes of the fields in a form.
- Definition of all operations that can be performed on the form type being defined.
- Association of access rights with the fields.
- Association of access rights with the operations.
- Association of pre-conditions and post-conditions with the fields.
- Association of pre-actions and post-actions with fields.
- Specification of external routines and names that are used in the form definition.
- Definition of local data and routines.
- Forms processing.

Figure 4. FORM DEFINITION LANGUAGE ISSUES

2.6 *OFS*

OFS is presented here as an example of an automated office system. [TSI80] D. Tsichritzis, a member of the Computer Systems Research Group at the University of Toronto, Toronto, Canada explains the ideas behind some of the design decisions for OFS. Their first objective was for the development of successful *integrated* office information systems. As a second goal he stressed the realization that a revolution in using available facilities was not

necessary; rather, an *evolutionary* process would be more acceptable to the work force than a totally new and different process. Finally, the potential of *automation* versus mechanization should be realized to free users from routine tasks.

OFS (Office Form System) is based on the concept of form processing in an office system. Forms were chosen for this system due to their capability to be used by an individual through convenient familiar methods. Forms can be viewed as text and formatted data where the form contains form fields and dispersed printed text.

Although the Computer Systems Research Group recognizes the value of repeating fields on a form, they chose to limit their form processing system and not deal with unbounded repeating groups.

Form fields are classified as being any one of three types. The first type requires the value in the field be entered at form instance creation time with no changes permitted later. The second type allows delayed entry of the field value but doesn't permit later changes. The third type allows the fields to be updated.

Work stations, not individual users, have defined access

rights in OFS. In this way each station has a unique signature and every time a field value is entered or changed in a form the system retains the station signature which initiated the change.

To identify each form instance a unique key is assigned at creation time. This key includes a copy meter to help in identifying multiple copies of the same form. The key is assigned from a common centralized service. Forms can be identified and traced according to this key.

Form processing is accomplished by mailing forms to a station where the operations on the form fields take place. Thus, a form must be moved into a station, or already reside there, before it can be retrieved, edited, etc. by that station.

To incorporate data management and form processing capabilities into an integrated system, OFS uses common files. That is, they store all form instances of the same form type in a form file. Each form instance corresponds to a relational tuple in that file. Included in each tuple the respective signatures and form key are also stored. The form blank is stored separately.

The form processing system and the data base system share

indices. So, if a change is made in a form field value the index used for the data base interface is updated. In this way there is a unique copy of data and a common access path. And the method allows the data to be accessed as either a form or a relation.

To incorporate automation in the office procedure, the system would require the ability of initiating activities automatically at a suitable time depending on the state of the system and predefined information. To facilitate this, the office procedure in the OFS environment consists of three parts: a *condition*, an *action* and a *notification*. The *condition* part specifies the terms under which the procedure is initiated. The *action* part specifies the operation to be performed. The *notification* part specifies the other objects which are notified when the procedure is completed.

Three activity types of office procedure are defined. The first type is called a desk activity due to the fact that a specific action is initiated at a station when certain conditions local to the station are present. A second type is called a mail activity which automatically routes mail. The third type is a coordination activity initiated by a family of forms arriving or being present in a specific file or mail tray. Mail and coordination activities provide a

way of relating different desk activities at more than one station.

2.7 *OFFICETALK-D*

A second example of a system which is experimenting with the concepts of distributed office information systems is OfficeTalk-D. This was developed by C. Ellis, A. Daniels and M. Bernal. [BER82] Another system, OfficeTalk-Zero, was developed earlier by some of these same people; however, it did not support traditional data base operations.

The user interface of OfficeTalk-D was developed with the idea of providing a transparent data base utilization through the use of forms. The user is aided by facilities to move electronic forms around via a cursor or mouse. This action can be accomplished with no prior knowledge of data base interation. Other activities users can engage are expanding the form, shrinking the form and scrolling it.

Initially, OfficeTalk-D allowed the user to work on specific units of work specified by scripts which were interactive or automatic. These scripts automated interface operations like menus and commands selection, text typing, window manupulation, etc. The script capabilities have since been extended to allow the invocation of a 'compute' menu for a

specific field of a form. To activate the scripts the user is presented a blank form and a menu of activities. When an activity is selected by the user the corresponding script is run which in turns fills in the form as the user desires.

The typical data base operations such as create, update, delete and retrieve are programmed at a different level than user scripts and are hidden from the user.

2.8 *SYNOPSIS*

Chapter two has been a discussion on the electronic form, our IDO, in an automated office environment. The next chapter will begin to be more specific about the data base interface of the IDO.

3. *CHAPTER THREE - DATA BASE AND MESSAGE SYSTEMS*

Until recently, there has not been much emphasis on integrating data base management systems and message or information retrieval systems. This was due in part to the fact that these areas of research and development were separated in most organizations. The data base management studies were largely done by persons with a computing background while the message systems research was done by persons with a communications and networks background.

Today, however, with microprocessor technology as it is, computers are becoming more tied to network theory. The distributed data bases available today imply that networks and data base management sciences are more closely related than first thought.

Many applications require an integrated data base management system and message/information system. Some examples are hospital information systems on patients' data, library information and retrieval systems, pharmaceutical data bases and office information systems. Common to all of these examples is the fact that it is "necessary to combine text management and retrieval with usual formatted data manipulation. Therefore, a single user interface is necessary." [SCH/PIS] To understand why the initial approach

to data base systems and message systems was made separately it is important to look at the two concepts involved: communicating through data bases and communicating through messages. [TSI81]

By examining these two areas more closely one can begin to see that, even though the initial basis of study had been different, many similarities exist between communicating through data bases and communicating through messages. As these similarities emerge, one can begin to appreciate how the IDO (as a form message) may be stored effectively in the data base management system.

The *target* of the communication is the first basis for why the two areas were studied separately. Generally, data base systems imply that anyone with access to the systems is eligible to access the information contained therein. Unless access rights are issued for the data base the default is a broadcast operation. In a message system, individuals or at least a well defined group, are usually the object of a message. Initially, the audience for the two types of communication had been varied; data base systems tended to talk to the masses while message systems were aimed at an individual or selected subset of persons. This situation is changing though with data base views

making individual targetting feasible and electronic mail systems allowing message broadcasting to large groups.

Another area that initially was cause for the separate research of data base systems and message systems deals with *recording* of the information. Data base transactions are suppose to have lasting importance. To make sense of the transaction a related record should be physically present. Messages, on the other hand, were generally thought of as transient. They were not recorded just communicated. However, now it is recognized that data base records can be as temporary as the recording of a message while messages can be made as permanent as a record.

Data base transactions and messages generally differ also in *structure*. Data base formats usually appear more formal with transactions issued with respect to some schema. Messages have evolved from a format-free perspective and therefore can be sent without using a schema. This difference is slowly disappearing. Data base research shows the need for handling text, graphs, etc. as an integrated part of the data base. This would require a more flexible structure for data base transactions. Message research also reveals that messages may often be of a structured variety, with, for instance, a header and body.

A significant difference between data base transactions and messages is the concept of *ownership*. The data base record is theoretically "owned" by a third party (the data base.) Temporary ownership of the record is managed by the user that initiates the transaction and other users who observe the effects of the transaction. Messages are considered private property; first they are owned by the sender and then they are owned by the recipient. This difference seems small when one views the data base record as being "owned" temporarily be a user who may in fact be locking others from using the record. Afterwards, the record is released and reverts back to the system's ownership.

Many of the differences mentioned in this view of data base transactions and messages have an historical basis. Or, in some cases, a different emphasis may have been applied to the data base transaction vs. the message. When a broader perspective is taken and consideration is given the purposes of data base systems and message systems, it can be said there are no substantial differences between the two systems. In automated office systems there is a real effort and need to integrate these two areas of communication.

In the IDO project under study, the data base interface that is performed is an example where communicating through the

data base has merged with communicating through a message system. An automated office environment, such as one that would accommodate the IDO, takes the electronic form message and stores it in a data base file where it can be manipulated by the customary data base operations. Thus, the IDO project under study is a prime example of integrating data base and message systems.

The next chapter reviews Ingres, the data base management system used for the project. Introductory background information to Ingres is presented along with examples of how queries to the DBMS work.

4. *CHAPTER FOUR: INGRES - A RELATIONAL DATA BASE MANAGEMENT SYSTEM*

The data base management system used for the project is Ingres, a product of Relational Technologies, Inc. Before introducing Ingres, its design philosophy and implementation considerations, some background information relevant to this project will be presented on DBMS systems.

4.1 *STRUCTURING DATA IN A DATA BASE*

There are three commonly accepted models of logically structuring data in a data base; the network model, hierarchical model and relational model. [KRA84]

The network data model is a structured model (graphical) in which the entities (things) are represented by records and the relationships between these things are represented as arcs. The record types which are represented commonly as files can be related to any other record type. This leads to a many-to-many data structuring between entities. There may be many owner files for a single member file in the network model. Although relatively flexible and most efficient (generally), the network model is the most complex system for humans to understand and use.

The hierarchical model is a degenerate case of a network structure. Here the owner entity relationship is one-to-many with the member entities. The relationships between entities must always be represented as a tree. This is a less flexible system than the network model but it is relatively efficient. Its great simplicity makes it more understandable and usable for the human.

A relational data base system such as Ingres, is one in which the entities and relationships are all stored in a common logical structure called a relation. A relation can be thought of as a table or a file of information. It is made up of relations (records) comprised of various attributes (fields). Records that share common fields are stored in the same file or table. Records from different files can be combined according to the rules in the relational algebra or calculus. This system is by far the easiest to understand and modify, however, it is extremely difficult to implement on a vonNeumann machine such that the system runs efficiently. Entries in the relations (tables) must be single-valued; this does not allow repeating groups or arrays. Relations are also called flat files. All entries of an attribute (column) must be from the same attribute domain (type). Each column has a unique name. The order of the attributes (columns) and the entries (rows)

is not important. No two rows in a table are identical.

Not all relational data bases designed using the relational model are equally good. In some poor designs, changing data can lead to unexpected undesirable results, called modification anomalies. [KRO83] There are three types of anomalies called update, insertion and deletion anomalies. Anomalies can be eliminated by changing the data base design. A way of eliminating the anomalies is called the normalization process.

"Good" data base designs are structured by clearly understanding the data and the relationships that are present between data items. By recognizing the purpose of the data being stored one can then structure the data base in such a way to avoid the problems encountered by the anomalies that might occur. This area of data base design will not be addressed further since it is beyond the scope of this paper. Good judgement and sound data base design considerations have gone into the structuring of any data base files mentioned from hereon. The data base designed for the project has been normalized to 3NF. This normal form minimizes the maintenance anomalies discussed above and is considered a reasonable level of normalization.

4.2 *INGRES - HISTORICAL*

The data base management system used on the project's data base interface is Ingres, as mentioned earlier. Ingres evolved over a number of years at the University of California, Berkeley, California. [ALL/STO82] The initial design and implementation period was from 1973 to 1976. The project directors were Michael Stonebraker and Eugene Wong who acted largely as creative consultants. There were very little software engineering principals applied during this phase with their goal to "make the system work" regardless of the methods used. About 60,000 lines of code was produced during this time.

The next phase of the evolution was from 1976 to 1978. Much effort was spent on making improvements to the system during this time. Approximately ten different sites were using Ingres as their DBMS. With their feedback, and from the group's desire to make improvements to the system, a number of changes were made to Ingres. During these years adding crash recovery, concurrency control, additional access methods and algorithmic modifications took considerable time.

More users were attracted to Ingres by a growing positive reputation the system achieved. Early feedback from the

installations had been very helpful to the design enhancements made to the system but as users were added, with increasing less sophistication, the feedback became more concerned with the reference manual, misinterpretations or the Unix (a trademark of AT&T Bell Laboratories) operating system. The University of California people began to act more like a software house and less like a research group at this point.

Since 1978 the project has expanded its goals to include development in areas such as networking, language processing, and operating systems. With these added areas of development it has become increasingly more difficult for the varied development groups, lead by Lawrence Rowe and Michael Stonebraker, to set priorities and goals.

4.3 *INGRES - DESIGN AND IMPLEMENTATION*

Ingres (Interactive Graphics and Retrieval System) was designed considering the Unix environment in which it would run. [STO/WON/KRE76] There are two ways that Ingres can be invoked. First, it can be directly accessed from Unix by executing "Ingres data-base-name". Secondly, it can be invoked by using EQUEL, a precompiler program.

4.3.1 *DIRECT ACCESS* When one invokes Ingres from Unix a four process structure is created. See Figure 5.

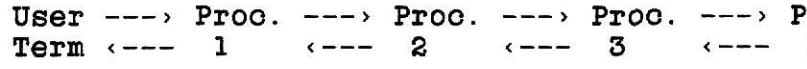

Figure 5. PROCESS STRUCTURE

Process 1 is an interactive terminal monitor which allows the user to edit, print form and execute collections of Ingres commands. In this process a workspace is maintained for the user while that user is interacting with the data base. When the interaction is completed the contents of the workspace is passed to process 2 as an ASCII character string.

Process 2 contains a lexical analyzer, a parser, query modification routines for integrity control and concurrency control. Physical locks that lock all required resources for a user are the means by which concurrency control is implemented. This avoids deadlock rather than preventing it. When process 2 finishes, it passes a string of tokens to process 3.

Process 3 accepts the string of tokens from process 2. Internal to this process are execution routines for the

commands Retrieve, Replace, Append. Here the Command is either RETRIEVE, APPEND, REPLACE or DELETE. In the case of RETRIEVE and APPEND, result-name is the name of the relation which will be created or to which data will be appended. For the other two commands, REPLACE and DELETE, Result-name is the name of a tuple variable which, depending on the qualification, indentifies the relations to be replaced or deleted. The target-list is a list of the form

Result-domain = Function ...

where the Result-domains are domain names in the result relation which are to be assigned the value of the corresponding function.

Given the sample relation from Figure 6 a few examples will be presented. See Figure 7 through Figure 10 for these examples.

	Dept	Floor #
Dept	Toy	1
	Candy	2
	Admin	3

	Name	Dept	Salary	Manager	Age
Emp	Smith	Toy	10000	Jones	25
	Jones	Toy	15000	Johnson	32
	Adams	Candy	12000	Baker	36
	Johnson	Toy	14000	Harding	29
	Baker	Admin	20000	Harding	47
	Harding	Admin	40000	none	58

Figure 6. SAMPLE RELATION

Example 1: Insert the tuple (Humphrey, candy, 16000, Baker, 40) into Emp

```
APPEND TO EMP (NAME = "Humphrey",
        DEPT = "candy", SALARY = 16000,
        Manager = "Baker", AGE = 40)
```

Here, the result relation EMP is modified by adding the indicated tuple to the relation.

Figure 7. INGRES - EXAMPLE 1

Example 2: Give a 12 percent raise to Smith if she works on the first floor.

```
RANGE OF E IS EMP
RANGE OF D IS DEPT
REPLACE E(SALARY by 1.12*E.SALARY)
        WHERE E.NAME = "Smith"
        AND E.DEPT = D.DEPT
        AND D.FLOOR # = 1
```

Here, E.SALARY is to be replaced by 1.12*E.SALARY for those tuples in EMP where the qualification is true.

Figure 8. INGRES EXAMPLE 2

Example 3: Replace the salary of all toy department employees by the maximum toy department salary.

```
RANGE OF E IS EMP
REPLACE E(SALARY BY MAX(E.SALARY
        WHERE E.DEPT = "Toy"))
WHERE E.DEPT = "Toy"
```

Here, MAX is to be taken of the salary domain for those tuples satisfying the qualification E.DEPT = "Toy". This MAX value will be substituted into E.SALARY for those tuples satisfying the condition E.DEPT = "Toy".

Figure 9. INGRES EXAMPLE 3

Example 4: Find the departments whose average salary exceeds the company's average salary, both averages taken on those employees whose salary exceeds $9,000.

```
RANGE OF E IS EMP
RETRIEVE INTO HIGHPAY (E.DEPT)
        WHERE AVG(E.SALARY BY E.DEPT
                WHERE E.SALARY › 9000)›AVG
                (E.SALARY WHERE E.SALARY›9000)
```

Here, AVG(E.SALARY BY E.DEPT WHERE E.SALARY › 9000) is an AGGREGATE FUNCTION and takes a value for each value of E.DEPT. This value is the aggregate AVG(E.SALARY WHERE E.SALARY ›9000) AND (E.DEPT = value).
The qualification expression for the statement is then true for departments for which this aggregate function exceeds the aggregate AVG(E.SALARY WHERE E.SALARY › 9000).

Figure 10. INGRES EXAMPLE 4

4.3.2 *UTILITY COMMANDS* Ingres also supports a variety of utility commands in addition to the QUEL commands already mentioned. These utility commands can be classified into seven main types. See Figure 11 for a listing of these commands and a brief description of them.

COMMANDS	*DESCRIPTION*
INGRES data-base-name	invokes Ingres, this logs-in a user to a given data base
CREATDB data-base-name DESTROYDB data-base-name	creation and destruction of data bases; proper authorization must exist for the user before this can be invoked
CREATE relname (domain-name IS format,etc) DESTROY relname	creation and destruction of relations; the creator of the relation becomes the 'owner' who may also destroy it
COPY relname (domain-name IS format,etc) direction "filename" PRINT relname	bulk copy of data; copy transfers an entire relation to or *from* a Unix file; print copies a relation to the user's terminal
MODIFY relname TO storage-structure ON (key1, key2,...) INDEX ON relname IS indexname (key1, key2,...)	storage structure modification; modify changes storage structure from one access method to another; index creates a a secondary index for a relation
INTEGRITY CONSTRAINT is qualification INTEGRITY CONSTRAINT LIST relname INTEGRITY CONTRAINT OFF relname INTEGRITY CONSTRAINT OFF (int,...,int) RESTORE data-base-name	consistency and integrity control; the first four commands support activities of integrity constraints enforced for all interactions with a relation; restore is used by the DBA to restore data base after a system crash
HELP [relname/manual-sec] SAVE relname UNTIL exp-date RELKILLER data-*base-name*	misc; Help gives information about system or data-base; Save allows time to keep relation; Relkiller deletes expired relations

Figure 11. UTILITY COMMANDS

Many of these utility commands along with the QUEL commands are used to create, structure and maintain relations in Ingres. Relations in Ingres are also called data bases. The individual members of the relation are referred to as a tuple. The columns of the relation are the domain set of that relation. A relation can't have more than 49 domains and the tuple width cannot exceed 498 bytes. Since Unix allocates space on a disk in units of 512 byte page there is a small amount of overhead per page. No tuple is split between two pages.

The utility command CREATDB is used to create a data base through Unix shell. The user who does this becomes the data base administrator (DBA) for that data base. This enables that user to run sysmod, restore and purge on that data base. It also, by default, allows multiple concurrent users.

Two ways to create new relations in Ingres are to use CREATE and RETRIEVE INTO. "Create" gives a new relation with no tuples in it. "Retrieve Into" is used to form a new relation from one that already exists.

If information is required on a relation the HELP command can be used. When it is used in this manner ‹HELP data-base-name› the following information would be printed with

the specific data for that relation supplied:

```
Relation:        ---
Owner:           ___
Tuple width:     ___
Saved until:     ---
No of tuples:    ---
Storage struc:   ---
Relation type:   ---

Attribute name     Type     Length     Key No.
      .              .         .          .
      .              .         .          .
      .              .         .          .
```

4.3.3 *STORAGE STRUCTURE* In the utility command "MODIFY" storage structure was mentioned. Ingres can store a relation in three different internal structures. [EPS77] The first is called "heap". Whenever a relation is created it is automatically created as a "heap". Two properties of a heap are that duplicate tuples are not removed and nothing is recorded about the location of the tuple. When large relations are stored as a heap it may take quite a while (minutes to hours) for searches to be completed.

The second means of storage is called "hash". When a relation is stored in hash structure there are no duplicate tuples. This is also called a keyed domain since searches are done on specified domains which are stored in some pattern on the disk and the system knows approximately where to look.

"Isam" is the third structure for storage. The relation is sorted on one or more domains also referred to as keyed domains. There are no duplicates on isam stored relations. When new tuples are added they are inserted approximately in their sorted order.

Compression can be applied to any of these three storage structures by specifying this in the MODIFY statement. This reduces the internal amount of space each tuple needs by suppressing trailing blanks in character domains.

Many other considerations that won't be elaborated on in this paper went into the design and implementation of Ingres. New features are emerging as this process continues. Several features, in addition to those already discussed that are available today on Ingres, will only be mentioned here. [RTI84] They can be very valuable features of a data base management system. Included in this set are QUERY (Query by Forms; QBF), FORMS (Visual Forms Editor; VIFRED), REPORTS (Report by Forms), GRAPHICS (Graph by Forms), APPLICATIONS-BY-FORMS (ABF) and NET (Ingres' Network).

This chapter has included discussion on the topics of data base design and the data base system Ingres, used on the project. Many considerations and alternatives must be weighed before a "good" design for a data base has been

achieved.

5. *CHAPTER FIVE: PROBLEM TO BE SOLVED*

This chapter describes two problems involving the data base interfaces which need to be solved for the IDO project. The IDO, represented by a form routing itself around an automated office environment, requires data from the user to instantiate instances of that type. When this data is provided it must be stored in a data base file for future retrievals and updates. See Figure 12.

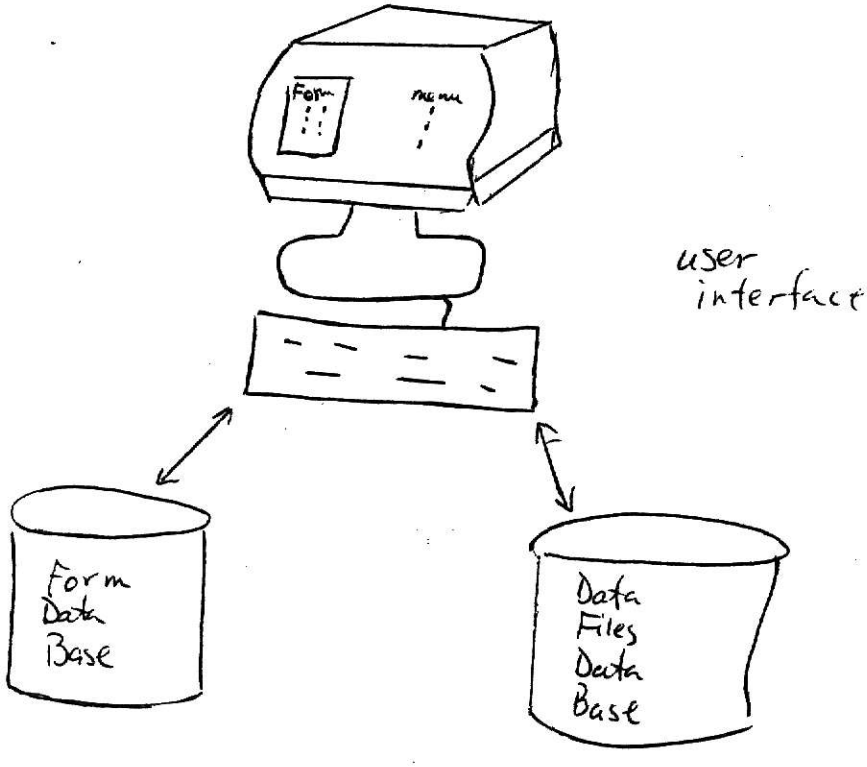

Figure 12. DATA BASE INTERFACE

There exists two separate and distinct possibilities here, then, for an implementation project for the IDO data base interface.

The first project possibility will be called form operations. For this project a user would interface with a terminal to edit, copy, file and mail form instances. While the form is being created the user would be posed queries from a form data base interface that is collecting enough information from the user to build a form instance. It might request the form type and then enough information about some of the key fields on that form type to begin building an image of that form. If it is a new form type, the user would be posed questions to build the form type, its history, routing and other internal instructions necessary for that form type.

For a form instance of an established form type, some fields would be filled in automatically based upon the particular form type and its internal intelligence that signals retrievals from/to data files within other data bases. The IDO has a sequence of instructions internal to it to allow calculations upon the retrieved data object. The data base queries in the form operation project would pose questions to the user about the particular form type instance being

built, while at the same time drawing information from data base files that are required to complete that form instance.

Data that the form requires for building a form instance of a particular form type might be kept in the form data base. Other sources of information for building the form instance might also come from the user or the node where that form instance is being built. This data includes each form instance's unique identification number, routing instructions and processing instructions.

A second possibility for a project pertains to data base retrievals and maintenance operations using the IDO concept. Based on the form types being used in the automated office environment there would be a variety of data base files required. These files would contain data that might not only be supplied by a user during a form instance building session, as described in the form operation project, but also data that is required for historical and calculation purposes.

In this project a user who is familiar with the form types and their associated data bases would interactively work at a terminal and perform a variety of operations. Under the category of data base retrievals would be operations that Ingres provides for this purpose. Included in this category

are the operations Retrieve, Retrieve Into, Retrieve Unique, Sort By, Aggregations and Join. The user would be given a choice about the operation to perform and then be posed interactive queries to complete the operation.

In maintenance operations a user would complete various updates to a data base using the Ingres commands Append, Replace and Delete. This also would be done interactively between a user and the data bases being queried.

These two distinct possibilities for IDO data base interface projects pose for an interesting choice of project assignment. The one chosen to be addressed in this particular study is the second one, data base retrievals and update operations. The next chapter will outline how this project will be designed.

6. *CHAPTER SIX: DESIGN OF IMPLEMENTATION PROJECT*

This chapter describes the design of the implementation project. The first section gives a sample data base that would be accessed by the user making queries and is populated with data to be used for the data base retrievals and maintenance operations. The second section explains how the user would interact with the data base.

6.1 *DATA BASE*

6.1.1 *DESIGN OF SAMPLE DATA BASE* The office environment chosen for use as an example in the IDO data base interface project is one that has responsibilities for tracking orders placed to certain suppliers of various part types. This office buys parts for a small manufacturing facility at the same location. When a buyer (an office employee who has the responsibility of buying parts from the suppliers) places an order, a record is kept of when the order was placed, the promised date of delivery from the supplier, how the parts are to be paid for and how the parts are to be shipped to the manufacturing site.

To design the data base, functional dependencies were established for the attributes and their relationships and keys were identified for each functional dependency (FD).

See Figure 13. The data base design has been shown to be in 3rd normal form by using Berstein's 2nd algorithm. See the appendix for the computer output from Bern2. A data dictionary for those attributes used in the project's sample data base has been developed. See the appendix for this listing.

File Name	Functional Dependencies
Supplier	Supplier_Num -› Supplier_Name
Deliver	Supplier_Num, Part_num -› Qty, Date_ Ordered, Date_Promised, Payment_ Method, Transportation_Method
Order	Part_Num -› Order_Num
Buyer	Buyer_Num -› Buyer_Name
Part	Part_Name -› Part_Num

Figure 13. SAMPLE DATA BASE

6.1.2 *INGRES* To create these data base relations Ingres requires that the data base be opened first. To invoke Ingres on a new data base the command is ‹ $ creatdb project › where 'project' is the name given to the data base file being created. Once a data base has been created another command ‹ $ ingres project › is run. This creates a table in our data base by opening the data base. When the table has been created, a wide range of Ingres commands can then be performed on the data in the table.

Assume the files used in the data base interface project have been previously populated with data from several sources. These sources might include users interacting with the files, office workers suppling historical information required such as each supplier's identification number and each supplier's name, and data that may have been placed in the data base file by a calculation. In the Figure 13 example, Qty (quantity of parts of that part number from that supplier) may have been a calculation from a week's worth of orders to that supplier for that part.

When data is entered to the data base file by whomever, that person must have a basic knowledge of Ingres commands for this system to work. A new employee told to execute the command to enter data into a new data base file would require some instructions on how to do it. So, this project assumes that people using the Ingres commands have had some preliminary training before they are expected to use them. It is also necessary for persons using the data base files to have some familiarity with the file structure and the contents of each file.

In addition to people directly placing data in the data base, data may also be placed in the data base through some activities that might result from user interaction at the

nodes with the IDO. This was described briefly in chapter five as another approach to the data base interface project for an IDO.

Either method of initially placing data in the data base is considered feasible for this project. The next section describes how a user would interface with the data that already exists in the file.

6.2 *USER INTERFACE WITH DATA BASE FILES*

The user of the code written for the data base retrieval and maintenance system will be required to run each module of code independently. There are five separate modules that the user may select from to perform basic operations on the data base. These five are append, delete, replace, retrieve and join operations.

To run the desired module the user must type in the following commands:

```
equel filename.q
cc filename.c -lq
a.out
```

To assist the user of the system a revised method has been provided whereby the user need only type in the desired

operation name to invoke the operation. For example, if one were in the file "/usrb/att/busaok/project" and the append operation is desired, then all one must do is type "append" at the prompt to invoke the append module. The appropriate queries will appear on the screen to prompt the user for appends to the data base named "project" already established.

All the basic operations required for the data base maintenance are provided in the commands Append, Replace, and Delete. Append allows the user to add a record to a data base; Replace allows substitutions in a record; Delete allows one to eliminate a record that is no longer required. Retrieve and Join gives the user a means of pulling out records from the data base file in a variety of combinations.

The code for these commands is embodied in the C programming language and EQUAL statements. As described in Chapter 4, EQUEL is embedded QUEL statements that is used with the C code to perform the Ingres data base operations.

Code from all the modules can be seen in the appendix. Also in the appendix is a data dictionary for the sample data base.

7. *CHAPTER SEVEN: CONCLUDING REMARKS*

7.1 *PROJECT*

The IDO has been a fascinating area of study for a master's project. It has been discussed enough in the literature so that at least one may read what a few others have learned from studying it. At the same time, however, the IDO is such a recently developed idea that many aspects remain to be investigated and studied.

The preceding master's report has been a brief summary of the literature reviewed. Many articles that were read were not used in this report, not because they were poor articles, but, rather due to time and space constraints. The premier researchers in this area though have been represented in this document. Among these authors are Gehani, Liskov, Zilles and Tsichritzis.

Completing the implementation project that accompanied this report has also been a learning experience. Programming in C and EQUEL was a unique experience. Several code revisions were necessary before the final versions were reached. One major point that was learned while writing the code was that C statements may not be interspersed with EQUEL commands. Due to the interactive nature of the project, all answers to

queries had to be stored using a "gets" or "scanf" statement and storing the results in a character pointer or integer value.

7.2 *EXTENSIONS*

One extension for the implementation project would be to provide the user with a menu to drive the available Ingres modules. Additional modules also might be developed to give the user more flexibility in the data base operations. Included might be Retrieve modules for Into, Unique, Sort By, and Aggregation operations.

The code for the data base operations was written for a specific example. Another extension to the present project might be to make the code more general to accommodate other data base relations.

A third extension relates to the discussion from Chapter 5 concerning the two project possibilities. Addressed in this paper was the second possibility discussed where data base retrievals and maintenance operations were developed. The first possibility for a project would be another area for development. This would allow a user to interface with a terminal to edit, copy, file and mail form instances.

8. *REFERENCES*

[ALL/STO82] Allman, E.and Stonebraker, M. Observations on the Evolution of a Software System, *IEEE*, June, 1982.

[BER82] Bernal, M. Interface Concepts for Electronic Design and Manipulation, *Office Information Systems*, N. Naffra(ed), 1982.

[ELL/NUT80] Ellis, C. A. and Nutt G. J. Office Information Systems and Computer Science, *Computing Surveys*, Vol. 12, No. 1, March, 1980.

[EPS77] Epstein, R. Creating and Maintaining a Database Using Ingres, University of California, Berkeley, California, Memorandum No. ERL-M77-71, December 16, 1977.

[GEH81] Gehani, N. H. An Electronic Form System: An Experience in Prototyping, Bell Laboratories, Technical Memorandum, 81-11359-9, May 1, 1981.

[GEH83] Gehani, N. H. High Level Form Definition in Office Information Systems, *The Computer*

Journal, Vol. 26, No. 1, 1983.

[GEH82] Gehani, N. H. The Potential of Forms in Office Automation, *IEEE Transactions on Communication*, Vol. Com-30, No. 1, January, 1982.

[GRI/GEH77] Gries, D. and Gehani, N. Some Ideas on Data Types in High-Level Languages, *Communications of the ACM*, Vol. 20, No. 6, June, 1977.

[KRA84] Krajewski, R. Database Types, *Byte*, October, 1984.

[KRO83] Kroenke, D. *Database Processing: Fundamentals, Design, Implementation*, 1983.

[LIS/ZIL] Liskov, B. and Zilles, S. Programming with Abstract Data Types.

[McB/UNG83] McBride, R. A. and Unger, E. A. Modeling Jobs in a Distributed System, *ACM*, 1983.

[RTI84] Relational Technology, Inc. *An Introduction to Ingres*, Berkely, California, 1984.

[SCH/PIS] Schek, H. -J. and Pistor, P. Data Structures for an Integrated Data Base Management and Information Retrieval System, IBM Scientific

Center, Heidelberg, W. Germany.

[SHU/LUM/TUN/CHA82] Shu, N. C. and Lum, V. Y. and Tung, F. C. and Chang, C. L. Specification of Forms Processing and Business Procedures for Office Automation, *IEEE on Software Engineering*, Vol. SE-8, No. 5, September, 1982.

[STO/WON/KRE76] Stonebraker, M. and Wong, E. and Kreps, P. *The Design and Implementation of Ingres*, University of California, Berkely, January, 1976.

[TSI80] Tsichritzis, D. OFS: An Integrated Form Management System, *IEEE*, 1980.

[TSI81] Tsichritzis, D. Integrating Data Base and Message Systems, *IEEE*, 1981.

[ZLO77] Zloof, M. M. Query-By-Example: A Data Base Language, *IBM Syst J.*, No. 4, 1977.

9. *APPENDIX ONE: BERN2*

```
 THE INPUT TO THE PROGRAM IS :
 K > H ;

 L > B ;

 B > J ;

 A, B > C, D, E, F, G ;

 A > I ;

 END.
```

 THIS IS THE LIST OF ATTRIBUTES WITH THEIR ABBREVIATIONS.

```
K00  K
H00  H
L00  L
B00  B
J00  J
A00  A
C00  C
D00  D
E00  E
F00  F
G00  G
I00  I
```

 THE TOKENS MARKED *TRUE* ARE EXTRANEOUS IN THE FDS :

```
 FD NUMBER :001 TOKEN: K00
F
 FD NUMBER :002 TOKEN: L00
F
 FD NUMBER :003 TOKEN: B00
F
 FD NUMBER :004 TOKEN: B00
F
 FD NUMBER :004 TOKEN: A00
F
 FD NUMBER :005 TOKEN: A00
F
 FD NUMBER :005 TOKEN: B00
F
 FD NUMBER :006 TOKEN: B00
F
 FD NUMBER :006 TOKEN: A00
F
 FD NUMBER :007 TOKEN: A00
F
 FD NUMBER :007 TOKEN: B00
F
 FD NUMBER :008 TOKEN: B00
F
 FD NUMBER :008 TOKEN: A00
```

```
F
 FD NUMBER :009 TOKEN: A00
F
 THE REDUNDANT FDS ARE MARKED *TRUE* :
FD-NUMBR001
F
FD-NUMBR002
F
FD-NUMBR003
F
FD-NUMBR004
F
FD-NUMBR005
F
FD-NUMBR006
F
FD-NUMBR007
F
FD-NUMBR008
F
FD-NUMBR009
F
 THE FOLOWING FDS HAVE THE SAME LHS AND ARE THEREFORE GROUPED
 TOGETHER INTO PARTITION CLASSES:


 PARTITION CLASS-NUMBER 001:008007006005004
 PARTITION CLASS-NUMBER 002:009
 PARTITION CLASS-NUMBER 003:003
 PARTITION CLASS-NUMBER 004:002
 PARTITION CLASS-NUMBER 005:001
002
003
001
004
005
 THE FOLLOWING FDS ARE REDUNDANT AFTER ADDING THE BIJECTIONS
 TO THE FD STRUCTURE :

 THIS IS THE SCHEMA IN 3NF  :
  (A  ) ) I


  (B  ) ) J


  (B A  ) ) G  F  E  D  C


  (L  ) ) B


  (K  ) ) H
```

10. *APPENDIX TWO: DATA DICTIONARY*

DATA DICTIONARY

NAME: Buyer_Name
DEFINITION: First and last name of person making purchase of part from supplier.
TYPE: Text
FORMAT: x(20)
VALUE: alphanumeric

NAME: Buyer_Num
DEFINITION: Unique identifying number of buyer who made purchase.
TYPE: Text
FORMAT: xxx
VALUE: 0 - 9

NAME: Date_Ord
DEFINITION: Data that part was ordered from supplier.
TYPE: Date
FORMAT: MM/DD/YY
VALUE: MM:01-12 DD:01-31 YY:00-99

NAME: Date_Prom
DEFINITION: Date supplier promised to deliver part ordered.
TYPE: Date
FORMAT: MM/DD/YY
VALUE: MM:01-12 DD:01-31 YY:00-99

NAME: Order_Num
DEFINITION: Unique number that identifies order placed by buyer to part supplier.
TYPE: Text
FORMAT: xxxx-xx
VALUE: (0-9) - (0-9)

NAME: Part_Name
DEFINITION: Official name of part.
TYPE: Text
FORMAT: x(20)
VALUE: Alphanumeric

NAME: Part_Num
DEFINITION: Unique identifying number for each part.
TYPE: Text

FORMAT: xx-xxx
VALUE: (A-Z) - (0-9)

NAME: Pay_Meth
DEFINITION: Means by which the buyer has agreed to pay for the parts ordered from the supplier.
TYPE: Text
FORMAT: xxxx
FORMAT: Alphanumeric

NAME: Qty
DEFINITION: Total number of parts ordered for that respective part.
TYPE: Integer
FORMAT: xxx
VALUE: 0 - 9

NAME: Sup_Name
DEFINITION: Supplier's official company name.
TYPE: Text
FORMAT: x(50)
VALUE: Alphanumeric

NAME: Sup_Num
DEFINITION: Unique identifying number of supplier.
TYPE: Text
FORMAT: xxxx
VALUE: 0 - 9

NAME: Trans_Meth
DEFINITION: Means by which supplier will transport part to manufacturing location.
TYPE: Text
FORMAT: x(15)
VALUE: Alphanumeric

11. *APPENDIX THREE: APPEND CODE*

```
/*
*  This is an EQUEL program.  It will Append
*  tuples to files within a database.  It assumes
*  Project is the data base and Order, Supplier,
*  Buyer, Deliver, and Part are the relations.
*
*  To compile and run this program do the following:
*
*  equel append.q
*  cc append.c -lq
*  a.out
*/
#include<stdio.h>
#define EQUAL 0
main()
{
## char range_var[10];
## char *var;
## char data_base[10];
## char *dbname;
## char column_var[10][50];
## char *cvar[10];
## char value_char[10][50];
## char *vchar[10];
## int value_int;
## int vint[10];
## char column[50];
## char *col;
## int intval;
## int ivalue;
## char charval[50];
## char *cvalue;
    char ans1[2], ans3[2], ans5[2], ans6[2];
    int ans2,ans4,h,j,m,n;
    char num[2];
    char ans7[2];
    char r;


    printf ("what data base do you want to work in?\n\n");
    scanf ("%s",data_base);
    dbname = data_base;
##    ingres dbname
##    help
    strcpy (ans7,"y");
    while ((strcmp(ans7,"y"))==EQUAL)
    {
        printf ("what is name of the table?\n\n");
        scanf ("%s",range_var);
        var = range_var;
##    range of var is var
##    help var
##    print var
        if ((strcmp(range_var,"deliver"))==EQUAL)
        {
            printf ("do you have any integer values to enter?\n");
            printf ("print 'y' for yes or 'n' for no.\n");
            scanf ("%s", ans1);
            ans2 = 0;
            printf ("ans1 = %s\n",ans1);
            if ((strcmp(ans1,"y"))==EQUAL)
            {
                printf ("print the column variable name and the associated");
                printf ("integer value to be\n appended to the");
                printf (" file in the form 'column_var newline integer_value'\n");
                scanf ("%s",column_var);
                cvar[0] = (char *) column_var;
                printf ("cvar = %s\n",cvar[n]);
                scanf ("%d",&value_int);
                scanf ("%c",&r);
                vint[0] = value_int;
                printf ("vint = %d\n",vint[n]);
            }
            printf ("do you have any character string values to enter?\n ");
            printf ("print 'y' for yes or 'n' for no.\n");
            scanf ("%s",ans3);
            if ((strcmp(ans3,"y"))==EQUAL)
            {
                printf ("how many columns will be entered with character");
```

```
                printf (" string value(s)?\n type the number required.\n");
                printf ("a maximum of 6 is allowed.\n");
                scanf ("%d",&ans4);
                scanf ("%c",&r);
                printf ("print the column variable name(s) and the associated ");
                printf ("character value(s) to be appended\n");
                printf ("to the file in the format 'column_var newline character_value.'\n");
                for ( m=1; m<=ans4;m++)
                {
                    cvar[m] = (char *) gets(column_var[m]);
                    printf ("cvar =%s\n",cvar[m]);
                    vchar[m] = (char *) gets(value_char[m]);
                    printf ("vchar = %s\n",vchar[m]);
                }
            }
            if ((strcmp(ans1,"y"))==EQUAL)
            {
                if ((strcmp(ans3,"y"))==EQUAL)
                {
                    if (ans4 ==1)
                    {
##              append to var (cvar[0] = vint[0], cvar[1] = vchar[1])
                    }
                    if (ans4 ==2)
                    {
##              append to var (cvar[0] = vint[0], cvar[1] = vchar[1],
##                             cvar[2] = vchar[2])
                    }
                    if (ans4 ==3)
                    {
##              append to var (cvar[0] = vint[0], cvar[1] = vchar[1],
##                  cvar[2] = vchar[2], cvar[3] = vchar[3])
                    }
                    if (ans4 ==4)
                    {
##              append to var (cvar[0] = vint[0], cvar[1] = vchar[1],
##                  cvar[2] = vchar[2], cvar[3] = vchar[3],
##                  cvar[4] = vchar[4])
                    }
                    if (ans4 == 5)
                    {
##              append to var (cvar[0] = vint[0], cvar[1] = vchar[1],
##                  cvar[2] = vchar[2], cvar[3] = vchar[3],
##                  cvar[4] = vchar[4], cvar[5] = vchar[5])
                    }
                    if (ans4 == 6)
                    {
##              append to var (cvar[0] = vint[0], cvar[1] = vchar[1],
##                  cvar[2] = vchar[2], cvar[3] = vchar[3],
##                  cvar[4] = vchar[4], cvar[5] = vchar[5],
##                  cvar[6] = vchar[6])
                    }
                }
                else
##              append to var (cvar[0] = vint[0])
            }
            else
            {
                if (ans4 == 1)
                {
##          append to var (cvar[1] = vchar[1])
                }
                if (ans4 == 2)
                {
##          append to var (cvar[1] = vchar[1], cvar[2] = vchar[2])
                }
                if (ans4 == 3)
                {
##          append to var (cvar[1] = vchar[1], cvar[2] = vchar[2],
##              cvar[3] = vchar[3])
                }
                if (ans4 == 4)
                {
##          append to var (cvar[1] = vchar[1], cvar[2] = vchar[2],
##              cvar[3] = vchar[3], cvar[4] = vchar[4])
                }
                if (ans4 == 5)
                {
##          append to var (cvar[1] = vchar[1], cvar[2] = vchar[2],
```

```
##              cvar[3] = vchar[3], cvar[4] = vchar[4],
##              cvar[5] = vchar[5])
                  }
                  if (ans4 == 6)
                  {
##          append to var (cvar[1] = vchar[1], cvar[2] = vchar[2],
##              cvar[3] = vchar[3], cvar[4] = vchar[4],
##              cvar[5] = vchar[5], cvar[6] = vchar[6])
                  }
            }
          }
          else
          {
              printf ("how many columns will be entered with character");
              printf (" string value(s)?\n type the number required.\n");
              printf ("a maximum of 2 is allowed.\n");
              scanf ("%d",&ans4);
              scanf ("%c",&r);
              printf ("print the column variable name(s) and the associated ");
              printf ("character value(s) to be appended\n");
              printf ("to the file in the format 'column_var newline character_value.'\n");
              for ( m=0; m<=ans4-1;m++)
              {
                  cvar[m] = (char *) gets(column_var[m]);
                  printf ("cvar =%s\n",cvar[m]);
                  vchar[m] = (char *) gets(value_char[m]);
                  printf ("vchar = %s\n",vchar[m]);
              }
              if (ans4 == 1)
              {
##          append to var (cvar[0] = vchar[0])
              }
              if (ans4 == 2)
              {
##          append to var (cvar[0] = vchar[0], cvar[1] = vchar[1])
              }
              if (ans4 == 3)
              {
##          append to var (cvar[0] = vchar[0], cvar[1] = vchar[1],
##              cvar[2] = vchar[2])
              }
              if (ans4 == 4)
              {
##          append to var (cvar[0] = vchar[0], cvar[1] = vchar[1],
##              cvar[2] = vchar[2], cvar[3] = vchar[3])
              }
              if (ans4 == 5)
              {
##          append to var (cvar[0] = vchar[0], cvar[1] = vchar[1],
##              cvar[2] = vchar[2], cvar[3] = vchar[3],
##              cvar[4] = vchar[4])
              }
              if (ans4 == 6)
              {
##          append to var (cvar[0] = vchar[0], cvar[1] = vchar[1],
##              cvar[2] = vchar[2], cvar[3] = vchar[3],
##              cvar[4] = vchar[4], cvar[5] = vchar[5])
              }
          }
          printf ("do you have any more rows to append?\n");
          printf ("type 'y' for yes or 'n' for no.\n");
          scanf ("%s",ans7);
      }
##    print var
##    exit
      }
```

12. *APPENDIX FOUR: DELETE CODE*

```
/*
*  This is an EQUEL program.  It will Delete
*  tuples within a database.  It assumes
*  Project is the data base and Order, Supplier,
*  Buyer, Deliver, and Part are the relations.
*
*  To compile and run this program do the following:
*
*  equel delete.q
*  cc delete.c -lq
*  a.out
*/
# include<stdio.h>
#define EQUAL 0
main()
{

## char range_var[10];
## char *var;
## char column_var[50];
## char *cvar;
## int op,compar_int;
## char compar_char[50];
## char *compchar;
## char data_base[10];
## char *dbname;
    char ans1[2];
    char ans2[2];
    char ans3[2];
    char r;

    printf ("what data base do you want to work in?\n\n");
    scanf ("%s",data_base);
    dbname = data_base;
##    ingres dbname
##    help
    strcpy (ans3,"y");
    while ((strcmp(ans3,"y"))==EQUAL)
    {
        printf ("what is name of the table?\n\n");
        scanf ("%s",range_var);
        var = range_var;
##    range of var is var
##    print var
        printf ("is column name required for comparison?\n");
        printf ("type 'y' for yes or 'n' for no.\n\n");
        scanf ("%s",ans1);
        if ((strcmp(ans1,"n"))==EQUAL)
##    delete var
        else
        {
            printf ("give column name.\n");
            scanf ("%s",column_var);
            cvar = column_var;
            printf ("what comparison operator will be used?\n");
            printf ("type one of the following numbers:\n");
            printf ("1 for '='\n");
            printf ("2 for '<'\n");
            printf ("3 for '>'\n");
            printf ("4 for '<='\n");
            printf ("5 for '>='\n");
            printf ("6 for '!='\n");
            scanf ("%d",&op);
            scanf ("%c",&r);
            printf ("is the  comparison value an integer or character?\n");
            printf (" type 'i' for integer ");
            printf ("or 'c' for character.\n\n");
            scanf ("%s",ans2);
            printf ("type the value for comparison to be used?\n");
            if ((strcmp(ans2,"i"))==EQUAL)
            {
                scanf ("%d",&compar_int);
                scanf ("%c",&r);
            }
            else
            {
                scanf ("%c",&r);
```

```
                compchar = (char *) gets(compar_char);
            }
            if ((strcmp(ans2,"i"))==EQUAL)
                switch (op)
                {
                case 1:
##                delete var where var.cvar
##                                    = compar_int
                    break;

                case 2:
##                delete var where var.cvar
##                    < compar_int
                    break;

                case 3:
##                delete var where var.cvar
##                    > compar_int
                    break;

                case 4:
##                delete var where var.cvar
##                    <= compar_int
                    break;

                case 5:
##                delete var where var.cvar
##                    >= compar_int
                    break;

                case 6:
##                delete var where var.cvar
##                    != compar_int
                    break;
                }
            else
                switch (op)
                {
                case 1:
##                delete var where var.cvar
##                    = compchar
                    break;

                case 2:
##                delete var where var.cvar
##                    < compchar
                    break;

                case 3:
##                delete var where var.cvar
##                    > compchar
                    break;

                case 4:
##                delete var where var.cvar
##                    <= compchar
                    break;

                case 5:
##                delete var where var.cvar
##                    >= compchar
                    break;

                case 6:
##                delete var where var.cvar
##                    != compchar
                    break;
                }
            printf ("do you have more rows to delete?\n");
            printf ("type 'y' for yes or 'n' for no.\n");
            scanf ("%s",ans3);
        }
    }
##  print var
##  exit
}
```

13. *APPENDIX FIVE: REPLACE CODE*

```
/*
*  This is an EQUEL program.  It will Replace
*  tuples within a database.  It assumes
*  Project is the data base and Order, Supplier,
*  Buyer, Deliver, and Part are the relations.
*
*  To compile and run this program do the following:
*
*  equel replace.q
*  cc replace.c -lq
*  a.out
*/
#include<stdio.h>
#define EQUAL 0
main()
{
## char range_var[10];
## char *var;
## char data_base[10];
## char *dbname;
## char column_var[10][50];
## char *cvar[10];
## char value_char[10][50];
## char *vchar[10];
## int value_int;
## int vint[10];
## char quals[100];
## char *qual;
## char column[50];
## char *col;
## int intval;
## int ivalue;
## char charval[50];
## char *cvalue;
    char ans1[2], ans3[2], ans5[2], ans6[2];
    int ans2,ans4,h,j,m,n;
    char num[2];
    char ans7[2];
    char r;


    printf ("what data base do you want to work in?\n\n");
    scanf ("%s",data_base);
    dbname = data_base;
##      ingres dbname
##      help
    strcpy (ans7,"y");
    while ((strcmp(ans7,"y"))==EQUAL)
    {
        printf ("what is name of the table?\n\n");
        scanf ("%s",range_var);
        var = range_var;
##      range of var is var
##      help var
##      print var
        if ((strcmp(range_var,"deliver"))==EQUAL)
        {
            printf ("do you have any integer values to enter?\n");
            printf ("print 'y' for yes or 'n' for no.\n");
            scanf ("%s", ans1);
            ans2 = 0;
            printf ("ans1 = %s\n",ans1);
            if ((strcmp(ans1,"y"))==EQUAL)
            {
                printf ("print the column variable name and the associated");
                printf ("integer value to be\n replaced in the");
                printf (" file in the form 'column_var newline integer_value'\n");
                scanf ("%s",column_var);
                cvar[0] = (char *) column_var;
                printf ("cvar = %s\n",cvar[n]);
                scanf ("%d",&value_int);
                scanf ("%c",&r);
                vint[0] = value_int;
                printf ("vint = %d\n",vint[n]);
            }
            printf ("do you have any character string values to enter?\n ");
            printf ("print 'y' for yes or 'n' for no.\n");
            scanf ("%s",ans3);
            if ((strcmp(ans3,"y"))==EQUAL)
```

```
{
    printf ("how many columns will be entered with character");
    printf (" string value(s)?\n type the number required.\n");
    printf ("a maximum of 6 is allowed.\n");
    scanf ("%d",&ans4);
    scanf ("%c",&r);
    printf ("print the column variable name(s) and the associated ");
    printf ("character value(s) to be replaced\n");
    printf ("in the file in the format 'column_var newline character_value.'\n");
    for ( m=1; m<=ans4;m++)
    {
        cvar[m] = (char *) gets(column_var[m]);
        printf ("cvar =%s\n",cvar[m]);
        vchar[m] = (char *) gets(value_char[m]);
        printf ("vchar = %s\n",vchar[m]);
    }
}
printf ("is there a qualification to this replace? \n");
printf ("type 'y' for yes or 'n' for no.\n\n");
scanf ("%s",ans5);
if ((strcmp(ans5,"y"))==EQUAL)
{
    printf ("print your qualification in the format used for ");
    printf ("entering the column variable name(s) and the ");
    printf ("associated value(s),\ni.e., 'column_var, newline,  ");
    printf ("character value' for a character constraint\n");
    scanf ("%c",&r);
    col = (char *) gets(column);
    printf ("col is %s\n",col);
    cvalue = (char *) gets(charval);
    printf ("cvalue is %s\n",cvalue);
    qual = cvalue;
}
if ((strcmp(ans5,"y"))==EQUAL)
{
    if ((strcmp(ans1,"y"))==EQUAL)
    {
        if ((strcmp(ans3,"y"))==EQUAL)
        {
            if (ans4 ==1)
            {
##          replace var (cvar[0] = vint[0], cvar[1] = vchar[1])
##          where var.col = qual
            }
            if (ans4 ==2)
            {
##          replace var (cvar[0] = vint[0], cvar[1] = vchar[1],
##                       cvar[2] = vchar[2])
##          where var.col = qual
            }
            if (ans4 ==3)
            {
##          replace var (cvar[0] = vint[0], cvar[1] = vchar[1],
##                 cvar[2] = vchar[2], cvar[3] = vchar[3])
##          where var.col = qual
            }
            if (ans4 ==4)
            {
##          replace var (cvar[0] = vint[0], cvar[1] = vchar[1],
##                 cvar[2] = vchar[2], cvar[3] = vchar[3],
##                 cvar[4] = vchar[4])
##          where var.col = qual
            }
            if (ans4 == 5)
            {
##          replace var (cvar[0] = vint[0], cvar[1] = vchar[1],
##                 cvar[2] = vchar[2], cvar[3] = vchar[3],
##                 cvar[4] = vchar[4], cvar[5] = vchar[5])
##          where var.col = qual
            }
            if (ans4 == 6)
            {
##          replace var (cvar[0] = vint[0], cvar[1] = vchar[1],
##                 cvar[2] = vchar[2], cvar[3] = vchar[3],
##                 cvar[4] = vchar[4], cvar[5] = vchar[5],
##                 cvar[6] = vchar[6])
##          where var.col = qual
            }
        }
```

```
                    else
                    {
##                      replace var (cvar[0] = vint[0])
##                      where var.col = qual
                    }
                }
                else
                {
                    if (ans4 == 1)
                    {
##              replace var (cvar[1] = vchar[1])
##                      where var.col = qual
                    }
                    if (ans4 == 2)
                    {
##              replace var (cvar[1] = vchar[1], cvar[2] = vchar[2])
##                      where var.col = qual
                    }
                    if (ans4 == 3)
                    {
##              replace var (cvar[1] = vchar[1], cvar[2] = vchar[2],
##                      cvar[3] = vchar[3])
##                      where var.col = qual
                    }
                    if (ans4 == 4)
                    {
##              replace var (cvar[1] = vchar[1], cvar[2] = vchar[2],
##                      cvar[3] = vchar[3], cvar[4] = vchar[4])
##                      where var.col = qual
                    }
                    if (ans4 == 5)
                    {
##              replace var (cvar[1] = vchar[1], cvar[2] = vchar[2],
##                      cvar[3] = vchar[3], cvar[4] = vchar[4],
##                      cvar[5] = vchar[5])
##                      where var.col = qual
                    }
                    if (ans4 == 6)
                    {
##              replace var (cvar[1] = vchar[1], cvar[2] = vchar[2],
##                      cvar[3] = vchar[3], cvar[4] = vchar[4],
##                      cvar[5] = vchar[5], cvar[6] = vchar[6])
##                      where var.col = qual
                    }
                }
            }
            else
            {
                if ((strcmp(ans1,"y"))==EQUAL)
                {
                    if ((strcmp(ans3,"y"))==EQUAL)
                    {
                        if (ans4 ==1)
                        {
##                      replace var (cvar[0] = vint[0], cvar[1] = vchar[1])
                        }
                        if (ans4 ==2)
                        {
##                      replace var (cvar[0] = vint[0], cvar[1] = vchar[1],
##                                   cvar[2] = vchar[2])
                        }
                        if (ans4 ==3)
                        {
##                      replace var (cvar[0] = vint[0], cvar[1] = vchar[1],
##                              cvar[2] = vchar[2], cvar[3] = vchar[3])
                        }
                        if (ans4 ==4)
                        {
##                      replace var (cvar[0] = vint[0], cvar[1] = vchar[1],
##                              cvar[2] = vchar[2], cvaZ[] = vchar[3],
##                              cvar[4] = vchar[4])
                        }
                        if (ans4 == 5)
                        {
##                      replace var (cvar[0] = vint[0], cvar[1] = vchar[1],
##                              cvar[2] = vchar[2], cvar[3] = vchar[3],
##                              cvar[4] = vchar[4], cvar[5] = vchar[5])
                        }
                        if (ans4 == 6)
```

```
                    }
##                  replace var (cvar[0] = vint[0], cvar[1] = vchar[1],
##                          cvar[2] = vchar[2], cvar[3] = vchar[3],
##                          cvar[4] = vchar[4], cvar[5] = vchar[5],
##                          cvar[6] = vchar[6])
                    }
                }
                else
##                  replace var (cvar[0] = vint[0])
            }
            else
            {
                if (ans4 == 1)
                {
##              replace var (cvar[1] = vchar[1])
                }
                if (ans4 == 2)
                {
##              replace var (cvar[1] = vchar[1], cvar[2] = vchar[2])
                }
                if (ans4 == 3)
                {
##              replace var (cvar[1] = vchar[1], cvar[2] = vchar[2],
##                      cvar[3] = vchar[3])
                }
                if (ans4 == 4)
                {
##              replace var (cvar[1] = vchar[1], cvar[2] = vchar[2],
##                      cvar[3] = vchar[3], cvar[4] = vchar[4])
                }
                if (ans4 == 5)
                {
##              replace var (cvar[1] = vchar[1], cvar[2] = vchar[2],
##                      cvar[3] = vchar[3], cvar[4] = vchar[4],
##                      cvar[5] = vchar[5])
                }
                if (ans4 == 6)
                {
##              replace var (cvar[1] = vchar[1], cvar[2] = vchar[2],
##                      cvar[3] = vchar[3], cvar[4] = vchar[4],
##                      cvar[5] = vchar[5], cvar[6] = vchar[6])
                }
            }
        }
##      print var
    }
    else
    {
        printf ("how many columns will be entered with character");
        printf (" string value(s)?\n type the number required.\n");
        printf ("a maximum of 2 is allowed.\n");
        scanf ("%d",&ans4);
        scanf ("%c",&r);
        printf ("print the column variable name(s) and the associated ");
        printf ("character value(s) to be replaced\n");
        printf ("in the file in the format 'column_var newline character_value.'\n");
        for ( m=0; m<=ans4-1;m++)
        {
            cvar[m] = (char *) gets(column_var[m]);
            printf ("cvar =%s\n",cvar[m]);
            vchar[m] = (char *) gets(value_char[m]);
            printf ("vchar = %s\n",vchar[m]);
        }
        printf ("is there a qualification to this replace? \n");
        printf ("type 'y' for yes or 'n' for no.\n\n");
        scanf ("%s",ans5);
        if ((strcmp(ans5,"y"))==EQUAL)
        {
            printf ("print your qualification in the format used for ");
            printf ("entering the column variable name(s) and the ");
            printf ("associated value(s),\ni.e., 'column_var, ");
            printf ("newline,character value' for a character ");
            printf ("contraint.\n");
            scanf ("%c",&r);
            col = (char *) gets(column);
            printf ("col is %s\n",col);
            cvalue = (char *) gets(charval);
            printf ("cvalue is %s\n",cvalue);
        }
```

```
            if ((strcmp(ans5,"y"))==EQUAL)
            {
                if (ans4 == 1)
                {
##              replace var (cvar[0] = vchar[0])
##              where var.col = cvalue
                }
                if (ans4 == 2)
                {
##              replace var (cvar[0] = vchar[0], cvar[1] = vchar[1])
##              where var.col = cvalue
                }
                if (ans4 == 3)
                {
##              replace var (cvar[0] = vchar[0], cvar[1] = vchar[1],
##                      cvar[2] = vchar[2])
##              where var.col = cvalue
                }
                if (ans4 == 4)
                {
##              replace var (cvar[0] = vchar[0], cvar[1] = vchar[1],
##                      cvar[2] = vchar[2], cvar[3] = vchar[3])
##              where var.col = cvalue
                }
                if (ans4 == 5)
                {
##              replace var (cvar[0] = vchar[0], cvar[1] = vchar[1],
##                      cvar[2] = vchar[2], cvar[3] = vchar[3],
##                      cvar[4] = vchar[4])
##              where var.col = cvalue
                }
                if (ans4 == 6)
                {
##              replace var (cvar[0] = vchar[0], cvar[1] = vchar[1],
##                      cvar[2] = vchar[2], cvar[3] = vchar[3],
##                      cvar[4] = vchar[4], cvar[5] = vchar[5])
##              where var.col = cvalue
                }
            }
            else
            {
                if (ans4 == 1)
                {
##              replace var (cvar[0] = vchar[0])
                }
                if (ans4 == 2)
                {
##              replace var (cvar[0] = vchar[0], cvar[1] = vchar[1])
                }
                if (ans4 == 3)
                {
##              replace var (cvar[0] = vchar[0], cvar[1] = vchar[1],
##                      cvar[2] = vchar[2])
                }
                if (ans4 == 4)
                {
##              replace var (cvar[0] = vchar[0], cvar[1] = vchar[1],
##                      cvar[2] = vchar[2], cvar[3] = vchar[3])
                }
                if (ans4 == 5)
                {
##              replace var (cvar[0] = vchar[0], cvar[1] = vchar[1],
##                      cvar[2] = vchar[2], cvar[3] = vchar[3],
##                      cvar[4] = vchar[4])
                }
                if (ans4 == 6)
                {
##              replace var (cvar[0] = vchar[0], cvar[1] = vchar[1],
##                      cvar[2] = vchar[2], cvar[3] = vchar[3],
##                      cvar[4] = vchar[4], cvar[5] = vchar[5])
                }
            }
##      print var
        }
        printf ("do you have any more rows to update?\n");
        printf ("type 'y' for yes or 'n' for no.\n");
        scanf ("%s",ans7);
    }
##  exit
```

14. *APPENDIX SIX: RETRIEVE CODE*

```
/*
*  This is an EQUEL program.  It will Retrieve
*  tuples from files within a database.  It assumes
*  Project is the data base and Order, Supplier,
*  Buyer, Deliver, and Part are the relations.
*
*  To compile and run this program do the following:
*
*  equel retrieve.q
*  cc retrieve.c -lq
*  a.out
*/

# include<stdio.h>
#define EQUAL 0
main()
{

## char range_var[10];
## char *var;
## char column_var[10][50], column_var2[50];
## char *cvar[10], *cvar2;
## char cname1[50],cname2[50], cname3[50], cname4[50];
## int iname1;
## int op, compar_int;
## char compar_char[50];
## char *charptr;
## char data_base[10];
## char *dbname;
    char ans1[2], ans3[2], ans4[2], ans5[2], ans6[2];
    int ans2,m,n;
    char r;


    printf ("what data base do you want to work in?\n\n");
    scanf ("%s",data_base);
    dbname = data_base;
##      ingres dbname
##      help
    strcpy (ans5,"y");
    while ((strcmp(ans5,"y"))==EQUAL)
    {
        printf ("what is name of the table?\n\n");
        scanf ("%s",range_var);
        var = range_var;
##      range of var is var
##      help var
##      print var
        if ((strcmp(range_var,"deliver"))==EQUAL)
        {
            printf ("do you want to retrieve the integer column?\n");
            printf ("print 'y' for yes or 'n' for no.\n");
            scanf ("%s",ans4);
            if ((strcmp(ans4,"y"))==EQUAL)
            {
##          retrieve (iname1 = var.qty)
##                  {
                    printf ("%d\n",iname1);
##                  }
            }
        }
        printf ("how many character columns will be viewed for their ");
        printf ("contents?\n  type the number required.\n");
        printf ("a maximum of 4 is allowed.\n");
        scanf ("%d",&ans2);
        scanf ("%c",&r);
        printf ("ans2 is %d\n",ans2);
        printf ("print the column variable name(s) ");
        printf ("to be retrieved in the file.\n");
        for (n=0;n<= ans2-1;n++)
        {
            cvar[n] = (char *) gets (column_var[n]);
            printf ("cvar is %s\n",cvar[n]);
        }
        printf ("is there a qualification to this retrieve?\n");
        printf ("type 'y' for yes or 'n' for no.\n\n");
        scanf ("%s",ans3);
        if ((strcmp(ans3,"n"))==EQUAL)
```

```
        {
            if (ans2 == 1)
            {
##      retrieve(cname1 = var.cvar[0])
##              {
                printf("%s\n",cname1);
##              }
            }
            if (ans2 == 2)
            {
##      retrieve (cname1 = var.cvar[0], cname2 = var.cvar[1])
##              {
                printf ("%s,%s\n",cname1,cname2);
##              }
            }
            if (ans2 == 3)
            {
##      retrieve (cname1 = var.cvar[0], cname2 = var.cvar[1],
##                cname3 = var.cvar[2])
##              {
                printf ("%s,%s,%s\n",cname1,cname2,cname3);
##              }
            }
            if (ans2 == 4)
            {
##      retrieve (cname1 = var.cvar[0], cname2 = var.cvar[1],
##                cname3 = var.cvar[2], cname4 = var.cvar[3])
##              {
                printf ("%s,%s,%s,%s\n",cname1,cname2,cname3,cname4);
##              }
            }
        }
        else
        {
            scanf ("%c",&r);
            printf ("give character column name required for comparison.\n");
            cvar2 = (char *) gets (column_var2);
            printf ("cvar2 is %s\n",cvar2);
            printf ("what comparison operator will be used?\n");
            printf ("type one of the following numbers:\n");
            printf ("1 for '='\n");
            printf ("2 for '<'\n");
            printf ("3 for '>'\n");
            printf ("4 for '<='\n");
            printf ("5 for '>='\n");
            printf ("6 for '!='\n");
            scanf ("%d",&op);
            scanf ("%c",&r);
            printf ("op is %d\n",op);
            printf ("type the value for comparison to be used?\n\n");
            charptr = (char *) gets (compar_char);
            if (ans2 == 1)
                switch (op)
                {
                case 1:
##                              retrieve (cname1 =var.cvar[0])
##                              where var.cvar2 = charptr
##                              {
                    printf ("%s\n",cname1);
##                              }
                    break;

                case 2:
##                              retrieve (cname1 =var.cvar[0])
##                              where var.cvar2 < charptr
##                              {
                    printf ("%s\n",cname1);
##                              }
                    break;

                case 3:
##                              retrieve (cname1 =var.cvar[0])
##                              where var.cvar2 > charptr
##                              {
                    printf ("%s\n",cname1);
##                              }
                    break;

                case 4:
```

```
##                             retrieve (cname1 =var.cvar[0])
##                             where var.cvar2 <= charptr
##                             {
                        printf ("%s\n",cname1);
##                             }
                        break;

                    case 5:
##                             retrieve (cname1 =var.cvar[0])
##                             where var.cvar2 >= charptr
##                             {
                        printf ("%s\n",cname1);
##                             }
                        break;

                    case 6:
##                             retrieve (cname1 =var.cvar[0])
##                             where var.cvar2 != charptr
##                             {
                        printf ("%s\n",cname1);
##                             }
                    }
                if (ans2 == 2)
                    switch (op)
                    {
                    case 1:
##                             retrieve (cname1 =var.cvar[0], cname2 = var.cvar[1])
##                             where var.cvar2 = charptr
##                             {
                        printf ("%s,%s\n",cname1,cname2);
##                             }
                        break;

                    case 2:
##                             retrieve (cname1 =var.cvar[0], cname2 = var.cvar[1])
##                             where var.cvar2 < charptr
##                             {
                        printf ("%s,%s\n",cname1,cname2);
##                             }
                        break;

                    case 3:
##                             retrieve (cname1 =var.cvar[0], cname2 = var.cvar[1])
##                             where var.cvar2 > charptr
##                             {
                        printf ("%s,%s\n",cname1,cname2);
##                                      break;

                    case 4:
##                             retrieve (cname1 =var.cvar[0], cname2 = var.cvar[1])
##                             where var.cvar2 <= charptr
##                             {
                        printf ("%s,%s\n",cname1,cname2);
##                             }
                        break;

                    case 5:
##                             retrieve (cname1 =var.cvar[0], cname2 = var.cvar[1])
##                             where var.cvar2 >= charptr
##                             {
                        printf ("%s,%s\n",cname1,cname2);
##                             }
                        break;

                    case 6:
##                             retrieve (cname1 =var.cvar[0], cname2 = var.cvar[1])
##                             where var.cvar2 != charptr
##                             {
                        printf ("%s,%s\n",cname1,cname2);
##                             }
                    }
                if (ans2 == 3)
                    switch (op)
                    {
                    case 1:
##                             retrieve (cname1 =var.cvar[0], cname2 = var.cvar[1],
##                                     cname3 = var.cvar[2])
##                             where var.cvar2 = charptr
##                             {
```

```
                printf ("%s,%s,%s\n",cname1,cname2,cname3);
##                      }
                break;

            case 2:
##                      retrieve (cname1 =var.cvar[0], cname2 = var.cvar[1],
##                              cname3 = var.cvar[2])
##                      where var.cvar2 < charptr
##                      {
                printf ("%s,%s,%s\n",cname1,cname2,cname3);
##                      }
                break;

            case 3:
##                      retrieve (cname1 =var.cvar[0], cname2 = var.cvar[1],
##                              cname3 = var.cvar[2])
##                      where var.cvar2 > charptr
##                      {
                printf ("%s,%s,%s\n",cname1,cname2,cname3);
##                      }
                break;

            case 4:
##                      retrieve (cname1 =var.cvar[0], cname2 = var.cvar[1],
##                              cname3 = var.cvar[2])
##                      where var.cvar2 <= charptr
##                      {
                printf ("%s,%s,%s\n",cname1,cname2,cname3);
##                      }
                break;

            case 5:
##                      retrieve (cname1 =var.cvar[0], cname2 = var.cvar[1],
##                              cname3 = var.cvar[2])
##                      where var.cvar2 >= charptr
##                      {
                printf ("%s,%s,%s\n",cname1,cname2,cname3);
##                      }
                break;

            case 6:
##                      retrieve (cname1 =var.cvar[0], cname2 = var.cvar[1],
##                              cname3 = var.cvar[2])
##                      where var.cvar2 != charptr
##                      {
                printf ("%s,%s,%s\n",cname1,cname2,cname3);
##                      }
            }
        if (ans2 == 4)
            switch (op)
            {
            case 1:
##                      retrieve (cname1 =var.cvar[0], cname2 = var.cvar[1],
##                              cname3 = var.cvar[2], cname4 = var.cvar[3])
##                      where var.cvar2 = charptr
##                      {
                printf ("%s,%s,%s,%s\n",cname1,cname2,cname3,cname4);
##                      }
                break;

            case 2:
##                      retrieve (cname1 =var.cvar[0], cname2 = var.cvar[1],
##                              cname3 = var.cvar[2], cname4 = var.cvar[3])
##                      where var.cvar2 < charptr
##                      {
                printf ("%s,%s,%s,%s\n",cname1,cname2,cname3,cname4);
##                      }
                break;

            case 3:
##                      retrieve (cname1 =var.cvar[0], cname2 = var.cvar[1],
##                              cname3 = var.cvar[2], cname4 = var.cvar[3])
##                      where var.cvar2 > charptr
##                      {
                printf ("%s,%s,%s,%s\n",cname1,cname2,cname3,cname4);
##                      }
                break;

            case 4:
```

```
##                                  retrieve (cname1 =var.cvar[0], cname2 = var.cvar[1],
##                                          cname3 = var.cvar[2], cname4 = var.cvar[3])
##                                  where var.cvar2 <= charptr
##                                  {
                        printf ("%s,%s,%sname2,cname3,cname4);
##                                  }
                        break;

                case 5:
##                                  retrieve (cname1 =var.cvar[0], cname2 = var.cvar[1],
##                                          cname3 = var.cvar[2], cname4 = var.cvar[3])
##                                  where var.cvar2 >= charptr
##                                  {
                        printf ("%s,%s,%s,%s\n",cname1,cname2,cname3,cname4);
##                                  }
                        break;

                case 6:
##                                  retrieve (cname1 =var.cvar[0], cname2 = var.cvar[1],
##                                          cname3 = var.cvar[2], cname4 = var.cvar[3])
##                                  where var.cvar2 != charptr
##                                  {
                        printf ("%s,%s,%s,%s\n",cname1,cname2,cname3,cname4);
##                                  }
                }
        }
##      print var
        printf ("do you have more columns to retrieve?\n");
        printf ("type 'y' for yes or 'n' for no.\n");
        scanf ("%s",ans5);
    }
##      exit
}
```

15. *APPENDIX SEVEN: JOIN CODE*

```
/*
*  This is an EQUEL program.  It will Join
*  tuples within a database.  It assumes
*  Project is the data base and Order, Supplier,
*  Buyer, Deliver, and Part are the relations.
*
*  To compile and run this program do the following:
*
*  equel join.q
*  cc join.c -lq
*  a.out
*/

# include<stdio.h>
#define EQUAL 0

main()
{


## char range_var1[10];
## char *var1;
## char range_var2[10];
## char *var2;
## char table[10];
## char *tableptr;
## char column_var1[50];
## char *cvar1;
## char column_var2[50];
## char *cvar2;
## char cname1[50];
## char cname2[50];
## char comcol[50];
## char *comptr;
## char col[50];
## char *colptr;
## int op,compar_int;
## char compar_char[50];
## char *charptr;
## char data_base[10];
## char *dbname;
    char r;
    char ans3[2];


    printf ("what data base do you want to work in?\n\n");
    scanf ("%s",data_base);
    dbname = data_base;
##    ingres dbname
##    help
    strcpy (ans3,"y");
    while ((strcmp (ans3,"y"))==EQUAL)
    {
        printf ("what is name of the first table?\n\n");
        scanf ("%s",range_var1);
        printf ("table one is %s\n",range_var1);
        var1 = range_var1;
##    range of var1 is var1
##    help var1
##    print var1
        printf ("what is name of the second table?\n\n");
        scanf ("%s",range_var2);
        printf ("table two is %s\n",range_var2);
        var2 = range_var2;
##    range of var2 is var2
##    help var2
##    print var2
        scanf ("%c",&r);
        printf ("print the column variable name to be ");
        printf ("retrieved from table one.\n");
        cvar1 = (char *) gets (column_var1);
        printf ("table 1 col is %s\n",cvar1);
        printf ("print the column variable name to be ");
        printf ("retrieved from table two.\n");
        cvar2 = (char *) gets (column_var2);
        printf ("table 2 col is %s\n",cvar2);
        printf ("what is the column that is common to ");
        printf ("these 2 tables?\n  print the name of ");
        printf ("that column.\n");
```

```
        comptr = (char *) gets (comcol);
        ;
        printf ("common col is %s\n",comptr);
        printf ("what is the qualification to this retrieve?\n");
        printf ("give the table name, newline, column name required ");
        printf ("for the join.\n");
        tableptr = (char *) gets (table);
        printf ("table qual is %s\n",tableptr);
        colptr = (char *) gets (col);
        printf ("col qual is %s\n",colptr);
        printf ("what comparison operator will be used?\n");
        printf ("type one of the following numbers:\n");
        printf ("1 for '='\n");
        printf ("2 for '<'\n");
        printf ("3 for '>'\n");
        printf ("4 for '<='\n");
        printf ("5 for '>='\n");
        printf ("6 for '!='\n");
        scanf ("%d",&op);
        scanf ("%c",&r);
        printf ("op is %d\n",op);
        printf ("type the character value for comparison to be used?\n");
        charptr = (char *) gets (compar_char);
        printf ("compare val is %s\n\n",charptr);
        if ((strcmp(ans3,"y"))==EQUAL)
            switch (op)
            {
            case 1:

##              retrieve (cname1 = var1.cvar1, cname2 = var2.cvar2)
##              where var1.comptr = var2.comptr
##              and tableptr.colptr = charptr
##              {
                printf ("%s,%s\n\n",cname1,cname2);
##              }
                break;

            case 2:

##              retrieve (cname1 = var1.cvar1, cname2 = var2.cvar2)
##              where var1.comptr = var2.comptr
##              and tableptr.colptr < charptr
##              {
                printf ("%s,%s\n\n",cname1,cname2);
##              }
                break;

            case 3:

##              retrieve (cname1 = var1.cvar1, cname2 = var2.cvar2)
##              where var1.comptr = var2.comptr
##              and tableptr.colptr > charptr
##              {
                printf ("%s,%s\n\n",cname1,cname2);
##              }
                break;

            case 4:

##              retrieve (cname1 = var1.cvar1, cname2 = var2.cvar2)
##              where var1.comptr = var2.comptr
##              and tableptr.colptr <= charptr
##              {
                printf ("%s,%s\n\n",cname1,cname2);
##              }
                break;

            case 5:

##              retrieve (cname1 = var1.cvar1, cname2 = var2.cvar2)
##              where var1.comptr = var2.comptr
##              and tableptr.colptr >= charptr
##              {
                printf ("%s,%s\n\n",cname1,cname2);
##              }
                break;

            case 6:
```

```
##                retrieve (cname1 = var1.cvar1, cname2 = var2.cvar2)
##                where var1.comptr = var2.comptr
##                and tableptr.colptr != charptr
##                {
                printf ("%s,%s\n\n",cname1,cname2);
##              }

            }
        printf ("do you desire to try more joins?\n");
        printf ("type 'y' for yes or 'n' for no.\n");
        scanf ("%s",ans3);
    }
##  exit
}
```

THE INTELLIGENT DATA OBJECT AND ITS DATA BASE INTERFACE

BY

NANCY LONG BUSACK

B. A., M. S., Ohio University, 1976, 1979

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1985

1. *ABSTRACT*

This paper, "The IDO and It's Data Base Interface", discusses the Intelligent Data Object project with particular attention given to the data base interface for the IDO.

The IDO is defined as a superset of an abstract data type. The abstract data type is a data structure and the set of operations defined on the object. An intelligent data object extends the set of operations to provide for "triggered" actions, retrievals and routing within a potentially distributed system. The IDO is used to implement an electronic form and is described in the context of an automated office system.

The integration of data base management systems and message systems is described prior to introducing the data base management system, Ingres, used on the project. A DBMS such as Ingres has the capability to store and retrieve the information extracted from the electronic form used in an automated office environment. Some specifics of how Ingres was designed and used are described.

Two options for an implementation project for the IDO data base retrieval are presented. One deals with interacting with a form data base, user and node to collect data to build the desired form. A second possibility is data base

retrievals and maintenance operations using the IDO concept.

The second approach is chosen as the implementation project and the design of it is described in more detail. A user will be instructed on the methods needed to direct the activities that user wants to have performed on the data base files.

1430-60
CD-53