Design and Implementation of A Relational Information Management System in Turbo-Pascal (PRIMS)

Dissertation submitted in partial fulfilment of the requirements for the award of the Degree of

MASTER OF PHILOSOPHY

(COMPUTER SCIENCE)

PARAMJIT SINGH

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI - 110067

1987

JAWAHARLAL NEHRU UNIVERSITY,

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES,

NEW DELHI - 110067.

CERTIFICATE

This is to certify that the dissertation entitled "Design and Implementation of a Relational Information Management System in turbo-Pascal" submitted by Paramjit Singh is in partial fulfilment of the requirement for the award of degree of Master of Philosophy.

The work is origional and has not been submitted, in part or full, elsewhere for the award of a degree.

(Dr. R. C. PHOHA)
Supervisor

(Prof. K. K. NAMBIAR)
Dean

DEDICATED TO PAPA

ACKNOWLEDGEMENT

I remain indebted to many people during the preparation of this dissertation.

First and foremost, I wish to express my deep gratitude to my supervisor, Dr. R. C. Phoha, for his scholarly guidance and encouragement at every stage of this work.

I also wish to express my heartfelt thanks to Prof. K. K. Nambiar, Dean, School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi, for providing the necessary facilities to complete this research.

I shall be failing in my duty if I do not thank my senior Ashwani who provided me with lots of insights throughtout the course of this work.

Last but not the least, my sincere thanks to all other faculty members and collegues for their helpful comments and co-operation.

Paramjit Singh

(PARAMJIT SINGH)

CONTENTS

CHAPTER

1

DATABASE SYSTEMS

—AN OVERVIEW

* INTRODUCTION
* WHY DATABASE ?
* AN ARCHITECTURE FOR A DATABASE SYSTEM
* DATABASE MODELS WITH RELATIVE ADVANTAGES
* RELATIONAL DATABASE TERMINOLOGY

INTRODUCTION

What exactly is a database? Basically, it is nothing more than a computer-based recordkeeping system : that is, a system whose overall purpose is to record and maintain information.

A database, then, is a repository for stored data. In general, it is both integrated and shared.

By "integrated" we mean that the database may be thought of as a unification of several otherwise distinct data files, with any redundancy among those files partially or wholly eliminated.

By "shared" we mean that individual pieces of data in the database may be shared among several different users, in the sense that each of those users may have access to the same piece of data,and may use it for different purposes. Sharing, in fact, is a consequence of the fact that the database is integrated. The term "shared" is frequently extended to cover, not only sharing as just described, but also concurrent sharing : that is, the ability for several different users to be actually accessing the database (possibly even the same piece of data) at the same time. A database system supporting this form of sharing is sometimes referred to as a multiuser system.

WHY DATABASE ?

The broad answer to this question is that a database

system provides the enterprise with centralized control of its operational data,which is one of its most valuable assets. This is in sharp contrast to the situation that prevails in many enterprises today, where typically each application has its own private files (quite often its own private tapes and disk packs, too) so that the operational data is widely dispersed, and is therefore probably difficult to control. This implies that in an enterprise with a database system there will be some one identifiable person who has this central responsibility for the operational data. This person is the database administrator (DBA).

Let us consider some of the advantages that accrue from having centralized control of the data.

Redundancy can be reduced.

In nondatabase systems each application has its own private files.This can often lead to considerable redundancy in stored data, with resultant waste in storage space. In database environment, the files can be integrated, and the redundancy eliminated,if the DBA is aware of the data requirements for all applications.

We do not mean to suggest that all redundancy should necessarily be eliminated. Sometimes there are sound business or technical reasons for maintaining multiple copies of the same data. In a database system,

however, redundancy should be controlled - that is,the system should be aware of the redundancy and should assume responsibility for propagating updates.

Inconsistency can be avoided.

This, actually, follows from the previous point. Suppose a particular fact is represented by two distinct entries in the database, and the system is not aware of this duplication (in other words,the redundancy is not controlled). Then there will be some occasions on which the two entries will not agree (that is one and only one has been updated). At such times the database is said to be inconsistent. Obviously, a database that is in an inconsistent state is capable of supplying incorrect or conflicting information.

It is clear that if the given fact is represented by a single entry (i.e.,if the redundancy is removed) such an inconsistency cannot occur. Alternatively, if the redundancy is not removed but is controlled, then the system could guarantee that the database is never inconsistent as seen by the user, by ensuring that any change made to either of the two entries is automatically made to the other. This process is known as propagating updates.

The data can be shared.

We have already mentioned about this. But this point is

so important that we stress it again here. It means not only that existing applications can share the data in the database, but also that new applications can be developed to operate against that same stored data. In other words, the data requirements of new applications may be satisfied without having to create any new stored files.

Security restrictions can be applied.

Having complete jurisdiction over the operational data, the DBA (a) can ensure that the only means of access to the database is through the proper channels,and hence (b) can define authorization checks to be carried out whenever access to sensitive data is attempted. Different checks can be established for each type of access (retrieve, modify, delete, etc.) to each piece of information in the database.

Integrity can be maintained.

The problem of integrity is the problem of ensuring that the data in the database is accurate. Inconsistency between two entries representing the same "fact" is an example of lack of integrity (which of course can occur only if redundancy exists in the stored data). Even if redundancy is eliminated, however, the database may still contain incorrect data. Centralized control of database helps in avoiding these situations,insofar as they can be avoided, by permitting the DBA to define validation

procedures to be carried out whenever any update operation is attempted.

Most of the advantages listed above are fairly obvious. however one other point, which is not so obvious must be added to the list, namely, the provision of data independence. Strictly speaking, this is an objective rather than an advantage.

AN ARCHITECTURE FOR A DATABASE SYSTEM

We will now give an outline of an architecture for a database system. We do not claim that every database system can be neatly matched to this particular framework, nor do we mean to suggest that this particular architecture provides the only possible framework. However, the architecture does seem to fit a large number of systems reasonably well; moreover, it is in broad agreement with that proposed by the ANSI/SPARC Study Group on Data Base Management Systems.

The architecture is divided into three general levels : internal, conceptual and external (Fig. 1.1). Broadly speaking, the internal level is the one closest to the physical storage, that is, the one concerned with the way in which the data is actually stored; the external level is the one closest to the users,that is,the one concerned with the way in which the data is viewed by individual users; and the conceptual level is a "level of

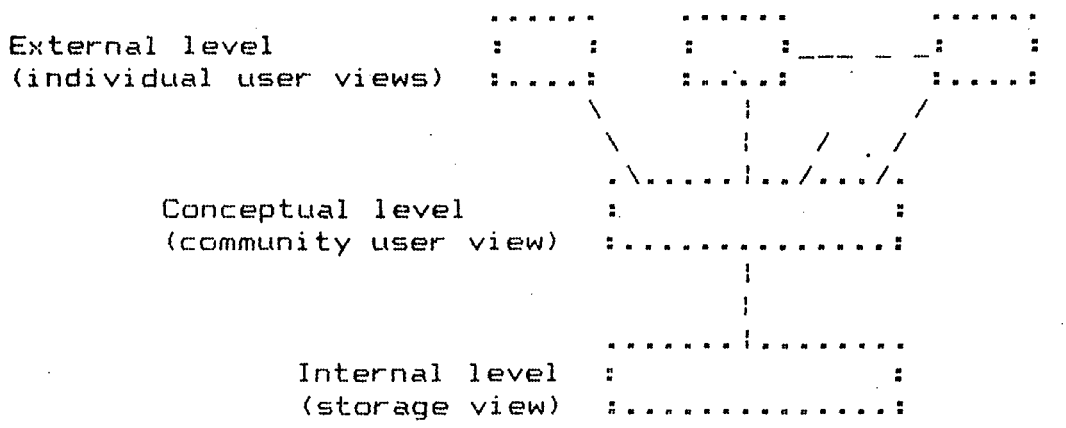

Fig. 1.1 The three levels of the architecture.

indirection" between the other two. If the external level is concerned with individual user views, the conceptual level may be thought of as defining a community user view. In other words, there will be many "external views" each consisting of a more or less abstract representation of some portion of the database, and there will be a single "conceptual view", consisting of a similarly abstract representation of the database in its entirety. Most of the users will not be interested in total database, but only in some restricted portion of it. Likewise, there will be single "internal view", representing the total database as actually stored.

We will, now, examine the components of the architecture in somewhat more detail (Fig. 1.2).

The users are either application programmers or on-line terminal users of any degree of sophistication. (The DBA is an important special case.) Each user has a language

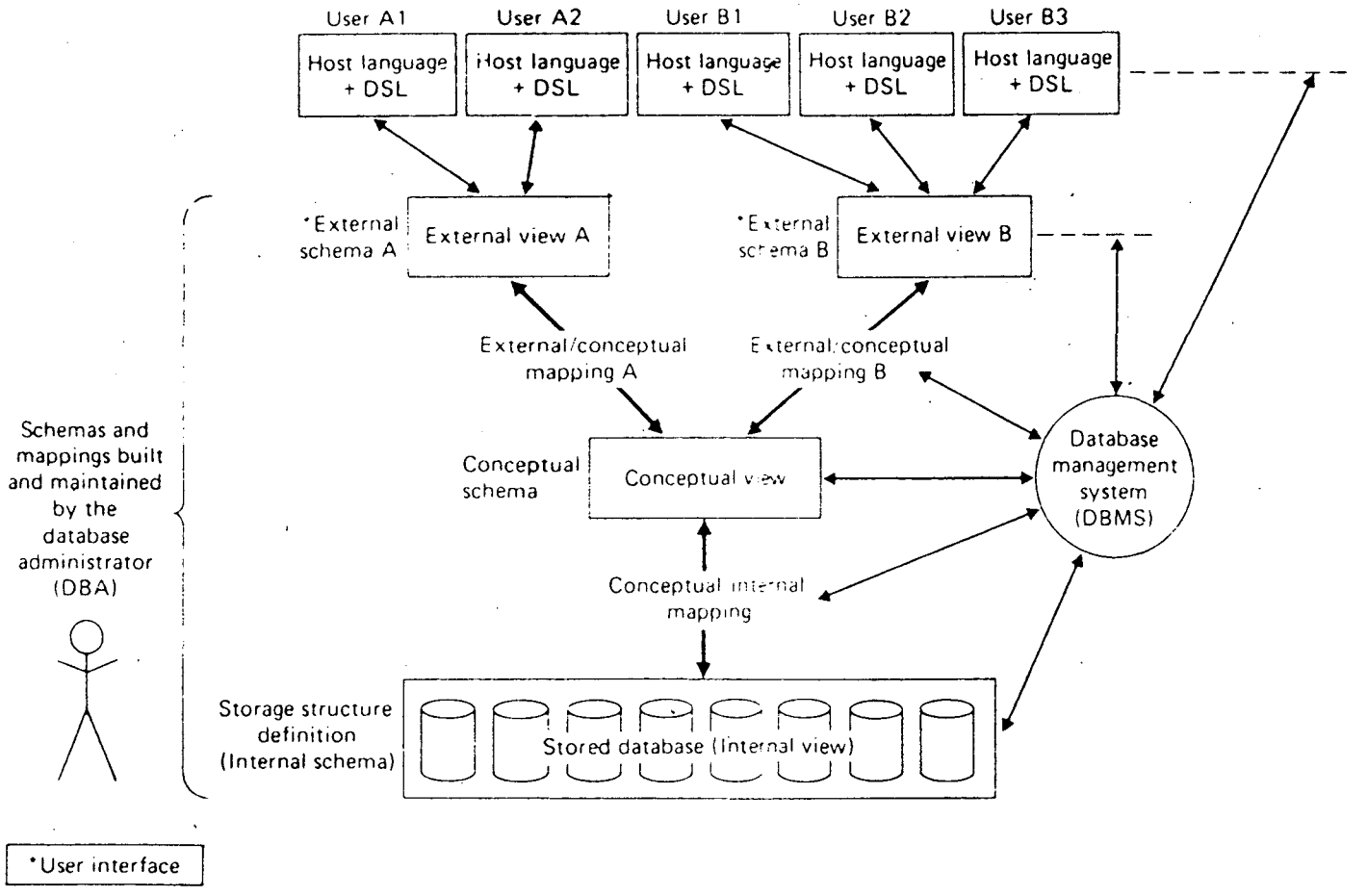

Fig. 1.2 Database system architecture.

at his or her disposal. For the application programmer it will be a conventional programming language, such as COBOL or PL/1; for the terminal user it will be either a query language or a special-purpose language tailored to that user's requirements and supported by an on-line application program. For our purposes the important thing about the user's language is that it will include a data sublanguage(DSL), that is, a subset of the total language that is concerned with database objects and operations. We talk about the data sublanguage as being embedded in a host language. A given system may support multiple host languages and multiple data sublanguages.

In practice, any given data sublanguage is really a combination of two languages: a data definition language (DDL), which provides for the definition or description of database objects (as they are perceived by the user), and a data manipulation language (DML), which supports the manipulation or processing of such objects.

We must note that the data sublanguage and the host language (such as COBOL or PL/1) are fairly "tightly coupled" - that is, to the user the two are not really separable. In current practice this is usually not the case, at least so far as programming languages are concerned. Instead (a) the definitions are completely outside the application program, and written in a DDL that does not even faintly resemble the user's host language, and (b) the manipulation is done by CALLing

standard subroutines (provided as part of the DBMS), and is therefore again outside the host language framework. In other words, in most systems today the data sublanguage and the host are very loosely coupled. A tightly coupled system provides a more uniform set of facilities for the user, but obviously involves more effort on the part of the designers and developers of the system.

To return to the architecture: We have already indicated that an individual user will generally be interested only in some portion of the total database; moreover, the user's view of that portion will generally be somewhat abstract when compared with the way the data is physically stored. In ANSI/SPARC terms an individual user's view is called an external view. An external view is thus the content of the database as it is seen by some particular user (that is, to that user the external view is the database). In general, then, an external view consists of multiple occurrences of multiple types of external record. An external record is not necessarily the same as a stored record. The user's data sublanguage is defined in terms of external records; for example, a DML "get" operation will retrieve an external record occurrence, rather than a stored record occurrence. The term "logical record" is also, sometimes, used to refer to an external record.

Each external view is defined by means of an external

schema, which consists basically of definitions of each of the various types of external record in that external view. (The external schema is written using the DDL portion of the data sublanguage. That DDL is therefore sometimes called an external DDL.) In addition there must be a definition of the mapping between the external schema and the underlying conceptual schema.

We turn now to the conceptual level. The conceptual view is a representation of the entire information content of the database, again in a form that is somewhat abstract in comparison with the way in which the data is physically stored. (It may also be quite different from the way in which the data is viewed by any particular user. Broadly speaking, it is intended to be a view of the data "as it really is" rather than as users are forced to see it by the constraints of [for example] the particular language or hardware they are using.) The conceptual view consists of multiple occurrences of multiple types of conceptual record. A conceptual record is not necessarily the same as either an external record, on the one hand, or a stored record, on the other. The conceptual view is defined by means of the conceptual schema, which includes definitions of each of the various types of conceptual record. (The conceptual schema is written using another data definition language - the conceptual DDL.) If data independence is to be achieved, these definitions must not involve any considerations of

storage structure or access strategy - they must be definitions of information content only. Thus,there must be no reference to stored field representations, physical sequence, indexing, hash-addressing, or any other storage/access details. If the conceptual schema is made truly data-independent in this way, the external schemas, which are defined in terms of the conceptual schema will be data-independent too. In most existing systems the conceptual view is really little more than a simple union of all individual users' views, possibly with the addition of some simple authorization and validation procedures.

The third level of architecture is the internal level. The internal view is a very low-level representation of the entire database; it consists of multiple occurrences of multiple types of internal record. "Internal record" is the ANSI/SPARC term for the construct that we have been calling a stored record. It does not deal in terms of physical records or blocks, nor with any device-specific constraints such as cylinder or track sizes. The internal view is described by means of the internal schema, which not only defines the various types of stored record but also specifies what indexes exist, how stored fields are represented, what physical sequence the stored records are in, and so on. The internal schema is written using yet another data definition language - the internal DDL.

Referring again to Fig. 1.2, we observe two levels of mapping, one between the external and conceptual levels of the system and the one between the conceptual and internal levels. The conceptual/internal mapping defines the correspondence between the conceptual view and the stored database; it specifies how conceptual records and fields map into their stored counterparts. If the structure of the stored database is changed - i.e., if a change is made to the storage structure definition - the conceptual/internal mapping must be changed accordingly, so that the conceptual schema may remain invariant (it is the responsibility of the DBA to control such changes.) In other words, the effect of such changes must be contained below the conceptual level, so that data independence can be achieved.

An external/conceptual mapping defines the correspondence between a particular external view and the conceptual view. In general, the same sort of differences may exist between these two levels as may exist between the conceptual view and the stored database. For example, fields may have different data types , records may be differently sequenced, and so on. Any number of external views may exist at the same time; any number of users may share a given external view; different external views may overlap. We can express the definition of one external view in terms of others, rather than always requiring an explicit definition of the mapping to the conceptual

level. This feature can be permitted, particularly, if several external views are closely related to one another.

We shall now discuss other components of the database system architecture (Fig. 1.2) - the database management system, the database administrator, and the user interface.

The database management system (DBMS) is the software that handles all access to the database. Conceptually what happens is the following : (1) A user issues an access request, using some particular data manipulation language; (2) the DBMS intercepts the request and interprets it; (3) the DBMS inspects, in turn, the external schema, the external/conceptual mapping, the conceptual schema, the conceptual/internal mapping, and the storage structure definition; and (4) the DBMS performs the necessary operations in the stored database.

The database administrator (DBA) is the person (or a group of persons) responsible for overall control of the database system. The DBA's responsibilities include : deciding the information content of the database, deciding the storage structure and access strategy, defining the authorization checks and validation procedures, defining a strategy for backup and recovery.

One of the most important DBA tools is the data dictionary (not shown in Fig. 1.2) . The data dictionary is effectively a database in its own right - a database that contains "data about data" (that is, descriptions of other objects in the system, rather than simply "raw data"). In particular, all the various schemas (external, conceptual, internal) are physically stored, in both source and object form, in the dictionary.

The last component of the architecture is the user interface. This may be defined as a boundary in the system below which everything is invisible to the user.

DATABASE MODELS WITH RELATIVE ADVANTAGES

A database model is a way of describing database structures and database processing that is general enough to encompass all or at least a large majority of data base applications. In general a database model consists of two elements - (1) A mathematical notation for expressing data and relationships; and (2) Operations on the data that serve to express queries and other manipulations of the data.

Efforts to develop database models have been underway since the late 1960's. At present, there are three database models of importance. These models are hierarchical, network, and relational. We will discuss these models in brief.

Database models are broken into two parts. One part, referred to as the data definition language (DDL), describes the structure of the database. The other part, referred to as the data manipulation language (DML), describes the way database is manipulated (processed).

The data definition language (DDL) describes the name and type (numeric, string etc.) of each field, as well as the way the fields are grouped into records. Also the DDL must indicate the primary and secondary keys (if any). The DDL preserves the independence of logical and physical representations of data. Given the DDL, programs need not be dependent on or locked into particular physical representations of the data. The physical structure of data can be changed without modifying any part of the DDL description. Since users, in general, process only some portions of the database depending upon their requirements, this implies that the DDL must be able to describe portions of the database. In addition to representing the structure of the entire database and any particular portion of that structure viewed by a particular user, DDL must specify security restrictions on the database. It must indicate the fields or records that are restricted and the type of each restriction (read, write, read/write).

The data manipulation language (DML) describes the techniques used to process the database. It tells how the records can be retrieved, replaced, inserted and deleted.

This includes processing records directly using keys or indirectly via relationships between database records. The DML should enable the user to deal with the database in logical or symbolic terms. Keys, for example, should be symbolic identifiers rather than physical addresses. This preserves the independence of the programs from the physical representation of the database. Further, the DML should free the user from database structure maintenance . For example, take the case of secondary key maintenance - a big overhead. When a record is added to a set (a group of database records having a common value in a secondary key field), the database system should automatically cause the appropriate tables or links to be modified. The user should not be required to do this; in fact, it should be transparent to the user. This also helps to preserve the independence of programs from physical structure. Further, to permit a wide range of programs to use the database, the DML should support as many languages as possible. This means that the DML must not be structured around any particular programming language. It should be possible to implement the DML in any language that is potentially useful for database processing.

Typically, a DML consists of verbs and operands that provide way to retrieve, replace, insert and delete records. Some generalized DML verbs are shown below.

verb	operands
READ	Record name, key name, key value, field names, passwards.
REPLACE	Record name, key name, key value, field names, field values, passwards.
INSERT	Record name, field names, field values, passwards.
DELETE	Record name, key name, key value, passwards.

The READ verb requires the name of the record type plus the identifying data such as key name and key value. The DML probably has a provision for sequentially reading records in a file or in a set as well. The field names operand is a list of fields to be read, and password is a list of passwards for security. These operands have the same meaning for other DML verbs. In addition, the field values operand is needed to provide data to the database system for the REPLACE and INSERT verbs.

We will now give a brief discription of the three database models and compare them.

Hierarchical model requires the data to be reresented by hierarchical (tree) constructs. Simple and complex network structures can not be directly represented by hierarchical constructs but can be decomposed into tree structures , say, using logical pointers. This means that the user's view of the data, whether he/she sees a tree or a simple or complex network, must somehow be forced

into the tree representation.

Network model requires use of the set concept for representing the data. A set is a collection of occurences of records of a particular type or types.A set has an owner, which is an occurence of a record of different type. Every record occurence is elligible for to be a member of a set or an owner of a set , but no record can be both a member and an owner of the same set. Set is the key concept for holding relationships in the network model. Tree and simple network structuers can be represented by sets in a straight forward manner. However, complex networks cannot be directly represented, but can be transformed or reduced into simple networks by defining a new record type and letting records of this type hold data about intersection of two records.

The relatinal database model differs in several aspects from the hierarchical and network models. For one, the relational model is based on a foundation of theory from relational mathematics.Hierarchical and network models (say DL/1 and DBTG) are directed at programming systems; the step from either of these to a programming language is a short one. The relational model consists of a group of cocepts that are not particularly related to any programming language. Finally, the relational model tends to represent data as it exists, that is in tabular form. The relational model does not force the use of an artificial construct (like tree or set); rather, it

reduces data relationships to simpler components and represnt the components directly. The relational model can be used to represent trees and both simple and complex networks. Both the hierarchical and network models require artificial constructs to represent networks. The relational model does not; it represents data as it exists.The hierarchical and network models tend to add complexity as they force the user to formalise his/her view of the data; the relational model tends to simplify.

To evaluate the three models, further, as discussed above, we must state the criteria by which they shold be judged. We see two primary concerns.

1. Ease of Use.

Especially in small databases, the principal cost may be the time spent by the programmer writing application programs and by the user posing queries. We want a model that makes accurate programming and the phrasing of queries easy.

2. Efficiency of Implementation.

When databases are large, the cost of storage space and computer time dominate the total cost of implementing a database. We need a database model in which it is easy for the DBMS to translate a specification of the

conceptual scheme and the conceptual-to-physical mapping into an implementation that is space efficient and in which queries can be answered efficiently.

By the criterion of easy use, there is no doubt that the relational model is superior. It provides only one concept, the relation, that the programmer or user must understand. Moreover, the relational algebra and calculus clearly provide a notation that is quite succinct and powerful, and this power carries over naturally to real relational query languages. These languages make systems based on the relational model available to persons whose programming skill is not great. Compare, for example, the effort needed to specify the join of relations with the work required to write a program in the DBTG data manipulation language or the DL/1.

Further, the network model requires our understanding of both record types and links,and their interrelationships. The implementation of many-many relationships and relationships on three or more entity sets is not straightforward. Although this problem can be overcome by introducing dummy record types, which is quite typical. Similarly, the hierarchical model requires understanding the use of pointers (virtual record types) and has the same problem as the network model regarding the representation of relationships that are more complex than many-one relationships between two entity sets.

When we consider the potential for efficient implementation, the network and hierarchical models score high marks. Certainly,the implementations of variable length records facilitate the task of following links. Also, data structures such as the multilist and the pointer-based implementation of variable length records do not generalize readily to many-many mappings. Since relations can, and often do, represent many-many mappings, we see that efficient implementation can be more difficult for relations than for networks or hierarchies.

Certainly, there is no fundamental reason why all these implementation ideas for networks and hierarchies cannot be carried over to the implementation of relations, and indeed, many of them have been.

RELATIONAL DATABASE TERMINOLOGY.

The relational model of databases was first introduced by Codd (1970). Informally, in the relational model, data is regarded as stored in tables (called relations). We will give the formal definitions.

A relational scheme consists of a (finite) set of attributes.

With each attribute, we associate a domain of values.

Let R be a relation scheme. A tuple on R is a function

(say, f) mapping each attribute in R to a value in its domain.

A relation on the relational scheme R is a set of tuples on R.

Note that a relation does not have duplicate tuples, since we defined it as a set of tuples.

A relation can be visualized as a table with one column for each attribute and one row for each tuple.

Further, we can note, first, that all the entries in a table are single-valued, that is atomic; neither repeating groups nor arrays are allowed. Second, the entries in any column are of the same type. Each column has a unique name, and their order is immaterial. Also,the order of rows (tuples) is insignificant. If a relation has n columns, then each row is referred to as an n-tuple. Also, a relation that has n attributes (columns) is said to be of degree n. Similarly, the number of tuples (rows) in a relation is referred to as its cardinality. Each attribute has a domain, which is the set of values that the attribute can have. Each attibute is given a unique identifier called an attribute name.

An attribute or combination of attributes that uniquely identifies a tuple is referred to as a candidate key. One of the candidate keys is selected to be used as the

unique identifier and is referred to as the primary key.

To sum up, we can say that, in traditional terms a relation resembles a file, a tuple a record (occurance, not type), and an attribute a field (type, not occurance). These correspondences are at best approximate, however.

CHAPTER

2

PRIMS DESIGN AND IMPLEMENTATION

* WHY PRIMS ?
* INTRODUCTION TO PRIMS
* PRIMS ARCHITECTURE
* REPRESENTATIONAL STRUCTURE OF THE DATA
* IMPLEMENTATION DETAILS

WHY PRIMS ?

There are two important reasons for having chosen this problem of developing PRIMS : " design and implementation of a Relational Information Management System in turbo-Pascal ". The first one is as an exercise - an exercise that helps in understanding the subject, that is database, in more depth so far as its design and implementation are concerned. It shows how the data in a database is represented and manipulated at lower levels which usually is not visible to the ordinary users. Secondly, besides being a fruitful exercise, PRIMS has been designed from commercial point of view as well. PRIMS, in its complete form, is supposed to be compact and portable product essentially meant for micro computers.

INTRODUCTION TO PRIMS

PRIMS is relational because it meets all those conditions and requirements which are essential for a relational system. The requirements for a system to be called relational (as per Schmidt and Broide) are :

1. All information in the database is represented as values in tables.

2. There are no user visible navigation links between these tables.

3. The system supports at least the select, project and equijoin operators of the relational algebra.

PRIMS can be used by on-line user in one of the two interactive modes, namely - the menu driven mode and the query language.

There are two abstract query languages called relational algebra and relational calculus. The real query languages are based on these two abstractions. Currently, due to limitations of time and resources, only menu mode for PRIMS has been developed. In order to complete this exercise within the prescribed time limit, only those DDL and DML commands have been chosen which fulfil the needs for a system to be at least minimally relational. Menu mode has been chosen to make the system user friendly. While interacting through menu mode, the user need not bother to remember the syntax of any DDL and DML commands. What the user simply has to do is to give one of the choice numbers, which he/she picks up from the menu that will be displayed on the video screen. On doing so, the syntax or format in which the data is needed for that database operation will be displayed on the screen. So the user can give the data in the format and obtain the corresponding results. In case there is some error in the input data, the corresponding error message will be displayed which can help the user in rectifying his/her errors. Once that operation is over the system returns control back to the menu mode, displaying the menu and

asking for the fresh choice number. Some choice numbers in the menu may result in subsequent new menus. The procedure to be followed in those new menus is also similar. Once all the required operations on the database are over, the user can exit from the database mode into the operating system mode, again by giving a proper choice number meant for exit.

PRIMS ARCHITECTURE

The architecture for PRIMS ,as shown in figure 2.1, has been represented in five levels. The first level known as

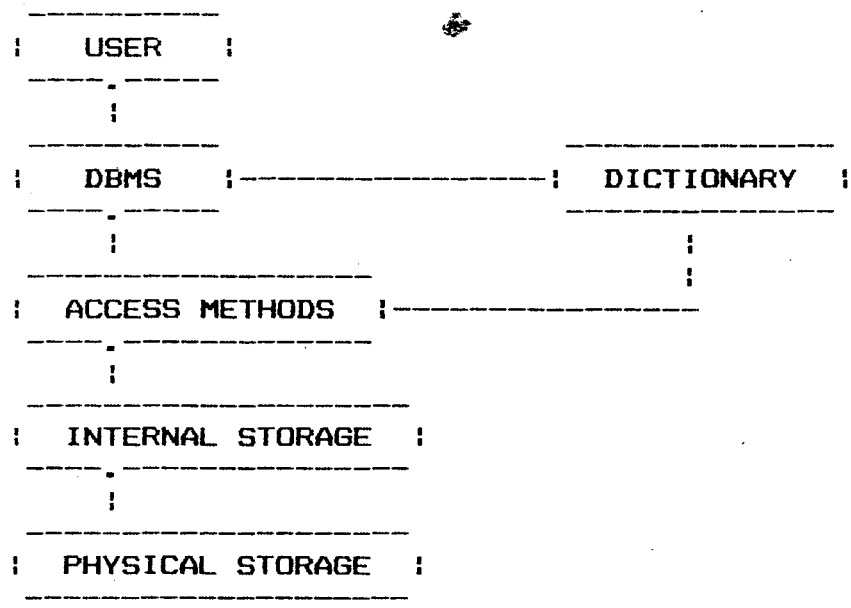

Fig. 2.1 - PRIMS Architecture.

user is either an application programmer or an on-line terminal user. The user has a language at his/her disposal. For the conventional application programmer it

will be a conventional programming language such as turbo-pascal or COBOL. For the terminal user the language is simply a query language (in menu mode only at present). User's language will include a DSL, i.e., a subset of the total language that is concerned with database objects and operations. We talk about DSL as being embedded in a host language. The DSL is really a combination of DDL which consists of declarative constructs and DML which consists of those executable statements that transfer information to and from the database.

The second level of the architecture known as DBMS is the software that handles all access to the database. When a user issues an access request, using DSL, the DBMS intercepts the request and interprets it. Incase the request is error free, the DBMS performs the necessary operations on the internal storage via access methods which constitute the third level of the architecture.

The access modules are the modules which are used for accessing the data at the internal storage level. Access methods have information about the relation scheme (that is, about the structure of a relation, viz., its name, what are various attributes that constitute the relation, their names, types etc.). All this information is not known to DBMS but is made available to it through access methods only.

The fourth level of architecture, the internal storage,

is the level where data in the database is represented and is not yet the lowest level of database which is known as the physical storage level. The physical storage level deals with all device dependent details. The lowest level is handled by the operating system modules which are responsible for conversion of data from internal storage to physical storage. So, it is the operating system only which finally is aware of access and storage of the data as stored on the physical medium like a disk.

REPRESENTATIONAL STRUCTURE OF THE DATA

As seen in fig. 2.1, both DBMS and access methods are interfaced with dictionary which provides their corresponding modules with the information about the corresponding relation under database operation.

In the program the dictionary is given a fixed file name, namely RELATION.DIR. Whenever the user wants to create a new relation, the corresponding relation name is put in the dictionary and the information about the structure of the relation (how many attributes this relation has, what are the names of these attributes) is put into the REL file which is a file created, named after the name of the relation (for example, if the user wants to create the relation named STUDENT, STUDENT will be inserted into RELATION.DIR file and immediately a file named STUDENT.REL will be created which will contain the

information about this relation.).

As soon as the user talks about a relation, first of all, the relation name is checked for its validity (i.e., whether or not the relation name is a valid one). It is done so since after this relation name we have to create REL file as already described and DAT file which will contain data for this relation.

Since all the relations in a relational database are supposed to be distinct, so before adding a new relation name in DIR file ,this file is scanned for whether or not this relation name is already present in it. If so, a suitable message is displayed and the user is asked to give the relation name again.

The DAT file will contain the data corresponding to the relation the user wants to create. To facilitate the user, in addition to the data corresponding to the actual list of attributes supposed to be present in DAT file, we add two more attributes - 1. serial number (s_no) of the tuple (record) ; and 2. record status (rec_sts) of the record (p for present and a for absent).

IMPLEMENTATION DETAILS

In this section, we will describe various access methods, a set of modules which are used for accessing the data from the internal storage level. We will describe the program structure and details of each procedure (module),

their purpose, the input to and output from each procedure and the way they are called together to accomplish a fixed task.

At present, PRIMS can accomadate twenty-five relations each containing a maximum of twenty-five attributes.

THE Main Program

The main program starts by invoking the procedure initialize. Further it issues a call to the procedure list_options and the procedure interpret_option. These procedures are invoked time and again until the user wants to come out of the present session with system, by giving the option '3' in menu B.

We will now describe each procedure in detail.

check_if_valid_string

This procedure is called to check whether the response given by the user at a particular time of execution is a valid string or not. This is needed since in Turbo Pascal any valid file-name can start only with a letter (a - z) or (A - Z) and followed by one or more digits (0 - 9). And since when the system asks to give the relation name, and after this name a corresponding file is to be created, it becomes necessary to check that the user's response should be such that we can create a file after

this name. Similarly, when specifying the names of various attributes in a relation we want that it should be meaningful attribute. This procedure accepts a string and returns whether it is valid or not.

check_if_integer

This procedure accepts a string as input and returns whether or not the given string is a positive integer. The procedure is called when a user is supposed to give a response and we expect it to be a positive integer.

initialize

Here the dictionary is assigned a fixed name RELATION.DIR and some other running variables are given initial values. RELATION.DIR stores the names of various relations which we will be dealing with.

list_options

This procedure lists out the various options available for the user. It displays Menu A with three options :- 1. To create a relation scheme; 2. To access data, and 3. Stop.

interpret_option

This procedure asks for the option and then interprets. If the option is one of those displayed in the menu

A (consisting of three options - 1. To create a relation scheme ; 2. To access data ;and 3. Stop), the control is sent to the appropriate procedure otherwise the procedure error_handler is invoked displaying the suitable error message and user is asked to retype the option.

rel_info

This procedure asks about the relation name and checks for the validity of the relation name by calling check_if_valid_string. If the relation name is valid (i.e., it does not contain any special characters including blank and starts with any of the characters a - z or A - Z) the control is returned back to the calling procedure.

rel_fmt

This procedure asks the information about that how many attributes the relation under consideration has. The response to this question is supposed to be a positive integer. So the user response is checked for validity by calling the procedure check_if_integer. If the response is not a positive integer the user is directed to retype the response. After this the attribute names constituting the relation are asked as many times as there are number of attributes. These attribute names are checked for validity to give some meaning. Lastly this information is added to the REL file created after the name of the

relation currently under consideration.

show_relation_directory

This procedure simply lists out all the relations currently present in the DIR file, if any.

check_if_already_exists_in_the_relation_directory

This procedure accepts a relation name, scans through the DIR file and gives the message 'already exists' if the relation is there in the DIR file. This procedure is written to avoid duplicate relation names as all the relations in a relational database are supposed to be distinct.

append_rel_dir

Once a relation name is valid and not a duplicate one that relation name is added to the DIR file by this procedure.

relation_and_existence

This procedure is invoked when we want to access data about a particular relation. Firstly, the relation name is checked for its validity. Once it is so, then DIR file is searched for this relation since once we want to access (insert, delete, update, retreive) data about a particular relation it must be present in DIR file. If

the relation is not there in DIR file, the message 'does not exist' is displayed and control is returned back to the calling procedure. If the relation is there in the DIR file a DAT file is assigned after the name of the relation name.

The DAT file contains the actual data about a particular relation. To facilitate the user, two extra attributes are added to the list of attributes actually supposed to constitute the relation. The first is serial number (s_no) of the record and the second is the record status (rec_sts) of a particular tuple (record) which at any time helps to determine whether the record is present in the relation or not.

create_a_relation_scheme

When the user is interested in creating a new relation this procedure is invoked. The system asks for the relation name checks for its validity (if it is not a valid relation name the user is directed to retype it). Once it is a valid relation name the system checks if the RELATION.DIR contains some relations (it is achieved by using the built-in Turbo Pascal function 'filesize', in terms of records). If the DIR file is empty (i.e., the 'filesize' returns a zero value), the procedure rel_info is called which supplies the valid relation name and then it is added to the list of relations in DIR file. If the DIR file is not empty (i.e., the 'filesize'

returns a value greater than zero, certainly it cannot be negative), then all the relation names in DIR file are displayed by invoking the procedure show_relation_directory. This helps the user to know about all the existing relations created so far in the DIR file. After this, the procedure rel_info is called to know about the relation name. If this relation exists already in DIR file (checked by procedure check_if_already_exists_in_rel_dir), the message that this relation already exists in DIR file is displayed and the user is supposed to give the relation name again. If not, then this relation name is added to the DIR file by calling the procedure append_rel_dir. Further a REL file is created after the name of the relation currently under consideration. The information about the number of attributes and the names of the attributes consttituting that relation is put into the REL file (by calling the procedure rel_fmt).

insert_a_tuple

This procedure begins by calling the procedure relation_and_existence which asks the relation name, checks for its validity, scans the DIR file to see that the relation is very much there in this file, then assigns the corresponding DAT file after the name of the relation to the file variable, and returns the control to the procedure insert_a_tuple again. If the

DAT file is empty it is written afresh, otherwise tuples will be appended at the end of the DAT file. Consulting the REL file all the attribute names supposed to be in that relation, which the user has already specified, are displayed one at a time, and the user gives the actual data for that attribute. This tuple is appended into the DAT file. The user is given the option that if he wants to insert some more tuples in the same relation, and if he/she intends to do so, he/she can specify it and the process is repeated again until he/she gives the response in negative. Further, the user is given the option to insert any more tuples in any other relation. If the answer is in affirmative the tuples can be inserted into the desired relation. Lastly the control is given back to the calling procedure.

Note that whenever we insert a tuple in a relation, the record status(rec_sts) in that tuple is set to 'p' to indicate that the tuple is present in that relation.

delete_a_tuple

This procedure is called when a user wants to delete one or more tuples in any relation by giving the option meant for this operation in menu B. The procedure works exactly similar to the procedure insert_a_tuple except that the record status(rec_sts) is set to 'a' to indicate that the record is absent in that relation and in further processing this tuple is taken care of. The procedure

also, takes care of the situation if the record was already deleted. In this case, the message is displayed and he/she is to type a fresh record_number.

The record_number is treated as the key for any tuple in a relation. On the basis of this key we process (delete, update or retreive) a tuple of a desired relation. Further, note we cannot delete a tuple from a relation if no data is inserted into a relation. This procedure also takes care of this. The key is checked for its validity that is the record_number should be a positive integer and it should not exceed the maximum number of tuples present in the DAT file corresponding to a particular relation.

update_a_tuple

The procedure calls the procedure relation_and_existence checking the validity and existence of the relation in DIR file. Once it is so it checks if the corresponding DAT file contains some records. If the DAT file is not empty only then we can update a tuple of that relation otherwise, if the file is empty appropriate message is given and we cannot update any tuple. Then the key i.e., record_number is asked and is checked for its validity as in case of delete_a_tuple procedure. After this, the REL file and DAT file for this relation is referred. The old value of the attribute alongwith its name is

displayed and the system asks for the new value of that attribute. If the user gives the return key, the previous old value for that attribute stands as such. If the user intends to change the data for an attribute he will have to give the new value for that attribute. The tuple is overwritten into the DAT file under consideration at its previous place. Further, the user can update as many records(tuples) as he/she wants into the same or any other relation by giving appropriate answer. The procedure also takes care of the situation if the record was already deleted. In this case, the message is displayed and he/she is to type a fresh record_number.

retreive_a_tuple

This procedure displays all tuples in a particular relation. If the corresponding DAT file is empty, the message 'the relation contains no records' is displayed. Further, the user can retreive tuples of any other relation if he/she intends so.

error_handler

If the user response is not error free, this procedure is called which gives an appropriate message and the user is supposed to retype the response.

access_data

This procedure is invoked when the user gives the option

'2' meant for this operation. Note that, if the DIR file is empty,i.e.,we have not created any relation so far, we cannot access data. Otherwise, this procedure lists out four options under the heading menu B. The user is asked about his/her option and the control is sent to the corresponding procedure meant for that operation.

Two menus are displayed during the course of execution of the program. First is menu A comprising of three options,viz. - 1. create a relation scheme; 2. access data; and 3. stop. If the user gives option '2', this inturn displays menu B giving four options, viz. - 1. insert a tuple; 2. delete a tuple; 3. update a tuple and 4. retreive a tuple.

The menus A and B are interleaving each other. After doing the desired operation in menu B, the control, again, is returned back to menu A and asking for fresh choice. The program runs until the user gives the option '3' in menu A to stop execution.

CHAPTER

3

FURTHER ENHANCEMENTS

* AT DESIGN LEVEL

* AT IMPLEMENTATION LEVEL

In the previous chapter, we have given one possible solution to achieve the results. In this chapter, we will mention some of the possible improvements in PRIMS and further scope in the work.

There could be two possible improvements in this work - one, those at the design level; and second, those at the implementation level. We will descibe them separately.

At Design Level

We have decided to put relation names in the file RELATION.DIR; and the information about that how many attributes are there constituting a particular relation and their names is put into the REL file corresponding to the relation name. These two pieces of information could have been located in one file only namely the dictionary, the DIR file. (In fact, we have tried to achieve the results in a simplest possible way.) This way consulting the dictionary and getting information about a relation could have been much easier; and certainly it would have decreased the unnessary overhead of opening up and closing down the files. Further, it would have lead to efficiency in terms of storage space and access time also.

Further, we have not included the information about a relation, like what is the type of each attribute (viz. - real, integer, string or boolean); the range of values

each attribute can take; and the key for a relation (at present, we have treated sequence or tuple number to be as a key). This information could have been included (in fact, PRIMS suggests a simplest possible solution avoiding many complexities involved).

At the Implementation Level

One such possible improvement is when we delete a tuple in any relation. Infact, the tuple is not deleted physically. Rather, it is just a logical deletion meaning by that we add up a tag field in each tuple and setting it to 'a' indicating that the tuple is no more there in the relation (of course, we take care of this fact in futher processing). This is desirable if we are dealing with a small database with not too many deleted entries. On the contrary, if we have a large database in which the relations contain too many such deleted tuples, there is a wastage of storage space which in turn leads to slower accessing of the database. To overcome this, one way is to reorganize the data file named after the relation name - reading this file and rewriting it skipping those tuples (records) in which tag field (rec_sts) has value 'a'. One other possible solution would have been to use pointer to the next tuple (record) present in the file. In case the user wants to delete a record, we will have to change just one pointer pointing to the next present record thus skipping the desired tuple, and releasing the

space occupied by that tuple, thereby increasing the efficiency.

At present, DDL (description of the data) portion for the system has been developed successfully. Due to the time limit, DML part (i.,e. how to process or manipulate the data) involving operatios like selection, join, divide, etc., is not touched at all. For that, a suitable real query language (like SQL) based on abstract query languages (like relational algebra or relational calculus) is to be developed.

REFERENCES

1. An Introduction to Database Systems, Vol. 1, Third Edition - C. J. Date.

2. Data Base : Structured Techniques for Design, Performance, and Management - S. Atre.

3. Principles of Database Systems
- Jeffrey D. Ullman.

4. Database Processing : Fundamentals, Design, Implementation, Second Edition - David M. Kroenke.

5. Introduction to Pascal, Second Edition
- Jim Welsh, John Elder.

6. Data Structures Using Pascal, Second Edition
- Aaron M. Tenenbaum, Moshe J. Augenstein.

* SOURCE CODE LISTING

```
program prims(input,output);

(*  This is PRIMS source code listing,
    written in turbo-Pascal language;
    and implemented on DCM TANDY-1000.  *)

(*  Author - PARAMJIT SINGH  *)

type

   response = string[25];

   rel_rec = record

             rel_name : string[25];

             end;

   rel_file_rec = record

                  no_of_attribute :integer;

                  attr_name : array[1..25] of response;

                  end;

   rel_data_rec = record

                  s_no : integer;

                  rec_sts : char;

                  attr_name : array[1..25] of response;

                  end;

var

   sub_response,respons,rsponse : string[25];

   v_s_no:integer;

   v_att_name : string[25];

   var_rel_name,o_var_rel_name : string[25];

   var_rel_rec : rel_rec;

   rel_dir : file of rel_rec;

   var_rel_file_rec : rel_file_rec;

   rel_file_name : array[1..25] of response;

   rela_file : file of rel_file_rec;
```

```
    var_rel_data_rec : rel_data_rec;
    rel_data_file_name : array[1..25] of response;
    rela_data_file : file of rel_data_rec;
    alldone,if_int,if_str : boolean;
    i,j,k,z,t,u,v_num_of_att : integer;
    rltfile : text;

procedure initialize;
begin
      alldone := false;
      assign(rel_dir,'RELATION.DIR');
      assign(rltfile,'rims.rlt');
      rewrite(rltfile);
      i := 0;
      j := 0;
      k := 0;
      z := 0;
      v_s_no := 0;
end;

procedure list_options;
begin
      writeln;
      writeln('MENU A.');
      writeln('--------');
      writeln('  1 : To create a relation scheme.');
      writeln('  2 : To Access Data.');
      writeln('  3 : Stop.');
      writeln;
```

```
      writeln(rltfile);
      writeln(rltfile,'MENU A.');
      writeln(rltfile,'--------');
      writeln(rltfile,'  1 : To create a relation scheme.');
      writeln(rltfile,'  2 : To Access Data.');
      writeln(rltfile,'  3 : Stop.');
      writeln(rltfile);
end;

procedure check_if_valid_string;
label 5,3;
var
   x,y,z,t : integer;
begin
   if_str := false;
   respons := rsponse;
   if ((length(respons) = 0)) then goto 5;
   y := length(respons);
   for x := 1 to length(respons) do
   begin
       if (respons[x] = ' ')
       then y := y - 1
       else
       begin
           z := x;
           goto 3;
       end;
   end;
```

```
    if y = 0 then goto 5;
3 :sub_response := copy(respons,z,length(respons) - z + 1);
    if not (((sub_response[1] >= 'a') and (sub_response[1] <= 'z')) or
        ((sub_response[1] >= 'A') and (sub_response[1] <= 'Z')))
    then goto 5;
    for x := 2 to length(sub_response) do
    begin
    if not (((sub_response[x] >= 'a') and (sub_response[x] <= 'z')) or
        ((sub_response[x] >= 'A') and (sub_response[x] <= 'Z')) or
        ((sub_response[x] >= '0') and (sub_response[x] <= '9')))
    then
    goto 5;
    end;
    if_str := true;
    rsponse := sub_response;
5 : ;
end;

procedure rel_info;
label 1;
begin
1: writeln;
    write('RELATION NAME : ');
    read(var_rel_name);
    writeln(rltfile);
    write(rltfile,'RELATION NAME : ');
    write(rltfile,var_rel_name);
    rsponse := var_rel_name;
```

```
    check_if_valid_string;
    if not if_str then
    begin
    writeln;
    writeln(#7,'not valid relation name.');
    writeln(rltfile);
    writeln(rltfile,'not valid relation name.');
    goto 1;
    end;
    var_rel_name := rsponse;
    var_rel_rec.rel_name := var_rel_name;
    o_var_rel_name := var_rel_name;
end;

procedure check_if_integer;
label 1,5,3;
var
   x,y,z,t : integer;
begin
   if_int := false;
   respons := rsponse;
      if ((length(respons) = 0)) then goto 5;
   y := length(respons);
   for x := 1 to length(respons) do
   begin
       if (respons[x] = ' ')
       then y := y - 1
       else
```

```
        begin
             z := x;
             goto 3;
        end;
    end;
    if y = 0 then goto 5;
3 :sub_response := copy(respons,z,length(respons) - z + 1);
    for x := 1 to length(sub_response) do
    begin
        if (sub_response[x] = '0')
        then
            t := 0
        else
        begin
            t := 1;
            z := x;
            goto 1;
        end;
  end;
  if t = 0 then goto 5;
1 :sub_response := copy(sub_response,z,length(sub_response) - z + 1);
    for x := 1 to length(sub_response) do
    begin
    if ((sub_response[x] < '0') or (sub_response[x] > '9')) then goto 5;
    end;
    if_int := true;
    rsponse := sub_response;
5 : ;
```

```
end;

procedure rel_fmt;
label 1,2;
begin
    u := -1;
2 :writeln;
    write('how many attributes does ',o_var_rel_name,' have? : ');
    read(rsponse);
    writeln;
    writeln(rltfile);
    write(rltfile,'how many attributes does ',o_var_rel_name,' have? : ');
    write(rltfile,rsponse);
    writeln(rltfile);
    check_if_integer;
    if not if_int
    then
       begin
            writeln;
            writeln(#7,'give positive integer.');
            writeln(rltfile);
            writeln(rltfile,'give positive integer.');
            goto 2;
       end;
    val(rsponse,v_num_of_att,u);
    var_rel_file_rec.no_of_attribute := v_num_of_att;
      for j := 1 to v_num_of_att do
        begin
```

```
1 :       writeln;
          writeln('ATTRIBUTE(',J,')');
          write('  NAME : ');
          read(v_att_name);
          writeln(rltfile);
          writeln(rltfile,'ATTRIBUTE(',J,')');
          write(rltfile,'  NAME : ');
          write(rltfile,v_att_name);
          rsponse := v_att_name;
          check_if_valid_string;
          if not if_str then
          begin
          writeln;
          writeln(#7,'not meeningful attribute name.');
          writeln(rltfile);
          writeln(rltfile,'not meeningful attribute name.');
          goto 1;
          end;
          v_att_name := rsponse;
          var_rel_file_rec.attr_name[j] := v_att_name;
          writeln;
          writeln(rltfile);
        end;
          write(rela_file,var_rel_file_rec);
          writeln;
          writeln(rltfile);
end;
```

```
procedure add_up_a_rel;
begin
   rel_info;
   write(rel_dir,var_rel_rec);
   i := i + 1;
   rel_file_name[i] := var_rel_name + '.rel';
   assign(rela_file,rel_file_name[i]);
   rewrite(rela_file);
   rel_fmt;
   close(rela_file);
end;

procedure show_relation_directory;
begin
        writeln('we have following relations in relation directory :');
        writeln(rltfile,'we have following relations in relation directory :');
        reset(rel_dir);
        while not eof(rel_dir) do
           begin
                 read(rel_dir,var_rel_rec);
                 writeln(var_rel_rec.rel_name);
                 writeln(rltfile,var_rel_rec.rel_name);
           end;
        close(rel_dir);
end;

procedure append_rel_dir;
begin
              reset(rel_dir);
```

```
                t := filesize(rel_dir);
                seek(rel_dir,t);
                write(rel_dir,var_rel_rec);
                close(rel_dir);
end;

procedure check_if_already_exists_in_rel_dir;
begin
             reset(rel_dir);
             while not eof(rel_dir) do
                  begin
                       read(rel_dir,var_rel_rec);
                       if var_rel_rec.rel_name = var_rel_name then
                        begin
                          writeln;
                          writeln(#7,var_rel_name,' already exits.');
                          writeln;
                          writeln(rltfile);
                          writeln(rltfile,var_rel_name,' already exits.');
                          writeln(rltfile);
                          rel_info;
                          check_if_already_exists_in_rel_dir;
                        end;
                  end;
             close(rel_dir);
end;

procedure create_a_relation_scheme;
var
```

```
    k : integer;
begin
     if (filesize(rel_dir) = 0) then
        begin
             rewrite(rel_dir);
             add_up_a_rel;
             close(rel_dir);
        end
     else
        begin
          show_relation_directory;
          rel_info;
          check_if_already_exists_in_rel_dir;
          var_rel_rec.rel_name := o_var_rel_name;
          append_rel_dir;
          i := i + 1;
          rel_file_name[i] := var_rel_name + '.rel';
          assign(rela_file,rel_file_name[i]);
          rewrite(rela_file);
          rel_fmt;
          close(rela_file);
        end;
end;

procedure relation_and_existence;
label  1,2,3;
begin
3 :  writeln;
```

```
      write('Relation  Name : ');
      read(var_rel_name);
      writeln;
      writeln(rltfile);
      write(rltfile,'Relation  Name : ');
      write(rltfile,var_rel_name);
      writeln(rltfile);
    rsponse := var_rel_name;
    check_if_valid_string;
    if not if_str then
    begin
    writeln;
    writeln(#7,'not valid relation name.');
    writeln(rltfile);
    writeln(rltfile,'not valid relation name.');
    goto 3;
    end;
    var_rel_name := rsponse;
{  check  if this relation already exits. }
{search rel dir for this rel. }
       reset(rel_dir);
  2 : if not eof(rel_dir) then
            begin
                 read(rel_dir,var_rel_rec);
                 if var_rel_rec.rel_name = var_rel_name then
                   begin
                        close(rel_dir);
```

```
                    goto 1;
                end
            else
                goto 2;
        end
    else
        begin
            writeln(#7,var_rel_name,' does not exist.');
            writeln(rltfile,var_rel_name,' does not exist.');
            goto 3;
        end;
1 : j := j + 1;
    rel_data_file_name[j] := var_rel_name + '.dat';
    assign(rela_data_file,rel_data_file_name[j]);
end;

procedure insert_a_tuple;
label 4,5,6;
var
   more_relations,more_tuples : string[5];
begin
    more_relations := 'y';
repeat
begin
    relation_and_existence;
    if (filesize(rela_data_file) = 0) then
       begin
           v_s_no := 0;
```

```
            rewrite(rela_data_file);
        end
    else
        begin
          reset(rela_data_file);
          k :=filesize(rela_data_file);
          seek(rela_data_file,k);
        end;
      assign(rela_file,var_rel_name + '.rel');
      reset(rela_file);
      read(rela_file,var_rel_file_rec);
      more_tuples := 'y';
repeat
begin
      v_s_no := filesize(rela_data_file) + 1;
      var_rel_data_rec.s_no := v_s_no;
      var_rel_data_rec.rec_sts := 'p';
      with var_rel_file_rec do
           begin
                for z := 1 to no_of_attribute do
                    begin
                      writeln;
                      write(attr_name[z],' : ');
                      read(var_rel_data_rec.attr_name[z]);
                      writeln;
                      writeln(rltfile);
                      write(rltfile,attr_name[z],' : ');
                      write(rltfile,var_rel_data_rec.attr_name[z]);
```

```
                    writeln(rltfile);

                end
        end;

    write(rela_data_file,var_rel_data_rec);

5:  writeln;

    write('you want to insert any more tuples of this relation?...(y/n): ');

    read(more_tuples);

    writeln;

    writeln(rltfile);

    write(rltfile,'you want to insert any ',
                  'more tuples of this relation?...(y/n): ');

    write(rltfile,more_tuples);

    writeln(rltfile);

    if ((more_tuples <> 'y') and (more_tuples <> 'n'))

    then

       begin

         write(#7,'wrong response, type again.');

         write(rltfile,'wrong response, type again.');

         goto 5;

       end;

end

until more_tuples = 'n';

    close(rela_data_file);

    close(rela_file);

6 :  writeln;

    write('you want to insert tuples in anyother relation?....(y/n): ');

    read(more_relations);

    writeln;

    writeln(rltfile);
```

```
        write(rltfile,'you want to insert tuples ',
                      'in anyother relation?....(y/n): ');
        write(rltfile,more_relations);
        writeln(rltfile);
        if ((more_relations <> 'y') and (more_relations <> 'n'))
        then
           begin
             write(#7,'wrong response, type again.');
             write(rltfile,'wrong response, type again.');
             goto 6;
           end;
end
until more_relations = 'n';
        close(rel_dir);
end;

procedure delete_a_tuple;
label 1,5,6,7;
var
   more_relations,more_tuples : string[25];
   rec_no : integer;
begin
     more_relations := 'y';
repeat
begin
        relation_and_existence;
        if (filesize(rela_data_file) = 0) then
         begin
              writeln(#7,'The relation contains no records.');
```

```
            writeln(rltfile,'The relation contains no records.');
            goto 7;
        end;
      assign(rela_file,var_rel_name + '.rel');
      reset(rela_file);
      read(rela_file,var_rel_file_rec);
      more_tuples := 'y';
repeat
begin
1 :       writeln;
          write('which record number? : ');
          writeln(rltfile);
          write(rltfile,'which record number? : ');
   u := -1;
   read(rsponse);
   writeln;
   write(rltfile,rsponse);
   writeln(rltfile);
   check_if_integer;
   if not if_int
   then
      begin
          writeln;
          writeln(#7,'give positive integer.');
          writeln(rltfile);
          writeln(rltfile,'give positive integer.');
          goto 1;
```

```
    end;
      val(response,rec_no,u);
       if rec_no > filesize(rela_data_file) then
            begin
               writeln(#7,'record number exeeds file size.');
               writeln('type again.');
               writeln(rltfile,'record number exeeds file size.');
               writeln(rltfile,'type again.');
               goto 1;
            end;
      reset(rela_data_file);
      seek(rela_data_file,rec_no - 1);
      read(rela_data_file,var_rel_data_rec);
     if var_rel_data_rec.rec_sts = 'a' then
        begin
          writeln(#7,'already deleted.');
          writeln(rltfile,'already deleted.');
          goto 1;
        end;
      var_rel_data_rec.rec_sts := 'a';
      seek(rela_data_file,rec_no - 1);
      write(rela_data_file,var_rel_data_rec);
5:  writeln;
    write('you want to delete any more tuples of this relation?...(y/n): ')
    read(more_tuples);
    writeln;
    writeln(rltfile);
```

```
        write(rltfile,'you want to delete ',
              'any more tuples of this relation?...(y/n): ');

        write(rltfile,more_tuples);

        writeln(rltfile);

        if ((more_tuples <> 'y') and (more_tuples <> 'n'))

        then

           begin

             write(#7,'wrong response, type again.');

             write(rltfile,'wrong response, type again.');

             goto 5;

           end;

end;

until more_tuples = 'n';

        close(rela_data_file);

        close(rela_file);

6   :   writeln;

        write('you want to delete tuples in anyother relation?....(y/n): ');

        read(more_relations);

        writeln;

        writeln(rltfile);

        write(rltfile,'you want to delete tuples ',
                      'in anyother relation?....(y/n): ');

        write(rltfile,more_relations);

        writeln(rltfile);

        if ((more_relations <> 'y') and (more_relations <> 'n'))

        then

           begin

             write(#7,'wrong response, type again.');

             write(rltfile,'wrong response, type again.');
```

```
            goto 6;
          end;
end;
until more_relations = 'n';
       close(rel_dir);
7 :  ;
end;

procedure update_a_tuple;
label 1,5,6,7;
var
   buffer,more_relations,more_tuples : string[25];
   rec_no : integer;
begin
     more_relations := 'y';
repeat
begin
       relation_and_existence;
      if (filesize(rela_data_file) = 0) then
         begin
              writeln('The relation contains no records.');
              writeln(rltfile,'The relation contains no records.');
              goto 7;
         end;
        assign(rela_file,var_rel_name + '.rel');
        reset(rela_file);
        read(rela_file,var_rel_file_rec);
        more_tuples := 'y';
```

```
repeat
begin
1 :write('which record number? : ');
   u := -1;
   read(rsponse);
   writeln;
         write(rltfile,'which record number? : ');
   write(rltfile,rsponse);
   writeln(rltfile);
   check_if_integer;
   if not if_int
   then
      begin
            writeln;
            writeln(#7,'give positive integer.');
            writeln(rltfile);
            writeln(rltfile,'give positive integer.');
            goto 1;
      end;
         val(rsponse,rec_no,u);
          if rec_no > filesize(rela_data_file) then
               begin
                   writeln(#7,'record number exeeds file size.');
                   writeln('type again.');
                   writeln(rltfile,'record number exeeds file size.');
                   writeln(rltfile,'type again.');
                   goto 1;
```

```
        end;
reset(rela_data_file);
seek(rela_data_file,rec_no - 1);
read(rela_data_file,var_rel_data_rec);
if var_rel_data_rec.rec_sts = 'a' then
  begin
    writeln(#7,'record doesnot exist.');
    writeln('type again.');
    writeln(rltfile,'record doesnot exist.');
    writeln(rltfile,'type again.');
    goto 1;
  end;
with var_rel_file_rec do
  begin
      for z := 1 to no_of_attribute do
          begin
            buffer := var_rel_data_rec.attr_name[z];
            writeln;
            writeln(attr_name[z],'.');
            writeln('              old value : ',buffer);
            write('              new value : ');
            read(var_rel_data_rec.attr_name[z]);
            writeln(rltfile,attr_name[z],'.');
            writeln(rltfile,'              old value : ',buffer);
            write(rltfile,'              new value : ');
            write(rltfile,var_rel_data_rec.attr_name[z]);
              if var_rel_data_rec.attr_name[z] = ''
                 then var_rel_data_rec.attr_name[z] := buffer;
```

```
                        writeln;
                        writeln(rltfile);
                      end
              end;
            seek(rela_data_file,rec_no - 1);
            write(rela_data_file,var_rel_data_rec);
5   :   writeln;
        write('you want to update any more ',
                        'tuples of this relation?...(y/n): ');
        read(more_tuples);
        writeln;
    writeln(rltfile);
    write(rltfile,'you want to update any more ',
                              'tuples of this relation?...(y/n): ');
    write(rltfile,more_tuples);
    writeln(rltfile);
      if ((more_tuples <> 'y') and (more_tuples <> 'n'))
      then
        begin
          write(#7,'wrong response, type again.');
          write(rltfile,'wrong response, type again.');
          goto 5;
        end;
end;
until more_tuples = 'n';
      close(rela_data_file);
      close(rela_file);
6   :   writeln;
```

```
        write('you want to update tuples in anyother relation?....(y/n): ');
        read(more_relations);
        writeln;
        writeln(rltfile);
        write(rltfile,'you want to update tuples ',
                                  'in anyother relation?....(y/n): ');
        write(rltfile,more_relations);
        writeln(rltfile);
        if ((more_relations <> 'y') and (more_relations <> 'n'))
        then
           begin
             write(#7,'wrong response, type again.');
             write(rltfile,'wrong response, type again.');
             goto 6;
           end;
end;
until more_relations = 'n';
       close(rel_dir);
7 :    ;
end;

procedure error_handler;
begin
     writeln(#7,'wrong option, type again.');
     writeln(rltfile,'wrong option, type again.');
end;

procedure retrieve_a_tuple;
label 6,7;
```

```
var
   more_relations,more_tuples : string[25];
   rec_no : integer;
begin
     more_relations := 'y';
repeat
begin
     relation_and_existence;
     if (filesize(rela_data_file) = 0) then
        begin
             writeln(#7,'The relation contains no records.');
             writeln(rltfile,'The relation contains no records.');
             goto 7;
        end;
     assign(rela_file,var_rel_name + '.rel');
     reset(rela_file);
     read(rela_file,var_rel_file_rec);
     write('s.no.':5,'rec sts':8);
     write(rltfile,'s.no.':5,'rec sts':8);
     with var_rel_file_rec do
     begin
               for z := 1 to no_of_attribute do
               begin
                     write(attr_name[z]:12);
                     write(rltfile,attr_name[z]:12);
               end;
                     writeln;
                     writeln(rltfile);
```

```
    end;
    close(rela_file);
    reset(rela_data_file);
  while not eof(rela_data_file) do
  begin
  read(rela_data_file,var_rel_data_rec);
  write(var_rel_data_rec.s_no:5,var_rel_data_rec.rec_sts:8);
  write(rltfile,var_rel_data_rec.s_no:5,var_rel_data_rec.rec_sts:8);
  with var_rel_data_rec do
            begin
                for z := 1 to var_rel_file_rec.no_of_attribute do
                begin
                   write(attr_name[z]:12);
                   write(rltfile,attr_name[z]:12);
                end;
                writeln;
                writeln(rltfile);
            end;
  end;
  close(rela_data_file);
6 :  writeln;
     write('you want to retrieve tuples of anyother relation?....(y/n):
     read(more_relations);
     writeln;
   writeln(rltfile);
   write(rltfile,'you want to retrieve tuples of ',
                          'anyother relation?....(y/n): ');
   write(rltfile,more_relations);
```

```
    writeln(rltfile);
      if ((more_relations <> 'y') and (more_relations <> 'n'))
      then
         begin
           write(#7,'wrong response, type again.');
           write(rltfile,'wrong response, type again.');
           goto 6;
         end;
end
until more_relations = 'n';
7 : ;
end;

procedure access_data;
label 1;
var
   option : string[25];
begin
     if (filesize(rel_dir) = 0) then
        begin
             writeln(#7,'we have not created any relation so far.');
             writeln(rltfile,'we have not created any relation so far.')
             goto 1;
        end;
     writeln;
     writeln('MENU B.');
     writeln('--------');
     writeln('  1. Insert a Tuple.');
```

```
writeln('  2. Delete a Tuple.');
writeln('  3. Update a Tuple.');
writeln('  4. Retrieve a Tuple.');
writeln;
write('GIVE YOUR OPTION : ');
readln(option);
writeln(rltfile);
writeln(rltfile,'MENU B.');
writeln(rltfile,'--------');
writeln(rltfile,'  1. Insert a Tuple.');
writeln(rltfile,'  2. Delete a Tuple.');
writeln(rltfile,'  3. Update a Tuple.');
writeln(rltfile,'  4. Retrieve a Tuple.');
writeln(rltfile);
write(rltfile,'GIVE YOUR OPTION : ');
writeln(rltfile,option);
if option = '1' then insert_a_tuple
else
if option = '2' then delete_a_tuple
else
if option = '3' then update_a_tuple
else
if option = '4' then retrieve_a_tuple
else
  begin
        error_handler;
        access_data;
  end;
```

```
1 : ;
end;

procedure interpret_option;
label 1,2,3;
var
     option : string[25];
begin
     write('GIVE YOUR OPTION : ');
     readln(option);
     write(rltfile,'GIVE YOUR OPTION : ');
     writeln(rltfile,option);
     if option = '1' then create_a_relation_scheme
     else
     if option = '2' then access_data
     else
     if option = '3' then
     alldone := true
     else
     error_handler;
end;
{     main program     }
begin
     initialize;
     repeat
          begin
               list_options;
               interpret_option;
```

```
            end;
        until alldone;
        close(rel_dir);
        close(rltfile);
end.
```

* SAMPLE OUTPUTS

```
(*    This is the sample output from PRIMS.    *)

MENU A.
--------
  1 : To create a relation scheme.
  2 : To Access Data.
  3 : Stop.

GIVE YOUR OPTION : 2
we have not created any relation so far.

MENU A.
--------
  1 : To create a relation scheme.
  2 : To Access Data.
  3 : Stop.

GIVE YOUR OPTION : 1

RELATION NAME : fycyc^%&(
not valid relation name.

RELATION NAME :
not valid relation name.

RELATION NAME : student
how many attributes does student have? : -8

give positive integer.

how many attributes does student have? : 0

give positive integer.

how many attributes does student have? : 3

ATTRIBUTE(1)
  NAME : name

ATTRIBUTE(2)
  NAME : standard

ATTRIBUTE(3)
  NAME : address


MENU A.
--------
  1 : To create a relation scheme.
  2 : To Access Data.
  3 : Stop.

GIVE YOUR OPTION : 1
we have following relations in relation directory :
student
```

```
RELATION NAME : class
how many attributes does class have? : 3

ATTRIBUTE(1)
  NAME : standard

ATTRIBUTE(2)
  NAME : teacher

ATTRIBUTE(3)
  NAME : roomno


MENU A.
--------
  1 : To create a relation scheme.
  2 : To Access Data.
  3 : Stop.

GIVE YOUR OPTION : 2

MENU B.
--------
  1. Insert a Tuple.
  2. Delete a Tuple.
  3. Update a Tuple.
  4. Retrieve a Tuple.

GIVE YOUR OPTION : 2

Relation  Name : err
err does not exist.


Relation  Name : student
The relation contains no records.

MENU A.
--------
  1 : To create a relation scheme.
  2 : To Access Data.
  3 : Stop.

GIVE YOUR OPTION : 2

MENU B.
--------
  1. Insert a Tuple.
  2. Delete a Tuple.
  3. Update a Tuple.
  4. Retrieve a Tuple.

GIVE YOUR OPTION : 1
```

```
Relation  Name : student

name : sita ram

standard : m.phil.

address : sutlej

you want to insert any more tuples of this relation?...(y/n): y

name : param

standard : m.phil.

address : mahanadi

you want to insert any more tuples of this relation?...(y/n): y

name : suman

standard : m.tech.

address : godavri

you want to insert any more tuples of this relation?...(y/n): n

you want to insert tuples in anyother relation?....(y/n): n

MENU A.
-------
  1 : To create a relation scheme.
  2 : To Access Data.
  3 : Stop.

GIVE YOUR OPTION : 2

MENU B.
-------
  1. Insert a Tuple.
  2. Delete a Tuple.
  3. Update a Tuple.
  4. Retrieve a Tuple.

GIVE YOUR OPTION : 7
wrong option, type again.

MENU B.
-------
  1. Insert a Tuple.
  2. Delete a Tuple.
  3. Update a Tuple.
  4. Retrieve a Tuple.

GIVE YOUR OPTION : 3
```

```
Relation  Name : student
which record number? : 2
name.
              old value : param
              new value : paramjit
class.
              old value : m.phil.
              new value :
address.
              old value : mahanadi
              new value :

you want to update any more tuples of this relation?...(y/n): n

you want to update tuples in anyother relation?....(y/n): n

MENU A.
--------
  1 : To create a relation scheme.
  2 : To Access Data.
  3 : Stop.

GIVE YOUR OPTION : 2

MENU B.
--------
  1. Insert a Tuple.
  2. Delete a Tuple.
  3. Update a Tuple.
  4. Retrieve a Tuple.

GIVE YOUR OPTION : 2

Relation  Name : student

which record number? : 3

you want to delete any more tuples of this relation?...(y/n): n

you want to delete tuples in anyother relation?....(y/n): n

MENU A.
--------
  1 : To create a relation scheme.
  2 : To Access Data.
  3 : Stop.

GIVE YOUR OPTION : 2

MENU B.
--------
  1. Insert a Tuple.
  2. Delete a Tuple.
  3. Update a Tuple.
  4. Retrieve a Tuple.
```

```
GIVE YOUR OPTION : 4

Relation  Name : student
s.no. rec sts          name      standard      address
    1        p     sita ram       m.phil.       sutlej
    2        p     paramjit       m.phil.     mahanadi
    3        a        suman       m.tech.      godavri

you want to retrieve tuples of anyother relation?....(y/n): y

Relation  Name : class
The relation contains no records.

MENU A.
--------
  1 : To create a relation scheme.
  2 : To Access Data.
  3 : Stop.

GIVE YOUR OPTION : 3


(*   This is the sample output from PRIMS.    *)

MENU A.
--------
  1 : To create a relation scheme.
  2 : To Access Data.
  3 : Stop.

GIVE YOUR OPTION : 2

MENU B.
---------
  1. Insert a Tuple.
  2. Delete a—Tuple.
  3. Update a Tuple.
  4. Retrieve a Tuple.

GIVE YOUR OPTION : 4

Relation  Name : class
The relation contains no records.

MENU A.
---------
  1 : To create a relation scheme.
  2 : To Access Data.
  3 : Stop.

GIVE YOUR OPTION : 1
we have following relations in relation directory :
student
class
```

```
RELATION NAME : class
class already exits.


RELATION NAME : ^*%E$bv
not valid relation name.

RELATION NAME : teacher
how many attributes does teacher have? : 3

ATTRIBUTE(1)
  NAME : name

ATTRIBUTE(2)
  NAME : standard

ATTRIBUTE(3)
  NAME : roomno


MENU A.
--------
  1 : To create a relation scheme.
  2 : To Access Data.
  3 : Stop.

GIVE YOUR OPTION : 2

MENU B.
--------
  1. Insert a Tuple.
  2. Delete a Tuple.
  3. Update a Tuple.
  4. Retrieve a Tuple.

GIVE YOUR OPTION : 1

Relation  Name : error
error does not exist.

Relation  Name : class

standard : m.phil.

teacher : k.k.bhat

roomno : 123

you want to insert any more tuples of this relation?...(y/n): t
wrong response, type again.
you want to insert any more tuples of this relation?...(y/n): y

standard : m.tech.
```

```
teacher : g.v.singh

roomno : 131

you want to insert any more tuples of this relation?...(y/n): y

standard : m.c.a.

teacher : r.c.phoha

roomno : 145

you want to insert any more tuples of this relation?...(y/n): n

you want to insert tuples in anyother relation?....(y/n): y

Relation  Name : student

name : mohan

standard : m.tech.

address : sutlej

you want to insert any more tuples of this relation?...(y/n): y

name : phani nath

standard : m.phil.

address : ganga

you want to insert any more tuples of this relation?...(y/n): n

you want to insert tuples in anyother relation?....(y/n): n

MENU A.
--------
  1 : To create a relation scheme.
  2 : To Access Data.
  3 : Stop.

GIVE YOUR OPTION : 2

MENU B.
--------
  1. Insert a Tuple.
  2. Delete a Tuple.
  3. Update a Tuple.
  4. Retrieve a Tuple.

GIVE YOUR OPTION : 3

Relation  Name : class
which record number? : 6
```

```
record number exeeds file size.
type again.
which record number? : fdx

give positive integer.
which record number? : 3
standard.
              old value : m.c.a.
              new value :
teacher.
              old value : r.c.phoha
              new value :
roomno.
              old value : 145
              new value : 154

you want to update any more tuples of this relation?...(y/n): n

you want to update tuples in anyother relation?....(y/n): y

Relation  Name : student
which record number? : 3
record doesnot exist.
type again.
which record number? : 5
name.
              old value : phani nath
              new value :
standard.
              old value : m.phil.
              new value : m.tech.
address.
              old value : ganga
              new value :

you want to update any more tuples of this relation?...(y/n): n

you want to update tuples in anyother relation?....(y/n): n

MENU A.
--------
  1 : To create a relation scheme.
  2 : To Access Data.
  3 : Stop.

GIVE YOUR OPTION : 2

MENU B.
--------
  1. Insert a Tuple.
  2. Delete a Tuple.
  3. Update a Tuple.
  4. Retrieve a Tuple.

GIVE YOUR OPTION : 4
```

```
Relation  Name : class
s.no. rec sts     standard      teacher       roomno
    1       p      m.phil.     k.k.bhat          123
    2       p      m.tech.    g.v.singh          131
    3       p       m.c.a.    r.c.phoha          154

you want to retrieve tuples of anyother relation?....(y/n): y

Relation  Name : student
s.no. rec sts         name     standard      address
    1       p     sita ram      m.phil.       sutlej
    2       p     paramjit      m.phil.     mahanadi
    3       a        suman      m.tech.      godavri
    4       p        mohan      m.tech.       sutlej
    5       p  phani nath      m.tech.        ganga

you want to retrieve tuples of anyother relation?....(y/n): n

MENU A.
-------

  1 : To create a relation scheme.
  2 : To Access Data.
  3 : Stop.

GIVE YOUR OPTION : 3


(*    This is the sample output from PRIMS.    *)

MENU A.
-------

  1 : To create a relation scheme.
  2 : To Access Data.
  3 : Stop.

GIVE YOUR OPTION : 2

MENU B.
-------

  1. Insert a Tuple.
  2. Delete a Tuple.
  3. Update a Tuple.
  4. Retrieve a Tuple.

GIVE YOUR OPTION : 1

Relation  Name : teacher

name : g.v.singh

standard : m.phil.

roomno : 123
```

```
you want to insert any more tuples of this relation?...(y/n): y

name : r.c.phoha

standard : m.tech.

roomno : 154

you want to insert any more tuples of this relation?...(y/n): yy
wrong response, type again.
you want to insert any more tuples of this relation?...(y/n): y

name : paramjit s.

standard : m.sc.

roomno : 136

you want to insert any more tuples of this relation?...(y/n): n

you want to insert tuples in anyother relation?....(y/n): n

MENU A.
-------
  1 : To create a relation scheme.
  2 : To Access Data.
  3 : Stop.

GIVE YOUR OPTION : 2

MENU B.
-------
  1. Insert a Tuple.
  2. Delete a Tuple.
  3. Update a Tuple.
  4. Retrieve a Tuple.

GIVE YOUR OPTION : 4

Relation  Name : teacher
s.no. rec sts          name     standard       roomno
    1       p    g.v.singh      m.phil.          123
    2       p    r.c.phoha      m.tech.          154
    3       p paramjit s.         m.sc.          136

you want to retrieve tuples of anyother relation?....(y/n): n

MENU A.
-------
  1 : To create a relation scheme.
  2 : To Access Data.
  3 : Stop.

GIVE YOUR OPTION : 3
```

```
(*   This is the sample output from PRIMS.    *)

MENU A.
-------
  1 : To create a relation scheme.
  2 : To Access Data.
  3 : Stop.

GIVE YOUR OPTION : 2

MENU B.
-------
  1. Insert a Tuple.
  2. Delete a Tuple.
  3. Update a Tuple.
  4. Retrieve a Tuple.

GIVE YOUR OPTION : 2

Relation  Name : class

which record number? : 1

you want to delete any more tuples of this relation?...(y/n): n

you want to delete tuples in anyother relation?....(y/n): y

Relation  Name : student

which record number? : 1

you want to delete any more tuples of this relation?...(y/n): n

you want to delete tuples in anyother relation?....(y/n): n

MENU A.
-------
  1 : To create a relation scheme.
  2 : To Access Data.
  3 : Stop.

GIVE YOUR OPTION : 2

MENU B.
-------
  1. Insert a Tuple.
  2. Delete a Tuple.
  3. Update a Tuple.
  4. Retrieve a Tuple.

GIVE YOUR OPTION : 4

Relation  Name : class
s.no. rec sts    standard     teacher      roomno
```

```
    1        a      m.phil.     k.k.bhat          123
    2        p      m.tech.     g.v.singh         131
    3        p       m.c.a.     r.c.phoha         154

you want to retrieve tuples of anyother relation?....(y/n): y

Relation  Name : student
s.no. rec sts          name      standard      address
    1        a     sita ram       m.phil.       sutlej
    2        p     paramjit       m.phil.     mahanadi
    3        a        suman       m.tech.      godavri
    4        p        mohan       m.tech.       sutlej
    5        p   phani nath       m.tech.        ganga

you want to retrieve tuples of anyother relation?....(y/n): n

MENU A.
-------

  1 : To create a relation scheme.
  2 : To Access Data.
  3 : Stop.

GIVE YOUR OPTION : 3
```