# ARAB REPUBLIC OF EGYPT

## THE INSTITUTE OF NATIONAL PLANNING

Memo. No. (1475 )

# INP DATABASE MANAGEMENT SYSTEM

by

Dr. Abdalla El-daoushy

June, 1988

CAIRO
SALAH SALEM St-NASR CTTY

# Contents

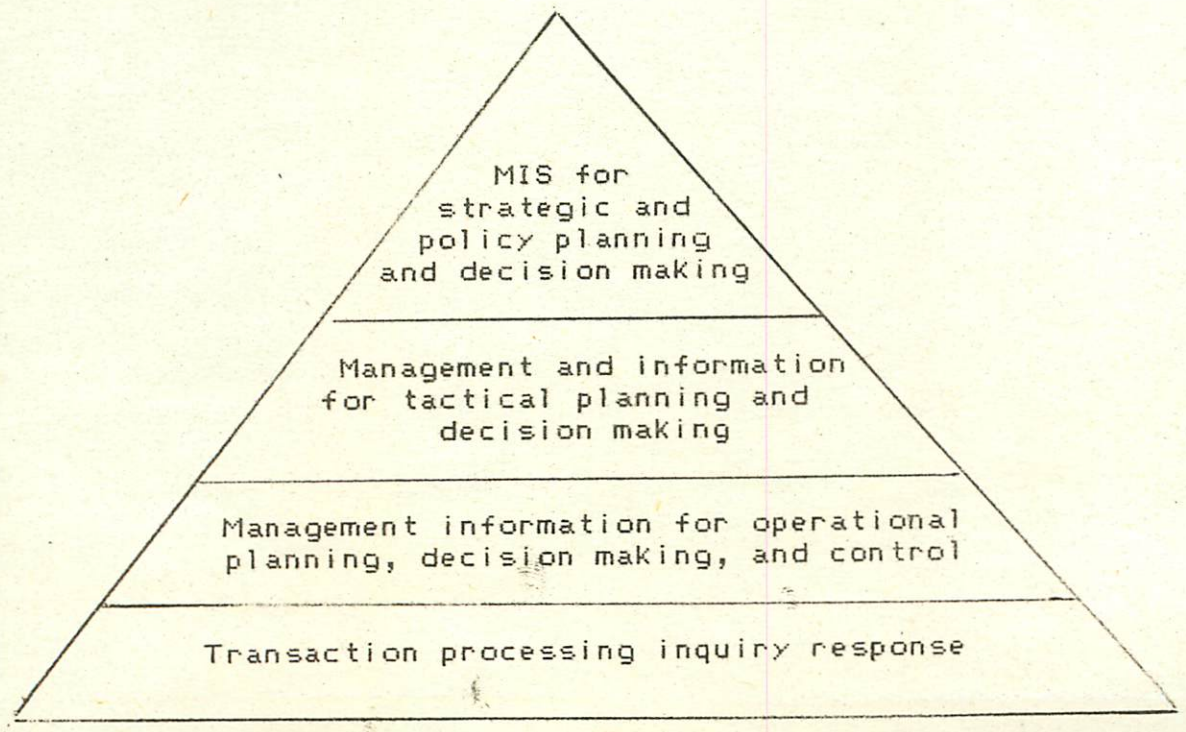
Preface :
---

INFORMATION in industry , commerce , planning , . . . , etc can be as vital as finance . Development may depend on efficiency of management of information , and efficiency , in turn , may depend on computerization .

The early use of computers in organizations tend to concentrate on clerical functions such as payroll accounting and processing of customer accounts. Now, they emphasize on applications which provide information processing support for managerial functions. A system built arround this emphasis is often termed a " Management Information System or MIS ".

An MIS utilizes computer hardware and software, manual procedures, management and decision models, and a data Base to provide information in support of operations management, and decision making functions in the organization.

In terms of the type of information and information processing provided by an MIS, the system may be described as a pyramid structure in which the bottom layer consists of the informaton for transaction processing, status inquires, etc; the next level consists of information resources in support of the day-to-day operations management; the third level consists of information system resources to aid in tactical planning and decision making for management control; and the top level consists of information resources in support of planning and decision making by higher levels of management.



The aim of this memo is to introduce the main concepts of Data Base Design process to meet information system requirements. In doing this, it discusses important issues such as the relational theory, data models and their relationship to relational theory, and the software used to support Data Bases.

The use of dBASE III PLUS is being considered in designing the INP's Information System and its Data Bases.

Although we shall often be concerned with limited examples , the usefulness of database technology extends beyond planning, industry and commerce .

Note :

This memo presupposes a basic understanding of general computer programming concepts before reading it.

## Brief Introduction to Data Base Systems Analysis and Design:

### What is a DataBase and What is a DataBase System ? :

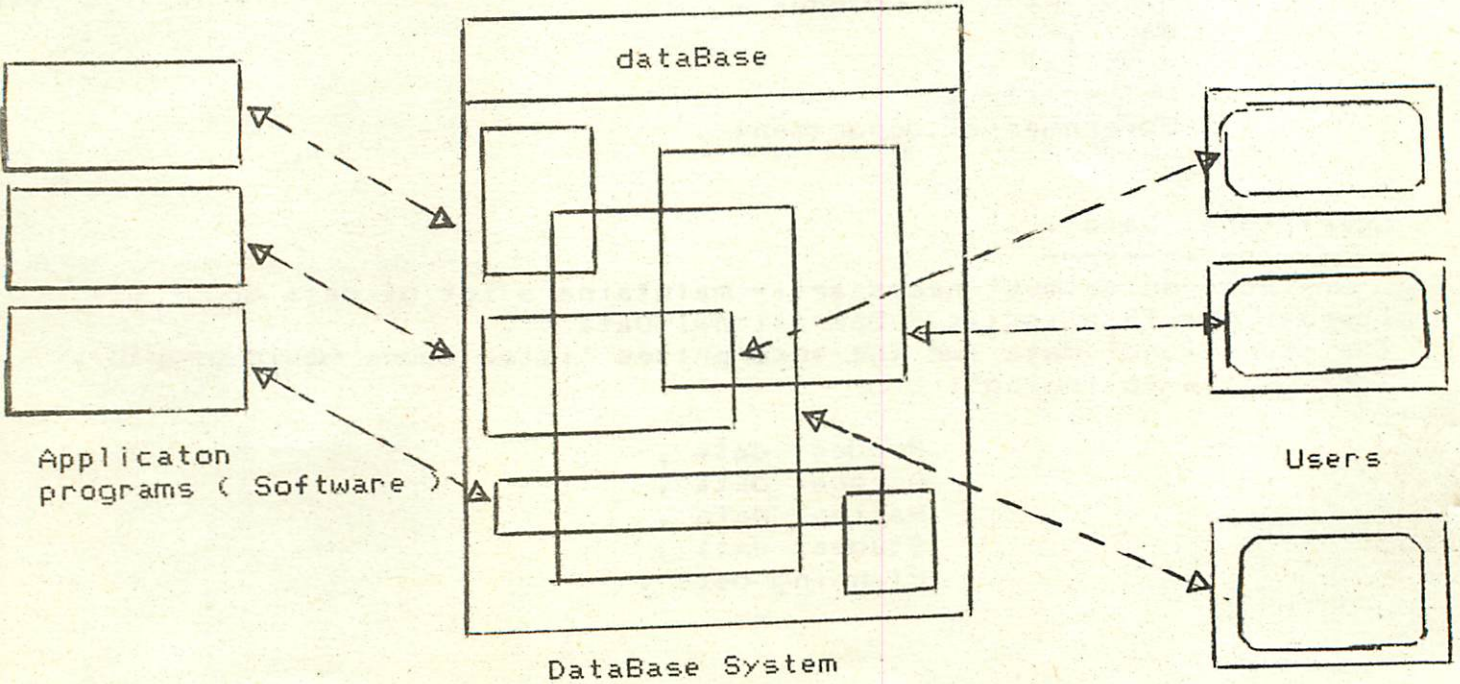
Databases are essential to an Organization's information system . The information system supports the Organization's functions , maintaining the data for these functions and assisting Users to interpret the data for decision-making .

The dataBase takes a central role in this process ; it is the repository ( the place where the data are stored ) of the data in the information system .

The database organizes data for easy storage and retrieval , or access.

A DataBase System is basically nothing more than a computer-based RECORD-keeping system : that is a System whose overall purpose is to record and maintain information . The information concerned can be anything but of significance to the Organization .

The following figure shows a simplified view of a dataBase System :



The figure shows a dataBase system involves 4 major components : data , hardware , software , and users . We will consider each of these briefly as follows :

DATA :
---

The data stored in the system is portioned into one or more dataBases . For simplicity , we assume that there is just one dataBase containing the totality of all stored data in the system .

*

A dataBase is a collection of stored "operational data" ( it is called operational data to distinguish it from input data , output data , and other kinds of data ) used by the application systems of some particular ENTERPRISE . This definition requires some explanation

ENTERPRISE :
---

Is simply any commercial , scientific , technical , or other organization . Some examples are :

Manufacturing company ,
Bank ,
Hospital ,
University ,
Governmental department .

Operational Data :
---

Any Enterprise must necessarily maintains a lot of data about its operation . This is its " Operational Data " .
The operational data for the enterprises listed above would probably include the following :

Product data ,
Account data ,
Patient data ,
Student data ,
Planning data .

---

* A modified version of Engles's original definition of a dataBase . C. J. DATE , IBM Corporation , Introduction to dataBase Systems , Third Edition --- Addison - Wesly publishing company .

Input Data :
---

Refers to information entering the system from the outside world ( typically on external files or from the KBD or a teminal ) ; such information may cause a change to be made to the " operational data " but it is not itself a part of the dataBase .

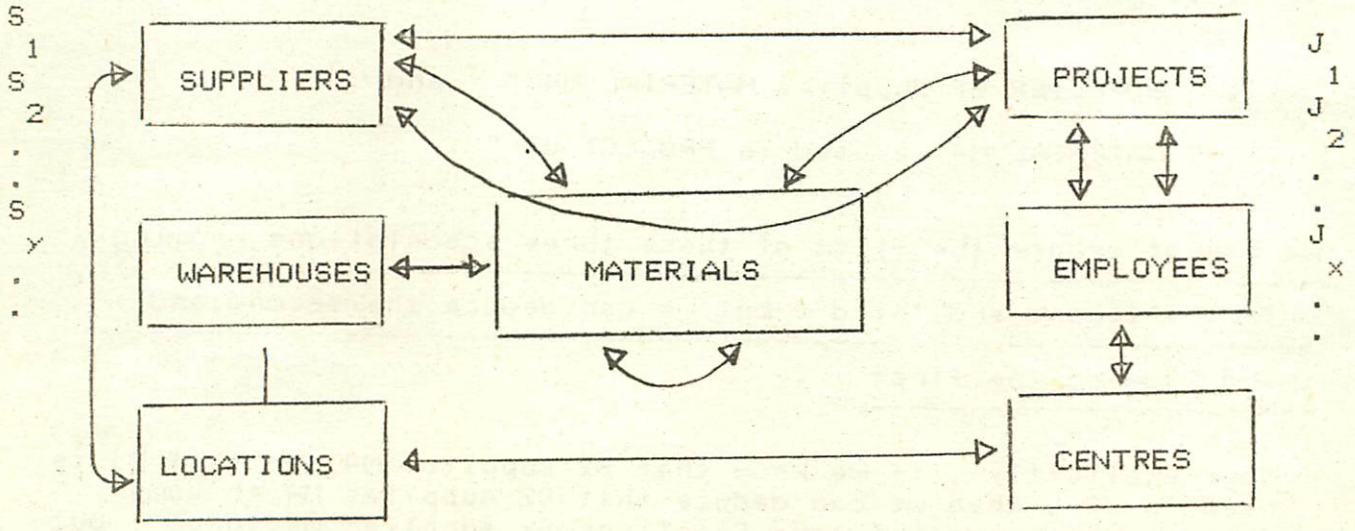
Output Data :
---

Refers to messages and reports printed or displayed from the system ; also , such reports contain information derived from the "operational data " but they are not part of the dataBase .

As an illustration of the concept of " Operational Data " , let us consider the case of a Govermental Organization ( INP for example ) in a little more details.

This INP wishes to retain information about the plan of Research PROJECTS it has on hand ; the MATERIALS used in these projects ; the SUPPLIERS ( Governmental Departments , Ministry of Planning for example ) who supply the MATERIALS ; the WAREHOUSES in which the MATERIALS are stored ; EMPLOYEES who work on the PROJECTS; and so on. These are the basic ENTITIES about which data are RECORDed in the dataBase. ( The word ENTITY is widely used in dataBase systems to mean any distinguishable OBJECT that is to be represented in the dataBase ).

Consider the following figure :




[ let for example that:
- Ministry of Planning replaces one of SUPPLIERS, and
- Research PROJECTS replaces PROJECTS. ]

It is important to notice that :

1. There will be ASSOCIATIONs or RELATIONSHIPs linking the basic ENTITies together . These are represented by connecting arrows . For example , there is an ASSOCIATION between SUPPLIERS and MATERIALS; each SUPPLIER supplies certain MATERIALS and conversely each MATERIALS is supplied by certain SUPPLIERS.

2. Although most of the relationships in the diagram associate two types of ENTITY , this is not always the case . In the figure , for example , there is one arrow connecting three ENTITIes ( SUPPLIERS - MATERIALS - PROJECTS ) . This would represent the fact that certain SUPPLIERS supply certain MATERIALS to certain PROJECTS. This is not the same as the combination of the SUPPLIERS - MATERIALS association and the MATERIALS - PROJECTS association ( in general ).

   For example :

   The information that :

   i , " SUPPLIER S2 supplies MATERIAL M4 to PROJECT J3 "

   tells us more than the combination :

   ii , " SUPPLIER S2 Supplies MATERIAL M4 " and

   iii , " MATERIAL M4 is used in PROJECT J3 " .

   We cannot deduce the first of these three associations knowing only the second and third ( but we can deduce the second and third knowing the first ) .

   More explicitly , if we know that S2 supplies M4 and that M4 is used in J3 , then we can deduce that S2 supplies M4 to some project Jx , and that some Suppliers Sy supplies M4 to J3 , but we cannot conclude that Jx is J3 or that Sy is S4.

   False conclusion such as these are examples of what is sometimes referred to as the CONNECTION TRAP .

. The figure also shows one arrow involving only one type of ENTITY ( MATERIALS ). This represents an association between one MATERIAL and another .

For example :

The fact that some MATERIALS are components of other MATERIALS ( a Printer-rypon is a component of a Line-printer assembly , which is also considered as a MATERIAL ).

Similarly ,

MATERIALS are used in PROJECTS , and conversely , PROJECTS use MATERIALS;

MATERIALS are stored in WAREHOUSES , WAREHOUSES store MATERIALS;

and so on .

Note that these relationships are all BIDIRECTIONAL : that is , they may all be traversed in either direction . Using this Phenomena ( BIDIRECTIONAL ) , we can construct the following QUERY :

QUERY:

The relationship between EMPLOYEES and CENTRES may be used to answer both the following questions :

i, Given an EMPLOYEE , find the corresponding CENTRE,

ii, Given a CENTRE, find all corresponding EMPLOYEES .

3. In general , the same ENTITies may be associated in any number of relationships .

In the figure , PROJECTS and EMPLOYEES are linked in two relationships :

i, One might represent the relationship : " WORKS ON "

e.g., ( the Employee WORKS ON the Project ) .

ii, The other represents the relationships " IS THE MANAGER OF "

e.g., ( the Employee IS THE MANAGER OF the Project ) .

## Why DataBase ? :

Why should an Enterprise choose to store its "operational data" in an integrated dataBase System ? . 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 the case now for many Enterprises where typically each application has its own private FILEs ( quite often its own private Tapes and Disk packs ) .

## DataBase Administrator ( DBA ) :

For 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 ) .

For the role of the DBA , it is sufficient to note that the job will involve both a high degree of technical expertise and the ability to understand and interpret management requirements at a senior level . ( In practice , the DBA may consist of a Team of People rather than just one person ) . It is important to realize that the position of the DBA within the Enterprise is a very senior one .

## ome Advantages of having DataBase System :

### Redundancy can be reduced :

In non-dataBase 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 a dataBase system , redundancy should be controlled -- i.e., the system should be aware of the redundancy .

### . Inconsistency can be avoided ( to some extent ) :

An inconsistent dataBase system supplies incorrect or conflicting information .

### . The data can be shared :

It means not only that existing applications can share the data in the dataBase , but also that new applications can be developed to operate on the same stored data . In other words , the data requirements of new applications may be satisfied without having to create any new stored FILEs .

### . Standards can be enforced :

With the central control of the dataBase , the DBA can ensure that all applicable standards are followed in the representation of the data . Standardizing stored data formats is particularly desirable as an aid to data interchange or migration between systems .

. Security restrictions can be applied :
---

The DBA :

a, can ensure that the only means of access to the dataBase system is through the proper channels , and

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 . [ Note that without such checks , the security of the data may actually be more at risk in a dataBase system than in a traditional FILing system ]

. 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 .

. Conflicting requirements can be balanced :
---

Knowing the overall requirements of the Enterprise -- as opposed to the requirements of any individual user --- the DBA can structure the dataBase system to provide an overall service that is "best for the Enterprise " . For example , a representation can be chosen for the data in storage that gives fast access for the most important applications at the cost of poor performance in some other applications .

## Relational Algebra & Relational Models in Data Base :

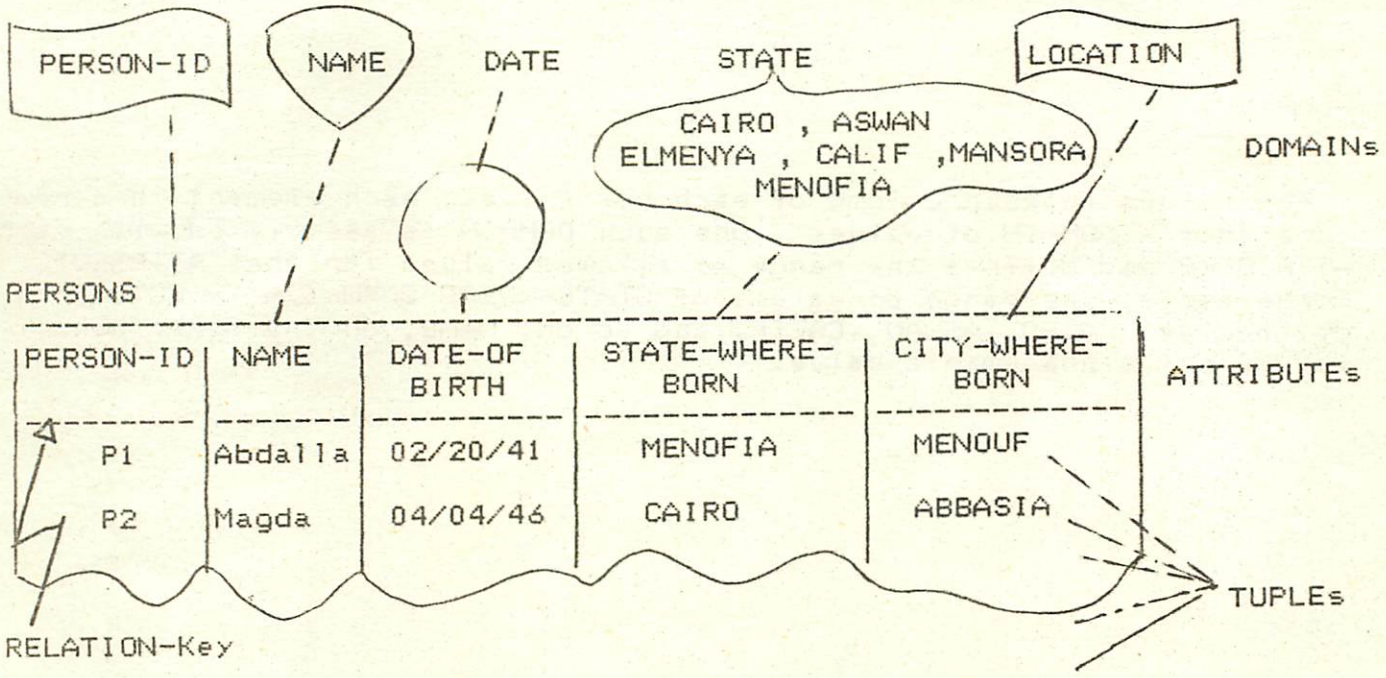
### BASIC TERMINOLOGY :

### Operational Data Files (RELATION ) :

A DataBase is made up of any number of RELATIONs ( Operational Data FILEs ) . Each RELATION is given a name.

Example :

The following figure ( Fig 1 ) represents a RELATION named PERSONS .




( Fig 1 )

The RELATION can be viewed as a table that is made up of a number of rows and columns.

ATTRIBUTE :
---

Each column in the table above is called an ATTRIBUTE ( Field ) and is given a name. The names of the columns in the table are: PERSON-ID, NAME, . . . etc. Each of these is called ATTRIBUTE.

TUPLEs :
---

The rows of the RELATION are called TUPLEs ( RECORDs ) and contain the DATA. The first row , for example , in the figure above contains data about a person identified by PERSON-ID = P1. This person has the name "Abdalla" and was born on Feb 20, 1941 , in MENOUFIA in MENOUF.

DOMAIN :
---

The values in each column of each row ( i.e., each element in a row come from a DOMAIN of values . One such DOMAIN is associated with each ATTRIBUTE and defines the range of allowed values for that ATTRIBUTE. For example, the range of values of STATE-WHERE-BORN ( ATTRIBUTE ) is defined as: CAIRO, ASWAN, CALIF, and so on. NAME, on the other hand, may be any alphanumeric value.

## Properties of RELATIONs :

Each RELATION has the following properties :

1. There is one column in the RELATION for each ATTRIBUTE. Each such column is given a name that is UNIQUE in the RELATION.

2. The entries in the column come from the same DOMAIN.

3. The order of the columns or ATTRIBUTEs in the RELATION has no significance.

4. The order of the rows is not significant.

5. There are no DUPLICATE rows.

## Keys of RELATIONs :

The RELATION-Key(s) is the most important property of a RELATION . This Key is the ATTRIBUTE or set of ATTRIBUTEs that uniquely identifies TUPLEs in a RELATION.

A RELATION-Key is formally defined as a set of one or more RELATION ATTRIBUTEs concatenated so that the following 3 properties hold for all time and for any instance of the RELATION :

1. Uniqueness : The set of ATTRIBUTEs takes on a unique value in the RELATION for each TUPLE.

2. Non-redundancy : If an ATTRIBUTE is removed from the set of ATTRIBUTES ( constituting the RELATION-Key ) , the remaining ATTRIBUTEs do not have the uniqueness property.

3. Validity : No ATTRIBUTE value in the RELATION-Key may be null.

It is possible for RELATIONs to have more than one RELATION-Key ; each Key is made up of a different set of ATTRIBUTEs.

## I. RELATIONal Algebra ( The Special RELATIONAL Operators ) :

We stress here on four basic Relational-Algebra-Operations viz:

SELECTION

PROJECTION

JOINING

DIVISION

All of these operators modified some what to operate on RELATIONs rather than arbitrary SETs.

### SELECTION Operation :

The SELECT Operator has one operand ( RELATION or result of a Relational-Algebra-Operation ).

### The general form is :

R1 = SELECT RELATION WHERE Condition

The result of this operation , R1 , is obtained by deleting from the RELATION all TUPLEs ( RECORDs ) that do not satisfy the condition.

or

The result of this operation , R1 , is obtained by SELECTing from the "RELATION" all TUPLEs satisfying the "Condition".

Example 1 :

Given the following RELATION :

RELATION : GDEPTS

| GDEPT-ID | PHONE | NO-TRAINS | LOCATED-IN-CITY |
|---|---|---|---|
| GD-A | 667 932 | 200 | Ismailia |
| GD-B | 725 172 | 310 | Port-said |
| GD-C | 636 182 | 75 | Port-said |
| GD-D | 679 305 | 105 | Menouf |

( fIG 2 )

( GDEPTS --> Governmental Departments and
NO-TRAINS --> No. of Trainees )

The Operation :

R1 = SELECT GDEPTS WHERE LOCATED-IN-CITY = 'Port-said'

will select all TUPLEs for GDEPTS that are located in Port-said and create the new RELATION R1 shown :

RELATION : R1

| GDEPT-ID | PHONE | NO-TRAINS | LOCATED-IN-CITY |
|---|---|---|---|
| GD-B | 725 172 | 310 | Port-said |
| GD-C | 636 182 | 75 | Port-said |

( fIG 3 )

Example 2 :
---

We can impose conditions on more than one ATTRIBUTE ( Field-name ).
For example:

R2 = SELECT GDEPTS WHERE LOCATED-IN-CITY = 'Port-said'

AND NO-TRAINS > 100

This operation would SELECT only one TUPLE ( RECORD ) from the RELATION: GDEPTS as follows :

RELATION : R2

| GDEPT-ID | PHONE | NO-TRAINS | LOCATED-IN-CITY |
|---|---|---|---|
| GD-B | 725 172 | 310 | Port-said |

( fIG 4 )

PROJECTion Operation :

The PROJECT Operator constructs a new RELATION from an existing one by SELECTing only specified ATTRIBUTEs ( Fields ) of the existing RELATION and at the same time eliminates the duplicate TUPLEs ( RECORDs ) in the newly formed RELATION.

Example 1 :

Given again the RELATION called GDEPTS:

RELATION: GDEPTS

| GDEPT-ID | PHONE | NO-TRAINS | LOCATED-IN-CITY |
|---|---|---|---|
| GD-A | 667 932 | 200 | Ismailia |
| GD-B | 725 172 | 310 | Port-said |
| GD-C | 636 182 | 75 | Port-said |
| GD-D | 679 305 | 105 | Menouf |

The operation :

R3 = PROJECT GDEPTS OVER GDEPT-ID , LOCATED-IN-CITY

results in the RELATION R3 shown :

RELATION : R3

| GDEPT-ID | LOCATED-IN-CITY |
|---|---|
| GD-A | Ismailia |
| GD-B | Port-said |
| GD-C | Port-said |
| GD-D | Menouf |

( fIG 5 )

Example 2 :

R4 = PROJECT GDEPTS OVER LOCATED-IN-CITY

results in :

RELATION : R4

| LOCATED-IN-CITY |
|---|
| Ismailia |
| Port-said |
| Menouf |

( fIG 6 )

Notice that one duplicated TUPLE has been eliminated .

## JOINing Operation :

JOINing is a method of combining two or more RELATIONs into a single RELATION.

At the beginning , it requires the choice of Fields ( ATTRIBUTEs ) to match RECORDs ( TUPLEs ) in the RELATIONs.

RECORDs in different RELATIONs , but with the same value of matching Fields ( ATTRIBUTEs ) , are combined into a single RECORD in the output RELATION.

## Example 1 :

Suppose we have the following two RELATIONs:

RELATION: GDEPTS

| GDEPT-ID | PHONE | NO-TRAINS | LOCATED-IN-CITY |
|---|---|---|---|
| GD-A | 667 932 | 200 | Ismailia |
| GD-B | 725 172 | 310 | Port-said |
| GD-C | 636 182 | 75 | Port-said |
| GD-D | 679 305 | 105 | Menouf |

RELATION: CLASSIFICATION

| GDEPT-ID | CLASS-CODE | NOS |
|---|---|---|
| GD-A | C1 | 50 |
| GD-A | C2 | 17 |
| GD-B | C3 | 20 |
| GD-C | C1 | 30 |
| GD-C | C4 | 11 |
| GD-C | C3 | 56 |

( CLASS-KODE --> Classification Code and
NOS --> Numbers )

( Fig 7 )

RELATION CLASSIFICATION contains the number of each Classification code grouped by each Governmental dept.

We can now JOIN the two RELATIONs using the common Field ( ATTRIBUTE ) , GDEPT-ID , by the operation :

R5 = JOIN CLASSIFICATION , GDEPTS OVER GDEPT-ID

The result is the following RELATION :

RELATION: R5

| GDEPT-ID | PHONE | NO-TRAINS | LOCATED-IN-CITY | CLASS-CODE | NOS |
|---|---|---|---|---|---|
| GD-A | 667 932 | 200 | Ismailia | C1 | 50 |
| GD-A | 667 932 | 200 | Ismailia | C2 | 17 |
| GD-B | 725 172 | 310 | Port-said | C3 | 20 |
| GD-C | 636 182 | 75 | Port-said | C1 | 30 |
| GD-C | 636 182 | 75 | Port-said | C4 | 11 |
| GD-C | 636 182 | 75 | Port-said | C3 | 56 |

( Fig 8 )

The ATTRIBUTE GDEPT-ID has been chosen as the matching ATTRIBUTE in both RELATIONs.

The condition of the JOIN is that RECORDs that have the same value ( equality JOIN ) of matching Fields in RELATIONs CLASSIFICATION and GDEPTS are combined into a single RECORD in the new RELATION , R5 ,.

Conditions of JOIN may be other than the equality , and we may , for example , define a " Greater than " or " less than " JOIN .

In the equi JOIN , two RECORDs are combined if their values of the two nominated fields are the same .

In a greater-than JOIN , two RECORDs would be combined if the value of one nominated field is > the value of another nominated field .

Example 2 :

Given :

RELATION : S

| S# | SNAME | CITY |
|---|---|---|
| S1 | Minst-of-Planning | Cairo |
| S2 | Minst-of-Agriculture | Tanta |
| S3 | Minst-of-Insurance | Asiout |
| S4 | Minst-of-Industry | Alex |

( Fig 9 )

RELATION : SP

| S# | CLASS-CODE | NOS |
|---|---|---|
| S1 | C1 | 300 |
| S1 | C2 | 200 |
| S1 | C3 | 400 |
| S1 | C4 | 200 |
| S1 | C5 | 100 |
| S2 | C1 | 300 |
| S2 | C2 | 400 |
| S3 | C2 | 200 |

To JOIN S and SP over Supplier number ( S# ) and the number of Trainees attending ( NOS ) is greater than 100 , we have :

RSSP = JOIN S , SP OVER S# , NOS > 100

The result is shown as follows :

RELATION : SSP

| S# | SNAME | CITY | CLASS-CODE | NOS |
|---|---|---|---|---|
| S1 | Minst-of-Planning | Tanta | C1 | 300 |
| S1 | Minst-of-Planning | Tanta | C2 | 200 |
| S1 | Minst-of-Planning | Tanta | C3 | 400 |
| S1 | Minst-of-Planning | Tanta | C4 | 200 |
| S2 | Minst-of-Agriculture | Tanta | C2 | 300 |
| S2 | Minst-of-Agriculture | Tanta | C2 | 400 |
| S3 | Minst-of-Insurance | Asiout | C2 | 200 |

( Fig 10 )

DIVISION Operation :
---

Given RX , RY ; The result of the operation :

Z = RX DIVISION RY

or

Z = RX / RY

is a RELATION , Z , whose TUPLEs ( RECORDs ) are those of RX with Fields ( ATTRIBUTEs ) that are not in RY .

Example 1 :
---

Given the RELATIONs R9 and R6 as follows :

RELATION : R9

| COMPANY | LOCATION |
|---|---|
| COMP1 | Port-said |
| COMP2 | Menouf |
| COMP1 | Cairo |
| COMP3 | Port-said |
| COMP2 | Alex |
| COMP3 | Alex |
| COMP3 | Menouf |

RELATION : R6

| LOCATION |
|---|
| Port-said |
| Alex |

RELATION : R7

| COMPANY |
|---|
| COMP3 |

The operation :

R7 = R9 DIVISION R6 , or

R7 = R9 / R6

yields the result shown in RELATION R7 above . In this RELATION , COMP3 is the only COMPANY (Field-value) for which there is a RECORD with Port-said and Alex ( i.e., COMP3 , Port-said and COMP3 , Alex ) in R9 . The other Companies , COMP1 and COMP2 , do not satisfy this condition .

Example 2 :

Given the following two RELATIONS :

RELATION : FNAME1

| Rec # | G2 | G1 | F1 |
|---|---|---|---|
| 1 | A | d | 3 |
| 2 | C | h | 7 |
| 3 | B | h | 7 |
| 4 | B | y | 4 |
| 5 | A | y | 4 |
| 6 | A | h | 7 |
| 7 | B | d | 3 |
| 8 | C | y | 4 |


RELATION : FNAME2

| Rec # | G1 | F1 |
|---|---|---|
| 1 | d | 3 |
| 2 | h | 7 |
| 3 | y | 4 |

Then ,

FNAME = FNAME1 / FNAME2

yields the RELATION : FNAME shown as follows :

RELATION : FNAME

| Rec # | G2 |
|---|---|
| 1 | A |
| 2 | B |

in which there is only one Field ( ATTRIBUTE ) which is the only one that belongs to FNAME1 and not to FNAME2 .

The value A of ATTRIBUTE G2 is included in the result of DIVISION because A appears concatenated with every RECORD of FNAME2 in FNAME1 ; that is to say , FNAME1 includes all of :

| G2 | G1 | F1 |
|---|---|---|
| A | d | 3 |
| A | h | 7 |
| A | y | 4 |

The value C of G2 is not included in the result od DIVISION because :

| G2 | G1 | F1 |
|---|---|---|
| C | d | 3 |

does not occur in FNAME1 .

## II. RELATIONal Models in Data Base :

The usefulness of the Relational Models in Data Analysis comes from the 3 objectives :

1. It can be used to identify the Users requirements and present these requirements in a way that is easily understood both by the Users and by the computer professionals .

2. It can be easily converted to a Technical implementation .

3. It provides rules and criteria for efficient logical data representation .

The Enterprise's Data Base is specified as a set of tables , or RELATIONs . For example :

The following table :

RELATION: PERSONS

| Name | Address | Date-Of-Birth |
|---|---|---|
| Abdalla | Nasr-city, Cairo | 02/20/41 |
| Magda | Abbasia, Cairo | 04/04/46 |
| Afaf | Heliopolis,Cairo | 04/04/44 |

( Fig 11 )

illustrates one such RELATION ( named ,again, PERSONS ).

The RELATION: PERSONS ,here, contains information on people in an Organization ( INP ,for example, ); it stores each person's Name , Address , and Date-Of-Birth.

A complete specification of Enterprises usually consists of a large number of RELATIONs.

Note that :
---

The tabular presentation of RELATIONs satisfies the first objective of Data Analysis. It is understood by Users as well as Computer Professionals. It is also complete in the sense that it can be used to specify the data within any organization.

The second objective ---- the conversion to physical implementation --- is also satisfied by the Relational Model. To do this, a DataBASE Management System that supports the Relational Model must be available on a computer. As a result, direct conversion from a Relational specification to physical implementation is becoming possible.

The third objective deals with the following criteria for good logical data structure :

i, Each fact should be stored once in the DataBase.

ii, The DataBase should be consistent following DataBase operations.

iii, The DataBase should be resilient to change.

For first criterion: Storing each fact once at most not only removes storage redundancy but also improves DataBase consistency ( the second criterion ).

If the same fact is stored twice, the DataBase then becomes inconsistent. In an inconsistent DataBase, it is possible to obtain different DataBase outputs about the same fact from different places, raising doubt about the reliability of information in the DataBase. So, storing each fact only once improves DataBase consistency by ensuring that consistent information be obtained from the DataBase after any DataBase operations.

The third criterion deals with a different aspect. It is a consequent of the environment in which DataBase exist. This environment is usually in a state of constant change; consequently , the DataBase must also be continually redesigned to meet continually changing user requirements.

TERMINOLOGY ( continued ) :
---

ATTRIBUTEs and DOMAINs :
---

You may be wondering why there is a distinction between DOMAIN and ATTRIBUTE ?. In ( Fig 12 ) ,for example, why not use the same name for both DOMAIN and ATTRIBUTE ?. The reason for this separation is that a DOMAIN is a set of values: For example, in the following figure ( Fig 2 ):

RELATION: MANUFACTURE

| PARTS | PARTS | INTEGER | <-- DOMAINs |
|---|---|---|---|
| ASSEMBLY | COMPONENT | QTY | <-- ATTRIBUTEs |
| AUTO | ENGINE | 1 | |
| AUTO | WHEEL | 4 | |
| AUTO | CHASSIS | 1 | |
| WHEEL | TIRE | 1 | |
| WHEEL | NUTS | 6 | |
| ENGINE | CYLINDER | 4 | |

( Fig 12 )

Consider the DOMAIN PARTS. This DOMAIN contains the names of all PARTS of interest to an Organization. RELATION MANUFACTURE stores the components that make up each PART. Thus an AUTO is made up of 1 ENGINE, 4 WHEELs, and 1 CHASSIS. A WHEEL is composed of 1 TIRE and 6 NUTS; and an ENGINE has 4 CYLINDERs.

In this RELATION, the DOMAIN PARTS appears twice. Therefore, it is necessary to distinguish between the first and second set of the same DOMAIN in the one RELATION. This can be done by using ATTRIBUTEs.

Both the ATTRIBUTEs ASSEMBLY and COMPONENT map to the same DOMAIN ( PARTS ). But the values in each column ( although they come from the same DOMAIN ) have different meanings. For example, WHEEL in column ASSEMBLY means that WHEEL is the major PART or ASSEMBLY made up of some other PARTS; while, WHEEL in the column COMPONENT means that WHEEL is the component of some major PART or ASSEMBLY.

It is common to choose DOMAIN names to signify value sets ( INTEGER, ALPHANUMERIC, . . . etc. ).

The ATTRIBUTE names are chosen to identify as closely as possible the meaning of values in their columns.

Example :
---

Consider the following RELATION :

RELATION: WORK

| PERSON-ON-PROJECT | PROJECT-WORKED-ON | MONTHS-WORKED-ON-PROJECT |
|---|---|---|
| Amani | P#1 | 5 |
| Mohamed | P#2 | 3 |
| Ali | P#1 | 2 |

( Fig 13 )

The ATTRIBUTE names here are chosen to identify as closely as possible the meaning of values in their columns.
The ATTRIBUTE PERSON-ON-PROJECT comes from the DOMAIN ALPHANUMERIC because the values are alphanumeric ( the set of names of all the people in the INP ). Similarly , the ATTRIBUTE PROJECT-WORKED-ON comes from the DOMAIN Project-Number; and the ATTRIBUTE MONTHS-WORKED-ON-PROJECT maps onto the DOMAIN INTEGER.

It should be noticed that an ATTRIBUTE may also consist of more than one DOMAIN.

Example :
---

In the following figure ( Fig 14 ) :

RELATION: NAMES

| PERSON-ID | NAME | |
|---|---|---|
| | FIRST-NAME | LAST-NAME |
| S7530 | Abdelkader | Hamza |
| S7601 | Khalid | Lotfi |
| S7903 | Fatheia | Zaghlool |
| S8017 | Zolfa | Shalabi |

( Fig 14 )

the ATTRIBUTE NAME may map onto two DOMAINs: FIRST-NAME and LAST-NAME.

It is also possible to have RELATIONs where the ATTRIBUTE values are themselves RELATIONs. The ATTRIBUTE DOMAINs are INSTANCEs of that RELATION.

Example :

The following figure ( Fig 15 ) :

RELATION: LIVED-IN

| PERSON | RESIDENCE | |
|---|---|---|
| | CITY | DATE-MOVED-IN |
| Abdelhameed | Alex | 03/03/71 |
| | Port-said | 07/07/80 |
| | Cairo | 08/08/81 |
| | CITY | DATE-MOVED-IN |
| Aziz | Port-said | 04/05/73 |
| | Zagazig | 07/06/75 |
| | Cairo | 08/08/77 |

( Fig 15 )

illustrates the RELATION: LIVED-IN, which identifies the places of residence of people.
Here, the DOMAIN of ATTRIBUTE RESIDENCE is a RELATION. The RELATION ATTRIBUTEs are CITY and DATE-MOVED-IN. The values of these ATTRIBUTEs state the city in which a PERSON resided, together with the date on which that Person first took up residence in that city.

Examples :
---

The RELATION-Key of RELATION PERSONS in ( Fig 12 ) is PERSON-ID, as there is one TUPLE for each Person.

The RELATION WORK in ( Fig 13 ) ,on the other hand, has a RELATION-Key made up of two ATTRIBUTEs:

PERSON-ON-PROJECT and PROJECT-WORKED-ON

Values in these two columns together uniquely identify TUPLEs in WORK. PERSON-ON-PROJECT cannot be a RELATION-Key by itself because there can be more than one TUPLE with the same value of PERSON-ON-PROJECT. That is, one Person can work on more than one project.

Similarly, PROJECT-WORKED-ON cannot be a RELATION-Key by itself because more than one Person can work on the same project.

The RELATION-Key is often called the CANDIDATE KEY. If a KEY is the only KEY of the RELATION, it is generally refereed to as the PRIMARY KEY.

PRIME and NON-PRIME ATTRIBUTE :
---

PRIME ATTRIBUTE is ATTRIBUTE that is part of at least one RELATION-Key.

A Non-PRIME ATTRIBUTE is not part of any RELATION-Key.

Example :
---

In RELATION: PERSONS in ( Fig 11 ) , PERSON-ID is a PRIME ATTRIBUTE; NAME, DATE-OF-BIRTH, STATE-WHERE-BORN, and CITY-WHERE-BORN are all Non-PRIME ATTRIBUTEs.

In RELATION: WORK of ( Fig 13 ), PERSON-ON-PROJECT and PROJECT-WORKED-ON are both PRIME ATTRIBUTEs; MONTHS-WORKED-ON-ROJECT i the only Non-PRIME ATTRIBUTE in this RELATION.

Consistency :
---

To distinguish between good and bad RELATIONal Design, choose RELATIONs that preserve consistency following dBASE operations that store each fact at MOST once in the dataBase. RELATIONs that do this are said to be in NORMAL FORM. In Non-NORMAL RELATIONs, ANOMALIEs can arise after dBASE TUPLE operations.

## TUPLE Operations :

Three Tuple operations are as follows :

### 1. ADD TUPLE Operation :

General Form :

ADD TUPLE ( RELATION-name , ( ATTRIBUTE Values ) )

This operation Adds a new Tuple to a RELATION.
The ATTRIBUTE Values of the TUPLE are given as part of the operation.

Example :

ADD TUPLE ( PERSONS , ( Amani , Eldokki , 06/09/44 ) )

This operation Adds a new row to the RELATION in ( Fig 11 ).

Note that: An ADD TUPLE operation will not be allowed if it duplicates a RELATION-Key.

## 2. DELETE TUPLE Operation :

General Form :

DELETE TUPLE ( RELATION-name , ( ATTRIBUTE-values ) )

This operation deletes a TUPLE from a RELATION.

Example :

DELETE TUPLE ( PERSONS , ( Abdalla , Nasr-city , 02/20/41 ) )

This operation would delete the first row from the RELATION in ( Fig 1 ).

## 3. UPDATE TUPLE Operation :

General Form :

UPDATE TUPLE ( RELATION-name , ( Old-ATTRIBUTE-Values ) ,
( New-ATTRIBUTE-Values ) )

This operation changes the TUPLE in a RELATION.

Example :

UPDATE TUPLE ( PERSONS , ( Afaf , Heliopolis , 04/04/44 ) ,
( Afaf , Nasr-city , 04/04/44 ) )

This operation would change Afaf's address in the RELATION in ( Fig 11 ).

Note that: An Update will not be allowed if it duplicates a RELATION-Key.

ANOMALIES In TUPLE Operations :

Anomalies can be illustrated by considering, for example, the following RELATION: ASSIGN.

RELATION: ASSIGN

| PERSON-ID | PROJECT-BUDGET | PROJECT | TIME-SPENT-BY-PERSON-ON-PROJECT |
|---|---|---|---|
| S75 | 32 | P1 | 7 |
| S75 | 40 | P2 | 3 |
| S79 | 32 | P1 | 4 |
| S79 | 27 | P3 | 11 |
| S80 | 40 | P2 | 5 |
| --- | 17 | P4 | - |

( Fig 16

This RELATION describes the time spent by Persons on projects, together with the project Budget.

. It is seen that some project Budgets are repeated; for example, project P1 has a Budget of 32 and appears twice. The same for project P2.

. There is a Budget of a project P4 that has no Person assigned to it which requires dummy or null values to be used in the columns.

The existence of one or more of such things in a RELATION is regarded as an ANOMALY.

ANOMALies also arise in TUPLE Operations; for example:

1. Suppose the Operation:

   ADD TUPLE ( ASSIGN , ( S85 , 35 , P1 , 9 ) )

   is executed to Add the fact that PERSON S85 has worked on Project P1.

   This operation creates an ANOMALY because now there will be two TUPLEs in ASSIGN with conflicting PROJECT-BUDGET Values for the same Project P1.

2. Suppose the operation :

   DELETE TUPLE ( ASSIGN , ( S79 , 27 , P3 , 11 ) )

   is executed to Delete the fact that S79 worked on Project P3.

   This operation ,oversight, delete also the Project Budget of P3, thus creating a DEIETE ANOMALY.

3. Suppose the operation:

   UPDATE TUPLE ( ASSIGN , ( S75 , 32 , P1 , 7 ) ,
   ( S75 , 35 , P1 , 7 ) )

   is used to change P1's Budget.

   This operation is a type of ANOMALY because now there are two TUPLEs with different values for P1's Budget.

Solution ? ( Resolution of ANOMALies in RELATIONs Design ) :

The above undesirable properties can be removed by DECOMPOSING ASSIGN into the following two RELATIONs:

RELATION: PROJECTS

| PROJECT | PROJECT-BUDGET |
|---|---|
| P1 | 32 |
| P2 | 40 |
| P3 | 27 |
| P4 | 17 |

RELATION: ASSIGNMENTS

| PERSON-ID | PROJECT | TIME-SPENT-BY-PERSON-ON-PROJECT |
|---|---|---|
| S75 | P1 | 7 |
| S75 | P2 | 3 |
| S79 | P1 | 4 |
| S79 | P3 | 11 |
| S80 | P2 | 5 |

( Fig 17 )

So :

1. No ANOMALies arise if a Project Budget is changed by the operation:

   UPDATE TUPLE ( PROJECTS , ( P1 , 32 ) , ( P1 , 35 ) )

2. No dummy values are necessary to store Projects Budget values for Projects that have no Persons assigned to them.

3. DELETE TUPLE ( ASSIGNMENTS , ( S79 , P3 , 11 ) )

   does not delete the Budget of Project P3.

4. No ANOMALY can be created if a Person's contribution is Added to ASSIGNMENTS. For example:

   ADD TUPLE ( ASSIGNMENTS , ( S85 , P1 , 9 ) )

   cannot ,oversight, create an ANOMALY.

The RELATIONs in which the design does not create ANOMALies are said to be in NORMAL FORM.

## Functional Dependency ( FD ) :

FD is a term derived from algebra ; it means the Dependenc: of values of one ATTRIBUTE(s) on those of another ATTRIBUTE(s).

Formally,

A set of ATTRIBUTE values X is Functionally Dependent on a set of ATTRIBUTE values Y if a given set of values for each ATTRIBUTE in Y determine a unique value for the set of ATTRIBUTEs in X.

The notation:

Y ----> X ( i.e., Y determines X )

is often used to denote that X is Functionally Dependent on Y.

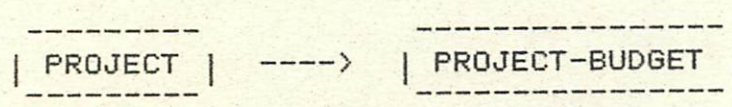
The ATTRIBUTE values in Y are sometimes known as the determinant of FD: Y ----> X .

## Example 1 :

In ( Fig 16 ) , PROJECT-BUDGET is FD on PROJECT because each PROJECT has one given BUDGET value. Thus once a PROJECT name is known, a unique value of PROJECT-BUDGET is immediately determined. Hence, we have the FD:

PROJECT ----> PROJECT-BUDGET

In the following figure:




( Fig 18 )

The arrow indicates that each value of PROJECT uniquely determines a value of PROJECT-BUDGET.

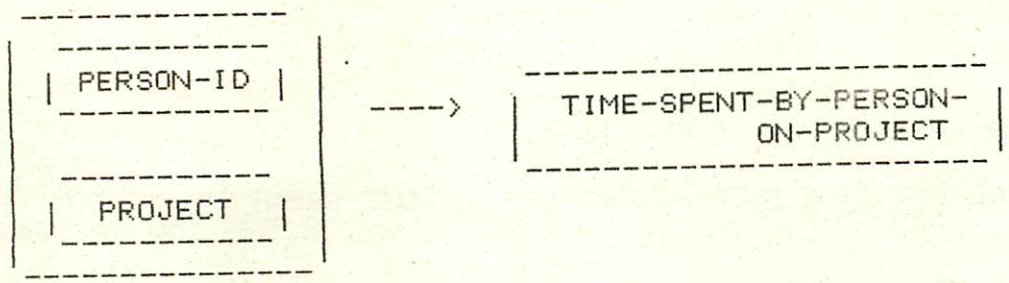
Example 2 :

In the RELATION ASSIGN of ( Fig 16 ) or ASSIGNMENT of ( Fig 17 ), once values for PERSON-ID and PROJECT are known, a unique value of TIME-SPENT by that Person in that Project is determined.

Formally, this FD is defined as:

PERSON-ID , PROJECT ----> TIME-SPENT-BY-PERSON-ON-PROJECT

or




( Fig 19 )

It is also possible to have 2 ATTRIBUTEs that are Functionally Dependent on each other. In this case, we have:

X ----> Y and Y ----> X or Y ----> X

It is called Mutual FD.

## Full Functional Dependency :

The term full FD is used to indicate the minimum set of ATTRIBUTEs in a Determinant of an FD.

Formally ,

Given Y ---> X , then X is fully FD on Y if and only if ( iff ):

i, X is FD on Y , and

ii, X is not FD on any subset of Y.

## Example 1 :

In the RELATION: ASSIGN of ( Fig 16 ),

i, PERSON-ID, PROJECT, PROJECT-BUDGET ----> TIME-SPENT-BY-PERSON-ON-PROJECT

is not a full FD.

But:

ii, PERSON-ID, PROJECT --> TIME-SPENT-BY-PERSON-ON-PROJECT

is FD because neither PERSON-ID ----> TIME-SPENT-BY-PERSON-ON-PROJEC

nor PROJECT ----> TIME-SPENT-BY-PERSON-ON-PROJEC

hold TRUE.

Example 2 :

In

RELATION : FIRST

| S# | CITY | CLASS-CODE | NOS |
|---|---|---|---|
| S1 | TANTA | C1 | 300 |
| S1 | TANTA | C2 | 200 |
| S1 | TANTA | C3 | 400 |
| S1 | TANTA | C4 | 200 |
| S2 | ASIOUT | C1 | 300 |
| S2 | ASIOUT | C2 | 400 |
| S3 | ASIOUT | C2 | 200 |

( Fig 20 )

i, S# , CLASS-CODE , CITY ----> NOS is not full FD .

ii, S# , CLASS-CODE ----> NOS is full FD .

NORMAL FORMs :

The RELATION is defined as made up of two components :

1. The ATTRIBUTEs , and

2. The FD between the ATTRIBUTEs .

So , a RELATION can take the form :

R1 = ( { X , Y , Z } , { X ---> Y , X ---> Z } )

in which the first component is the ATTRIBUTEs and the second component is the FDs .

Example :

In the RELATION ASSIGN of { Fig 16 } , the first component is :

{ PERSON-ID , PROJECT , PROJECT-BUDGET ,
TIME-SPENT-BY-PERSON-ON-PROJECT } .

The second component is :

{ PERSON-ID , PROJECT ---> TIME-SPENT-BY-PERSON-ON-PROJECT ,
PROJECT ---> PROJECT-BUDGET } .

Note that : the FDs between ATTRIBUTEs in a RELATION are obviously important when determining the RELATION-Key .

As mentioned before, the RELATIONs in which the design does not create ANOMALies are said to be in NORMAL FORM .

There are a number of NORMAL FORMs . The three best known of them are the 1st , 2nd , and 3rd NORMAL FORM , and often referred to as : 1NF , 2NF , and 3NF .

1NF :

RELATIONs are in 1NF if all DOMAINs are simple .

Example 1 :

Most of the RELATIONs that are discussed so far are in 1NF . The only RELATION not in 1NF is LIVED-IN of ( Fig 15 ) where the DOMAIN RESIDENCE is not simple .

RELATIONs that have non-simple DOMAINs are not in 1NF ; they are called Un-NORMALized RELATIONs .

A RELATION can be NORMALized by replacing the non-simple DOMAINs with simple DOMAINs. For example, the NORMALized FORM of the RELATION: LIVED-IN can be shown as follows:

RELATION : LIVED-IN

| PERSON | CITY | DATE-MOVED-IN |
|---|---|---|
| Abdelhameed | Alex | 03/03/71 |
| Abdelhameed | Port-said | 07/07/80 |
| Abdelhameed | Cairo | 08/08/81 |
| Aziz | Port-said | 04/05/73 |
| Aziz | Zagazig | 07/06/75 |
| Aziz | Cairo | 08/08/77 |

( Fig 21 )

Example 2 :

The following RELATION :

RELATION: SUPPLIERS

| S# | PN | |
|---|---|---|
| | CLASS-CODE | NOS |
| S1 | C1 | 300 |
| | C2 | 200 |
| | C3 | 400 |
| | C4 | 200 |
| | C5 | 100 |
| S2 | C1 | 300 |
| | C2 | 400 |
| S3 | C2 | 200 |

( Fig 22 )

is not in 1NF ( or un-NORMALized ) since the DOMAIN PN is not simple.

Replacing the non-simple DOMAINs by simple ones, we get the following NORMALized RELATION:

RELATION: SUPPLIERS

| S# | CLASS-CODE | NOS |
|---|---|---|
| S1 | C1 | 300 |
| S1 | C2 | 200 |
| S1 | C3 | 400 |
| S1 | C4 | 200 |
| S1 | C5 | 100 |
| S2 | C1 | 300 |
| S2 | C2 | 400 |
| S3 | C2 | 200 |

( Fig 23 )

## 2NF :

A RELATION is in a 2nd Normal Form ( 2NF ) iff it is in 1NF and every non-primary-Key ATTRIBUTE is full FD on the primary-Key .

## Example 1 :

The following RELATION :

RELATION: FIRST

| S# | CITY | CLASS-CODE | NOS |
|---|---|---|---|
| S1 | TANTA | C1 | 300 |
| S1 | TANTA | C2 | 200 |
| S1 | TANTA | C3 | 400 |
| S1. | TANTA | C4 | 200 |
| S2 | ASIOUT | C1 | 300 |
| S2 | ASIOUT | C2 | 400 |
| S3 | ASIOUT | C2 | 200 |
| S4 | ALEX | -- | --- |

( Fig 24 )

is not in 2NF since CITY is not full FD on the RELATION-Key S# .

## Example 2 :

The RELATION ASSIGN in ( Fig 16 ) is not in 2NF because the non-prime ATTRIBUTE PROJECT-BUDGET is not fully FD on the RELATION-Keys PERSON-ID and PROJECT. Here, PROJECT-BUDGET is, in fact, fully FD on PROJECT (which is a subset of the RELATION-Key).

But,

In ( Fig 17 ), RELATION: ASSIGN was decomposed into two RELATIONs: PROJECTS and ASSIGNMENTS. Both these RELATIONs are in 2NF.

## 3NF :

A RELATION is in 3NF if it has the following properties:

1. The RELATION is 2NF.

2. The non-prime ATTRIBUTEs are mutually independent ( i.e., no non-prime ATTRIBUTE is FD on another non-prime ATTRIBUTE ).

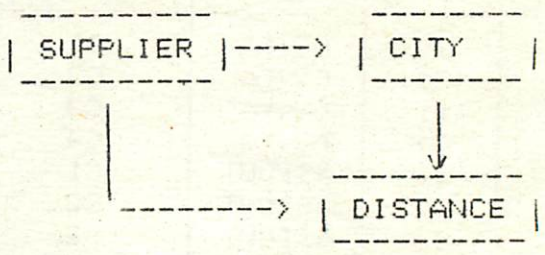
## Example 1 :

An example of a RELATION that is in 2NF but not in 3NF is the following RELATION:

RELATION: SUPPLIERS

| SUPPLIER | CITY | DISTANCE |
|---|---|---|
| S1 | Port-said | 200 |
| S2 | Port-said | 200 |
| S3 | Menouf | 1800 |
| S4 | Menouf | 1800 |

( Fig 25 )




( Fig 26 )

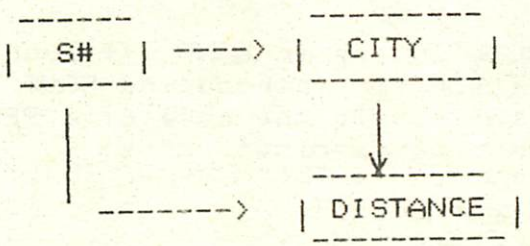
FD between SUPPLIERS ATTRIBUTEs

Example 2 :

The RELATION :

RELATION: R1

| S# | CITY | DISTANCE |
|---|---|---|
| S1 | TANTA | 110 |
| S2 | TANTA | 110 |
| S3 | ASIOUT | 375 |
| S4 | ASIOUT | 375 |




( Fig 27 )

is in 2NF since CITY and DISTANCE are full FDs on S#. Since CITY and DISTANCE ( non-prime ATTRIBUTEs ) are mutually dependent, then R1 is not in 3NF.

But,

R1 can be decomposed into the two RELATIONs: LOCATIONS and DISTANCES so that both are in 3NF:

RELATION: LOCATIONS

| S# | CITY |
|---|---|
| S1 | TANTA |
| S2 | TANTA |
| S3 | ASIOUT |
| S4 | ASIOUT |

RELATION: DISTANCES

| CITY | DISTANCE |
|---|---|
| TANTA | 110 |
| ASIOUT | 375 |

( Fig 28 )

## Boyce-Codd NORMAL FORM ( BCNF ) :

The above 3NF was subsequently replaced by a stronger definition. The new definition is due to Boyce and Codd; hence the term BCNF is used to distinguish the new 3NF from the old.

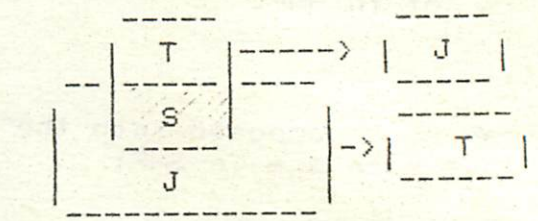
A RELATION is in BCNF iff every determinant is a candidate key. The reason for introducing BCNF is that the original 3NF definition does not handle the case of a RELATION possessing two or more overlapping candidate keys.

## Example 1 :

Consider the RELATION: SJT with ATTRIBUTEs S ( Student ), J ( subJect ), and T ( Teacher ). The meaning of an SJT Tuple is that the specified sTudent is taught specified subJect by specified Teacher.

RELATION: SJT

| S | J | T |
|---|---|---|
| Ayman | Data-Struct. | Prof. Amani |
| Ayman | Data-Base | Dr. Abdalla |
| Sherif | Data-Struct. | Prof. Amani |
| Sherif | Data-Base | Dr. Abdalla |




2 overlapping candidate keys

{ Fig 29 }

The FDs in SJT are :

S , J ----> T

S , T ----> J

We have two overlapping candidate keys: S, J, and S, T.

The RELATION is 3NF but not BCNF; and the RELATION suffers from certain ANOMALies. E.g., we cannot delete the information that ayman is studying Data-Structures without at the same time losing the information that Prof. Amani teaches Data-Structures. The difficulties are caused by the fact that T is a determinant but *not a candidate key.*

Again, we can get over the problem by DECOMPOSING SJT into two BCNF RELATIONs: ST ( S , T ) and TJ ( T , J ).

## RELATIONal Languages :

It is highly desirable that the Users who cannot program should be able to QUERY the Data Bases and extract from them the information they need. For this purpose, a wide variety of Data Base QUERY Languages exist. Some are simple ( the one mentioned here ) so that an unskilled User can compose a QUERY. Some can carry out more complex searching.

## Structured QUERY Language ( SQL ) :

The basic syntax takes the form:

SELECT < ATTRIBUTE(s) >

FROM < RELATION >

WHERE < condition >

The simplest form of the <condition> appears as follows :

ATTRIBUTE rel.op. Value

The rel.op. is one of: = , NEQ , > = , <= , > , < .

The output is a set of values . The values are chosen by selecting each RELATION-row that satisfies the <condition>.

Examples :
---

1. Given :

RELATION: GDEPTS

| GDEPT-ID | PHONE | NO-TRAINS | LOCATED-IN-CITY |
|---|---|---|---|
| GD-A | 667 932 | 200 | Ismailia |
| GD-B | 725 172 | 310 | Port-said |
| GD-C | 636 182 | 75 | Port-said |
| GD-D | 679 305 | 105 | Menouf |

( Fig 30 )

Then,

```
SELECT  GDEPT-ID
FROM    GDEPTS
```

will output the GDEPT-ID of all TUPLEs in the RELATION: GDEPTS.

It is possible to output more than one ATTRIBUTE by including all the needed ATTRIBUTEs after SELECT.

Thus :

```
SELECT  GDEPT-ID , PHONE
FROM    GDEPTS
```

would output the GDEPT-ID together with its PHONE number.

If all GDEPTS details are required, we can use :

```
SELECT  *
FROM    GDEPTS
```

This PROCEDURE will output all the ATTRIBUTE-values of the RELATION: GDEPTS.

If only Stores that satisfy some Condition are needed , the mapping is extended to include a condition clause .

Thus :

```
SELECT  GDEPT-ID
  FROM  GDEPTS
 WHERE  LOCATED-IN-CITY = 'Port-said'
```

would result in the output: GD-B and GD-C

2. Given :

RELATION: ITEMS

| ITEM-ID | DESCR | SIZE | WEIGHT |
|---|---|---|---|
| I1 | DOOR | SMALL | 9 |
| I2 | LAMP | LARGE | 12 |
| I3 | WINDOW | LARGE | 15 |
| I4 | WINDOW | SMALL | 12 |

( Fig 31 )

Then ,

```
SELECT   ITEM-ID
  FROM   ITEMS
 WHERE   SIZE NEQ 'SMALL'
```

would result in the output: 12 and 13.

It is possible to specify compound Conditions in SQL using the logical operators: AND and OR as follows:

```
SELECT   ITEM-ID
  FROM   ITEMS
 WHERE   SIZE = 'SMALL' AND WEIGHT > 10
```

The output is: I4.

There can be any number of such conditions separated by AND or OR in mapping.

Optional features are available in SQL to ORDER the output values and to allow or suppress ( prevent ) the duplicate appearance of the same value.

Example :
---

Given :

RELATION: HOLD

| GDEPT-ID | CLASS-CODE | NOS-HELD |
|---|---|---|
| GD-A | C2 | 300 |
| GD-A | C1 | 200 |
| GD-A | C3 | 400 |
| GD-D | C3 | 150 |
| GD-B | C1 | 300 |
| GD-B | C2 | 400 |
| GD-A | C4 | 150 |
| GD-C | C3 | 400 |
| GD-D | C4 | 100 |

( Fig 32 )

Then ,

```
SELECT   GDEPT-ID
  FROM   HOLD
 WHERE   CLASS-CODE = 'C4'
 ORDER BY  GDEPT-ID  ASCENDING
```

would output all the GDEPT-IDs of GDEPTS that hold C4 in ascending order.

To suppress ( prevent ) the duplicate appearance of a value, the word UNIQUE is included in the SELECT statement.

Example :
---

```
SELECT UNIQUE  ITEM-ID
  FROM  HOLD
 WHERE  NOS-HOLD > 250
```

would insure that each CLASS-CODE that satisfies the condition appears only once in the output.

If UNIQUE were not included, the C3 would appear twice in the output because its holding in GDEPTS: GD-A and GD-B exceeds 250.

Nested Mapping in SQL ( Retrievals with Multiple Levels of Nesting ) :

Examples :

```
SELECT  CLASS-CODE
  FROM  HOLD
 WHERE  GDEPT-ID  IN  SELECT  GDEPT-ID
                        FROM  GDEPTS
                       WHERE  LOCATED-IN-CITY = 'Port-said'
```

results in the output: C1, C2, C3 ---these are the numbers registered by Port-said GDEPTS.

This nested expression is the same as :

```
SELECT  CLASS-CODE
  FROM  HOLD
 WHERE  GDEPT-ID  IN  ( the set of Port-said GDEPTS )
```

where the Set in brackets is found by the mapping :

```
SELECT  GDEPT-ID
  FROM  GDEPTS
 WHERE  LOCATED-IN-CITY = 'Port-said'
```

Experience with using SQL has led to the development of various alternatives for stating QUERIes that include more than one RELATION.

We will consider here 3 such alternatives , one involving the expression of RELATIONal JOINS , and the other two using the terms: EXISTS or ANY .

## Expressing RELATIONal JOINs :

The RELATIONs to be JOINed are specified in the FROM Clause , whereas the JOIN Condition is in the WHERE Clause .

Thus , the previous QUERY becomes :

```
SELECT  CLASS-CODE
  FROM  HOLD , GDEPTS
 WHERE  HOLD.GDEPT-ID = GDEPTS.GDEPT-ID
   AND  GDEPTS.LOCATED-IN-CITY = 'Port-said'
```

Now , ATTRIBUTEs from more than one RELATION can appear in the WHERE clause.

RELATION names can precede the ATTRIBUTE names in the WHERE Clause to resolve any naming ambiguities.

## Retrievals Using EXISTS :

The word EXISTS in SQL Statement can be explained as follows:

Consider the QUERY :

```
SELECT  CLASS-CODE
  FROM  HOLD
 WHERE  EXISTS
        ( SELECT  *
            FROM  GDEPTS
           WHERE  GDEPT-ID = HOLD.GDEPT-ID
             AND  LOCATED-IN-CITY = 'Port-said' )
```

The expression in parentheses evaluates to TRUE for a given HOLD TUPLE if there EXISTS a GDEPTS TUPLE that has a GDEPT-ID value equaling the value GDEPT-ID in the HOLD TUPLE and also has a LOCATED-IN-CITY value of 'Port-said'.

If the expression in brackets evaluates to TRUE for a given HOLD TUPLE the CLASS-CODE value of that HOLD TUPLE is output.

Retrieval Using ANY :
---

The previous QUERY is expressed as :

```
SELECT  CLASS-CODE
  FROM  HOLD
 WHERE  GDEPT-ID = ANY ( SELECT GDEPT-ID
                           FROM  GDEPTS
                          WHERE  LOCATED-IN-CITY = 'Port-said' )
```

Here , a HOLD-TUPLE is SELECTed if its GDEPT-ID value is ANY of the values output defined by the mapping in parentheses .

Note:
---

The term = ANY is equivalent to IN in a nested mapping .

The term ANY can be used together with the relational operators : < , > , NOT > , NOT < , = > , or =< .

Example :
---

```
SELECT  ORDER-NO
  FROM  ITEMS-ORDERED
 WHERE  CLASS-CODE = 'C2'
   AND  QTY-ORDERED < ANY ( SELECT  NOS-HELD
                              FROM  HOLD
                             WHERE  CLASS-CODE = 'C2' )
```

Here , ORDER-NOs that contain item C2 are examined . The ORDER-NO is output if the QTY-ORDERED is < some NOS-HELD of the holdings of the item in one GDEPT. The order can then be satisfied by withdrawals from one GDEPT.

Set Exclusion :
---

Example :
---

```
SELECT  ITEM-ID
  FROM  HOLD
 WHERE  GDEPT-ID  NOT IN  SELECT  GDEPT-ID
                             FROM  GDEPTS
                            WHERE  LOCATED-IN-CITY = 'Port-said'
```

Here , items carried by stores NOT IN Port-said are output.
---

Compound Conditions :
---

Example :
---

```
SELECT  CLASS-CODE
  FROM  HOLD
 WHERE  GDEPT-ID  IN  ( SELECT  GDEPT-ID
                          FROM  GDEPTS
                         WHERE  LOCATED-IN-CITY = 'Port-said' )
   AND  NOS-HELD > 20
```

Here , AND NOS-HELD > 20 is not within the brackets. It is therefore considered to be in the mapping that includes HOLD.

If the brackets were not included , this AND Clause would be considered to be in mapping that includes STORES; in this case, a diagnostic would be output because NOS-HELD is not an ATTRIBUTE of GDEPTS.

FUNCTIONs :
---

FUNCTIONs allow the User to specify AVG, MAX, MIN, and COUNT of a SELECTed set of ATTRIBUTEs. Thus

```
SELECT  AVG ( NOS-HELD )
  FROM  HOLD
 WHERE  GDEPT-ID = 'GD-A'
```

output the Average amount of each part held by GDEPT GD-A.

Grouping and Partitioning :
---

A RELATION may be partitioned into groups according to the values of some ATTRIBUTE and then a function applied to each group.

Example :
---

```
        SELECT  GDEPT-ID
          FROM  HOLD
      GROUP BY  GDEPT-ID
    HAVING SUM ( NOS-HELD ) > 500
```

will output: GD-A, GD-B.

Here, all rows are grouped by GDEPT-ID, and the ATTRIBUTE columns NOS-HELD are SUMmed for each GROUP.

Thus for GD-A, the SUM is 1050, for GD-B is 700, for GD-C is 400, and for GD-D is 250.

Only GD-A and GD-B satisfy the conditions.

## INP Data Base Management System :

This chapter will be devoted to a custom software system for managing a single Data Base. We will be working with an INP Personnel Data Base as an example, but the techniques discussed here can be used to manage any single data base.

In developing this very simple project, the following items will be discussed:

- The basics of creating User-friendly "Menu-driven" System,
- Creating and using Custom Screens and Reports,
- Using Indexed-Files for maximum speed, and
- Basic programming techniques universal to all business applications.

## Introduction :

Any software project, large or small, can be broken down into a series of steps, starting with a basic idea and ending with a finished product. Usually there are 6 steps in developing a software project:

1. Defining the goal of the project and the user-level ( Problem Definition ),
2. Specifying the input and output of the project ( Input/Output Specification ),
3. Designing the data base structure of the project ( Data Base Design ),
4. Isolating specific program functions ( Modular Program Design ),
5. Writing the individual programs ( Module Development ), and
6. Testing and making corrections ( Testing and Debugging ).

## 1. Problem Definition :

Problem definition for a software project usually starts out as a vague description of the actual problem, such as;

" Create an INP Personnel Directory Management System "( referre to as INP System in what follows ).

The experience-level of the end-User is a key element in system design, so a refinement is made in problem definition to include it. For example,

" Create an INP Management System that can be used by an Individual with little or no computer experience"

Defining the problem more specifically helps to think in terms of specific tasks that a DBMS can perform; For example, with the basic dBASE capabilities in mind, the task can be defined more specifically as follows:

Create an INP Management System that allows an inexperienced User to do the following:

1. Add new names and addresses to a data base,
2. Arrange the data in alphabetical order by last name for displaying a specific RECORD.
3. View selected data by title, position, first name, last name, or all.
4. Print phone directory grouped by title, and ordered by last name.
5. Make changes to the data base, and delete RECORDs as necessary.
6. Maintain the data by backing-up the data base.

. .

.

.

Now, the large task of creating an INP System is clearly specified and , more importantly , broken into smaller tasks that are relatively easy to accomplish.

More specific, the above problem definition can be restated ( problem definition refinement ) as follows:

Create an INP Management System that allows an unexperienced User to do the following:

1. Add new names and addresses to a data base file.

   - use a custom screen, including instructions for controlling the cursor rather than the usual APPEND screen.

2. Arrange the data into alphabetical order by last name for displaying a specific RECORD.

3. View selected data by title, position, first name, last name, or all.

4. Print phone directory grouped by title, and ordered by last name.

5. Make changes to the data base and delete RECORDs as necessary.

   - Look up information to edit or delete based on name rather than RECORD number.

6. Maintain the data by backing-up the data base.

## 2. Input/Output Specifications :

### Output Specification:

We want the computer to produce 3 items :

1. Display a RECORD,
2. View selected data by title, position, first name, last name, or all.
3. Printed phone directory.

### Output Refinement:

1. Display a RECORD contains:
   - Serial number,
   - Name,
   - Title,
   - Position,
   - Address,
   - Home phone,
   - Work phone,
   - Degree,
   - Date awarded,
   - Major field,
   - Birth date,
   - Birth place, and
   - Notes.

2. View selected data RECORDs:

   i, by title and contains name and position.

   ii, by position and contains name and title.

   iii, by first letter or last name and contains name , title, home phone, and work phone.

   iv, by last name and contains name, title, and home phone.

   v, all data base contains name, title, home phone, work phone, and position.

Input Specifications:
---

Defining the output really determines the input, so the input must be:

- Serial number,
- Name,
- Title,
- Position,
- Address,
- Home phone,
- Work phone,
- Degree,
- Date awarded,
- Major field,
- Birth date,
- Birth place, and
- Notes.

## 3. Data Base Design :

Now that we have decided what information needs to be stored, we can begin designing the data base. We need to think about the type of data in each field and how much space the data require.
We can break out the fields in INP Data Base File as follows:

| Field | name | Type | Width | Dec | |
|---|---|---|---|---|---|
| 1 | SENO | N | 4 | 0 | --> Serial Number |
| 2 | FIRST_NAME | C | 15 | | |
| 3 | LAST_NAME | C | 20 | | |
| 4 | TITLE | C | 5 | | |
| 5 | POSITION | C | 30 | | |
| 6 | ADDRESS | C | 30 | | |
| 7 | CITY | C | 15 | | |
| 8 | HPHONE | C | 8 | | --> Home phone |
| 9 | WPHONE | C | 4 | | --> Work phone |
| 10 | DEGREE | C | 3 | | |
| 11 | AWADATE | D | 8 | | --> Date awarded |
| 12 | MAJFLD | C | 15 | | --> Major field |
| 13 | BDATE | D | 8 | | --> Birth date |
| 14 | BPLACE | C | 15 | | --> Birth place |
| 15 | NOTES | M | 10 | | |

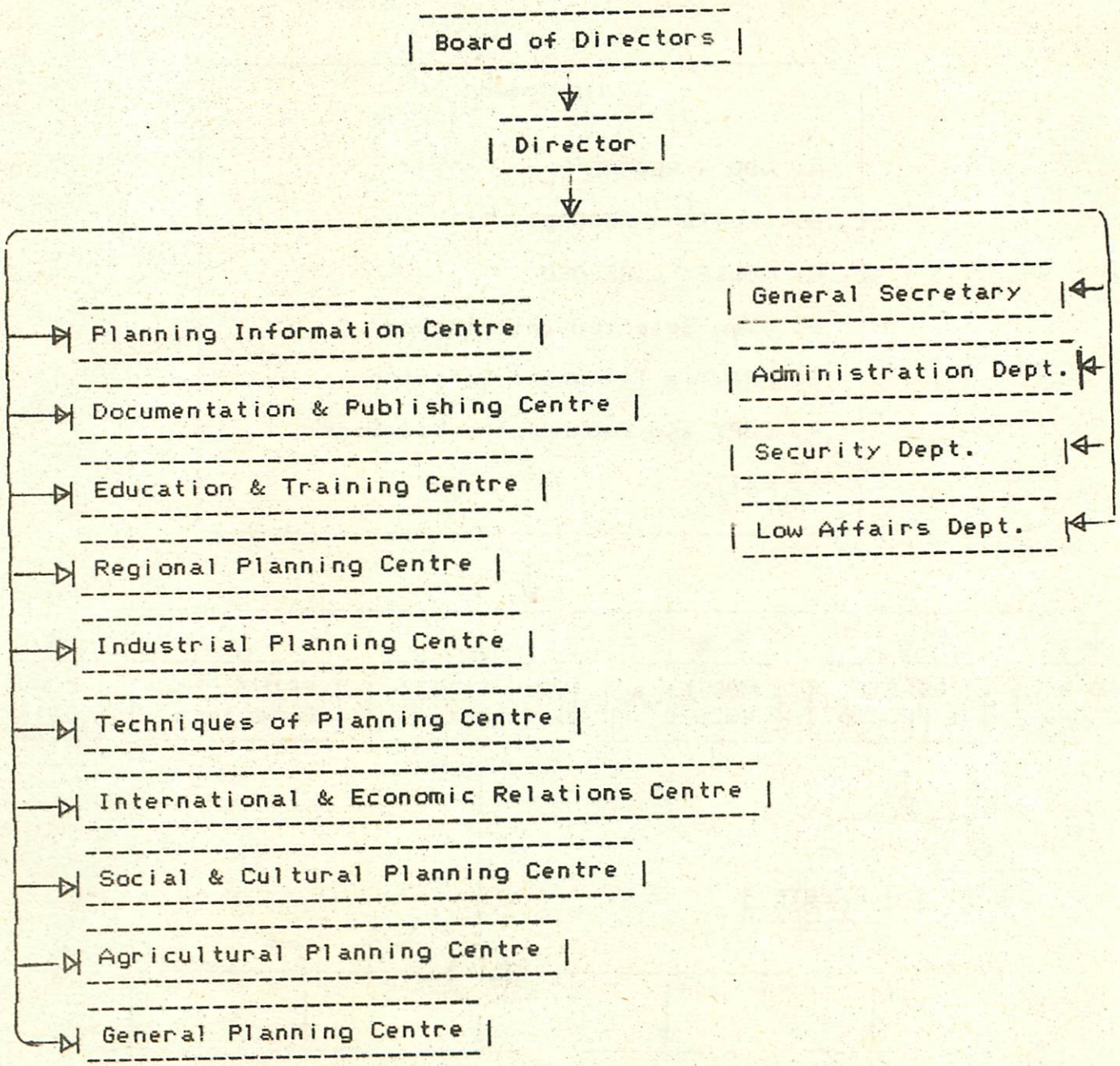
Now, we are ready to create the Data Base File.

## 4. Modular Program Design :

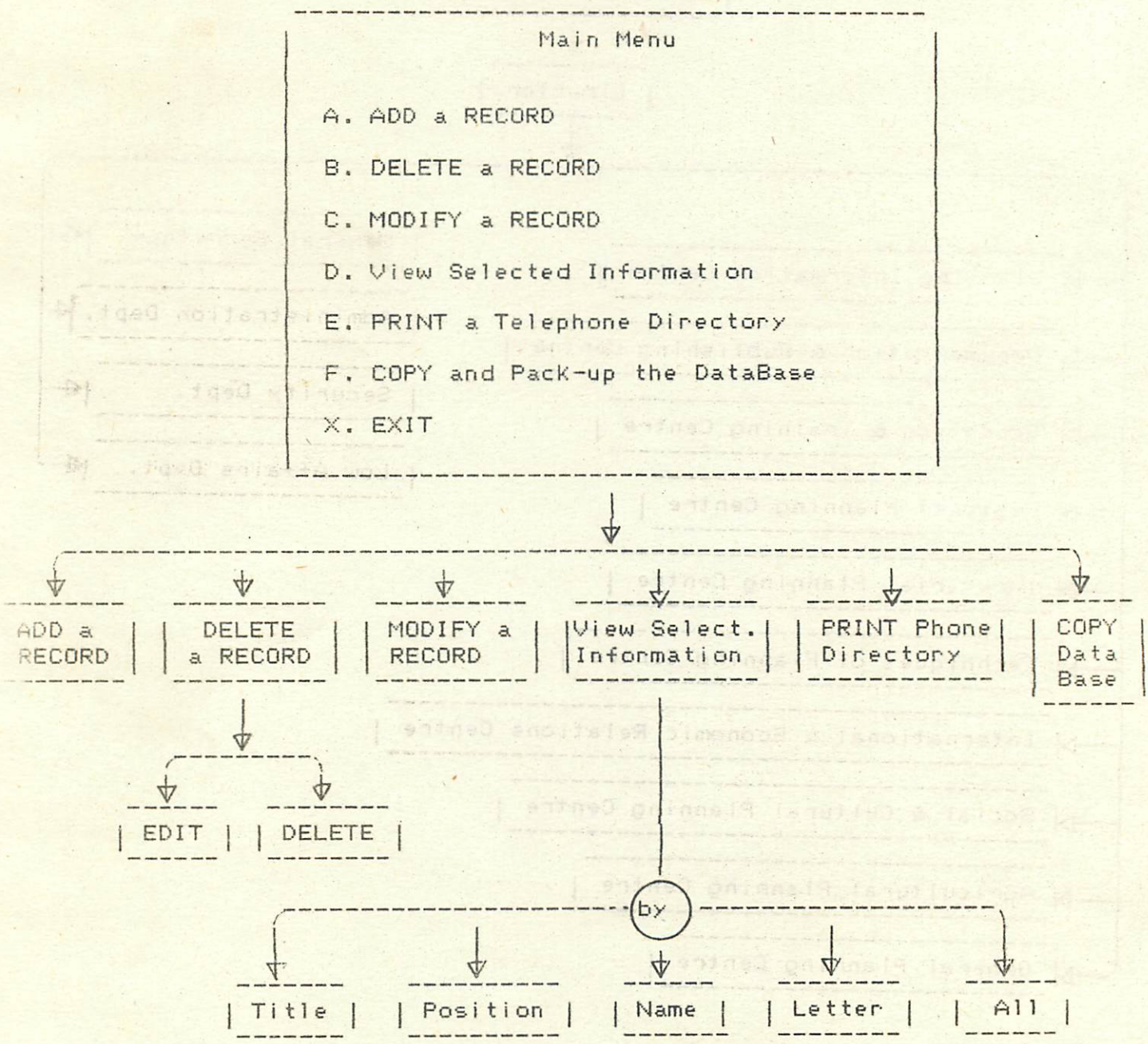
To make the INP System easy to use, we will have the computer display a Menu of options. All the User will have to do is select an option, and the program will take over from there. So rather than to write one very large program to handle all these tasks, we will create a separate, smaller program for each individual task. This modular design will make the programming task much simpler and the program logic much easier to follow.

The following illustrates the Organizational and Functional Structures of the INP and structure charts that allows us to see how the various parts of the INP System are related.

Organizational & Functional Structure of INP :



For each centre of the above structures, we can have :



## PERSPECTIVE :

### Developments in Information Systems :

The use of computers is extending into a wide variety of applications. A major area of development is in use of computers by organizations to support management and decision making.

An organization usually has transactions that must be processed in order to carry out its day-to-day activities. The payroll must be prepared, sales and payments on accounts must be posted, research projects must be followed up, ... etc.

A Management Information System ( MIS ) is an integrated, man/machine system for providing information to support the operations, management, and decision making functions in the organization.

The MIS utilizes computer hardware and software , manual procedures, management and decision models, and a data base.

When an MIS is in use, there is an increased availability of current, accurate information for all levels of personnel. Reports, responses to information requests, analysis, planning, and decision making receive improved processing and information support.

The implementation of such systems is one of the most challenging areas in the application of computers. The development and refinement of systems based on the MIS concept will continue to be a major objective of organizations using computers.

### Impact of Computers on Organizations and Individuals :

Computers are already present at most levels of organizations. An individual in an organization making extensive use of computers cannot remain unaffected by changes in information processing systems. A significant change expected in such organizations is in organizational structure. The effect on the individual is in such areas as job content, reduction in allowable individual behaviour, and privacy.

Impact on the Structure of Organization :

Most attention has been concentrated on the middle-management level. The reasoning is that computers provide top management with a greater span of control and that many of the judgmental decisions usually handled by middle management will be programmed into the computer. Therefore, less middle management will be required, and its power and status will decline.

However, one might argue that most of the time of middle-management personnel is spent on tasks which are motivational or which require unstructured decision making or judgmental reasoning. Therefore, although some decisions may be taken from them and central planning may further restrict their range of decision making, all other middle-management will remain. The problem of managing people are still critical and are not computerized. Furthermore, the frequency of change may take it more difficult than was thought to routinize many middle-management decisions.

The question of whether complex computer systems will cause organizations to centralize operations that have been decentralized is not yet answered. In a decentralized organization, planning and decision making at the individual plant or other facility are delegated to the local manager. In a centralized organization, all important decisions are made by a central staff at the head quarters. It is clear that some individual decisions are being centralized---an example is centralized inventory control, in which a computer system keeping track of all inventory at all locations is usually able to make better decisions than the individual managers. On the other hand, the economic and managerial considerations which have prompted some organizations to decentralize are still strong. Examples are the need for local response to changing conditions and enhanced motivation.

On balance, however, it seems quite clear that the computer has already had some impact on organization and management and will have greater impact as more complex computer-based information and control systems are implemented. It is likely that the most successful systems will be ones that divide tasks between the worker and computer ( a man-machine system ) rather than fully automated systems. In a task division system, the decisions by computer are strucured, routine decisions, the decisions made by the worker are the unstructured, difficult ones. And even computer decisions will need to be monitored. It would take an executive with an excess of faith in computer programs to let important decisions be handled automatically by a computer without human review.

## References


. Ashton - Tate ,
Learning , Using , and Programming with dBASE III PLUS

. C. J. Date , IBM Corporation ,
An introduction to DataBase Systems , 3rd Edition ,
Addison - Wesley Publishing Company ,
Roading , Massachusetts , Amsterdam , London , Manila ,
Singapore , Sydney , Tokyo .

. I. T. Hawryszkiewycz , Canberra College of Advanced Education ,
Australia .
DataBase Analysis and Design
Science Research Associates , Inc.
Chicago , Henley-on-Thames , Toronto
A subsidiary of IBM .

. James Martin , IBM Systems Research Institute ,
Principles of DataBase Management
Prentice-Hall , Inc.
Englewood Cliffs ,New Jersey .

. James Martin , IBM Systems Research Institute ,
Computer DATABASE Organization , second edition
Prentice-Hall , Inc.
Englewood Cliffs ,New Jersey .

. Gordon B. Davis, Prof., Management Information Systems,
Univ. of Minnesota,
Computers and Information Processing
McGraw-Hill Kogakusha, Ltd.
Tokyo, Beirut, Paris,....

. Abdalla El-daoushy, Institute of National Planning ,
A First Course in DataBase Analysis and Programming with
dBASE III Plus , Memo no. 880, April, 1988.

مطبعة معهد التخطيط القومى
القاهرة