



CONTENTS

<u>Topic:</u>	<u>Page Number:</u>
CHAPTER 1: PREFACE	1
CHAPTER 2: INTRODUCTION	2
2.1- What is logical database design	2
CHAPTER 3: DATAMODELS	5
3.1- Conceptual Data Models	5
3.2- Logical Data Models	6
3.3- Physical Data Models	7
CHAPTER 4: LOGICAL DATABASE DESIGN	22
4.1- Entity Attributes Vs Object-Relation Approach	23
4.2- Rules to guide Database design	25
CHAPTER 5: FOX PRO & dBASE	31
CHAPTER 6: HOSPITAL DESIGN DEFINITION	35
CHAPTER 7: CONCLUSION	55
CHAPTER 8: APPENDIX	56
8.A- Flowcharts	56
8.B- Model Diagram	67
8.C- Selected Codes	68

ACKNOWLEDGMENTS

First thank *Allah* for what ever he gives, secondly I would like to dedicate my thesis to my dad, mom, family back home, and all my friends whom sat behind me during the hard times, and special thanks to Miss: Besime ERIN my project supervisor who encouraged me to finish this thesis and provided me with all the material, finally I appreciate all the help from the professors and teachers in Near East University through out the years.

Samir ZAIDAN

Jan 26th 2001

PREFACE

The purpose of this thesis is to define the Near East Hospital Information system Database. As all of the data described in the database are derived from data captured by project.

It is useful to review the Data Element Dictionary. This document contains definitions and file description for each of the data elements to be collected as part of the project. In addition, a high level overview of the design of the system, and structure of the various records and fields to be submitted to hospital system are provided.

Purpose of this Thesis

This project defines the Near East Hospital Information System database files.

Audience for this Thesis

The intended audience for this project includes the follow:

- (1) Codes - any codes that are responsible for creating and maintaining the data elements and file description specified in this project.
- (2) Screens - those individuals who wish to view the data collected and processed as part of the Hospital Project from a "summary" or "subtotaled" point of view. The database files are used by the hospital Reception staff to verify that the underlying unitary data reported as part of the project are valid and consistent from term-to-term and year-to-year.

Scope of this thesis

The central role of this project is to provide information concerning the hospital Information System Database files. This document provides an overview of the database design as well as detailed specifications for each of the database files and data elements that comprise them.

Also, This document is both intended, to give a detailed explanation of the sources of the data used to create the files, and a detailed explanation as to how the files can be used. The source for all files is the data collected and processed as part of the Hospital Information System project.

Introduction

The success of a database is completely dependent on the logical database design. Even if we buy expensive and fast hardware and software, the quality of the database design will dictate whether a project will succeed. In a way, it is the Achilles heel of a project.

A good database design does the following:

1. Provides minimum search time when locating specific record.
2. Stores data in the most efficient manner possible to keep the database from growing too large.
3. Makes data updates as easy as possible.
4. Is flexible enough to allow inclusion of new functions required of the program.

The database design process can be divided into six steps:

1. Requirement analysis.
2. Conceptual database design.
3. Logical database design.
4. Schema refinements.
5. Physical database design.
6. Security database design.

What is logical database design?

It is the phase in the system development life cycle concerned with constructing tables and their columns. During this phase decisions are made about which piece of data should be stored and how those pieces should be arranged logically in tables. This phase precedes physical database design where the emphasis is on how the data is really stored on disk. Physical database design deals with issues such as storage and performance.

This one-day workshop discusses all the aspects of logical database design. Structured techniques for developing a logical design are discussed. Additionally, many guidelines, tips, and tricks are given. And logical database design is looked at

from a transactional and from a data warehouse environment. Because the use of data warehouse is different, different rules apply.

Two opposing techniques exist to perform logical database design. The bottom-up approach, which is based on normalization, is the oldest and most well known one. One starts with placing all columns in one wide table and then these tables are decomposed into more well structured tables. The decomposition is based on rules called the normal forms. The second technique, the top-down approach, uses as a starting point information models where semantic and object-oriented concepts, such as subtypes and aggregates, are used. These concepts are translated into tables and columns using an algorithm. Both approaches have their advantages and disadvantages and are, therefore, discussed thoroughly.

Although some of the rules described in this workshop do stem from relational theory, such as the normal forms, the emphasis will be on practical issues. The workshop is a culmination of many years of experience of designing large operational databases and data warehouses.

The aim of logical design is to construct a logical schema that correctly and efficiently represents all of the information described by an entity relationship schema produced during the conceptual design phase. This is not just a simple translation from one model to another for two reasons, first, there is not a close correspondence between the models involved because not all the constructs of the entity-relationship model can be translated naturally into the relational model. For example, while an entity can easily be represented by a relation. There are various options for the generalizations. Secondly, the aim of conceptual design is to represent the data accurately and naturally from a high-level, computer-independent point of view. Logical design is instead the basis for the actual implementation of the application, and must take into account, as far as possible, the performance of the final product. The schema must therefore be constructed in such a way as to make the execution of the projected operations as efficient as possible. In sum, we must plan a task that is not only a translation (from the conceptual model to the logical) but also reorganization. Since the organization can for the most part be dealt with independently of the logical model, it is helpful to divide the logical design into two steps:

- Restructuring of the Entity-Relationship schema, which is independent of the chosen logical model and is based on criteria for the optimization of the schema and the simplification of the following step.
- Translation into the logical model, which refers to a specific logical model (In our case, the relational model) and can include a further optimization, based on the features of the logical model itself.

The input for the first step is the conceptual schema produced in the preceding phase and the estimated database load. In terms of the amount of data, and the operational requirements. The result obtained is a restricted Entity-Relationship schema.

DATA MODELS

Levels of abstraction usually categorize data models:

- Conceptual
- Logical
- Physical

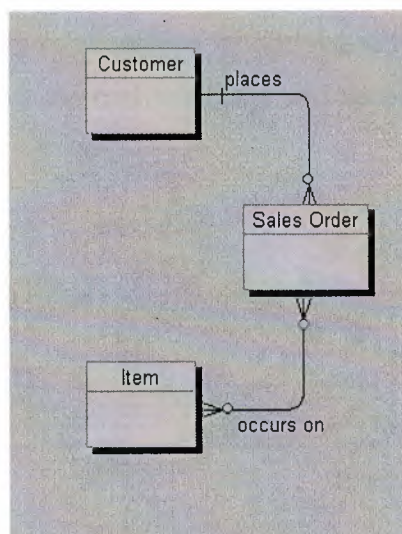
These have no agreed formal definitions. Professional data modelers understand the approximate scope of each.

Conceptual Data Model:

A conceptual data model shows data through business eyes.

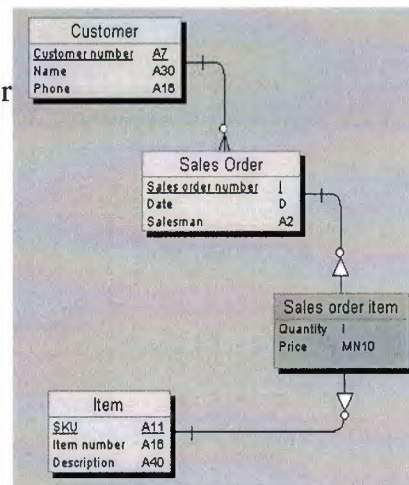
It suppresses technical details to emphasize:

- All entities, which have business meaning.
- Important relationships (including many-to-many).
- A few significant attributes in the entities.
- A few identifiers or candidate keys.



The Logical Data Model

- Is a generic relational schema (in at least 1NF) which -
- Replaces many-to-many relationships with associative entities
- Defines a full population of entity attributes
- May Use non-physical entities for domains and sub-types
- Establishes entity identifiers
- Has *no* specifics for any RDBMS or configuration



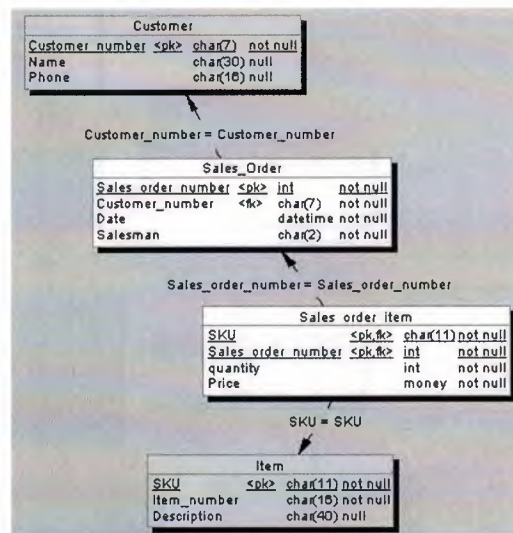
Propagation of foreign keys may be explicit or implied in a logical data model. As long as the resulting physical schema includes the necessary foreign key columns and joins, the representation of foreign keys in the logical model is a matter of convenience and taste.

Replacing many-to-many relationships with associative entities is necessary to model 1st normal form, support internal attributes and secondary relationships, and enable alternate identifiers.

Physical Data Model

A physical data model is a database design for:

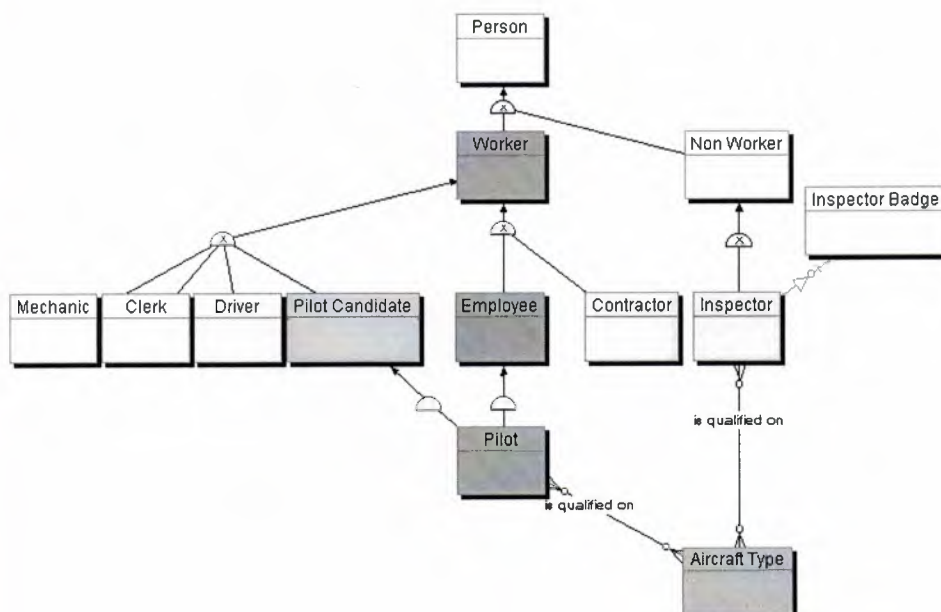
- One DBMS product
- One site configuration



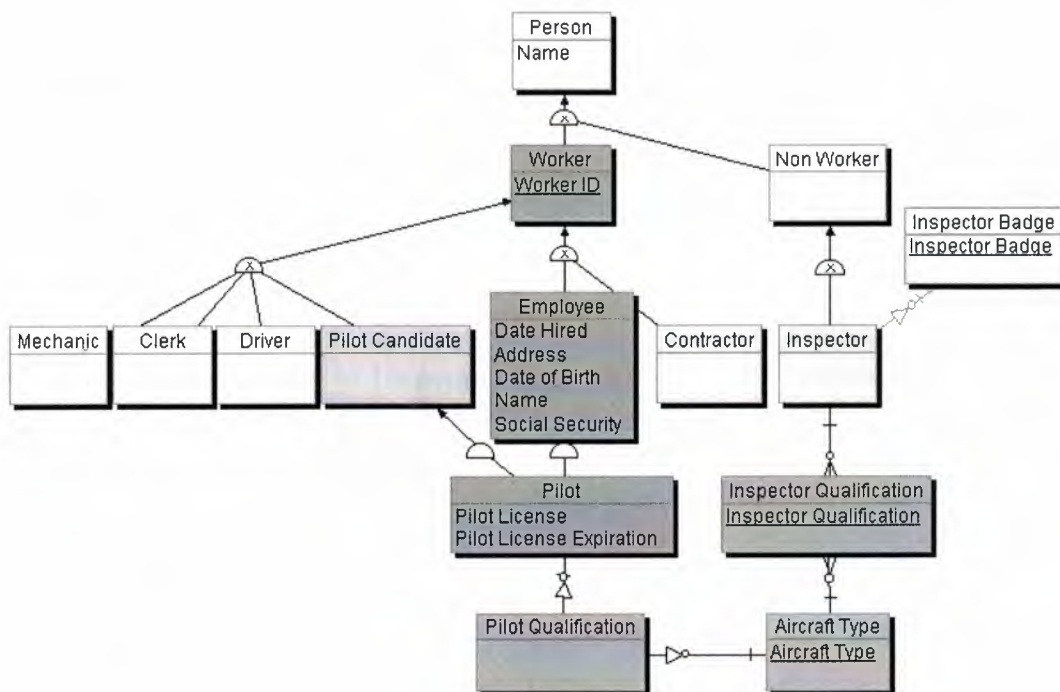
A physical data model may include:

- Referential integrity
- Indexes
- Views
- Alternate keys and other constraints
- Tablespaces and physical storage objects

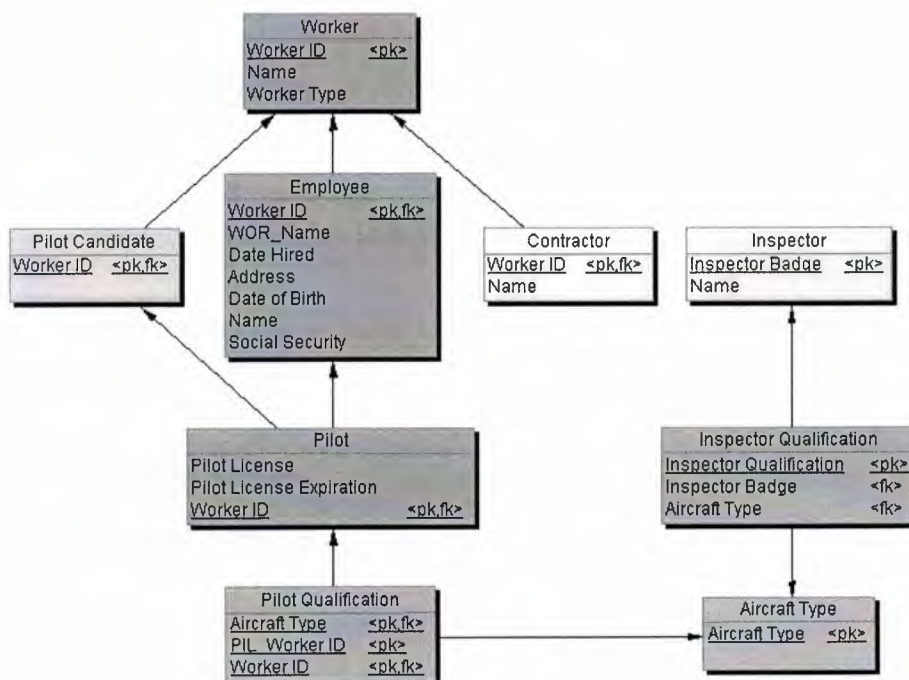
Conceptual Data Model - An Example:



Logical Data Model - Same Example:



Physical Data Model - Same Example:



What is an Entity?

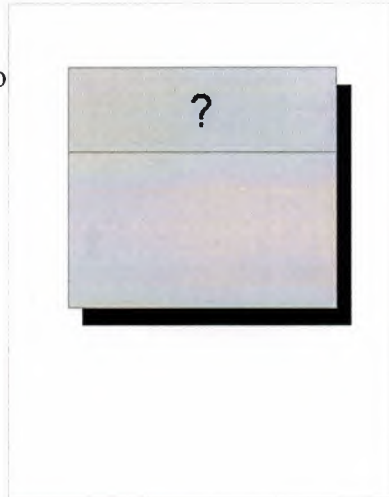
Person, place, or thing? (Too specific)

Any thing in which the business has an interest? (Too vague)

A synonym for a relational table? (Misses the point)

An information container in 1st normal form which:

- Records a fixed set of attributes
- Holds 0 to n occurrences
- Is a relational abstraction of some real-world concept
- May or may not map to a physical table in the database



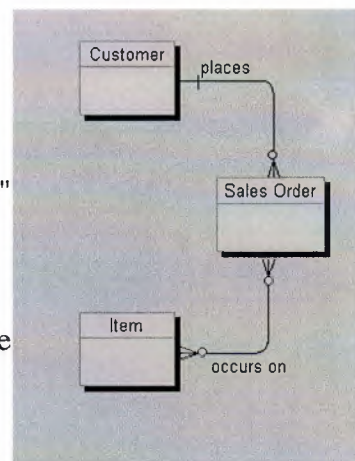
What is a Relationship?

A connection, association, or rule among entities:

"Customer places Sales Order"

"Item occurs on Sales Order"

In a conceptual model, it is sufficient to state or draw the relationship.



A logical model defines specific means of joining two entities via implied or expressed foreign keys.

Relationships can be classified into a few relationship types.

Foreign Keys (FK):

A foreign key is a function, not a fact!

A foreign key is the result of relationship and identity.

If the relationship changes, so does the foreign key.

State		
state name	state code	row id
CK	CK	CK
Alabama	AL	1
California	CA	2
New York	NY	3
Tennessee	TN	4

The "child" entity gets foreign key attributes to match identifier of its "parent" entity.

If the parent identifier changes, so does the child's foreign key.

↑

Disaster	
row id	disaster year
FK	
2	1979
2	1982
2	1995

the

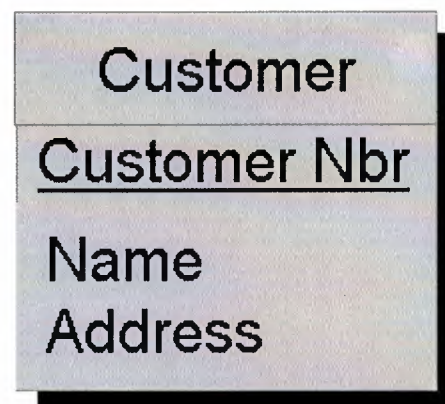
What is an Entity Identifier?

The identifier of an entity is some set of attributes whose combined value is unique for all instances of that entity.

Thus an entity's identifier is one of its (possibly several) candidate keys.

If an entity has more than one candidate key, the choice of one to be the identifier is an arbitrary convenience for RDBMS operation.

While an entity identifier is not absolutely mandatory, it is hard to think of a useful entity without one.



Evolving the Logical Model:

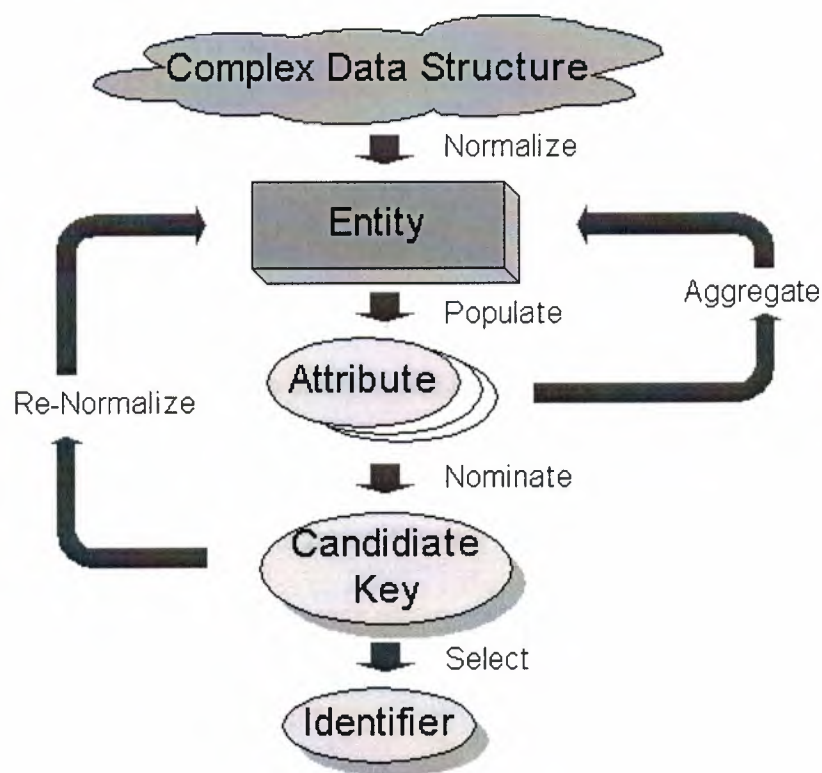
Normalize structures

Populate attributes

Aggregate data items into new entities

Nominate candidate keys

Re-Normalize on the new candidate keys



Normalization:

Normal forms are the property that we can use to evaluate the quality of relational database. We will see that when a relation does not satisfy a normal form, then it presents redundancies and produces undesirable behaviour during update

operations. This principle can be used to carry out quality analysis on relational databases and so constitutes a useful tool for database design. For the schemas that do not satisfy the normal form, we can provide a procedure called Normalization. Normalization allows the non-normalized schemas to be transformed into new schemas for which the satisfaction of a normal form is guaranteed.

Normalization theory constitutes a useful verification tool, which indicates amendments, it has been developed in the context of the relational model and for this reason, it provides analysis and design techniques for the outcome of logical design.

Normalization can also be used on the Entity-Relationship schemas and during the quality analysis of the conceptual design.

Normalizing to a Logical Model:

Every raindrop, every snowflake, every hailstone has a single speck of dust at the core.

Every logical entity has a single idea at its core.

The essence of normalization is one entity = one idea:

- A customer is a person or organization that buys from us.
- A service order holds one customer request for service.

Examine complex data structures for hidden entities in:

- Nouns - tangible or intangible
- Adjectives whose value is one of a known list ...
female | male; green | yellow | red; 6' | 8' | 10'
- Embedded ideas, which can exist on their own

Populating Attributes:

For each entity, ask, "What properties does this thing have - even if nothing else exists around it?"

- A person has age - even the last person on earth.
- A building has height - even if it is abandoned.
- A song has a key, even if it is unsung.

Ask of each property, "Does this entity have only one of these?"

- A person is of one age.
- A building is of one height.
- A song may be written in one key but sung in another!

An attribute is a property of an entity, which depends solely on its entity - nothing else
- and can have only one value at a time

Domains:

6 feet 1/4 inches + 51.6 years =?

A domain (in *relational* usage) is a set of values and operations that may be used to populate and operate on one or more columns.

The values may be specified by list or formula.

While is not yet any theoretical or practical way to limit the operations applied to a domain, this example shows the need.

Aggregating Entity Attributes:

Sometimes you can reveal entities by looking in the data dictionary for homeless attributes:

- "To whom does atomic weight belong?
And what about the year in which an element was discovered?"

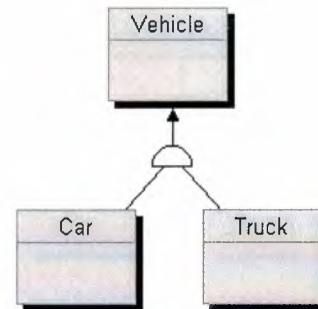
- Aggregate atomic weight and year into a new entity called Atomic Element.

Crosscheck data elements captured in the data dictionary from data flow diagrams, use cases, or other analysis.

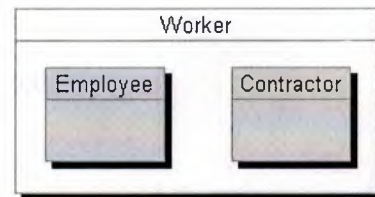
All data structures and data elements discovered in analysis must be accounted for in the logical model.

Looking for Hierarchies

Entities often occur in hierarchies - family trees related by inheritance.



This is sub-typing or specialization and generalization - the same as building OO class structures.



Each child entity inherits all attributes and relationships from its We define properties at their highest level in the hierarchy to avoid redundancy.

In a conceptual model, we ignore how inheritance operates.

Later we want to specify how super-and sub-types map from the logical model to physical structures parent.

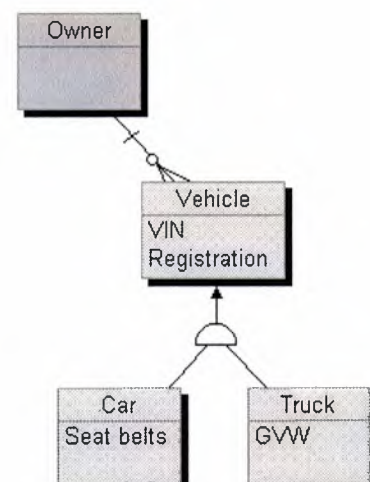
Generalization:

Hierarchies let us locate attributes and relationships at the appropriate level.

All vehicles have:

- VIN and Registration
- Owner

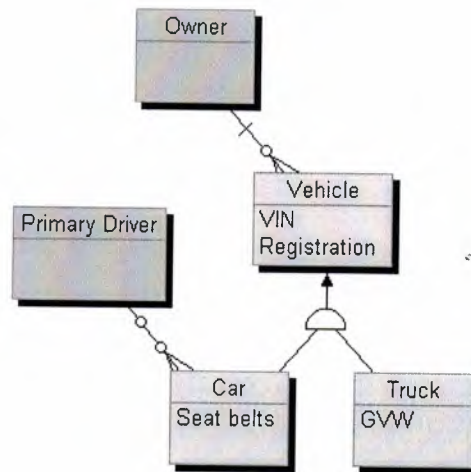
The attributes and relationship are *generalized* To all vehicles.



Specialization:

- Only cars have primary drivers and seat belts.
- Only trucks have gross vehicle weight.

These attributes and relationships are *specialized* to the child level.



Specialization of Relationships:

Optional parent relationships usually hide a need for specialization. Ask your self:

- Is the relationship *sometimes* true for *each* instance?

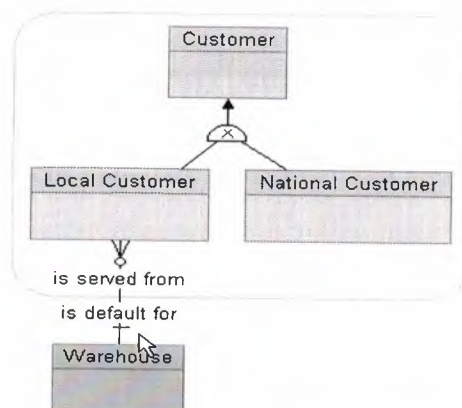
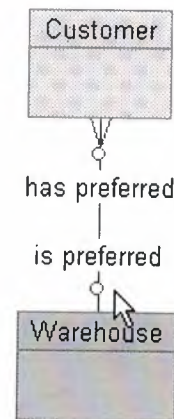
Then it is correctly modeled as optional.

- Is the relationship *always* true for *some* instances?

Then it requires some customers must have a default warehouse.

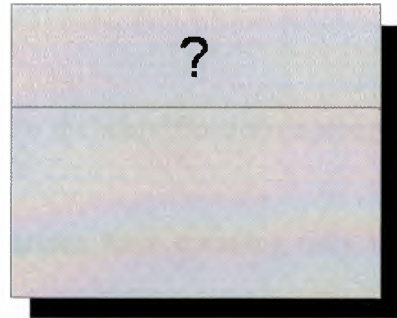
For others it does not exist (in this model).

By splitting the Customer entity into two sub-types, we can model the relationship to Warehouse precisely.



Populating a Conceptual Data Model:

- Diagram entities
- Diagram relationships
- Look for hierarchies:
 - Sub-types / super-types
 - Specialization /generalization
 - Class hierarchies



Diagramming Entities:

The entity symbol is a rectangle with the name at the top.

The entity name must be descriptive and meaningful. An unambiguous text definition is important.

The entity graphic symbol may differ slightly by CASE tool or author but the shape is unimportant.

The entity name is how people will refer to the entity. In the conceptual model an entity name should not be limited by RDBMS product limits - this is a generic name for the business. When physical DBMS object names are assigned, be sure to take into account the naming limits and reserved words of your target.

For example, what is a customer?

- Someone who has purchased?
- Any organization or person who may purchase?

Will this entity definition be clear two years later to a new team?

An Entity represents

- A tangible thing, a real-world event, or any intangible concept: "Product", "Sales visit", "Customer class discount"
- A class of things, not any one instance. "Person" has instances of "Tom" and "Simone".

Entities are *not* –

- Independent or Dependent. Those terms apply only to the identification choice you make.
- Fundamental, Attributive, or Associative. Classifications have meaning only in a model context of entities *and* relationships.

Diagramming Relationships:

The relationship symbol is a line between two entities. Define a relationship with:

- Predicate statements in one or both directions
- Unambiguous text description
- (A name is not important)
- cordiality symbols at each end

Will the relationship be clear two years later to a new team?

Relationships are -

- Unambiguous, immutable expressions of business rules.
- Binary or unary in IE, SSADM, IDEF1X and OO methods.
- Logical objects. Relationships can be reattached, with their properties intact, to different entities.

Relationships are *not* –

- Identifying or Non-Identifying. Those apply only to entity identification.
- Information containers. If you sense a need for information about a "relationship" then it is an entity!
- DBMS objects. Relationships only define joins between entities.

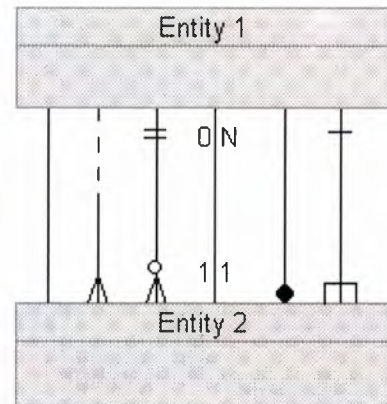
Relationship Notations:

There are many notation styles for relationships.

There are no standards for relationship style.

Different styles are read in different directions

But they all express the same information!



Relationship Cardinality:

Cardinality specifies the number of *instances* which may be involved in each entity of a relationship.

Most methods show the *Boolean abstract*, not the absolute number,

because this determines the relationship type...

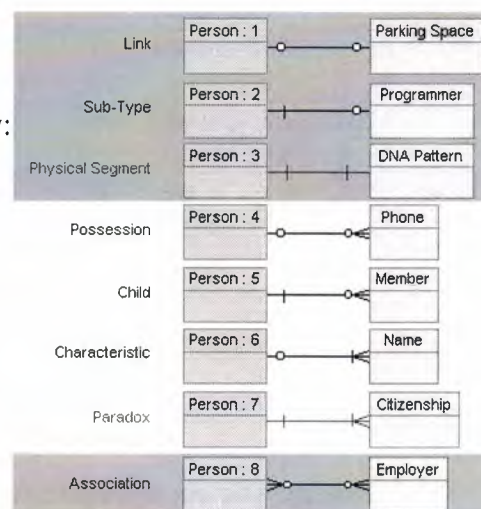
		minimum of	maximum of	
Customer	places	0	n	Sales Order
Sales Order	is placed by	1	1	Customer
		at least 1	possibly n	
Customer	places	No	Yes	Sales Order
Sales Order	is placed by	Yes	No	Customer

Relationship Types:

Relationships are grouped by their cardinality:

- One-to-Many is the *only* relational form.
- >99% of logical model

Relationship Types:



One-to-One is a special case of One-to-Many;

<1% of a logical model

Populating Attributes:

For each entity, ask "What properties does this thing have - even if nothing else exists around it?"

- A person has age - even the last person on earth.
- A building has height - even if it is abandoned.
- A song has a key, even if it is unsung.

Ask of each property, "Does this entity have only one of these?"

- A person is of one age.
- A building is of one height.
- A song may be written in one key but sung in another!

An attribute is a property of an entity which depends solely on its entity - nothing else - and can have only one value at a time.

What are Candidate Keys?

A candidate key is any set of one table's columns whose combined value is unique throughout that table.

- In the U.S. each state has a unique code - one candidate key.
- Each state name is also unique - another candidate key.
- And so is the order of admission to the union.

Code	Name	Admission
NV	Nevada	36
TN	Tennessee	16
MO	Missouri	24
PA	Pennsylvania	1
HI	Hawaii	50
IN	Indiana	19

Since both code and name are unique, code and name together are also unique.

That's another candidate key - seven with all the combinations.

Why are Candidate Keys?

- As a *candidate* for selection as the one identifier or *primary* key. A candidate key usually holds the core idea inside an entity:
- This state table is *about* states, which are known by their names.

A candidate key *always* expresses a business rule of uniqueness:

- Every state has a unique state code for mailing.

A table or entity with no candidate key is probably not normalized, and almost certainly not useful in an information system.

Testing Candidate Keys

A candidate key is unique. Is that enough?

Social security number is unique. Were you born with one?

A candidate key's value must exist. It cannot be null.

Your driver's license number is unique. Can it change?

The value of a candidate key must be stable. Its value cannot change outside the control of the system.

The value of a candidate key is unique, extant, and stable.

Re-Normalize on the Candidate Key:

After at least one candidate key has been noted, every attribute and relationship of the entity must be tested -

- Does this property depend solely and completely on the candidate key?

If not, move the property (normalize it) to the entity where it depends solely and completely on the candidate key.

Repeat these steps -

- Normalization
- Population
- Candidate keys
- Re-normalization

- until every object in the data dictionary is consumed and every entity is normalized to at least 1NF.

1st Normal Form:

For a given table, every row must have the same columns. To remove embedded lists:

Separate children from parents, or -

Provide a series of columns to hold all of any parent's children.

But the latter method has drawbacks:

Multiple columns hold the children

More children require disruptive database redesign

Ugly nulls where any parent has less than the maximum.

Not normalized			
Mother	1st child	2nd child	3rd child
Sarah	Sally	Mark	Ashley
Mother	1st child		
Oblieh	Raoul		

1st normal form		
Mother	Mother	Child
Sarah	Sarah	Sally
Oblieh	Sarah	Mark
	Sarah	Ashley
	Oblieh	Raoul

Alternate 1st normal form			
Mother	1st child	2nd child	3rd child
Sarah	Sally	Mark	Ashley
Oblieh	Raoul	(nul)	(nul)

LOGICAL DATABASE DESIGN

Logical database design uses several rules or concepts which are reasonably well understood and accepted. Disagreement arises in formulating a particular methodology—the place to start and the sequence of steps to follow in applying those rules. After a brief discussion of database design methodologies, this section presents several concepts, principles, or rules which are generally recognized and applied regardless of the particular methodology used.

Database Design Methodologies

A *database design methodology* specifies a sequence of steps to follow in developing a “good” database design—one that meets user needs for information and that satisfies performance constraints. Each step consists of the application of a set of techniques or rules that may be formalized to varying degrees and embodied in software tools. A methodology should be (1) usable in a wide variety of design situations and (2) reproducible in different designers. The second objective implies that the methodology be teachable, and that those trained in applying the methodology would arrive at the same end result. This is not evident in the present state of the art. Logical database design remains very much an art.

Theory and Pry [1982] outline a database design methodology consisting of four steps;

1. *User Information Requirements*—involving the users in analyzing organizational needs, setting the scope of interest, investigating what people do (organizational tasks; usage patterns), and determining the data elements needed to perform those tasks,
2. *Conceptual Design*—developing a high-level diagrammatic representation of a logical data structure; a structure which includes object domains, events, entities, attributes, and relationships; a structure which seeks to model the users’ world.
3. *Implementation Design*—refining the conceptual design, checking for satisfaction of user needs and for consistency, and adjusting it to meet processing and performance constraints in a particular computer and DBMS environment.

4. *Physical Design*—developing record storage designs, clustering, and establishing access paths.

The techniques and rules in the steps of a methodology are applied iteratively in the process of unfolding, growing, and refining a database design. For a starting point, some suggest applying the methodology to individual user application areas or local views. Different user views may contain related complementary parts or overlapping pans. Multiple local views are then consolidated into a global logical structure or conceptual schema. The process of consolidation seeks to resolve inconsistencies, and to integrate related pieces. Even within a local view, there may be redundant, overlapping, and inconsistent pads. The rules of a methodology are intended to assist the designer in asking the right questions and representing the data structure in a coherent and consistent way regardless of the scope of the design activity, and regardless of whether it begins with individual local views or a global perspective. The product of the design activity will grow as it unfolds over the area of interest; and it will be refined as the rules are applied to focus attention on particular aspects of an infinitely complex reality and to resolve ambiguities and inconsistencies in the developing database structure.

Entity-Attribute-Relation versus Object-Relation Approaches

Perhaps the most significant difference among methodologies or approaches to logical database design is found in the point at which data items are clustered or grouped into records. The top division of the taxonomy of data structures presented in Chapter 4 reflects this division. The number of basic constructs distinguishes the two approaches: Those which presume an early clustering are often called “Entity Attribute-Relation” or “E-A-R” approaches; the alternative is called the “Object Relation” or “O-R” approach.

Historically data processing has always worked with records. Programming languages such as COBOL and FORTRAN cluster data items into records. The formation of records as a contiguous set of data items is necessary for efficient data processing. A record is the unit of access for getting data in and out of programs. Data is moved to and from secondary storage in blocks of records. Earlier data processing

systems forced a “unit record” view, that is, all data for an application had to reside in a single sequence of records (this reflected the technology of the day, which used what was called “unit record equipment”). Even today, with DBMSs supporting a multiframe data structure, data exists in the form of records in most organizations. Users are very familiar and comfortable with a record-oriented view of their data. Most designers today use an E-A-R approach to logical database design.

The major problem with the E-A-R approach to logical database design is that it allows the relationships among data items within a record to be hidden. It does not force the designer to explicitly consider and define inter record structures. This accounts for the recent emphasis in the literature on record decomposition and normalization based on an analysis of functional and multivalued dependencies. These techniques are all aimed at uncovering and making explicit the relationships among individual data items within records.

The end result of repeatedly applying record decomposition rules is irreducible varies—at which point there exists at *most one* non-identifier data item within each record. By then the designer will have considered all inter item relationships.

At the implementation or physical level, data items must be clustered into records for efficient data processing. Even at the logical level, it is still relevant and useful to think of attributes which cluster around and describe entities, whether the attribute items are considered as part of entity records or as individual object domains. It is relatively unimportant whether the design activity starts with records which are decomposed to analyze inter item relationships, or starts with object domains which are clustered to form records. In practice a designer will do both. It is important that certain rules and concepts be applied in the design process. Early formation of records is dangerous only if it inhibits the designer from properly analyzing intrarecord relationships among data items, and from considering alternative groupings of items into records -Ideally, the formation of records should be part of the implementation phase of database design since it is done primarily for system convenience and processing efficiency. In fact, it is desirable to have software tools to perform the clustering, leaving the designer to concentrate on defining the individual data objects, relationships, and performance factors and constraints.

In a strict application of the object-relation approach to logical design, all object domains are treated equally. In the E-A-R approach, attention is initially focused on entities, then on the attributes of those entities, which may turn out to be

other entities. In fact, the distinction between attributes and entities is often confusing and arbitrary. Again, regardless of the approach taken, it is important for the designer to focus attention on the more important parts of the users' world being modeled in the data structure. This is automatically done in the E-A-R approach but can also be done in the O-R approach. The designer needs a high level of abstraction when developing a data structure and may start out by representing the main entities as boxes labeled with a name only.

Rules to Guide Logical Database Design

Even though there is no widespread acceptance of any particular design methodology, there is general recognition of many underlying rules and concepts used in logical database design. They relate to conceptual design and part of implementation design.

A good database designer will generally know these rules and apply them, often intuitively, wherever they are relevant in the process of developing, checking, and refining a database design.

The following rules are presented here in a reasonably logical order, but there is no implication that they should be applied in any strict sequence. There is also no implication that these rules are sufficient or complete for the database design task. While progress is being made in formalizing the principles and process of database design, it still depends heavily on human intelligence and experience. Even experienced designers can arrive at different database designs, which purport to model the same user environment.

ENTITY: *Clearly identify the entities to be represented in the database.*

An entity is any object (person, place, thing), event, or abstract concept within the scope of interest about which data is collected. An entity is the object of decisions and actions within an organization. Entities are the pivotal elements in a data structure and must be well defined. Start out by focusing on the main entities, gradually expanding the logical data structure view to include related entities. When looking at

an existing database, clearly define the primary entity, which is described in each file (record type).

INCLUSION: *Specify the criteria for including (or excluding) entity instances from a defined class of entities.*

The ENTITY rule names a class of entities and the INCLUSION rule specifies the conditions for membership in that class. For example, does the EMPLOYEE entity class include managers, job applicants, rejected job applicants, those fired or laid off, those who quit, or employees on definite or indefinite leave? Consideration of these other “EMPLOYEES” may suggest broadening the name of the entity class, or it may give rise to another entity class. Narrowing (subsetting) or broadening the definition of the entity class represents movement along the generalization hierarchy.

ATTRIBUTE: *Identify the attributes of each entity.*

Initially focus on the major attributes of each entity. Some will be clear and obvious, some will seem to be artificial, and some may also relate to other entities. Include all attributes, which assist in understanding the nature of the entity being described. Include at least one attribute from each set of similar attributes.

ATTRIBUTE CHARACTERISTICS: *Define the characteristics of each attribute.*

Clearly define the characteristics of each attribute. Initially focus on name, type, (size), existence, uniqueness, and some indication of the nature of the value set. When describing an existing database, specify any encoding of data item values. Description of other characteristics can be deferred until later in the database design process. Eventually plan to describe one attribute per page in the final database documentation.

DERIVED ATTRIBUTE: identify and define derived attributes.

The values of an attribute may be derived from the values of other attributes in the database. Specify the derivation rule, which may be an expression for a derived item or a statistical calculation across instances of an entity type or a repeating group.

IDENTIFIER: Designate the attribute(s), which uniquely identify entity instances in each entity class.

An entity identifier may be a single attribute (EMPNO) or multiple attributes (UNIT and JOECODE for POSITION). There may be multiple identifiers for the same entity (EMPNO and SOCIAL SECURITY NUMBER). Indicate if the identifier is not guaranteed to be unique. The identifier can be a good clue to understanding the nature of the entity described in an existing file.

RELATIONSHIP: identify the primary relationships between entities.

RELATIONSHIP CHARACTERISTICS:

Define the characteristics of interentity relationships, particularly exclusivity and exhaustibility (or dependency).

Exclusivity refers to whether instances of one entity type can be related to at most one or more than one instance of another entity type. Since it is defined in both directions there are four possibilities: 1:1, 1:Many, Many: 1, Many: Many. Exhaust stability (also called dependency or personality) specifies whether or not an instance of one entity type must be related to an instance of another entity type. Indicate if there is some condition on the dependency of a relationship. Also indicate if there is some minimum or maximum cardinality on the "many" side of a relationship. (See section 6.3.3 for more detail on these characteristics.)

FOREIGN IDENTIFIER: indicate the basis for each relationship by including, as an attribute in one entity type, the identifier from each related entity type.'

Every relationship is based upon common domain(s) in the related entity records. At the logical level, it is necessary to include the identifier of a related entity as a foreign identifier. In the storage structure, if the common domain is not explicitly

stored in a related record, then some form of physical pointer is necessary to represent the relationship.

DERIVED RELATIONSHIPS: Suppress derived relationships.

The logical database design should not include relationships, which can be derived from other relationships. For example, it is reasonable to think of organizational units as possessing a pool of skills. Furthermore, such information can be retrieved from the database. However, such a relationship should not be defined since it is derived from the ORGANIZATION-EMPLOYEE relationship and the EMPLOYEE-SKILL relationship. An organizational unit only possesses skills because it has employees who possess skills.

REPEATING GROUP: *isolate any multivalued data item or repeating group of data items within a record.*

This rule ensures that a record only contains atomic (single-valued) data items, thus allowing only flat files. This is also called first normal form. The real importance of this rule is to force the designer to explicitly recognize a “something-to-many” relationship and possibly a new entity type. If a repeating group of data items becomes a new entity record type, the identifier of its parent record must propagate down into the new record. If the relationship was actually many-to-many, the propagated identifier becomes part of the identifier of the new record; if the relationship was one-to-many, the propagated identifier becomes a foreign identifier in the new record (but not part of the identifier). Multi-valued data items or nested repeating groups of data items may be included in the storage structure of a record (as they are in a hierarchical data structure).

PARTIAL DEPENDENCY: *Each attribute must be dependent upon the whole record (entity) identifier.*

An attribute that is dependent upon only part of the identifier should be removed from the record, and placed in a record where that part of the identifier is the

whole identifier. Suppose we had a record with the following data items: EMPNO, SKILLCODE, SKILL DESCRIPTION, and PROFICIENCY. The identifier would have to be the first two data items jointly since PROFICIENCY relates to both of them together. However, DESCRIPTION relates only to the SKILLCODE and, therefore, should not be in this record. A record with no partial dependencies is said to be in second normal form.

TRANSITIVE DEPENDENCY: *Each attribute within a record must be directly dependent upon the entity identifier.*

Any attribute, which is not directly dependent upon the record identifier, should be removed from the record, and related directly to the object on which it is functionally dependent. For example, if the EMPLOYEE record contained UNIT and BOSS, and the employee was moved to another organizational unit, it would not be sufficient to update the employee's UNIT—the BOSS data item would also have to be changed. The update anomaly results because BOSS is directly dependent upon UNIT and not EMPNO. BOSS does not belong in the EMPLOYEE record *even if processing is faster and easier*; it belongs in the ORGANIZATIONAL UNIT record. A record with no partial or transitive dependencies is said to be in third normal form. Restated: An attribute should be dependent upon the identifier, the whole identifier, and nothing but the identifier.

Application of the previous three rules to arrive at third normal form requires an examination of every attribute in a record. A record not in third normal form produces undesirable update anomalies. To identify these anomalies, the designer can ask: If a given attribute is updated, what other attributes must change, or if another attribute is updated, what effect will it have on the given attribute?

NAMING: *Assign names to entities, attributes, and relationships using a consistent, well-defined naming convention.*

When describing an existing database, watch for naming inconsistencies—different names for the same object, or the same name used to refer to different objects.

STORAGE & ACCESS: *Suppress any consideration of physical storage structures and access mechanisms in describing the logical structure of the data.*

This includes any stored ordering on the records in a file, and whether or not a data item is indexed. Do not be concerned with questions of how to find or access a particular record in a file, perhaps along a relationship. Remember, all relationships are inherently bi-directional.

FOXPRO & Dbase

FoxPro enables you to convert your existing dBase programs and applications. To make the process seamless, FoxPro version 2.6 includes:

A Catalog Manager Interface that provides convenient access to database files and functions.

A collection of wizards, which simplify common database tasks.

Language additions that provide dBASE IV language compatibility.

FoxPro 2.6 Professional Edition

FoxPro 2.6 is available in two editions, Standard and Professional. The Professional edition includes all the features of the Standard edition, plus:

Client-Server Query Wizard

Connectivity Kit

Distribution Kit

Library Construction Kit

These additional features allow you to connect to remote data, distribute your applications to others, and build sharable libraries of functions.

Terminology Differences Between dBASE and FoxPro

Most of the terminology that a dBASE user encounters in FoxPro will be familiar, but there are a few differences. This table lists some of the more prominent differences.

dBASE Term	FoxPro Term
Database	Table
Form	Screen
Horizontal Bar Menu	Menu Bar
Pop-up Menu	Popup
Pull-down Menu	Menu or Menu Popup
View	Cursor

File Extensions

Following is a list of the extensions assigned to files used by FoxPro. In the "FoxPro Platform" column; "All" indicates three platforms, FoxPro for MS-DOS, Windows and Macintosh.

File Extension	FoxPro Files	FoxPro Platform
.ACT	FoxDoc Action Diagram	All
.APP	Generated Application	All
.BAK	File Backup	All
.CDX	Compound Index File	All
.DBF	Table/Database	All
.DBT	FoxBASE+ Memo File	All
.DLL	Dynamic Link Library	Windows
.DOC	FoxDoc Reports	All
.ERR	Compilation Error File	All
.EXE	Executable Program	MS-DOS and Macintosh
.FCT	FoxPro Catalog Memo File	All
.FPC	FoxPro Catalog	All
.FPQ	Wizard-generated Updatable Query	All
.FKY	Macro Save File	All
.FLL	FoxPro for Windows Library	Windows
.FMT	Format File	All
.FPT	Table/Database Memo File	All
.FRT	Report Memo File	All
.FRX	Report File	All
.FXD	FoxDoc Supporting File	All
.FXP	Compiled FoxPro Program	All
.HLP	Graphical Help File	Windows and Macintosh
.IDX	Single-Entry Index File	All
.INT	Code Page File	All
.INT	Collation Sequence File	All
.LBT	Label Memo	All
.LBX	Label File	All
.MEM	Memory Variable Save File	All
.MLB	Macintosh Library	Macintosh
.MNT	Menu Memo	All
.MNX	Menu File	All
.MPR	Generated Menu Program	All
.MPX	Compiled Menu Program	All
.MSG	FoxDoc Message File	All
.PJT	Project Memo	All
.PJX	Project File	All
.PLB	FoxPro for MS-DOS Library	MS-DOS
.PRG	FoxPro Program	All
.PRX	Compiled Format	All
.QPR	Generated Query Program	All
.QPX	Compiled Query Program	All
.SCT	Screen Memo	All

File Name and Extension	File Description	FoxPro Platform
.SCX	Screen File	All
.SPR	Generated Screen Program	All
.SPX	Compiled Screen Program	All
.TBK	Memo File Backup	All
TMP	Temporary File	All
.TXT	Text File	All
.VUE	View File	All
.WIN	Window Save File	All
CONFIG.FP	Configuration File	MS-DOS
CONFIG.FPW	Configuration File	Windows
CONFIG.FPM	Configuration File	Macintosh
FOXHELP.DBF		
FOXHELP.FPT	DBF-style Help	All
FOXHELP.HLP	Graphical Help File	Windows and Macintosh
FOXUSER.DBF	Resource File	All

File Extension Differences Between dBASE and FoxPro

The following table lists differences in file extensions between dBASE and FoxPro.

File Type	dBASE Extension(s)	FoxPro Extension(s)
Applications	.APP, .PRG	.APP
Catalogs	.CAT	.FPC, .FCT
Indexes	.MDX, .NDX	.IDX, .CDX
Labels	.LBL, .LBG	.LBX, .LBT
Menus	(none)	.MNX, .MNT, .MPR
Programs	.PRG, .PRS	.PRG, .FXP
Projects	(none)	.PJX, .PJT
Queries	.QBE, .QBO, .UPD, UPO	.QPR, .FPQ, .CSQ
Reports	.FRM, .FRG, .FRO	.FRX, .FRT
Screens	.SCR, .FMT, .FMO	.SCX, .SCT, .SPR
Tables	.DBF, .DBT •	.DBF, .FPT

Switching from dBASE

dBASE and FoxPro are dialects of the same language, Xbase. All dBASE III Plus programs and most dBASE IV programs will run in FoxPro without making any changes. If a program does not run correctly, or if you want to enhance a dBASE program in FoxPro, you must convert your dBASE files to FoxPro. If you want, the Catalog Manager can automatically convert these files for you.

Overview

The following procedure is an overview of the process you will use to convert your dBASE files into FoxPro.

1. Create a backup copy of your dBASE files.
2. Determine if your program requires changes.
3. Modify your files.
4. Enhance your dBASE program or create a new FoxPro application.

To enhance your program or create new applications, you can use the FoxPro Power Tools, including the Screen Builder, Report Writer, and RQBE. FoxPro also provides a set of wizards for quickly creating tables, queries, screens, reports, and applications. Power Tools and wizards are accessible from within the Catalog Manager when you choose New or Modify, or from the Command Window.

Terminology Differences Between dBASE IV and FoxPro

Catalog Manager

The Catalog Manager is a graphical interface that enables you to manage catalogs (.FPC or .CAT files) easily. Catalogs can contain tables, queries, screens, reports, labels and programs. From this central location, you can create, modify, and run these FoxPro files. You can also use AutoMigrate to migrate existing dBASE III and dBASE IV files. The Catalog Manager, written entirely in FoxPro, is an example of the type of application you can develop in FoxPro.

FoxPro catalogs are single-user files. If you run more than one concurrent session of FoxPro, each session must have a local copy of the FOXUSER resource file. You can add table, query, screen, report, label and program files to more than one catalog.

Using Wizards

Microsoft FoxPro 2.6 provides the following wizards to help you complete common data management tasks:

Client-Server Query Wizard	Report Wizard
Group/Total Report Wizard	Screen Wizard
Label Wizard	SQL Query Wizard
Mail Merge Wizard	Table Wizard
Multi-Column Report Wizard	Updateable Query Wizard

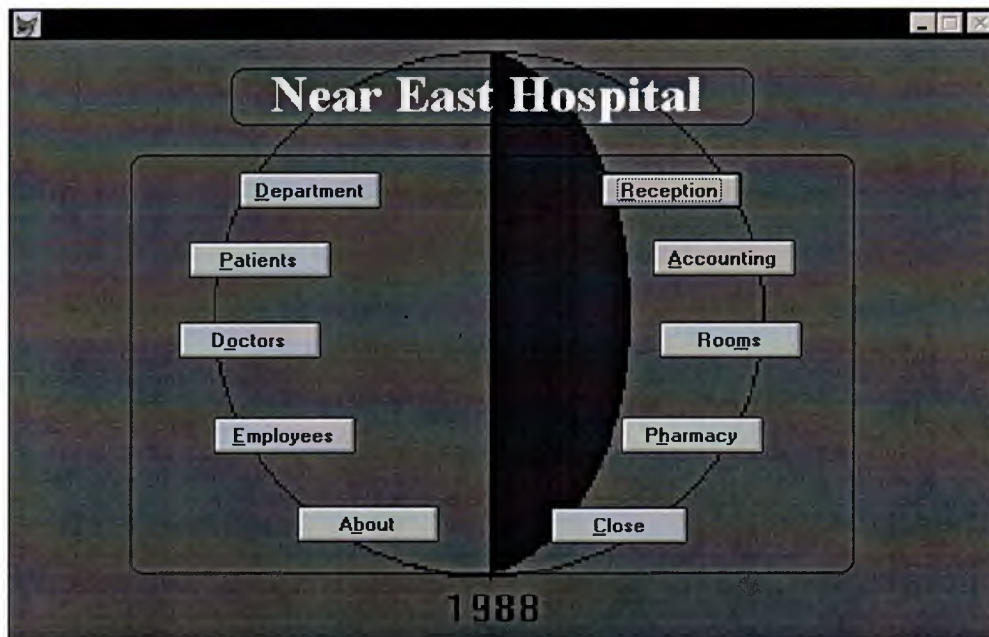
FoxPro Enhancements to the Xbase Language

FoxPro contains many commands and functions that do not exist in dBASE IV (or, of course, dBASE III PLUS).

Main Screen:

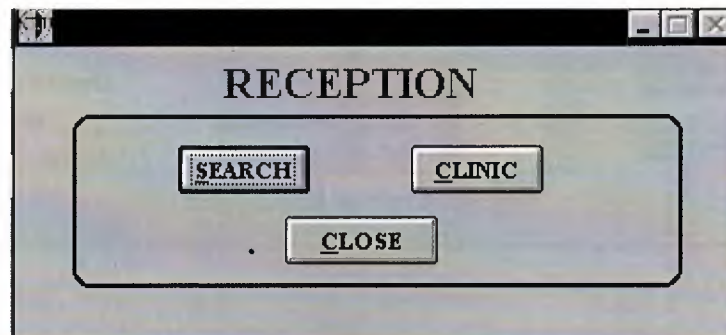
The main screen describes the 1st user interface, which guide the user through in the different screens in the program.

It includes 8 bottoms: Reception, Departments, Patients, Doctors, Employees, Rooms, Accounting, External Pharmacy, About bottom, and Exit bottom to quiet the program.



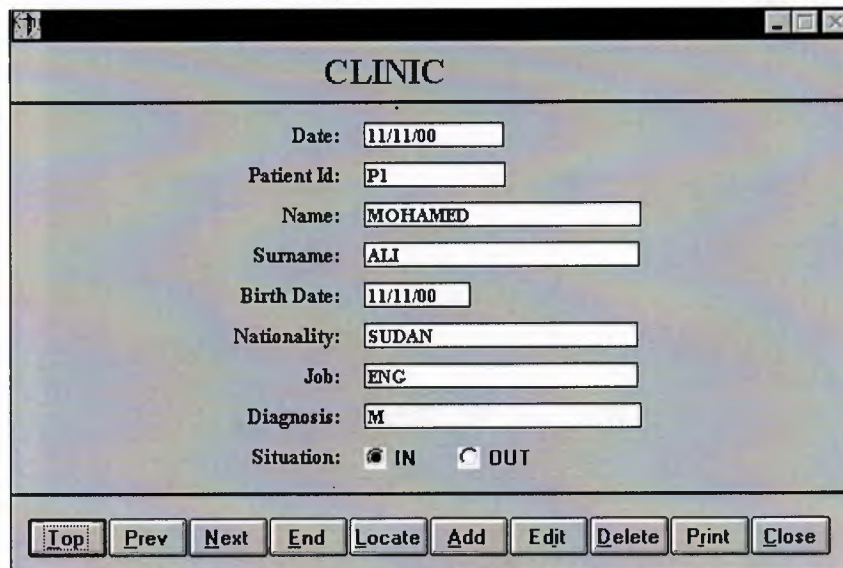
1. Reception:

Here we automate a real life procedure, when we divide the Reception into tow sub screens: Search & clinic

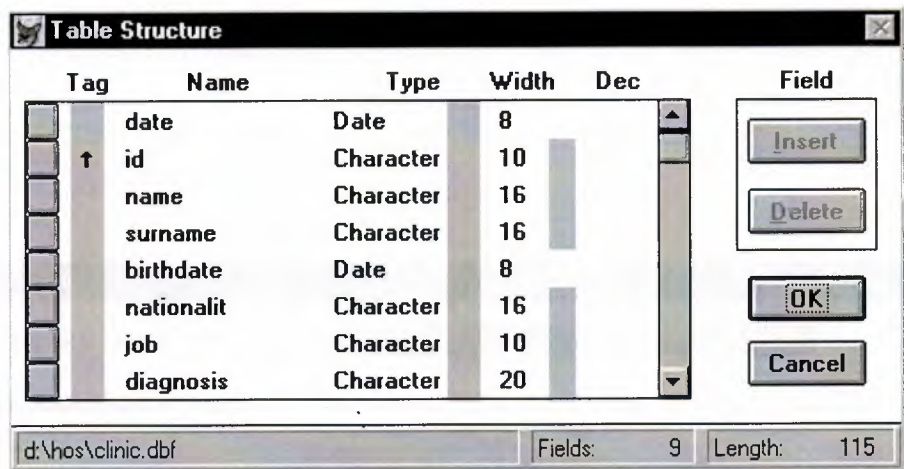


1.1 Clinic:

Although the clinic screen inside the reception screen but it is acting as the essentially data entry In the system, here we ask the new coming patients to fill the admission form, which we can get by pressing the print out bottom in the button of the screen. If the patient considered as an IN patient he will have a record inside our PATINTS database file, otherwise he will be an Outpatient, and he can take his prescription to the Pharmacy department.



A screenshot of a software window titled "CLINIC". It contains a form with the following fields: Date (11/11/00), Patient Id (P1), Name (MOHAMED), Surname (ALI), Birth Date (11/11/00), Nationality (SUDAN), Job (ENG), Diagnosis (M), and Situation (radio buttons for IN and OUT, with IN selected). At the bottom, there is a row of buttons: Top, Prev, Next, End, Locate, Add, Edit, Delete, Print, and Close.



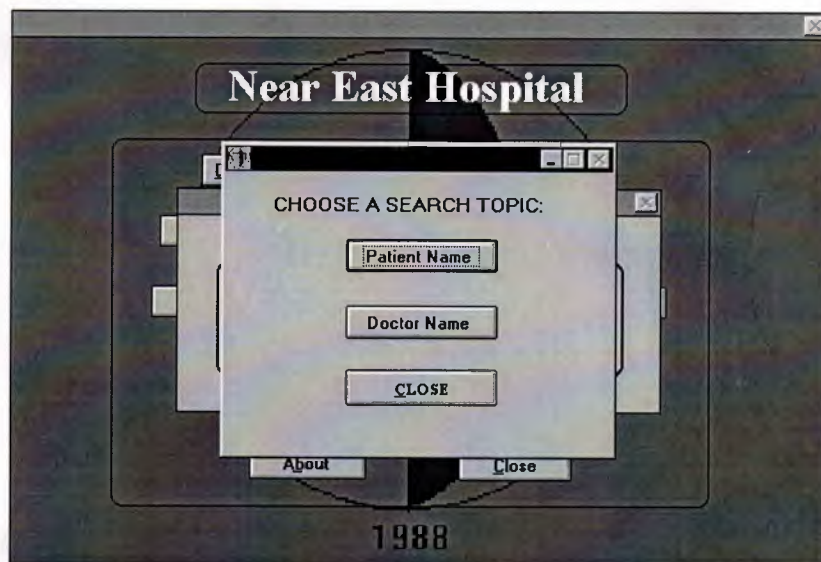
A screenshot of a "Table Structure" dialog box. It displays a table with columns: Tag, Name, Type, Width, Dec, and Field. The table lists fields for a database: date (Date, 8), id (Character, 10, primary key), name (Character, 16), surname (Character, 16), birthdate (Date, 8), nationalit (Character, 16), job (Character, 10), and diagnosis (Character, 20). To the right of the table are buttons for Insert, Delete, OK, and Cancel. At the bottom, it shows the file path "d:\hos\clinic.dbf", "Fields: 9", and "Length: 115".

Tag	Name	Type	Width	Dec	Field
	date	Date	8		
↑	id	Character	10		
	name	Character	16		
	surname	Character	16		
	birthdate	Date	8		
	nationalit	Character	16		
	job	Character	10		
	diagnosis	Character	20		

d:\hos\clinic.dbf Fields: 9 Length: 115

1.2 Search:

In order to offer a high quality services inside the system, the 2nd sub screen of the reception menu is the SEARCH engine, our searching engine can give the users a very soon result for any question about the main tow categories inside our system: DOCTORS & PATIENTS. And locate the users with their available information.

A screenshot of a search input field. It consists of a black title bar with a close button (X) on the right. Below the title bar, the text 'Enter Patient Name:' is followed by a text input field.

Patient						
Patint_id	Name	Surname	Datebirth	Gender	Address	Job
P11	YASIR	OMER	12/12/79	M	NEU	STUE
P13	PIETER	CHAK	10/05/66	M	MAGUSA	EMP
P2	MOHAMED	alamin	07/19/88	M	neu	student
P3	samir	gafar	01/09/75	M	Girne	Student
P4	atilla	kerem	12/30/77	M	turkey	student
P5	mehmet	fatih	04/04/70	M	magusa	worker
P9	TEEM	RETXHARD	07/30/66	M	GIRNE	ENG

Search					
Dr_id	Name	Surname	Address	Searching is OK	
D1	SAMIR	ZAIDAN	LEFKOSA	2	
D2	jhon	smith	lefkosa	2277645	3
D3	abdullah	suliman	girne	8155423	6
D4	WATSON	SEEP	KK	8789654	0

Enter The Doctor Name:

2. Departments:

The department screen give information about the various DEPARTMENT inside the system, as the name of department, number of doctors, patients, employees and rooms, and also the head of the department.

The image shows two windows from a database application. The top window is titled "DEPARTMENTS" and contains a form with the following fields and values:

Field	Value
Dep_id:	D1
Dep_name:	Internal medicine
Extension:	111
No of rooms:	28
No of doctors:	7
No of patients:	70
No of emp:	14
Head of dep:	ali ali

Below the form are buttons: Top, Prev, Next, End, Locate, Add, Edit, Delete, Print, and Close.

The bottom window is titled "Table Structure" and displays the following table:

Tag	Name	Type	Width	Dec	Field
	dep_id	Character	10		Insert
	dep_name	Character	30		
	extension	Character	10		Delete
	noofroom	Character	10		
	noofdoctor	Character	10		OK
	noofpatien	Character	10		
	noofemp	Character	10		Cancel
	headofdep	Character	20		

At the bottom of the "Table Structure" window, it shows the file path "d:\hos\departme.dbf", "Fields: 8", and "Length: 111".

3. Doctors:

Doctor's names, specialization, and privet information are located in the designated DOCTORS screen.

The screenshot shows a window titled "DOCTORS" with a light blue background. It contains several text input fields for doctor information. At the bottom, there is a row of buttons: Top, Prev, Next, End, Locate, Add, Edit, Delete, Print, and Close.

Dr ID:	D1	Name:	SAMIR
Surname:	ZAIDAN	Address:	LEFKOSA
Phone:	2235410		
Age:	30	Nationalit:	SUDANESE
Date Of Hiring:	07/07/93	Salary:	2,000 \$
Rank:	A	Extention:	6661
Specilization:	Ear & Nose & Throat		

Tag	Name	Type	Width	Dec	Field
	dr_id	Character	10		<div>Insert</div> <div>Delete</div> <div>OK</div> <div>Cancel</div>
	name	Character	16		
	surname	Character	16		
	address	Character	30		
	phone	Character	10		
	extention	Character	10		
	specilize	Character	20		
	salary	Numeric	12	0	

d:\hos\doctor.dbf

Fields: 12

Length: 163

4. Patients:

The main and most important category in our system, which all the system run and automates to provide a good service for them are the PATIENTS. In here we complete the patient's file which had come from the clinic by the detailed information, like address, job, sex and cause of admission, as a matter of how the system is setup, no such patients can have a file unless they are described IN patients, otherwise the system give a message saying that " This Patient is out".

Inside the Patients screen we have tow sub screens:

Patients

Patient ID: P13 Name: PIETER

Surname: CHAK Date of Birth: 10/05/66

Gender: ☒ M ☐ F Address: MAGUSA

Job: EMP

Phone: 8839237 Nationality: INDIAN

Date Of Adm: 02/02/99 Cause of Admis: SOME HEADACHE

Discharge Date: 02/14/99

Dept Name: STOMATOLOGY

Room No: R4 Bed No: B6

Daily Checkup History

Top Prev Next End Locate Add Edit Delete Print Close

Table Structure

Tag	Name	Type	Width	Dec	Field
	patint_id	Character	10		<input type="button" value="Insert"/>
	name	Character	16		<input type="button" value="Delete"/>
	surname	Character	16		
	datebirth	Date	8		<input type="button" value="OK"/>
	gender	Character	8		<input type="button" value="Cancel"/>
	address	Character	40		
	job	Character	10		
	nationalit	Character	14		

d:\hos\patient.dbf Fields: 15 Length: 234

4.1 History:

Here the patients that has been revisiting the hospital would have some information about their previous conditions declared in file has the name HISTORY. This screen display medical information about the patient, such like allergies, inherited families' diseases, exams tested for him and diagnosis. This file has great utilities, cause it make a good relation between the present and past medical problems for the patient, and also assist the new doctors with the wanted information about their patients.

THIS PATIENT IS OUT

HISTORY OF PATIENT

Patient ID: Name: Sur Name:

Personal:

Family: Allergies:

Past: Present:

Diff Diagnosis: Diagnosis:

Physicexam: Biochemical:

Blood:

RIOLOGICAL TEST

X_ray: Nuiclear:

Sonography:

Chief Complaint:

Top Prev Next End Locate Add Save Cancel Print Close

Table Structure							
Tag	Name	Type	Width	Dec		Field	
	patient_id	Character	10			<input type="button" value="Insert"/> <input type="button" value="Delete"/>	
	name	Character	16				
	surname	Character	16			<input type="button" value="OK"/> <input type="button" value="Cancel"/>	
	chiefcomp	Character	30				
	personal	Character	15				
	past	Character	15				
	family	Character	15				
	present	Character	15				
d:\hos\history.dbf						Fields: 17	Length: 273

4.2 Daily Checkup:

The 2nd sub screen inside the patient's screen is the DAILY CHECKUP screen, which concern in watching and daily control of the patient's situation. By clicking the print bottom we can get a printed out schedule showing the patient's situation at the time.

DAILY CHECK-UP

Patient Id:

Name: Sur Name:

Check Date:

Exam:

Medicine:

Doctor Name:

Table Structure

Tag	Name	Type	Width	Dec	Field
	patint_id	Character	10		<input type="button" value="Insert"/> <input type="button" value="Delete"/> <input type="button" value="OK"/> <input type="button" value="Cancel"/>
	name	Character	15		
	surname	Character	15		
	checkdate	Date	8		
	exam	Character	10		
	medicine	Character	10		
	doctor	Character	16		

d:\hos\daychkup.dbf Fields: 7 Length: 85

5. Employees:

The EMPLOYEES screen shows up some information about the employees like their name, job, date of hiring and salaries.

EMPLOYEES

Emp ID: Name:

Surname: Address:

Job:

Date Of Hiring: Nationality:

Phone: Salary: \$

Table Structure

Tag	Name	Type	Width	Dec	Field
	emp_id	Character	10		<input type="button" value="Insert"/>
	emp_name	Character	16		<input type="button" value="Delete"/>
	emp_surnam	Character	16		
	address	Character	70		
	jop	Character	12		
	date_hire	Date	8		<input type="button" value="OK"/>
	nationalit	Character	14		<input type="button" value="Cancel"/>
	phone	Character	10		

d:\hos\employe.dbf Fields: 9 Length: 172

6. Rooms:

To offer a good service inside the hospital, and to provide the patients with high cares for their health, our room must conforms the medical specifications. In the ROOMS screen we enter some data like room number, department, type and beds in the room, full or empty.

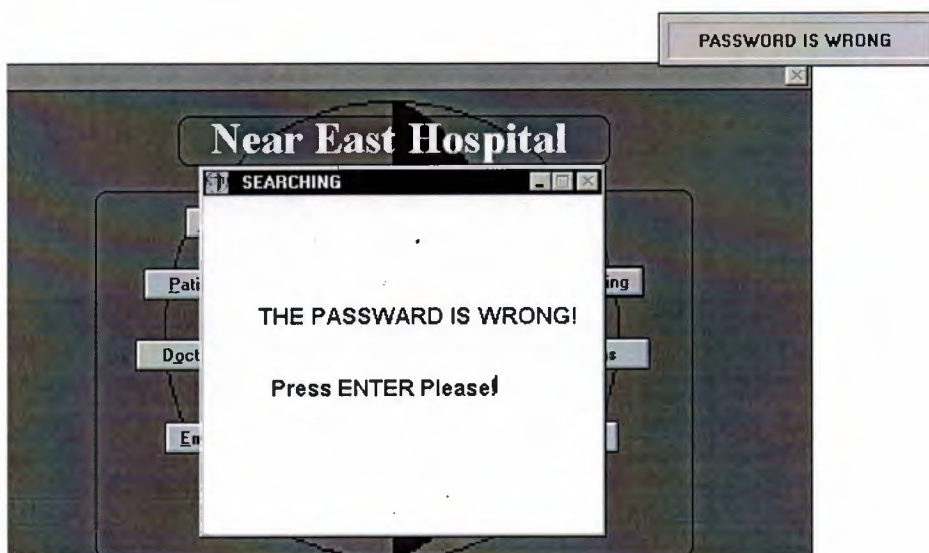
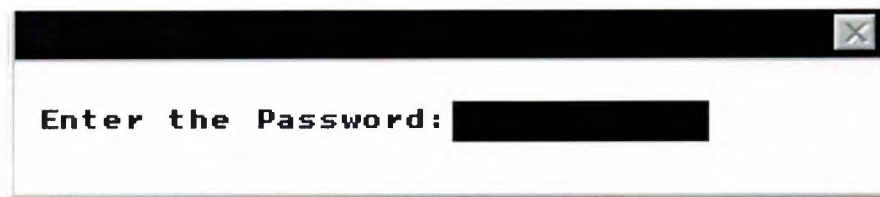
The screenshot shows a window titled "ROOMS" with a form for entering room data. The form is divided into two columns. The left column contains: "Room No:" with a text box containing "R3", "Department:" with a text box containing "BLOOD BANK", "Room Type:" with a dropdown menu showing "NORMAL", and "No of Beds:" with a text box containing "3". The right column contains: "Bed No:" with a text box containing "R3 B2", "Bed Price:" with a text box containing "22" followed by a dollar sign, and "Situation:" with two radio buttons, "FULL" (selected) and "EMPTY". At the bottom of the window is a row of buttons: "Top", "Prev", "Next", "End", "Locate", "Add", "Edit", "Delete", "Print", and "Close".

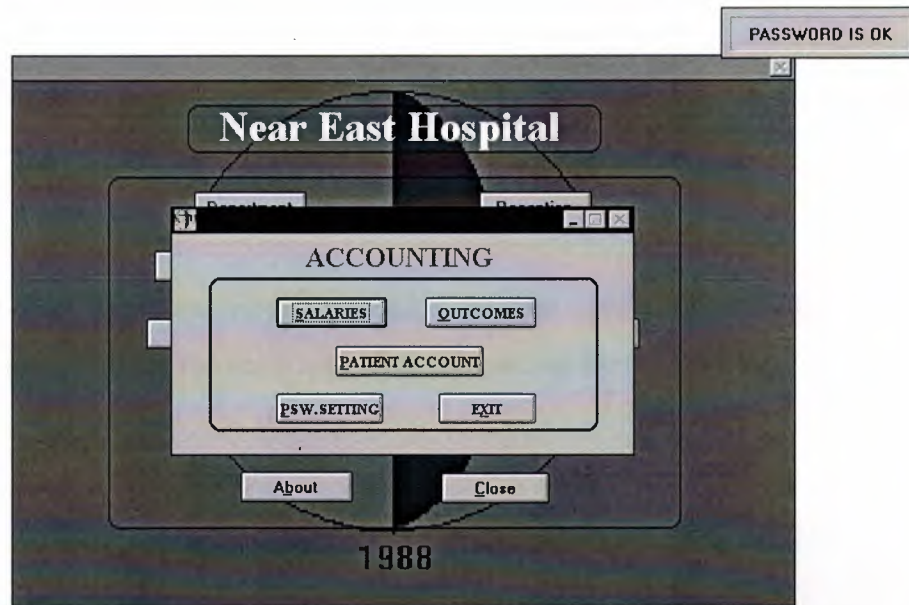
The screenshot shows a "Table Structure" dialog box for a table named "room.dbf". It displays a list of fields with their respective tags, names, types, widths, and decimal places. The fields are: room_no (Character, 10), dep (Character, 25), room_type (Character, 12), bedno (Numeric, 10, 0), bed_no (Character, 10), bed_price (Numeric, 10, 0), and situation (Character, 10). On the right side of the dialog are buttons for "Insert", "Delete", "OK", and "Cancel". At the bottom, it shows the file path "d:\hos\room.dbf", the number of fields "7", and the total length "88".

Tag	Name	Type	Width	Dec
	room_no	Character	10	
	dep	Character	25	
	room_type	Character	12	
	bedno	Numeric	10	0
	bed_no	Character	10	
	bed_price	Numeric	10	0
	situation	Character	10	

7. Accounting:

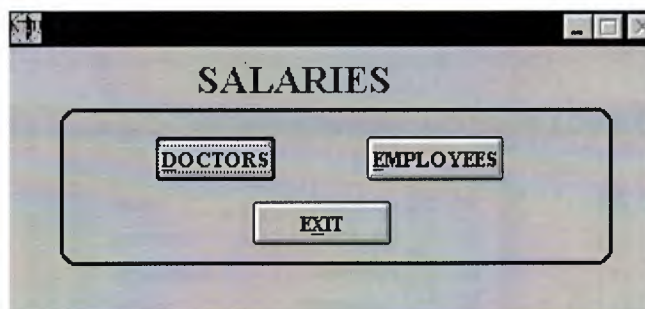
This is the only department in our system that has a certain need for security, so here the system will ask you about your authority, and you should know the PASSWORD, otherwise you will receive a message saying "The password is wrong". If you have the authority then the ACCOUNTING screen will appear to you. This screen consists of four sub screens.





1. Salaries:

This screen also has tow sub screens:



7.1.1 Doctors:

Here there is a view over the doctor's salaries, specialize, and under special function we can calculate the net wages after subtracting the tax from it.

DOCTORS SALARIES

Doctor ID:

Name:

Surname:

Salary:

Specialization:

Tax: \$

Total: \$

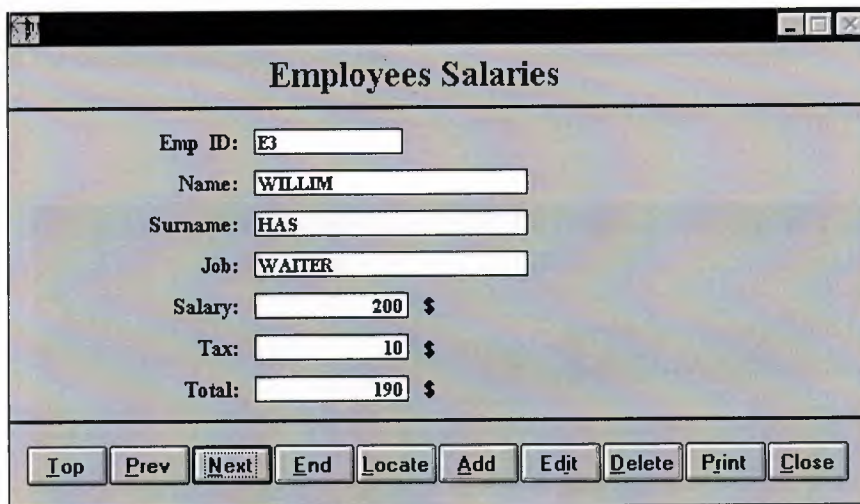
Table Structure

Tag	Name	Type	Width	Dec	Field
	name	Character	16		<input type="button" value="Insert"/>
	surname	Character	16		<input type="button" value="Delete"/>
	specialize	Character	20		
	salary	Numeric	12	0	<input type="button" value="OK"/>
	tax	Numeric	12	0	<input type="button" value="Cancel"/>
	total	Numeric	12	0	

d:\hos\salaries.dbf Fields: 7 Length: 99

7.1.2 Employees:

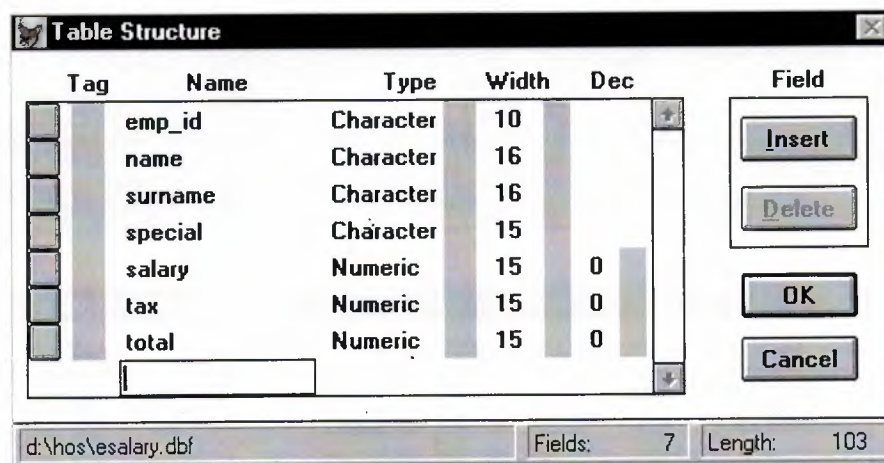
The same like the former one, this screen views the salaries of employees after taking the tax out.



The 'Employees Salaries' form displays employee information and salary details. It includes input fields for Emp ID, Name, Surname, Job, Salary, Tax, and Total, each followed by a dollar sign. The form also features a set of navigation buttons at the bottom.

Field	Value
Emp ID:	E3
Name:	WILLIM
Surname:	HAS
Job:	WAITER
Salary:	200
Tax:	10
Total:	190

Buttons: Top, Prev, Next, End, Locate, Add, Edit, Delete, Print, Close



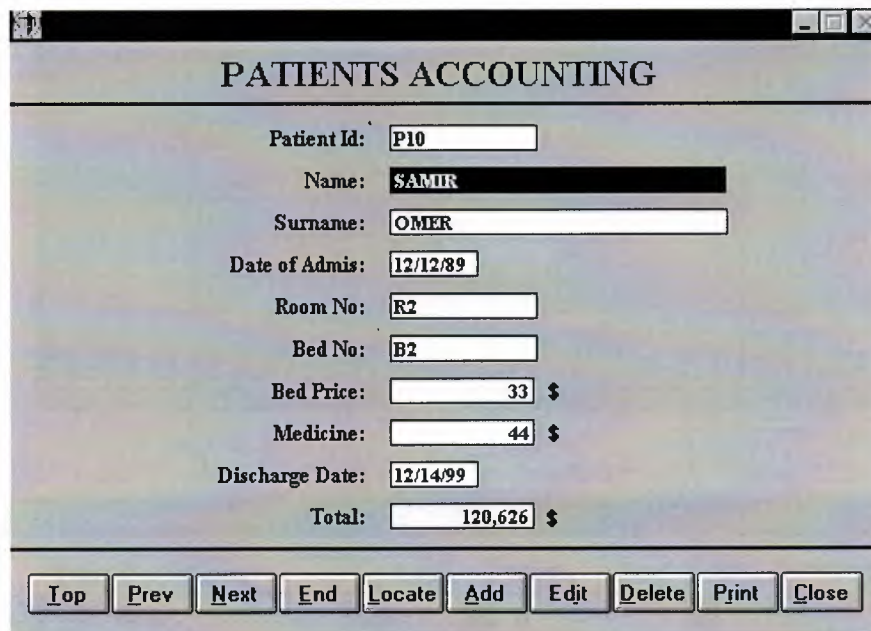
The 'Table Structure' dialog shows the structure of the 'd:\hos\esalary.dbf' table. It lists fields with their names, types, widths, and decimal places. The 'Field' column on the right contains buttons for Insert, Delete, OK, and Cancel.

Tag	Name	Type	Width	Dec	Field
	emp_id	Character	10		<div>Insert</div> <div>Delete</div> <div>OK</div> <div>Cancel</div>
	name	Character	16		
	surname	Character	16		
	special	Character	15		
	salary	Numeric	15	0	
	tax	Numeric	15	0	
	total	Numeric	15	0	

d:\hos\esalary.dbf Fields: 7 Length: 103

7.2 Patients Account:

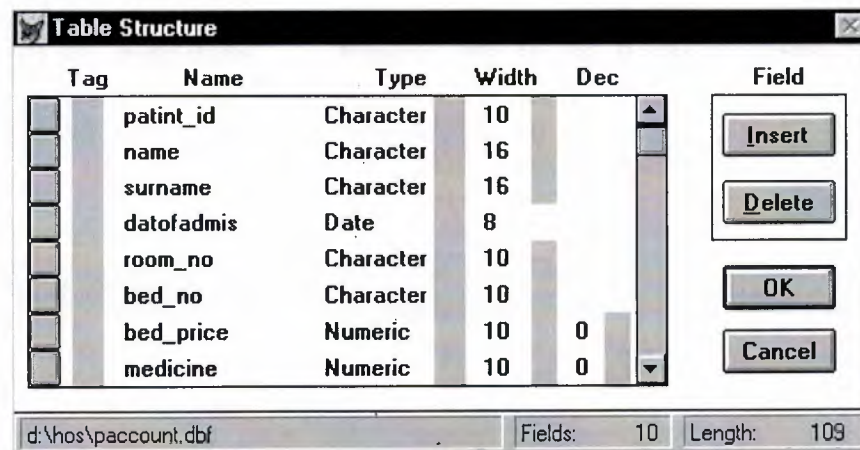
The patients account screen describe the financial relation between the inpatient and the accounting department, we have another function calculate the total amount the patient should pay. By obtaining the duration that the patient stay inside the hospital, price of the room and the price of medicines. And here we can get a receipt by clicking the print bottom.



A screenshot of a software window titled "PATIENTS ACCOUNTING". The window contains a form with the following fields and values:

Field	Value
Patient Id:	P10
Name:	SAMIR
Surname:	OMER
Date of Admis:	12/12/89
Room No:	R2
Bed No:	B2
Bed Price:	33 \$
Medicine:	44 \$
Discharge Date:	12/14/99
Total:	120,626 \$

At the bottom of the form, there is a row of buttons: Top, Prev, Next, End, Locate, Add, Edit, Delete, Print, and Close.



A screenshot of a "Table Structure" dialog box. It displays a table with columns: Tag, Name, Type, Width, Dec, and Field. The table lists the fields of the "paccount.dbf" table.

Tag	Name	Type	Width	Dec	Field
	patint_id	Character	10		Insert
	name	Character	16		
	surname	Character	16		Delete
	datofadmis	Date	8		
	room_no	Character	10		OK
	bed_no	Character	10		
	bed_price	Numeric	10	0	Cancel
	medicine	Numeric	10	0	

At the bottom of the dialog box, there is a status bar showing the file path "d:\hos\paccount.dbf", the number of fields "10", and the total length "109".

7.3 Outcomes:

As so as we automate a real life application, we should consider the financial relation between the hospital and outside environments, and considering the total amount of OUTCOMES can bound that. Not more than buying of foods, new medical instrument and maintenance.

The screenshot shows a window titled "OUTCOMES". Inside, there is a "Date:" label followed by a text box containing "02/12/99". Below this are four rows of labels and text boxes: "Foods:" with "56", "Medicines:" with "56", "Medinstrum:" with "78", and "Other Stocks:" with "45". A "Total:" label is followed by a text box containing "235". At the bottom of the window is a row of buttons: "Top", "Prev", "Next", "End", "Locate", "Add", "Edit", "Delete", "Print", and "Close".

The screenshot shows a "Table Structure" dialog box for a table named "outcomes.dbf". The table has 6 fields. The fields are listed in a table with columns: Tag, Name, Type, Width, Dec, and Field. The fields are: date (Date, 8), foods (Numeric, 10, 0), medicines (Numeric, 10, 0), medinstrum (Numeric, 10, 0), otherstock (Numeric, 10, 0), and total (Numeric, 10, 0). There is an empty row below the "total" field. To the right of the table are buttons for "Insert", "Delete", "OK", and "Cancel". At the bottom of the dialog box, the file path "d:\hos\outcomes.dbf" is shown, along with "Fields: 6" and "Length: 59".

Tag	Name	Type	Width	Dec	Field
	date	Date	8		
	foods	Numeric	10	0	
	medicines	Numeric	10	0	
	medinstrum	Numeric	10	0	
	otherstock	Numeric	10	0	
	total	Numeric	10	0	

d:\hos\outcomes.dbf Fields: 6 Length: 59

7.4 Password Setting:

The last sub screen in the accounting department expand the security of the system by giving the ability for authority people to change the password from time to time, as a routine to avoid any unexpected hacking to the system security.



A screenshot of a software dialog box titled "PASSWORD SETTING". The dialog box has a standard Windows-style title bar with minimize, maximize, and close buttons. Below the title bar, the text "PASSWORD SETTING" is centered. Underneath, there is a label "Enter the New Password:" followed by a text input field containing the text "OMER". At the bottom of the dialog box, there are three buttons: "Edit", "Delete", and "Close".

8. Pharmacy:

The system provides the patients in and outside the hospital with an EXTERNAL PHARMACY.

The screenshot shows a window titled "PHARMACY". It contains four data entry fields: "Medicin ID:" with the value "023", "Medicine:" with the value "SCORNED", "Price:" with the value "12" and a dollar sign symbol, and "Pharmacist:" with the value "MOHAMED HUSSAM". Below these fields is a row of navigation buttons: "Top", "Prev", "Next", "End", "Locate", "Add", "Edit", "Delete", "Print", and "Close".

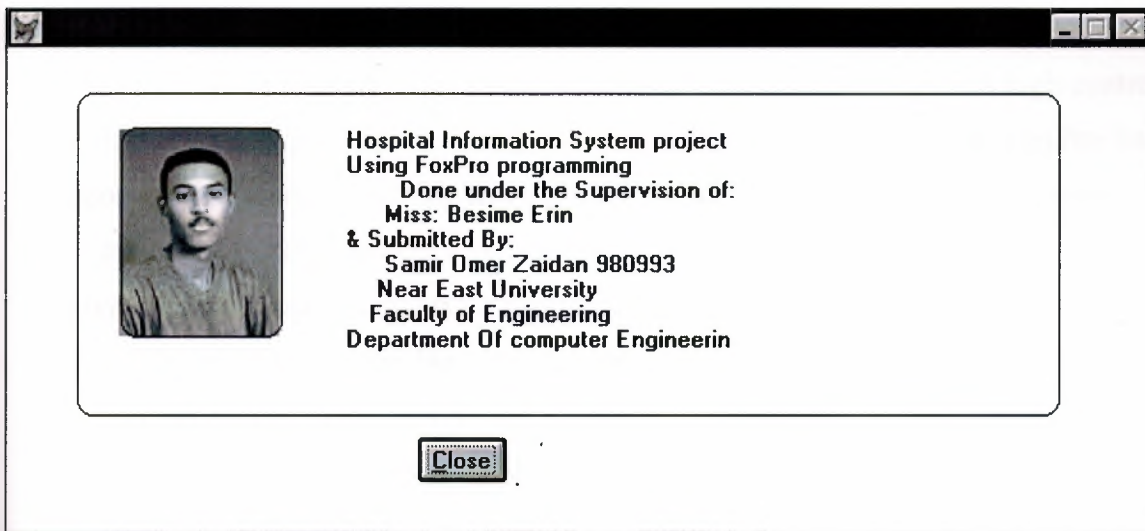
The screenshot shows a "Table Structure" dialog box for a database table. It contains a table with the following columns: Tag, Name, Type, Width, Dec, and Field. The table lists four fields: "medicin_id" (Character, 8), "medicine" (Character, 10), "price" (Float, 12, 0), and "pharmacist" (Character, 20). To the right of the table are buttons for "Insert", "Delete", "OK", and "Cancel". At the bottom, the file path "d:\hos\pharmacy.dbf" is shown, along with "Fields: 4" and "Length: 51".

Tag	Name	Type	Width	Dec	Field
	medicin_id	Character	8		<input type="button" value="Insert"/>
	medicine	Character	10		<input type="button" value="Delete"/>
	price	Float	12	0	<input type="button" value="OK"/>
	pharmacist	Character	20		<input type="button" value="Cancel"/>

d:\hos\pharmacy.dbf Fields: 4 Length: 51

9. About Box:

The ABOUT BOX bottom is a small preview about the project, supervisor and the programmer.



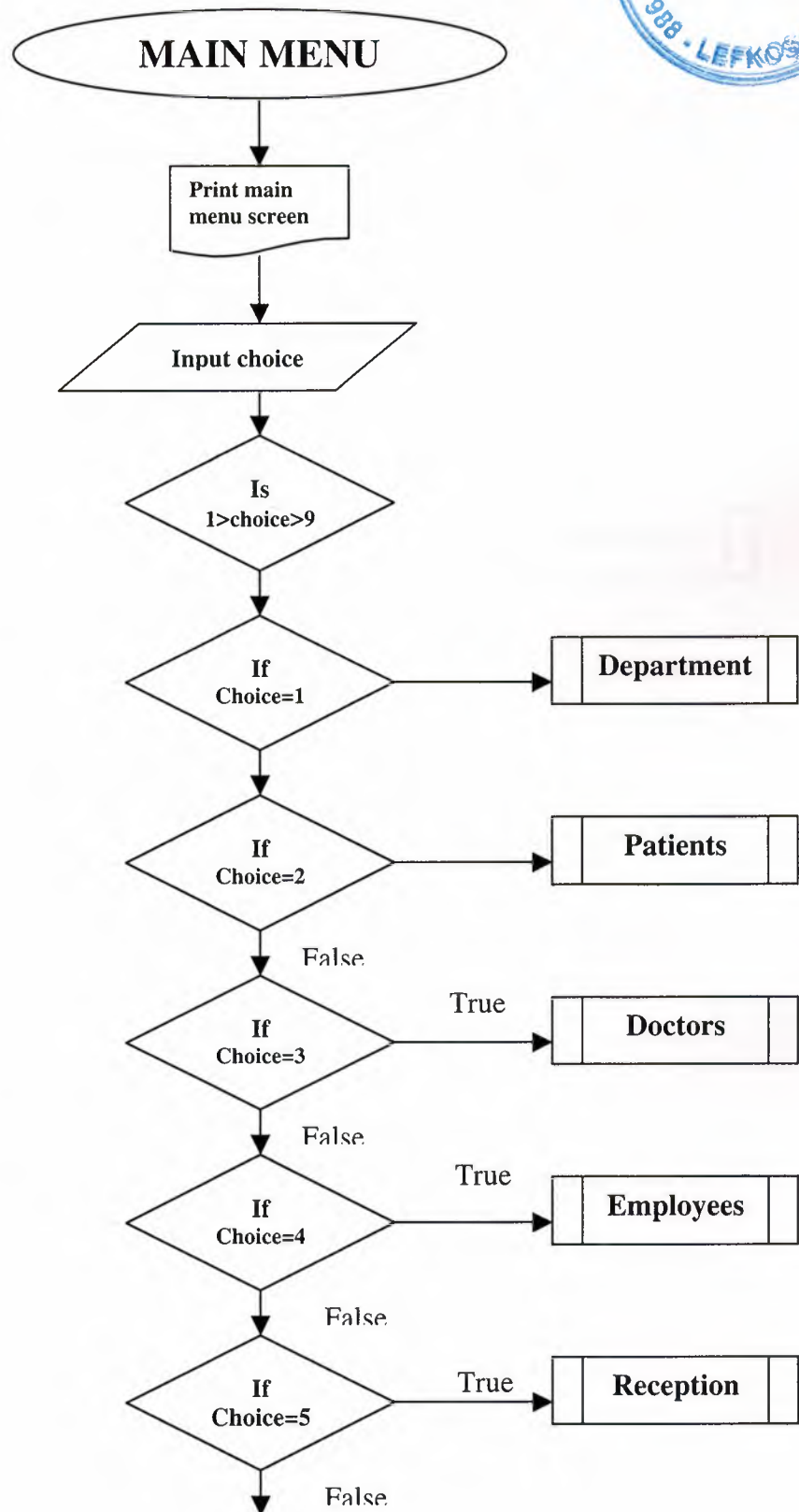
Conclusion

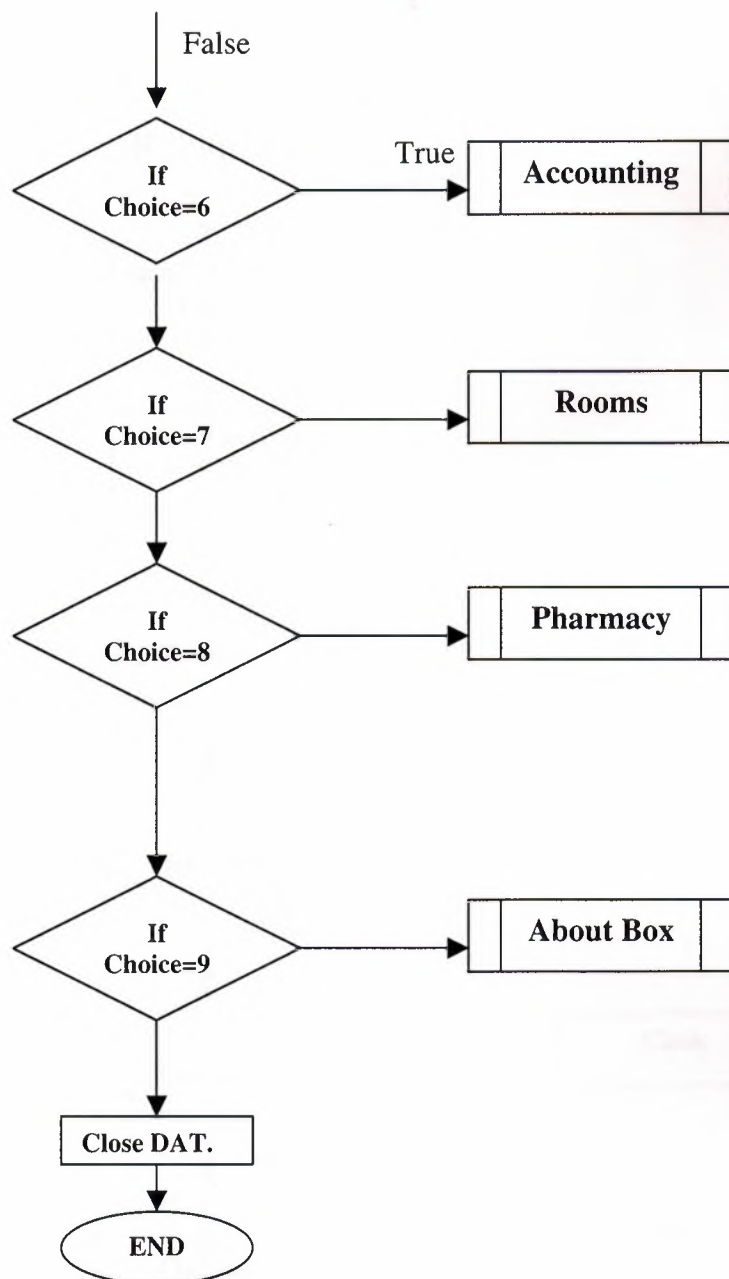
The database programming are one of the most growing up fields in the computer world, and the applications of it are spreading and entering everywhere in our real life.

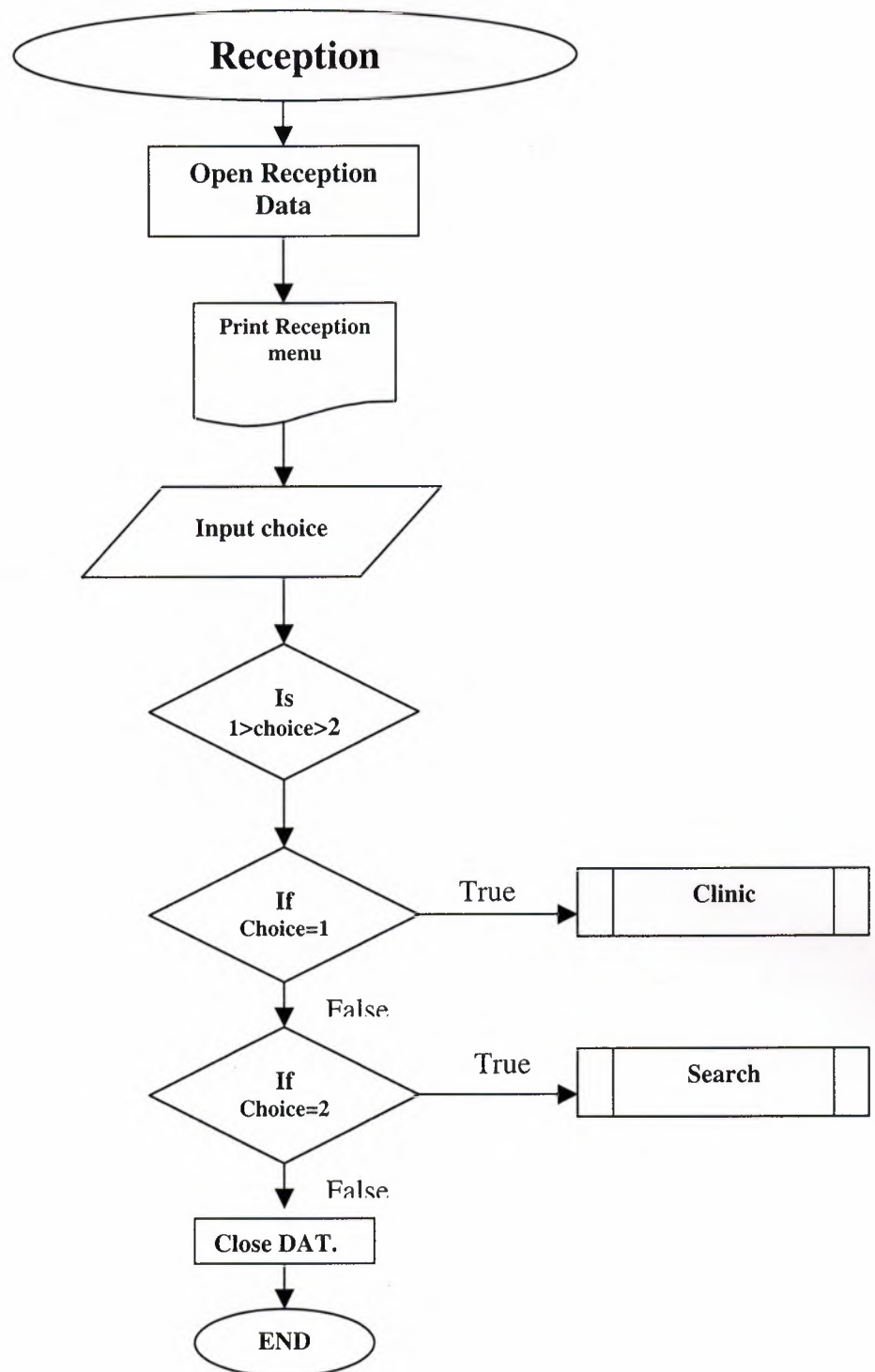
The FoxPro is one of the most used programming referring to its high control in database management and design, I believe that the entire Visual FoxPro had become more useful is gaining more ground on the application, but the main reason for me to use FoxPro because I've been familiar with it, and I got some experience in solving its problems and making a good control through in the program.

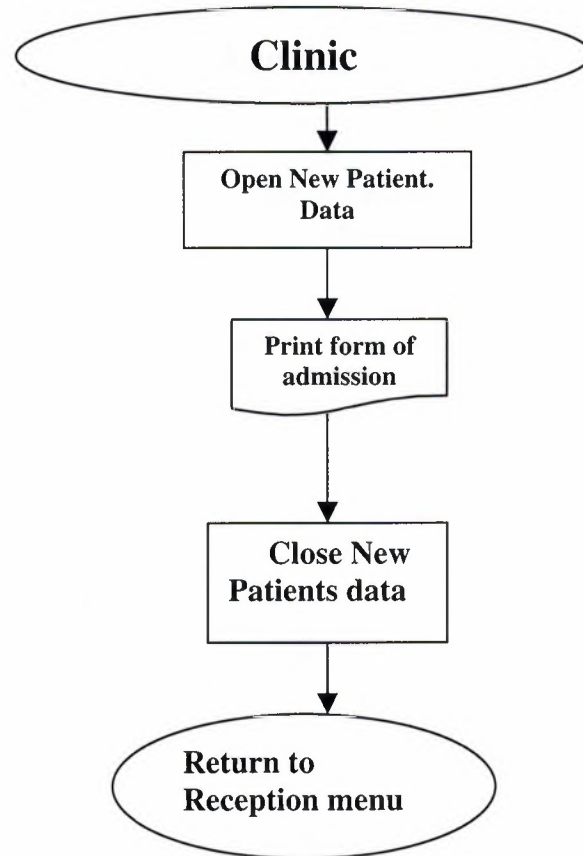
In this project, the Hospital database has been taking regarding to a small Hospital in Sudan, and it can do the same services that provided by any Hospital or Clinic, with a slight changes to the design cope with it.

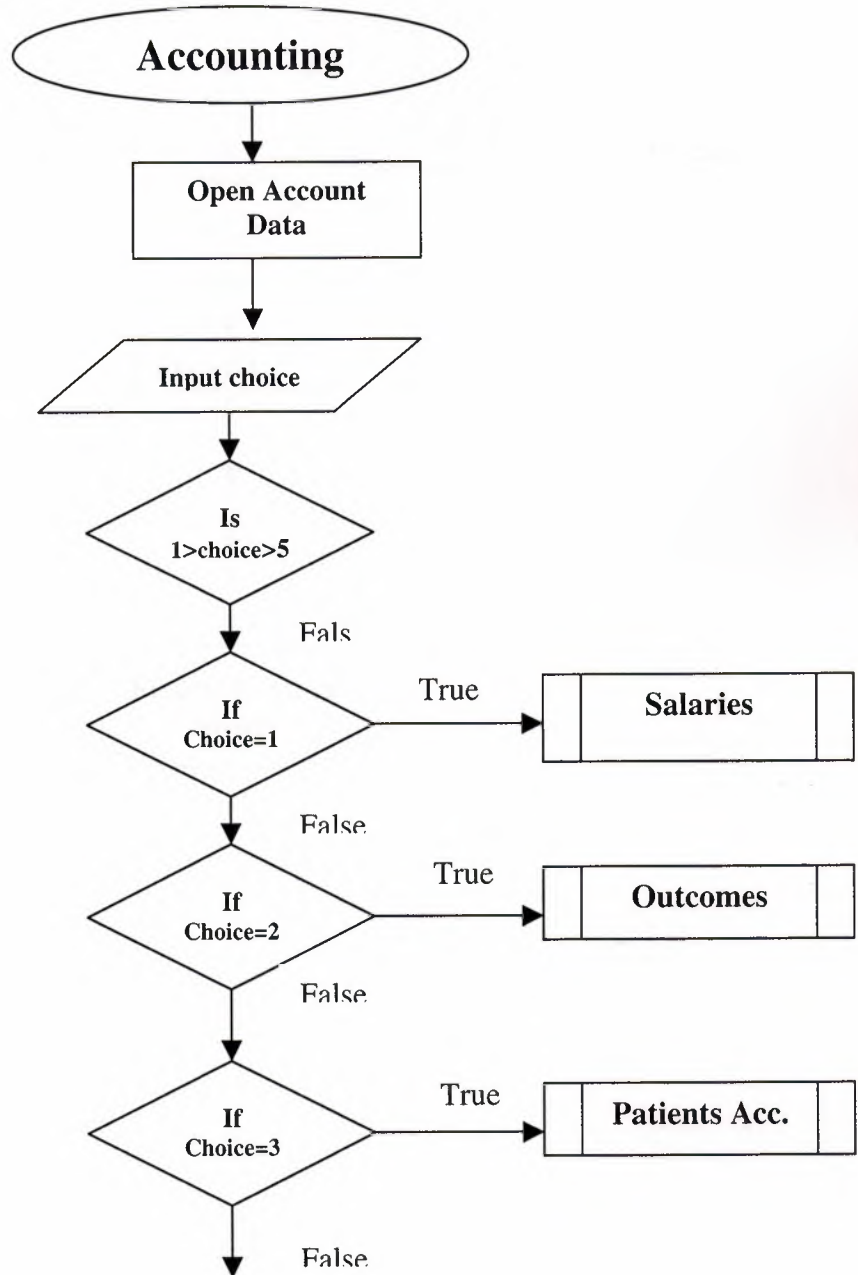
I had given attended to make it easy to the use of any public user, security abilities are capable and upgrading are easy, and all these features fall in favors of the program.

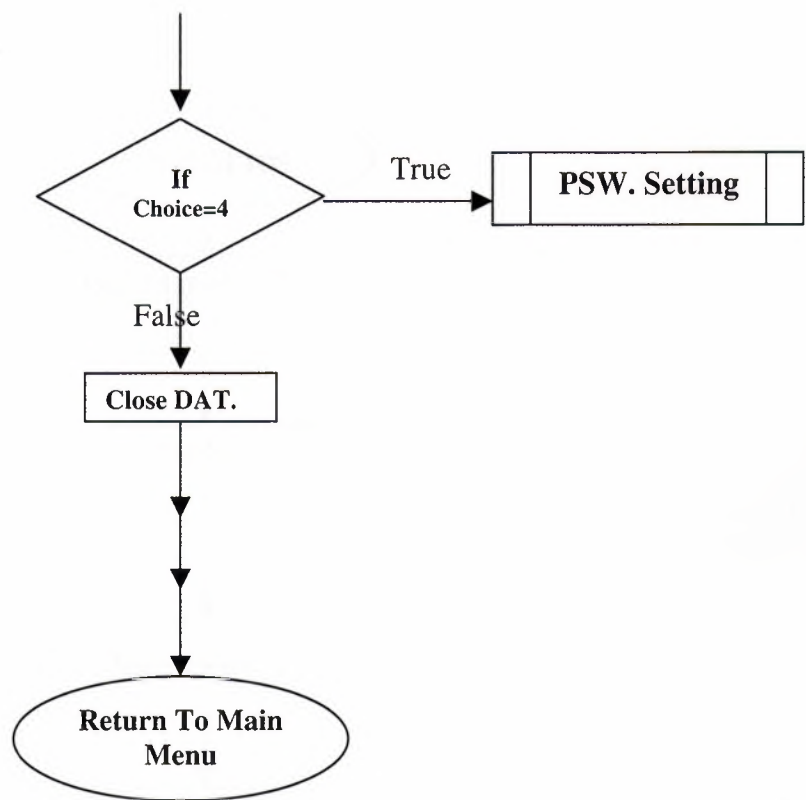


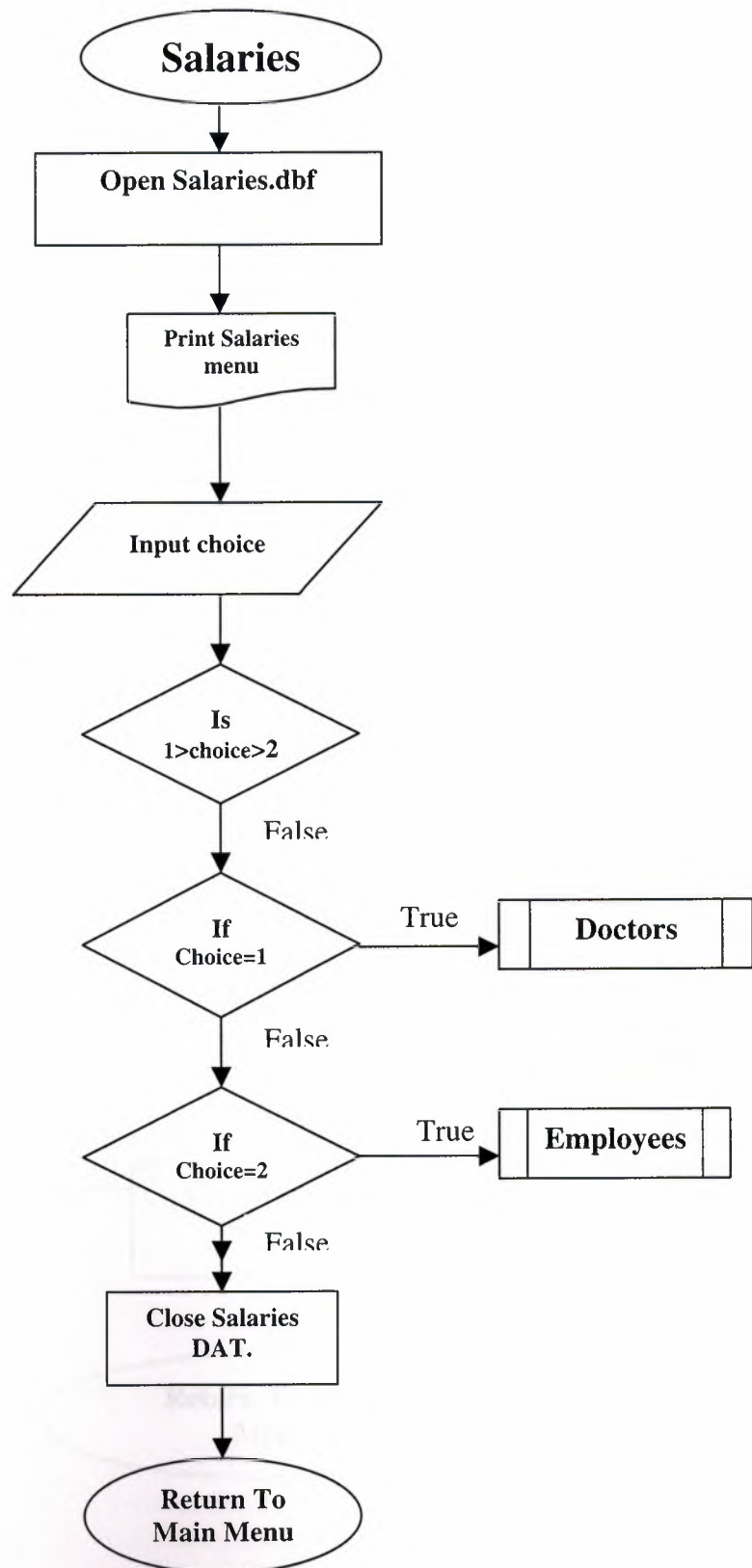


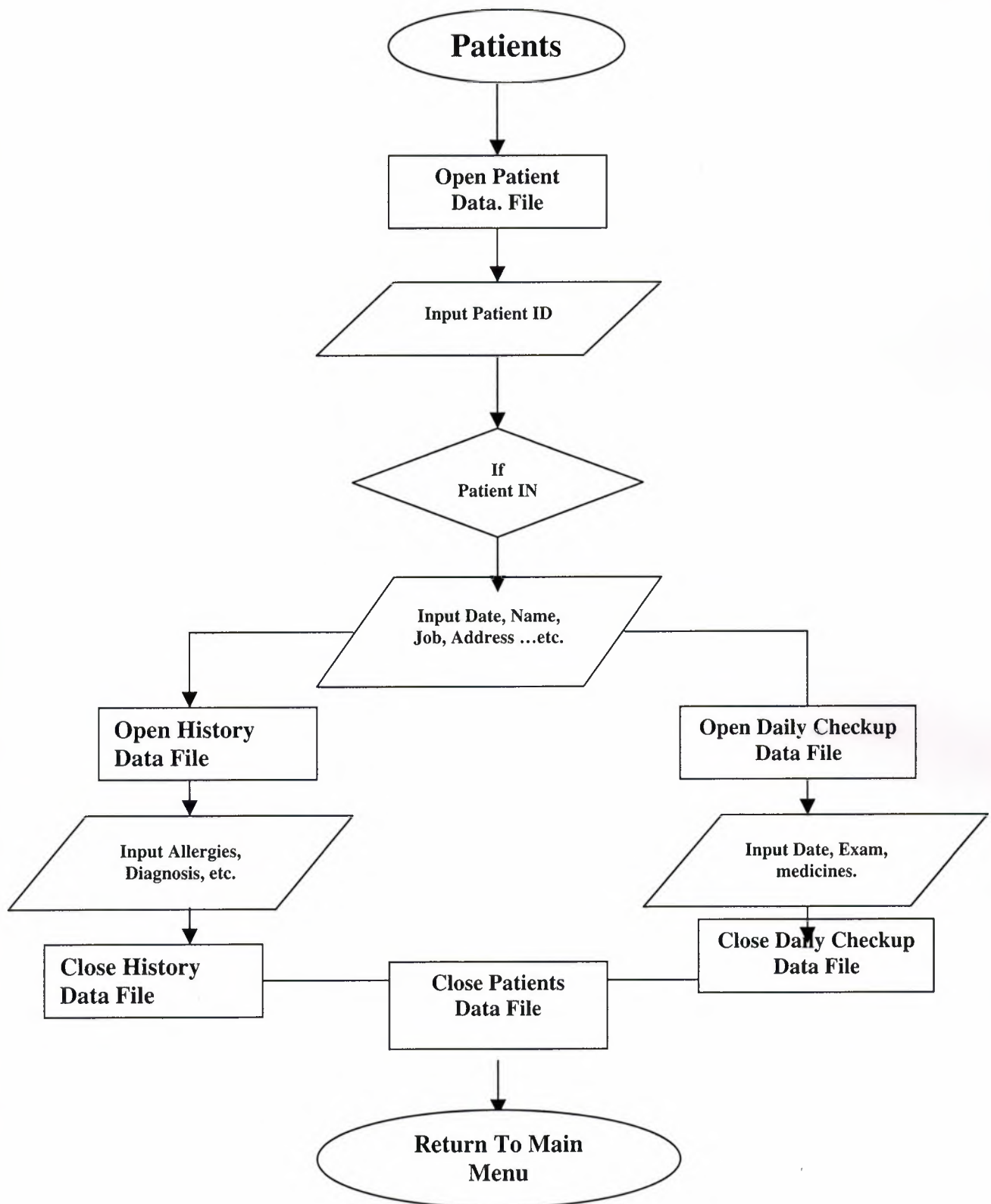


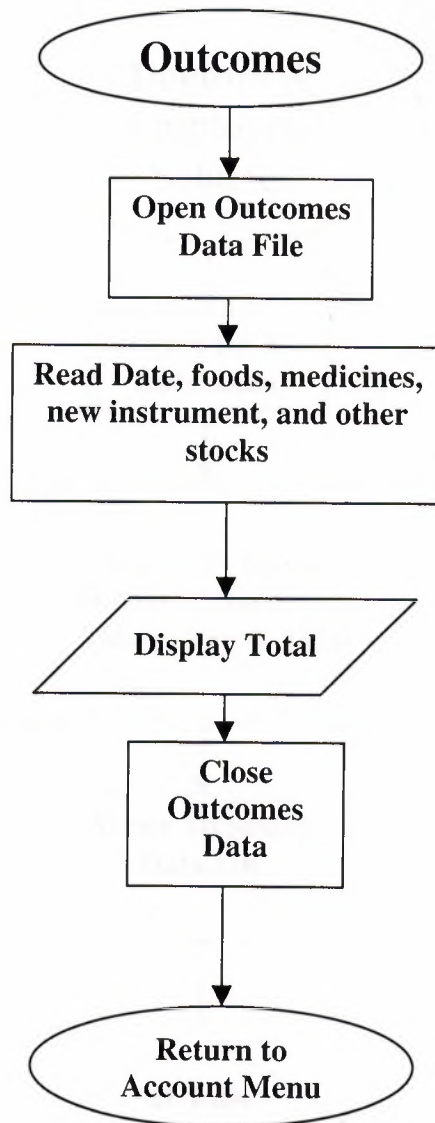


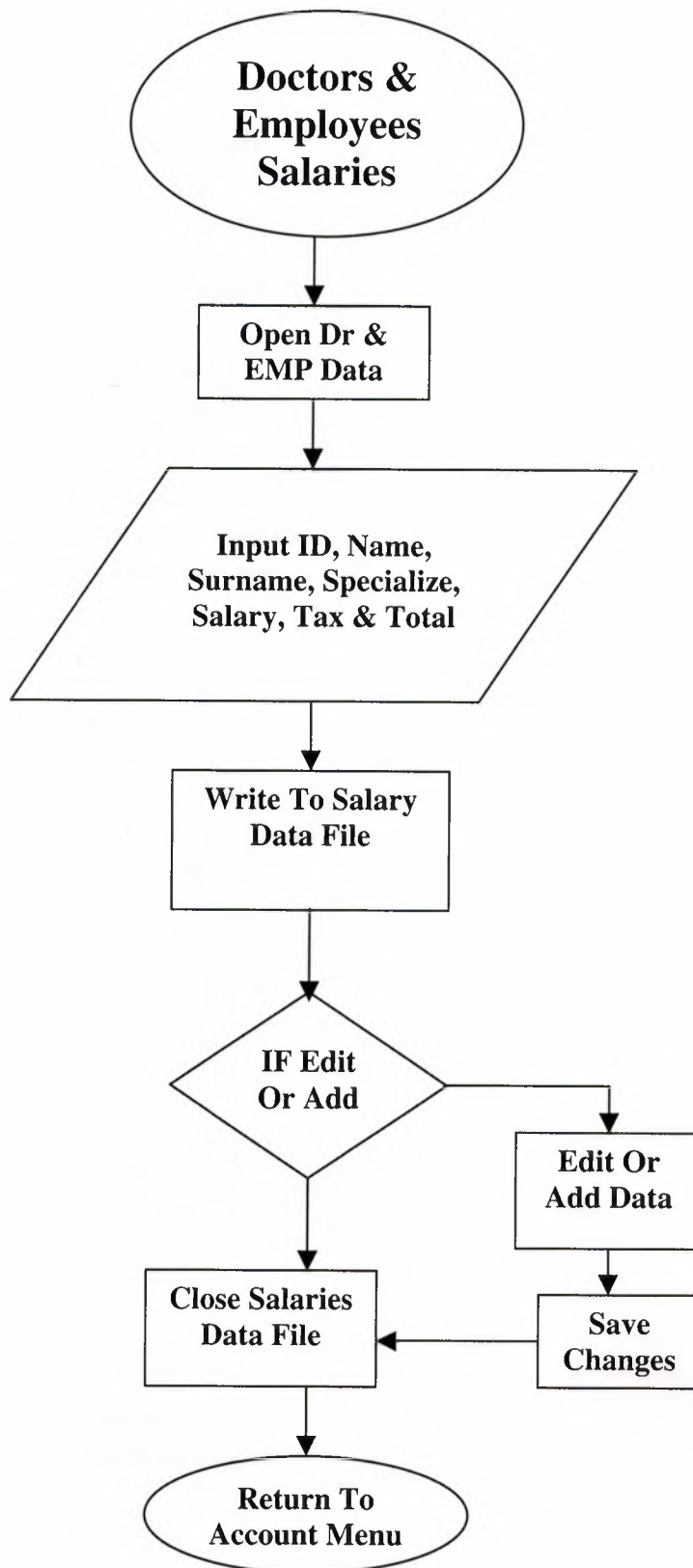


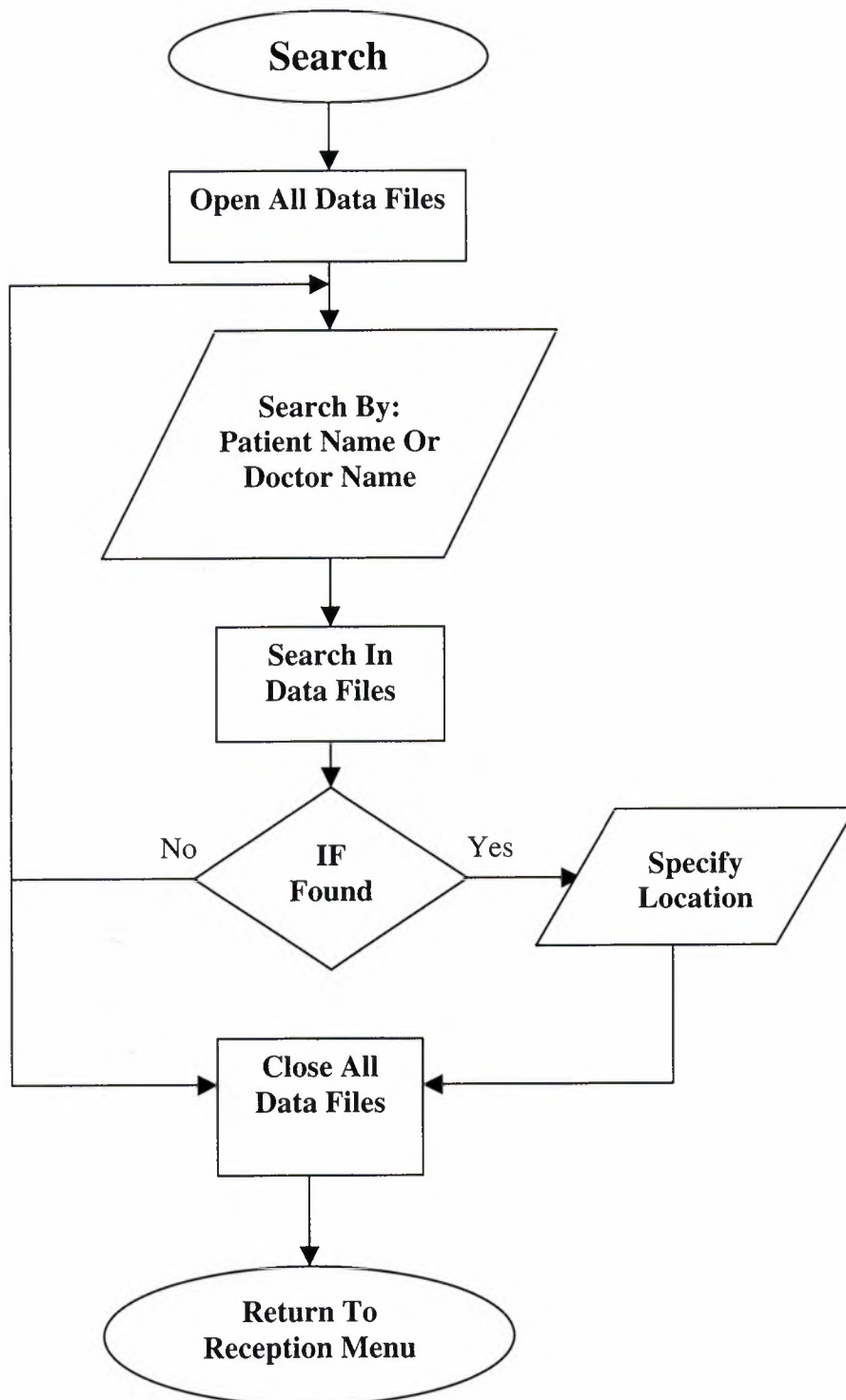


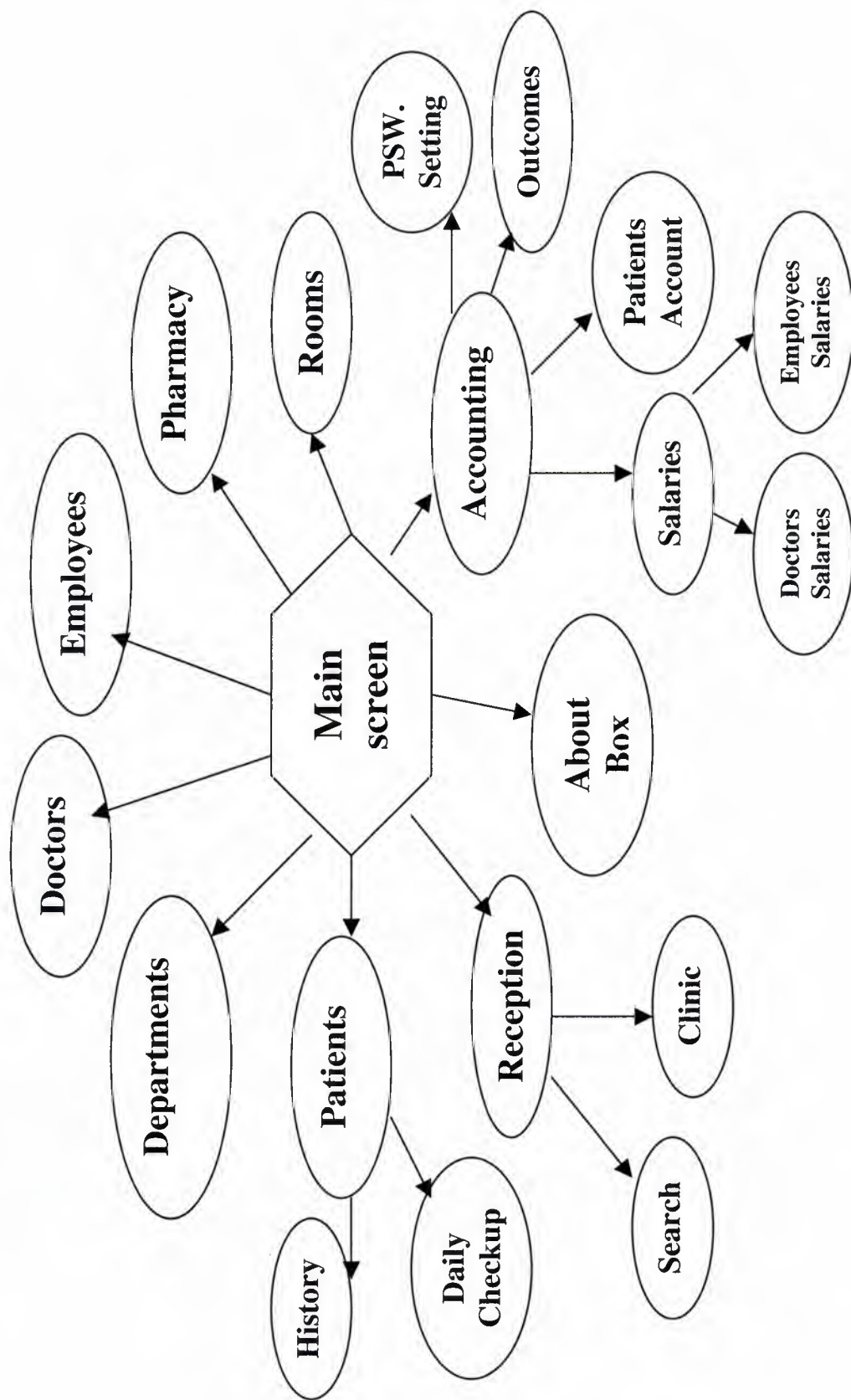












Main Screen Code:

```
SET TALK OFF
SET CLOCK off
close all
SET READBORDER ON
    IF NOT WEXIST("mmm")
        DEFINE WINDOW mmm ;
            AT 0.000,0.000 ;
            SIZE 30.308,98.167 ;
            FONT "MS Sans Serif", 8 ;
            STYLE "B" ;
            FLOAT ;
            CLOSE ;
            MINIMIZE ;
            COLOR RGB(,,0,128,128) ;
            ICON FILE LOCFILE("NEU2.ICO","ICO", ;
                "Where is neu2?")
            MOVE WINDOW mmm CENTER
    ENDIF

    IF WVISIBLE("mmm")
        ACTIVATE WINDOW mmm SAME
    ELSE
        ACTIVATE WINDOW mmm NOSHOW
    ENDIF

    @ 0.000,19.667 SAY (LOCFILE("neu3.bmp","BMP|ICO|PCT|ICN", "Where is
neu3?" )) BITMAP ;
        SIZE 28.538,58.000 ;
        STRETCH ;
        STYLE "T"

    @ 1.231,25.833 SAY "Near East Hospital" ;
        FONT "Times New Roman", 24 ;
        STYLE "BT" ;
        COLOR RGB(255,255,255,0,255,255)

    @ 6.385,22.833 GET B ;
        PICTURE "@*HN \<Department" ;
        SIZE 1.769,14.167,0.667 ;
        DEFAULT 1 ;
        FONT "MS Sans Serif", 8 ;
        STYLE "B" ;
        VALID BT('dep') ;
```

MESSAGE 'Show Different Departments.'

@ 9.769,17.833 GET B ;
 PICTURE "@*HN \<Patients" ;
 SIZE 1.769,14.167,0.667 ;
 DEFAULT 1 ;
 FONT "MS Sans Serif", 8 ;
 STYLE "B" ;
 VALID BT('pat') ;
 MESSAGE '.'

@ 13.692,16.833 GET B ;
 PICTURE "@*HN D\<ctors" ;
 SIZE 1.769,14.167,0.667 ;
 DEFAULT 1 ;
 FONT "MS Sans Serif", 8 ;
 STYLE "B" ;
 VALID BT('doc') ;
 MESSAGE '.'

@ 18.308,20.333 GET B ;
 PICTURE "@*HN \<Employees" ;
 SIZE 1.769,14.167,0.667 ;
 DEFAULT 1 ;
 FONT "MS Sans Serif", 8 ;
 STYLE "B" ;
 VALID BT('emp') ;
 MESSAGE '.'

@ 22.615,28.667 GET B ;
 PICTURE "@*HN A\<bout" ;
 SIZE 1.769,14.167,0.667 ;
 DEFAULT 1 ;
 FONT "MS Sans Serif", 8 ;
 STYLE "B" ;
 VALID BT('abt') ;
 MESSAGE '.'

@ 6.385,58.833 GET B ;
 PICTURE "@*HN \<Reception" ;
 SIZE 1.769,14.167,0.667 ;
 DEFAULT 1 ;
 FONT "MS Sans Serif", 8 ;
 STYLE "B" ;
 VALID BT('rec') ;
 MESSAGE '.'

@ 9.692,64.167 GET B ;
PICTURE "@*HN \<Accounting" ;
SIZE 1.769,14.167,0.667 ;
DEFAULT 1 ;
FONT "MS Sans Serif", 8 ;
STYLE "B" ;
VALID BT('acc') ;
MESSAGE '.'

@ 13.692,64.833 GET B ;
PICTURE "@*HN Roo\<ms" ;
SIZE 1.769,14.167,0.667 ;
DEFAULT 1 ;
FONT "MS Sans Serif", 8 ;
STYLE "B" ;
VALID BT('rom') ;
MESSAGE '.'

@ 18.308,61.000 GET B ;
PICTURE "@*HN P\<harmacy" ;
SIZE 1.769,14.167,0.667 ;
DEFAULT 1 ;
FONT "MS Sans Serif", 8 ;
STYLE "B" ;
VALID BT('pha') ;
MESSAGE '.'

@ 22.692,54.000 GET B ;
PICTURE "@*HN \<Close" ;
SIZE 1.769,13.667,0.667 ;
DEFAULT 1 ;
FONT "MS Sans Serif", 8 ;
STYLE "B" ;
VALID bt('EXIT') ;
MESSAGE 'Close screen.'

@ 1.385,22.000 TO 4.154,74.167 ;
PEN 1, 8 ;
STYLE "16" ;
COLOR RGB(0,0,0,,,) ;

@ 5.538,12.000 TO 25.923,84.167 ;
PEN 1, 8 ;
STYLE "16" ;
COLOR RGB(0,0,0,,,) ;

```

* *****
*
* *   WindowsREAD contains clauses from SCREEN main
* *
* *****
*

```

```

*****

```

```

READ CYCLE
PROCEDURE BT
    PARAMETER m.btnname
    DO CASE
CASE m.btnname='dep'
    DO department.prg

CASE m.btnname='pat'
    DO patients.prg

CASE m.btnname='doc'
    DO doctors.prg

CASE m.btnname='emp'
    DO employees.prg

CASE m.btnname='abt'
    DO about.prg

CASE m.btnname='rec'
    DO reception.prg

CASE m.btnname='acc'
    DO password.prg

CASE m.btnname='rom'
    DO rooms.prg

CASE m.btnname='pha'
    DO pharmacy.prg

CASE m.btnname='EXIT'
set sysm to defa
DEACTIVATE WINDOW mmm
RELEASE WINDOW mmm
SET CLOCK OFF

```

```
ENDCASE
RETURN
*****
```

Password Code:

```
CLOSE DATABASES
define window Sam00 at 12,12 size 4,40 system
activate window Sam00
@ 1.308,1 say "Enter the Password:"
@ 1.308,20.550 GET m.ps ;
    SIZE 1.000,18.000 ;
    DEFAULT " " ;
    FONT "Times New Roman", 8 ;
    STYLE "B" ;
    PICTURE "@K XXXXXXXXXXXXXXXXXXXX" ;
    COLOR ,RGB(0,0,0,0,0,0)

read
release window Sam00
show gets
USE pas.dbf
set ORDER TO TAG 'wrđ'
seek pass
*IF FOUND()
LOCATE ALL FOR M.PS=pass

IF FOUND()
    WAIT WINDOW " PASSWORD IS OK " NOWAIT
    DO accounting.PRG

ELSE
    WAIT WINDOW " PASSWORD IS WRONG " NOWAIT
    DO MEXXSS1
ENDIF
*USE
*endif
return

PROCEDURE MEXXSS1
    DEFINE WINDOW _qjn112 ;
        AT 0.000,0.000 ;
        SIZE 18.615,57.667 ;
        TITLE " SEARCHING " ;
        FONT "MS Sans Serif", 8 ;
        FLOAT NOCLOSE MINIMIZE SYSTEM
    MOVE WINDOW _qjn112 CENTER
    ACTIVATE WINDOW _qjn112 NOSHOW
    @ 5.923,7.000 SAY " THE PASSWARD IS WRONG!";
```

```
FONT "Arial", 13 ;  
    STYLE "BT" ;  
    PICTURE "@J" ;  
    COLOR RGB(,,,0,128,128)  
@ 9.923,9.000 SAY "Press ENTER Please! " ;  
FONT "Arial", 12 ;  
    STYLE "BT" ;  
    PICTURE "@J" ;  
    COLOR RGB(,,,0,128,128)  
ACTIVATE WINDOW _qjn112  
    READ  
    RELEASE WINDOW _qjn112  
RETURN
```


Patients Code

```
* *****
*
* * 23/01/01      PATIENT.SPR      01:10:48
*
* *****
*
* * Author's Name
*
* * Copyright (c) 2001 Company Name
* * Address
* * City,  Zip
*
* * Description:
* * This program was automatically generated by GENSCRN.
*
* *****
```

```
* *****
*
* *      PATIENT/Windows Setup Code - SECTION 1
*
* *****
*
```

```
#REGION 1
PRIVATE wzfields,wztalk
IF SET("TALK") = "ON"
    SET TALK OFF
    m.wztalk = "ON"
ELSE
    m.wztalk = "OFF"
ENDIF
m.wzfields=SET('FIELDS')
SET FIELDS OFF
IF m.wztalk = "ON"
    SET TALK ON
ENDIF
```

```

#REGION 0
REGIONAL m.currarea, m.talkstat, m.compstat

IF SET("TALK") = "ON"
    SET TALK OFF
    m.talkstat = "ON"
ELSE
    m.talkstat = "OFF"
ENDIF
m.compstat = SET("COMPATIBLE")
SET COMPATIBLE FOXPLUS

m.rborder = SET("READBORDER")
SET READBORDER ON

m.currarea = SELECT()

* *****
* *
* *   PATIENT/Windows Databases, Indexes, Relations
* *
* *****
*

IF USED("patient")
    SELECT patient
    SET ORDER TO TAG "ddd"
ELSE
    SELECT 0
    USE (LOCFILE("patient.dbf","DBF","Where is patient?"));
    AGAIN ALIAS patient ;
    ORDER TAG "ddd"
ENDIF

* *****
* *
* *   Windows Window definitions
* *
* *****
*

IF NOT WEXIST("_0aulhyxus")
    DEFINE WINDOW _0aulhyxus ;
        AT 0.000, 0.000 ;
        SIZE 25.692,97.167 ;
        TITLE "" ;
        FONT "MS Sans Serif", 8 ;
        STYLE "B" ;

```

```

        FLOAT ;
        CLOSE ;
        MINIMIZE ;
        COLOR RGB(,,,192,192,192)
    MOVE WINDOW _0au1hyxus CENTER
ENDIF

```

```

*      *****
*      *
*      *      PATIENT/Windows Setup Code - SECTION 2
*      *
*      *****
*

```

```

#REGION 1

```

```

#DEFINE C_DBFEMPTY      'Database is empty, add a record?'
#DEFINE C_EDITS          'Please finish your edits.'
#DEFINE C_TOPFILE       'Top of file.'
#DEFINE C_ENDFILE       'End of file.'
#DEFINE C_BRTITLE       'Locate Record'
#DEFINE C_NOLOCK        'Sorry, could not lock record -- try again later.'
#DEFINE C_ECANCEL       'Edits Canceled.'
#DEFINE C_DELREC        'Delete selected record?'
#DEFINE C_NOFEAT        'Feature not available yet.'
#DEFINE C_NOWIZ         'Wizard application is not available.'
#DEFINE C_MAKEREPO      'Creating report with Report Wizard.'
#DEFINE C_NOREPO        'Could not create report.'
#DEFINE C_DELNOTE       'Deleting records...'
#DEFINE C_READONLY      'Table is read-only. No editing allowed.'
#DEFINE C_NOTABLE       'No table selected. Open table or run query.'
#DEFINE C_BADEXP        'Invalid expression.'
#DEFINE C_LOCWIZ        'Locate WIZARD.APP:'
#DEFINE C_MULTITABLE    'You have multiple related tables. Adding records in not
allowed.'

```

```

MOVE WINDOW '_0au1hyxus' CENTER
PRIVATE isediting,isadding,wztblarr
PRIVATE wzoldddelete,wzolderror,wzoldesc
PRIVATE wzalias, tempcurs,wzlastrec
PRIVATE isreadonly,find_drop,is2table

```

```

IF EMPTY(ALIAS())
    WAIT WINDOW C_NOTABLE
    RETURN
ENDIF

```

```

m.wztblarr= "
m.wzalias=SELECT()

```

```

m.isediting=.F.
m.isadding=.F.
m.is2table = .F.
m.wzolddelete=SET('DELETE')
SET DELETED ON
m.tempcurs=SYS(2015) &&used if General field
m.wzlastrec = 1
m.wzolderror=ON('error')
ON ERROR DO wizerrorhandler
wzolddesc=ON('KEY','ESCAPE')
ON KEY LABEL ESCAPE
m.find_drop = IIF(_DOS,0,2)

m.isreadonly=IIF(ISREAD(),.T.,.F.)
IF m.isreadonly
    WAIT WINDOW C_READONLY TIMEOUT 1
ENDIF

IF RECCOUNT()=0 AND !m.isreadonly AND fox_alert(C_DBFEMPTY)
    APPEND BLANK
ENDIF

GOTO TOP
SCATTER MEMVAR MEMO

*      *****
*      *
*      *      PATIENT/Windows Screen Layout
*      *
*      *****
*

#REGION 1
IF WVISIBLE("_0aulhyxus")
    ACTIVATE WINDOW _0aulhyxus SAME
ELSE
    ACTIVATE WINDOW _0aulhyxus NOSHOW
ENDIF
@ 0.385,34.000 SAY "Patients" ;
    FONT "Times New Roman", 16 ;
    STYLE "BT" ;
    COLOR RGB(128,0,0,,,)
@ 2.692,0.000 TO 2.692,96.833 ;
    PEN 2, 8 ;
    STYLE "1"
@ 22.077,0.000 TO 22.077,96.833 ;
    PEN 2, 8 ;
    STYLE "1"
@ 3.692,2.833 SAY "Patient ID:" ;

```



```

SIZE 0.938,13.667 ;
FONT "Times New Roman", 10 ;
STYLE "BT" ;
PICTURE "@J" ;
COLOR RGB(,,,255,255,255)
@ 3.692,18.833 GET m.patint_id ;
SIZE 1.000,16.000 ;
DEFAULT " " ;
FONT "Times New Roman", 8 ;
STYLE "B" ;
PICTURE "@K XXXXXXXXXXXXX" ;
WHEN isediting ;
COLOR ,RGB(0,0,0,255,255,255)
@ 3.692,45.333 SAY "Name:" ;
SIZE 0.938,18.000 ;
FONT "Times New Roman", 10 ;
STYLE "BT" ;
PICTURE "@J" ;
COLOR RGB(,,,255,255,255)
@ 3.769,65.333 GET m.name ;
SIZE 1.000,30.400 ;
DEFAULT " " ;
FONT "Times New Roman", 8 ;
STYLE "B" ;
PICTURE "@ XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX" ;
WHEN SAM1() ;
COLOR ,RGB(0,0,0,255,255,255)
@ 5.769,2.833 SAY "Surname:" ;
SIZE 0.938,13.667 ;
FONT "Times New Roman", 10 ;
STYLE "BT" ;
PICTURE "@J" ;
COLOR RGB(,,,255,255,255)
@ 5.769,18.833 GET m.surname ;
SIZE 1.000,27.400 ;
DEFAULT " " ;
FONT "Times New Roman", 8 ;
STYLE "B" ;
PICTURE "@K XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX" ;
WHEN isediting ;
COLOR ,RGB(0,0,0,255,255,255)
@ 5.846,45.333 SAY "Date of Birth:" ;
SIZE 0.938,18.000 ;
FONT "Times New Roman", 10 ;
STYLE "BT" ;
PICTURE "@J" ;
COLOR RGB(,,,255,255,255)
@ 5.846,65.333 GET m.datebirth ;
SIZE 1.000,9.200 ;
DEFAULT " " ;

```

```

FONT "Times New Roman", 8 ;
STYLE "B" ;
PICTURE "@K" ;
WHEN isediting ;
COLOR ,RGB(0,0,0,255,255,255)
@ 7.846,2.833 SAY "Gender:" ;
SIZE 0.938,13.667 ;
FONT "Times New Roman", 10 ;
STYLE "BT" ;
PICTURE "@J" ;
COLOR RGB(,,,255,255,255)
@ 7.769,18.667 GET m.gender ;
PICTURE "@*RHN M;F" ;
SIZE 1.308,6.167,0.667 ;
DEFAULT 1 ;
FONT "MS Sans Serif", 8 ;
STYLE "BT" ;
WHEN isediting
@ 7.846,45.333 SAY "Address:" ;
SIZE 0.938,18.000 ;
FONT "Times New Roman", 10 ;
STYLE "BT" ;
PICTURE "@J" ;
COLOR RGB(,,,255,255,255)
@ 7.923,65.333 EDIT m.address ;
SIZE 3.000,29.833,0.000 ;
PICTURE "@K
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXX" ;
DEFAULT " " ;
FONT "MS Sans Serif", 8 ;
STYLE "B" ;
SCROLL ;
WHEN isediting ;
COLOR ,RGB(,,,255,255,255)
@ 9.923,2.833 SAY "Job:" ;
SIZE 0.938,13.667 ;
FONT "Times New Roman", 10 ;
STYLE "BT" ;
PICTURE "@J" ;
COLOR RGB(,,,255,255,255)
@ 10.000,18.833 GET m.job ;
SIZE 1.000,27.400 ;
DEFAULT " " ;
FONT "Times New Roman", 8 ;
STYLE "B" ;
PICTURE "@K XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX" ;
WHEN isediting ;
COLOR ,RGB(0,0,0,255,255,255)
@ 12.000,2.833 SAY "Phone:" ;

```

SIZE 0.938,13.667 ;
 FONT "Times New Roman", 10 ;
 STYLE "BT" ;
 PICTURE "@J" ;
 COLOR RGB(,,255,255,255)
 @ 12.077,18.833 GET m.phone ;
 SIZE 1.000,16.000 ;
 DEFAULT " " ;
 FONT "Times New Roman", 8 ;
 STYLE "B" ;
 PICTURE "@K XXXXXXXXXXXX" ;
 WHEN isediting ;
 COLOR ,RGB(0,0,0,255,255,255)
 @ 12.231,45.833 SAY "Nationality:" ;
 SIZE 0.938,18.000 ;
 FONT "Times New Roman", 10 ;
 STYLE "BT" ;
 PICTURE "@J" ;
 COLOR RGB(,,255,255,255)
 @ 12.231,65.833 GET m.nationalit ;
 SIZE 1.000,16.000 ;
 DEFAULT " " ;
 FONT "Times New Roman", 8 ;
 STYLE "B" ;
 PICTURE "@K XXXXXXXXXXXX" ;
 WHEN isediting ;
 COLOR ,RGB(0,0,0,255,255,255)
 @ 14.077,2.833 SAY "Date Of Adm:" ;
 SIZE 0.938,13.667 ;
 FONT "Times New Roman", 10 ;
 STYLE "BT" ;
 PICTURE "@J" ;
 COLOR RGB(,,255,255,255)
 @ 14.154,18.833 GET m.datofadmis ;
 SIZE 1.000,16.000 ;
 DEFAULT " " ;
 FONT "Times New Roman", 8 ;
 STYLE "B" ;
 PICTURE "@K XXXXXXXXXXXX" ;
 WHEN isediting ;
 COLOR ,RGB(0,0,0,255,255,255)
 @ 14.462,45.833 SAY "Cause of Admis:" ;
 SIZE 0.938,18.000 ;
 FONT "Times New Roman", 10 ;
 STYLE "BT" ;
 PICTURE "@J" ;
 COLOR RGB(,,255,255,255)
 @ 14.462,65.833 GET m.causeadmis ;
 SIZE 1.000,28.000 ;
 DEFAULT " " ;

```

FONT "Times New Roman", 8 ;
STYLE "B" ;
PICTURE "@K XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX" ;
WHEN isediting ;
COLOR ,RGB(0,0,0,255,255,255)
@ 16.538,0.333 SAY "Discharge Date:" ;
SIZE 0.938,16.167 ;
FONT "Times New Roman", 10 ;
STYLE "BT" ;
PICTURE "@J" ;
COLOR RGB(,,255,255,255)
@ 16.538,18.833 GET m.outdate ;
SIZE 1.000,16.000 ;
DEFAULT " " ;
FONT "Times New Roman", 8 ;
STYLE "B" ;
PICTURE "@K XXXXXXXXXXXX" ;
WHEN isediting ;
COLOR ,RGB(0,0,0,255,255,255)

@ 18.538,0.333 SAY "Dept Name:" ;
SIZE 0.938,16.167 ;
FONT "Times New Roman", 10 ;
STYLE "BT" ;
PICTURE "@J" ;
COLOR RGB(,,255,255,255)
@ 18.538,18.833 GET m.dep_name ;
SIZE 1.000,35.000 ;
DEFAULT " " ;
FONT "Times New Roman", 8 ;
STYLE "B" ;
PICTURE "@K
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXX" ;
WHEN isediting ;
COLOR ,RGB(0,0,0,255,255,255)
@ 20.538,0.333 SAY "Room No:" ;
SIZE 0.938,16.167 ;
FONT "Times New Roman", 10 ;
STYLE "BT" ;
PICTURE "@J" ;
COLOR RGB(,,255,255,255)
@ 20.538,18.833 GET m.room_no ;
SIZE 1.000,16.000 ;
DEFAULT " " ;
FONT "Times New Roman", 8 ;
STYLE "B" ;
PICTURE "@K XXXXXXXXXXXX" ;
WHEN isediting ;
COLOR ,RGB(0,0,0,255,255,255)

```



```

@ 20.538,30.333 SAY "Bed No:" ;
    SIZE 0.938,16.167 ;
    FONT "Times New Roman", 10 ;
    STYLE "BT" ;
    PICTURE "@J" ;
    COLOR RGB(,,255,255,255)
@ 20.538,50.833 GET m.bed_no ;
    SIZE 1.000,16.000 ;
    DEFAULT " " ;
    FONT "Times New Roman", 8 ;
    STYLE "B" ;
    PICTURE "@K XXXXXXXXXXXX" ;
    WHEN isediting ;
    COLOR ,RGB(0,0,0,255,255,255)
@ 23.000,7.833 GET m.top_btn ;
    PICTURE "@*HN \<Top" ;
    SIZE 1.769,7.833,0.667 ;
    DEFAULT 1 ;
    FONT "MS Sans Serif", 8 ;
    STYLE "B" ;
    VALID btn_val("TOP") ;
    MESSAGE 'Go to first record.'
@ 23.000,15.833 GET m.prev_btn ;
    PICTURE "@*HN \<Prev" ;
    SIZE 1.769,7.833,0.667 ;
    DEFAULT 1 ;
    FONT "MS Sans Serif", 8 ;
    STYLE "B" ;
    VALID btn_val("PREV") ;
    MESSAGE 'Go to previous record.'
@ 23.000,23.833 GET m.next_btn ;
    PICTURE "@*HN \<Next" ;
    SIZE 1.769,7.833,0.667 ;
    DEFAULT 1 ;
    FONT "MS Sans Serif", 8 ;
    STYLE "B" ;
    VALID btn_val("NEXT") ;
    MESSAGE 'Go to next record.'
@ 23.000,31.833 GET m.end_btn ;
    PICTURE "@*HN \<End" ;
    SIZE 1.769,7.833,0.667 ;
    DEFAULT 1 ;
    FONT "MS Sans Serif", 8 ;
    STYLE "B" ;
    VALID btn_val("END") ;
    MESSAGE 'Go to last record.'
@ 23.000,39.833 GET m.loc_btn ;
    PICTURE "@*HN \<Locate" ;
    SIZE 1.769,7.833,0.667 ;
    DEFAULT 1 ;

```

```

FONT "MS Sans Serif", 8 ;
STYLE "B" ;
VALID btn_val('LOCATE') ;
MESSAGE 'Locate a record.'
@ 23.000,47.833 GET m.add_btn ;
PICTURE "@*HN \<Add" ;
SIZE 1.769,7.833,0.667 ;
DEFAULT 1 ;
FONT "MS Sans Serif", 8 ;
STYLE "B" ;
VALID btn_val('ADD') ;
MESSAGE 'Add a new record.'
@ 23.000,55.833 GET m.edit_btn ;
PICTURE "@*HN Ed\<it" ;
SIZE 1.769,7.833,0.667 ;
DEFAULT 1 ;
FONT "MS Sans Serif", 8 ;
STYLE "B" ;
VALID btn_val('EDIT') ;
MESSAGE 'Edit current record.'
@ 23.000,63.833 GET m.del_btn ;
PICTURE "@*HN \<Delete" ;
SIZE 1.769,7.833,0.667 ;
DEFAULT 1 ;
FONT "MS Sans Serif", 8 ;
STYLE "B" ;
VALID btn_val('DELETE') ;
MESSAGE 'Delete current record.'
@ 23.000,71.833 GET m.prnt_btn ;
PICTURE "@*HN P\<rint" ;
SIZE 1.769,7.833,0.667 ;
DEFAULT 1 ;
FONT "MS Sans Serif", 8 ;
STYLE "B" ;
VALID btn_val('PRINT') ;
MESSAGE 'Print report.'
@ 23.000,79.833 GET m.exit_btn ;
PICTURE "@*HN \<Close" ;
SIZE 1.769,7.833,0.667 ;
DEFAULT 1 ;
FONT "MS Sans Serif", 8 ;
STYLE "B" ;
VALID btn_val('EXIT') ;
MESSAGE 'Close screen.'

@ 16.615,60.000 GET m.dal_btn;
PICTURE "@*HN Daily Checkup" ;
SIZE 1.769,15.167,0.667 ;
DEFAULT 1 ;
FONT "MS Sans Serif", 8 ;

```

```
STYLE "B" ;
VALID btn_val('DAL')
```

```
@ 16.615,78.000 GET m.his_btn;
  PICTURE "@*HN History" ;
  SIZE 1.769,12.167,0.667 ;
  DEFAULT 1 ;
  FONT "MS Sans Serif", 8 ;
  STYLE "B" ;
  VALID btn_val('HISTORY')
```

```
IF NOT WVISIBLE("_0au1hyxus")
  ACTIVATE WINDOW _0au1hyxus
ENDIF
```

```
* *****
* *
* *   WindowsREAD contains clauses from SCREEN patient
* *
* *****
*
```

```
READ CYCLE ;
  ACTIVATE READACT() ;
  DEACTIVATE READDEAC() ;
  NOLOCK
```

```
RELEASE WINDOW _0au1hyxus
```

```
* *****
* *
* *   Windows Closing Databases
* *
* *****
*
```

```
IF USED("patient")
  SELECT patient
  USE
ENDIF
```

```
SELECT (m.currarea)
```

```
#REGION 0
```

```
SET READBORDER &rborder
```

```
IF m.talkstat = "ON"
```

```
    SET TALK ON
```

```
ENDIF
```

```
IF m.compstat = "ON"
```

```
    SET COMPATIBLE ON
```

```
ENDIF
```

```
* *****
*
*      *      PATIENT/Windows Cleanup Code
*
* *****
*
```

```
#REGION 1
```

```
SET DELETED &wzolddelete
```

```
SET FIELDS &wzfields
```

```
ON ERROR &wzolderror
```

```
ON KEY LABEL ESCAPE &wzoldesc
```

```
DO CASE
```

```
CASE _DOS AND SET('DISPLAY')='VGA25'
```

```
    @24,0 CLEAR TO 24,79
```

```
CASE _DOS AND SET('DISPLAY')='VGA50'
```

```
    @49,0 CLEAR TO 49,79
```

```
CASE _DOS
```

```
    @24,0 CLEAR TO 24,79
```

```
ENDCASE
```

```
****Procedures****
```

```
* *****
*
*      *      PATIENT/Windows Supporting Procedures and Functions
*
* *****
*
```

```
#REGION 1
```

```
PROCEDURE readdeac
```

```
    IF isediting
```

```
        ACTIVATE WINDOW '_0au1hyxus'
```

```
        WAIT WINDOW C_EDITS NOWAIT
```

```
    ENDIF
```

```
    IF !WVISIBLE(WOUTPUT())
```

```
        CLEAR READ
```

```
        RETURN .T.
```



```
ENDIF
RETURN .F.
```

```
PROCEDURE readact
  IF !isediting
    SELECT (m.wzalias)
    SHOW GETS
  ENDIF
  DO REFRESH
RETURN
```

```
PROCEDURE wizerrorhandler
  * This very simple error handler is primarily intended
  * to trap for General field OLE errors which may occur
  * during editing from the MODIFY GENERAL window.
  WAIT WINDOW message()
RETURN
```

```
PROCEDURE printrec
  PRIVATE sOldError,wizfname,saverec,savearea,tmpcurs,tmpstr
  PRIVATE prnt_btn,p_recs,p_output,pr_out,pr_record
  STORE 1 TO p_recs,p_output
  STORE 0 TO prnt_btn
  STORE RECNO() TO saverec
  m.sOldError=ON('error')
  DO pdialog
  IF m.prnt_btn = 2
    RETURN
  ENDIF
  IF !FILE(ALIAS()+'.FRX')
    m.wizfname=SYS(2004)+'WIZARDS\'+'WIZARD.APP'
    IF !FILE(m.wizfname)
      ON ERROR *
      m.wizfname=LOCFILE('WIZARD.APP','APP',C_LOCWIZ)
      ON ERROR &sOldError
      IF !'WIZARD.APP'$UPPER(m.wizfname)
        WAIT WINDOW C_NOWIZ
        RETURN
      ENDIF
    ENDIF
  WAIT WINDOW C_MAKEREPO NOWAIT
  m.savearea=SELECT()
  m.tmpcurs='_'+LEFT(SYS(3),7)
  CREATE CURSOR (m.tmpcurs) (comment m)
  m.tmpstr = '* LAYOUT = COLUMNAR'+CHR(13)+CHR(10)
  INSERT INTO (m.tmpcurs) VALUES(m.tmpstr)
  SELECT (m.savearea)
  DO (m.wizfname) WITH
  ', 'WZ_QREPO','NOSCRN/CREATE',ALIAS(),m.tmpcurs
```

```

        USE IN (m.tmpcurs)
WAIT CLEAR
        IF !FILE(ALIAS()+'.FRX') &&wizard could not create report
        WAIT WINDOW C_NOREPO
        RETURN
        ENDIF
ENDIF

m.pr_out=IIF(m.p_output=1,'TO PRINT NOCONSOLE','PREVIEW')
m.pr_record=IIF(m.p_recs=1,'NEXT 1','ALL')
REPORT FORM (ALIAS()) &pr_out &pr_record
GO m.saverec
RETURN

```

```

PROCEDURE BTN_VAL
PARAMETER m.btnname
DO CASE
CASE m.btnname='TOP'
    GO TOP
    WAIT WINDOW C_TOPFILE NOWAIT
CASE m.btnname='PREV'
    IF !BOF()
        SKIP -1
    ENDIF
    IF BOF()
        WAIT WINDOW C_TOPFILE NOWAIT
        GO TOP
    ENDIF

CASE m.btnname='HIS'
DO D:\HOS\HISTORY.PRG

CASE m.btnname='DAL'
DO D:\HOS\DAYCHKUP.PRG

CASE m.btnname='NEXT'
    IF !EOF()
        SKIP 1
    ENDIF
    IF EOF()
        WAIT WINDOW C_ENDFILE NOWAIT
        GO BOTTOM
    ENDIF
CASE m.btnname='END'
    GO BOTTOM
    WAIT WINDOW C_ENDFILE NOWAIT
CASE m.btnname='LOCATE'
    DO loc_dlog

```

```

CASE m.btnname='ADD' AND !isediting &&add record
    isediting=.T.
    isadding=.T.
    =edithand('ADD')
    _curobj=1
    DO refresh
    SHOW GETS
    RETURN
CASE m.btnname='EDIT' AND !isediting &&edit record
    IF EOF() OR BOF()
        WAIT WINDOW C_ENDFILE NOWAIT
        RETURN
    ENDIF
    IF RLOCK()
        isediting=.T.
        _curobj=1
        DO refresh
        RETURN
    ELSE
        WAIT WINDOW C_NOLOCK
    ENDIF
CASE m.btnname='EDIT' AND isediting &&save record
    IF isadding
        =edithand('SAVE')
    ELSE
        GATHER MEMVAR MEMO
    ENDIF
    UNLOCK
    isediting=.F.
    isadding=.F.
    DO refresh
CASE m.btnname='DELETE' AND isediting &&cancel record
    IF isadding
        =edithand('CANCEL')
    ENDIF
    isediting=.F.
    isadding=.F.
    UNLOCK
    WAIT WINDOW C_ECANCEL NOWAIT
    DO refresh
CASE m.btnname='DELETE'
    IF EOF() OR BOF()
        WAIT WINDOW C_ENDFILE NOWAIT
        RETURN
    ENDIF
    IF fox_alert(C_DELREC)
        DELETE
        IF !EOF() AND DELETED()
            SKIP 1
        ENDIF

```

```

        IF EOF()
            WAIT WINDOW C_ENDFILE NOWAIT
            GO BOTTOM
        ENDIF
    ENDIF
CASE m.btnname='PRINT'
    DO printrec
    RETURN
CASE m.btnname='EXIT'
    m.bailout=.T. &&this is needed if used with FoxApp
    CLEAR READ
    RETURN
ENDCASE
SCATTER MEMVAR MEMO
SHOW GETS
RETURN

```

PROCEDURE REFRESH

```

DO CASE
CASE m.isreadonly AND RECCOUNT()=0
    SHOW GETS DISABLE
    SHOW GET exit_btn ENABLE
CASE m.isreadonly
    SHOW GET add_btn DISABLE
    SHOW GET del_btn DISABLE
    SHOW GET edit_btn DISABLE
CASE (RECCOUNT()=0 OR EOF()) AND !m.isediting
    SHOW GETS DISABLE
    SHOW GET add_btn ENABLE
    SHOW GET exit_btn ENABLE
CASE m.isediting
    SHOW GET find_drop DISABLE
    SHOW GET top_btn DISABLE
    SHOW GET prev_btn DISABLE
    SHOW GET loc_btn DISABLE
    SHOW GET next_btn DISABLE
    SHOW GET end_btn DISABLE
    SHOW GET add_btn DISABLE
    SHOW GET prnt_btn DISABLE
    SHOW GET exit_btn DISABLE
    SHOW GET edit_btn,1 PROMPT "\<Save"
    SHOW GET del_btn,1 PROMPT "\<Cancel"
    ON KEY LABEL ESCAPE DO BTN_VAL WITH 'DELETE'
    RETURN
OTHERWISE
    SHOW GET edit_btn,1 PROMPT "Ed\<it"
    SHOW GET del_btn,1 PROMPT "\<Delete"
    SHOW GETS ENABLE
ENDCASE

```



```

IF m.is2table
    SHOW GET add_btn DISABLE
ENDIF
ON KEY LABEL ESCAPE
RETURN

```

```

PROCEDURE edithand
    PARAMETER m.paction
    * procedure handles edits
    DO CASE
    CASE m.paction = 'ADD'
        SCATTER MEMVAR MEMO BLANK
    CASE m.paction = 'SAVE'
        INSERT INTO (ALIAS()) FROM MEMVAR
    CASE m.paction = 'CANCEL'
        * nothing here
    ENDCASE
RETURN

```

```

PROCEDURE fox_alert
    PARAMETER wzalrtmess
    PRIVATE alrtbtn
    m.alrtbtn=2
    DEFINE WINDOW _qec1ij2t7 AT 0,0 SIZE 8,50 ;
    FONT "MS Sans Serif",10 STYLE 'B' ;
    FLOAT NOCLOSE NOMINIMIZE DOUBLE TITLE WTITLE()
    MOVE WINDOW _qec1ij2t7 CENTER
    ACTIVATE WINDOW _qec1ij2t7 NOSHOW
    @ 2,(50-txtwidth(wzalrtmess))/2 SAY wzalrtmess;
    FONT "MS Sans Serif", 10 STYLE "B"
    @ 6,18 GET m.alrtbtn ;
    PICTURE "@*HT \<OK;\?!\<Cancel" ;
    SIZE 1.769,8.667,1.333 ;
    FONT "MS Sans Serif", 8 STYLE "B"
    ACTIVATE WINDOW _qec1ij2t7
    READ CYCLE MODAL
    RELEASE WINDOW _qec1ij2t7
RETURN m.alrtbtn=1

```

```

PROCEDURE pdialog
    DEFINE WINDOW _qjn12zbvh ;
    AT 0.000,0.000 ;
    SIZE 13.231,54.800 ;
    TITLE "Microsoft FoxPro" ;
    FONT "MS Sans Serif", 8 ;
    FLOAT NOCLOSE MINIMIZE SYSTEM
    MOVE WINDOW _qjn12zbvh CENTER
    ACTIVATE WINDOW _qjn12zbvh NOSHOW

```

```

@ 2.846,33.600 SAY "Output:" ;
    FONT "MS Sans Serif", 8 ;
    STYLE "BT"
@ 2.846,4.800 SAY "Print:" ;
    FONT "MS Sans Serif", 8 ;
    STYLE "BT"
@ 4.692,7.200 GET m.p_recs ;
    PICTURE "@*RVN \<Current Record;\<All Records" ;
    SIZE 1.308,18.500,0.308 ;
    DEFAULT 1 ;
    FONT "MS Sans Serif", 8 ;
    STYLE "BT"
@ 4.692,36.000 GET m.p_output ;
    PICTURE "@*RVN \<Printer;Pre\<view" ;
    SIZE 1.308,12.000,0.308 ;
    DEFAULT 1 ;
    FONT "MS Sans Serif", 8 ;
    STYLE "BT"
@ 10.154,16.600 GET m.prnt_btn ;
    PICTURE "@*HT P\<rint;Ca\<ncel" ;
    SIZE 1.769,8.667,0.667 ;
    DEFAULT 1 ;
    FONT "MS Sans Serif", 8 ;
    STYLE "B"
ACTIVATE WINDOW _qjn12zbvh
READ CYCLE MODAL
RELEASE WINDOW _qjn12zbvh
RETURN

PROCEDURE loc_dlog
    PRIVATE gfields,i
    DEFINE WINDOW wzlocate FROM 1,1 TO 20,40;
        SYSTEM GROW CLOSE ZOOM FLOAT FONT "MS Sans Serif",8
    MOVE WINDOW wzlocate CENTER
    m.gfields=SET('FIELDS',2)
    IF !EMPTY(RELATION(1))
        SET FIELDS ON
        IF m.gfields # 'GLOBAL'
            SET FIELDS GLOBAL
        ENDIF
        IF EMPTY(FLDLIST())
            m.i=1
            DO WHILE !EMPTY(OBJVAR(m.i))
                IF ATC('M.',OBJVAR(m.i))=0
                    SET FIELDS TO (OBJVAR(m.i))
                ENDIF
                m.i = m.i + 1
            ENDDO
        ENDIF
    ENDIF

```

```

ENDIF
BROWSE WINDOW wzlocate NOEDIT NODELETE ;
    NOMENU TITLE C_BRTITLE
SET FIELDS &gfields
SET FIELDS OFF
RELEASE WINDOW wzlocate

```

```

RETURN

```

```

FUNCTION SAM1
CLOSE DATABASES
USE (LOCFILE("clinic.dbf","DBF","Where is clinic?"));
    SHARE;
    AGAIN ALIAS clinic ;
    ORDER TAG "id"
SEEK m.patint_id
IF in="in"
*IF FOUND(IN="IN")
*DO CASE
*    situation="in"
m.name=name
m.surname=surname
m.datebirth=birthdate
m.nationalit=nationalit
m.job=job
m.datofadmis=date

ELSE
    WAIT WINDOW " THIS PATIENT IS OUT " NOWAIT

ENDIF

```

```

show gets
CLOSE DATABASES
USE (LOCFILE("patient.dbf","dbf","WHERE IS patient ?"));
share ;
AGAIN ALIAS patient ;
ORDER TAG "ddd"

RETURN

```