

CASE TOOL FOR DATABASE SYSTEMS



BY:

NC SAANIA FEROZE KHAN

**Project Report for partial fulfillment of the requirement of
MCS/NUST for the award of the B.E Degree in
Software Engineering**

**Department of Computer Science
Military College of Signals
Rawalpindi**

September, 2000

DECLARATION:

No portion of the work presented in this dissertation has been submitted in support of another award or qualification either in this institution or elsewhere.

ACKNOWLEDGMENTS:

For their help with this project, I would like to thank all my friends and instructors, in particular:

Gen. Shua-ul-Qamar, Lec. Asad-ullah- Khan, and Col. Waseem Suleman.

Furthermore, I would like to thank Dr. Riaz for his guidance and encouragement.

ABSTRACT:

Many methodologies exist that can make the database design process easier for both designers and users. They vary from general design technique described in the literature to commercial products whose aim is to automate the design process. CASE (Computer-aided Software Engineering) packages provide the engineer with the ability to automate software development.

CASE Tools are used in this project, in order to reduce the manual effort involved in mapping ERDs onto relational model and generating the SQL code. The project is aimed at taking input from the user in the form of ERD and then generating SQL code, after going through all the stages of mapping and normalization automatically.

CONTENTS	Pages
<u>Chapter 1:</u>	
CASE TOOLS AND DATABASE SYSTEMS IN BRIEF.....	2
1.1 <u>Introduction</u>	2
1.2 <u>CASE Tools</u>	2
1.2.1 Building Blocks for CASE.....	3
1.2.2 Categories.....	4
1.3 <u>Database Systems</u>	4
1.3.1 Data and Information.....	5
1.3.2 Levels of Data.....	5
1.3.2.1 Level-1.....	5
1.3.2.2 Level-2.....	6
1.3.2.3 Level-3.....	7
1.3.3 Three Level Architecture.....	7
1.3.3.1 External Views.....	8.
1.3.3.2 Conceptual Model.....	9
1.3.3.3 Internal Model.....	9
1.4 <u>Logical Data Models</u>	10
1.4.1 Semantic Data Models.....	10
1.4.1.1 Entity Relationship Model.....	11
1.4.2 Record Based Models.....	11
1.4.2.1 Network Model.....	11
1.4.2.2 Hierarchical Model.....	11
1.4.2.3 Relational Model.....	12
1.5 <u>Mapping ER model to Relational Model</u>	12
1.6 <u>Normalization</u>	13
1.7 <u>Computerization of Database Design</u>	13
1.8 <u>The Project</u>	13

Chapter 2:

DATABASE DESIGN CONCEPTS.....	15
2.1 <u>Database Design</u>	15
2.2 <u>Database Design Process</u>	16
2.2.1 Functional Requirements.....	16
2.2.2 Physical Constraints.....	17
2.2.3 Information Level Design Process.....	17
2.2.4 Physical Level Design Process.....	18
2.3 <u>Database Design Goals</u>	18
2.4 <u>Database Design Methodology</u>	18
2.5 <u>Entity Relationship Diagrams</u>	19
2.5.1 Entities.....	20
2.5.2 Attributes.....	20
2.5.2.1 Candidate and Primary Keys.....	20
2.5.3 Relationships.....	21
2.5.3.1 Degree of Relationships.....	21
2.5.3.2 Cardinalities in Relationships.....	21
2.6 <u>Relational Model</u>	22
2.6.1 Relations.....	23
2.6.2 Properties of Relations.....	23
2.7 <u>Transforming ERDs to Relations</u>	24
2.7.1 Represent Entities.....	24
2.7.2 Represent Relationships.....	24
2.7.2.1 Binary 1:N Relations.....	25
2.7.2.2 Binary N:M Relations.....	25
2.7.2.3 Unary Relations.....	26
2.8 <u>Normalization</u>	26
2.8.1 Steps in Normalization.....	26
2.8.2 Dependencies.....	27
2.8.2.1 Functional Dependence.....	27
2.8.2.2 Partial Functional Dependence.....	28
2.8.2.3 Transitive Dependence.....	28

2.8.3	Basic Normal Forms.....	28
2.8.3.1	First Normal Form.....	28
2.8.3.2	Second Normal Form.....	29
2.8.3.3	Third Normal Form.....	29
2.8.4	Additional Normal Forms.....	29
2.8.5	<u>Merging Relations</u>	29

Chapter 3:

	PROJECT DESIGN.....	31
3.1	<u>Object-Oriented Data Model</u>	31
3.2	<u>Objects Identified</u>	31
3.3	<u>Detailed Design</u>	33
3.4	<u>Attributes Identified</u>	34
3.5	<u>Complete Object-Oriented Design</u>	35
3.6	<u>Class Definitions</u>	36

Chapter 4

	FLOW OF CONTROL.....	45
4.1	<u>Behavioral Diagrams</u>	45
4.1.1	Sequence Diagrams.....	45
4.1.1.1	Notation.....	46
4.2	<u>Explanation</u>	46
4.2.1	Entity.....	47
4.2.2	Attribute.....	47
4.2.3	End Points.....	49
4.2.4	Relationships.....	50
4.2.5	2NF.....	51
4.2.6	3NF.....	52
4.3	<u>Complete Sequence Diagram</u>	53

Chapter 5

	IMPLEMENTATION.....	54
--	---------------------	----

5.1 <u>Steps in Implementation</u>	54
5.1.1 Drawing ERDs.....	54
5.1.2 Mapping.....	58
5.1.3 Normalization.....	59
<u>Chapter 6</u>	
RESULTS AND RECOMMENDATIONS.....	64
6.1 <u>Problems Encountered</u>	64
6.2 <u>Results</u>	64
6.3 <u>Conclusions</u>	66
6.4 <u>Recommendations</u>	66
6.4.1 Weak/Strong Entities.....	66
6.4.2 Unary/Ternary Relationships.....	67
6.4.3 Attributes with Relationships.....	67
6.4.4 Aggregation/Generalization.....	67
6.4.5 Mapping.....	68
6.4.6 Normalization Beyond 3NF.....	69
SUMMARY OF THE PROJECT.....	71
REFERENCES.....	72

CHAPTER 1

CASE TOOLS AND DATABASE SYSTEMS

IN BRIEF

1.1. INTRODUCTION:

Computer Aided Software Engineering (CASE) is one of the largest approaches to solving the problems of software application development quickly. CASE provides automated support for the process of software engineering, thereby speeding up the development process and increasing efficiency, while at the same time retaining the benefits of quality.

Databases represent centralization. A poorly designed database may fail to provide the required information, or may provide outdated, flawed, or contradictory information; there is therefore a need for a comprehensive tool for engineering Databases.

This chapter gives an introduction to Case Tools and database systems. After that types of logical models and methods of improving logic design process are explained.

1.2 CASE TOOLS: -

CASE is the automation of software engineering methods and practices, with the use of automated tools to support the creation of computer applications. Organizations today invest enormous resources in CASE technologies with the hope of gaining significant increase in the programmer productivity. It is now established that most of the CASE projects are coded in less time than the non-CASE projects.

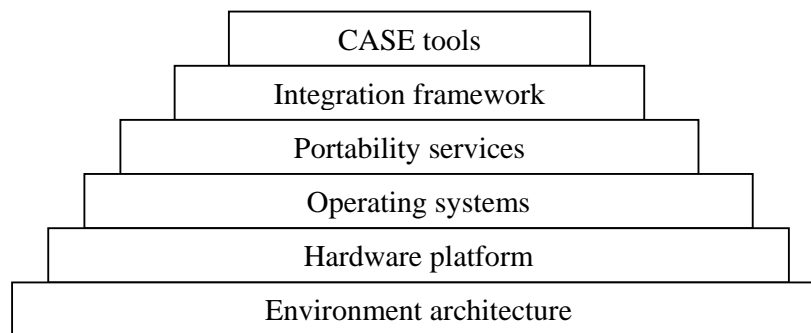
CASE tools add to the software engineer's toolbox. CASE provides the engineer with the ability to automate manual activities and to improve engineering insight. Like computer-

aided engineering and design tools that are used by engineers in other disciplines, CASE tools help to ensure that quality is designed in before the product is built.

Quality and efficiency are important features of an application, so that fewer errors occur while running the application. Software engineering methods have been developed in an attempt to alleviate both the efficiency and quality problems of development. CASE is an attempt towards removing the tradeoff between quality and efficiency.

1.2.1 BUILDING BLOCKS FOR CASE:

The building blocks for CASE are illustrated in the following figure:



Each building block forms a foundation for the next, with the tools sitting at the top of the heap. However, the foundation for effective CASE environment has relatively little to do with software engineering tools themselves. Rather, successful environments for software engineering are built on *environment architecture* that encompasses appropriate hardware and systems software. In addition, the environment architecture must consider the human work patterns that are applied during the software engineering process. The environment architecture, composed of *hardware platform* and *operating system* support (including networking and databases management software), lays the groundwork for CASE. But the CASE environment itself demands other building blocks. A set of *portability services* provides

a bridge between CASE tools and their integration framework and the environment architecture.

1.2.2 CATAGORIES: -

Of the various CASE Tool categories that are currently in use there are two main types:

- a) CASE- Workbenches are integrated tools that automate the software life cycle phase of analysis, design and implementation. This integration enables these tools to check for inconsistencies and incompleteness between the various phases of the life cycle.
- b) CASE tool-kits are the collection of tools that focus on providing support for one particular phase of software development.

Whether one implements a number of CASE tool-kits or a single CASE workbench, it has gained wide acceptance that CASE tools should support a method. Basically, CASE products that do not support a proven method only help to develop poor systems far faster than without the use of CASE. However, a problem for CASE manufacturers is the large number of methods available, and which of these to automate.

CASE tools provide many productivity benefits, whether they are stand-alone, tool-kits, or sophisticated workbenches. However to ensure that CASE adequately improves “*the time a software takes to market*” it must be implemented correctly.

1.3 DATABASE SYSTEMS:

Databases are used in thousands of organizations, ranging from large government agencies to small businesses. Databases often replace earlier file processing systems, in which departments have control over their own data. The resulting data redundancy leads to inconsistency from data entry errors or inconsistent updates. In an integrated database environment, all data is kept in a single large repository called the database and is managed by a database administrator (DBA). All access to the database is through the database

management system (DBMS), in a software package that sets up storage structures, loads data, provides access for programs and on-line users, formats retrieved data, hides certain data, does updates, controls concurrency, and does backup and recovery for the database. Data in a database is integrated, self- describing and shared concurrently by many users. The DBMS provides a program interface and a user interface for on-line queries that are expressed in the query language of the particular DBMS.

1.3.1 DATA AND INFORMATION:

The term data refers to the base facts recorded in the database. They may be items about people, events or concepts. Information is processed data that is in a form that is useful for making decisions. Information is derived from the stored data by rearranging, selecting, combining, summarizing, or performing other operations on the data.

1.3.2 LEVELS OF DATA:

When the term data is discussed, it is important to distinguish between objects in the real world, the structure of the database, and the data stored in the database. These are actually three levels of discussion or abstraction to be considered.

1.3.2.1 LEVEL – 1:

The first level is the real world or reality. On this level the organization for which the database is designed is discussed. In the realm of reality, we identify entities, which are persons, places, events, objects or concepts about which we collect data. We group similar entities into entity sets. Each entity has certain attributes, which are characteristics or properties that describe the entity and that the organization considers important. Some entities may have relationships or associations between them.

As an organization functions in the real world, it is impossible to obtain information needed for decision-making by direct observation of reality. There is just too much detail involved for us to keep track of all the facts we need. Instead a model of the organization is

developed. The database should be designed to be a faithful model or representation of the organization and its operations. Every entity should be represented, along with its attributes and the relationships in which it participates. To keep track of the facts about entities and the changes made in real world, we need to develop a model that allows not only the representation of the basic entities, attributes, and relationships, but also allows us to make changes that mirror the changes, that is, reality.

1.3.2.2 LEVEL – 2:

The structure of the database, called the logical model of the database, forms the second level. At this level we talk about the metadata or data about data. For each entity set in the real world, we have a record type in the logical model of the database. A record type contains several data item types, each of which represents an attribute of an entity. A data item is in fact, the smallest unit of stored data.

Other words sometimes used for data item are data element field or attribute . Generally , a field means a set of adjacent bytes identified as the physical location for a data item , so it has a more physical meaning than the term data item . Attribute usually refers to a characteristic of an entity in the real world , but since there is a correspondence between attributes and data items , the two words are interchanged .

Data items are sometimes grouped together to form data aggregates, which are named groups of data items within a record. Data aggregates, which are named groups of data items within a record. Data aggregates allow us to refer to the group of data items as a whole or to the individual items in the group . A record is a named collection of related data items and/or data aggregates .

Information about the logical structure of the database is stored in a data dictionary or data directory. This repository of information contains the database schema , a written description of the logical structure of the database . It contains description of the record types , data item types , and data aggregates in the database as well as other information . For data aggregates , the dictionary / Directory would list the components . The data dictionary / directory is usually a database about the database . However , data dictionaries usually do

much more than simply store the schema . They often control database documentation , and for some systems , are actively involved in all database accesses .

1.3.2.3 LEVEL - 3:

The third level is concerned with actual data in the database itself . It consists of data instances or occurrences . For each entity in the real world , there is an occurrence of a corresponding record in the database . Similarly , there are many instances of each of the data item types the correspond to attributes . A file (sometimes called a data set) is a named collection of record occurrences . Usually a file contains all occurrences of one record type.

These levels of data lead to the three – level architecture concept. When we discuss a database , we need some means of describing different aspects of it's structure .

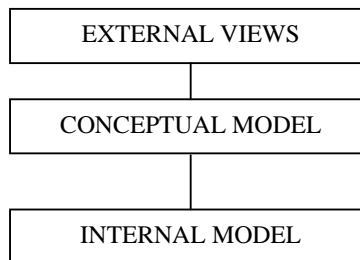
1.3.3 THREE -LEVEL ARCHITECTURE:

Databases infact can be viewed at three levels of abstraction. The levels form a three – level architecture and are depicted by three schemas , or three models . The models refer to the permanent structure of the database , not to the data that is stored at any given moment . This permanent structure is called the intention of the database , or the database schema , while the information store at given moment is called an extension of the database , or a database instance .

The purpose of the three – level architecture is to separate the way the database is physically represented from the way the users think about it. There are several reasons why this separation is desirable.

- Different users need different views of the same data .
- The way a particular user needs to see the data may change over time .
- Users should not have to deal with the complexities of the database storage structures.
- Database structure should be unaffected by changes to the physical aspects of storage, such as changes to storage devices.

The way users think about data is called the external view level. The way the operating system and the DBMS see the data is the internal or physical level . There is a middle level that provides both the mapping and the desired independence between the external and physical levels . This is called the conceptual level.



1.3.3.1 EXTERNAL VIEWS:

The external level consists of many different external views or external models of the database. Each user has the model of the real world represented in a form that is suitable for the user. A particular user interacts with only certain entities in the real world and is interested in only some of their attributes and relationships. Therefore, that user's view will contain only information about those aspects of the real world. Other entities or other attributes or relationships not of interest may actually be represented in the database, but the user will be unaware of them. Views may even include data combined or calculated from several records. An external record is a record as seen by a particular user, a part of his or her external views.

Thus, an external view is actually a collection of external records. The external views are described in external schemas, which are written in data definition language (DDL). Each user's schema gives a complete description of each type of external record that appears in that user's views. The DBMS uses the external schema to create a user interface, which are both a facility and a barrier. An individual user sees the database through this interface. It also acts as a boundary below which the user is not permitted to see. It hides the conceptual, internal and physical details from the user.

1.3.3.2 CONCEPTUAL OR LOGICAL MODEL:

The middle level, in the three level architecture is the logical or conceptual level. This model includes the entire information structure of the database, as seen by the database administrator (DBA). All the entities, their attributes, and their relationships are represented in the logical model. The data record types that represent the entities, the data item types that represent their attributes, the relationships that will be stored, the constraints on the data, semantic information about the data meanings, and security and integrity information are all part of the conceptual model.

The conceptual model supports the external views, in that any data available to any user must be present in or derivable from the conceptual model. The conceptual model is relatively constant. When the DBA originally designs it, he or she tries to determine present and future information needs and attempts to develop a lasting model of the organization.

The conceptual schema is a complete description of the information content of the database, including every record type with all its fields. The DBMS uses the conceptual schema to create the logical record interface, which is a boundary below which everything is invisible to the conceptual level and which defines and creates the working environment for the conceptual level. The conceptual level is actually a collection of logical records.

1.3.3.3 INTERNAL OR PHYSICAL MODEL:

The internal, or physical, level covers the physical implementation of the database. It includes the data structures and file organizations used to store data on physical storage devices. The DBMS chosen determines, to a large extent, what structures are available. It works with the operating system access methods to lay out the data on the storage devices, build the indexes, and /or set the pointers that will be used for data retrieval. The internal schema, written in DDL is a complete description of the internal model. It includes such items as how data is represented, how records are sequenced, what indexes exist, what pointers exist, and what hashing schemes, if any, are used. The stored record interface is the boundary between the

physical level, for which the operating system may be responsible, and the internal level, for which the DBMS is responsible.

The operating system creates the physical record Interface, which is a boundary below which storage details such as exactly what portion of what track contains what data are hidden.

This project deals mainly with the conceptual or logical data model. The conceptual model should be a complete and accurate representation of the workings of the organization and its environment. If the conceptual model is correct, it should support any external view needed. The conceptual model should be the most permanent part of the database architecture. External views can and should be changed. The internal model should change when new devices or new techniques for representing data and improving performance are developed. However, the conceptual model represents the permanent structure of the data resource of the organization. It should require changes only when the actual workings of the organization or the environment change.

1.4 LOGICAL DATA MODELS:

Developing the logical model is the most challenging, interesting, and rewarding part of database design. During the logical design process, there may be many errors and several false starts. Like many other problem-solving processes, logical database design is a matter of intuition, guided by knowledge. Logical models are classified as semantic models and record-based models.

1.4.1 SEMANTIC DATA MODELS:

These models are used to describe the conceptual and external levels of data and are independent of the internal and physical aspects. In addition to specifying what is to be represented in the database, they attempt to incorporate some meanings or semantic aspects of data, such as explicit representation of objects, attributes, and relationships, categorization of objects, abstractions, and explicit data constraints. These include the Entity-Relationship Model, among others.

1-4-1-1 ENTITY-RELATIONSHIP MODEL:

This model is widely used for logical design and is used for this project as well because record-based models do not provide much semantic information about the database. It is based on identifying real-world objects called entities which are described by their attributes and which are connected by relationships.

Entities are defined as any object that exists and is distinguishable from other objects. The attributes describe the entities and distinguish them from one another. Relationships may themselves have descriptive attributes. The E-R model also provides utility of expressing constraints, or restrictions, on the entities or relationships.

One of the most useful attractive features of E-R model is that it provides a graphical method for depicting the logical structure of the database. E-R diagrams contain symbols for entities, attributes and relationships.

1-4-2 RECORD-BASED MODELS:

These models are used to describe the external, conceptual, and to some extent, the internal levels of the database. They allow the designer to develop and specify the logical structure and provide some options for implementation of the design. These are further classified as “relational”, “hierarchical”, and “network”.

1-4-2-1 NETWORK MODEL:

A network model database is perceived by the user as a collection of record types and relationships between these record types. Such a structure is called a network, and it's from this that the model takes its name.

1-4-2-2 HIERARCHICAL MODEL:

A hierarchical model database is perceived by a user as a collection of hierarchies, or trees. A hierarchy is really a network with an added restriction; no box can have more than one arrow entering the box.

1-4-2-3 RELATIONAL MODEL:

The relational model uses the theory of relations from mathematics and adapts it for use in database theory. In the relational model, both entities and relationships are represented by relations, which are physically represented as tables or two-dimensional arrays, and the attributes as columns of those tables.

The basic structure of the relational model is simple, making it easy to understand on an intuitive level. It allows separation of the logical and physical levels, so that logical design can be performed without considering storage structures.

A relational database consists of any number of relations. We can represent relation schemes by giving the name of the relation, followed by the attribute names in parentheses. Normally, primary key is underlined. When an attribute appears in more than one relation, its appearance usually represents a relationship or interconnection between tuples of the two relations. These common attributes play an important role in performing data manipulation.

1-5 MAPPING AN E-R MODEL TO A RELATIONAL MODEL: -

An E-R diagram can be converted to a relational model fairly easily. The entity sets represented by rectangles become relations represented by tables. The table name is the same as the entity name, which is the name written inside the rectangle. For strong entity sets, the attributes represented by ovals become attributes of the relation, or column headings of the table. Weak entity sets are also represented by tables, but may require additional attributes.

Relationship sets can be translated directly into tables as well. Although not explicitly represented on the E-R diagram, it is understood that a relationship set has the primary keys of

the associated entities as attributes. To analyze the design of a relational database to see whether it is good or bad we need to normalize the relations.

1.6 NORMALIZATION:

“Normalization” means putting a relation into a higher normal form. A relation is in a specific normal form if it satisfies the set of requirements or constraints for that form. All of the normal forms are nested in that each satisfies the constraints of the previous one but is a better form because each eliminates flaws found in the previous form.

1.7 COMPUTERIZATION OF DATABASE DESIGN:

No matter how straightforward the application of a methodology, the process of database design can be very time-consuming, particularly for large, complex systems. Some form of computer assistance for the design process is clearly desirable.

CASE (Computer Aided Software Engineering) tools are becoming increasingly popular. These tools assist in various phases of the systems development life cycle. Such tools include components that relate to database design. These tools provide:

1. Assistance in the creation and modification of entity-relationship or data structure diagrams.
2. Some assistance in the normalization process.
3. Generation of the SQL code.

1.8 THE PROJECT:

The aim of the project is to draw ERDs quickly and efficiently, by using Case Tools. This reduces much of the manual effort involved in mapping ERDs onto Relational model and its normalization. Much work has already been done in the past on database systems. This

project is however aimed at helping users in designing error free databases. As case tools are being used, thus this project also provides some knowledge of how software applications with sophisticated user interfaces are developed.

CHAPTER 2

DATABASE DESIGN CONCEPTS

2.1 DATABASE DESIGN:

Database design is really a two step design process. In the first step, user requirements are gathered together and a database is designed to meet these requirements as cleanly as possible. This step is called information-level design and is independent of any individual DBMS.

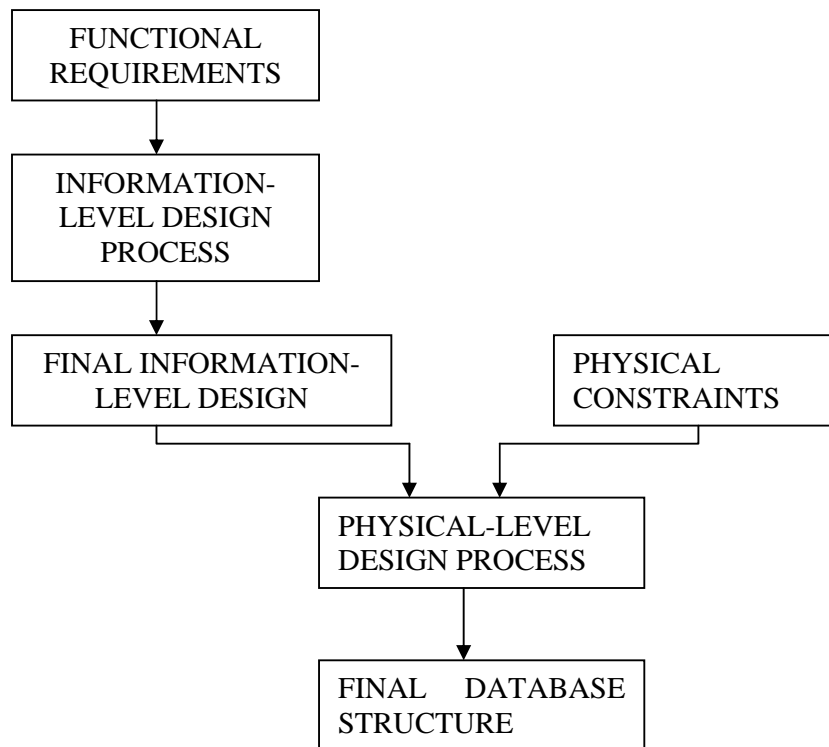
In the second step, the information-level design is transformed into a design for the specific DBMS that will be used to implement the system in question. In this step, which is called physical-level design, we are concerned with the characteristics of the specific DBMS that will be used. We are also concerned with the adequate performance of the system.

Both of these database design steps are critical. A poor effort made with regard to the information-level design is extremely difficult to counteract when it comes to the physical-level design. On the other hand, even an excellent information-level design is not enough to avoid a poorly performing system if the physical-level design is not done well.

Various approaches or methodologies have been proposed for the information-level database design. Among them are the canonical schema, the entity-relationship model, and the semantic data models. This project adopts a methodology that depends heavily on the relational model.

2.2 DATABASE DESIGN PROCESS:

The overall design process is represented as:



For any design process one expects the user requirements to be as complete as possible. Specifically this means that the user requirements for the system should address both the functional requirements and the physical constraints of the target system.

2.1.1 FUNCTIONAL REQUIREMENTS:

The functional requirements must include:

- All reports that must be produced.

- All inquiries that must be supported.
- All other outputs that must be sent to other systems or to external destinations.
- All update transactions that must be processed.
- All calculations that must be performed.
- All restrictions that the system must enforce.
- All synonyms that are used for each attribute. (Synonyms are different names used by different users for the same attribute).

2.2.2 PHYSICAL CONSTRAINTS:

When physical design process is considered, we also need the user requirements to provide information about processing volumes and performance measurements. We call these volumes and measurements physical constraints. These include:

- The number of occurrences of each type of entity.
- The frequency with which each report will be printed.
- The length in number of lines for each report.
- The response-time requirements for each query.
- The response-time requirements for each update transaction.
- The special security constraints that define who can access which data and in what way.

Taking these user requirements as input, the information-level and physical-level design processes should produce a database design for a specific DBMS that supports these requirements and performs in an acceptable manner.

2.2.3 INFORMATION-LEVEL DESIGN PROCESS:

Information-level design process does not entail the physical constraints from the user requirements. Instead, the information-level design process is based on all other user requirements. This process results in a logical design that cleanly supports the user requirements and is independent of the characteristics of any individual DBMS.

2.2.4 PHYSICAL-LEVEL DESIGN PROCESS:

Physical-level design process utilizes this logical design, the physical constraints from the user requirements and information concerning the particular DBMS involved producing the final database structure.

2.3 DATABASE DESIGN GOALS:

Database design is the process that takes a set of user requirements as input and produces database structures capable of supporting these requirements as output.

We must keep certain goals in mind as we proceed through the information-level and physical-level design processes. The information-level design process must result in a design that is complete. That is, all user functional requirements must be satisfied by this design. The design itself should enforce as many of the requirements as possible, rather than force programs to do so. Hence the ultimate goal during this step is a clean, redundancy-free design.

The physical-level design process must result in a DBMS-specific design that is basically concerned with system efficiency in terms of storage space, processing time and response time. The goals in this step can conflict with the goals of the information-level design process.

2.4 DATABASE DESIGN METHODOLOGY:

The methodology adopted here involves representing user views, refining them to eliminate any problems and then merging them into a cumulative design. For each user view we need to complete the following steps:

- Represent the user view in the form of an ERD.
- Mapping this ERD onto a relational model to get a collection of relations.
- Normalize these relations

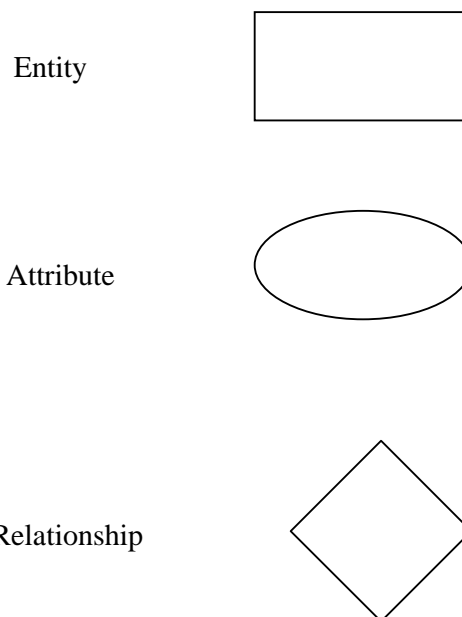
- Represent all keys.
- Determine any special restrictions.
- Merge the result of the previous steps to get SQL code.

Obtaining information on user views is a critical task, but not an easy one. The only source for obtaining the information is the users who will be involved in the new system. One of the ways of getting the required information is by using ERDs (Entity Relationship Diagrams).

2.5 ENTITY RELATIONSHIP DIAGRAMS:

An Entity-relationship data model is a detailed, logical representation of the data for an organization or for a business area. The E-R model is expressed in terms of entities in the business environment, the relationships among those entities, and the attributes of both the entities and their relationships. An E-R model is normally expressed as an Entity-Relationship Diagram (or E-R Diagram), which is a graphical representation of an E-R model.

The major constructs of E-R models are entities, relationships, and associated attributes. The notations used to represent these are:



2.5.1 ENTITIES:

An entity is defined as a person, place, object, event, or concept in the user environment about which the organization wishes to maintain data. There is an important distinction between entity types and entity instances. An entity type is a collection of entities that share common properties or characteristics. Each entity type in an E-R model is given a name. Since the name represents a class, it is singular. In an E-R diagram the name is placed inside the box representing the entity.

An entity instance is a single occurrence of an entity type. An entity type is described once in a database, while many instances of that entity type may be represented by data stored in the database.

2.5.2 ATTRIBUTES:

Each entity type has a set of attributes associated with it. An attribute is a property or characteristic of an entity that is of interest to the organization (relationships may also have attributes). In E-R diagram, we represent an attribute by placing its name in an ellipse with a line connecting it to its associated entity

2-5.2.1 CANDIDATE KEYS AND PRIMARY KEYS:

Every entity type must have an attribute or set of attributes that uniquely identifies each instance and clearly distinguishes that instance from other instances of the same type. A candidate key is an attribute (or a combination of attributes) that uniquely identifies each instance of an entity type.

Some entities may have more than one candidate key. If there is more than one candidate key the designer must choose one of the candidate keys as the primary key. A primary key is a candidate key that has been selected as the identifier for an entity type. The criteria for selecting primary keys, as suggested by some experts, is as:

- 1- Choose a candidate key that will not change its value over the life of each instance of the entity type.
- 2- Choose a candidate key such that for each instance of the entity, the attribute is guaranteed to have valid values and not be null. If the candidate key is a combination of two or more attributes, make sure that all parts of the key will have valid values.
- 3- Avoid the use of so-called intelligent keys, whose structure indicates classification, locations, and so on.
- 4- Consider substituting single-attribute surrogate keys for large composite keys.

The primary key could be highlighted by underlining it on the E-R diagram.

2.5.3 RELATIONSHIPS:

Relationships are the glue that holds together the various components of an E-R model. A relationship is an association between the instances of one or more entity types that is of interest to the organization. A diamond shape in the E-R diagram indicates a relationship.

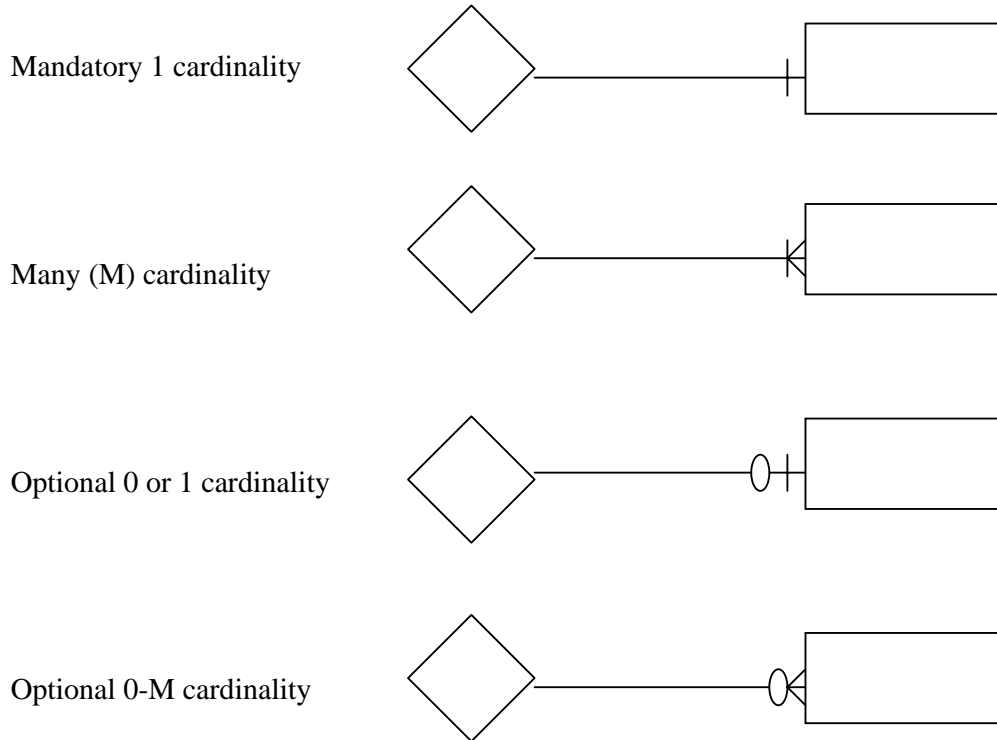
2.5.3.1 DEGREE OF A RELATIONSHIP:

The degree of a relationship is the number of entity types that participate in that relationship. The three most common relationships in E-R model are unary (degree one), binary (degree two), and ternary (degree three). Higher degree relationships are possible but they are rarely encountered during practice.

2.5.3.2 CARDINALITIES IN RELATIONSHIPS:

Suppose there are two entity types, A and B, that are connected in a relationship. The cardinality of a relationship is the number of instances of entity B that can be associated with each instance of entity A.

The minimum cardinality of a relationship is the minimum number of instances of entity B that may be associated with each instance of entity A. the maximum cardinality is the maximum number of instances. It is possible for the maximum cardinality to be a fixed number, not an arbitrary ‘many’ value.



The E-R Diagram is mapped onto the relational model for further data manipulation.

2.6 RELATIONAL MODEL:

The relational database model represents data in the form of tables or relations. The relational model is based on the mathematical theory, and therefore has a solid theoretical foundation. The relational model consists of the following three components

- 1- Data structure: Data are organized in the form of tables or relations.

- 2- Data manipulation: Powerful operations (such as those incorporated in the SQL language) are used to manipulate data stored in the relations.
- 3- Data integrity: Facilities are included to specify business rules that maintain the integrity of data when they are manipulated.

2.6.1 RELATIONS:

A relation is named, two-dimensional table of data. Each relation (or table) consists of a set of named columns and an arbitrary number of unnamed rows. Each column in a relation corresponds to an attribute of the relation. Each row of the relation corresponds to a record that contains data values for an entity.

We can express the structure of a relation by a shorthand notation in which the name of the relation is followed (in parenthesis) by the names of the attributes in the relation.

2.6.2 PROPERTIES OF RELATIONS:

Although relations are defined as two-dimensional tables of data; however, not all tables are relations. Relations have several properties that distinguish them from nonrelational tables. These properties are defined as:

- 1- An entry at the intersection of each row and column is atomic (or single-valued). There can be no multivalued attributes or repeating groups in a relation. An attribute that can have more than one value for each entity instance is termed as a multivalued attribute.
- 2-In a relation, all entries in a given column are drawn from the same domain. Where domain is a definition of the types and ranges of values that attributes may assume.
- 3- No two rows in a relation are identical. Uniqueness in a relation is guaranteed by the designation of a primary key for each relation.
- 4- The sequence of columns (left to right) is insignificant. Hence the columns of a relation can be interchanged without changing the meaning or use of the relation. As a consequence of this property, there is no hidden meaning implied by the order in which columns occur.

Another consequence is that columns of a table can be stored in any arbitrary sequence, and users can also retrieve columns in any sequence.

- 5- As with columns, the rows of a relation may be interchanged or stored in any sequence.

2.7 TRANSFORMING E-R DIAGRAM TO RELATIONS:

This transformation consists of four steps: represent entities and then represent relationships. The relations that result from this transformation can then be normalized using the techniques described later. The redundant relations could then be removed by merging these relations.

2.7.1 REPRESENT ENTITIES:

Each entity type in an E-R diagram is transformed into a relation. The primary key (or the identifier) of the entity type becomes the primary key of the corresponding relation. Make sure that this key satisfies the following two properties of a candidate key for a relation described earlier:

- 1- The value of the key must uniquely identify every row of the relation.
- 2- The key should be nonredundant; that is, no attribute in the key can be deleted without destroying its unique identification.

Each nonkey attribute (or descriptor) of the entity type becomes a nonkey attribute of the relation. The relations that are formed from the entity type may be modified as relationships are represented.

2.7.2 REPRESENT RELATIONSHIPS:

The procedure for representing relationships depends on both the degree of the relationship (unary, binary, ternary) and the cardinalities of the relationship.

2.7.2.1 BINARY 1:N RELATIONSHIPS:

A binary one-to-many relationship in an E-R diagram is represented by adding the primary key attribute of the entity on the one-side of the relationship, as a foreign key in the relation that is on the many-side of the relationship. A foreign key is an attribute or a combination of attributes of an entity that is the primary key of another entity.

A special case of the above rule applies for a binary one-to-one relationship between two entities A and B. In this case, the relationship can be represented by:

- 1- Adding the primary key of A as a foreign key of B.
- 2- Adding the primary key of B as a foreign key of A.
- 3- Both of the above.

2.7.2.2 BINARY M: N RELATIONSHIP:

Suppose that there is a binary many-to-many relationship between two entity types A and B. For such a relationship, we create a separate relation C. The primary key of this relation is a composite key consisting of the primary key for each of the two entities in the relationship. Any nonkey attributes that are associated with the M: N relationship are included with the relation C.

Occasionally, the relation created from an M: N relationship requires a primary key that includes more than just the primary keys from the two related relations.

In some cases, there may be an n-ary relationship among three or more entities. In such cases, we create a separate relation that has as a primary key the composite of the primary keys of each of the participating entities (plus any necessary additional key elements). This rule is a simple generalization of the rule for a binary M:N relationship.

2.7.2.3 UNARY RELATIONSHIPS:

For a unary relationship, the entity type is modeled as a relation. The primary key of that relation is the same as for the entity type. Then a foreign key is added to the relation that references the primary key values.

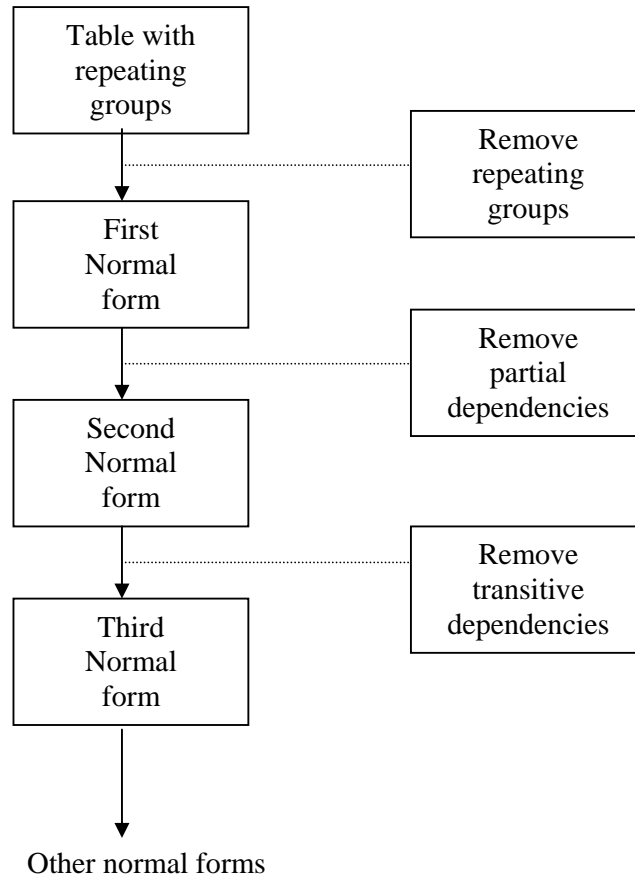
A relation that contains a minimum amount of redundancy and allows users to insert, modify and delete the rows in a table without errors or inconsistencies is said to be a well-structured relation. In order to get well-structured relations we need to normalize them.

2.8 NORMALIZATION:

Normalization is the process for converting complex data structures into simple, stable data structures.

2.8.1 STEPS IN NORMALIZATION:

Normalization is often accomplished in stages, each of which corresponds to a normal form. A normal form is a state of a relation that can be determined by applying simple rules regarding dependencies (or relationships between attributes) to that relation.



In order to understand the concept of normalization the basic idea behind dependencies should be well understood.

2.8.2 DEPENDENCIES:

2.8.2.1 FUNCTIONAL DEPENDENCE:

Normalization is based on the analysis of functional dependence. A functional dependency is a particular relationship between two attributes. For any relation R, attribute B is functionally dependent on attribute A if, for every valid instance of A, that value of A uniquely determines the value of B. The attribute B is said to be the determinant.

2.8.2.2 PARTIAL FUNCTIONAL DEPENDENCE:

A dependency in which one or more nonkey attributes are functionally dependent on part (but not all) of the primary key. The partial functional dependency creates redundancy in the relation, which results in anomalies of various types (insertion, deletion and modification anomalies).

2.8.2.3 TRANSITIVE DEPENDENCY:

A functional dependency between two (or more) nonkey attributes in a relation. Anomalies can also arise due to transitive dependencies.

2.8.3 THE BASIC NORMAL FORMS:

After examining functional and transitive dependencies the next step is the normalization of relations.

2.8.3.1 FIRST NORMAL FORM:

A relation is in first normal form (1NF) if it contains no repeating groups. This idea emerges from the basic property of a relation, that is, that the value at the intersection of each row and column is atomic. Thus a table that contains multivalued attributes or repeating groups is not a relation.

A table with repeating groups is converted to a relation in first normal form by extending the data in each column to fill cells that are empty because of the repeating group structure.

2.8.3.2 SECOND NORMAL FORM:

A relation is in second normal form (2NF) if it is in first normal form and every nonkey attribute is fully functionally dependent on the primary key. A relation that is in first normal form will be in second normal form if any one of the following conditions apply.

- 1- The primary key consists of only one attribute
- 2- No nonkey attributes exist in the relation.
- 3- Every nonkey attribute is functionally dependent on the full set of primary key attributes.

A relation could be converted into second normal form by forming a new relation that consists of the nonkey attributes dependent on a part of the primary key along with that part of the primary key. So the original relation is left with the composite primary key and the nonkey attributes that are dependent on all of the primary key.

2.8.3.3 THIRD NORMAL FORM

A relation is in third normal form if it is in second normal form and no transitive dependencies exist. In order to remove transitive dependency we form a new relation consisting of the nonkey attributes, one of which is dependent on the other. In this case the nonkey attribute on which the other depends forms the primary key. This attribute (the primary key) is also included in the original relation in order to remove anomalies.

2.8.4 ADDITIONAL NORMAL FORMS

Relations in third normal form are sufficient for most practical database applications. However other normal forms also exist that are used to remove all types of anomalies.

2.9 MERGING RELATIONS:

If normalized relations have been created from a number of separate E-R diagrams and other user views, some of the relations may be redundant; that is, they might refer to the same entities. If so, we should merge those relations to remove the redundancy.

Most people agree that a relational database is perceived by its users as a collection of tables in which all data relationships are represented by common values, not links. A relational database management system (RDBMS), then, is a data management system that uses this view of data. To provide some direction for the development of RDBMSs, the American Standards Institute (ANSI) and the International Organization for Standardization have approved a standard for the SQL relational query language.

CHAPTER 3

PROJECT DESIGN

3.1 OBJECT-ORIENTED DATA MODEL:

The project is designed using object-oriented data model, in which the entities, attributes, their relationship etc. are treated as objects.

An object is a structure that encapsulates attributes and method that operate on those objects. The state of an object is expressed in the values of the attributes of the object. The behavior of an object is expressed by a set of methods that operate on its attributes.

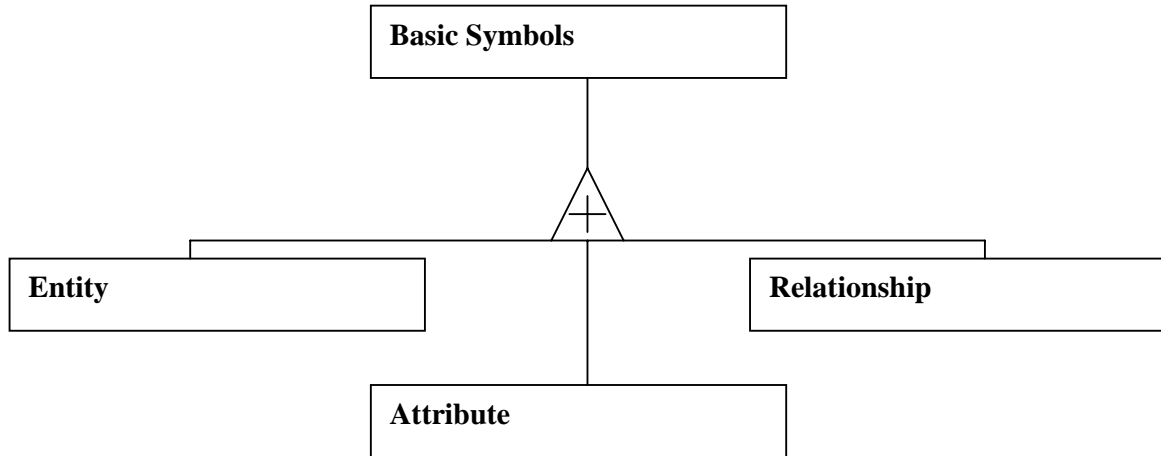
3.1.1 NOTATION:

Classes are represented by rectangles which either bear only the name of the class (in bold), or show attributes and operations as well. In the second case, the three rubrics-- class name, attributes, operations-- are divided from each other by a horizontal line. Class names begin with an upper case letter and are singular nouns (collective classes or similar may be in the plural form, if required).

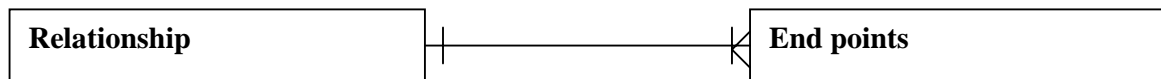
The objects defined in the project and their corresponding attributes and methods are as:

3.2 OBJECTS IDENTIFIED:

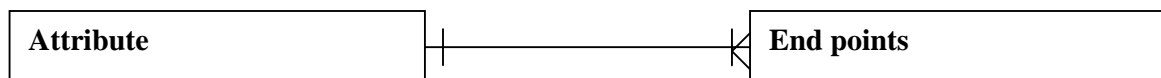
- The entities, attributes and relationships are all derived from the class Basic Symbols. The features that are common in all the symbols drawn are included in the class Basic Symbols



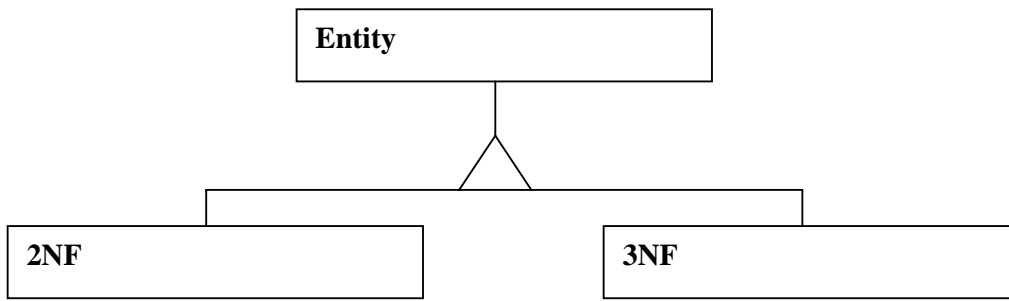
- For each relationship in order to generate error messages the symbols at the end need to be known (Because relationships can only be shown between entities).



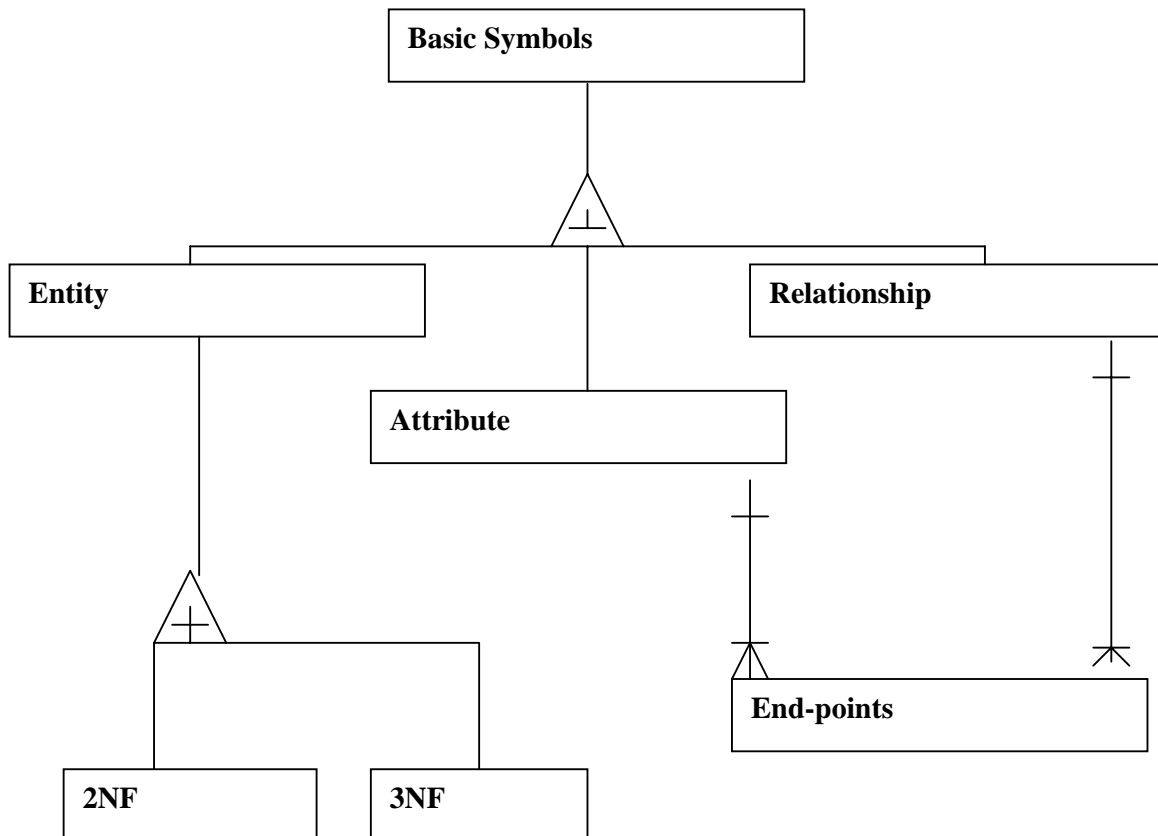
- For each Attribute End points are required, in order to know the entity to which it is going to be attached.



- The Relations obtained after normalization to 2NF are stored in an object which has features in common with the class Entity, namely the attribute 'name', and the linked list that stores information about the attributes attached to the entity.
- Similarly the Relations in 3NF also have features in common with the class Entity.

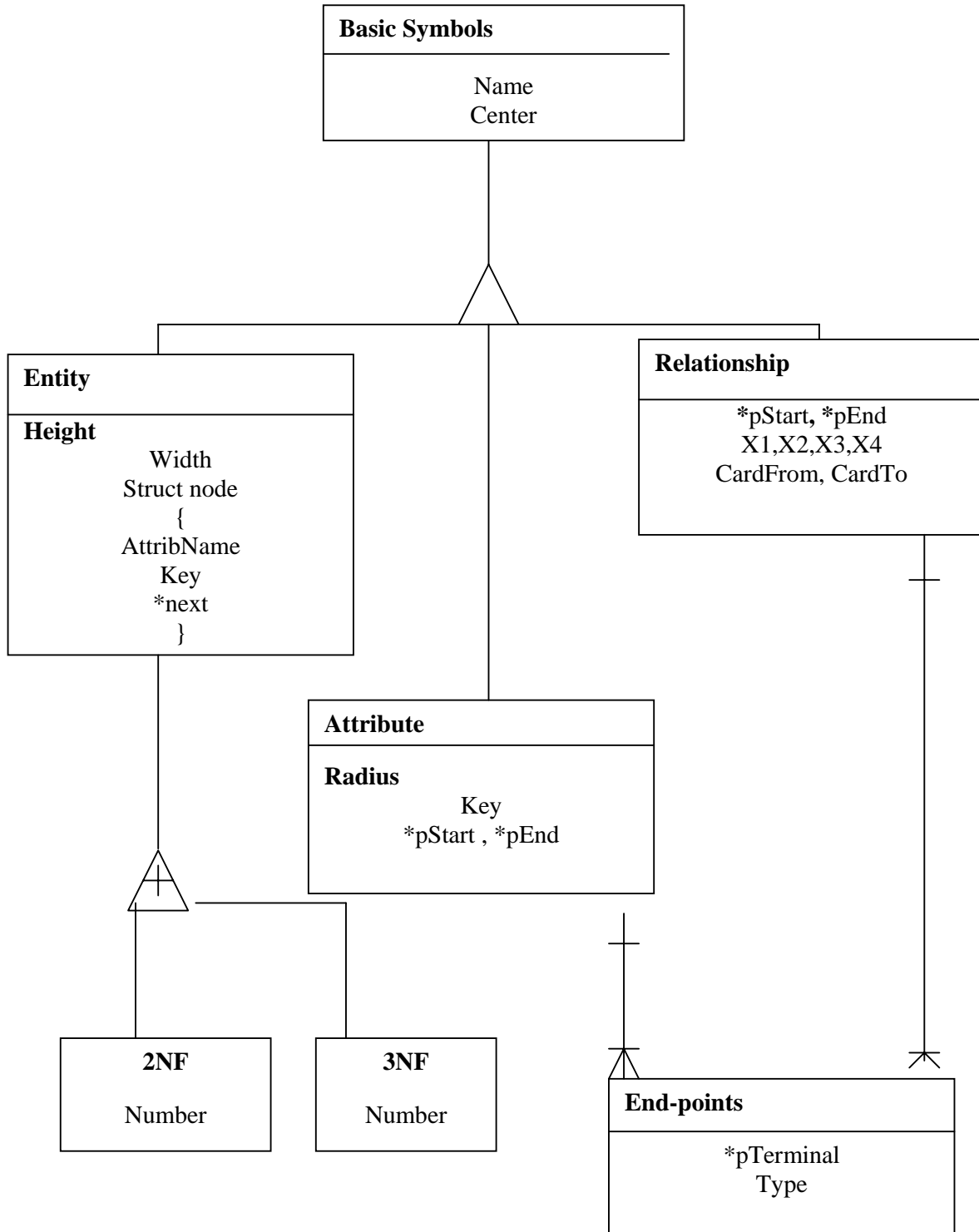


3.3 DETAILED DESIGN:

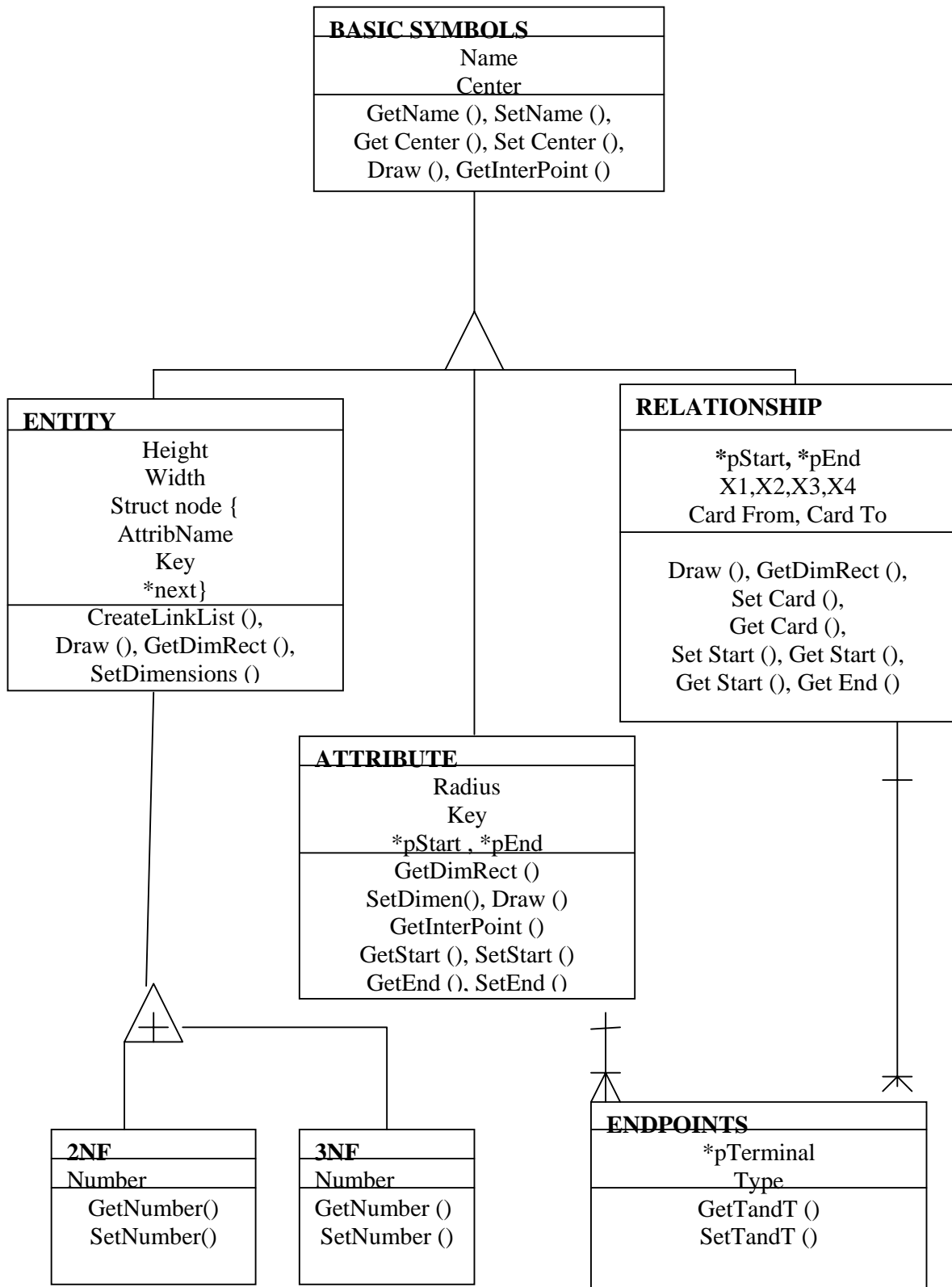


3.4 IDENTIFYING ATTRIBUTES:

The next step is related to identification of attributes for each object. Each object with its related attributes is indicated as:



3.5 COMPLETE OBJECT-ORIENTED DESIGN:



3.6 CLASS DEFINITIONS:

Class name: Basic Symbols

Definition: A set of symbols used to represent entities, relationships and attributes in an ERD.

Relationships: Derived from class CObject.

Attributes:

- Name {text}

A string of characters used to identify each symbol.

- Center {text}

A number computed to calculate screen coordinates of the symbol drawn.

Public Methods:

- Draw

Purpose: Used to draw symbols on screen.

Arguments: CDC *PDC

Return Values: none

- GetDimRect

Purpose: to define a bounding rectangle (The basis for each symbol drawn)

Arguments: none

Return Values: a rectangle

- SetName

Purpose: to set the value of the attribute 'name'

Arguments: a string of characters

Return Values: none

- GetName

Purpose: to get the value of attribute name

Arguments: none

Return Values: a string of characters

- SetCenter

Purpose: To set the value of attribute 'center'

Arguments: a point

Return Values: none

- GetCenter

Purpose: to get the value of protected member variable center

Arguments: none

Return Values: a Point (representing the center)

- GetIntersectionPoint

Purpose: To find the point of intersection of the center and any of the bounding lines of the symbol drawn.

Arguments: Two Points

Return Values: Point of intersection.

BASIC SYMBOLS
Name Center
GetName (), SetName (), Get Center (), Set Center (), Draw (), GetInterPoint ()

Class name: Entity.

Definition: Used to draw the symbol for entity in the ERD.

Relationships: Derived from class Basic Symbols.

Attributes:

- Height {text}

A number representing the height of the symbol drawn.

- Width {text}

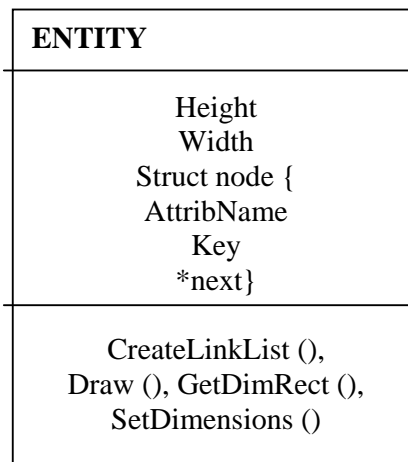
A number used to represent the width of the symbol drawn..

- Node {structure}

A structure used to store the names and key values of the attributes attached to the Entity, in the form of a linked list.

Public Methods:

- Draw
Purpose: Overriding function, Used the draw the symbol of Entity on screen
Arguments: CDC *pDC (i-e a pointer to the device context object)
Return Values: none
- GetDimRect
Purpose: To get the bounding rectangle of the symbol of Entity.
Arguments: none
Return Values: a Rectangle
- SetDimensions
Purpose: To Set the values for the protected members height and width.
Arguments: height, width
Return Values: none
- CreateLinkList
Purpose: To create a new node in the linked list, and assign values to the info field of the linked list.
Arguments: Name of the Attribute, Value of key
Return Values: none



Class name: Attribute

Definition: Used to draw the symbol of an attribute, and store its values.

Relationships: Derived from class Basic Symbols.

Attributes:

- Radius {text}

A number used to determine the radius of the elliptical attribute symbol

- Key {BOOL}

A number that is TRUE if the key is primary and FALSE if it is not.

- *pStart {pointer}

An instance of the class End Points. Represents the type of symbol at the start of the line drawn from attribute to entity

- *pEnd {pointer}

An instance of the class End Points. Represents the type of symbol at the end of the line drawn from attribute to entity

Public Methods:

- GetDimRect

Purpose: Overriding function used to compute the bounding rectangle of the elliptical symbol.

Arguments: none

Return Values: coordinates of a rectangle.

- Draw

Purpose: overriding function. Used to draw the symbol of attribute

Arguments: CDC *pDC

Return Values: none

- SetKeyValues

Purpose: To set the value of the protected member variable key.

Arguments: value of key

Return Values: none

- SetStart

Purpose: To set the value of *pStart

Arguments: pointer to the class Basic symbols and number to represent the type.

Return Values: none

- GetStart

Purpose: to get pointer to the starting symbol

Arguments: none

Return Values: pointer to the starting symbol

- SetEnd

Purpose: To set the value of *pEnd

Arguments: pointer to the class Basic symbols and number to represent the type.

Return Values: none

- GetEnd

Purpose: to get pointer to the ending symbol

Arguments: none

Return Values: pointer to the class Basic Symbols

- GetIntersectionPoint _____

Purpose: Overriding function used to find the intersection point.

Arguments: Two points

Return Values: a point

ATTRIBUTE
Radius Key *pStart , *pEnd
GetDimRect () SetDimen(), Draw () GetInterPoint () GetStart (), SetStart () GetEnd (), SetEnd ()

Class name: Relationship

Definition: Used to store the values of the symbol relationship and draw the symbol.

Relationships: Derived from class Basic Symbols.

Attributes:

- CardFrom {text}

A number used to store the cardinality of the strong entity.

- CardTo {text}

A number to store the value of cardinality of the weak entity

- X1,X2,X3,X4 {point}

Four points used to represent the four vertices of the diamond symbol

- *pStart {pointer}

An instance of the class End Points. Represents the type of symbol at the start of the line drawn from attribute to entity

- *pEnd {pointer}

An instance of the class End Points. Represents the type of symbol at the end of the line drawn from attribute to entity

Public Methods:

- Draw

Purpose: overriding function used to draw the symbol of diamond and to connect the two entities with a line.

Arguments: CDC *pDC

Return Values: none

- SetCard

Purpose: To set the values of the member functions cardfrom and cardto.

Arguments: numbers CardFrom and CardTo

Return Values: none

- SetStart

Purpose: To set the value of *pStart

Arguments: pointer to the class Basic symbols and number to represent the type.

Return Values: none

- GetStart

Purpose: to get pointer to the starting symbol

Arguments: none

Return Values: pointer to the starting symbol

- SetEnd

Purpose: To set the value of *pEnd

Arguments: pointer to the class Basic symbols and number to represent the type.

Return Values: none

- GetEnd

Purpose: to get pointer to the ending symbol

Arguments: none

Return Value: pointer to the class Basic symbols

RELATIONSHIP
<p>*pStart, *pEnd X1,X2,X3,X4 Card From, Card To</p>
<p>Draw (), GetDimRect (), Set Card (), Get Card (), Set Start (), Get Start (), Get Start (), Get End ()</p>

Class name: End points

Definition: A class to obtain the type of the symbol at the end of the line to be drawn.

Relationships: Collaboration with attribute and relationship

Attributes:

- *pTerminal {pointer}

Pointer to the Basic symbols.

- Type {text}

A number of type enum used to represent the type of symbol.

Public Methods:

- GetTandT:

Purpose: to get the value of the protected members

Arguments: protected members

Return Values: pointer to class Basic symbols

- SetTandT:

Purpose: to set the values of the protected members

Arguments: protected members

Return Values: none

ENDPOINTS
*pTerminal Type
GetTandT () SetTandT ()

Class name: 2NF

Definition: A set of relations in second normal form

Relationships: Derived from class Entity.

Attributes:

- Number {text}

A number computed to set the name of the new relations.

Public Methods:

- GetNumber

Purpose: to get the value of the protected member

Arguments: none

Return Values: number

- SetNumber

Purpose: to set the value of the protected member

Arguments: number

Return Values: none

2NF
Number
GetNumber() SetNumber()

Class name: 3NF

Definition: A set of relations in third normal form

Relationships: Derived from class Entity.

Attributes:

- Number {text}

A number computed to set the name of the new relations.

Public Methods:

- GetNumber

Purpose: to get the value of the protected member

Arguments: none

Return Values: number

- SetNumber

Purpose: to set the value of the protected member

Arguments: number

Return Values: none

3NF
Number
GetNumber() SetNumber()

CHAPTER 4

FLOW OF CONTROL

4.1 BEHAVIORAL DIAGRAMS:

The overall behavior of the object-oriented design is depicted using Unified Modeling Language (UML). The UML is a modeling language, and as such predestined for translation into programming languages, that is, for code generation. It offers extensive modeling capabilities, which makes it easier to use. If the development of a major software project is carried out without the use of any conscious application of a development strategy, the problems encountered become increasingly complex during the course of program development and more and more sub- and side-problems are likely to arise. Applications of a well-proven procedure help to avoid such problems.

The behavioral diagram used to explain the flow of control in this project is the sequence diagram.

4-1-1 SEQUENCE DIAGRAMS:

A sequence shows a series of messages exchanged by a selected set of objects in a temporally limited situation, with an emphasis on the chronological course of events.

In the sequence diagram the emphasis is on the chronological course of messages. Objects are merely shown by vertical lifelines. This highlights the chronological sequence of the messages.

4-1-1-1 NOTATION:

Dashed vertical lines represent objects. On top of the line, we find the name. Messages are drawn as horizontal arrows between the object lines, on which the message itself is noted in the form message (arguments). Similarly the response is shown in textual form as a separate, but dashed arrow with an open head.

Overlapping of the dashed lifelines with broad, empty (or gray) bars symbolizes the control focus. The control focus specifies which object currently holds program control, that is, which object is currently active. The left or right hand borders can be used to note freely formulated explanations, time requirements, and the like.

Creation and removal of objects can also be represented in sequence diagrams. The creation of an object is indicated by a message that meets an object symbol; the destruction of an object by a cross at the end of the control focus.

Messages can be additionally provided with conditions in the notation [condition] message (). To indicate iterations, that is, multiple sending of a message, an asterik * is placed in front of the message.

As in drawing of ERDs the mapping and normalization can take place only when the ERD has been drawn, so the flow of control followed is sequential.

4.2 EXPLANATION:

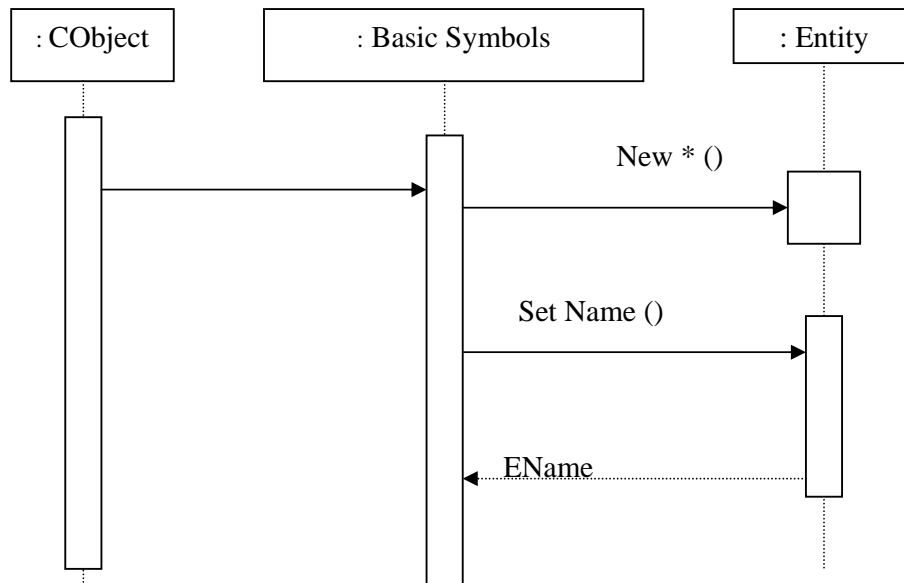
The main class or the base class of Basic Symbols is derived from the class CObject. All the remaining classes are derived from the class Basic Symbols. So the control focus lies with the class Basic symbols through out the course of program execution.

Each object in the sequence diagram is explained separately, and in the order in which the control focus is shifted.

4.2.1 ENTITY:

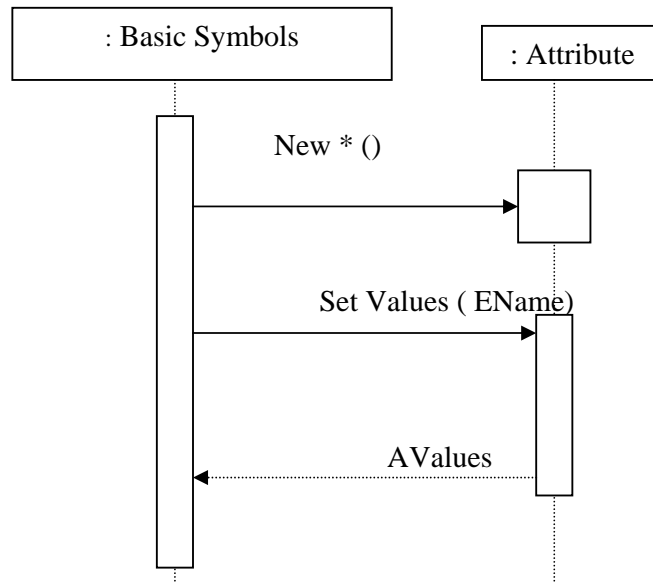
The Class Entity is derived from the Class Basic Symbols. So the message is sent from the base class (Basic Symbols) to Entity, as indicated by the direction of the arrow. The message sent is to initialize its name (Set Name ()). The response, which is in the form of the name of the entity, is given a name, that is, EName.

The class Entity is given the sequence number 1, because attributes can only be displayed with entities, same is the case with relationships, so an entity should exist before an attribute or a relationship is drawn. The Esteric depicts that it is an iterative procedure, as there is no restriction on the number of entities that can be drawn.



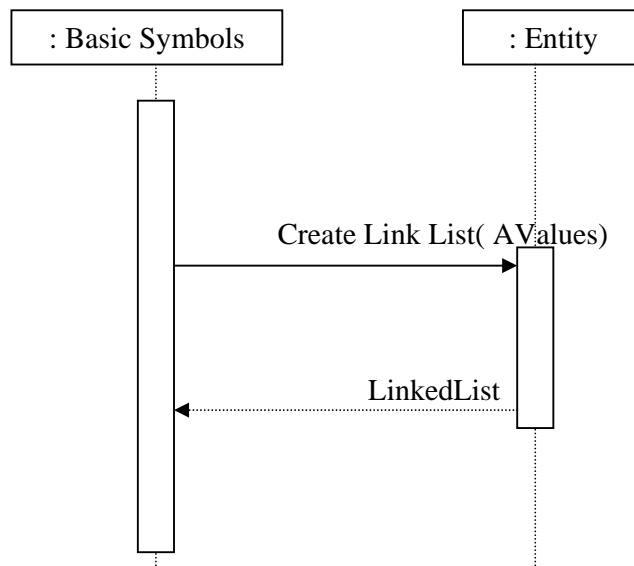
4.2.2 ATTRIBUTE:

The sequence number shows that this Object needs to be created after drawing the entity. This is also an iterative message. The message passed to this object is to set its value. The argument of this message is the name of the entity. Because an attribute sets its values according to the Entity to which it is attached. The response given by this object is Avalues.



The response `Avalues` basically consists of two parameters. One is the name of the attribute and the other is the value of the key, that is, whether the attribute has been specified as a primary key or not.

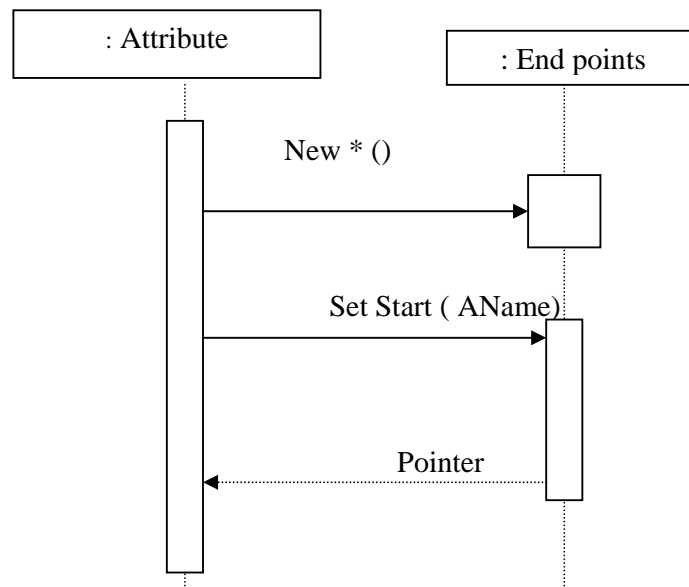
The response `Avalues` is passed as argument to a message of the class `Entity`. The message tells the object `Entity` to create a linked list of the attributes attached to that entity. Whenever an attribute gets created, a new node of the linked list is created.

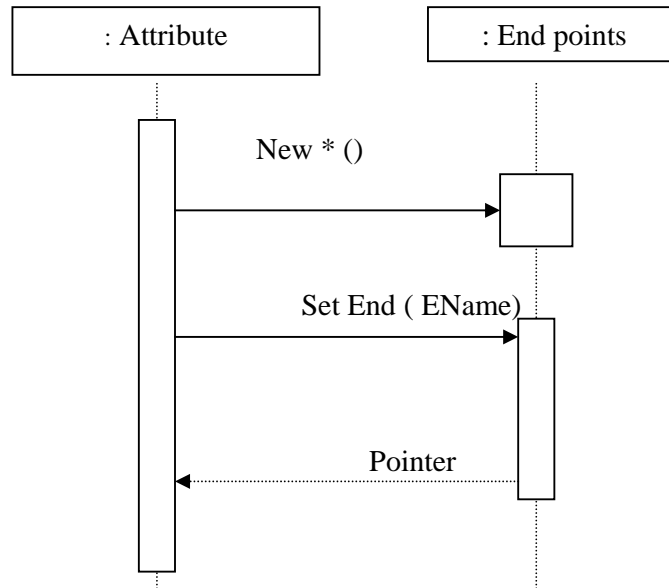


This sequence shows only that part of information that is stored and required later on for the process of mapping and normalization. In order to display the connection between the entity and attribute physically we need to know the pointers and type of symbols at the two end points of the line.

4.2.3 END POINTS:

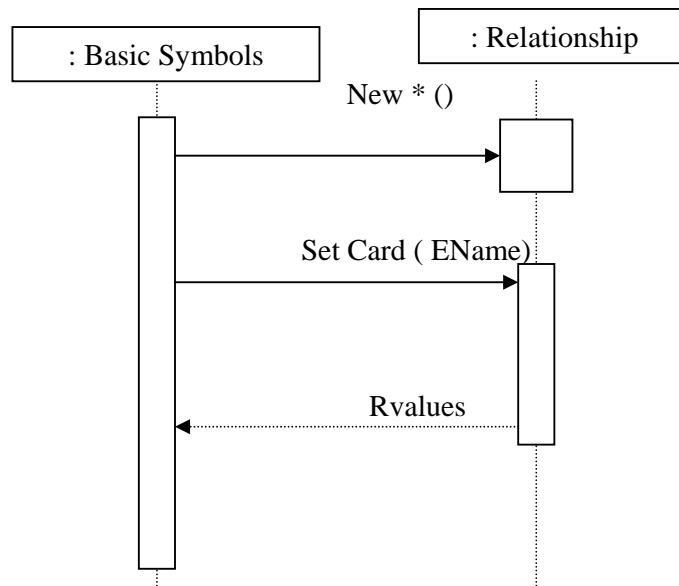
This object gets created whenever an attribute or a relationship is drawn on the screen. When an attribute is passed the message to set its values, a new object of the class end points gets created twice. Once to get the starting point of the connecting line between the entity and the attribute, and the next time to get the end points.





4.2.4 RELATIONSHIP:

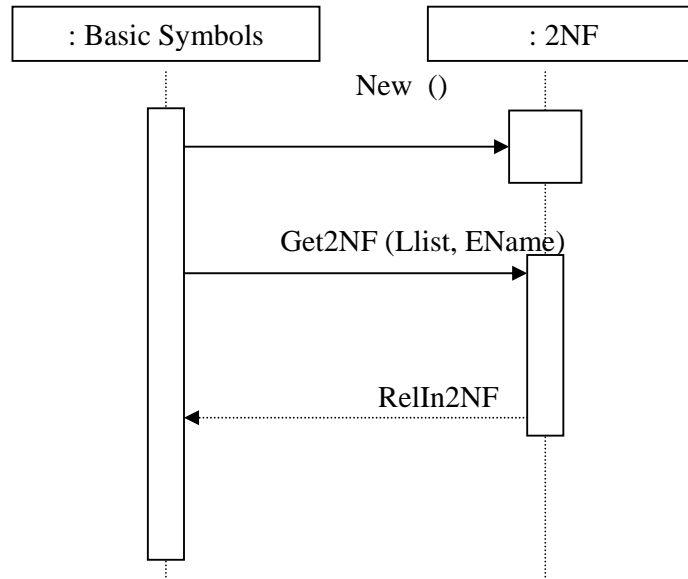
Before mapping the ERD onto relational model each entity should have some relationship with another entity of the ERD. Further more this project only deals with binary relations, that is, unary and ternary relations are not supported, thus before creating an Instance of Relation the condition that should be checked is that the number of entities should be greater than two. The message passed to the Relation class is to Set Cardinalities () and here too the argument passed is the name of the Entity. The response given is the name of the relationship, and the value of its cardinalities.



As was the case with attributes, here too the end points need to be known, so the instances of the class End points are also needed here twice. The creation of a new object of the class Relationship is also an iterative procedure. Any number of relationships can be drawn. But the number cannot exceed beyond the number of entities present. In case of binary relations this number would always be one less than the number of entities

4.2.5 2NF:

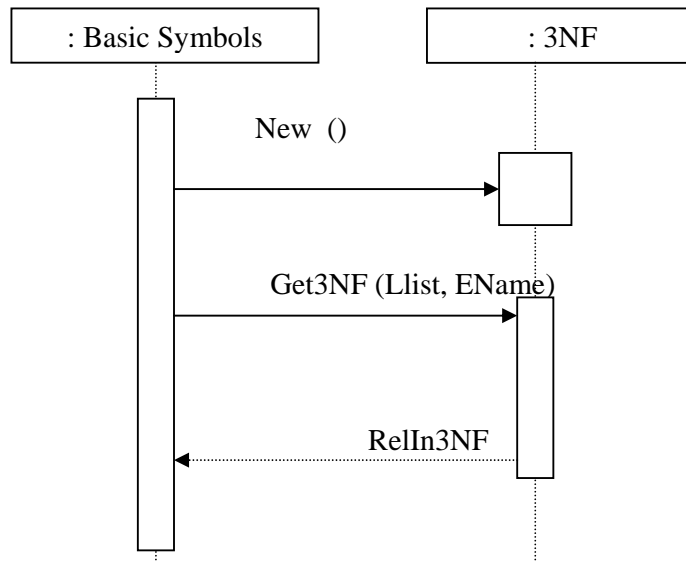
Once the relational model has been generated the relations need to be normalized. For this purpose the basic information required is available from the object Entity. Further more the response from this object is required in generating relations in the third normal form. But there are certain preconditions that must be satisfied before going to 2NF. These conditions show that a completed ERD should be present before proceeding with normalization. Hence the sequence is given accordingly.



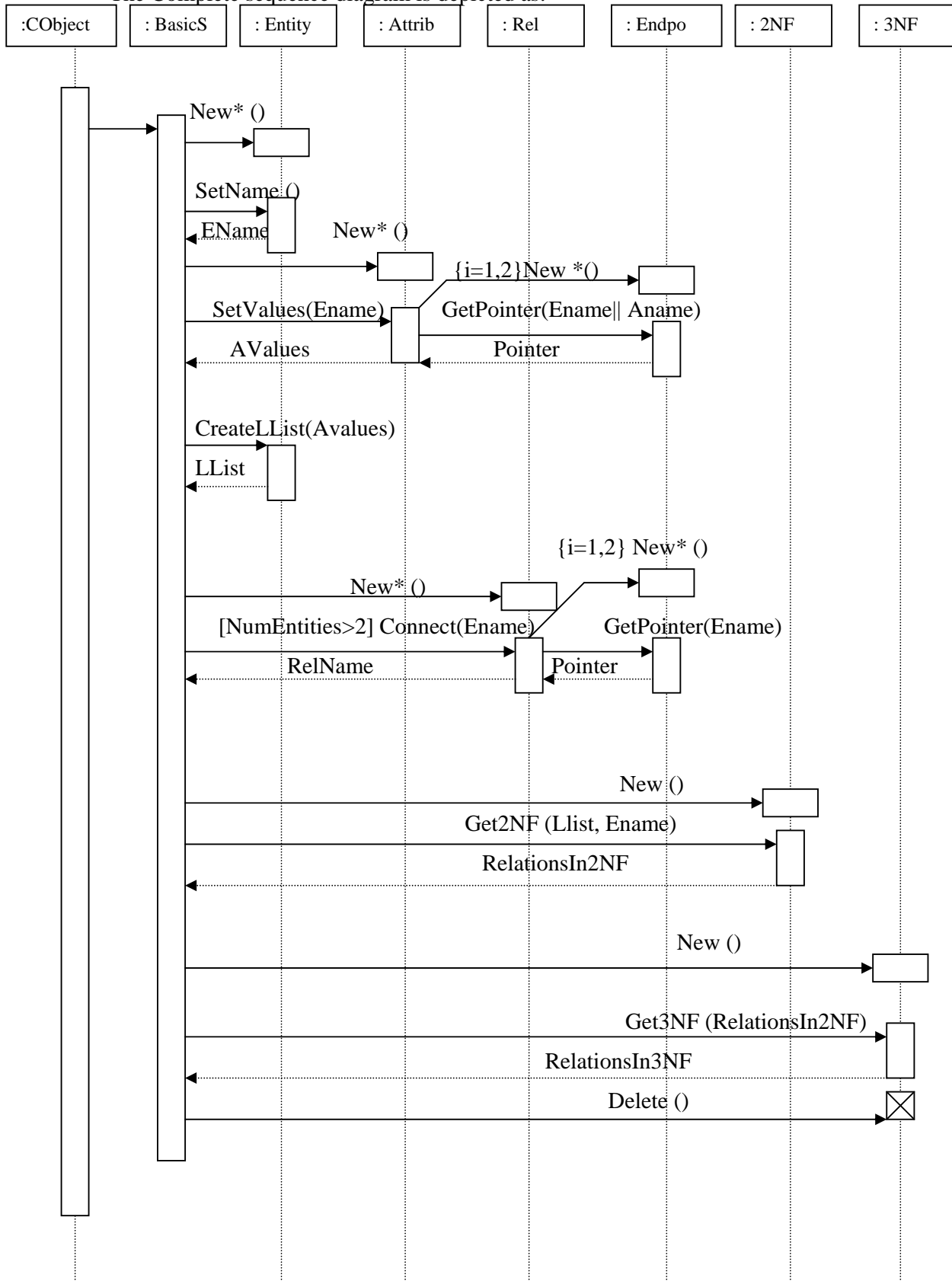
It is not an iterative procedure because once the relations have been mapped, you cannot have relations in 2NF recursively.

4.2.5 3NF:

To create a linked list of relations in third normal form the sequence shows that it should be in 2NF. The object of the class 3NF is passed a message with the relations in second normal form as its arguments. The response from this object is used in the generation of the SQL code.



The Complete sequence diagram is depicted as:



CHAPTER 5

IMPLEMENTATION

After going through all the phases of the software development process, namely Planning, Analysis and Design, the final and most important phase of Implementation is reached. The language used for implementation of the object-oriented design is Visual C++.

5.1 STEPS IN IMPLEMENTATION:

5.1.1-DRAWING ERDs:

Most of the information related to ERDs has to be entered by the user. So the first step in the implementation part is a user interface. As was mentioned during the design phase, the symbols for Entity, Attribute and Relationships are treated as Objects. Whenever a symbol is drawn a new Instance of the related class gets created.

The features that are common in all the Symbols drawn are specified in the Base class, named as Basic Symbols. Thus an important feature of the object oriented design is made use of, that is, Inheritance. The Objects are referenced by their name, which in case of entities and relationships should be unique.

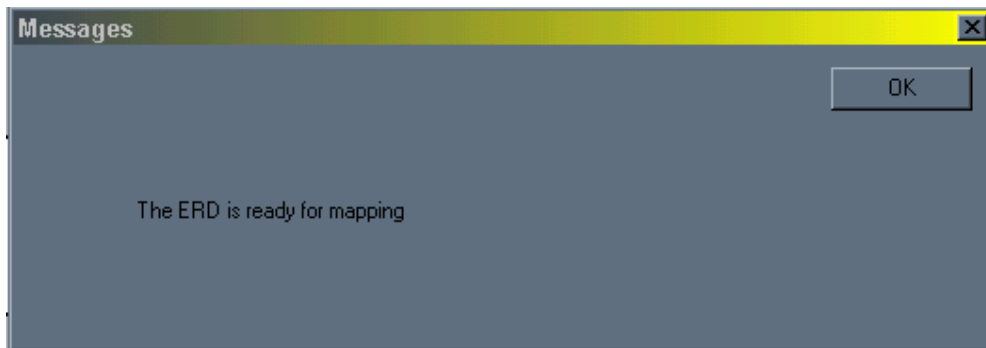
The user draws the Symbols using fixed dimensions, these dimensions are modifiable. The attribute Center is used to move the symbols to the desired location on the main window.

In the user interface the first symbol drawn is the symbol for Entity. As the attributes can only be attached to entities, so one entity atleast has to exist before an attribute can be drawn. Also relationships can be shown between entities only. Appropriate error messages are generated if the user violates these conditions.

A toolbar button named Develop is available, which helps in checking whether the ERD is ready for mapping or not. If none of the symbols is drawn the develop button is disabled (

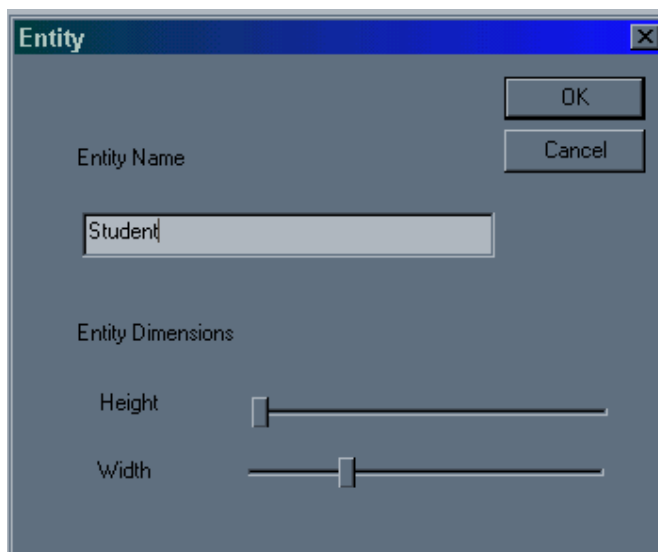
dimmed). The buttons for Mapping, 2NF and 3NF are also dimmed and are operational only when a particular condition is satisfied.

For instance, the button of mapping is operational only if the message displayed by the Develop button is that the ERD is ready for mapping.



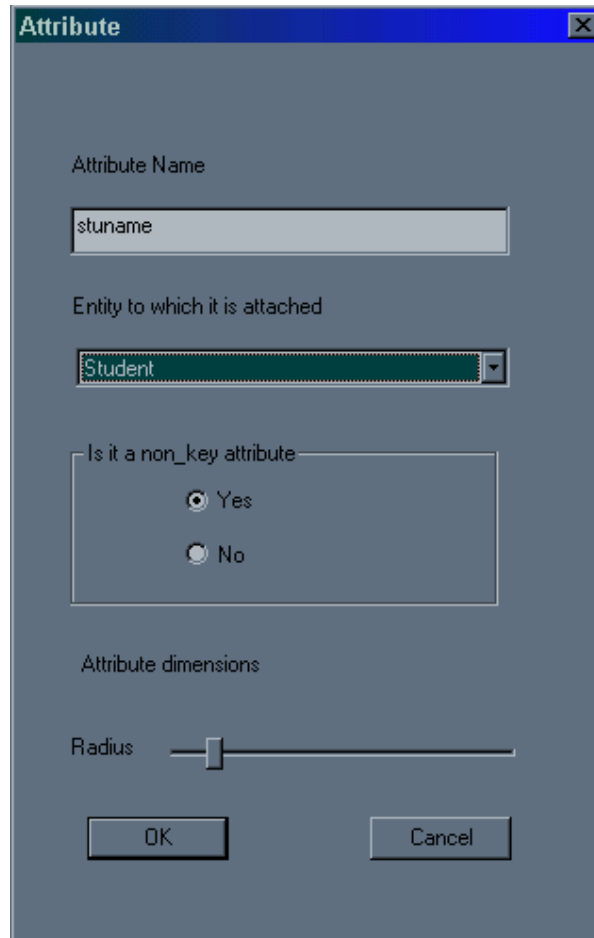
Similarly once the relations are mapped and functional dependencies specified only then can the relations be viewed in second normal form.

When a user selects the entity button from the toolbar, a dialog box of the form shown below is displayed.



The instance created is stored in an array of entities, uniquely identified by its name.

The next symbol drawn is an attribute.



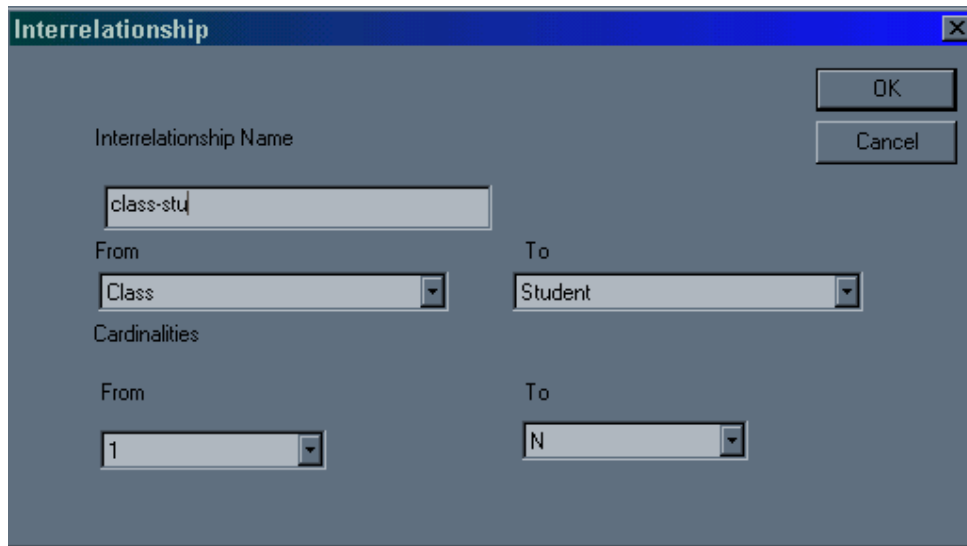
The screenshot shows a dialog box titled "Attribute". It contains the following elements:

- Attribute Name:** A text input field containing the text "stuname".
- Entity to which it is attached:** A dropdown menu with "Student" selected.
- Is it a non_key attribute:** A group box containing two radio buttons: "Yes" (which is selected) and "No".
- Attribute dimensions:** A slider control labeled "Radius".
- Buttons:** "OK" and "Cancel" buttons at the bottom.

The user is prompted to enter the information relating to name of the attribute and the key value, that is, whether it is a primary key or not. The name of the entity to which this attribute is to be attached also needs to be specified. The names of only those entities are displayed which exist. The user is not given the option to specify the name of an entity that has not been drawn yet.

When an instance of this class gets created a method of the class Entity is called. This method creates a new node of the linked list which stores these attribute values with the specified entity.

The relationship exists between two entities. The user is prompted to specify the name of the relationship, plus the cardinalities also need to be specified.

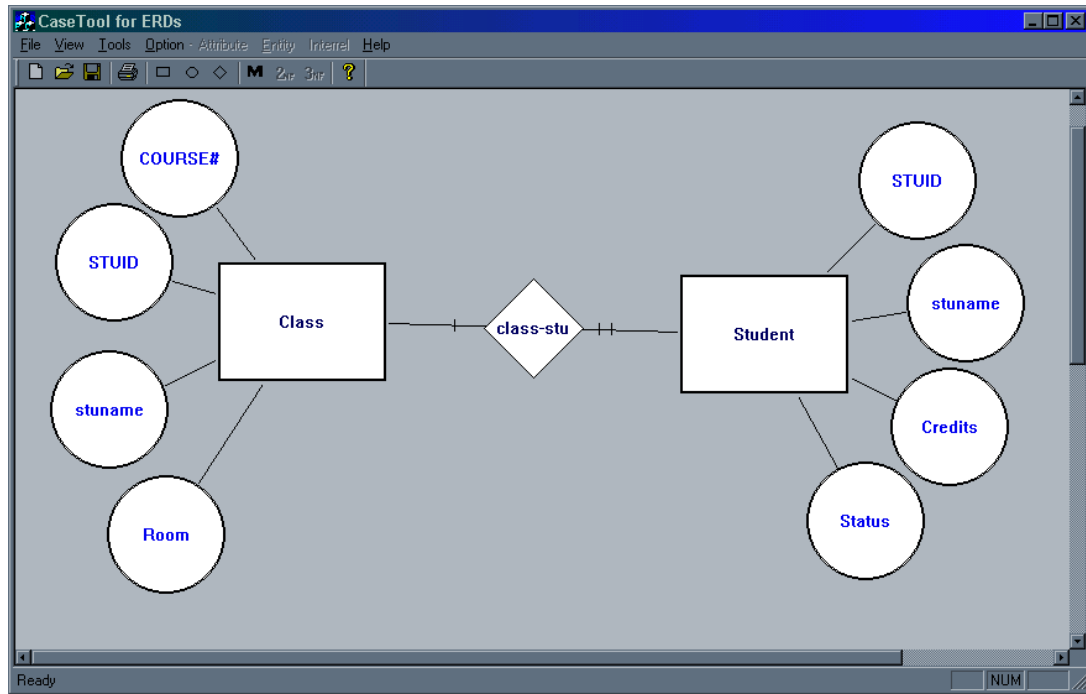


The screenshot shows a dialog box titled "Interrelationship". It has a close button (X) in the top right corner. Below the title bar, there are two buttons: "OK" and "Cancel". The main area of the dialog contains the following fields:

- Interrelationship Name:** A text box containing "class-stu".
- From:** A dropdown menu with "Class" selected.
- To:** A dropdown menu with "Student" selected.
- Cardinalities:** Two dropdown menus. The "From" dropdown shows "1" and the "To" dropdown shows "N".

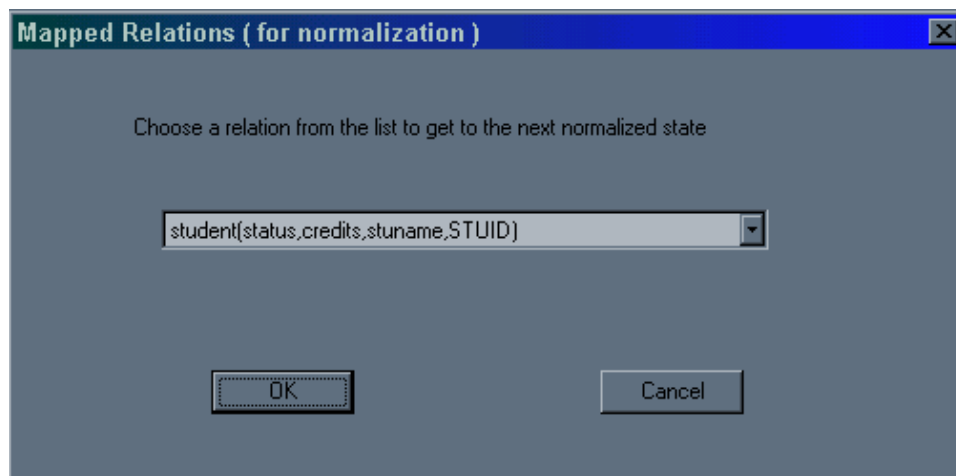
There are certain features which are not incorporated in this project. But the basic idea behind developing a comprehensive ERD development tool is kept into view.

The ERD thus drawn is as:



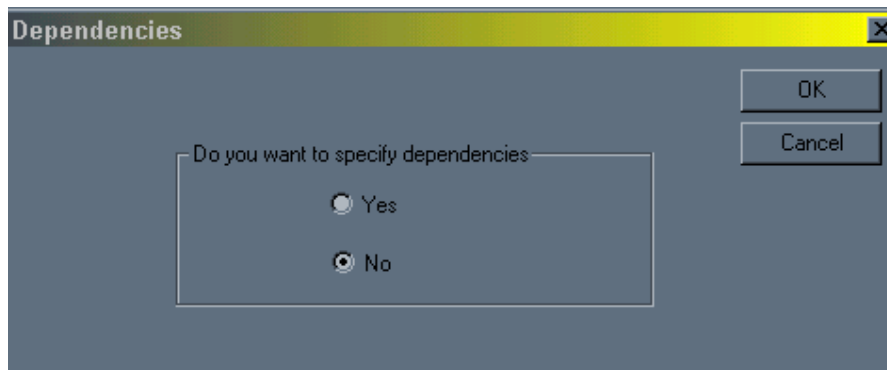
5.1.2- MAPPING:

This ERD can now be mapped onto relational model. For this purpose the name of the Relation is taken to be same as that of the Entity, also the attribute values are also the same. As the user is not given the option to attach an attribute with a relationship so the mapping is quite simple. The mapped relation can be viewed by selecting the specified button in the toolbar.

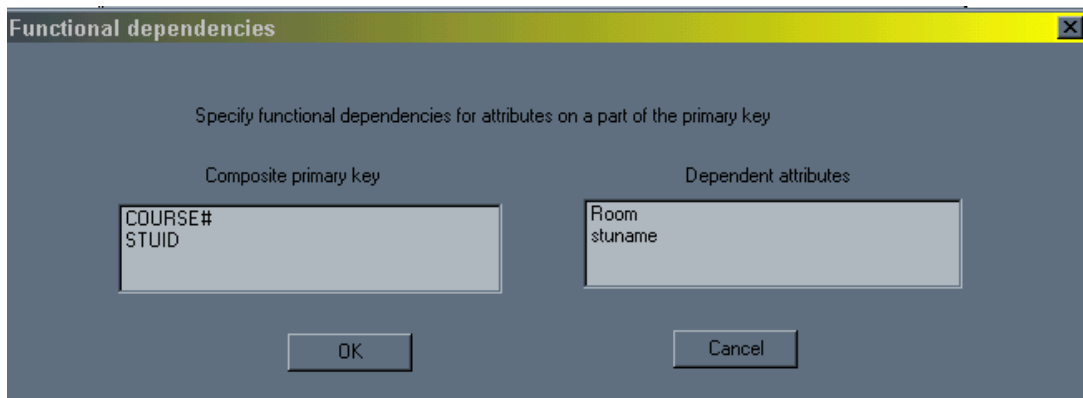


5.1.3 NORMALIZATION:

The mapped relations are considered to be in first normal form by default. Because 1NF can be checked, only when data is entered into the database. So, we directly move onto second normal form. As 2NF deals with functional dependencies, hence functional dependencies can only be specified with relations that have a composite primary key. The user is asked to specify the functional dependencies at this stage.



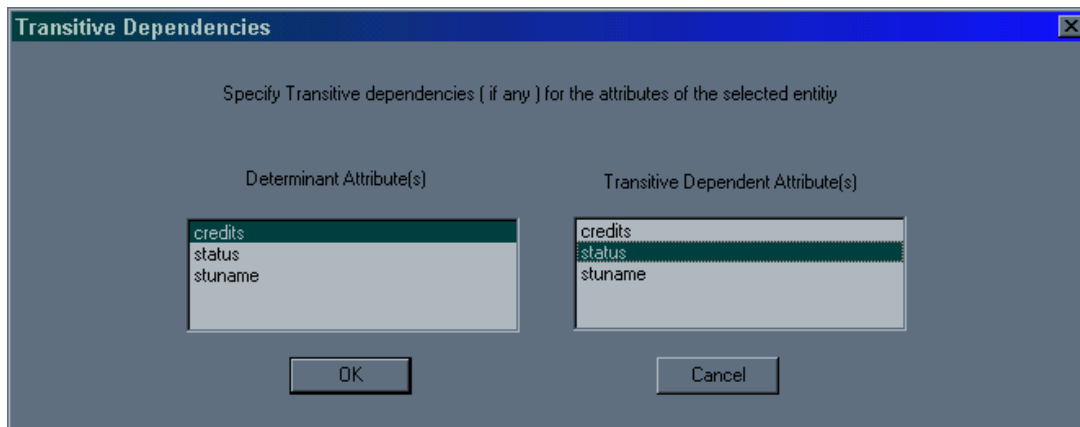
If he selects the option Yes then the dialog box is displayed with the composite primary key on one side and the non-key attributes on the other.



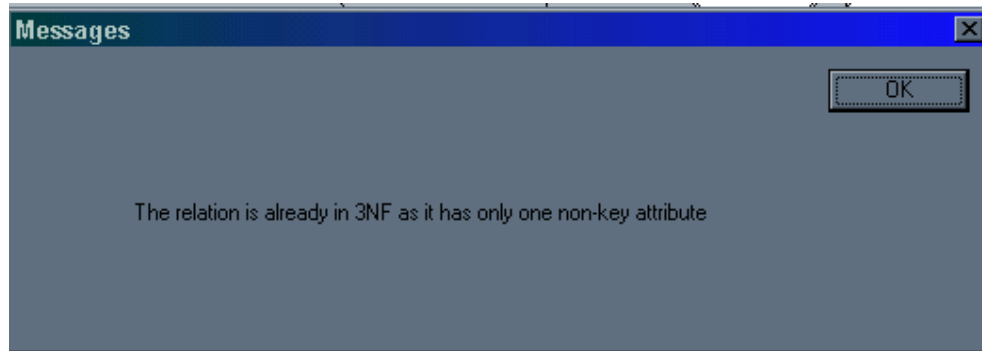
If the user does select to specify the functional dependencies, we get the normalized relations in 2NF. This is done by creating a new object of class 2NF. This class is derived from the class Entity. Because it also contains attributes for name of the relation, and a linked list for the associated attributes. The additional feature in this class is the attribute Number. This attribute is used to assign numbers to name of the normalized relations. For instance, if the relation has the name class and the user specifies functional dependencies on its attributes two new relations are created, one is given the name class1 and the other is given the name class2.

The relations in 2NF can be viewed by selecting the option from the toolbar.

Similarly the relations in third normal form can be obtained by specifying transitive dependencies.



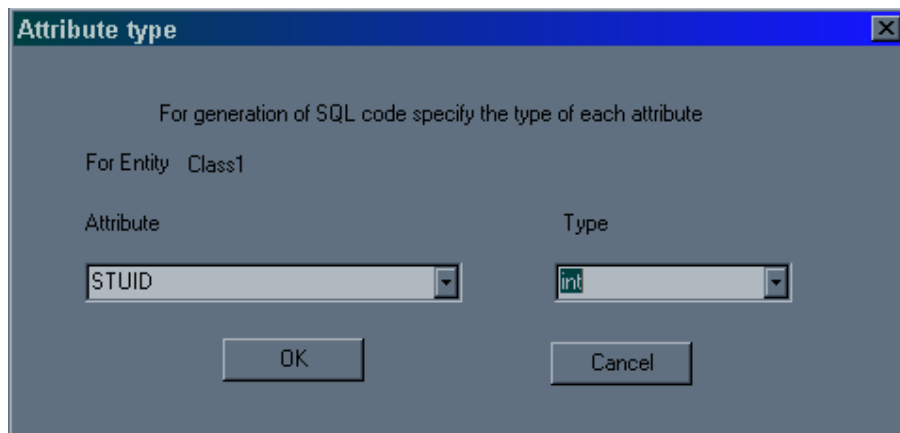
For relations with a single primary key, the transitive dependencies cannot be specified. So the relation is already in 3NF.



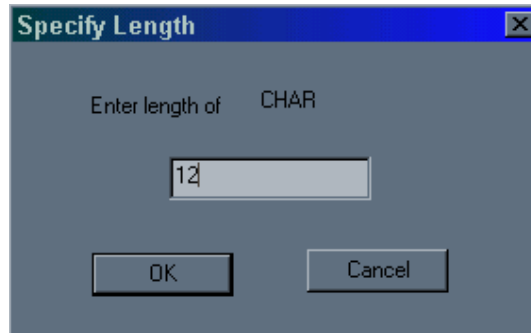
Which gives relations in third normal form.

If the relation is already in 3NF then the new relation has the same name as the name of the relation in 2NF, also it has the same attributes.

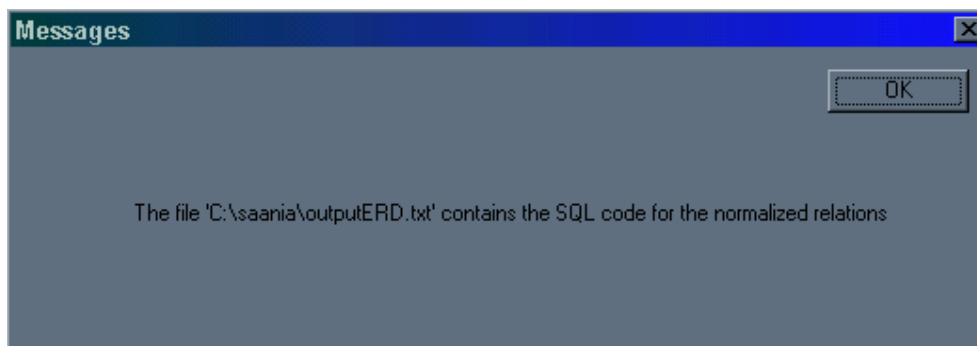
Once the normalization has been completed the SQL code is generated and stored in a file. For this purpose you need to specify the type of each attribute.



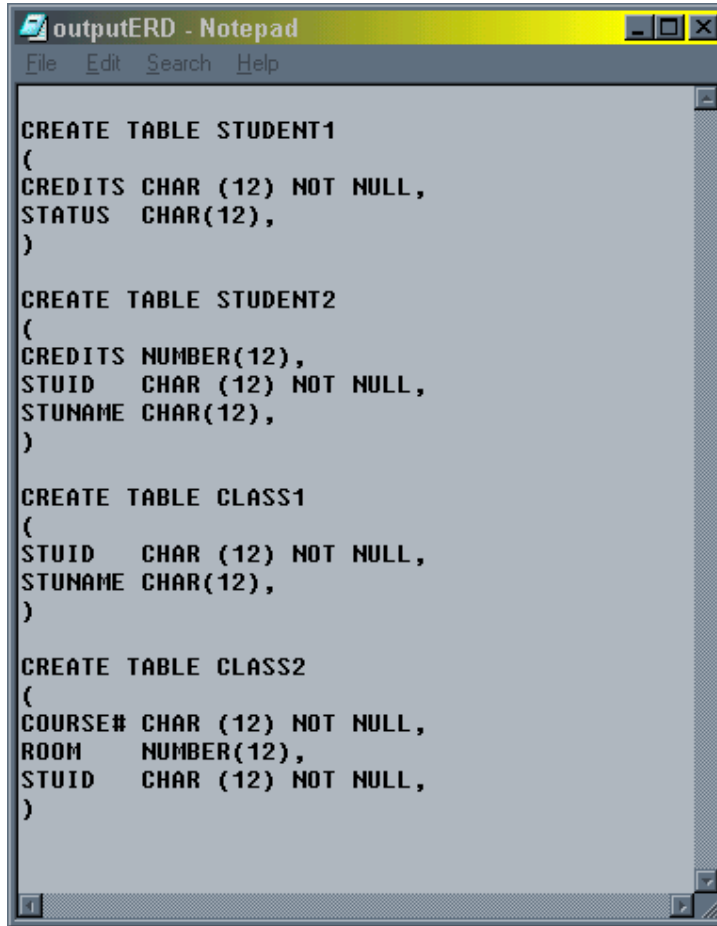
When the Type of the attribute is selected as being CHAR or NUMBER than the user is prompted to specify the number of characters or numbers. Thus floating point attribute types can also be specified.



The SQL code created is stored in a file, the path of which is displayed to the user.



These relations can now be simply copied from the file and pasted onto the SQL compiler.



```
outputERD - Notepad
File Edit Search Help

CREATE TABLE STUDENT1
(
CREDITS CHAR (12) NOT NULL,
STATUS CHAR(12),
)

CREATE TABLE STUDENT2
(
CREDITS NUMBER(12),
STUID CHAR (12) NOT NULL,
STUNAME CHAR(12),
)

CREATE TABLE CLASS1
(
STUID CHAR (12) NOT NULL,
STUNAME CHAR(12),
)

CREATE TABLE CLASS2
(
COURSE# CHAR (12) NOT NULL,
ROOM NUMBER(12),
STUID CHAR (12) NOT NULL,
)
```

CHAPTER 6

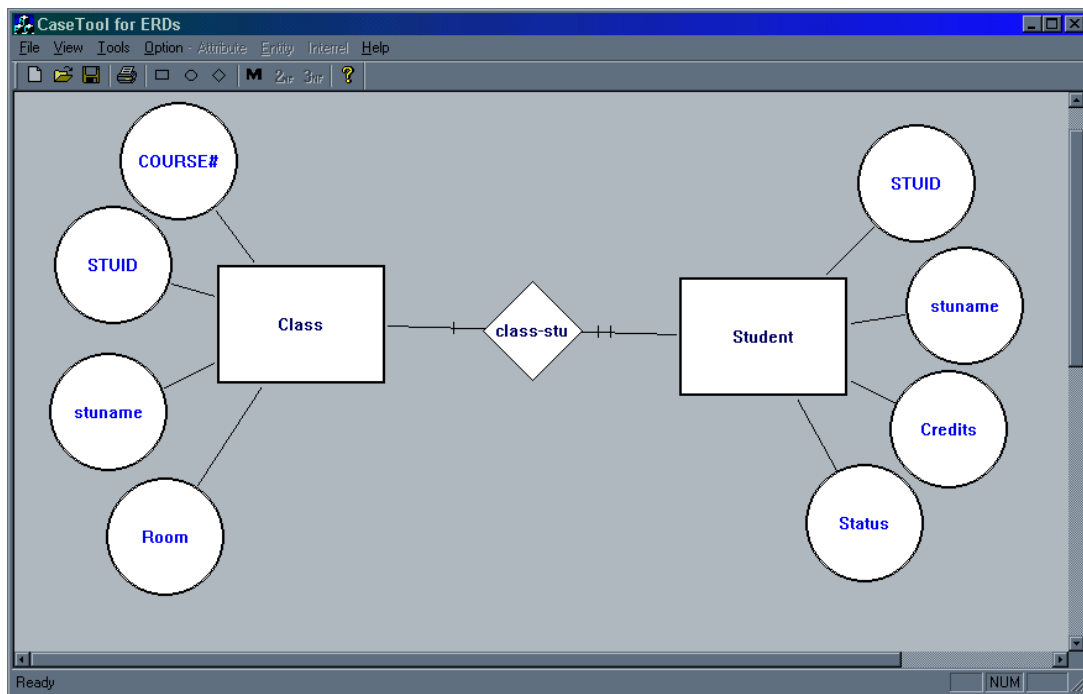
RESULTS AND RECOMMENDATIONS

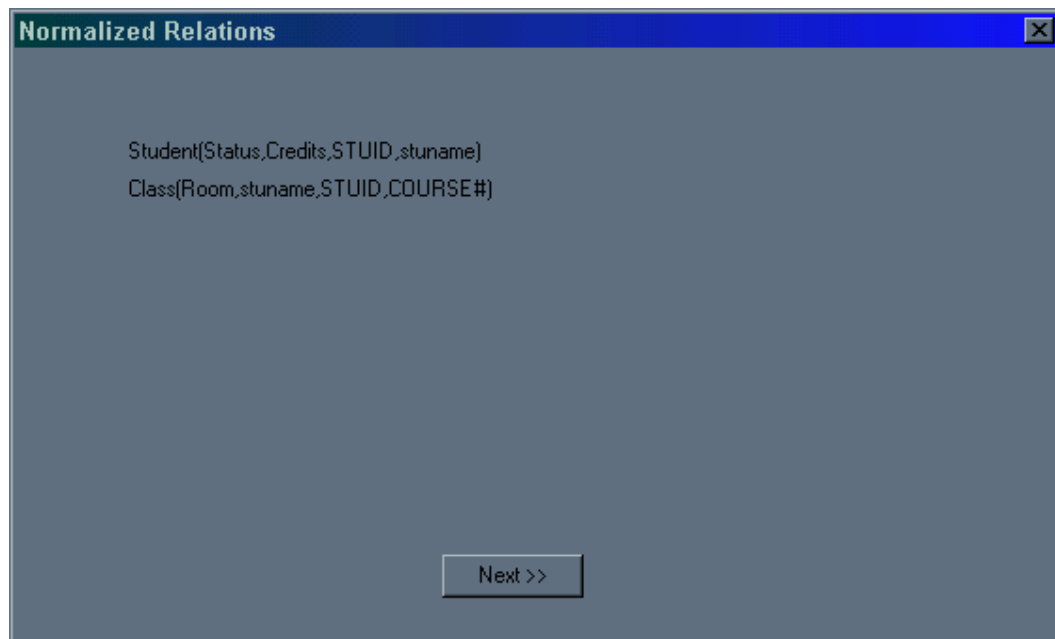
6.1 PROBLEMS ENCOUNTERED:

Most of the problems encountered during this project were basically due to lack of knowledge of the language used. The basic concepts related to drawing of ERDs, their mapping onto Relational model and their normalization were quite clear right from the start., as most of it was covered as a part of course on 'Database Systems'.

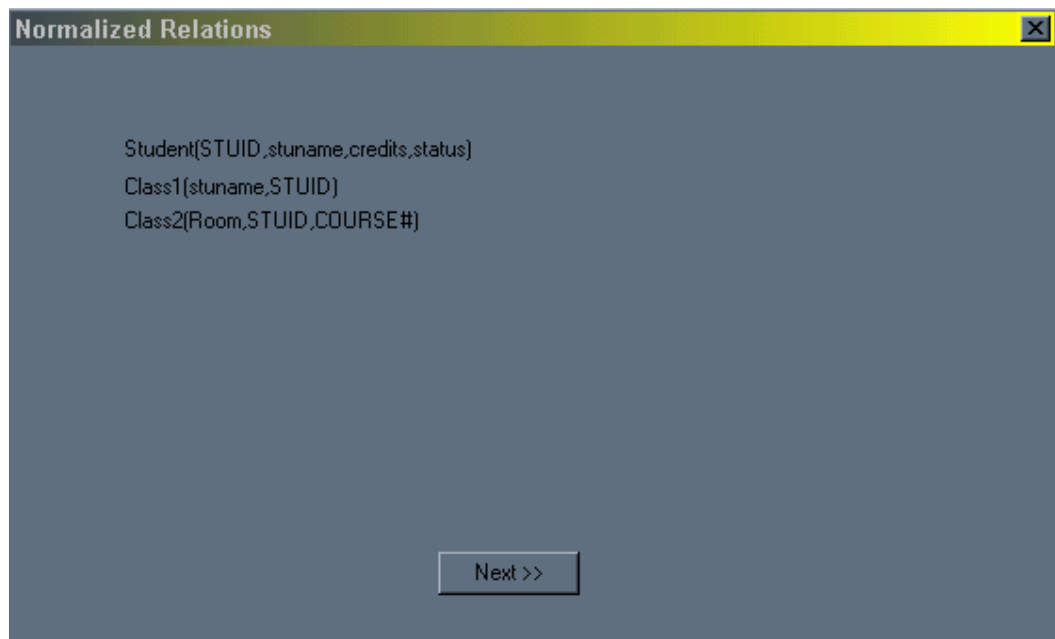
6.2 RESULTS:

After drawing the ERD, the relations are mapped onto the relational model. The ERD and the corresponding relational model are viewed as:



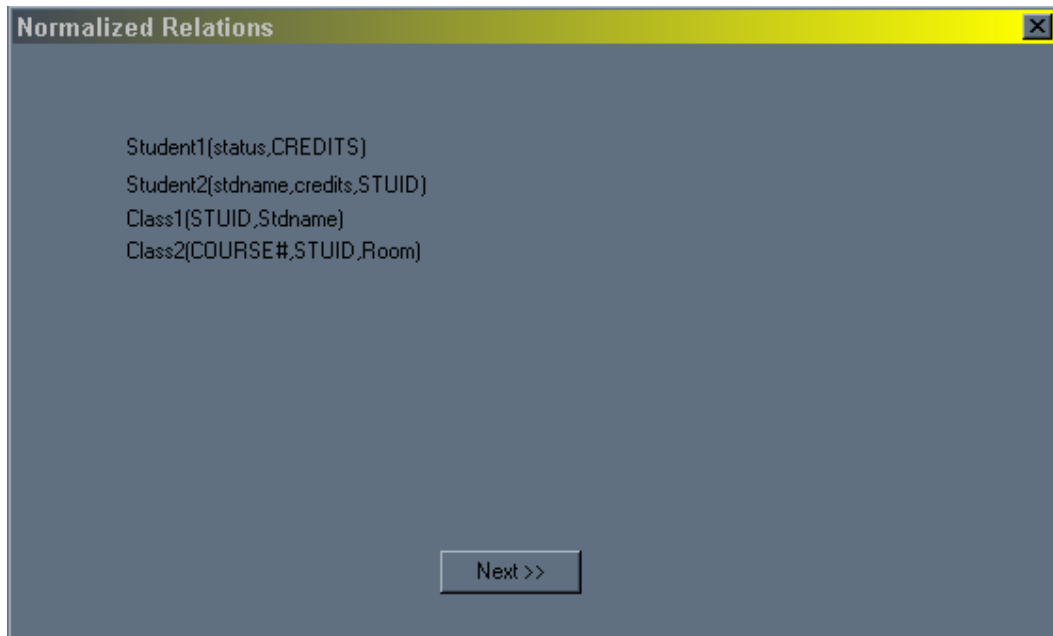


These relations are normalized, following the rules for 2NF and 3NF. The relations in 2NF are viewed as:



Here the new relation formed consists of that part of the primary key on which another non-key attribute depends, plus the dependent attribute. The original relation is left with the whole of the primary key and all the remaining attributes, other than the one which showed dependency on a part of the primary key. In the given example, this is shown for the relation 'class'.

Similarly after 2NF, the relations in 3NF are as:



The relation 'student' is normalized to 3NF, where the dependent attribute and the non-key attribute on which it depends (determinant) form the new relation. The determinant is the Primary key of the new relation. In the original relation the dependent attribute is removed. The determinant appears as foreign key in the original relation.

The relations in 3NF can now be used to generate the SQL code.

6.3 CONCLUSIONS:

The results show that given any ERD the entities, attributes and relationships would be mapped onto the relational model according to the rules of mapping. These relations would be normalized upto 3NF and after the user has specified the type of the attributes, the SQL code gets stored in a file.

This ensures that the decomposition is loss-less and that the data stored in the database is accurate. Thus the aims outlined in the beginning of the project were reached successfully.

6.4 RECOMMENDATIONS:

ERD design is a very vast field. This project does not cover some of the features, which were not required to understand the basic concept, but for an efficient and comprehensive CASE Tool these features should be included. Some of the very prominent of these features are:

- Symbols to distinguish weak Entity from Strong Entity.
- Representation of Unary and ternary Relationships.
- Ability to attach an attribute to a Relationship.
- Representing Aggregation and Generalization etc.
- Keeping the above-mentioned features into view, mapping the relations accordingly.
- Normalization of relations beyond 3NF i.e. Incorporating the conditions for BCNF, 4NF and 5NF.

6.4.1 WEAK ENTITY/STRONG ENTITY:

If X and Y are entities and each instance of Y must have a corresponding instance of X, it means that Y entity cannot exist without some X entity. A Y cannot enter the database unless its corresponding X is there, and if the X is dropped from the database, the Y must be dropped as well. X is referred to as Strong, Parent or Dominant Entity and Y as the Weak, Child or Dependent entity.

When the cardinalities are specified in the relationships, we are taking into account the dependency of one entity on the other, but there are no distinguishing features available for identifying these strong and weak entities as far as the view goes. This can be accomplished by drawing another rectangle outside or inside the rectangle already drawn for the Entity, to represent weak entity.

6.4.2 UNARY/TERNARY RELATIONSHIPS:

As a relationship can involve more than two entity sets, or it may exist for a single entity only, the features of unary and ternary relations can be incorporated to get a very comprehensive CASE Tool.

6.4.3 ATTRIBUTES WITH RELATIONSHIPS:

Attributes can be attached with relationships as well. This feature is not included in this project. This should be taken into account as well in the future designs.

6.4.4 AGGREGATION/GENERALIZATION:

Two types of abstractions can further be added in ERDs, called aggregation and generalization. These abstractions make the model more powerful by allowing the designer to express concepts that are not easily included in the basic model.

The purpose of any type of abstraction is to allow different perceptions of the objects under discussion. The aggregation abstract allows the designer to either decompose objects, breaking them into more detailed components, or to aggregate objects, grouping them together into higher level objects. Decomposition, is simply the opposite process to aggregation. Both processes are considered part of the aggregation abstract.

Another powerful abstraction is generalization. Generalization allows objects of different types to be considered as examples of a higher-level set or, conversely, objects in a

set to be categorized into specialized types. Breaking up a set of objects into the various types it contains, or categorizing the objects in the set according to their roles in a relationship, is called Specialization. Its inverse is generalization, which means combining different types of objects into a higher-level set.

6.4.5 MAPPING:

Including the above mentioned features in the ERD would result in some changes in mapping to be incorporated as well.

In converting an ERD into a relational model, strong entities become tables having a column for each of the entity's attributes. Tables of weak entities have columns for the key attributes of the corresponding strong entity. Relationship tables have columns for the primary key attributes of the related entities and a column for each descriptive attribute of the relationship. Many-to-many relationships require a separate table, but one-to-one and one-to-many relationships can often be represented by foreign keys, instead of by separate tables.

When generalization has been used, there may be no table for the higher level entity, but each lower level entity table can contain columns for all the attributes of the higher level entity in addition to the columns for their own attributes. Alternatively, there may be a table for the higher-level entity that contains its attributes and the tables for the lower-level entities contain columns for the key of the higher-level entity plus columns for their own attributes. Aggregation is not represented explicitly by tables.

6.4.6 NORMALIZATION BEYOND 3NF:

Although normalization till 3NF is sufficient for relations with a single candidate key, but it is deficient in cases where there are multiple candidate keys and where candidate keys are composite and overlapping. Therefore, an improved definition of third normal form, named for its developer, Boyce and Codd, was formulated to take care of all the cases.

A relation is in Boyce-Codd Normal Form (BCNF) if and only if every determinant is a candidate key.

So for a relation with one candidate key, 3NF and BCNF are equivalent.

Similarly a relation is in 4NF, if and only if it is in BCNF and there are no multivalued dependencies.

A relation is in 5NF if no remaining nonloss projections are possible, except the trivial one in which the key appears in each projection.

SUMMARY OF THE PROJECT

In this project CASE Tools are used to develop ERDs and generate SQL code. A Database is designed in various stages. The staged approach to database design is a top-down approach that allows the designer to develop a logical model that mirrors the operations of the organization. Standard database architecture uses three levels of abstraction: external, logical and internal.

This project focuses on the logical model. The logical model is the heart of the database. A good logical design is easy to implement and supports the desired external views. Logical design is a challenging and rewarding task. The design of the project starts with the design of the Entity Relationship Diagram (ERD), which are used to show entity sets, attributes of entities, relationship sets, and describe attributes relationships. A relation is any subset of the Cartesian product of the domains of the attributes. Properties of database relations are: each cell is single-valued, column names are distinct, column values come from the same domain, column order is immaterial, row order is immaterial, and there are no duplicate rows.

Here the symbols for entities, attributes and relationships are treated as objects. Whenever a symbol is drawn, an instance of the respective class is created. This ERD is then mapped onto relational model. The mapping is quite simple, because here attributes are attached with entities only and not with relationships. As is the case with relations, it must have at least one primary key. Entity integrity is a constraint that states that no attribute of a primary key may be null. These features are also incorporated in the design.

Then the relations are normalized. The relations are considered to be in First Normal Form by default (as it can be checked only when the data is placed in the database). The normalization goes only till the Third Normal Form. After normalization has been completed the SQL code is generated, once the user has specified the type of each Attribute.

REFERENCES

- Ricardo, Catherine. *Database Systems: Principles, Design and Implementation*. New York: Macmillan Publishing Company, 1990.
- Tsichritzis, D. and F. Lochovsky. *Data Models*. Englewood Cliffs, NJ: Prentice-Hall, 1982.
- Oestereich, Bernd. *Developing Software with UML: Object-Oriented analysis and Design in Practice*. Addison- Wesley, 1997.
- Shaw, Robert and Osier, Dan. *Teach Yourself MFC in 21 days*. SAMS Publishing, 1995.
- Young, Michael J. *Mastering Visual C++ 6*, BPB Publications, 1998.
- Maruzzi, Stefano. *The Microsoft Windows 95: Developers guide*. Ziff-Davis Press, 1996.
- Pressman, Robert S. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Publications, 1997.