②

NAVAL POSTGRADUATE SCHOOL
Monterey, California

UNITED STATES NAVY • NAVAL POSTGRADUATE SCHOOL •

THESIS

A CONCEPTUAL DATABASE DESIGN AND PERFORMANCE ANALYSIS OF THE MILSATCOM REQUIREMENTS DATABASE

by

Ronald G. Kearns

June, 1993

Thesis Co-Advisor:	Daniel R. Dolk
Thesis Co-Advisor:	Magdi N. Kamel

Approved for public release; distribution is unlimited.

Unclassified
Security Classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification: Unclassified		1b Restrictive Markings			
2a Security Classification Authority		3 Distribution/Availability of Report Approved for public release; distribution is unlimited.			
2b Declassification/Downgrading Schedule					
4 Performing Organization Report Number(s)		5 Monitoring Organization Report Number(s)			
6a Name of Performing Organization Naval Postgraduate School	6b Office Symbol *(if applicable)* 39	7a Name of Monitoring Organization Naval Postgraduate School			
6c Address *(city, state, and ZIP code)* Monterey CA 93943-5000		7b Address *(city, state, and ZIP code)* Monterey CA 93943-5000			
8a Name of Funding/Sponsoring Organization U.S. Space Command	6b Office Symbol *(if applicable)* J6	9 Procurement Instrument Identification Number N62271RLNTB			
Address *(city, state, and ZIP code)* Peterson AFB, CO 80914		10 Source of Funding Numbers MJPR NS-93-76			
		Program Element No	Project No	Task No	Work Unit Accession No
11 Title *(include security classification)* A CONCEPTUAL DATABASE DESIGN AND PERFORMANCE ANALYSIS OF THE MILSATCOM REQUIREMENTS DATABASE					
12 Personal Author(s) *Kearns, Ronald Gene*					
13a Type of Report Master's Thesis	13b Time Covered From To	14 Date of Report *(year, month, day)* 1993, June	15 Page Count 87		
16 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.					

17 Cosati Codes			18 Subject Terms *(continue on reverse if necessary and identify by block number)*
Field	Group	Subgroup	MILSATCOM, MRDB, database design, normalization, database performance

19 Abstract *(continue on reverse if necessary and identify by block number)*

The Military Satellite Communications Decision Support System (MDSS) project for U.S. Space Command (USSPACECOM) is intended to provide decision makers an integrated inform ion tool to effectively manage military satellite communication (MILSATCOM) resources while satisfying communications requirements. An important aspect of MDSS is the information provided by the MILSATCOM Requirements Database (MRDB). The objective of this thesis is twofold. First, it develops a new conceptual schema for the MRDB using the Entity Relationship Model and transforms it into a relational schema for implementation. Second, it conducts a comparative performance analysis to examine the tradeoffs between normalization and performance.

This thesis concludes that a structured approach to designing the MRDB results in a normalized structure that is simple, easy to implement, and provides acceptable performance. A fully normalized database structure does not seem to have a significant impact on overall MRDB retrieval performance.

20 Distribution/Availability of Abstract _ unclassified/unlimited _ same as report _ DTIC users	21 Abstract Security Classification Unclassified	
22a Name of Responsible Individual Daniel R. Dolk, Magdi N. Kamel	22b Telephone *(include Area Code)* (408)656-2260, (408)656-2494	22c Office Symbol AS/Dk, AS/Ka

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted — security classification of this page

All other editions are obsolete — Unclassified

A Conceptual Database Design and Performance Analysis of the MILSATCOM Requirements Database

by

Ronald G. Kearns
Captain, United States Air Force
B.G.S., University of Nebraska at Omaha

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS TECHNOLOGY

from the

NAVAL POSTGRADUATE SCHOOL
June 1993

Author: Ronald G. Kearns
Ronald G. Kearns

Approved by: Daniel R. Dolk
Daniel R. Dolk, Thesis Co-Advisor

[illegible]
Magdi N. Kamel, Thesis Co-Advisor

Dan C Boger
for Paul H. Moose, Chairman
Command, Control and Communications Academic Group

ABSTRACT

The Military Satellite Communications Decision Support System (MDSS) project for U.S. Space Command (USSPACECOM) is intended to provide decision makers an integrated information tool to effectively manage military satellite communication (MILSATCOM) resources while satisfying communications requirements. An important aspect of MDSS is the information provided by the MILSATCOM Requirements Database (MRDB). The objective of this thesis is twofold. First, it develops a new conceptual schema for the MRDB using the Entity Relationship Model and transforms it into a relational schema for implementation. Second, it conducts a comparative performance analysis to examine the tradeoffs between normalization and performance.

This thesis concludes that a structured approach to designing the MRDB results in a normalized structure that is simple, easy to implement, and provides acceptable performance. A fully normalized database structure does not seem to have a significant impact on overall MRDB retrieval performance.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. BACKGROUND

The Military Satellite Communications Decision Support System (MDSS) project for U.S. Space Command (USSPACECOM) at Peterson Air Force Base, Colorado is intended to provide decision makers an integrated information tool to effectively manage military satellite communications (MILSATCOM) resources while satisfying communications requirements. One important aspect of the MDSS is the integration of multiple databases, including the MILSATCOM Requirements Database (MRDB). The MRDB is a relational database used by USSPACECOM to track MILSATCOM requirements. It was initially developed as a "proof-of-concept" and has evolved into an operational system without undergoing a formal design process. The current MRDB structure lends itself to the problem of update anomalies.

B. PURPOSE

The objectives of this study are to:

- Develop a new conceptual database design for the MRDB using a structured database design process
- Examine the tradeoffs between database normalization and performance.

Normal forms in relational database design refer to guidelines or rules for record design. These rules are devised to prevent update anomalies and data inconsistencies. Database normalization, however, tends to penalize retrieval time since data may have to be retrieved from multiple tables. Conversely, performance requirements may preclude a database from having all records fully normalized, which may result in increased problems maintaining data integrity.

Currently, the MRDB is less than fully normalized and contains data redundancies which allow queries to be executed more efficiently. This situation leads to updated anomalies. It is desired that the MRDB be more normalized in order to reduce these anomalies.

The main benefit of this thesis will be an improved database structure for the MRDB. This results in the elimination of update anomalies insuring data integrity and ensures that future modifications of the MRDB are smooth and easy.

C. METHODOLOGY

The following steps will be undertaken to accomplish the thesis objectives:

- An overview of user's data requirements will be gathered and the current MRDB structure is examined.
- A high-level conceptual model of the MRDB will be developed using the Entity Relationship Model (ERM).
- A set of relations or tables will be created from the conceptual model. These tables will be implemented using a database management system.
- The performance of the new database structure will be compared to the performance of the original MRDB to examine the tradeoffs between a highly normalized database structure and one that is less than normalized.

D. SCOPE OF THESIS

The Scope of this thesis is the design of the MRDB and the normalization of its structure. A thesis by Bill Major [Ref. 1] implements a prototype version of the new normalized MRDB structure using Microsoft Access, a PC-based DBMS with user-friendly graphical interfaces.

Another thesis by Hugh Henry [Ref. 2] defines the functional requirements of the MDSS, proposes a functional architecture, and designs a user interface for the MDSS.

E. THESIS ORGANIZATION

Information for this thesis was collected from the MRDB source code, user's guide, interviews with the contractors who developed the MRDB, and users of the MRDB.

Chapter II of the thesis reviews a structured approach to thc database design process that uses the Entity Relationship (ER) Diagram to model user's data requirements. Chapter II also describes an algorithm that translates an ER diagram into a relational model that is used to implement the physical database. Chapter III executes the approach presented in Chapter II to develop a normalized relational schema for the MRDB. Chapter IV is a comparative performance analysis of the original MRDB and the new normalized MRDB. Finally, Chapter V concludes the thesis and makes recommendations concerning improvements and future study.

II. RELATIONAL DATABASES AND NORMALIZATION

This chapter provides background information on the database design process, the Entity Relationship Model, the relational model, and the process of normalization. First, an overview of a well known database design process is presented. Second, a technique known as the Entity-Relationship (ER) model that represents the logical structure of a database is described. Third, some general guidelines are offered for classifying entities and attributes within the ER diagram. Fourth, steps for transforming an ER diagram (logical model) into a relational model are given. Finally, the process of normalization, with all its associated normal forms, is defined.

A. DATABASE DESIGN OVERVIEW

This section will summarize the database design process focusing on the conceptual design phase to represent the user's data requirements. In a nutshell, database design is accomplished in two separate activities: logical and physical. The logical portion is involved with the structure of the data independent of any particular Database Management System (DBMS) whereas the physical is DBMS-specific. This chapter deals mainly with the logical activity of database design.

1. The Database Design Process

Designing a database is a process that involves, most of all, creativity. To aid the creative process, it is helpful to have a knowledge of a design process. The process/methodology presented here is *data-driven*. A data-driven approach means that the data that will populate the database is the focus rather than the applications which eventually use the data. Unfortunately, design methodologies are not popular with those who develop databases. Often the DBMS is used as a means to developing a database without going through a logical process. A database implemented in a specific DBMS should be the end product of a complete design process. Hence, many databases developed are inadequate or inefficient in meeting the demands of users and their applications.

A well known design process is shown in Figure 1. In the first phase, database designers interview users to understand and document data requirements. The

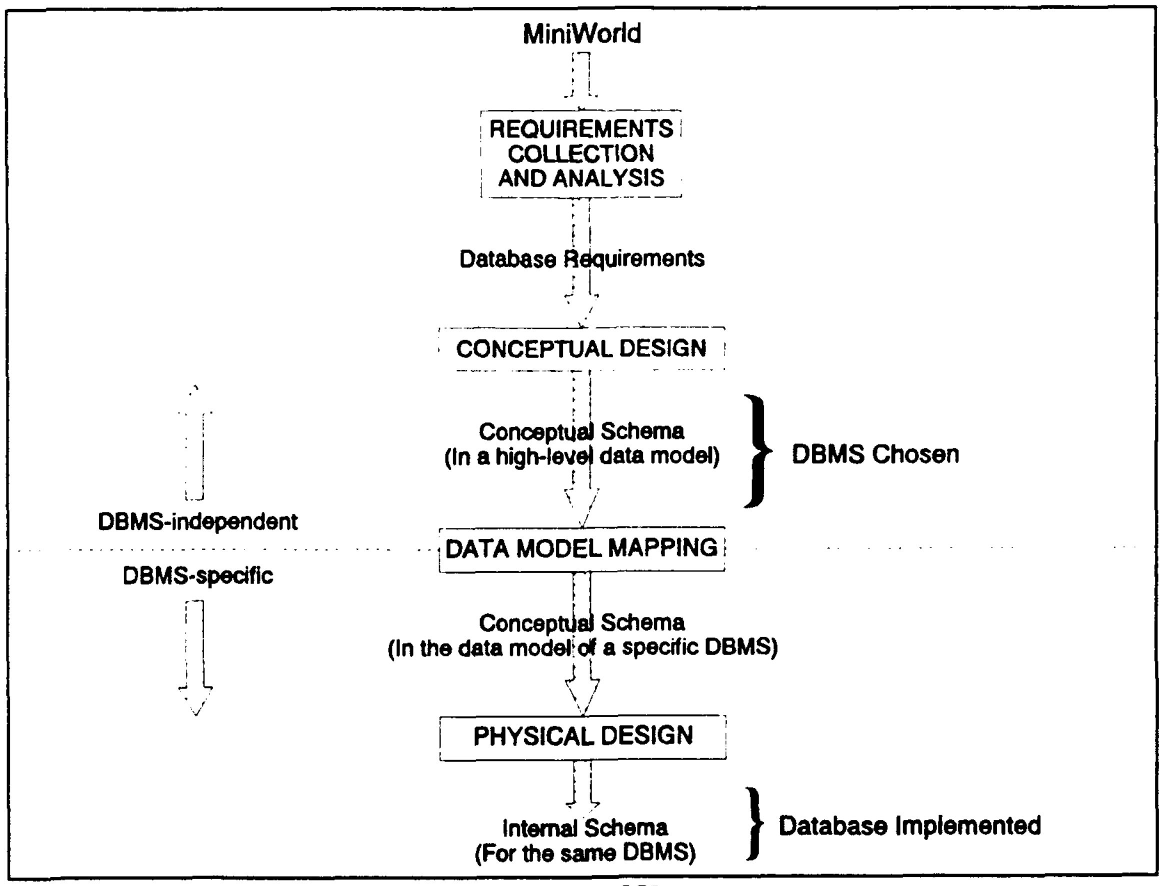
MiniWorld
REQUIREMENTS COLLECTION AND ANALYSIS
Database Requirements
CONCEPTUAL DESIGN
Conceptual Schema (In a high-level data model)
DBMS Chosen
DBMS-independent
DBMS-specific
DATA MODEL MAPPING
Conceptual Schema (In the data model of a specific DBMS)
PHYSICAL DESIGN
Internal Schema (For the same DBMS)
Database Implemented

Figure 1. Database design process [Ref. 6:p. 38].

result of this phase is a set of concise database requirements.

The second phase is the conceptual database design. The conceptual design attempts to identify the structure of data in the form of data objects and relationships among those objects. A conceptual schema of the database is produced by using a high-level data model. This model should be an accurate description of the data requirements of the users. The conceptual schema used for this thesis will be presented using the Entity-Relation (ER) model which is a tool for database modeling and design A brief overview of using the ER diagram to model data appears in section B of this chapter.

The next step in the design process is data model mapping wherein the conceptual data model is transformed into a data model of a specific DBMS. It can be regarded as the logical design of the database. Logical models used to describe the logical schema of the database are classified as relational, network, and hierarchical. We are only concerned with relational models here, although the result of a logical design does not require the use of a relational model. Transforming the conceptual model into a relational model is outlined in section E.

The last step is the physical design phase where the internal storage structures (schema) and access methods are defined for a specific DBMS.

2. Conceptual Design Approach

In phase two of the database design process, a high-level conceptual model is developed which consists of a conceptual schema that describes the structure and information content of a database. As stated earlier, the ER model will be used in this thesis to represent the conceptual schema. The ER model is the most popular formal structure for conceptual data representation. It is based on a few simple modeling concepts and has an effective graphical representation where each element of the model is mapped to a distinct graphic symbol.

The conceptual approach to database design is deliberate in that it follows an engineering-like regimen requiring the analyst to think, discuss, and reason. Automatic tools cannot assist this process significantly because the designer has the responsibility for understanding the requirements and transforming them into a conceptual schema. Once an initial conceptual schema has been established, tools are available for data model mapping or quick prototyping. This includes fourth-generation languages (4GL) such as INFORMIX-4GL. These tools can be used for screen, application, and report generation.

The most successful designs are achieved by complete cooperation between the designer and the user, who is responsible for describing requirements and explaining the meaning of the data. Another advantage of this cooperation, or joint development, is that users involved in the design process are more willing to accept and use the system.

Batini, et al., give three advantages of using the conceptual design approach over using a particular DBMS to develop the schema.

1. The choice of the target DBMS can be postponed, and the conceptual schema can survive a late decision to change the target DBMS.

2. If the DBMS or application requirements change, the conceptual schema can still be used as a starting point of the new design activity.

3. Different databases, described through their conceptual schema, can be compared in a homogeneous framework. This feature eases the building of federated systems from several preexisting databases and the creation of an integrated data dictionary [Ref. 4:p. 11].

It is important that the conceptual schema be included with all the database specifications. Together with the database documentation, the data schemes and applications can be easily understood. The conceptual schema is an important document for future maintenance and upgrade activities.

B. ENTITY-RELATIONSHIP MODEL

An Entity-Relationship (ER) diagram is a technique for representing the logical structure of a database in a pictorial manner [Ref. 4:p. 587]. It is a popular and widely accepted method of representing features of any given structure. It is important to realize that it is developed independently of any specific DBMS.

The Entity-Relationship (ER) diagram is a tool to facilitate communication to verify data requirements between the database designer and the end-user. "The goal of conceptual schema design, where the ER approach is most useful, is to capture real-world data requirements in a simple and meaningful way that is understandable by both the database designer and the end-user" [Ref. 5:p. 2]

There are three basic classes of objects represented in an ER diagram: entity, attribute, and relationship.

1. Entity

a. Regular Entity

The first class of object is the **entity**. An entity is a representation of a real-world object. This is the principal data object about which information is collected. It can denote a person, place, thing, or event. In an ER diagram, a rectangle represents an entity. For example, Figure 2 shows two entities from the MILSATCOM Requirements Database (MRDB), NETWORK and CONNECTIVITY. NETWORK is the set of requirements a user has for MILSATCOM resources while a CONNECTIVITY is the point-to-point link within a network.

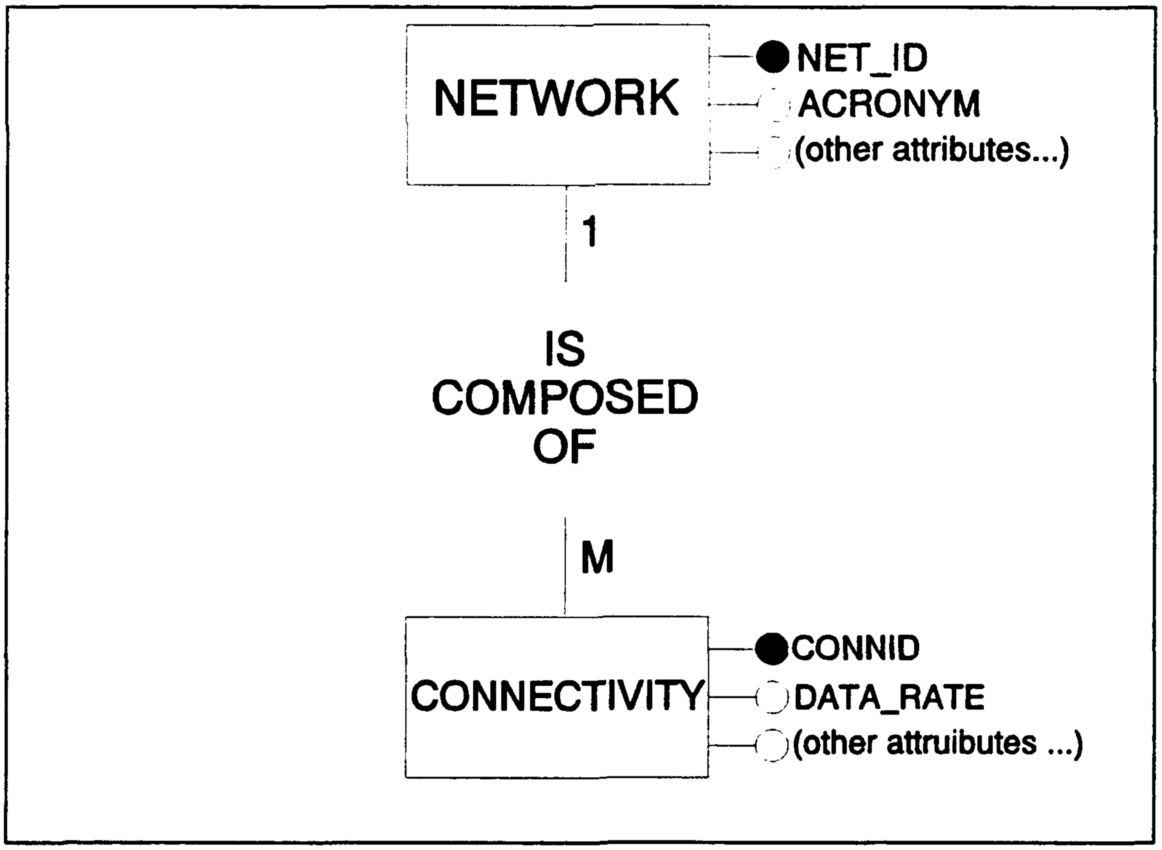

Figure 2. Relationship between two entities.

An **instance** is a particular occurrence of an entity. In the NETWORK/CONNECTIVITY example an instance of connectivity would be a connection between a terminal at Falcon Air Force Base, Colorado and a terminal at Thule, Greenland. This instance would also include values for its attributes such as mode of operation, data rates, throughput rates, etc.

b. Weak Entity

Weak entities are derived from the attributes of one or more parent entities, they cannot exist on their own. Consider the entity COMPONENT (related to CINC) who is responsible for network requirements. Two values for COMPONENT can be identical but can still represent different entities. For example, PACAF (Pacific Air Forces) can be either a component of USAF or CINCPAC. Although PACAF in this example is the same entity, the two different chains of command are represented. A weak entity can also be modeled as a multivalued attribute associated with an entity or as a many-to-many relationship between two entities [Ref. 5:p. 15]. In an ER diagram, a double-lined rectangle represents a weak entity (For an example of a weak entity, see the COMPONENT entity in the ER diagram in Figure 6, Chapter III.)

2. Attribute

The second class of objects in an ER model is the **attribute.** An attribute provides descriptive information about an entity or a relationship. An attribute is also known as a **field**. The order of attributes within an entity or relationship is not important. They are represented on an ER diagram by using either ovals connected to the entity or by small bubbles (see Figure 2).

A special class of attribute is called a **key** attribute. Its purpose is to provide uniqueness to each instance of an entity. For example, if the name of a network is a key attribute, then each instance should uniquely identify a network and no two networks could have the same name. It is a constraint that prohibits any two entities from having the same value for the key attribute at the same time.

Attributes can be **single-valued** or **multivalued**. A single-valued attribute is the most common. For example, the NETWORK entity can have only one primary mission, therefore the attribute PRIMARY_MISSION is considered a single-valued attribute. A multivalued attribute is an attribute of an entity that can have a different number of values. In the MRDB, NETWORK can have zero, one, or more secondary missions, therefore, the attribute SECONDARY_MISSION is a multivalued attribute.

A **derived** attribute is one whose value is determined from the value(s) of one or more other attributes within the same entity or from one or more related entities.

Null refers to an attribute with no value associated with it. A null is not a zero value, nor is it a field with only blanks. It can be considered to be *not applicable* or *unknown*. The following are rules for null values:

- Null values are allowed only for foreign keys (foreign keys will be defined later in this section) of optional entities in an entity table.
- Null values are not allowed for foreign keys of mandatory entities in an entity table.

- Null values are not allowed for any key in a relationship table because only complete row entries (records) are meaningful in a relationship table [Ref. 5:p. 62].

3. Relationship

The third class of objects is the **relationship.** A relationship represents the real world association between one or more entities. Relationships are usually defined using verbs to describe roles between entities. In an ER diagram, relationships are represented by diamonds.

Relationships are described as being "one-to-one," "one-to-many," or "many-to-many." This is known as the **cardinality ratio** and refers to the mapping between instances of each entity. It describes the constraint on the number of entity instances that are related through a relationship. For example, NETWORK and CONNECTIVITY have a one-to-many (1:M) relationship. One network can have many connectivities but a connectivity can belong to only one network.

A relationship can have attributes just like entities. For a one-to-one (1:1) relationship between entities, attributes can be included as attributes for either the entity or the relationship. For a 1:M relationship, a relationship attribute can be included only as an attribute of the entity on the "many" side of the relationship. For both of these relationship types (1:1 and 1:M), "the decision as to where a relationship attribute should be placed--as a relationship type attribute or as an attribute of a participating entity type--is determined subjectively by the schema designer" [Ref. 6:p. 52]

In many-to-many (M:M) relationship types, the value for a particular attribute is determined by a combination of contributing entities and not by any single entity. This attribute must be associated with the relationship. This is because entities from any participating entity can be in numerous relationship instances.

C. RELATIONAL MODEL

1. Tables

A relational model represents the data in a database as a collection of **relations**. A relation can be thought of as a table. Each row in the table represents a set of related data values. These values are facts that describe an entity or relationship in the ER model. In the relational model for the MRDB, some examples of tables are NETWORK, CONNECTIVITY, POC, DOCUMENT, etc. (see Figure 10, Chapter III).

2. Key Attributes in a Table

a. Primary Key

A primary key is an attribute or group of attributes that uniquely identify each element in a relation (representing an entity or a relationship). A primary key can be **atomic** (single value) or **composite** (more than one field). A primary key is also referred to simply as a **key** or an **identifier**. In relational model, a primary key is identified by underlining the attribute(s) that makes up the primary key (see Figure 10, Chapter III). In the ER model, primary keys are represented using solid bubbles connected to the entity (see Figures 2, 3, or 4).

b. Foreign Key

An attribute is a **foreign key** if it is not the primary key of an entity but its elements, or values, are constrained to be values of the primary key of another relation [Ref. 4:p. 117]. For example, in the relational model (Figure 10, Chapter III), *nomenclature*, *user_id*, and *geo_reference_code* are foreign keys in TERMINAL. The foreign key *nomenclature* is the primary key for TERMINAL_TYPE, *user_id* is the primary key for USER, and *geo_reference_code* is the primary key for GEO_LOC.

D. GUIDELINES FOR CLASSIFYING ENTITIES AND ATTRIBUTES

1. Entities with only a identifier

Entities should contain descriptive information. If there is information about an object, the object should be classified as an entity. If an object requires only an

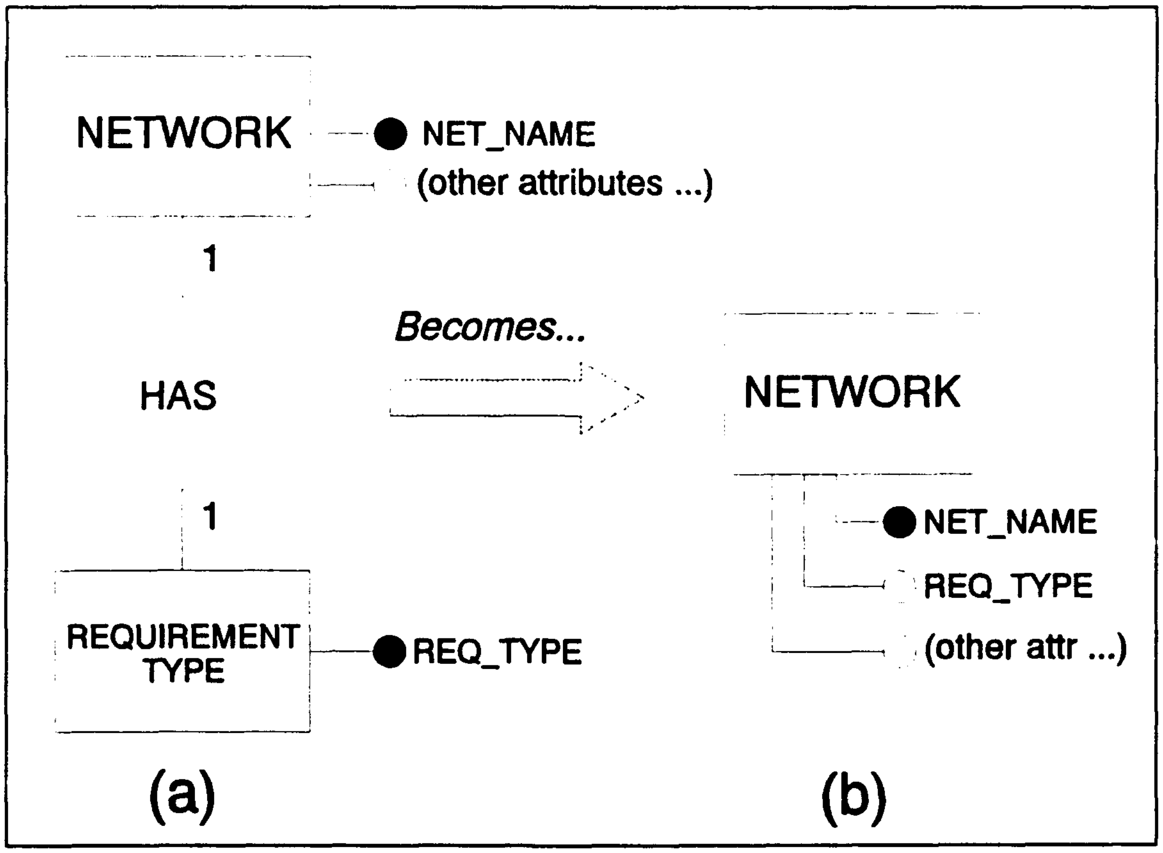

Figure 3. Entity becomes an attribute.

identifier, the object should be classified as an attribute [Ref. 5:p. 36]. For example, consider the entity REQUIREMENT_TYPE in the MRDB (see Figure 3-a). The type of requirement for the network should be classified as an attribute since the object (REQUIREMENT_TYPE) requires only an identifier (see b).

2. Multivalued Attributes

It is best to treat multivalued attributes as entities. If more than one value of a nonkey attribute corresponds to one value of a primary key, the nonkey attribute should be represented as an entity instead of an attribute. This requires creating an entity for the multivalued attribute. It will have a one-to-many relationship with the entity from

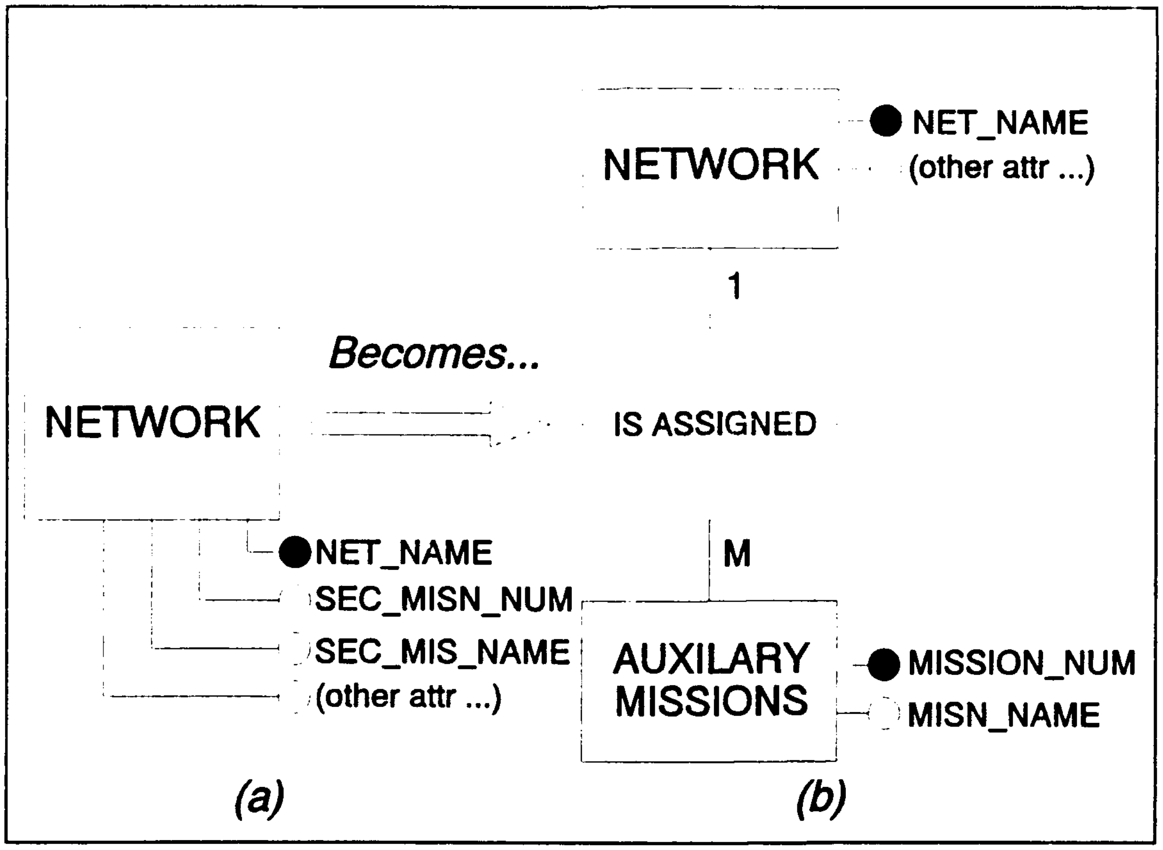

Figure 4. AUXILIARY MISSION is separate entity from NETWORK.

which it was removed. For example, in Figure 4 AUXILIARY_MISSION is a multivalued attribute since a network can have more than one (if any) secondary missions. Therefore, AUXILIARY_MISSION should be represented as an entity rather than as an attribute.

E. TRANSFORMING THE ER MODEL TO A RELATIONAL MODEL

Once the data model is complete, it is ready to be transformed into a conceptual schema in the model of a specific DBMS. This phase is also known as **mapping** the ER model to the relational model (refer to Figure 1).

There are seven steps for mapping an ER model into a relational model [Ref. 6:p. 329]. In the following discussion, we will use the terms "table" and "relation" interchangeably.

1. Regular Entities

Transform each regular entity E into a table T_R with all key and nonkey attributes of E mapping into columns within T_R. The key attribute(s) of E becomes the primary key for T_R.

2. Weak Entities

For each weak entity W in the ER model, create a table T_W. Include all simple attributes of W as attributes in T_W. Additionally, include as foreign keys of T_W the primary key attributes of the relation that corresponds to the owner entity. The primary key of T_W is a composite key consisting of the primary key of the owner and the key of the weak entity.

3. Binary One-to-one Relationships

For binary one-to-one relationships, identify the relations that correspond to the entities that are involved. We will arbitrarily call one A and the other B. Add the primary key of relation A to relation B, thus making that attribute in B a foreign key. Technically, it doesn't matter in which relation the primary key is used. However, look for natural parent-child relationships. For example, in Figure 5 the tables NETWORK and STATS have a one-to-one relationship. NETWORK can be considered the "parent" of the "child" STATS. Therefore, NETWORK's primary key, NET_ID, will be added to the STATS entity.

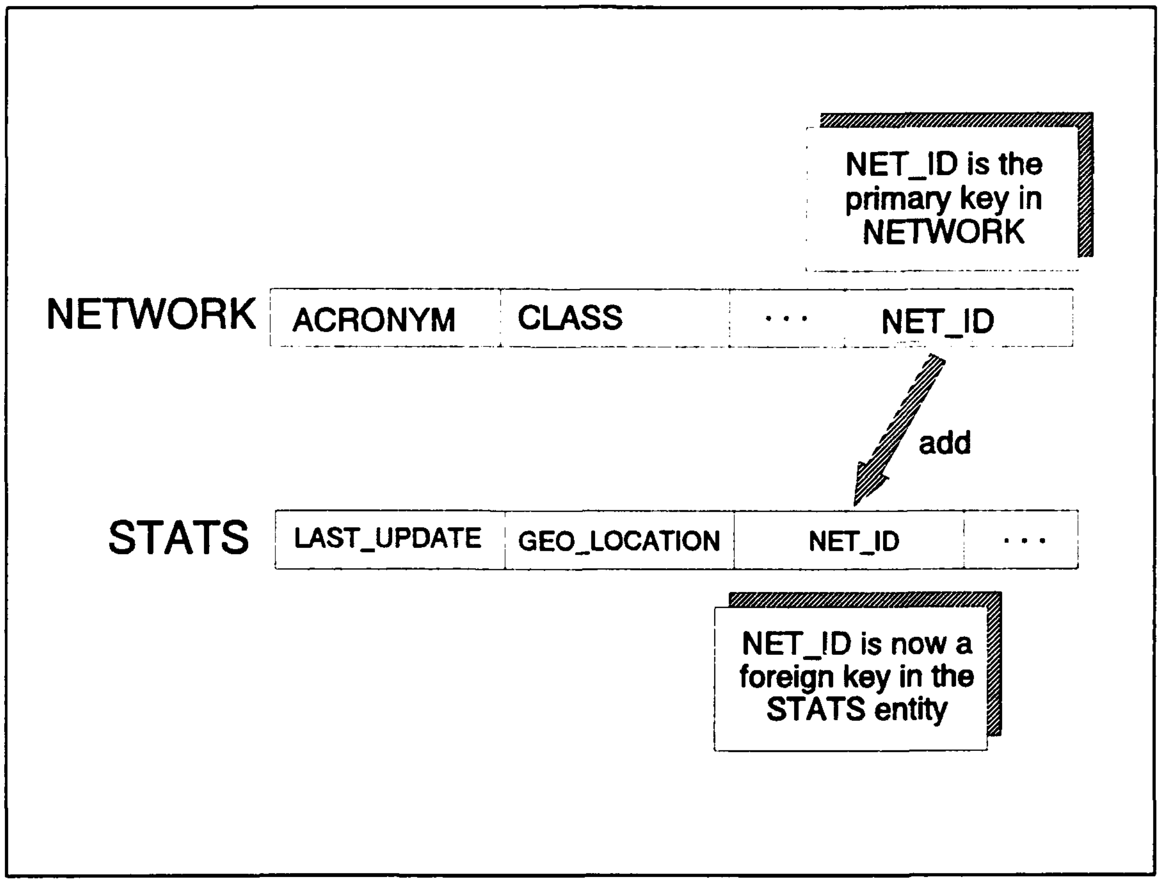
NET_ID is the primary key in NETWORK
NETWORK
ACRONYM
CLASS
. . .
NET_ID
add
STATS
LAST_UPDATE
GEO_LOCATION
NET_ID
. . .
NET_ID is now a foreign key in the STATS entity

Figure 5. Transforming relationship between NETWORK and STATS into tables.

4. Binary One-to-many Relationships

For binary one-to-many relationships add the primary key of the parent ("one" side) entity to the child ("many" side) entity. This attribute will be a foreign key in the child entity. For example, in the MRDB, the tables NETWORK (parent) and CONNECTIVITY (child) have a one-to-many relationship since one network/requirement can have many connectivities associated with it. Therefore, the NET_ID attribute will be added to the table CONNECTIVITY. (This is accomplished in the same way as depicted in Figure 5.)

5. Binary Many-to-many Relationships

Transform every many-to-many relationship in the ER model into a new table and add the primary keys of the related entities to the new table. Also add any attributes of the relationship to that table. The keys from the related entities will form a composite primary key for the new relationship table. Remember the null rules apply for all relationship tables.

6. Multivalued Attributes

For each multivalued attribute create a new table that includes the attribute and the primary key of the entity that represents the multivalued attribute. This is similar to the treatment of multivalued attributes in the ER model. These two steps are mutually exclusive. If the ER modeling process treats multivalued attributes as entities, then this step in mapping to the relational model will not be necessary. (See Section D.2)

7. N-ary Relationships

The final step is to transform every ternary or higher n-ary relationship into a new table. Keys from the associated entities make up the primary key of the new table. However, if one of the participating entities has a constraint of 1 (for example 1:M:M) then the key of that entity does not need to be included as part of the primary key of the relationship table. However, it will be added as a foreign key of the relationship table.

F. NORMALIZATION

Normalization is a step-by-step process for analyzing data in relations and redesigning these relations to prevent update anomalies [Ref. 7:p. 120]. It is built around the concept of *normal forms*. A table or *relation* is said to be in a particular normal form if it satisfies a certain set of rules.

Numerous normal forms have been defined. Most common are the rules for first, second, and third normal form (1NF, 2NF, 3NF). Additionally, there are other normal forms such as Boyce-Codd normal form (BCNF), 4NF, and 5NF. Fourth normal form considers an additional constraint called "multivalued dependency" while 5NF considers a constraint called "join dependency."

Normal forms are defined by considering key constraints and **functional dependencies**. A field Y is said to be *functionally dependent* on field X (written as $X \rightarrow Y$) if it is invalid to have two records with the same X value but different Y values, i.e. a given X value must always occur with same Y value [Ref. 7:p. 121]. For example, in the table TERMINAL_TYPE, *antenna_size* → *nomenclature*. This means that for an occurrence of the terminal type AN/FSC-79, the antenna (dish) size will be 60 feet. (Note that the converse, *nomenclature* → *antenna_size*, is not true. More than one terminal type can have an antenna that is 60 feet diameter.)

The reason for normalizing is to prevent insertion, deletion, and update anomalies. Good top-down database design methodology tends to generate fully normalized designs. It isn't necessary to normalize a database to the highest possible form [Ref. 4:p. 560]. Relations can be left in lower normal forms for performance reasons.

1. First Normal Form

All occurrences of a record type must contain the same number of fields. A simple design principle that leads to 1NF is that the smallest pieces of information that may be required by users should be allocated to separate attributes, i.e. remove all repeating groups. Relational database theory does not deal with records that have a variable number of fields [Ref. 7:p. 120]. A good summary of what 1NF is trying to accomplish is reflected in the following quote from Lacy-Thompson:

> The first stage of normalization is accomplished by ensuring that all attributes contain only *atomic* values, i.e. values that cannot or will not need to be broken down any farther, and by the removal of repeating groups, rewriting them as new entities with the appropriate identifier or identifiers. [Ref. 8:p. 23]

2. Second Normal Form

A relation is in 2NF if and only if it is in 1NF and every nonkey attribute is fully dependent on the primary key. In other words, every field within a record is either part of the key or provides a fact about exactly the whole key and nothing else.

Second normal form can only be violated when the primary key is composite. Second normal form is violated when a nonkey field is a fact about a subset of the composite primary key. It is important, therefore, to ensure that all nonkey attributes for an entity are dependent on the whole of the key, and not just a part of it.

3. Third Normal Form

A relation is in 3NF if and only if it is 2NF and every nonkey attribute is mutually independent and dependent on the primary key. When a situation arises when a nonkey attribute is functionally dependent on another nonkey attribute it is referred to as a *transitive dependency*. Therefore, a transitive dependency is when a third attribute, Z, is functionally dependent on Y which is functionally dependent on X (written as $Z \rightarrow Y \rightarrow X$).

In 3NF, we remove any transitive dependencies by separating the attributes that are dependent on another nonkey attribute and creating a new relation (two relations will exist, one with {X, Y} and one with {Y, Z}). An example of this will be given in Chapter III.

4. Boyce-Codd Normal Form

Boyce-Codd normal form (BCNF) is a stronger form of 3NF. Two terms need to be defined. First, a **candidate** key is one or more attributes that can uniquely identify a record in a relation. A primary key is selected from the set of candidate keys. For example, in the table NETWORK, the network's acronym or full name can uniquely identify a network. Each could be considered a candidate key. Second, a **determinant** is any attribute on which some other attribute is functionally dependent.

A relation is in BCNF if and only if every determinant is a candidate key [Ref. 4:p. 543]. In other words, only determinants are candidate keys. BCNF takes into account all candidate keys, not just the primary key.

Normalization often results in a relatively large number of tables. Additionally, there is some duplication of attributes, but a database in 3NF or BCNF offers an assurance of data integrity. When the database grows and expands in the future, disruption will be minimized if restructuring is required [Ref. 8:p. 26]. The next chapter develops a normalized version of the MRDB that complies with the rules for 1NF, 2NF, 3NF and BCNF.

III. MILSATCOM REQUIREMENTS DATABASE

This chapter will use the concepts presented in the previous chapter to develop a normalized version of the MRDB. First, an overview of the MRDB is presented. Second, the structure of the current MRDB is examined and violations to the various normal forms are identified. Third, using the ER model, a conceptual database schema of the MRDB is generated. Fourth, a relational schema is created from the ER model. Finally, normalization is discussed as it relates to the new MRDB schema.

A. BACKGROUND

1. Database Origin

The MILSATCOM Requirements Database (MRDB) was developed for Headquarters Air Force Space Command (AFSPACECOM) and U.S. Space Command (USSPACECOM) by ARINC Research Corporation of Colorado Springs, Colorado to track military satellite communication (MILSATCOM) requirements. AFSPACECOM/DRFC is the executive agent for the MRDB and can be considered its "owner".

The MRDB was developed as an information management tool to support AFSPACECOM and USSPACECOM in tracking MILSATCOM requirements, planning future systems acquisition, and conducting system architecture studies. It was designed to overcome the shortcomings of the old Users Requirements Database (URDB) and its follow-on, the Integrated SATCOM Database (ISDB).

The URDB was the original tool for documenting MILSATCOM requirements. It was a set of flat files that the Defense Information Systems Agency (DISA) used to track Joint Chiefs of Staff (JCS) validated SATCOM requirements. The MRDB was to track not only JCS validated requirements, but also non-validated requirements in a relational database environment. The MRDB was originally developed as a prototype without constructing a conceptual schema.

The URDB was upgraded by DISA and became the ISDB. According to Memorandum of Policy No. 37, the ISDB is the database of all JCS approved MILSATCOM requirements. Like the URDB, the ISDB contains shortcomings and does not support the information needs of AFSPACECOM and USSPACECOM.

2. User Community

The MRDB user community consists of unified commands (referred to as CINCs)[1] or government agencies that require MILSATCOM support. Requirements are usually associated with a CINC's service component (e.g., AFCENT, ARCENT, etc.) Requirements, however, can come directly from the CINC (a joint requirement) or a subcomponent of a service (e.g., 82nd Airborne Division, 4th Air Wing, etc). It is often difficult to associate or identify a certain set of requirements with a particular CINC, component, or subcomponent.

[1]Each unified command is represented in the MRDB by CINC. e.g. U.S. Central Command is represented as CINCCENT, Forces Command as CINCFOR, U.S. Space Command as CINCSPACE, etc.

3. MRDB Data

The MRDB data are maintained by ARINC and are collected periodically from the ISDB, the Worldwide Online System (WWOLS)[2], and from various system managers.

The data from the ISDB can be transferred electronically and then converted to the MRDB format. However, according to ARINC, the formats for the ISDB and MRDB don't match and some anomalies exist when data are transferred from the ISDB to the MRDB.

Recently, a conference was conducted by ARINC and USSPACECOM with attendees from all the CINCs and agencies to "scrub" the MRDB and examine every requirement. This process greatly improved the accuracy of the MRDB.

B. Current Design

This section will discuss the version of the MRDB as provided by ARINC. This version was current as of January 1993. Entities, tables, and relations will be designated by using capital letters, e.g., NETWORK, and attributes or fields by italics, e.g., *acronym*.

[2]WWOLS is a database that contains information that breaks each connectivity within a network into physical components (land lines, LOS, satellite links, etc). A reference to WWOLS is in the MRDB through the CCSD attribute in the CONNECTIVITY table. A CCSD number will exist only if the connection has been implemented.

1. Tables and Attributes

The MRDB contains 25 tables with 244 attributes. Half of these attributes belong to 3 tables whereas 13 tables have 3 or fewer fields and six tables have only one field. The largest table is NETWORK with 79 attributes.

To determine the level of normalization of the MRDB, it is necessary to examine the MRDB for various levels of normal forms. I will be discussing violations to 1NF, 2NF and 3NF.

2. Normal Forms

a. First Normal Form

First normal form (1NF) is accomplished by ensuring all attributes in each table are atomic or *nondecomposable*, that is, no repeating groups exist. A repeating group can be found in REFERENCE where *pagenum* consists of beginning page number and ending page number. For example, a *pagenum* value of "C-21-1/6" refers to pages 21-1 to 21-6 in annex C of the AFSCN Communications Resources Document. Without breaking down these values, it would be difficult to find all networks referenced by Annex C. From this example, we can see that this field contains a repeating value of page numbers. If the user wants to separate these values, it would be a long and tedious manual process.

b. Second Normal Form

Violations to second normal form (2NF) involve nonkey attributes which are dependent on a subset of a composite primary key. Only one table, NETUSER, in the MRDB has a composite primary key and it does not violate the rules for 2NF. The key fields of NETUSER are *username, mobility,* and *platform.* The nonkey fields are *place, state, radius,* and *region.* Each nonkey field is dependent on the whole composite primary key. Although *radius* is applicable for only mobile or transportable terminals, it is still dependent on the whole composite primary key. For example, a terminal that is mobile for one user may have a *radius* value of 500, where another terminal that is also mobile for a different user can have a *radius* value of 900.

c. *Third Normal Form*

Violations of rules for third normal form (3NF) occurs when a nonkey attribute is functionally dependent on another nonkey attribute. This usually leads to transitive dependencies as discussed in Chapter II.

A violation of 3NF occurs in the CONNECTIVITY table. A unique CONNECTIVITY record is identified by the *connid* field since that field is defined as a *serial*[3] field. There are several fields which are not facts about the connectivity (the link between two terminals), but rather are facts about the actual terminal at either end of the connection. In other words, some nonkey attributes are functionally dependent on other nonkey attributes.

The attributes *org_platform* and *org_mobility* are functionally dependent on the attribute *org_terminal* which in turn is functionally dependent on *org_trm_id*; hence, *(org_platform, org_mobility)* → *org_terminal* → *org_trm_id.* This can be further

[3]Serial is an attribute type within INFORMIX. This type of field will have its value automatically assigned. It is intended to be used as a primary field.

explained as follows: a given value of *org_trm_id*, e.g., "88", will always occur with the same value of *org_terminal*, e.g., "commercial." Also, for each given value for *org_terminal*, such as "commercial" or "FSC-78," the attribute *org_mobility* will have the same value, "FIX."

These violations of normal forms can be the result of a schema that has been "denormalized" for performance reasons or, most likely, has not been normalized in the first place. In the following section a conceptual model of the MRDB is developed based on the ISDB, the existing MRDB schema, and interviews with users.

C. CONCEPTUAL MRDB DESIGN

The first phase of an effective database design is Requirements Collection and Analysis which involves collecting information about the intended use of the database, examining the existing system, documentation, and interviewing users.

The second phase, Conceptual Database Design, produces a conceptual schema for the MRDB from the information collected during requirements analysis. The conceptual schema will be represented using an Entity-Relationship (ER) model as follows:

- all entities are identified,
- attributes of those entities are determined,
- relationships among entities are defined.

After the ER model is complete, it will be mapped into a relational model whose tables will then be examined to determine if further normalization is necessary. The main features of the ER model derived from this process are shown in Figure 6 and consist of the following:

- NETWORK is a set of network requirements and is composed of one or more connectivities. A NETWORK can have one or more POC's and can be referenced by one or more documents. Each network is described by the network name, an acronym, and several parameters that describe the network's survivability, availability, and performance. No two networks can have the same name or acronym.

- CINC (or agency) is the organization responsible for generating the requirements for the network. A CINC can be responsible for one or more networks. Each CINC can have zero or more components who can be the responsible organization for a network requirement. A CINC is described only by name. COMPONENT is a weak entity to CINC in that each COMPONENT is described by the parent CINC and the name of the COMPONENT.

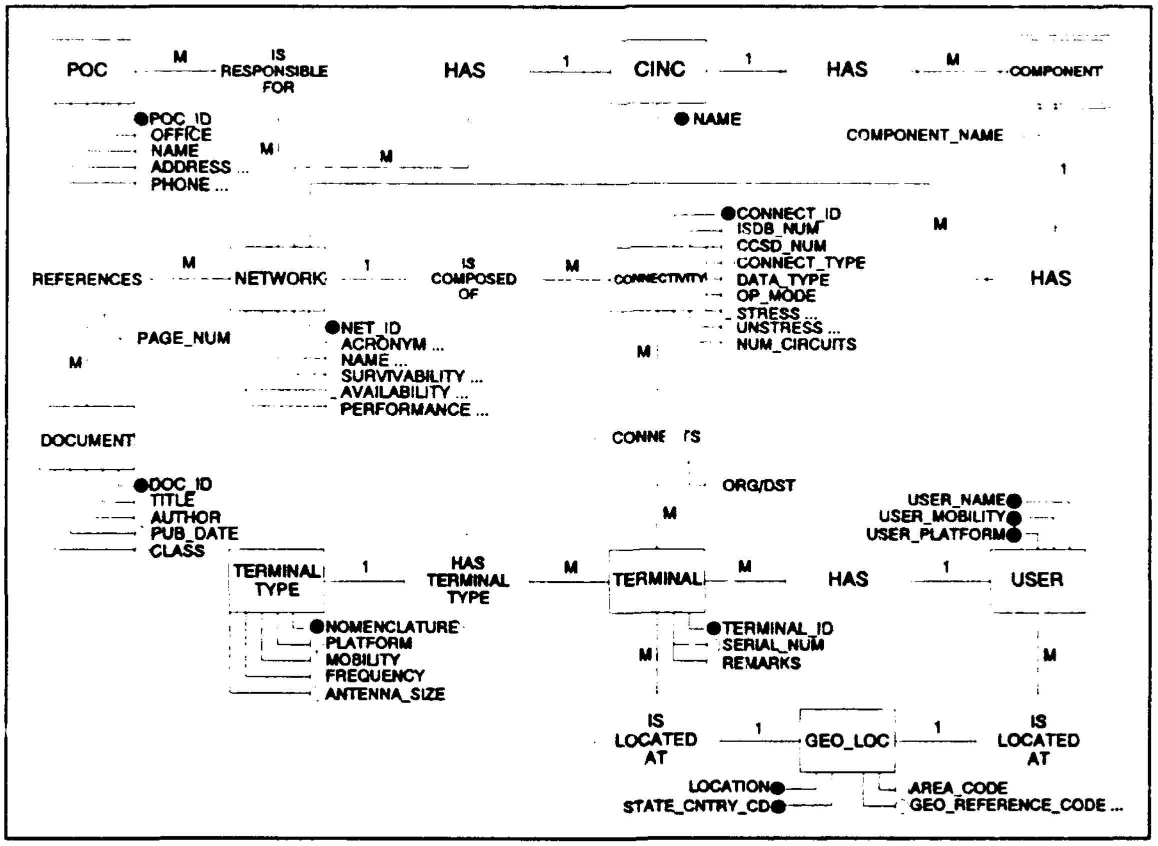

Figure 6 ER model for the MRDB.

- POC (point of contact) is responsible for one or more sets of network requirements. Each POC will be describe by name, organization, address, commercial phone number, DSN number, secure fax number, unsecure fax number, and STU-III number, and a identification number. No two POCs have the same identification number.

- DOCUMENT references the requirement for the existence of the network. DOCUMENT can reference one or more networks. Each instance of a reference will have a page number for the NETWORK and the DOCUMENT referenced.

- CONNECTIVITY describes the connection between exactly two terminals. If the connection has been implemented, then it will have an associated ISDB number and CCSD number. Each unique connectivity will be described by a connection identifier, the type of data on the connection, the mode of operation, data rates, throughput rates, and more. No two connections have the same identifier.

- A TERMINAL is a piece of communication equipment that has a user (owner), is at a particular location (if fixed), and is of a particular model (type). A TERMINAL can belong to one or more connectivities. Different terminals that are the same model and at the same location are differentiated by serial number. A mobile or transportable terminal will not be given a location.

- A TERMINAL_TYPE describes the characteristics of the terminal. There can be one or more of each type of terminal. Each TERMINAL_TYPE will be described by nomenclature, type of platform, mobility, frequency, antenna size, etc. No two TERMINAL_TYPEs can have the same nomenclature.

- A USER is a unit that owns one or more terminals. It is identified by name, usually the unit's name (for example, 101 AD, 4 AW, etc.). A user will be at only one geographic location.

- A GEO_LOC (geographic location) will name a unique location. This name can be a military installation, a remote site, or a foreign country. It is described by a name, country code, region code, geographic reference code, the U.S. region code, and the latitude/longitude. No two places can have the same geographic reference code or the same name/country code pair.

D. DATA MODEL MAPPING

The third phase of the database design process is to create a relational schema from the ER diagram in Figure 6. This is done by using the rules for mapping outlined in Chapter II.

1. Regular Entities

For each regular entity in the MRDB, create a corresponding relation. These relations are NETWORK, DOCUMENT, POC, CINC, CONNECTIVITY, TERMINAL, TERMINAL_TYPE, USER, and GEO_LOC (Figure 7). The primary keys for the

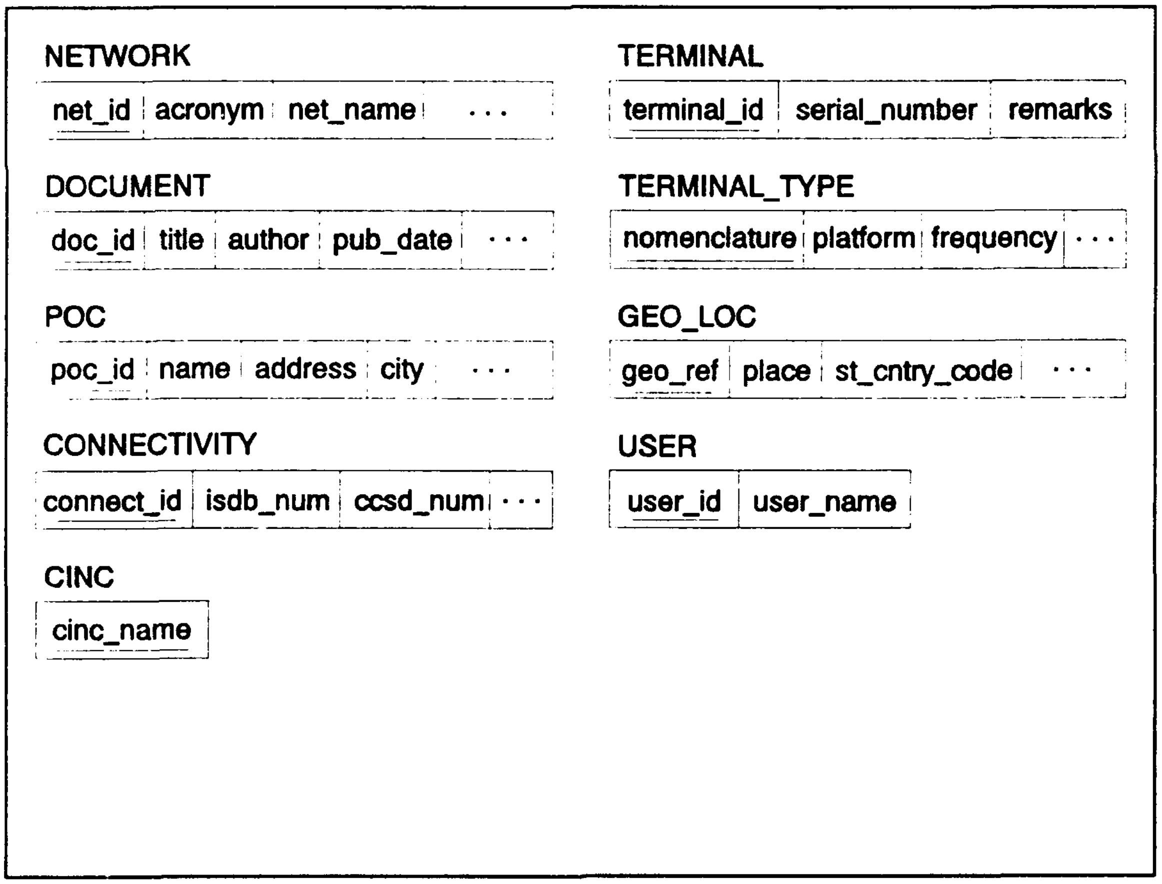

Figure 7 Relational database schema after mapping regular entities.

relations are: *net_id*, *doc_id*, *poc_id*, *cinc_name*, *connect_id*, *terminal_id*, *nomenclature*[4], *user_id*, and *geo_ref*.

2. Weak Entity

For the weak entity COMPONENT with owner CINC, create a new relation. It will be called COMPONENT. Additionally, include as a foreign key attribute the primary key of the relation CINC. The primary key for the relation COMPONENT will

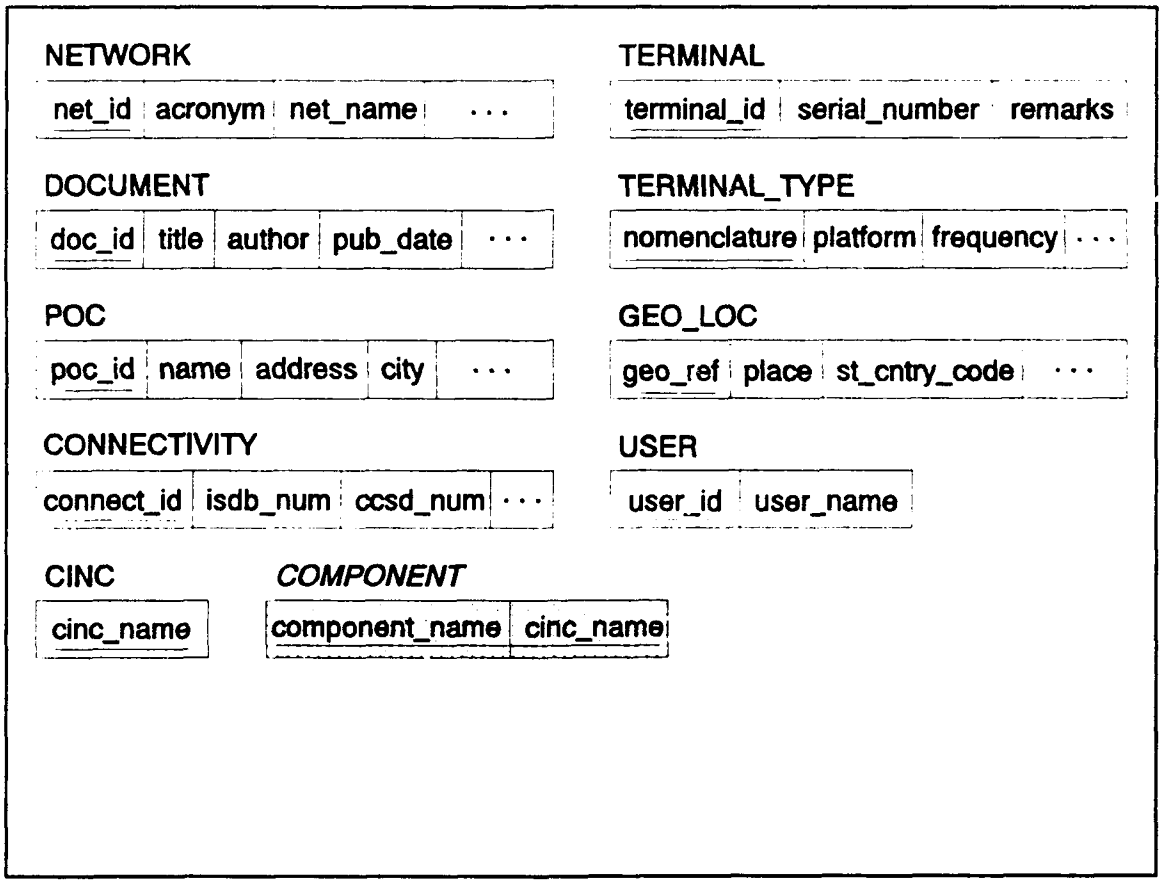
NETWORK
net_id | acronym | net_name | ...
TERMINAL
terminal_id | serial_number | remarks
DOCUMENT
doc_id | title | author | pub_date | ...
TERMINAL_TYPE
nomenclature | platform | frequency | ...
POC
poc_id | name | address | city | ...
GEO_LOC
geo_ref | place | st_cntry_code | ...
CONNECTIVITY
connect_id | isdb_num | ccsd_num | ...
USER
user_id | user_name
CINC
cinc_name
COMPONENT
component_name | cinc_name

Figure 8 Relational database schema after mapping the weak entity.

[4]*Nomenclature* is the model for a terminal. For example, TSC-100, GSC-43, ARC-171, etc.

be the combination of *component_name and cinc_name*. Figure 8 shows the result of mapping the weak entity COMPONENT.

3. One-to-many Relationship

For each binary 1:M relationship, identify the relation that corresponds to the entity at the "many" side of the relationship. Include in that relation, as a foreign key, the primary key of the relation corresponding to the entity on the "one" side of the relationship. The following list details the mapping of the 1:M relationships. Figure 9 shows the relational model following the completion of mapping 1:M relationships.

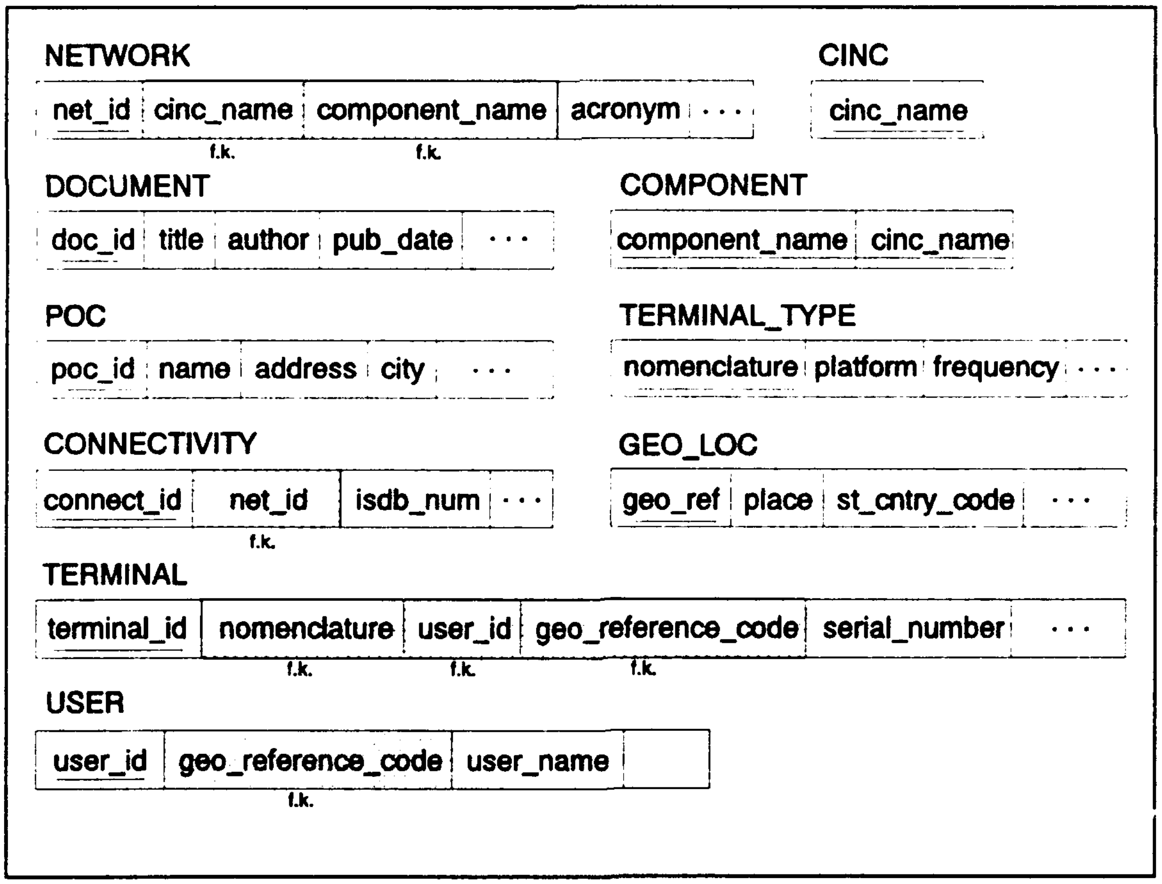

Figure 9 Relational database schema after mapping 1:M relationships.

- For the CINC_HAS_NETWORK relationship, add the primary key *cinc_name* as a foreign key to the relation NETWORK.

- For the COMPONENT_HAS_NETWORK relationship, add only the partial key *component_name* to NETWORK, since *cinc_name* was already added to NETWORK in the previous step.

- For the relationship IS_COMPOSED_OF add the primary key *net_id* to the relation CONNECTIVITY as a foreign key.

- For the relationship HAS_TERMINAL_TYPE add the primary key *nomenclature* to the relation TERMINAL as a foreign key.

- For the USER_HAS_TERMINAL relationship, add the primary key *user_id* to the relation TERMINAL as a foreign key.

- For the TERMINAL_IS_LOCATED_AT_GEO_LOC relationship add the primary key *geo_reference_code* to the relation TERMINAL.

- For the USER_IS_LOCATED_AT_GEO_LOC relationship add the primary key *geo_reference_code* to the relation USER.

4. Many-to-many Relationship

For each M:M relationship create a new relation. Include as foreign key attributes the primary keys of the relations that represent the participating entities. The key of the intersection relation is the combination of foreign keys. Figure 10 shows the result of this mapping. The following list gives the details of this process.

- The first M:M relationship is REFERENCES (a document REFERENCES network requirements). A new intersection relation, NET_DOC, is created. The primary keys *net_id* and *doc_id* are added as foreign keys to the new relation. The primary key for NET_DOC is the combination of the foreign key attributes (*net_id* and *doc_id*). This relation will also have one attribute, *page number*.

- The second M:M relationship is IS_RESPONSIBLE_FOR (a POC is responsible for a specific set of network requirements). Create a new intersection relation NET_POC and add the primary keys net_id and poc_id as foreign keys. The primary key for this relation is the combination of net_id and poc_id. Their are no additional attributes.

- The third M:M relationship is CONNECTS. Create a new intersection relation TERMINAL_CONNECT and add the primary keys *terminal_id* and *connectivity_id* as foreign keys. Again, the primary key for this relation will be the combination *terminal_id* and *connectivity_id*. This relation will have an additional attribute, *org_dst*. This attribute will designate the terminal as either the originating or destination terminal.

A complete relational schema model showing tables and relationships between tables can be found in Appendix A and a listing of all attributes for each relation can be found in Appendix B.

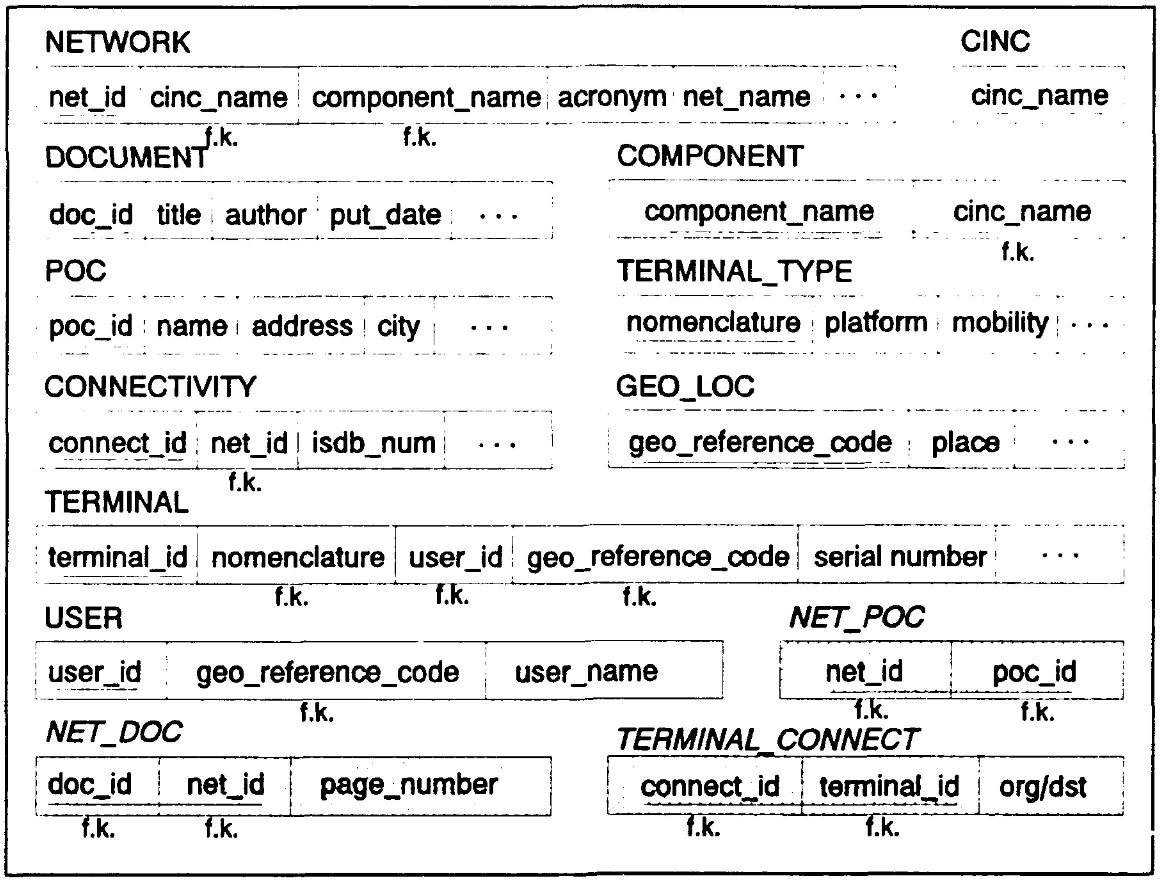

Figure 10 Relational database schema after mapping M:M relationships.

E. NORMALIZATION

The last step in the database design process is to take the relational model and design and implement a physical database within a relational DBMS environment. This process involves selecting the storage structures and access paths to guarantee good performance. To make intelligent physical design decisions, more information needs to be gathered concerning frequency of queries/updates and all applications that will run on the MRDB.

Implementing a good physical design is easier when a comprehensive database design process is followed. A good top-down methodology will tend to generate a fully normalized logical design. The reason for this is that the process of developing normal forms is carried out in the careful development of the conceptual and relational models.

The relational model shown in Figure 10 is in 3NF and BCNF since every candidate key is a determinate. The relational model was designed in a way as to take all attributes from one entity or relationship and make a single relation. This ensures that the relationships between each object in the ER model were correctly translated into the relational model.

For example, in the relation DOCUMENT the candidate keys are *doc_id* or *doc_title*. All nonkey attributes, *author, pub_date,* and *classification*, are functionally dependent on the candidate keys and no other nonkey attribute. Similarly, for every relation, all nonkey attributes are dependent upon determinants that are candidate keys. Hence, the schema can be considered to be in 3NF or BCNF.

The next chapter addresses the tradeoff between normalization and performance by comparing query response times of the normalized MRDB and the old MRDB.

IV. DATABASE COMPARATIVE PERFORMANCE ANALYSIS

Balancing the need for normalized structures and the performance of a relational database is a challenge. On one hand, short retrieval times require that data be stored in very few physical files to eliminate expensive joins. On the other hand, preventing update anomalies requires that data structures be simple and normalized. Normalization results in more tables and, therefore, can lead to performance degradation for specific applications. In order to improve performance for these applications, the normalization process must be reversed. This process is known as "denormalizing," or, combining tables.

Two questions concerning the MRDB are: 1) Do users require fast response time for queries? and 2) Can favorable query response times be sacrificed in order to have a structure that eliminates update, insert, and delete anomalies?

Some insight to these questions was gleaned from interviews with the users. In most cases, the turn-around time to receive desired reports was hours or even days. When the database was queried, the activity was heavy, but sporadic. However, because only one person could query the database, some of these heavy periods lasted all day and night for up to two weeks. To improve the situation, two copies of the database were used, but problems were encountered later when the updates were combined into one "master" MRDB. The contractor stated that although the MRDB experienced some update anomalies, they (the contractor) were aware of them and were able to fix them fairly well.

Therefore, the answer to the first question seems to be that fast response times in a matter of seconds is not a major requirement of the users, whereas the answer to the second question is that sacrificing some query response time would not hinder the efficiency of the MRDB. A normalized structure would help eliminate the update anomalies, but at the possible expense of increased query time response. In summary, "normalized design enhances the integrity of the data by minimizing redundancy and inconsistency, but at some possible performance cost for certain retrieval applications" [Ref. 7:p. 121]. This performance cost does not seem to be a major consideration for the MRDB. To verify this hypothesis, we conducted a comparative performance analysis on each database structure.

A. OPERATING ENVIRONMENT

This section will discuss the environment used for the performance evaluation, including descriptions of computer hardware, software, the databases used, the queries used, and the indexing strategy.

1. Hardware and Software

The hardware used to store the MRDB and run the queries is an IBM PS/2 with a 386 processor. All database files are stored on a hard disk. All routines are written in INFORMIX-4GL with embedded SQL commands to access the database.

2. Databases Used

The comparative analysis was conducted on two databases: The original MRDB and the normalized MRDB structure.

a. Original MRDB Structure

An unclassified version of the MRDB was provided by ARINC for the purpose of this study. It consists of 65 network and 415 connectivity records. Additional records were created to bring the total number of records in the network and connectivity tables to 195 and 1245 records respectively. This was done in order to accurately evaluate a database that is the same size as the classified version. This larger database is used to measure query response time. We use the original unnormalized schema without any changes.

b. Normalized MRDB Structure

A physical table is created for each relation in the relational model (Appendix A) of the normalized database. The data definition statements for the new MRDB appear in Appendix B. Data is transferred from the original MRDB to the normalized database. However, some table names were identical to table names in the original database. In order to avoid confusion, an "N" is added at the beginning of tables in the new MRDB to make them unique (e.g., NETWORK in the original MRDB is called NNETWORK in the new normalized MRDB).

3. Queries Used

We will investigate the performance of some hypothetical queries on both the original and the normalized versions of the MRDB. Five queries have been selected to represent a wide range of cardinality in the result tables and to reflect the effects of normalization on retrieval time. All queries are select-project-join except for query #4 which includes an aggregate function. The number of tables involved in the join and the number of records retrieved for each query are summarized in Figure 11.

a. Query #1

This query selects all networks and their associated connectivities and terminals that have had requirements validated by the JCS. This is the largest of the five queries. The SQL for this query on the original database is shown in Figure 12. Query #1 performs an outer join between NETWORK and CONNECTIVITY for all networks that have the value "JCS VALIDATED" in the field *urdb_status*. This join was selected because it is performed often in generating reports for the MRDB.

	ORIGINAL MRDB		**NORMALIZED MRDB**	
	Tables in join	**Records retrieved**	**Tables in join**	**Records retrieved**
Query #1	2	746	4	1492
Query #2	2	9	4	18
Query #3	2	65	4	130
Query #4	2	63	2	63
Query #5	no join	537	4	1145

Figure 11 Joins and records retrieved for the five queries

```
SELECT *
   FROM network, OUTER connectivity
   WHERE (network.urdb_status = "JCS VALIDATED") AND
        (network.net_id = connectivity.net_id)
```

Figure 12 Query #1 on the original database

The SQL for this query on the normalized database is shown in Figure 13. In contrast with the query on the original database, this query performs a join on four tables; NNETWORK, NCONNECTIVITY, TERMINAL_CONNECT, and TERMINAL. The conditions in the WHERE clause links the four tables together.

b. Query #2

This query selects all associated connectivities and terminals in the Automated Weather Network (AWN). This is the smallest query and selects only one network (AWN) and nine connectivities. The SQL for the this query on the original database is shown in Figure 14. Similar to query #1, this query also requires a join between NETWORK and CONNECTIVITY, but in the WHERE clause, only one network is selected.

```
SELECT *
   FROM nnetwork, OUTER (nconnectivity, terminal_connect, terminal)
   WHERE (nnetwork.urdb_status = "JCS VALIDATED") AND
      (nconnectivity.net_id = nnetwork.net_id) AND
      (terminal_connect.connect_id = nconnectivity.connect_id) AND
      (terminal.terminal_id = terminal_connect.terminal_id) AND
      (terminal.connect_id = terminal_connect.connect_id)
```

Figure 13 Query #1 on the normalized database

```
SELECT *
  FROM network, OUTER connectivity
  WHERE (network.net_id = "49") AND
      (network.net_id = connectivity.net_id)
```

Figure 14 Query #2 on the original database

The SQL for query #2 on the normalized database is shown in Figure 15. This query is similar to the first query except that only one network is retrieved.

c. *Query #3*

This query selects all networks and their associated connectivities and terminals that have had requirements not reviewed by the JCS. This condition selects nine networks and 65 total connectivities. The SQL for this query on the original database is in shown Figure 16. It performs a join on the same tables as the first two queries, with the only difference being that this query specifies a value of "UNREVIEWED" for *urdb_status*.

```
SELECT *
  FROM nnetwork, OUTER (nconnectivity, terminal_connect, terminal)
  WHERE (nnetwork.net_id = "49") AND
     (nconnectivity.net_id = nnetwork.net_id) AND
     (terminal_connect.connect_id = nconnectivity.connect_id) AND
     (terminal.terminal_id = terminal_connect.terminal_id) AND
     (terminal.connect_id = terminal_connect.connect_id)
```

Figure 15 Query #2 on the normalized database

```
SELECT *
  FROM network, OUTER connectivity
  WHERE (network.urdb_status = "UNREVIEWED") AND
      (network.net_id = connectivity.net_id)
```

Figure 16 Query #3 on the original database

The SQL for query #2 on the normalized database is shown in Figure 17. It also performs a join on the four tables, NNETWORK, NCONNECTIVITY, TERMINAL_CONNECT, and TERMINAL. It will select networks that have a value of "UNREVIEWED" for *urdb_status*. The other conditions in the WHERE clause link tables that were specified in the selection process.

d. Query #4

This query finds the total objective unstressed throughput rate for all networks whose requirements are CINC advocated. The term "unstressed" refers to a terminal that will be transmitting or receiving a signal in an environment that does not experience jamming, interception, or any phenomena that would impair the signal. The SQL for query #4 on the original database is shown in Figure 18. There is a join

```
SELECT *
  FROM nnetwork, OUTER (nconnectivity, terminal_connect, terminal)
  WHERE (nnetwork.urdb_status = "UNREVIEWED") AND
    (nconnectivity.net_id = nnetwork.net_id) AND
    (terminal_connect.connect_id = nconnectivity.connect_id) AND
    (terminal.terminal_id = terminal_connect.terminal_id) AND
    (terminal.connect_id = terminal_connect.connect_id)
```

Figure 17 Query #3 on the normalized database

```
SELECT network.net_id, SUM(unstress_thru_o)
  FROM network, connectivity
  WHERE (urdb_status = "CINC ADVOCATED") AND
      (network.net_id = connectivity.net_id)
  GROUP BY network.net_id
```

Figure 18 Query #4 on the original database

condition involved because the data on throughput rates are in CONNECTIVITY. The query gives the network name and the total throughput rate for an unstressed environment. This query differs from the other queries in that the total is determined by taking the aggregate sum of the unstressed throughput rates for all connectivities in the network. There will be one total for each network in the database. There are 63 networks that are CINC advocated.

The SQL for query #4 on the normalized database is shown in Figure 19. This query is similar to the query on the original database in that only two tables are joined and the same number of records will be retrieved.

```
SELECT nnetwork.net_id, SUM(unstress_thru_o)
  FROM nnetwork, nconnectivity
  WHERE (nnetwork.urdb_status = "CINC ADVOCATED") AND
      (nnetwork.net_id = nconnectivity.net_id)
  GROUP BY nnetwork.net_id
```

Figure 19 Query #4 on the normalized database

e. Query #5

This query selects all connectivities and their associated terminals that have full duplex connections. This is the only query that does not perform a join. All information needed for this query is in the CONNECTIVITY table. The SQL for this query is shown in Figure 20. It is another large query and retrieves 537 records from the CONNECTIVITY table.

The SQL for this query on the normalized database is shown in Figure 21. These two queries are unique in that the query on the original database uses only one table (and no join) whereas the query on the normalized database has a join of four tables. This type of query should demonstrate clearly the tradeoff between normalization and query retrieval time.

4. Indexing Strategy

Indexes are created on key fields to optimize the performance of queries. Significant time savings result from building the appropriate indexes. A disadvantage of having an index is that it can slow down insertions and updates to the database. However, this is only a problem when inserting a large number of records from one table to another. It isn't a serious problem when inserting one record at a time. The

```
SELECT acronym, org_user, org_terminal, dst_user, dst_terminal
   FROM connectivity
   WHERE modeop = "F"
```

Figure 20 Query #5 on the original database

```
SELECT *
  FROM nnetwork, OUTER(nconnectivity, terminal_connect, terminal)
  WHERE (op_mode = "F") AND
     (nnetwork.net_id = nconnectivity.net_id) AND
     (nconnectivity.connect_id = terminal_connect.connect_id) AND
     (terminal_connect.connect_id = terminal.connect_id) AND
     (terminal_connect.terminal_id = terminal.terminal_id)
```

Figure 21 Query #5 for normalized database

INFORMIX-4GL Reference Manual suggests the following rules-of-thumb when developing an indexing strategy:

- Do not create indexes for small tables with fewer than 200 records.
- Do not create indexes on a field that has only a few possible values (e.g., yes/no responses, data types and connectivity types in the CONNECTIV Y table).
- If the WHERE clause of a SELECT statement imposes a condition on a single field, put an index on that field.
- If there are conditions placed on several fields, make a composite index on all the affected fields.
- If there is a join condition between a single field in one table and a single field in another table, create an index on the field in the table with the larger number of records

Normally, no relationship exists between the physical order of the data in disk storage and the order in an index. The physical order of the data in disk storage can be the same as the order in an index through "clustering." This is done in INFORMIX-4GL either by using a CLUSTER clause in the CREATE INDEX statement or by altering an existing index.

The database with the old structure used the indexing specified by its designers. The indexing strategy for the new database structure is as follows:

- create an index on *net_id* in NNETWORK
- create a clustered index on *net_id* and an index on *connect_id* in NCONNECTIVITY. This will group together, in storage, all connectivities within the same network.
- create a clustered index on *connect_id* and an index on *terminal_id* in TERMINAL_CONNECT.
- create a clustered index on *connect_id* and an index on *terminal_id* in TERMINAL.

5. Implementation Issues

The data to populate the normalized MRDB is the same data that populates the original MRDB. The data is transferred using an INFORMIX-4GL program as shown in Appendix C. In order to transfer the data while maintaining relationships and information content, *connect_id*, the primary key in NCONNECTIVITY, is added to TERMINAL. This is done in order to uniquely identify a terminal at the end of each connectivity, since the fields *org_trm_id* and *dst_trm_id* in CONNECTIVITY do not uniquely identify a terminal, but rather a terminal location. It is very difficult to identify an individual terminal in the original MRDB.

Two other changes are needed in order to create the physical database. First, the composite primary key *user_name, user_mobility*, and *user_platform* from NETUSER in the original MRDB is used as the composite primary key in TUSER[5]. Second, it is

[5] TUSER is used because INFORMIX-4GL treats 'USER' as a reserved word. The CREATE TABLE statement in DDL for USER does not work. Therefore, TUSER is used as a substitute.

decided to use the composite primary key *location* and *st_cntry_cd* for NGEO_LOC instead of *geo_reference_code*. The reason for this selection is that it is easier to use the existing primary key than to convert each occurrence of these composite primary keys to new single-field primary keys.

In addition to the problems described above, the data translation process is a lengthy and difficult one for the following reasons:

- Data has to be extracted from tables in the original MRDB that are not normalized.
- Relationships among the different tables has to be preserved from one database to another.
- Values in fields of different records that are supposed to be identical tend to differ. For example, within *doc_title* of REFERENCE, one record uses "and" and another record uses "&" for the title. The document is the same, but because of the spelling difference, they translate as two separate documents into DOCUMENT. This is known as the "synonym" problem.

B. RESPONSE TIME MEASUREMENTS

This section describes and reports the results of measurements taken on the two different database structures. The choice of a performance metric depends upon how a database is used. A transaction processing database will use throughput (rate at which transactions are completed) as a performance metric. However, the MRDB is not used to process transactions from users, but rather is used as a decision support tool for ad hoc queries to support decisions by management. The performance metric for a decision support database is typically a measure of response time, also called query elapsed time [Ref. 9:p. 323]. Additionally, since the normalized database returns more records than the original database, we will examine the average response time per record retrieved.

For the purpose of this thesis, query elapsed time is the time it takes to retrieve all records satisfying the criteria specified in the query. Response times are measured with and without indexing. Figure 22 shows a summary of the response times for the various scenarios. Average record response time is calculated by dividing the response time by the total number of records retrieved. Figure 23 shows a summary of the average response time per record retrieved and the percentage of increase or decrease in response time from the original MRDB to the normalized MRDB.

1. Results of Query #1

Using an index, the response time for the normalized structure is about nine minutes greater than that of the original structure. This means an increase of 264 percent. However, the query on the new database structure retrieves twice as many records. This fact increases the response time per record about 30 percent. The reason twice as many records are retrieved is that the CONNECTIVITY table in the original

	ORIGINAL MRDB		NORMALIZED MRDB	
	w/index	w/o index	w/index	w/o index
Query #1	5:25 min	8:05 min	14:18 min	62:03 min
Query #2	6 sec	2:53 min	15 sec	2:45 min
Query #3	29 sec	3:36 min	1:17 min	7:07 min
Query #4	51 sec	3:55 min	45 sec	2:20 min
Query #5	37 sec	37 sec	9:01 min	55:11 min

Figure 22 Summary of total response times for the five queries

	ORIGINAL MRDB	NORMALIZED MRDB	% INCREASE
Query #1	.436	.575	30
Query #2	.666	.833	27
Query #3	.446	.592	33
Query #4	.810	.714	12 (decrease)
Query #5	.069	.472	580

Figure 23 Average response time (in seconds) per record retrieved

database contains attributes for two terminals, one at each end of the connection. In the process of normalizing, each terminal has a separate record in the TERMINAL table; hence, twice the number of tuples are retrieved.

2. Results of Query #2

The response time for query #2 on the original MRDB is 2.5 times faster than the normalized MRDB. Again, response time on both databases are penalized severely without indexing. Like query #1, the response time per record shows a much lesser percentage increase (27%) than the total response time increase (250%).

3. Results of Query #3

The difference in response times between the two schemas for this query is similar to queries #1 and #2. The query on the original database was 2.65 times faster than the query on the normalized database. Response times were also severely degraded without indexing. Like queries #1 and #2, query #3 experienced only a slight increase in response time per record retrieved (33%).

4. Results of Query #4

The response times on each database is the same (the query on the normalized database was slightly faster, but not enough to make any conclusions). Although these queries retrieve the same number of records as query #3 on the original database, elapsed time for these queries are almost double (on the indexed databases).

5. Results of Query #5

Total response time for query #5 on the normalized database is over 14 times slower than the query on the original database. Average response time is better but is still over five times slower. Without indexing, the response times on the normalized database was severely penalized. Because the condition in the WHERE clause is not an indexed field, the response times for queries on the indexed and non-indexed database were identical.

C. DISCUSSION OF RESULTS

The response times measured support the assertion that queries on the indexed original database were faster than the queries on the normalized database. This is particularly true because more tables are joined for the queries on the normalized database. When the query involved the same number of tables in a join (query #3), then response times were essentially the same on both databases.

As expected, total response times are smaller for queries that retrieve a smaller number of records. As the number of records to be retrieved increases, the total response time increases also. If we examine the average response time per record on the normalized database, we see that there is only a 30% increase. This increase is independent of the cardinality of the resultant table (see results of queries #1, #2, and #3 in Figure 23) and can be attributed to the extra time it takes to join the additional tables. This can be readily seen in the results for query #4 in which response times were equal an the cardinality of the result tables are equal.

Without indexing, response times will be degraded significantly whether the tables are normalized or not. An exception is made for query #5 on the original database. These time are identical because the condition in the WHERE clause is on a field that isn't indexed on either execution of the query. It should be restated that these response times are for data retrieval only. Additional tasks such as creating a detailed report on each network will increase overall processing time.

Overall, the response times for the indexed normalized database were not excessive. If significant processing were to be done in conjunction with these queries (in one of the current application programs, for example), then the differences in response times would be insignificant. For example, the case when reports are to be generated for the networks selected in query #1. The additional time for the application program to prepare and print the reports would make the seven minute difference in retrieval time insignificant. However, differences would be very significant if the MRDB were to be used in an interactive environment. This would require response times to be less than one second, or near real-time. In this case, a denormalized structure would be preferred over a normalized structure.

V. CONCLUSIONS AND RECOMMENDATIONS

The purpose of this chapter is to present conclusions based on research done in this thesis and to make recommendations for improvements and further study. In this thesis, we examined the current structure of the MRDB, re-engineered a new conceptual model of the real world system using the ER diagram and then translated it into a relational model. The relations in this model were implemented into the INFORMIX DBMS and a comparative performance analysis was performed.

A. CONCLUSIONS

The first half of this thesis examined the current MRDB and developed a new conceptual schema using a structured database design process. Based on this work, the following conclusions can be made:

- The current MRDB is not a normalized structure and can experience some update and insert anomalies.
- A structured approach to designing the MRDB will result in a normalized structure that is simple and easy to implement.
- It is imperative that a database be developed using a structured design process. Data modeling using a high-level model should be a major activity consisting of gathering user requirements and representing them in a high level model.
- Once that is accomplished, a logical database structure is derived from the high level data model.
- The relations in the logical database model can easily be implemented using an appropriate DBMS.

- Depending upon query and update patterns, the set of normalized table in the DBMS can be denormalized (tables combined) to obtain better performance.

Based upon the results of the comparative analysis of database response times, the following conclusions can be made:

- As the number of tables involved in a join increases, per record query response times will increase. For example in queries one, two, and three the number of tables in the join increased from two for the original database to four for the normalized database. The query times increase by an average of 30% for each query.

- The response times for the original database and the normalized database were the same when the number of tables involved were the same. For example, in query #4, the query response times were the same for both databases.

- The increased response times for the normalized database does not seem to have a significant impact on overall database performance (generating reports, ad hoc queries, etc.) based upon the current turn-around time for database output.

B. RECOMMENDATIONS

The work performed in this thesis is a starting point for an overall database design process. The following additional research and study on the development process is needed:

- It is important that a comprehensive database design process be performed, especially phase one, Requirements Collection and Analysis. Without a well-developed set of requirements, the database designed will not meet the needs of the users. These requirements would be the foundation of a responsive and flexible relational database that will support decision makers.

- A new MRDB should be hosted on a commercially available, PC-based, relational database. The selection of a relational DBMS should consider economic and technical factors. It should be affordable and it should have a good graphical user interface, a sophisticated database engine, and an interactive SQL facility to handle quick ad hoc queries.

- An analysis should be done on the query and update pattern in order to determine if denormalizing is necessary.

An advantage to using a COTS database product is transportability. For example, the MRDB and its database management system could be installed on a notebook PC with a 486 processor. This computer could travel with decision makers to provide quick access to applicable information and on-the-spot answers to questions as they arise.

If the query response times are not acceptable, then combining the physical tables that resulted form the design process may be the answer. When application programs, such as the MRDB, are developed around a particular data structure, denormalization or further normalizing becomes impractical. Even small changes to tables can have a serious impact on application programs. This is why it is important to accurately specify requirements at the onset.

A common method to overcome the effects of normalization is to create logical views of tables to produce denormalized virtual tables [Ref. 10: p. 58]. In the MRDB, the commonly used fields for reports could be combined into a logical view and the separate physical tables maintained for use by other applications. If this structure causes performance problems, the view can be materialized into a physical table.

This method would minimize the impact of denormalization. The trade-off of having a materialized view with faster access time would be additional storage space. Also, a strategy for maintaining the duplicated data would need to be developed. Programs to update the existing normalized tables would need to be revised to update the new materialized views.

Another factor to consider is the computer that hosts the MRDB. A 486 (or Pentium) will also reduce the query response times and the overall impact of denormalization.

The results of the comparative analysis show that although a highly normalized database will penalize query response time, normalization would not significantly impact the current turn-around time for MRDB products. It is more important that the MRDB contain data that's accurate and free from data anomalies. A normalized database structure will ensure that future modifications to the MRDB and associated applications are smooth and easy. This will give decision makers high confidence in the information provided by the MRDB.

APPENDIX A

RELATIONAL MODEL FOR THE MRDB

The chart in Figure 24 shows the relationships between the tables within the MRDB. For clarity and simplicity, only the primary and foreign keys are shown.

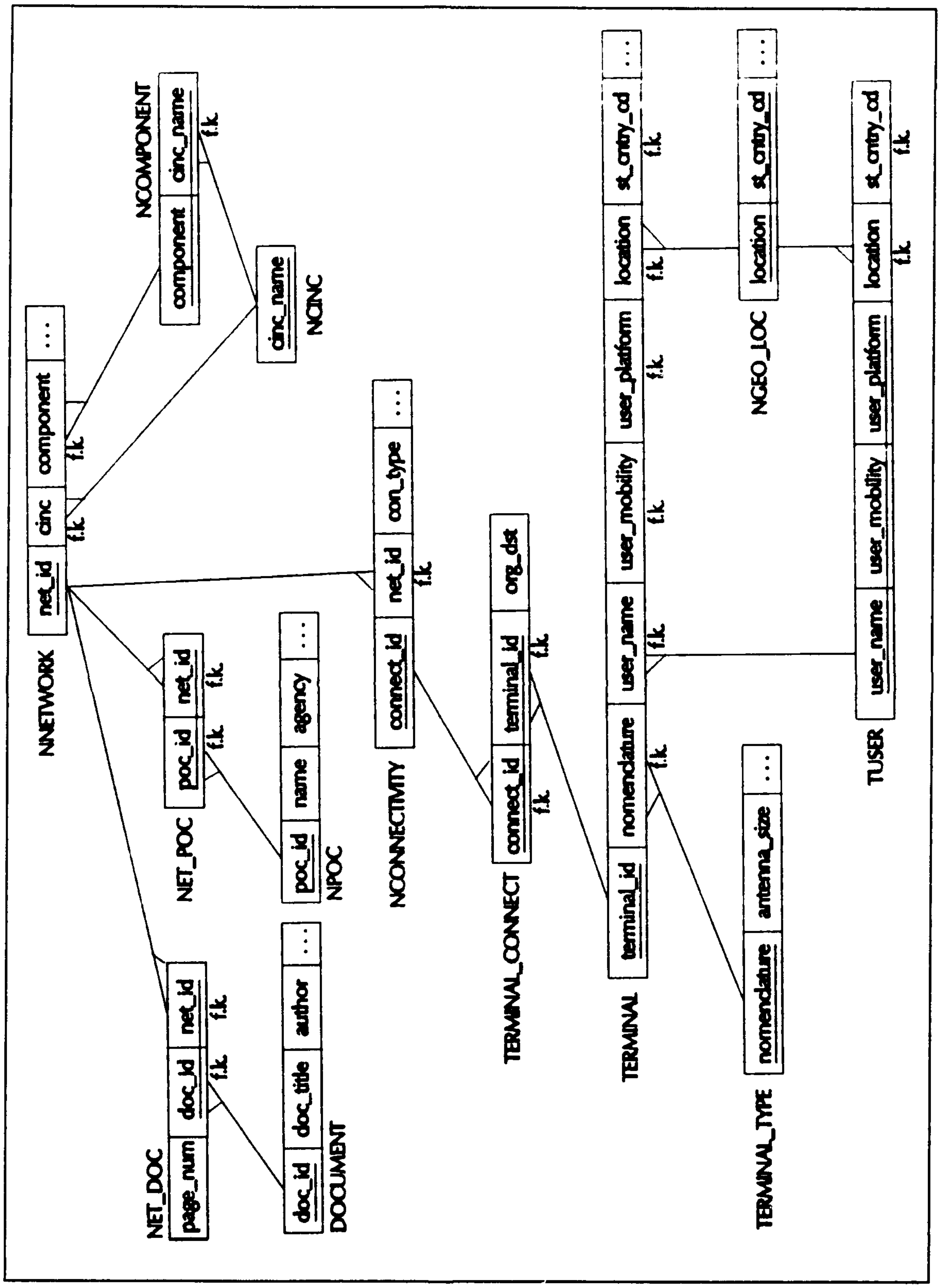

Figure 24 MRDB Relational Model

APPENDIX B

TABLES AND ATTRIBUTES FOR THE MRDB

This appendix lists the statements to define the tables in the new MRDB. Also included is a description of each attribute. Information describing these fields was obtained from the data dictionary for the original MRDB and the ISDB.

```
create table NGEO_LOC
  (
   geo_ref_cd     char(4),        {unique id for each geographic location}
   location       char(8),        {location of terminal or facility; together with st_cntry_cd forma composite
                                  primary key}
   st_cntry_cd    char(2),        {DCAC 310-65-1 defined code for state or country}
   area_cd        char(1),        {DCAC 310-65-1 code for operational area}
   us_rgn_cd      char(1),        {US region code}
   cntry_abrv     char(5),        {abbreviation of country}
   full_name      char(25),       {full name of location}
   lat_deg        smallint,       {latitude of terminal or user}
   lat_min        smallint,
   lat_sec        smallint,
   lat_hem        char(1),
   lon_deg        smallint,       {longitude of terminal or user}
   lon_min        smallint,
   lon_sec        smallint,
   lon_hem        char(1)
  );

create table NPOC
  (
   poc_id         serial not null, {primary key for NPOC}
   agency         char(20),        {organization to which poc is assigned}
   off_code       char(12),        {office code for mailing address}
   address1       char(20),        {first line of address}
   address2       char(20),        {second line of address}
   city           char(20),
   sc             char(2),         {state country code (see ngeo_loc)}
   zip            char(10),        {zip code}
   com_num        char(16),        {commercial phone number}
   dsn_num        char(16),        {DSN phone number}
   stu_num        char(16),        {STU-III number}
   sfax_num       char(16),        {secure fax number}
   ufax_num       char(16)         {unsecure fax number}
  );
```

```
create table NNETWORK
  (
   net_id          serial not null, {primary key for each network}
   cinc_name       char(12),        {foreign key; name of CINC responsible for generating requirements}
   component       char(12),        {foreign key; name of service component or CINC component}
   acronym         char(10),        {acronym of network}
   class           char(4),         {security classification of network requirement}
   net_name        char(35),        {long name of network}
   con_ops         char(100),       {concept of operations}
   cincprty        char(3),         {cinc-assigned priorty for requirement}
   globprty        char(2),         {priority assigned by ISDB}
   rqtype_o        char(11),        {support type: full period or contingency (objective)}
   rqtype_t        char(11),        {support type (threshhold)}
   rqtype_crit     char(1),         {is this type of support critical to network?}
   duration        smallint,        {length in days of requirement (for short term req.)}
   rqbegin         date,            {first day of validation}
   rqend           date,            {last day of validation}
   urdb_status     char(15),        {status of requirement in validation process (from ISDB)}
   satisfy         char(1),         {level of req. satisfaction (partial, full, etc.)}
   primary         char(1),         {primary system which req. is satisfied (freq. band)}
   secondary       char(1),         {backup to satisfy req.}
   milstar_loaded  char(1),         {in milstar loading analysis?}
   bermant         decimal(2,1),    {mantisa of bit error rate}
   berexp_o        smallint,        {exponent of bit error rate (obj.)}
   berexp_t        smallint,        {            "          (thresh.)}
   ber_crit        char(1),         {is BER critical to network service quality?}
   minavail_o      decimal(5,2),    {min percent availability of net to user}
   minavail_t      decimal(5,2),    {            "              }
   emp_o           char(1),         {does net reqire emp protection?}
   emp_t           char(1),         {            "          }
   emp_crit        char(1),         {is emp critical to operation of net?}
   nsp             char(1),         {will net xmit through nuc. scint. environment?}
   aj              char(1),         {does net reqire aj protection?}
   aj_lvl_o        smallint,        {expected power level of jammer (0 to 4)}
   aj_lvl_t        smallint,        {            "              }
   aj_dist_o       smallint,        {distance from user and jammer}
   aj_dist_t       smallint,        {            "          }
   aj_crit         char(1),         {is aj critical to requirement?}
   lpi             char(1),         {is lpi required for net?}
   lpi_dist_o      smallint,        {est. distance from user for signal intercept}
   lpi_dist_t      smallint,        {              "              }
   lpi_crit        char(1),         {is lpi critical for net success?}
   conflict_1      char(1),         {will net operate during conflict level 1? (peace)}
   conflict_2      char(1),         {    -low-intensity conflict}
   conflict_3      char(1),         {    -crisis}
   conflict_4      char(1),         {    -theater conventional war}
   conflict_5      char(1),         {    -global conventional war}
   conflict_6      char(1),         {    -theater nuclear war}
   conflict_7      char(1),         {    -strategic nuclear war}
   conflict_8      char(1),         {    -post-attack/reconstruction}
   maxout_o        smallint,        {max time for interruption due to weather, nuc. etc}
   maxout_t        smallint,        {                 "                  }
```

```
    dcycle_o      decimal(4,1),  {duty cycle; % net to support transmission}
    dcycle_t      decimal(4,1),  {                "                      }
    dcycle_crit   char(1),       {is this percent critical for net operation?}
    callfreq      smallint,      {frequency of calls per hour by user}
    calllen       decimal(3,1),  {average length of calls by user}
    maxmtt_o      decimal(5,1),  {max mean xmit time call rec after transmitted}
    maxmtt_t      decimal(5,1),  {                    "                       }
    maxmtt_crit   char(1),       {is max mean xmit time critical to operation of net?}
    maxaat_o      smallint,      {max allowable access time to access net after request}
    maxaat_t      smallint,      {                     "                       }
    maxaat_crit   char(1),       {is max allowable access time critical to net ops?}
    as_crit       char(1),       {is antispoof critical to net?}
    net_excl      char(1),       {has network been deleted from the MRDB?}
    net_crit      char(1),       {is net critical for completion of unit's mission?}
    misn_essen    smallint,      {is net essential for completion of unit's mission?}
    us_cntrl      char(1),       {is satellite system/terminals under US control?}
    avail_crit    char(1),       {is availability of net critical?}
    s1            char(1),       {eight various scenario for net to be operational}
    s2            char(1),
    s3            char(1),
    s4            char(1),
    s5            char(1),
    s6            char(1),
    s7            char(1),
    s8            char(1),
    last_update   date           {last date requirements were updated}
   );

create index n_net_id on nnetwork (net_id);

create table NET_POC             {relationship between network and poc}
   (
    net_id        smallint not null, {foreign key}
    poc_id        smallint not nul   {foreign key}
   );

create table DOCUMENT
   (
    doc_id        serial not null, {primary key for document records}
    doc_title     char(70),        {title of document}
    author        char(15),        {author of document}
    pub_date      date,            {pulication date of document}
    class         char(5)          {classification of document}
   );

create table NET_DOC             {relationship between NNETWORK and DOCUMENT}
   (
    doc_id        smallint not null, {foreign key of DOCUMENT}
    net_id        smallint not null, {foreign key of NNETWORK}
    page_num      char(15)           {page number in document which references net}
   );
```

```
create table NCINC
  (
   cinc_name      char(12) not null {name of cinc or government agency}
  );

create table NCOMPONENT
  (
   component      char(12) not null, {name of component of a cinc, service or agency}
   cinc_name      char(12) not null  {parent agency of component}
  );

create table NCONNECTIVITY
  (
   connect_id     serial not null,  {primary key; unique identifier of a connectivity}
   net_id         smallint not null, {foreign key of NNETWORK}
   conn_type      char(1),          {type of configuration (point-to-point, hub, etc.)}
   data_type      char(5),          {type of data (DATA, VOICE, IMAGE, VIDEO, etc.)}
   secure         char(1),          {is secure transmission required?}
   op_mode        char(1),          {mode of operation (duplex, half, etc.)}
   ccsd_num       char(8),          {id for operational circuits in DSCS or DCS; ISDB}
   ccsd_restore   char(2),          {importance of circuit for restoration}
   isdb_num       char(13),         {control number for a requirement in the ISDB}
   system_name    char(10),         {of satellite system carrying circuit}
   num_circ       smallint,         {number of circuits in the connectivity}
   critical       char(1),          {is connectivity critical to mission of network}
   stress_thru_o  decimal(16,3), {throughput rate in a stressed environment (obj.)}
   stress_thru_t  decimal(16,3), {             "                (thresh)}
   unstress_thr_o decimal(16,3), {throughput rate in an unstressed environment}
   unstress_thr_t decimal(16,3), {             "                  }
   stress_drate_o decimal(16,3),       {data rate in a stressed environment}
   stress_drate_t decimal(16,3), {             "          }
   unstress_drt_o decimal(16,3), {data rate in an unstressed environment}
   unstress_drt_t decimal(16,3)  {             "             }
  );

create index c_connect_id on nconnectivity (connect_id);
create cluster index c_net_id on nconnectivity (net_id);

create table TERMINAL_TYPE
  (
   nomenclature   char(15) not null, {primary key; type of terminal}
   frequency      char(3),          {operating frequency band}
   mobility       char(5),          {terminal mobility (fix, mobile, trans)}
   platform       char(5),          {terminal platform (ship, air, manpack, fix, etc)}
   antenna_size   smallint,         {diameter of antenna in feet}
   lo_eirp        smallint,         {low end of range for terminal/antenna combo}
   hi_eirp        smallint,         {high end of range for terminal/antenna combo}
   lo_gt          smallint,         {lowest gain to temperature ratio}
   hi_gt          smallint,         {highest gain to temperature ratio}
   min_drate      decimal(16,2), {min data rate in kbps}
   max_drate      decimal(16,2)  {max data rate in kbps}
  );
```

```
create table TUSER
  (
   user_id        serial not null, {primary key}
   user_name      char(15),        {together with mobility and platform form a composite primary key}
   user_mobility  char(5),
   user_platform  char(5),
   location       char(8),         {part of foreign key of ngeo_loc}
   st_cntry_cd    char(2)          {part of foreign key of ngeo_loc}
  );

create table TERMINAL_CONNECT {relationship between NCONNECTIVITY and TERMINAL}
  (
   connect_id     smallint,        {foreign key of NCONNECTIVITY}
   terminal_id    smallint,        {foreign key of TERMINAL}
   org_dst        char(3)          {is terminal the originating or destination terminal}
  );

create index tc_connect_id on terminal_connect (connect_id);
create index tc_terminal_id on terminal_connect (terminal_id);

create table TERMINAL
  (
   connect_id     smallint,        {together with terminal_id form composite key to uniquely identify a
                                   terminal in a connectivity.}
   terminal_id    smallint,        {part of composite primary key}
   nomenclature   char(15),        {foreign key of TERMINAL_TYPE}
   user_name      char(15),        {part of foreign key of TUSER}
   user_mobility  char(5),         {part of foreign key of TUSER}
   user_platform  char(5),         {part of foreign key of TUSER}
   location       char(8),         {part of foreign key of NGEO_LOC}
   st_cntry_cd    char(2),         {part of foreign key of NGEO_LCO}
   serial_number  char(10),        {serial number of terminal}
   trm_region     char(2),         {region code}
   trm_rd         char(3),         {DISA-assigned reporting designator for DSCS terminals}
   act_date       date,            {activation date; actual or projected for this terminal}
   deact_date     date             {deactivation date}
  );

create index t_connect_id on terminal (connect_id);
create index n_terminal_id on terminal (terminal_id);
```

APPENDIX C

4GL TO TRANSFER DATA

This appendix is a summary of the INFORMIX-4GL source code that transfers the data from the original database to the new normalized database. The actual data was transferred one table at a time. However, all the code was combined into one listing to make it easier to follow.

```
DATABASE new

MAIN

DEFINE tname CHAR(15), { For building the TUSER table }
     tmob CHAR(5),
     tplat CHAR(5),
     tplace CHAR(8),
     tstate CHAR(2)

DEFINE iconn_id  SMALLINT, { For building the TERMINAL_CONNECT table
     iterm_id  SMALLINT

{---------------------------------------
       Build the POC table
---------------------------------------}

INSERT INTO npoc (poc_id, agency, off_code, address1, address2,
             city, sc, zip, com_num, dsn_num, stu_num,
             sfax_num, ufax_num)
  SELECT poc_id, agency, off_code, address1, address2, city,
             sc, zip, com_num, dsn_num, stu_num,
             sfax_num, ufax_num
    FROM poc

{---------------------------------------
      Build the NETWORK table
---------------------------------------}

INSERT INTO nnetwork (net_id,
    cinc_name,
    component,
    acronym,
    class,
net_name,
con_ops,
cincprty,
globprty,
rqtype_o,
rqtype_t,
rqtype_crit,
duration,
rqbegin,
rqend,
urdb_status,
satisfy,
primary,
secondary,
milstar_loaded,
bermant,
berexp_o,
berexp_t,
ber_crit,
minavail_o,
minavail_t,
emp_o,
emp_t,
emp_crit,
nsp,
aj,
aj_lvl_o,
aj_lvl_t,
aj_dist_o,
aj_dist_t,
aj_crit,
lpi,
lpi_dist_o,
lpi_dist_t,
lpi_crit,
conflict_1,
conflict_2,
conflict_3,
conflict_4,
conflict_5,
conflict_6,
conflict_7,
conflict_8,
maxout_o,
maxout_t,
dcycle_o,
dcycle_t,
dcycle_crit,
callfreq,
calllen,
maxmtt_o,
maxmtt_t,
    maxmtt_crit,
    maxaat_o,
    maxaat_t,
    maxaat_crit,
    as_crit,
    net_excl,
    net_crit,
    misn_essen,
    us_cntrl,
    avail_crit,
    s1,
    s2,
    s3,
    s4,
    s5,
    s6,
    s7,
    s8)
SELECT net_id,
    cinc,
    component,
    acronym,
    class,
    netname,
    misndefn,
    cincprty,
    globprty,
    rqtype_o,
    rqtype_t,
    rqtype_crit,
    duration,
    rqbegin,
    rqend,
    urdb_status,
    satisfy,
    primary,
    secondary,
    milstar_loaded,
    bermant,
    berexp_o,
    berexp_t,
    ber_crit,
    minavail_o,
    minavail_t,
    emp_o,
    emp_t,
    emp_crit,
    nsp,
    aj,
    aj_lvl_o,
    aj_lvl_t,
    aj_dist_o,
    aj_dist_t,
    aj_crit,
    lpi,
    lpi_dist_o,
    lpi_dist_t,
    lpi_crit,
    conflict_1,
    conflict_2,
    conflict_3,
    conflict_4,
    conflict_5,
    conflict_6,
    conflict_7,
    conflict_8,
    maxout_o,
    maxout_t,
    dcycle_o,
    dcycle_t,
    dcycle_crit,
    callfreq,
    calllen,
    maxmtt_o,
    maxmtt_t,
    maxmtt_crit,
    maxaat_o,
    maxaat_t,
    maxaat_crit,
    as_crit,
    net_excl,
    net_crit,
    misn_essen,
    us_cntrl,
    avail_crit,
    s1,
    s2,
    s3,
    s4,
    s5,
    s6,
    s7,
    s8
  FROM network

{------------------------------
   Build the NET_POC Table
-------------------------------}

INSERT INTO net_poc (net_id, poc_id)
   SELECT net_id, poc_id
      FROM network
      WHERE poc_id IS NOT NULL
```

```
{------------------------------
   Build the DOCUMENT Table
-------------------------------}

INSERT INTO document (doc_id, doc_title, author, pub_date, class)
  SELECT idnum, document, author, published, refclass
    FROM reference

{------------------------------
   Build the NET_DOC Table
-------------------------------}

INSERT INTO net_doc (doc_id, net_id, page_num)
  SELECT idnum, net_id, pagenum
    FROM reference

{------------------------------
   Build the CINC Table
-------------------------------}

INSERT INTO ncinc (cinc_name)
  SELECT cinc FROM cincs

{------------------------------
  Build the COMPONENT Table
-------------------------------}

INSERT INTO ncomponent (component, cinc_name)
  SELECT DISTINCT component, cinc
    FROM network
    WHERE component IS NOT NULL

{------------------------------
  Build the CONNECTIVITY Table
-------------------------------}

INSERT INTO nconnectivity (connect_id,
    net_id,
    conn_type,
    data_type,
    secure,
    op_mode,
    num_circ,
    critical,
    stress_thru_o,
    stress_thru_t,
    unstress_thru_o,
    unstress_thru_t,
    stress_drate_o,
    stress_drate_t,
    unstress_drate_o,
    unstress_drate_t)
  SELECT connid,
    net_id,
    contype,
    dtype,
    secure,
    modeop,
    num_circ,
    critical,
    stress_thru_o,
    stress_thru_t,
    unstress_thru_o,
    unstress_thru_t,
    stress_drate_o,
    stress_drate_t,
    unstress_drate_o,
    unstress_drate_t
  FROM connectivity

{----------------------------------
   Create the TERMINAL_TYPE Table
-----------------------------------}

CREATE TABLE intermediate
  (
  nomenclature  CHAR(15),
  mobility  CHAR(5),
  platform  CHAR(5)
  )
INSERT INTO intermediate (nomenclature, mobility, platform)
  SELECT DISTINCT org_terminal, org_mobility, org_platform
    FROM connectivity
    WHERE org_terminal IS NOT NULL

INSERT INTO intermediate (nomenclature, mobility, platform)
  SELECT DISTINCT dst_terminal, dst_mobility, dst_platform
    FROM connectivity
    WHERE dst_terminal IS NOT NULL

INSERT INTO terminal_type (nomenclature, mobility, platform)
  SELECT DISTINCT nomenclature, mobility, platform
    FROM intermediate

DROP TABLE intermediate

{-------------------------------
   Build the TUSER Table
-------------------------------}

DECLARE tuser_ptr SCROLL CURSOR FOR
  SELECT username, mobility, platform, place, state
    FROM netuser
```

```
FOREACH tuser_ptr INTO tname, tmob, tplat, tplace, tstate
  INSERT INTO tuser
    VALUES (0, tname, tmob, tplat, tplace, tstate)
END FOREACH

{--------------------------------
 Build the TERMINAL_CONNECT Table
---------------------------------}

CREATE TABLE intermediate
   (connect_id    SMALLINT NOT NULL,
    terminal_id   SMALLINT NOT NULL,
    org_dst       CHAR(3))

DECLARE con_ptr1 SCROLL CURSOR FOR
    SELECT connid, org_trm_id
        FROM connectivity
        WHERE org_trm_id IS NOT NULL

FOREACH con_ptr1 INTO iconn_id, iterm_id
    INSERT INTO intermediate (connect_id, terminal_id, org_dst)
        VALUES (iconn_id, iterm_id, "ORG")

END FOREACH
DECLARE con_ptr2 SCROLL CURSOR FOR
    SELECT connid, dst_trm_id
        FROM connectivity
        WHERE dst_trm_id IS NOT NULL

FOREACH con_ptr2 INTO iconn_id, iterm_id
    INSERT INTO intermediate (connect_id, terminal_id, org_dst)
        VALUES (iconn_id, iterm_id, "DST")
END FOREACH

INSERT INTO terminal_connect (connect_id, terminal_id, org_dst)
    SELECT DISTINCT connect_id, terminal_id, org_dst
        FROM intermediate

DROP TABLE intermediate

{------------------------------
   Build the TERMINAL Table
-------------------------------}

INSERT INTO terminal (connect_id, terminal_id, nomenclature, user_name,
        user_mobility, user_platform, location, st_cntry_cd,
        serial_number, trm_region, trm_rd, act_date, deact_date)
   SELECT connid, org_trm_id, org_terminal, org_user, org_mobility,
        org_platform, trm_loc, trm_sc, trm_sn, trm_region,
        trm_rd, act_date, deact_date
     FROM connectivity, term_loc
     WHERE org_trm_id = trm_id
```

```
INSERT INTO terminal (connect_id, terminal_id, nomenclature, user_name,
        user_mobility, user_platform, location, st_cntry_cd,
        serial_number, trm_region, trm_rd, act_date, deact_date)
  SELECT connid, dst_trm_id, dst_terminal, dst_user, dst_mobility,
        dst_platform, trm_loc, trm_sc, trm_sn, trm_region,
        trm_rd, act_date, deact_date
    FROM connectivity, term_loc
    WHERE dst_trm_id = trm_id

END MAIN
```

LIST OF REFERENCES

1. Major, William, *Design and Implementation of a Prototype PC-based Graphical and Interactive MILSATCOM Requirements Database System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1993.

2. Henry, Hugh A., *Military Satellite Communications Decision Support System Requirements Analysis and User Interface*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1993.

3. Batini, C., Ceri, S., Navathe, S.B., *Conceptual Database Design: An Entity-Relationship Approach*, The Benjamin Cummings Publishing Company, Inc., 1992.

4. Date, C.J., *An Introduction to Database Systems*, vol 1, 5th Edition, Addison Wesley, 1990.

5. Teorey, Toby J., *Database Modeling and Design: The Entity-Relationship Approach*, Morgan Kaufmann Publishers, Inc., 1990.

6. Elmasri, R., Navathe, S.B., *Fundamentals of Database Systems*, The Benjamin Cummings Publishing Company, Inc., 1989.

7. Kent, William, "A Simple guide to Five Normal Forms in Relational Database Theory," *Communications of the ACM*, v. 26, pp. 120-125, February 1983.

8. Lacy-Thompson, Tony, *INFORMIX-SQL: A tutorial and reference*, Prentice Hall, 1991.

9. Dietrich, S.W., et al., "A Practitioner's Introduction to Database Performance Benchmarks and Measurments," *The Computer Journal*, v. 35, pp. 322-331, August 1992.

10. Edwards, Joe B., "High Performance Without Compromise," *Datamation*, July 1, 1990.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	U.S. Space Command Code J-6 Ops Peterson AFB, CO 80914	3
4.	Director for Command, Control and Communications Systems, Joint Staff Washington, DC 20318-6000	1
5.	C3 Academic Group, Code CC Naval Postgraduate School Monterey, CA 93943	1
6.	AFIT/CIRK Wright-Patterson AFB, OH 45433-6583	1
7.	Professor Tung Bui, AS/Bd Naval Postgraduate School Monterey, CA 93943	3
8.	Professor Daniel R. Dolk, AS/Dk Naval Postgraduate School Monterey, CA 93943	1
9.	Professor Magdi N. Kamel AS/Ka Naval Postgraduate School Monterey, CA 93943	2

10. Lt Col Anderson 1
Joint Staff
Satellite Communications Division (J6S)
Pentagon
Washington, DC 20318-6000

11. Capt Ronald Kearns 2
2620 Charlottsville Drive
Colorado Springs, CO 80915