UMECRL-TR-4-86

THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY

A LOGICAL DESIGN METHODOLOGY FOR RELATIONAL DATABASES USING THE EXTENDED ER MODEL

T. TEOREY, D. YANG, AND J. FRY

CRL-TR-4-86

1079 East Engineering Bldg.
Ann Arbor, MI 48109

THE UNIVERSITY OF MICHIGAN
COMPUTING RESEARCH LABORATORY[*]

A LOGICAL DESIGN METHODOLOGY FOR RELATIONAL DATABASES USING THE EXTENDED ER MODEL

T. TEOREY, D. YANG, AND J. FRY

CRL-TR-4-86

JANUARY 1986

Room 1079, East Engineering Building
Ann Arbor, Michigan 48109
USA
Tel: (313) 763-8000

[*]Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agency.

Table of Contents

A Logical Design Methodology for Relational Databases Using the Extended ER Model

Toby J. Teorey
Computing Research Laboratory
Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-1109
(313) 763-5216

Dongqing Yang
Computer Science and Technology
Peking University
Beijing, The People's Republic of China

James P. Fry
Computer and Information Systems
Graduate School of Business Administration
The University of Michigan
Ann Arbor, MI 48109

January 1986

Abstract

A practical database design methodology is defined for the design of large relational databases. First, the data requirements are conceptualized using an extended entity-relationship model, with the extensions being additional semantics such as ternary relationships, optional relationships, and the generalization abstraction. The extended entity-relationship model is then decomposed according to a set of basic entity-relationship constructs, and these are transformed into candidate relations. A set of basic transformations have been developed for the three types of relations: entity relations, extended entity relations, and relationship relations. Candidate relations are further analyzed and modified to attain the highest degree of normalization desired. Using the processing information additional refinements are made to the relations only if resulting efficiency can be justified. Tradeoffs between the degree of normalization and processing efficiency are analyzed before the final relation definitions are specified.

By capturing both the natural and usage relationships among data elements, this methodology produces designs that are not only accurate representations of reality, but are efficient and flexible to accommodate future processing requirements. The methodology also reduces the number of data dependencies that must be analyzed, using the extended ER model conceptualization, and maintains data integrity through normalization. This approach can be implemented manually or in a simple software package so long as a "good" solution is acceptable and absolute optimality is not required.

Categories and Subject Descriptors: H.2.1 [Database Management]: Logical Design - data models

General Terms: Databases, Design, Performance

Additional Key Words and Phrases: Relational databases, logical design, entity-relationship model, integrity, efficiency

INTRODUCTION

Relational database design has traditionally been a low level, bottom-up activity dealing with data elements. It has been defined as the process of analyzing inter data element dependencies obtained in the requirements analysis, and synthesizing these data elements into normalized relations based upon these known dependencies [Codd 1970,1974; Martin 1982; Date 1984, Smith 1985]. While the traditional process is vital to the design of relational databases, its complexity, particularly in large databases, can be overwhelming to the point where practical designers often do not bother to master it or even to use it (if understood) with any regularity. The complexity of these procedures can be dramatically reduced if the intermediate step of conceptual design is introduced and the well known tools of entity-relationship modeling are employed.

The entity-relationship (ER) model has been most successful as a tool for communication between the designer and the end-user during the requirements analysis and conceptual design phases because of its ease of understanding and its power in representation [Chen 1976]. One of the reasons for its effectiveness is that it is a top-down approach using a high level of abstraction. The number of entities (i.e. those objects that we want to collect information about) in a database is typically an order of magnitude less than the number of data elements. Therefore using entities as an abstraction for data elements and focusing on the inter entity relationships greatly reduces the number of objects under consideration and simplifies the analysis. While it is still necessary to represent data elements by attributes of entities at the conceptual level, their dependencies are normally confined to the other attributes within the entity or in some cases to those attributes associated with other entities that have a direct relationship to their entity.

The major inter attribute dependency is between the entity keys of different entities which is captured in the ER modeling process. Special cases such as dependencies among data elements of unrelated entities can be analyzed upon identification in the data analysis.

This relational database design approach uses both the ER model and the relational model in successive stages. It benefits from the simplicity and ease of use of the entity relationship model and the structure (and associated formalism) of the relational model. In order to accomplish this approach it is necessary to build a framework for transforming the variety of ER constructs into relations that can be easily normalized. Before we do this, however, we first define the major steps of the relational design methodology.

The relational design methodology is both a refinement of and an extension to the design methodology proposed in [Teorey and Fry 1982]. The basic steps of this methodology, as shown in Fig. 1, are summarized as follows:

<u>Step 1. Extented ER Modeling of Requirements</u>

The data requirements are analyzed and modeled using an extended ER diagram which includes semantics for optional relationships, ternary relationships and data generalization. Processing requirements are specified using natural language expressions along with the frequency of occurrence. Logical views from multiple sources are integrated into a common global view of the entire database.

Step 2. Transformation of the Extended ER Model to Relations

Based on a categorization of extended ER constructs and a set of mapping rules, each relationship and its associated entities is transformed into a set of candidate relations. Redundant relations are eliminated.

Step 3. Normalization of Relations

Functional dependencies (FDs) are derived from the extended ER diagram to represent the dependencies among data elements which are keys of entities. Additional FDs and multivalued dependencies (MVDs), that represent the dependencies among key and nonkey attributes within entities, are derived from the requirements specification. Candidate relations associated with all derived FDs and MVDs are then normalized to the highest degree desired using standard manual normalization techniques. Redundancies that occur in normalized candidate relations are then analyzed further for possible elimination, with the constraint that data integrity must be preserved.

Step 4. Refinement of Relations for Usage Efficiency

Additional refinements can be made to the normalized relations to increase the overall database processing (usage) efficiency. The basic technique is to add redundant attributes to existing relations to reduce the number of join operations required during the execution of an application. Because the added attributes could lower the degree of normalization of a relation, additional analysis is required to determine whether such denormalization is acceptable, given the potential increase in processing efficiency. If it is acceptable, then both the refined and unrefined relations are used as candidate relations for physical design.

This methodology simplifies the approach to designing large relational databases by reducing the number of data dependencies that need to be analyzed. This is accomplished by introducing a conceptual design step in the traditional relational modeling which uses an extended ER model to capture an accurate representation of reality. Data integrity is preserved through normalization of the candidate relations formed from the transformation of the extended ER model. Processing efficiency is increased through refinement of these relations for usage. Design extendability is achieved by maintaining the natural data relationships as much as possible during usage refinement.

Next we discuss in detail the steps of the relational database design methodology.

1.0 ER MODELING AND EXTENDED CONSTRUCTS

The Entity-Relationship approach initially proposed by Chen, although modified and extended by others, still remains the premier model for conceptual design. It is used to represent information in terms of entities, their attributes and associations among entity occurrences called relationships.

Numerous extensions incorporating greater semantics in the original ER model have been proposed by others [Smith and Smith 1977;Scheuermann et al. 1980;Atzeni et al. 1981;Navathe and Cheng 1983;Howe 1983;Lenzerini and Santucci 1983;Kent 1984]. Of particular note are extensions for generalization and subset hierarchies, relationship relations, existence dependencies, and conditional and unconditional membership classes. Although these extensions are attaining growing acceptance, their transformations to the relational model have not yet been well-defined or are just beginning to be defined for many real-world problems [Jajodia and Ng 1983,1984; Kent 1984].

1.1 Original Classes of Objects (ER Model)

Initially, Chen proposed three classes of objects: entities, attributes, and relationships (Fig. 2a). Entities (actually entity sets) were the principal objects about which information was to be collected usually denoting a person, place, thing, or event of informational interest. (We drop the set terminology for simplicity). Attributes were used to detail the entities by giving them descriptive properties such as name, color, and weight. Finally, relationships (actually relationship sets) represented real-world associations among one or more entities.

There are two types of attributes: identifiers and descriptors. The former is used to uniquely distinguish among the occurrences of a entity and the latter is purely descriptive of an entity occurrence. Entities can be distinguished by the "strength" of their identifying attributes. Strong entities have internal identifiers that uniquely determine the existence of entity occurrences. Weak entities derive their existence from the identifying attributes (sometimes called external attributes) of one or more "parent" entities. Relationships have semantic meaning which is indicated by the connectivity between entity occurrences (one-to-one, one-to-many, and many-to-many) and the participation in this connectivity by the member entities (either conditional or unconditional). For example, the entity "person" may or may not have a spouse. Finally, each of the entities may have one or more synonyms associated with it. The diagrams for representing entities, relationships, and attributes are shown in Fig. 2a.

1.2 <u>Extended Classes of Objects (EER Model)</u>

The introduction of abstraction into the ER model resulted in two additional types of objects: subset hierarchies and generalization hierarchies. The first type is a subset hierarchy, which is diagrammed in Fig. 2b.

<u>Subset Hierarchy Definition</u>:

An entity E1 is a subset of another entity E2 if every occurrence of E1 is also an occurrence of E2.

A subset hierarchy is the case when every occurrence of the generic entity may also be an occurrence of the other entities which are potentially overlapping subsets. For example, the entity EMPLOYEE may include 'employees attending college', 'employees which hold political office', or 'employees who are also shareholders' as specialized classifications.

The second type of object is the generalization hierarchy which is diagrammed in Fig. 2b.

Generalization Hierarchy Definition:

An entity E is generalization of the entities E1, E2, ...,En if each occurrence of E is also an occurrence of one and only one of the entities E1,E2,...,En.

A generalization hierarchy occurs when an entity (which we call the generic entity) is partitioned by different values of a common atttribute. For example, the entity EMPLOYEE is a generalization of ENGINEER, SECRETARY, AND TECHNICIAN. The generalization object (EMPLOYEE) is called an "IS-A" exclusive hierarchy because each occurrence of the entity EMPLOYEE is an occurrence of one and only one of the entities ENGINEER, SECRETARY, TECHNICIAN.

1.3 Fundamental EER Constructs

The following classification of EER constructs is defined to facilitate development of a concise and easy to understand EER diagram.

(1) Degree of a relationship.

The degree of a relationship is the number of entities associated in the relationship. An n-ary relationship is of degree n. Unary, binary, and ternary relationships are special cases where the degree is 1, 2, and 3 respectively. This is indicated in Fig. 3.

(2) Connectivity of a relationship.

The connectivity of a relationship specifies the mapping of the associated entity occurrences in the relationship. Values for connectivity are either "one" or "many". The actual number associated with the term "many" is called the cardinality of the connectivity. Cardinality may be represented by upper and lower bounds. Fig. 3 shows the basic constructs for connectivity: one-to-one (unary or binary relationship), one-to-many (unary or binary relationship), many-to-many (unary or binary relationship), and one-to-many-to-many ternary relationship. The shaded area in the unary or binary relationship diamond represents the "many" side, while the unshaded area represents the "one" side [Reiner et al. 1985]. We will use an n-sided figure to represent n-ary relationships for n>2 in order to explicitly show each entity associated in the relationship to be either "one" or "many" related to the other entities.

(3) Membership class in a relationship.

Membership class (or optionality) specifies whether the "one" side in a one-to-one or one-to-many relationship is unconditional or conditional. If an occurrence of the "one" side entity must always exist to maintain the relationship, then it is unconditional. When an occurrence of that entity need not exist, it is considered conditional. The "many" side of a relationship is always considered to be conditional unless explicitly defined otherwise. The conditional membership class, defined by a "0" on the connectivity line between an entity and a relationship, is shown in Fig. 3.

(4) Existence dependency of an entity in a relationship.

A strong entity is shown with a single-bordered rectangle, while a weak entity is depicted with a double-bordered rectangle (Fig. 4).

(5) Object class of entities and relationships.

The basic objects are the n-ary relationships with their associated entities. Objects resulting from abstraction are the generalization hierarchy and the subset hierarchy (Fig. 4). The generalization hierarchy implies that the subsets are a full partition, such that the subsets are disjoint and their combination makes up the full set. The subset hierarcy implies that the subsets are potentially overlapping.

2.0 EER MODELING OF REQUIREMENTS (STEP 1)

The objective of requirements analysis is many fold. It delineates the data requirements of the enterprise; it describes the information about the objects and their associations needed to model these data requirements; and it determines the types of transactions that are intended to be executed on the database. We use the extended entity-relationship (EER) model to describe these objects and their interrelationships, and use natural language expressions to describe transactions. A more detailed discussion of requirements analysis can be found in [Martin 1982; Teorey and Fry 1982; Yao 1985].

2.1 Design Step 1 Details

Step 1.1 Identify entities and attributes.

While it is easy to define entity, attribute, and relationship constructs (cf Sec. 1.1), it is not as easy to distinguish their role in modeling the database. What makes an object an entity, an attribute, or even a relationship? For example, stores are located in cities. Should CITY be an entity or an attribute? Registration records are kept for each student. Is REGISTRATION-RECORD an entity or a relationship? What is a "normalized" entity?

The following guidelines for identifying entities and attributes will help the designer converge to a normalized relational database design.

(1) Entities have descriptive information, identifying at-

tributes do not. If there is descriptive information about an object, the object should be identified as an entity. If only an identifier is needed for an object, the object should be identified as an attribute. For example, in the above store and city example, if there is some descriptive information such as STATE and POPULATION for cities, then CITY should be identified as an entity. If only CITY-NAME is needed to identify a city, then CITY should be identified as an attribute.

(2) Multivalued attributes should be specified as entities. If more than one value of a descriptor corresponds to one value of identifier, this descriptor should be identified as an entity instead of an attribute, even though it does not have descriptors for itself. For example, in the above store and city example, if one store (a chain) could locate in several cities, then CITY should be identified as an entity even it only needs an identifier CITY-NAME.

(3) Make an attribute which has a many-to-one relationship with another entity an entity. If a descriptor in one entity has a many-to-one relationship with another entity, the descriptor should be identified as an entity, even if it does not have its own descriptors. For example, if two entities have been identified: STORE (with identifier STORE-NUMBER, descriptors OWNER and CITY) and STATE. Because there is a many-to-one relationship between CITY and STATE, CITY should be identified as an

entity.

(4) <u>Attach attributes to entities which they describe most directly</u>. For example, attribute OFFICE-BUILDING should be an attribute of the entity DEPARTMENT instead of in entity EMPLOYEE.

(5) <u>Avoid composite identifiers as much as possible</u>. If an entity has been defined with a composite identifier, i.e. an identifier composed of two or more attributes, and the components of the identifier are all identifiers of other entities, then eliminate this entity. The corresponding object could be defined as a relationship in a subsequent step. If an entity has been defined with a composite identifier, but components of the identifier are not identifiers of other entities, then there are two possible solutions. One is to eliminate this entity and define new entities with components of the composite identifier as entity identifiers, and in a subsequent step define a relationship to represent this object. Another solution is to keep the entity with the composite identifier if it is reasonably natural.

As an example, if an entity REGISTRATION-RECORD has been defined, with STUDENT and COURSE as a composite identifier, then the entity REGISTRATION-RECORD could be eliminated, and two new entities STUDENT and COURSE could be defined; later in a subsequent step, a relationship between STUDENT and COURSE could be defined to represent the object REGISTRATION-RECORD. In another example, if an entity VOLLEYBALL-TEAM has been defined, with COUNTRY and GENDER as a composite identifier, then it seems suitable to keep this entity, because defining an entity GENDER is not very natural.

The procedure of identifying entities and attaching attributes to entities is iterative: identifying some objects as entities, attaching identifiers and descriptors to them, then finding some violation to the above guidelines, changing some objects from entity to attribute, or from attribute to entity, then attaching attributes to the new entities, etc.

<u>Step</u> <u>1.2</u> Identify any generalization hierarchies and subset hierarchies.

If there is a generalization or subset hierarchy among entities, then reattach attributes to the relevant entities. Put identifier and generic descriptors in the generic entity, and put identifier and specific descriptors in the subset entities.

For example, suppose the following entities were identified in the EER model: EMPLOYEE (with identifier EMP-NO and descriptors EMP-NAME, HOME-ADDRESS, DATE-OF-BIRTH, JOB-TITLE, SALARY, SKILL), ENGINEER (with identifier EMP-NO and descriptors EMP-NAME, HOME-ADDRESS, SPECIALTY), SECRETARY (with identifier EMP-NO and descriptors EMP-NAME, DATE-OF-BIRTH, SALARY, SPEED-OF-TYPING), TECHNICIAN (with identifier EMP-NO and descriptors EMP-NAME, SKILL, YEARS-OF-EXPERIENCE). We identify that EMPLOYEE is generalization of

ENGINEER,SECRETARY and TECHNICIAN. Then we reattach attributes to the entities. We put identifier EMP-NO and generic descriptors EMP-NAME, HOME-ADDRESS, DATE-OF-BIRTH, JOB-TITLE, and SALARY in the generic entity EMPLOYEE; put identifier EMP-NO and specific descriptor SPECIALITY in entity ENGINEER; put identifier EMP-NO and specific descriptor SPEED-OF-TYPING in entity SECRETARY; put identifier EMP-NO and specific descriptors SKILL, YEARS-OF-EXPERIENCE in entity TECHNICIAN.

Step 1.3 Define relationships.

We now deal with objects which were not identified as entities or attributes, but represent associations among objects. We define them as relationships. For every relationship the following should be specified: degree, connectivity, membership class, existence dependency, and attributes.

The following are some guidelines for defining relationships.

(1) Redundant relationships should be eliminated. Two or more relationships that are used to represent the same concept are considered to be redundant. Redundant relationships are more likely to result in unnormalized relations when transforming the EER model into relational schemas. Note that two or more relationships are allowed between the same two entities as long as the two relationships have different meanings. They are not

considered redundant.

One important case of redundancy is transitive dependency (see Fig. 5). If REL1 is a many-to-one relationship between ENTITY1 and ENTITY2, REL2 is a many-to-one relationship between ENTITY2 and ENTITY3, REL3 is a many-to-one relationship between ENTITY1 and ENTITY3, and both REL1 and REL2 are unconditional, then REL3 is redundant and should be eliminated.

(2) <u>Ternary relationships must be defined carefully</u>. We define a ternary relationship among three entities only when the concept (association) cannot be represented by several binary relationships among those entities. For example, there is an association among entities TEACHER, STUDENT, and PROJECT. The meaning of the association is that the student does a project under the instruction of teacher(s). If each student can only be involved in one project, but can work under the instruction of several teachers, and one teacher can instruct many students, then two binary relationships could be defined instead of one ternary relationship (Fig. 6a). If, however, each student can be involved in several projects and work under the instruction of several teachers, but if for every project the student works under the instruction of exactly one teacher, and if a teacher can instruct several students in doing their projects, then one ternary relationship could be defined (Fig. 6b).

The meaning of connectivity for ternary relationships is important. Fig. 7 shows that for a given pair of occurrences of ENTITY2 and ENTITY3, there is only one corresponding occurrence of ENTITY1; however, for a given pair of occurrences of ENTITY1 and ENTITY2, there could be many corresponding occurrences of ENTITY3.

Step 1.4 Integrate multiple views of entities, attributes, and relationships.

Typically when more that one person is involved in requirements analysis, multiple views occur. These views must eventually be consolidated into a single global view to eliminate redundancy and inconsistency from the model. View integration requires further use of the extended ER semantic tools of identity (identifying synonyms), aggregation, and generalization. A more detailed discussion of view integration tools can be found in [Teorey and Fry 1982; Yao 1985].

2.2 An Example Database: Company Personnel and Projects

We define a simple database design problem to illustrate the major steps in this relational database design methodology. Let us suppose it is desirable to build a company-wide database for a large engineering firm that keeps track of all personnel, their skills and projects assigned, departments worked in, and personal computers allocated. Each employee is given a job title (engineer, technician, secretary, manager). Engineers and technicians work on an average of two projects at one time, and each project could be headquartered at a different location (city). We assume that analysis of the detailed requirements for data relationships in this company results in the global view EER diagram in Fig. 8, which becomes the focal point for developing the normalized relations. Each relationship in Fig. 8 is based upon a verifiable assertion about the actual data in the company, such as those illustrated in Figs. 9-13. Analysis of those assertions leads to the transformation of EER constructs into candidate relations. Attributes are not included in Figures 8-13 for simplicity, but are defined later in this example.

As an example of view integration, the generalization of EMPLOYEE over JOB-TITLE could represent the consolidation of two views of the database, one based on EMPLOYEE as the basic unit of personnel, and another based on the classification of employees by job titles and special relationships with those classifications such as the allocation personal computers (PCs) to engineers.

3.0 TRANSFORMATION OF THE EER MODEL TO RELATIONS (STEP 2)

3.1 <u>Transformation Rules</u>

Let us now look at each EER construct in more detail to see how each transformation rule is defined and applied. Our example is drawn from the company personnel and project database EER schema illustrated in Fig. 8. All types of EER constructs we must transform to relations are shown at least once in the figure.

We note that the basic transformations result in three types of relations [McGee 1974; Sakai 1983; Martin 1983; Hawryszkiewycz 1984]:

(1) entity relation with the same information content as the original entity.

This transformation always occurs for entities with binary relationships that are many-to-many, one-to-many on the one (parent) side, or one-to-one where both entities are either conditional or unconditional; entities with unary relationships that are many-to-many or one-to-one; and entities with any ternary or higher degree relationship, generalization hierarchy, or subset hierarchy.

(2) entity relation with the embedded foreign key of the parent entity.

This transformation always occurs with binary relationships that are one-to-many for the entity on the many (child) side, and for one-to-one relationships for the entity on the conditional side; and for the entities with unary relationships that are one-to-many.

(3) relationship relation with the foreign keys of all

the entities that are thus related.

This transformation always occurs for relationships that are binary and many-to-many, or one-to-one when both entities are either conditional or unconditional; relationships that are unary and either many-to-many or one-to-one; and all relationships that are ternary or higher degree.

The general rules for null values allowed in these transformations are simple. Nulls are only allowed for foreign keys of any conditional entity in an entity relation, but are not allowed for foreign keys of any unconditional entity in an entity relation. Note that the entity in a unary relationship is considered to be unconditional if either side of the relationship is unconditional. Nulls are also not allowed for any foreign key in a relationship relation.

3.1.1 Two Entities, One (Binary) Relationship

The one-to-one relationship between entities is illustrated in Fig. 9a,b and c. When both entities are unconditionally related (Fig. 9a), each entity becomes a relation and the key of either entity can appear in the other entity's relation as a foreign key. One of the entities in a conditional relationship (see DEPARTMENT in Fig. 9b) should contain the foreign key of the other entity in its transformed relation. The other entity (EMPLOYEE) could also contain a foreign key (of DEPARTMENT), with nulls allowed, but would require more storage space because of the much greater number of EMPLOYEE entity occurrences than DEPARTMENT entity occurrences. When both entities are conditionally related (Fig. 9c) a relationship relation is created containing primary keys of both entities. Nulls are not allowed because both keys must be known for a tuple to have any meaning. An alternative to the relationship relation is to embed foreign keys in each of the entity relations; however, this is less meaningful than an explicit relationship relation and is not recommended.

The one-to-many relationship is always allowed to be conditional on the "many" side, and it may be either unconditional (Fig. 9d) or conditional (Fig. 9e) on the "one" side. In both cases the foreign key must appear on the "many" side, which represents the child entity, with nulls allowed for foreign keys, only in the conditional case.

The many-to-many relationship is totally conditional and requires a relationship relation with primary keys of both entities (Fig. 9f). Embedded foreign keys are not possible because of the "many" property in both directions.

3.1.2 <u>One Entity, One (Unary) Relationship</u>

One entity with a one-to-one relationship implies some form of entity occurrence pairing, as specified by the relationship name, and this must be either completely conditional or completely unconditional. Both the unconditional case (Fig. 10a) and the conditional case (Fig. 10b) are best implemented with a relationship relation that explicitly shows the meaning of the relationship. In both cases the two key attributes are taken from the same domain but are given different names to designate their unique use. The one-to-many relationship requires a foreign key in the entity relation for both the conditional case (Fig. 10c), with nulls allowed, and the unconditional case (Fig. 10d), without nulls allowed. The many-to-many relationship is always conditional (Fig. 10e) and uses a relationship relation.

3.1.3 n Entities, One (n-ary) Relationship (n > 2)

The allowable varieties of an n-ary relationship are the n+1 possible allocations of entities with "many" connectivity (and cardinality from 0 to n). Thus, the 3-ary (ternary) relationship in Fig. 11 has four possible varieties. All varieties are transformed by creating a relationship relation containing the primary keys of all n entities; however, in each case the meaning of the keys is different. When all relationships are "one" (Fig. 11a), the relationship relation consists of n possible distinct candidate keys, each consisting of n-1 entity keys. This represents the fact that there are n functional dependencies (FDs) needed to describe this relationship. The conditional "one" allows null foreign keys, but the unconditional "one" does not.

When all relationships are "many" (Fig. 11b), the relationship relation is all key, and no FDs are present. In general the number of entities with connectivity "one" determines the number of FDs, and each determinant of an FD (or the case with all key) determines the candidate key of the relationship relation. Multivalued dependencies (MVDs) are not easily detectable from the EER model, and thus must be determined from further requirements analysis related to the composite keys in the model.

3.1.4 Generalization and Subset Hierarchies

The generalization hierarchy resulting in disjoint subsets is produced by partitioning the generic entity by different values of a common attribute, e.g. JOB-TITLE in Fig. 12. The transformation of disjoint subset generalization produces a separate relation for the whole set (the generic entity) and each of the subsets. The generic entity relation contains the generic entity key and all common attributes (including the attribute used for partitioning). Each subset relation contains the generic entity key and only attributes specific to that subset. Update integrity is maintained by requiring all insertions and deletions to occur in both the set (generic entity) relation and relevant subset relation. If the change is to the key, then all subsets as well as the set relation must be updated. Changes to an attribute affects either the set or one subset relation.

Overlapping subsets are produced by partitioning the generic entity by values of different attributes (Fig. 13). The transformation of this construct produces separate relations for the generic entity and each of the subset entities. The key of each relation is the key of the generic entity; and while the generic entity relation contains only common attributes, the subset relations contain attributes specific to that subset entity. Thus, the transformation rules for the disjoint and overlapping subsets are the same.

The integrity rules between these two cases are different, however. With overlapping subsets, deletion from the set (generic entity) relation cascades to anywhere from none to all of the subsets. Also, before insertion to a subset relation, it is necessary to check whether a tuple with the same key value exists in the set relation. A change to a nonkey attribute affects the set or one of the subsets. A change to a key affects the set and at least one subset.

3.1.5 Multiple Relationships

Multiple relationships among n entities are always considered to be completely independent. Therefore, each relationship produces a completely new set of entity relations and relationship relations. Relationship relations will be unique, but entity relations that are either equivalent or differ only in the addition of a foreign key can be consolidated into a single entity relation containing all foreign keys.

3.1.6 Existence Dependent (Weak) Entities

Weak entities differ from (strong) entities only in the need for keys from other entities to establish their unique identities. Otherwise they have the same transformation properties as strong entities, and no special rules are needed. When a weak entity is already derived from two or more strong entities in the ER diagram, it can be directly transformed into a relationship relation without further change.

3.1.7 Aggregation

The aggregation abstraction can occur among entities, attributes of entities, or relate attributes to a single entity [Smith and Smith 1977]. Aggregation among entities, defined by the PART-OF relationship, is a special case of the collection of one-to-many binary relationships and can be transformed as defined in Section 3.1.1. As an example, BICYCLE can represent the whole entity; while SEAT, PEDALS, HANDLEBARS, etc. represent its parts; with each part being an entity with its own distinct attributes.

3.2 Design Step 2 Details

The following steps summarize the basic transformation rules given in Section 3.1.

Step 2.1 Transform every entity into one relation with the key and nonkey attributes of the entity. If there is a many-to-one relationship between an entity and another (or same type) entity, add the key of the other (parent) entity into the relation. If there is a one-to-one relationship between an entity and another (or same type) entity with unconditional membership class on the other side and conditional membership class on this side, then add the key of the other entity into the relation.

Every entity in a generalization hierarchy or subset hierarchy is transformed into a relation. Each of these relations contains the key of the generic entity. The generic entity relation also contains nonkey values that are common to all the entities so related, and the other relations contain nonkey values specific to each nongeneric entity.

Step 2.2 Transform every many-to-many binary (or unary) relationship, and every one-to-one binary (or unary) relationship with either conditional or unconditional membership class on both sides, into a relationship relation with the keys of the two entities and the attributes of the relationship.

Step 2.3 Transform every ternary (or higher n-ary) relationship into a relationship relation using the rules given in Fig. 11.

Entity normalization is not necessarily preserved under these transformations. The introduction of a foreign key into an entity relation may result in additional functional dependencies in an otherwise normalized relation [Wilmot 1984]. For example, in Fig. 14 the introduction of the foreign key DEPT-NO into the entity relation EMPLOYEE creates a transitive functional dependency EMP-NO -> DEPT-NO -> OFFICE-BLDG. Relationship relations, consisting of two or more entity keys and possibly some intersection non-key attributes, may also experience similar problems. Therefore the use of preliminary normalization of entities will not guarantee normalized relations after the EER transformations [Fong et al. 1985]. However, after the transformations such normalization can be accomplished using well-known methods [Bernstein 1976; Fagin 1977; Ullman 1980;Lien 1981;Zaniolo and Melkanoff 1981; Martin 1983; Maier 1983; Yao 1985].

3.3 Example

The transformation of EER diagrams to candidate relations is applied to our example database of company personnel and projects, as shown in Figs. 8-13. A summary of the transformation of all entities and their relationships to candidate relations (Steps 2.1 - 2.3) is illustrated in Table 1. Primary keys are underlined. We include some of the most typical nonkey attributes we assume have been obtained from the requirements analysis.

Step 2.1: Entities to relations

1. DIVISION(DIV-NO,...,HEAD-EMP-NO)
2. DEPARTMENT(DEPT-NO,DEPT-NAME,OFFICE-BLDG,..., DIV-NO,MANAG-EMP-NO)
3. EMPLOYEE(EMP-NO,EMP-NAME,JOB-TITLE,OFFICE-BLDG,..., DEPT-NO)
4. SKILL(SKILL-NO,....)
5. PROJECT(PROJ-NAME,...,LOC-NAME)
6. LOCATION(LOC-NAME,....)
7. EMP.MANAGER(EMP-NO,....)
8. EMP.ENGINEER(EMP-NO,....)
9. EMP.TECHNICIAN(EMP-NO,....)
10. EMP.SECRETARY(EMP-NO,....)
11. PC(PC-NO,....)

Step 2.2: Binary or unary relationships to relations

12. MARRIED-TO(EMP-NO,SPOUSE-EMP-NO)
13. HAS-ALLOCATED(EMP-NO,PC-NO)

Step 2.3: Ternary (or any n-ary) relationships to relations

14. AVAIL-SKILL(EMP-NO,SKILL-NO,PROJ-NAME)
15. ASSIGNED-TO(EMP-NO,LOC-NAME,PROJ-NAME)

Table 1. Transformation of entities and relationships to relations (example).

4.0 NORMALIZATION OF RELATIONS (STEP 3)

Normalization of candidate relations is accomplished by analyzing the FDs and MVDs associated with those relations. Fortunately these data dependencies are easily derivable from the same EER constructs we used to generate the candidate relations in Step 2. Further analysis may lead to elimination of data redundancies in the normalized candidate relations.

4.1 Design Step 3 Details

Step 3.1 Derive the primary FDs from the EER diagram.

Primary FDs represent the dependencies among data elements that are keys of entities, that is, the inter-entity dependencies. Secondary FDs, on the other hand, represent dependencies among data elements that comprise a single entity, that is, the intra-entity dependencies (see Step 3.2). Table 2 shows the type of primary FDs derivable from each type of EER construct defined in Section 1.3 and consistent with the derivable candidate relations in Figures 9-13. In fact, each primary FD is associated with exactly one candidate relation that represents a relationship among entities in the EER diagram.

Based on the transformations in Table 2 we summarize the basic types of primary FDs derivable from EER relationship constructs:

(1) key (many side) ---> key (one side)
(2) key (one side A) ---> key (one side B)
(3) key (many side A), key (many side B) ---> key (one

side)
(4) key (one side A), key (many side) ---> key (one side B)
(5) key (one side A), key (one side B) ---> key (one side C)
(6) composite-key ---> 0

Types (1) and (2) represent an embedded foreign key functionally determined by the primary key in a unary or binary relationship; types (3) through (5) apply only to ternary relationships; and type (6) applies to all degrees of relationships in which the relation is represented as all key. FDs for higher degree n-ary relationships can be obtained by extending (3) through (6).

Degree	Connectivity	Primary FD
Binary	1-to-1	key(one A) ---> key(one B)
		key(one B) ---> key(one A)
	1-to-1(opt)	key(one A) ---> key(one B)
		key(one B) ---> key(one A)
	1(opt)-to-1(opt)	key(one A) ---> key(one B)
		key(one B) ---> key(one A)
	1-to-many	key(many) ---> key(one)
	1(opt)-to-many	key(many) ---> key(one)
	many-to-many	composite-key ---> 0
Unary	1-to-1	key(one A) ---> key(one B)
		key(one B) ---> key(one A)
	1(opt)-to-1(opt)	key(one A) ---> key(one B)
		key(one B) ---> key(one A)
	1(opt)-to-many	key(many) ---> key(one)
	1-to-many	key(many) ---> key(one)
	many-to-many	composite-key ---> 0
Ternary	1-to-1-to-1	key(A),key(B) ---> key(C)
		key(A),key(C) ---> key(B)
		key(B),key(C) ---> key(A)
	1-to-1-to-many	key(one A),key(many) ---> key(one B)
		key(one B),key(many) ---> key(one A)
	1-to-many-to-many	key(many A),key(many B) ---> key(one)
	many-to-many-to-many	composite-key ---> 0
Generalization hierarchy		(secondary FD only)
Subset hierarchy		(secondary FD only)

Table 2. Primary FDs derivable from EER relationship constructs.

<u>Step 3.2</u> Examine all the candidate relations for MVDs and secondary FDs.

Each candidate relation is examined to determine what dependencies exist among primary key, foreign key, and nonkey attributes. If the EER constructs do not include nonkey attributes, the data requirements specification (or data dictionary) must be consulted.

The transformation process is a potential source of denormalization, particularly when a foreign key is embedded in a relation (Fig. 14), so that each foreign key must be accounted for in this analysis of secondary FDs. MVDs are most common in candidate relations that represent ternary relationships.

<u>Step 3.3</u> Normalize all candidate relations to the highest degree desired.

Each candidate relation now has possibly some primary FDs, secondary FDs, and MVDs uniquely associated with it. These dependencies determine the current degree of normalization of the relation (as defined in Appendix A). Any of the well-known techniques for increasing the degree of normalization can now be applied to each relation, with the highest degree desired as stated in the requirements specification.

<u>Step 3.4</u> Eliminate redundancies in the normalized relations.

The objective in this step is to minimize data redundancy, which in turn minimizes storage space and update cost, but without sacrificing data integrity. Integrity is maintained by constraining the new relation schema to include all data dependencies existing in the candidate normalized relation schema.

First, any relation A that is subsumed by another relation B can be unconditionally eliminated. A relation A is subsumed by another relation B when all data dependencies in A also occur in B. As a trivial case, any relation containing only a composite key and no nonkey attributes is automatically subsumed by any other relation containing the same key attributes because the composite key is the weakest form of data dependency.

Second, relations can also be subsumed by the join(s) of two or more other relations. When this occurs the elimination of a subsumed relation may result in the loss of retrieval efficiency, although storage and update costs will tend to be decreased. This tradeoff should be further analyzed in Step 4.2 with regard to usage requirements to determine whether elimination of the subsumed relation is reasonable.

4.2 Example

In Step 3.1 we obtain the primary FDs by applying the rules in Table 2 to each relationship in the EER diagram in Fig. 8. The results are shown below in Table 3.

```
 1. DIV-NO ---> HEAD-EMP-NO
 2. DEPT-NO ---> DIV-NO
 3. DEPT-NO ---> MANAG-EMP-NO
 4. EMP-NO ---> DEPT-NO
 5. EMP-NO ---> JOB-TITLE
 6. EMP-NO ---> SPOUSE-EMP-NO
 7. SPOUSE-EMP-NO ---> EMP-NO
 8. EMP-NO ---> PC-NO
 9. PC-NO ---> EMP-NO
10. EMP-NO,SKILL-NO,PROJ-NAME ---> 0
11. PROJ-NAME ---> LOC-NAME
12. EMP-NO,LOC-NAME ---> PROJ-NAME
```

Table 3. Functional dependencies derived from the EER diagram (example).

In Step 3.2 we determine the secondary FDs and MVDs from the EER diagram or requirements specification. Let us assume that the following dependencies are derived from the requirements specification:

```
1. DEPT-NO ---> DEPT-NAME
2. DEPT-NO ---> OFFICE-BLDG
3. EMP-NO ---> OFFICE-BLDG
4. EMP-NO ---> EMP-NAME
5. EMP-NO -->--> SKILL-NO
6. EMP-NO -->--> PROJ-NAME
7. EMP-NO -->--> LOC-NAME
```

Table 4. Secondary functional dependencies and MVDs (example).

Normalization of the candidate relations is accomplished in Step 3.3. In Table 1 all relations except 3, 14, and 15 are in 5NF already. Relation 14 is not even in 4NF because of the MVDs (dependencies 5 and 6 in Table 4). Also, relation 15, based on dependencies 11 and 12, is clearly not BCNF. Relation 14 must be decomposed into two relations containing the attributes EMP-NO and PROJ-NAME, and EMP-NO and SKILL-NO, respectively. Relation 15 must be decomposed into two relations containing dependencies PROJ-NAME ---> LOC-NAME and EMP-NO,PROJ-NAME ---> 0 to preserve the proper semantics while maintaining the proper form for BCNF.

Additional problems occur with the definition of secondary FDs (Table 4). For example, the EMPLOYEE relation (relation 3 in Table 1) having key EMP-NO; foreign key DEPT-

NO; and nonkeys EMP-NAME, JOB-TITLE, and OFFICE-BLDG; will not be 3NF because of the transitive functional dependency EMP-NO ---> DEPT-NO ---> OFFICE-BLDG. The simplest solution is to keep OFFICE-BLDG only in the DEPARTMENT relation and create a new relation EMP-OFF containing EMP-NO and OFFICE-BLDG (although the latter decision violates the guideline in Step 1.1(4)). We will see later that relation EMP-OFF will be subsumed in Step 3.4.

The modified or additional candidate relations reflecting these normalization decisions are:

Modified Relations

EMPLOYEE(EMP-NO,EMP-NAME,JOB-TITLE,...,DEPT-NO)
AVAIL-SKILL(EMP-NO,SKILL-NO)
ASSIGNED-TO(EMP-NO,PROJ-NAME)

New Relations

PROJ-LOC(PROJ-NAME,LOC-NAME)
EMP-OFF(EMP-NO,OFFICE-BLDG)

In Step 3.4 we attempt to eliminate data redundancies without losing data integrity. We can easily eliminate relation PROJ-LOC because it is subsumed by relation PROJECT. EMP-OFF can potentially be eliminated as well because it can be recreated by a join of relations EMPLOYEE and DEPARTMENT over the common attribute DEPT-NO. We will assume that this elimination will not decrease retrieval efficiency significantly, but this assumption should be verified in Step 4.2.

These normalization and reduction decisions produce the set of relations shown in Table 5 below.

1. DIVISION(DIV-NO,HEAD-EMP-NO,....)
2. DEPARTMENT(DEPT-NO,DEPT-NAME,OFFICE-BLDG,...,
   DIV-NO,MANAG-EMP-NO)
3. EMPLOYEE(EMP-NO,EMP-NAME,JOB-TITLE,...,DEPT-NO)
4. SKILL(SKILL-NO,....)
5. PROJECT(PROJ-NAME,...,LOC-NAME)
6. LOCATION(LOC-NAME,....)
7. EMP.MANAGER(EMP-NO,....)
8. EMP.ENGINEER(EMP-NO,....)
9. EMP.TECHNICIAN(EMP-NO,....)
10. EMP.SECRETARY(EMP-NO,....)
11. PC(PC-NO,....)
12. MARRIED-TO(EMP-NO,SPOUSE-EMP-NO)
13. HAS-ALLOCATED(EMP-NO,PC-NO)
14. AVAIL-SKILL(EMP-NO,SKILL-NO)
15. ASSIGNED-TO(EMP-NO,PROJ-NAME)

Table 5. Reduced normalized relations (example).

5.0 REFINEMENT OF RELATIONS FOR USAGE EFFICIENCY (STEP 4)

Database design techniques for network and hierarchical systems often make use of processing requirements to refine the logical schema before the physical design phase if there are obvious large efficiency gains to be made [Teorey and Fry 1982; Bertaina et al. 1983; Hawryszkiewycz 1984]. The justification for this approach is that once physical design begins, the logical schema is considered to be fixed, and is thus a constraint on efficiency. The database designer would often like to remove this inflexibility if possible. A similar technique could be applied to relational databases if it would produce more efficient database schemas without loss of data integrity, and would be relatively easy to implement.

Our goal is to define a relational schema refinement algorithm based on a process-oriented or usage view that could increase the database efficiency for current processing requirements, and yet retain all the information content of the functional dependency or natural view of data. Thus the database would still be an accurate representation of real-world relationships and potentially more adaptable to future processing requirements. The results of this algorithm could be used to specify alternative logical structures to be considered during physical design, and thus provide the physical designers with more feasible solutions to choose from. More efficient databases are therefore likely to be defined.

5.1 The Relation Usage Model

The relation usage approach, when applied to the design of centralized databases, is analogous to the use of fragmentation, data allocation, and data replication for distributed databases, except that it is much simpler for centralized databases [Ceri and Pelagatti 1984]. The original data is preserved in its entirety, and vertical fragments are replicated as extensions to current relations; also, data allocation is trivial because the data resides at a single site, and the replication is bounded by the number of processes that require relation refinements.

It is assumed that all attributes are initially assigned to relations based on functional dependencies, and that the relations are at least 3NF. This will establish the requirement for an accurate representation of reality and for flexibility of the design for future processing requirements. Efficiency for the current query requirements should increase by redundantly adding attributes, used together in a query, to an existing relation so that all attributes needed for that query reside in a new relation, called a join relation. This is known as materializing the join [Schkolnick and Sorenson 1980]. Access time will now be greatly reduced because fewer joins will be needed. However, the side effects of this redundancy include an increase in storage space required, an increase in the update cost, potential denormalization and loss of integrity, and program transformations for all applications containing joins that are materialized. These effects require careful consideration.

As an example of such a effect, let us assume that the relation PROJECT is associated with additional relations PART and SUPPLIER as shown in Fig. 15. We use Query by Exampler (QBE) to illustrate processes because of its extensive processing semantics [Zloof 1975]. The extension of the PART relation is shown as a means of reducing the number of joins required in the query. This extension results in a denormalization, with the side effects of add and update anomalies. However, the delete anomaly cannot occur because the original data is redundant in the extended schema. For example, SUPP-NO ---> SUPP-CITY in the extended PART relation (EXT-PART) is reproducible from PART-NO,PROJ-NAME ---> SUPP-NO in relation PART and SUPP-NO ---> SUPP-CITY in relation SUPPLIER.

The storage and processing cost of a logical relational database is to be computed for both the existing and new join relations:

$$COST = C_p * (T_q + T_u) + C_s * V_s \qquad (1)$$

where

C_p = unit cost per second for query or update processes

C_s = unit cost per byte for stored data

T_q = I/O service time for query processes (seconds)

T_u = I/O service time for update processes (seconds)

V_s = total volume in bytes for stored data

Unit costs are selected based on the computing environment defined in the requirements specification. I/O service time for query and update can be determined from the processing operations, their frequencies, and the hardware device characteristics given; while stored data volume can be obtained from the size of the relations defined [Teorey and Fry 1982; Hawryszkiewycz 1984]. Each query process must be expressed in terms of basic relational algebra operations such as selection, projection, and join. Some initial assumptions are made about sequential and random accesses needed to efficiently accomplish the query or update at this point, but detailed use of indexes, sorting, etc. are deferred to physical design when the final configuration decisions are made.

5.2 Design Step 4 Details

The relation usage strategy is to select only the most dominant processes to determine modifications to relations that will most likely improve performance. The basic modification is to add attributes to existing relations in order to reduce join operations.

Step 4.1 Select the dominant processes on the basis of criteria such as high frequency of execution, high volume of data accessed, response time constraints, or explicit high priority. As a rule of thumb any process with at least a factor of ten higher frequency of execution or data volume accessed than another process is considered to be more dominant.

Step 4.2 Define join relations, when appropriate, to materialize joins for dominant processes. Evaluate the total cost for storage, query, and update for the database schema, with and without the extended relation, and determine which configuration minimizes total cost. Consider also the possibility of denormalization due to a join relation and its side effects. If a join relation schema appears to have lower storage and processing cost and insignificant side effects, then consider that schema for physical design in addition to the original candidate relation schema. Otherwise consider only the original schema.

In general, joins based on nonkeys should be avoided. They are likely to produce very large relations, thus greatly increasing storage and update cost. For example, if two relations have 100 and 200 tuples, respectively, then a join based on the key of either one will result in a maximum of 200 tuples, but a join based on a nonkey of either one could result in a maximum of 100x200 or 20,000 tuples. Null values are also to be restricted to nonkey attributes so that they will not be inadvertently used in join operations.

5.3 Examples

The following examples, taken from the company personnel and project database design problem illustrated above, show the extremes of applicability and nonapplicability of the relation usage algorithm. In each case we apply the algorithm to a given relational schema and given processing requirements. Cost tradeoffs are then evaluated to determine if schema refinement is justifiable.

Example 5.3a

Example 5.3a illustrates the most favorable conditions for efficiency improvement with the relation usage algorithm (see Fig. 16). The query "display each pair of employee and project in which the project is located in the same city where the employee lives" is executed by a join of EMPLOYEE and ASSIGNED-TO over EMP-NO, followed by 20,000 random accesses to PROJECT (based on PROJ-NAME) to match LOC-NAME with each EMP-CITY in the temporary relation resulting from the join. To simplify the computation of query time the relations are assumed to be accessed as: EMPLOYEE (sequential, ordered on EMP-NO), PROJECT (indexed on PROJ-NAME), and ASSIGNED-TO (sequential, ordered on EMP-NO).

Using the hardware configuration for the Amdahl 5860 system at the University of Michigan (MTS), the following timing characteristics occur:

Page transfer time (at 4096 bytes per page):	3.4	ms
Average disk rotation time (half rotation):	8.3	ms
Average disk seek time:	16.0	ms

Average sequential page access = 11.7 ms
Average random page access = 27.7 ms
C_p= \$9.00 per I/O hour, C_s= \$.0031 per page-day

Given the number of bytes in each of the relations and the searching required for the query, we can calculate the I/O service time (T_q) for the query, and thus the total cost (Eq. 1). The remainder of the example is to determine the number of pages for query and update operations and storage space, and calculate total cost.

T_q = scan EMPLOYEE + scan ASSIGNED-TO + 20000 random accesses to PROJECT

$= \lceil 1{,}200{,}000/4096 \rceil * 11.7 + \lceil 400{,}000/4096 \rceil * 11.7 + 20000 * 27.7$ ms

= 558.575 sec

= .155 hour

I/O cost (query) = $C_p * T_q$

= 9.00*.155

= 1.396

I/O cost (at 100 queries per day) = 139.6

The update operation "delete a given employee from all associated projects" requires a random access to ASSIGNED-TO based on EMP-NO and a scan of an additional page to delete all tuples with a given EMP-NO.

T_u = 27.7 ms +11.7 ms

= .039 seconds

$$\text{I/O cost (update)} = C_p * T_u$$

$$= 9.00*.039/3600$$

$$= .0001$$

$$\text{I/O cost (at 100 updates per day)} = .01$$

$$\text{Storage cost} = C_s * V_s$$

$$= .0031 \text{ per page day } *(\lceil 1{,}200{,}000/4096 \rceil + \lceil 100{,}000/4096 \rceil + \lceil 400{,}000/4096 \rceil)$$

$$= .0031*416 \text{ pages}$$

$$= 1.29$$

$$\text{Total cost} = 139.6 + .01 + 1.29$$

$$= 140.9$$

The extended join relation solution is to append to ASSIGNED-TO the attributes EMP-CITY and LOC-NAME so that only a single scan of the new relation, which we will call EXT-ASSIGNED-TO, is needed to satisfy the query. EXT-ASSIGNED-TO now has 40 bytes per tuple; therefore at 20,000 tuples it has a total of 800,000 bytes and is double the size of ASSIGNED-TO. Redoing the calculations for query, update, and storage with EXT-ASSIGNED-TO, we obtain the cost figures shown in Table 6. We see that there is a dramatic reduction in cost using the extended join relation and avoiding the join and random indexing of the original solution.

Example 5.3b

Example 5.3b illustrates the least favorable conditions for efficiency improvement with the relation usage algorithm. The query given in Fig. 17 is executed by a join on the relations EMPLOYEE and DEPARTMENT over the common attribute DEPT-NO. This is accomplished by a scan of EMPLOYEE and DEPARTMENT. DEPARTMENT and EMPLOYEE are assumed to be accessed sequentially based on DEPT-NO.

T_q = scan of EMPLOYEE + scan of DEPARTMENT

$= \lceil 20{,}000{,}000/4096 \rceil * 11.7$ ms $+ \lceil 15{,}000/4096 \rceil * 11.7$ ms

= 57131 ms + 468 ms

= 57599 ms

I/O cost (query) = 9.00*57.599 sec/3600

= .144

I/O cost (query at frequency of 100 per day) = 14.4

The update of department number of every employee is accomplished with a scan of EMPLOYEE:

T_u = scan of EMPLOYEE

$= \lceil 20{,}000{,}000/4096 \rceil * 11.7$ ms

= 57131 ms

I/O cost (update) = 9.00*57.131 sec /3600

$= .143$

I/O cost (update at frequency of 100 per day) = 14.3

$$\text{Storage cost} = \lceil 20{,}000{,}000/4096 \rceil * .0031 + \lceil 15{,}000/4096 \rceil * .0031 \text{ per day} = 15.1 \text{ per day}$$

The extended join relation solution is to add the attributes DEPT-NAME and OFF-NO to relation EMPLOYEE, thus increasing the tuple size from 200 to 250 bytes. The size of the entire relation EXT-EMPLOYEE is 25 MB, compared to 20 MB for EMPLOYEE. The cost for query, update, and storage space for the extended relation is shown in Table 6. The results show higher cost in all three areas due to the extended join relation, mainly because the relation EMPLOYEE is much larger than the relation DEPARTMENT and the extension EXT-EMPLOYEE is larger than EMPLOYEE and DEPARTMENT combined. Thus, the join relation schema is not a candidate for physical design in this case.

To summarize, the extended join relation tends to significantly lower the storage and processing cost for one or more joins if either the joined relations are of comparable size, if only the smaller relation is extended, or if it can avoid a large number of random accesses to at least one of the relations.

Example 5.3a

	Original Relation	Join Relation	
C_p*T_q	139.6	.57	Query cost
C_p*T_u	.01	.01	Update cost
C_s*V_s	1.29	1.59	Storage cost
	------	------	
	140.9	2.17	Total cost

Example 5.3b

	Original Relation	Join Relation	
C_p*T_q	14.4	18.0	Query cost
C_p*T_u	14.3	17.9	Update cost
C_s*V_s	15.1	18.9	Storage cost
	-----	--------	
	43.8	54.8	Total cost

Table 6. Summary of total cost per day (examples)

6.0 CONCLUSION

We have shown that a practical step-by-step methodology for relational database design can be derived using the extended ER conceptual model. The methodology has been illustrated with a simple database design problem, showing each design step in detail. The strategy of first modeling the natural data relationships and later refining the design for processing efficiency was emphasized as two clearly separable phases. The methodology produces nearly perfectly reproducible designs and can be easily implemented as an interactive database design tool.

ACKNOWLEDGMENTS

The authors wish to thank Emerson Hevia for his original ideas on the concept of usage dependency as a refinement of functional dependency for processing efficiency.

Appendix A. Basic Normal Forms [Date 1985]

1NF -- a relation R is in 1NF if and only if all underlying domains contain atomic values only, i.e., tuples do not contain any repeating groups. ("the key")

2NF -- a relation R is in 2NF if and only if it is in 1NF and every nonkey attribute is fully dependent on the primary key. ("the whole key")

3NF -- a relation R is in 3NF if and only if it is in 2NF and every nonkey attribute is nontransitively dependent on the primary key. ("nothing but the key")

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

4NF -- a relation R is an 4NF if and only if, whenever there exists a multivalued dependency in R, say A -->--> B, then all attributes of R are also functionally dependent on A. A 4NF relation cannot have independent multivalued facts about an entity.

5NF -- a relation R is in 5NF if and only if every join dependency in R is implied by the candidate keys of R. A relation R satisfies the join dependency if and only if it is the join of its projections on the subsets of the set of attributes of R (see [Date 1985] and [Kent 1983] for examples).

Appendix B. Summary of Logical Relational Database Design Steps

1. Extended ER (EER) modeling of requirements.

   1.1 Identify entities and attach attributes to each.

   1.2 Identify generalization and subset hierarchies.

   1.3 Define relationships.

   1.4 Integrate multiple views of entities, attributes and relationships.

2. Transformation of the EER model to relations.

   2.1 Transform every entity into one relation with the key and nonkey attributes of the entity.

   2.2 Transform every many-to-many binary (or unary) relationship and every fully conditional or unconditional one-to-one binary (or unary) relationship into a relationship relation.

   2.3 Transform every ternary (or higher n-ary) relationship into a relationship relation.

3. Normalization of relations.

   3.1 Derive the primary FDs from the EER diagram.

   3.2 Examine all the candidate relations for MVDs and secondary FDs.

   3.3 Normalize all candidate relations to the highest degree desired.

   3.4 Eliminate redundancies in the normalized relations.

4. Refinement of relations for usage efficiency.

   4.1 Select the dominant processes on the basis of high frequency of execution, high volume of data accessed, response time constraints, or explicit high priority.

   4.2 Define a join relation, when appropriate, to materialize joins for dominant processes.

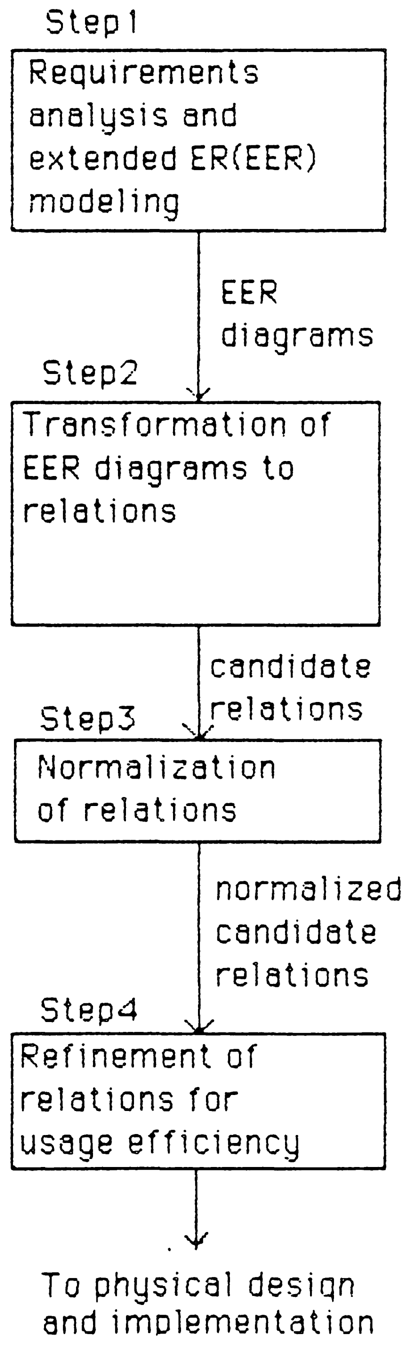

Step1
Requirements analysis and extended ER(EER) modeling
EER diagrams
Step2
Transformation of EER diagrams to relations
candidate relations
Step3
Normalization of relations
normalized candidate relations
Step4
Refinement of relations for usage efficiency
To physical design and implementation

Figure 1 Relational database design : basic steps

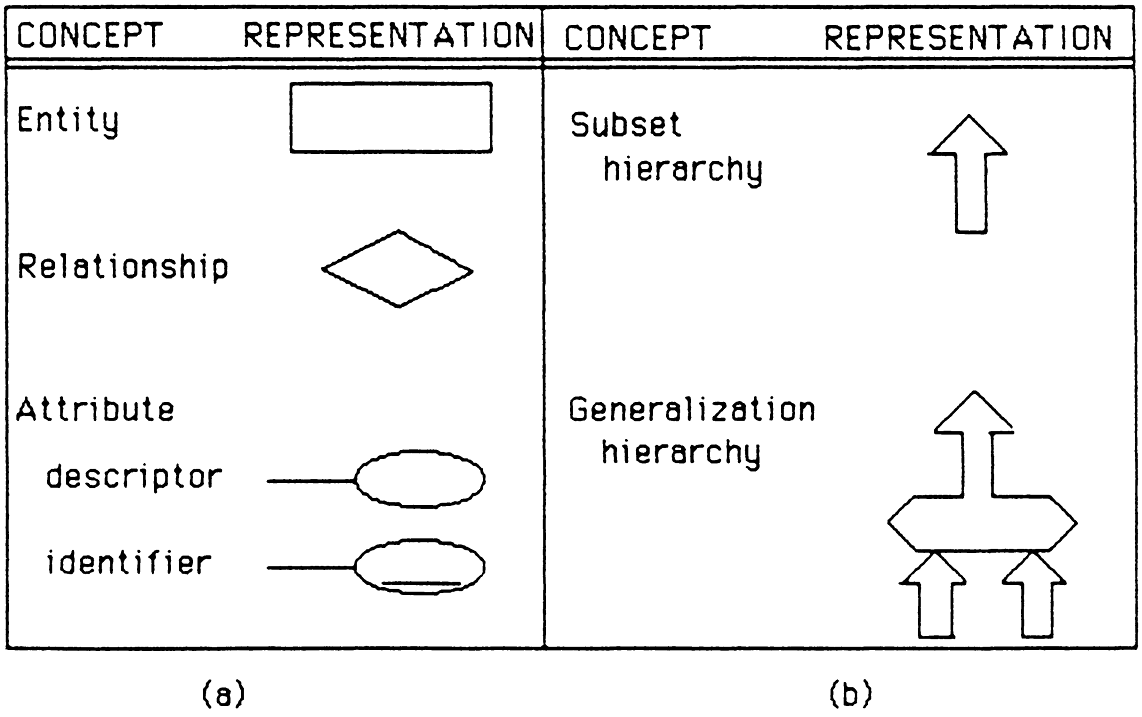


Figure 2 Extended ER(EER) model representations

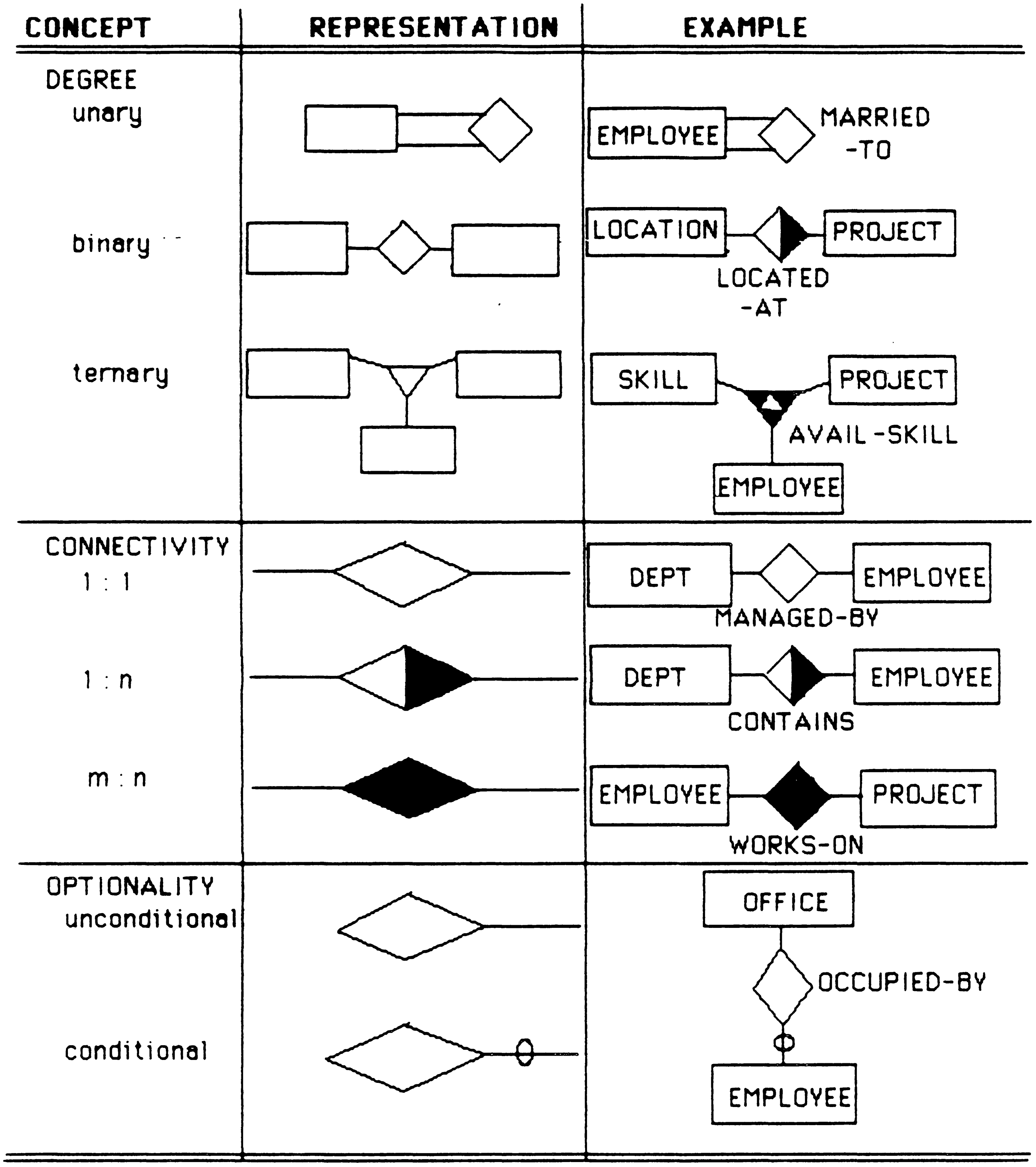


Figure 3 Fundamental EER constructs : relationship types

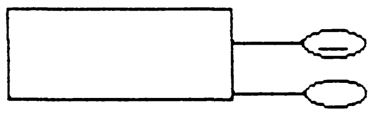
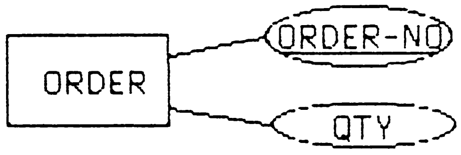
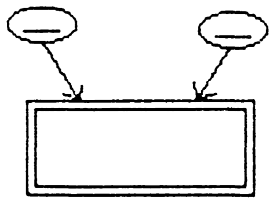
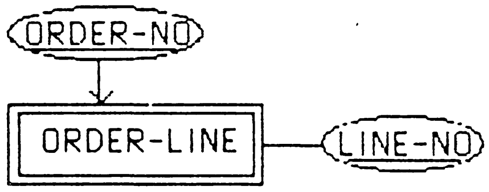
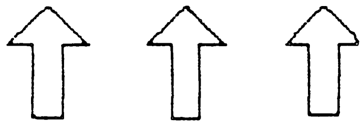
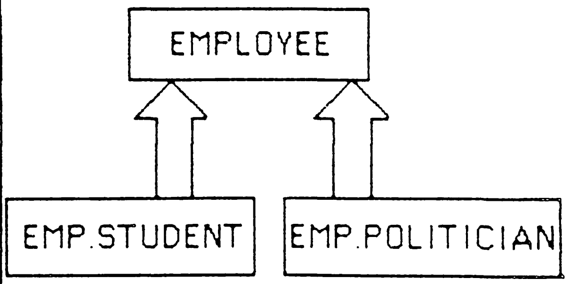
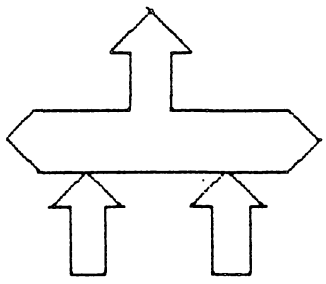
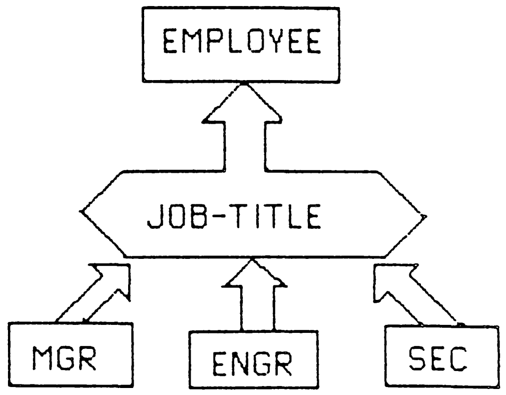
CONCEPT	REPRESENTATION	EXAMPLE
ENTITY		
entity(strong)		
entity(weak)		
OBJECT CLASS		
subset hierarchy		
generalization hierarchy		

Figure 4 Fundamental EER constructs : entity and object classes

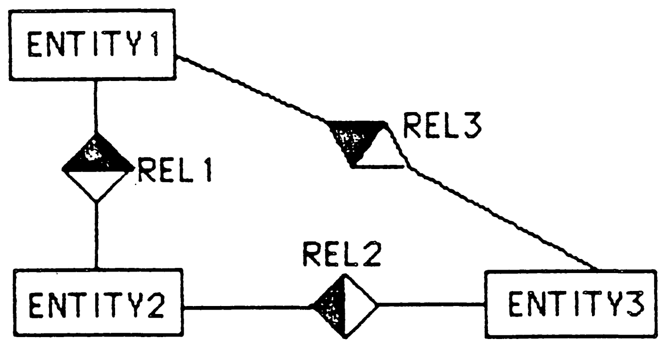


Figure 5 Transitive relationships

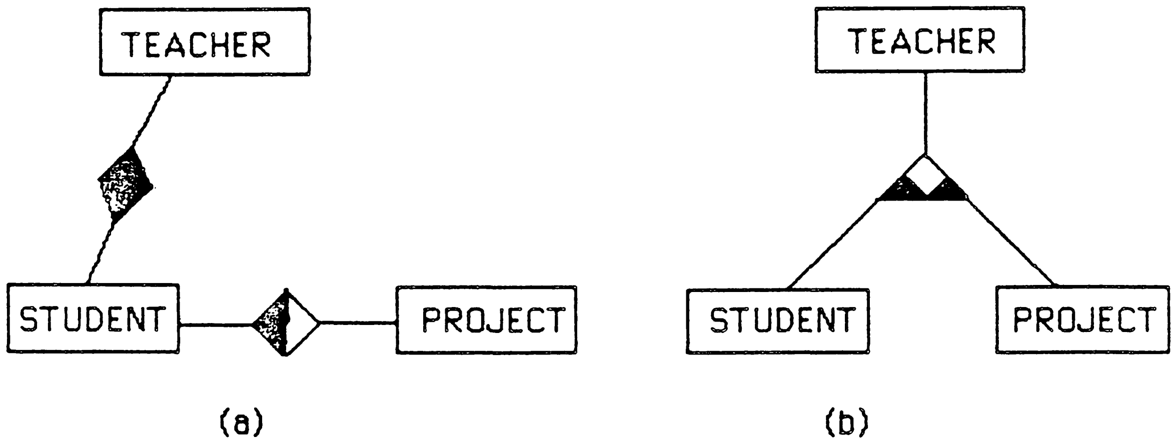


Figure 6 Comparison of binary and ternary relationships

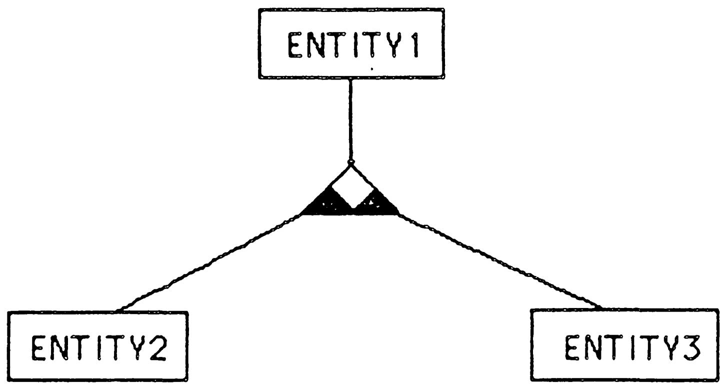


Figure 7 Example of connectivity for ternary relationship

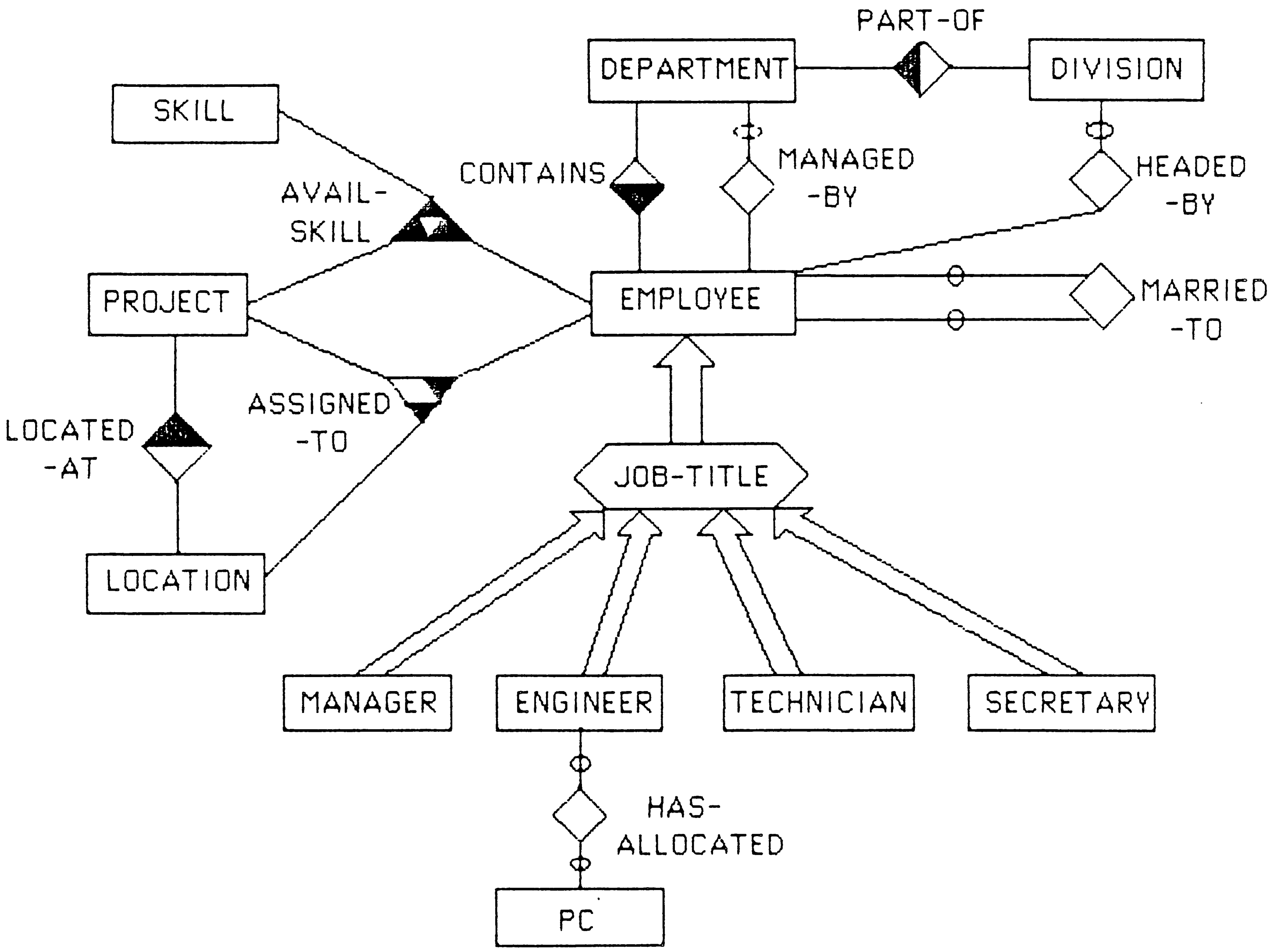
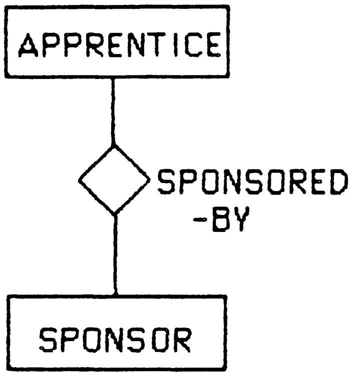


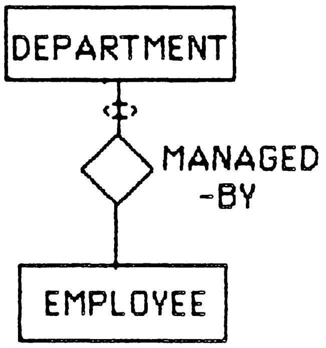
Figure 8 Company personnel and project database (EER diagram)



(a)

Every apprentice has one sponsor, and every sponsor sponsors one apprentice.

Relations:
APPRENTICE(EMP-NO, ...)
SPONSOR(SPON-EMP-NO, ...)
SPONSORED-BY(EMP-NO,SPON-EMP-NO)

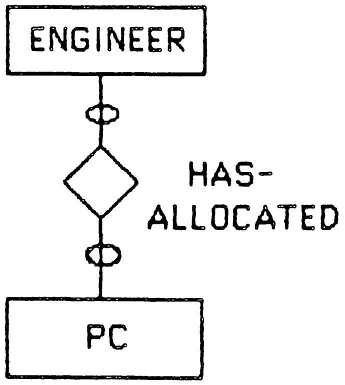


(b)

Every department must have a manager. An employee can be a manager of at most one department.

Relations :
DEPARTMENT(DEPT-NO, ... ,EMP-NO)
EMPLOYEE(EMP-NO, ...)

Null EMP-NO not allowed in DEPARTMENT.

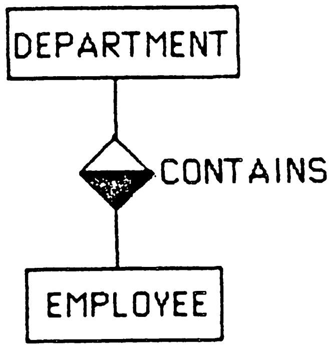


(c)

Some personal computer (PCs) are allocated to engineers, but not necessarily to all engineers.

Relations :
ENGINEER(EMP-NO, ...)
PC(PC-NO, ...)
HAS-ALLOCATED(EMP-NO,PC-NO)

Figure 9 Binary relationship transformation rules

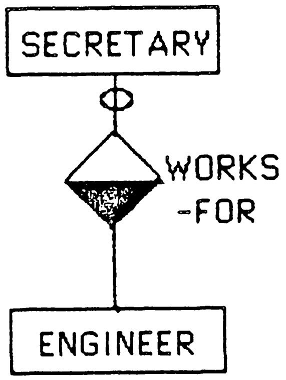


(d)

Every employee works in exactly one department. Every department could contain many employees.

Relations :
DEPARTMENT(DEPT-NO, ...)
EMPLOYEE(EMP-NO, ..., DEPT-NO)

Null DEPT-NO not allowed in EMPLOYEE.

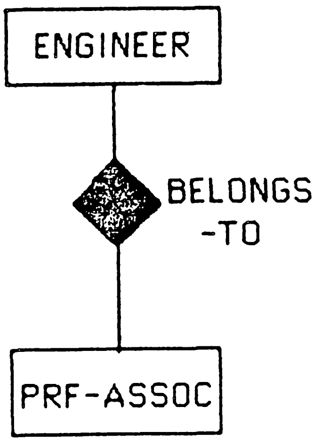


(e)

Each engineer can have at most one secretary. One secretary could work for several engineers.

Relations :
ENGINEER(EMP-NO, ... , SEC-EMP-NO)
SECRETARY(EMP-NO, ...)

Null SEC-EMP-NO allowed in ENGINEER.

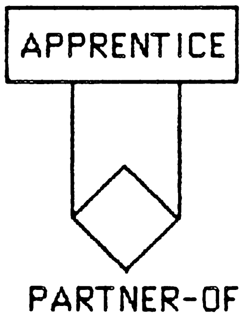


(f)

Every professional association could have many members who are engineers. Every engineer could belong to many professional associations.

Relations :
PRF-ASSOC(PA-NO, ...)
ENGINEER(EMP-NO, ...)
BELONGS-TO(PA-NO, EMP-NO)

Figure 9 continued

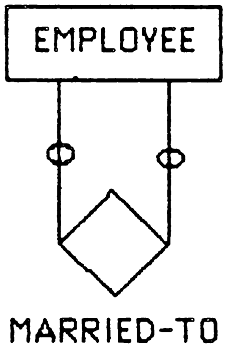


(a)

Every apprentice has exactly one of the other apprentice as a partner in a project.

Relations :

APPRENTICE(EMP-NO, ...)
PARTNER-OF(EMP-NO, PA-EMP-NO)

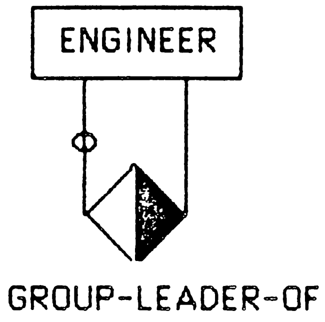


(b)

An employee could have one of the other employee as his or her spouse.

Relations :

EMPLOYEE(EMP-NO, ...)
MARRIED-TO(EMP-NO, SP-EMP-NO)



(c)

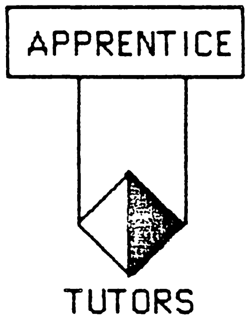
Engineers are divided into groups for certain projects. Each group has a leader.

Relation :

ENGINEER(EMP-NO, ..., ENG-EMP-NO)

Null ENG-EMP-NO allowed in ENGINEER.

Figure 10 Unary relationship transformation rules

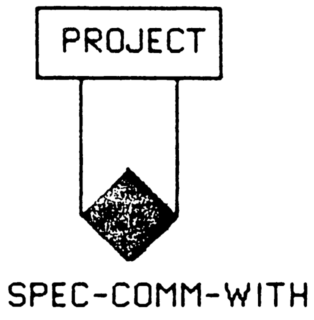


(d)

Every apprentice tutors one of the other apprentices. One may be tutored by several other apprentices.

Relation :
APPRENTICE(EMP-NO, ..., APP-EMP-NO)

Null APP-EMP-NO not allowed in APPRENTICE.

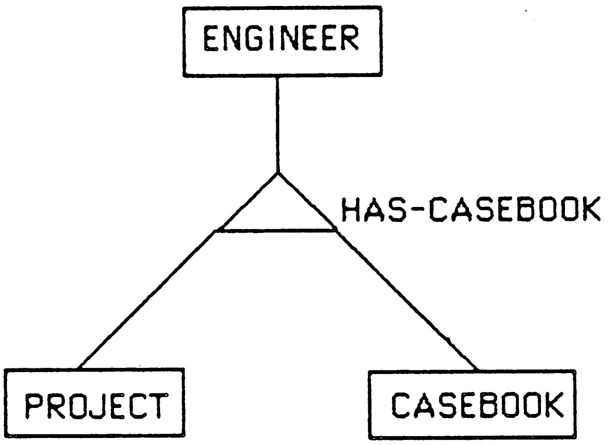


(e)

Each project may require special communication with many other projects.

Relations :
PROJECT(PROJ-NO, ...)
SPEC-COMM-WITH(PROJ-NAME, RELA-PROJ-NAME)

Figure 10 continued



An engineer will purchase one casebook for a given project. Different engineers use different casebooks for the same project. No engineer will use the same casebook for different projects.

<u>Relations :</u>
ENGINEER(<u>EMP-NO</u>, ...)
PROJECT(<u>PROJ-NAME</u>, ...)
CASEBOOK(<u>BOOK-NO</u>, ...)
HAS-CASEBOOK(<u>EMP-NO, PROJ-NAME</u>, <u>BOOK-NO</u>)

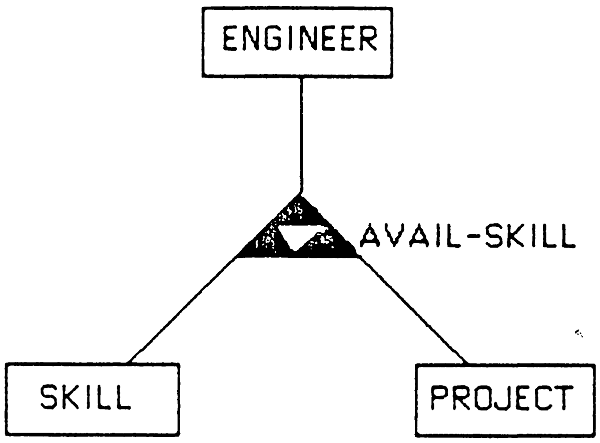
FDs : EMP-NO, PROJ-NAME ---> BOOK-NO
EMP-NO, BOOK-NO ---> PROJ-NAME
BOOK-NO, PROJ-NAME ---> EMP-NO

HAS-CASEBOOK

EMP-NO	PROJ-NAME	BOOK-NO
3	ALPHA	1001
3	BETA	1008
4	DELTA	1004
4	GAMMA	1005
8	BETA	1007
9	ALPHA	1009
9	EPSILON	1012

(a)

Figure 11 Ternary relationship transformation rules



Employees use a wide range of different skills on each project they are associated with.

Relations :
EMPLOYEE(EMP-NO, ...)
SKILL(SKILL-NO, ...)
PROJECT(PROJ-NAME, ...)
AVAIL-SKILL(EMP-NO, SKILL-NO, PROJ-NAME)

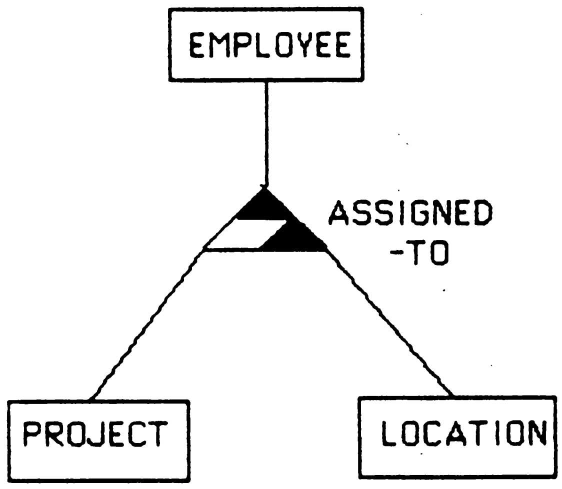
FDs : EMP-NO, SKILL-NO, PROJ-NO ---> ϕ
(all key)

AVAIL-SKILL

EMP-NO	SKILL-NO	PROJ-NAME
3	A3	ALPHA
3	A3	BETA
3	B6	ALPHA
3	B6	BETA
4	G12	DELTA
4	G12	GAMMA
8	A3	BETA
8	C4	BETA
9	A5	ALPHA
9	A5	EPSILON
9	C8	ALPHA
9	C8	EPSILON

(b)

Figure 1 1 continued



Employees are assigned to one or more projects, with each project at a different location. Many projects may reside a particular location, but each project may have only one location.

Relations :
EMPLOYEE(EMP-NO, ...)
PROJECT(PROJ-NAME, ...)
LOCATION(LOC-NAME, ...)
ASSIGNED-TO(EMP-NO, LOC-NAME,PROJ-NAME)

FDs : EMP-NO, LOC-NAME ---> PROJ-NAME

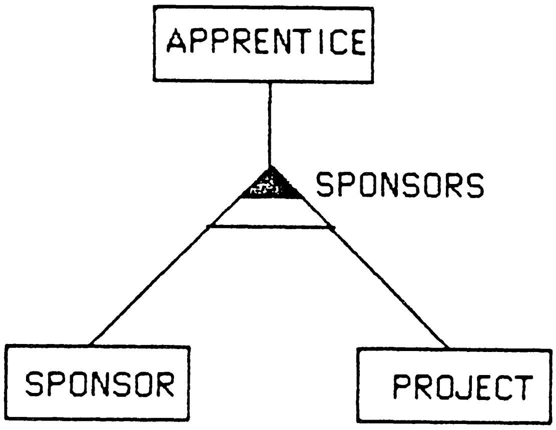
Additionally, PROJ-NAME ---> LOC-NAME is implied in this case by the narrative statement. This is represented in a binary relationship between PROJECT and LOCATION (see Fig. 8).

ASSIGNED-TO

EMP-NO	LOC-NAME	PROJ-NAME
3	DETROIT	BETA
3	NEW-YORK	ALPHA
4	CHICAGO	GAMMA
4	NEW-YORK	DELTA
8	DETROIT	BETA
9	CHICAGO	OMEGA
9	DETROIT	EPSILON

(c)

Figure 11 continued



An apprentice must have a different sponsor for each project.

Relations :
APPRENTICE(EMP-NO, ...)
SPONSOR(EMP-NO, ...)
PROJECT(PROJ-NAME, ...)
SPONSORS(SPON-EMP-NO, APP-EMP-NO, PROJ-NAME)

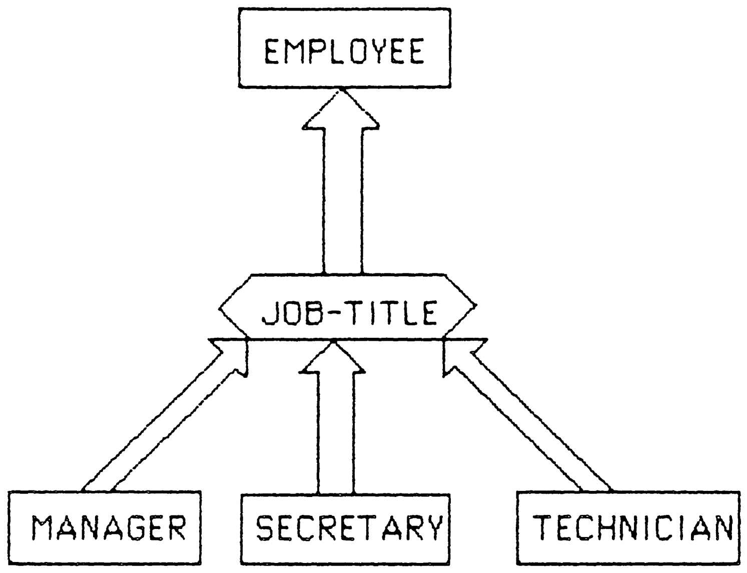
FDs : APP-EMP-NO,SPON-EMP-NO ---> PROJ-NAME
APP-EMP-NO,PROJ-NAME ---> SPON-EMP-NO

SPONSORS

APP-EMP-NO	SPON-EMP-NO	PROJ-NAME
101	3	BETA
101	9	EPSILON
207	3	ALPHA
207	4	DELTA
512	4	GAMMA
512	9	ALPHA

(d)

Figure 11 continued



Different types of employees are partitioned by values of a common attribute JOB-TITLE.

Relations :

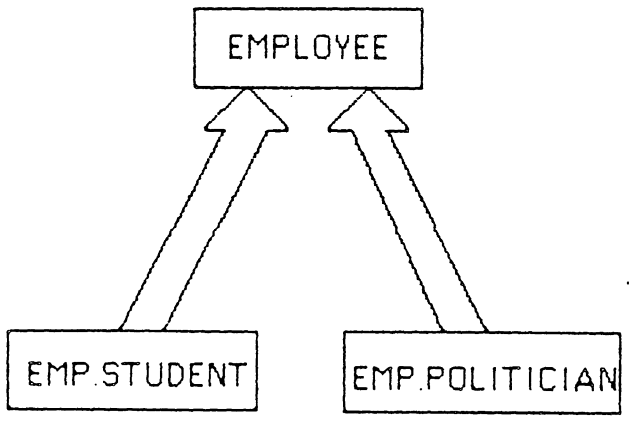
EMPLOYEE(EMP-NO, JOB-TITLE, common attributes)

EMP.MANAGER(EMP-NO, specific attributes)

EMP.SECRETARY(EMP-NO, specific attributes)

EMP.TECHNICIAN(EMP-NO, specific attributes)

Figure 12 Generalization hierarchy



Employees with special situations are shown as overlapping subsets based on partitions on values of different attributes.

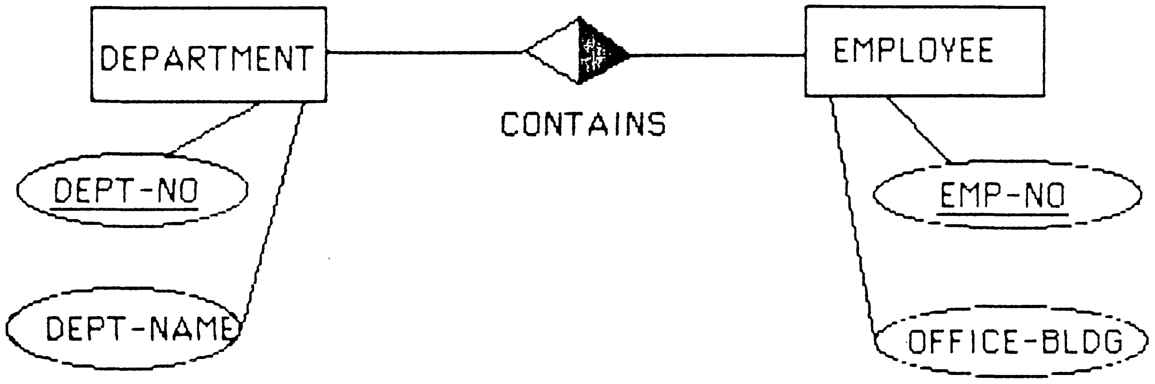
Relations :

EMPLOYEE(EMP-NO, common attributes)

EMP.STUDENT(EMP-NO, specific attributes)

EMP.POLITICIAN(EMP-NO, specific attributes)

Figure 13 Subset hierarchy



Relation with foreign key :
EMPOYEE(<u>EMP-NO</u>......,OFFICE-BLDG,DEPT-NO)

Figure 14 Example ER-to-relation transformation causing denormalization

Original relations and process (query)

PART(PART-NO,PROJ-NAME,SUPP-NO,PRICE)
SUPPLIER(SUPP-NO,SUPP-CITY,SUPP-MGR)
PROJECT(PROJ-NAME,LOC-NAME)

Query: For a given project, display the supplier numbers, supplier cities, and project city.

Functional dependencies

PART-NO,PROJ-NAME --> SUPP-NO | PRICE
SUPP-NO --> SUPP-CITY | SUPP-MGR
PROJ-NAME --> LOC-NAME

QBE representation of the query

PART	PART-NO	PROJ-NAME	SUPP-NO	PRICE
		*	P.X	

SUPPLIER	SUPP-NO	SUPP-CITY	SUPP-MGR
	X	P.Y	

PROJECT	PROJ-NAME	LOC-NAME
	*	P.Z

Extended relation PART in 1NF

EXT-PART(PART-NO,PROJ-NAME,SUPP-NO,SUPP-CITY,
LOC-NAME,PRICE)
SUPPLIER(SUPP-NO,SUPP-CITY,SUPP-MGR)
PROJECT(PROJ-NAME,LOC-NAME)

Figure 15. Relation extension causing denormalization from 3NF to 1NF and 2NF.

Relations	Bytes/tuple	Tuples	T.Bytes
EMPLOYEE(EMP-NO,EMP-CITY,..)	120	10000	1200 KB
PROJECT(PROJ-NAME,LOC-NAME,..)	200	500	100 KB
ASSIGNED-TO(EMP-NO,PROJ-NAME,..)	20	20000	400 KB

Query: Display each pair of employee and project in which the project is located in the same city where the employee lives.

Update: Delete a given employee from all associated projects.

QBE representation of the query:

EMPLOYEE	EMP-NO	EMP-CITY
	P.X	Z

ASSIGNED-TO	EMP-NO	PROJ-NAME
	X	Y

PROJECT	PROJ-NAME	LOC-NAME
	P.Y	Z

QBE representation of the update:

ASSIGNED-TO	EMP-NO	PROJ-NAME
D.	*	X

PROJECT	PROJ-NAME	LOC-NAME
	X	

Unit costs: C_p = 9.00 per disk-hour
C_s = .0031 per page-day
Frequency of all processes: 100/day

Figure 16. Example 5.3a relations and processes.

Relations	Bytes/tuple	Tuples	T.Bytes
EMP(EMP-NO,EMP-NAME,AUTO-TYPE,DEPT-NO)	200	10000	2000 KB
DEPT(DEPT-NO,DEPT-NAME,OFF-NO,EMP-NO)	250	60	15 KB

Query: Display employee number, name, office, and department name for all employees with a given automobile type.

Update: Scan the employee relation and make necessary changes as specified in an in-core update list.

QBE representation of the query:

EMPLOYEE	EMP-NO	EMP-NAME	AUTO-TYPE	DEPT-NO
	P.A	P.B	*	X

DEPARTMENT	DEPT-NO	DEPT-NAME	OFF-NO	EMP-NO
	X	P.C	P.D	

QBE representation of the update:

EMPLOYEE	EMP-NO	EMP-NAME	AUTO-TYPE	DEPT-NO
U.	*	*	*	*

Frequency of all processes: 100/day

Figure 17. Example 5.3b relations and processes.

References

ATZENI,P., BATINI,C. LENZERINI.M, AND VILLANELLI,F. 1981. INCOD: A System for Conceptual Design of Data and Transactions in the Entity-Relationship Model. Entity-Relationship Approach to Information Modeling and Analysis, ER Institute, Saugus, CA.

BERNSTEIN,P. 1976. Synthesizing 3NF Relations from Functional Dependencies. ACM Trans. on Database Systems 1,4, 272-298.

BERTAINA,P., DILEVA,A., AND GIOLITO,P. 1983. Logical Design in CODASYL and Relational Environmentl. In Methodology and Tools for Data Base Design, S. Ceri (Ed.), North-Holland, 85-117.

CERI,S. AND PELAGATTI,G. 1984. Distributed Databases: Principles and Systems, McGraw-Hill, New York.

CHIANG,T. AND BERGERON,R. 1980. A Data Base Management System with an E-R Conceptual Model. Entity-Relationship Approach to Systems Analysis and Design, P. Chen (Ed.), North-Holland, Amsterdam, 467-476.

CHEN,P. 1976. The Entity-Relationship Model - Toward a Unified View of Data. ACM Trans. of Database Systems 1,1, 9-36.

CODD,E. 1970. A Relational Model for Large Shared Data Banks. Comm. ACM 13,6, 377-387.

CODD,E. 1974. Recent Investigations into Relational Data Base Systems. Proc. IFIP Congress.

DATE,C. 1984. A Guide to DB2, Addison-Wesley, Reading, MA.

DATE,C. 1985. An Introduction to Database Systems, Vol. 1 (4th Ed.), Addison-Wesley, Reading, MA.

FAGIN,R. 1977. Multivalued Dependencies and a New Normal Form for Relational Databases. ACM Trans. on Database Systems 2,3, 262-278.

FONG,E., HENDERSON,M., JEFFERSON,D., AND SULLIVAN,J. 1985. Guide on Logical Database Design, NBS Spec. Pub. 500-122, U.S. Dept. of Commerce.

HAWRYSZKIEWYCZ,I. 1984.Database Analysis and Design, SRA, Chicago.

HOWE,D. 1983. Data Analysis and Data Base Design, Arnold

Pub., London.

JAJODIA,S. AND NG,P. 1983. On the Representation of Relational Structures by Entity-Relationship Diagrams. The Entity-Relationship Approach to Software Engineering, G.C. Davis et al. (Eds.), Elsevier (North-Holland), New York, 223-248.

JAJODIA,S. AND NG,P. 1984. Translation of Entity-Relationship Diagrams into Relational Structures. J. of Systems and Software 4, 123-133.

KENT,W. 1983. A Simple Guide to Five Normal Forms in Relational Database Theory. Comm. ACM 26,2, 120-125.

KENT,W. 1984. Fact-Based Data Analysis and Design. J. of Systems and Software 4, 99-121.

LENZERINI,M. AND SANTUCCI,G. 1983. Cardinality Constraints in the Entity-Relationship Model. The Entity-Relationship Approach to Software Engineering, G.C. Davis et al.(Eds.), Elsevier (North-Holland), New York, 529-549.

LIEN,Y. 1981. Hierarchical Schemata for Relational Databases. ACM Trans. on Database Systems 6,1, 48-69.

LIEN,Y. 1982. On the Equivalence of Data Models. J. ACM 29,2, 333-362.

LING,T. 1985. A Normal Form for Entity-Relationship Diagrams. Proc. 4th International Conf. on the E-R Approach, Chicago, 1984, IEEE Society Press, 24-35.

MAIER,D. 1983. Theory of Relational Databases, Computer Science Press, Rockville, MD.

MARTIN,J. 1982. Strategic Data-Planning Methodologies, Prentice-Hall, Englewood Cliffs, NJ.

MARTIN,J. 1983. Managing the Data-Base Environment, Prentice-Hall, Englewood Cliffs, NJ.

McGEE,W. 1974. A Contribution to the Study of Data Equivalence. Data Base Management, J.W. Klimbie and K.L. Koffeman, Eds., North-Holland Pub. Co., Amsterdam, 123-148.

NAVATHE,S. AND CHENG,A. 1983. A Methodology for Database Schema Mapping from Extended Entity Relationship Models into the Hierarchical Model. The Entity-Relationship Approach to Software Engineering, G.C. Davis et al.(Eds.), Elsevier (North-Holland), New York, 223-248.

REINER,D., BRODIE,M., BROWN,G., FRIEDELL,M., KRAMLICH,D., LEHMAN,J., AND ROSENTHAL,A. 1985. The Database Design and Evaluation Workbench (DDEW) Project at CCA. Database Engineering 7,4, 10-15.

SAKAI,H. 1983. Entity-Relationship Approach to Logical Database Design. Entity-Relationship Approach to Software Engineering, C.G. Davis, S. Jajodia, P. A. Ng, and R.T. Yeh, Eds., Elsevier Science Pub. B.V. (North-Holland), New York, 155-187.

SCHEUERMANN,P., SCHEFFNER,G., AND WEBER,H. 1980. Abstraction Capabilities and Invariant Properties Modelling within the Entity-Relationship Approach. Entity-Relationship Approach to Systems Analysis and Design, P. Chen (Ed.), North-Holland, Amsterdam, 121-140.

SCHKOLNICK,M. AND SORENSON,P. 1980. DENORMALIZATION: A Performance Oriented Database Design Technique. Proc. AICA 1980 Congress, Bologna, Italy, 363-377.

SMITH,H. 1985. Database Design: Composing Fully Normalized Tables from a Rigorous Dependency Diagram. Comm. ACM 28,8, 826-838.

SMITH,J. AND SMITH,D. 1977. Database Abstractions: Aggregation and Generalization. ACM Trans. on Database Systems 2,2, 105-133.

TEOREY,T. AND FRY,J. 1982. Design of Database Structures, Prentice-Hall, Englewood Cliffs, NJ.

ULLMAN,J. 1980. Principles of Database Systems, Computer Science Press, Potomac, MD.

WILMOT,R. 1984. Foreign Keys Decrease Adaptability of Database Designs. Comm. ACM 27,12, 1237-1243.

YAO,S. 1985. Principles of Database Design: Volume 1 - Logical Organizations, Prentice-Hall, Inc., Englewood Cliffs, NJ.

ZANIOLO,C. AND MELKANOFF,M. 1981. On the Design of Relational Database Schemas. ACM Trans. on Database Systems 6,1, 1-47.

ZLOOF,M. 1975. Query by Example. Proc. NCC, 1975, AFIPS Press, Montvale, NJ, 431-438.

0123456 A4 Page 6543210

AIIM SCANNER TEST CHART #2

Spectra

4 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

6 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

8 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

10 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

Times Roman

4 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

6 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

8 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

10 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

Century Schoolbook Bold

4 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

6 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

8 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

10 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

News Gothic Bold Reversed

4 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

6 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

8 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

10 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

Bodoni Italic

4 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

6 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

8 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

10 PT ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;:",./?$0123456789

Greek and Math Symbols

4 PT ΑΒΓΔΕΞΘΗΙΚΛΜΝΟΠΦΡΣΤΥΩΧΨΖαβγδεξθηικλμνοπφρστυωχψζ≧∓",./≦±=≠°>≮≯≶≷≡

6 PT ΑΒΓΔΕΞΘΗΙΚΛΜΝΟΠΦΡΣΤΥΩΧΨΖαβγδεξθηικλμνοπφρστυωχψζ≧∓",./≦±=≠°>≮≯≶≷≡

8 PT ΑΒΓΔΕΞΘΗΙΚΛΜΝΟΠΦΡΣΤΥΩΧΨΖαβγδεξθηικλμνοπφρστυωχψζ≧∓",./≦±=≠°>≮≯≶≷≡

10 PT ΑΒΓΔΕΞΘΗΙΚΛΜΝΟΠΦΡΣΤΥΩΧΨΖαβγδεξθηικλμνοπφρστυωχψζ≧∓",./≦±=≠°>≮≯≶≷≡

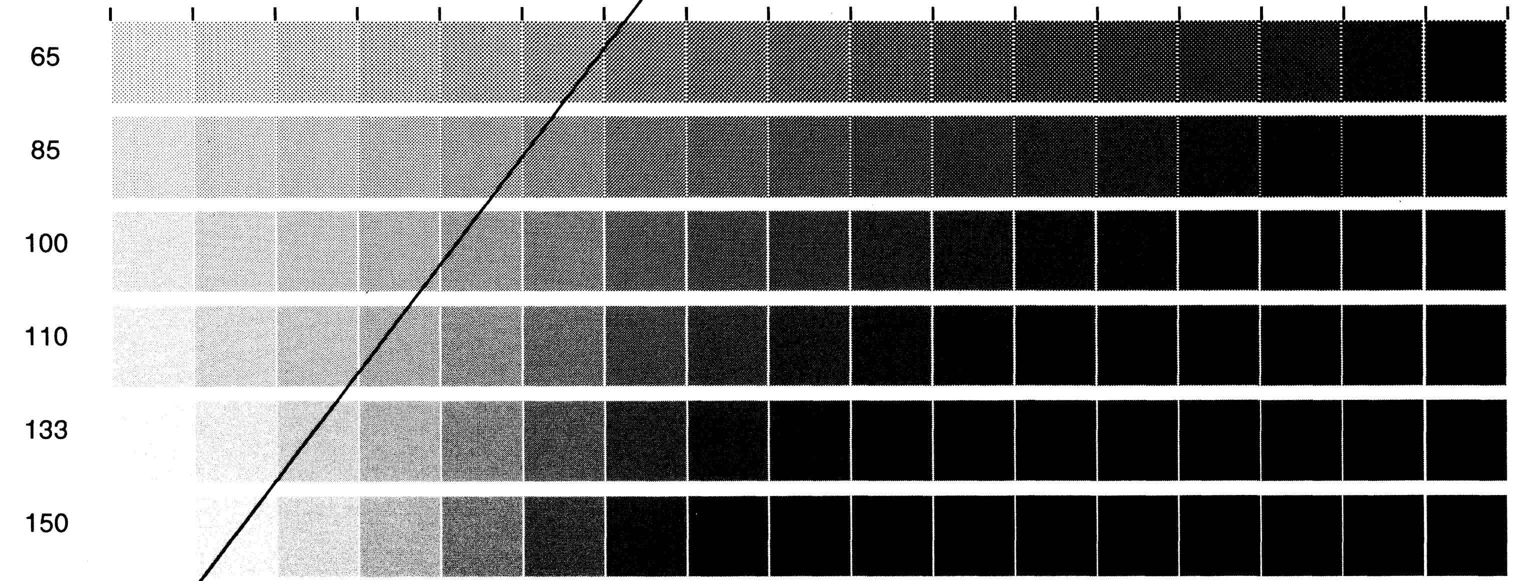
White ☐ Black ■

Isolated Characters

e	m	1	2	3	a
4	5	6	7	o	°
8	9	0	h	I	B

0123456 A4 Page 6543210

MESH HALFTONE WEDGES



A4 Page 6543210

0123456

RIT ALPHANUMERIC RESOLUTION TEST OBJECT, RT-1-71

PRODUCED BY GRAPHIC ARTS RESEARCH CENTER

ROCHESTER INSTITUTE OF TECHNOLOGY, ONE LOMB

MEMORIAL DRIVE, ROCHESTER, NEW YORK 14623