[BCN 92]

CHAPTER 6

Improving the Quality of a Database Schema

Like other industrial products, a database schema must be validated before it becomes a stable product of the design. This validation process is performed by checking the schema for several qualities that we examine in this chapter: completeness, correctness, expressiveness, readability, minimality, self-explanation, extensibility, and normality. These qualities should be checked periodically during the design process and should actually drive the design process. Testing these qualities is most effective when the design is complete, because we have a global view of the conceptual schema.

The fundamental tools at our disposal for improving the qualities of a schema are *schema transformations*. We have seen in Chapter 3 how primitive transformations can be used in the design process: in that case their role is to enrich the schema with new information. We have addressed other transformations in Chapter 5, which are used to achieve compatibility between schemas. In this chapter, the goal of applying transformations is to *restructure* the schema to produce a better version in terms of the above qualities.

The chapter is organized as follows. In Section 6.1 we formally define the qualities of a database schema. Section 6.2 describes the types of transformations that can be used to improve the overall quality of the schema. Sections 6.3, 6.4, and 6.5 deal with specific qualities (minimality, self-explanation, expressiveness, readability, normality) and define several rules and methods for achieving them. Section 6.6 deals with an example of schema restructuring: we will use the final schema (Figure 5.8) from the integration activity in Chapter 5.

15p

6.1 Qualities of a Database Schema

In this section we give an answer to the question: What is a good schema? We distinguish several qualities that are typical of a conceptual schema.

Completeness. A schema is complete when it represents all relevant features of the application domain. Completeness can be checked in principle by (1) looking in detail at all requirements of the application domain and checking to see that each of them is represented somewhere in the schema (in this case we say that *the schema is complete with respect to requirements*); and (2) checking the schema to see that each concept is mentioned in the requirements (in this case we say that *requirements are complete with respect to the schema*).

Completeness can be checked more effectively by comparing the data schema against the function schema. We will deal with this aspect in Chapter 9, which concerns joint data and function analysis. For this reason we do not dwell on completeness further in this chapter.

Correctness. A schema is correct when it properly uses the concepts of the ER model. We may distinguish two types of correctness, syntactic and semantic. A schema is syntactically correct when concepts are properly defined in the schema; for example, subsets and generalizations are defined among entities but not among relationships. A schema is semantically correct when concepts (entities, relationships, etc.) are used according to their definitions. For example, it is a semantic mistake to use an attribute to represent products in a manufacturing company database when we need to represent several properties of products (e.g., product-code, price, parts, etc.), because an attribute is an elementary property. The following is a list of the most frequent semantic mistakes:

1. Using an attribute instead of an entity
2. Forgetting a generalization (or a subset)
3. Forgetting the inheritance property of generalizations
4. Using a relationship with a wrong number of entities (e.g., a binary relationship instead of a ternary relationship)
5. Using an entity instead of a relationship
6. Forgetting some identifier of an entity, especially external composite identifiers
7. Missing some min- or max-card specification

Minimality. A schema is minimal when every aspect of the requirements appears only once in the schema. We can also say that a schema is minimal if no concept can be deleted from the schema without losing some information. The schema in Figure 6.1 represents employees and projects on which they work. One of the attributes of the entity PROJECT is NUMBER_OF_EMPLOYEES, which can be derived from the schema simply by counting the employees related to the project. Therefore the schema is not minimal, and the attribute NUMBER_OF_EMPLOYEES can be deleted without changing the information content of the schema. Note that sometimes we prefer to allow some redundancy in the schema;

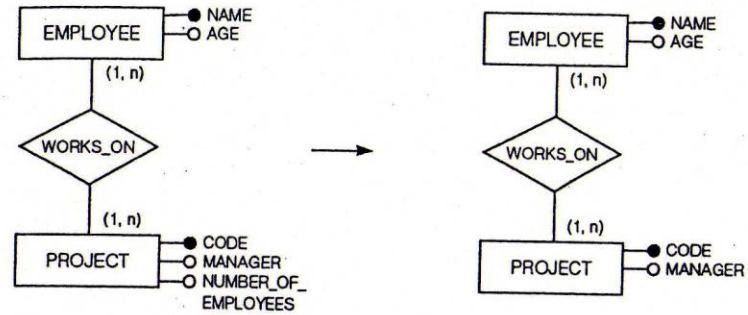


Figure 6.1 A redundant schema

however, redundancy should be documented. This is typically achieved by adding to the conceptual schema a table indicating how derived data are computed from the other data. Minimality will be addressed in Section 6.2. 6-3

Expressiveness. A schema is expressive when it represents requirements in a natural way and can be easily understood through the meaning of ER schema constructs, without the need for further explanation. As an example of expressiveness, consider the schema in Figure 6.2, which describes teaching and grading of courses and seminars. Expressiveness is improved by introducing the new entities TEACHING_STAFF (a generalization of the entities PROFESSOR and INSTRUCTOR) and OFFERINGS (generalization of entities COURSE and SEMINAR) and relating them by the single relationship TEACHES.

Readability. This is a property of the diagram that graphically represents the schema. A diagram has good readability when it respects certain aesthetic criteria that make the diagram graceful. The main criteria follow:

1. A diagram should be drawn in a grid, so that boxes representing entities and diamonds representing relationships have about the same size and connections run horizontally and vertically.
2. Symmetrical structures should be emphasized.
3. The global number of crossings is minimized (frequent crossings decrease the *bandwidth of perception* of the reader).
4. The global number of bends along connections should be minimized.
5. In generalization hierarchies, the father entity should be placed above the child entities, and children should be symmetrically placed with respect to the father. Similarly, in subset relationships the parent entity should be placed above and the subset entity below.

For a demonstration of readability, consider the schema in Figure 6.3. Readability is improved in Figure 6.3b by dropping crossings and emphasizing symmetry.

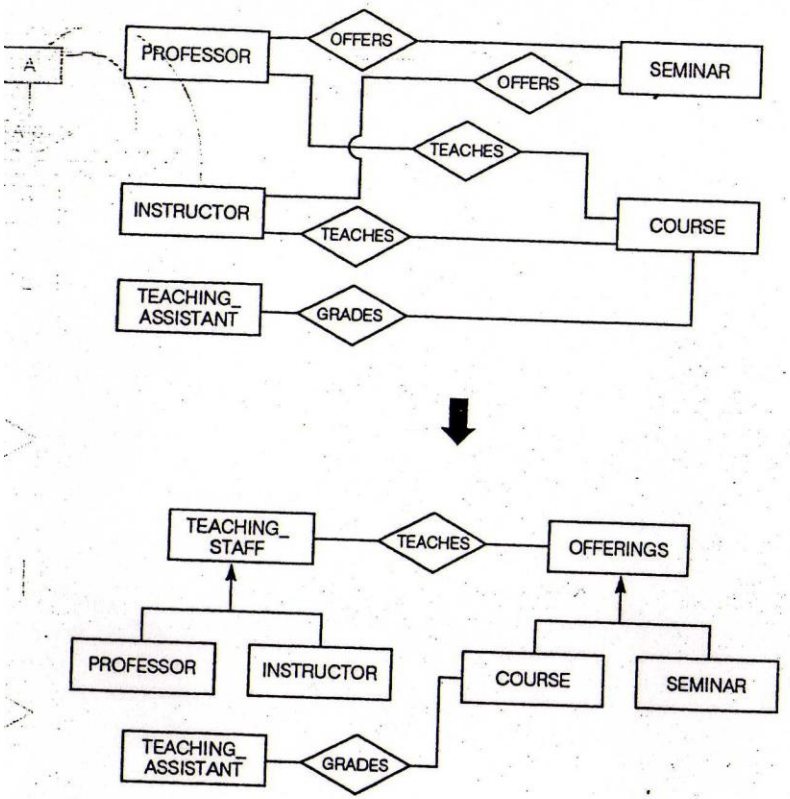


Figure 6.2 Improvement of expressiveness

Self-explanation. A schema is self-explanatory when a large number of properties can be represented using the conceptual model itself, without other formalisms (e.g., annotations in natural language). As an example of a schema that is not self-explanatory, let us represent students and their Master's and Ph.D. advisors. Assume that every student has at most one Master's advisor and one Ph.D. advisor and that the same student can (at different times) be both a Master's and Ph.D. student. This constraint cannot be fully represented in the schema of Figure 6.4a, because no concept in the model allows stating that "if a STUDENT object belongs to two instances of the HAS_ADVISOR relationship, then the TYPE attribute should take two distinct values." If instead we use two distinct relationships between students and professors (Figure 6.4b), then we may enforce the constraint by defining suitable minimal and maximum cardinalities of the relationships. Expressiveness and self-explanation are addressed in Section 6.3.

6.4

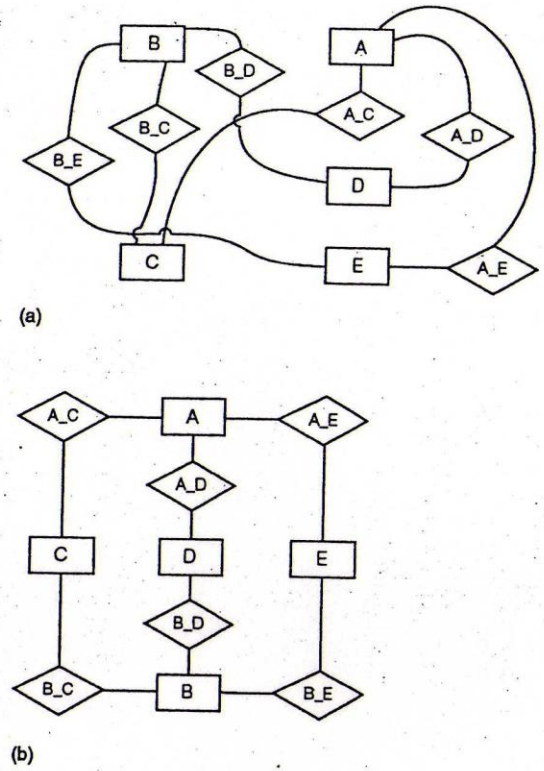


Figure 6.3 Improvement of readability

Extensibility. A schema is easily adapted to changing requirements when it can be decomposed into pieces (modules, or views), so that changes are applied within each piece. We will address extensibility in Chapter 7, where we define criteria for modularization and top-down documentation of the schema and use such concepts for schema maintenance.

Normality. The concept of normality here comes from the theory of normalization associated with the relational model. The *normal forms* (first, second, third, fourth, and a variation of third normal form called *Boyce-Codd* normal form) are intended to keep the logical structure of the data in a clean, "purified" normal form by alleviating the problems of insertion, deletion, and update anomalies, which cause unnecessary work because the same changes must be applied to a number of data instances, as well as the problem of accidental loss of data or the difficulty of representing given facts. For a full treatment of these normal forms and normalization, the reader should consult Chapters 13 and 14 of the Elmasri/Navathe text (see Bibliography of Chapter 1).

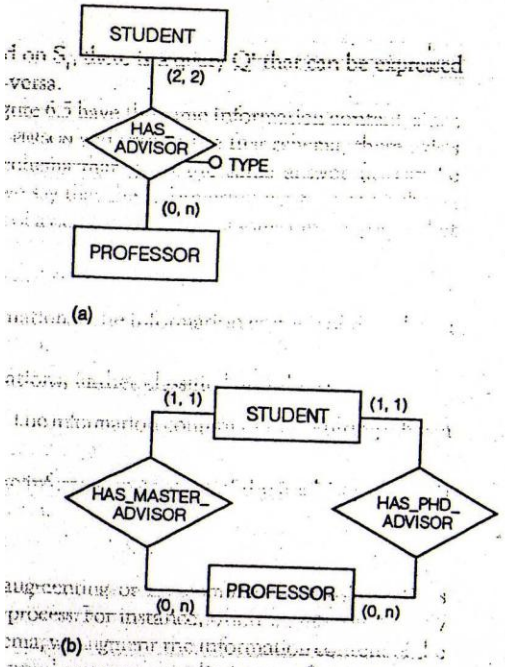


Figure 6.4 Example of self-explanatory relationships

Relational normalization theory uses the concepts of the key and functional dependency. We introduce these concepts in section 6.5 and then apply them in the ER model. Notice that we have implicitly captured most dependency information in the ER model in terms of the minimal and maximal cardinalities and of identifiers.

6.2 Schema Transformations

Schema transformations apply to an input schema S_1 and produce a resulting schema S_2. They can be classified according to the kind of quality improvement they produce. We initially discuss transformations in terms of how they change the *information content* of a schema. It is very difficult to formally define the information content of the schema or to prove that the information content of two schemas is identical. A means for comparing the information content of two schemas S_1 and S_2 is to compare their ability to reply to queries: We say that S_1 and S_2 have the same information content (or they are *equivalent*)

if for each query Q that can be expressed on S_1, there is a query Q' that can be expressed on S_2 giving the same answer, and vice versa.

For instance, the two schemas of Figure 6.5 have the same information content, since for each query that can be expressed on PERSON and CITY in the first schema, there exists a corresponding query on the second schema that gives the same answer (as can be checked intuitively). Correspondingly, we say that the information content of a Schema A is *greater* than the information content of a Schema B if there is some query Q on A that has no corresponding query on B, but not vice versa.

We are now able to classify transformations as follows:

1. **Information-preserving transformations:** The information content of the schema is not changed by the transformation.
2. **Information-changing transformations,** further classified as follows:
   a. **Augmenting transformations:** The information content of the resulting schema is greater than that of the input schema.
   b. **Reducing transformations:** The information content of the resulting schema is less than that of the input schema.
   c. **Noncomparable transformations,** otherwise.

So far, we have dealt mostly with augmenting or noncomparable transformations applied during the design and integration process. For instance, when we introduce a new interschema property in an integrated schema, we augment the information content of the schema; when we change a name into another name, to eliminate a homonym, we produce a new, noncomparable concept.

Quality checking makes frequent use of information-preserving transformations: the information content of the conceptual schema should not change, yet concept organization should improve. Reducing transformations are only used when the conceptual schema contains superfluous concepts that are not expressed in requirements and therefore should be eliminated. Table 6.1 summarizes the types of transformations most commonly used during conceptual design.

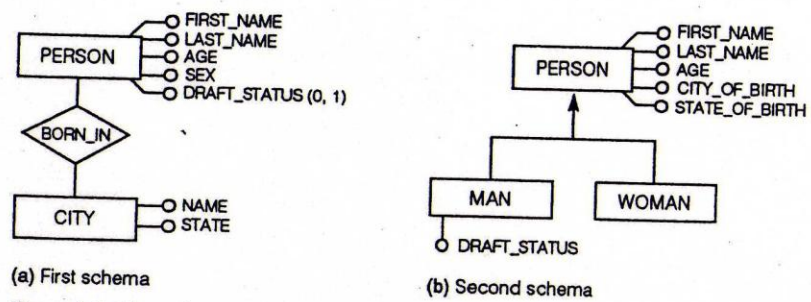


Figure 6.5 Two schemas with the same information content

Table 6.1 Schema Transformations Used During Conceptual Design

Transormation Type	When Used
Augmenting	Top-down design (Chapters 3–4)
Augmenting	Bottom-up design (Chapters 3–4)
Noncomparable	Conflict resolution (Chapter 5)
Augmenting	Addition of interschema properties (Chapter 5)
Preserving	Restructuring to achieve minimality, readability, and normality (Chapter 6)

6.3 Transformations for Achieving Minimality

Redundancy in ER schemas may arise for several reasons. One of the most common is that requirements have an inherent redundancy, and this redundancy often migrates into the schema. Also, in a bottom-up methodology, redundancy may arise when related concepts are expressed in different schemas; redundancy then appears when the schemas are merged. Redundancy of ER conceptual schemas is embedded in the following aspects of schemas.

Cycles of Relationships. Redundancy exists when one relationship R_1 between two entities has the same information content as a path of relationships $R_2, R_3, \ldots R_n$ connecting exactly the same pairs of entity instances as R_1. Obviously, not all cycles of relationships are sources of redundancy; as an example, consider the schema in Figure 6.6, representing employees, managers, and telephones. If the relationship HAS_TEL_2 connects each employee to his or her manager's telephone(s), then the schema is redundant; otherwise the schema is minimal. More generally, cycles of entities and relationships in a schema may or may not introduce redundancy, depending on their meaning.

Consider now the schema in Figure 6.7. The schema represents data about employees, directors, and departments; a director can be the manager of several departments, but each department has just one director. The relationships WORKS_WITH and HEADS are mutually redundant, whereas the relationship WORKS_IN is not redundant. When a cycle contains several relationships that are mutually redundant, it is possible to eliminate any one of them.

Even though redundancy in cycles of relationships depends on meaning, there are obvious syntactic checks that can be performed to give indications of redundancy. For instance, the relationship obtained by combining two or more one-to-one relationships is also one-to-one; therefore it cannot be equivalent to a one-to-many, a many-to-one, or a many-to-many relationship. Similarly, the relationship obtained by combining two or more one-to-many relationships is also one-to-many; therefore it cannot be equivalent to a one-to-one, a many-to-one, or a many-to-many relationship. In all other cases the cardinality of the combined relationship cannot be deduced; for instance, the combination of a one-to-many relationship with a many-to-one relationship may give rise to any

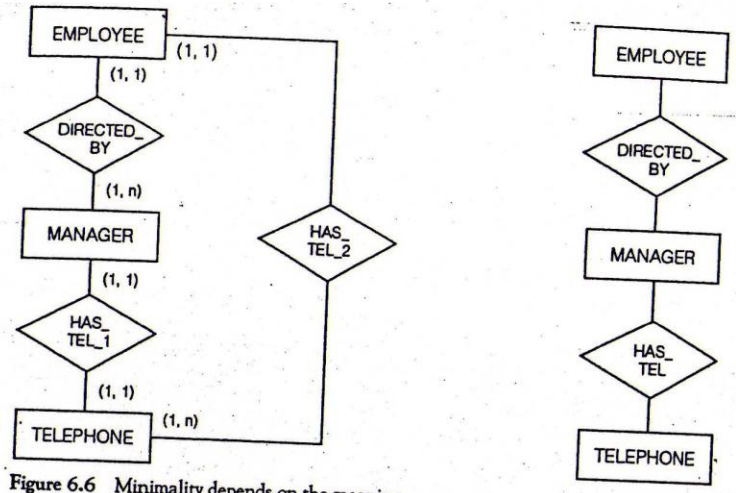
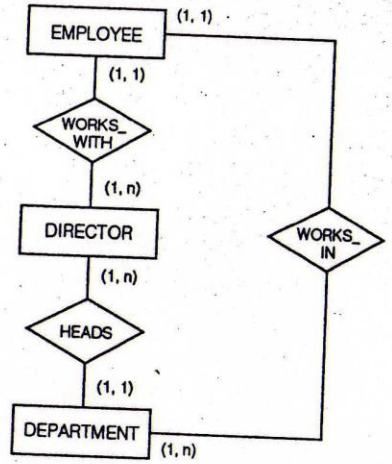


Figure 6.6 Minimality depends on the meaning



Relationship WORKS_WITH is equivalent to the path WORKS_IN, HEADS.
Relationship HEADS is equivalent to the path WORKS_WITH, WORKS_IN.

Figure 6.7 Example of redundant cycle of relationships

kind of relationship (it can be either one-to-one, one-to-many, many-to-one, or many-to-many).

Besides equivalence, cycles can force some kind of *containment constraints*. For instance, assume that STUDENT and COURSE are related by the binary relationship ENROLLED_IN and HAS_TAKEN_EXAM; then, although the relationships are not equivalent (and therefore the cycle is not redundant), we have the obvious constraint that a student cannot take an exam in a course in which he or she is not enrolled. Therefore the HAS_TAKEN_EXAM relationship is contained within the ENROLLED_IN relationship. This particular kind of redundancy can be avoided by including a Boolean attribute EXAM_TAKEN in the ENROLLED_IN relationship and dropping the HAS_TAKEN_EXAM relationship.

Derived Attributes. Redundancy may be due to the existence of an algorithm for computing the values of derived data from the other data; hence derived data can be omitted from a minimal ER schema. We have seen several examples of derived data in Chapter 4, in the income tax example. Derived data can be extremely useful for improving the efficiency of a database; this criterion will ultimately decide the convenience of storing derived data during logical design. We recommend including redundant derived data in the conceptual schema but indicating clearly the relevant rules for computation.

Implicit Subsets. After schema integration, some of the subsets might be derived from other subsets present in the schema. As an example, consider the two schemas in Figure 6.8: the subset among EMPLOYEE and ANALYST in the schema after integration is derivable; therefore we can eliminate the subset from the schema without changing its information content.

In summary, it is the designer's responsibility to decide whether to accept redundancy in the conceptual schema or to eliminate it. In any case, redundancy can be the source of anomalies in the management of data; hence, it has to be clearly indicated on the schema. We present a systematic discussion of dealing with redundancies in Chapter 11.

6.4 Transformations for Achieving Expressiveness and Self-Explanation

Self-explanation is achieved when properties of data are expressed using only the conceptual model itself instead of additional annotations; expressiveness is enhanced by simplifying the schema.

We describe here some typical transformations performed to improve expressiveness.

Elimination of Dangling Subentities in Generalization Hierarchies. It may happen that the designer creates a generalization in the process of assigning different properties to entities in the hierarchy. If, at the end of the design process, the subentities are not distinguished by any specific property, they can be reduced to the superentity. The distinction among the different entities is then expressed by means of an attribute. An

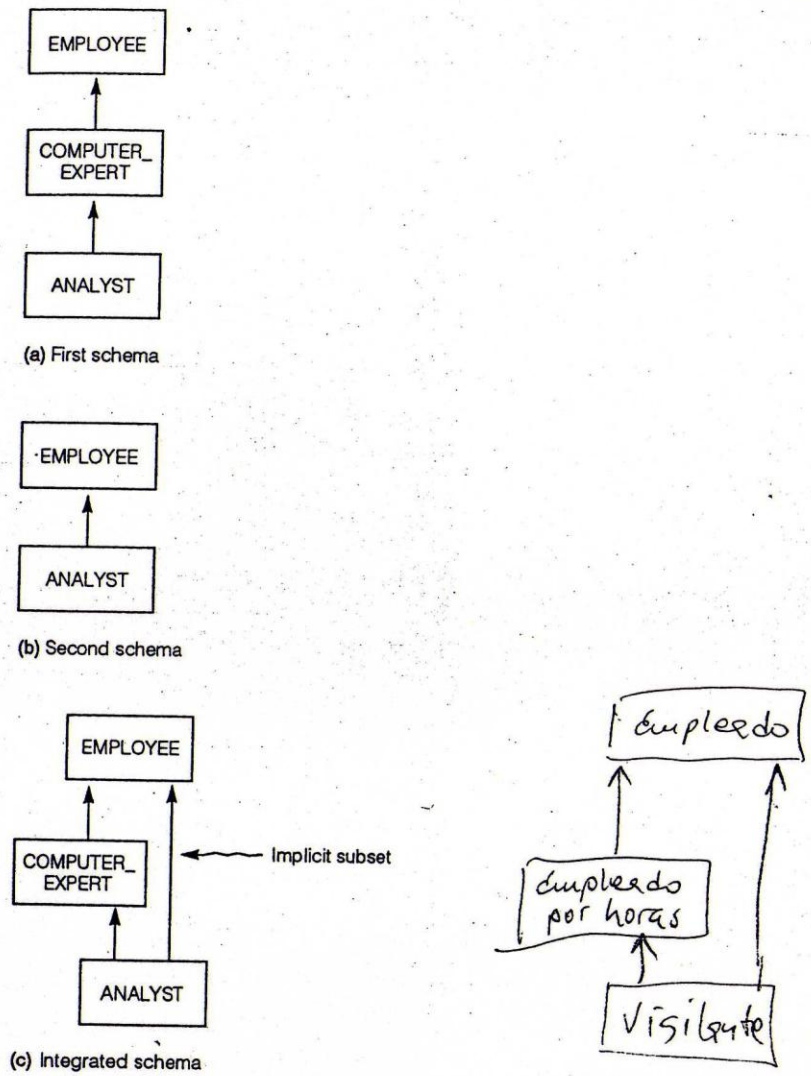


Figure 6.8 Example of an implicit subset

example of eliminating a subtype is shown in Figure 6.9; the values of the domain of the attribute RANK are (PROFESSOR, INSTRUCTOR, GRADUATE_STUDENT) or any encoding scheme that includes these options.

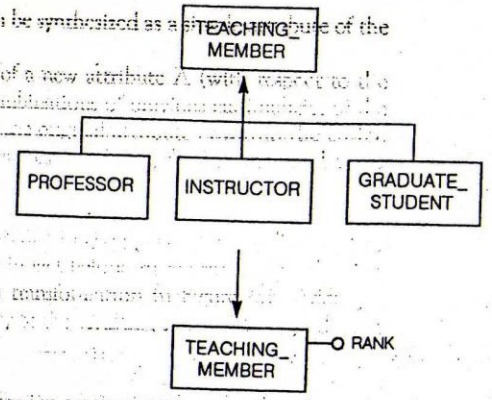


Figure 6.9 Elimination of a nonrepresentative generalization hierarchy

This transformation can be applied also when the subentities have only a few distinguishing properties; in this case the superentity acquires all the properties of the collapsed subentities. This transformation produces a simpler schema, and generates a typical trade-off situation between expressiveness and self-explanation: a compact schema is sometimes more understandable than a larger one; however, by collapsing entities, we may lose the possibility of being more precise in describing their properties.

Elimination of Dangling Entities. We regard an entity E as *dangling* if it has few (possibly one) attributes A_i and one connection to another entity (the *principal* entity) through one relationship R; in this case, it might be convenient to simplify the schema by eliminating both the dangling entity and the connecting relationship, passing the attributes A_i from the dangling entity to the principal entity. As in the previous case, dangling entities may have been generated during the process of assigning more properties to them. Figure 6.10 shows an example of this transformation: the concept CITY_OF_BIRTH has been

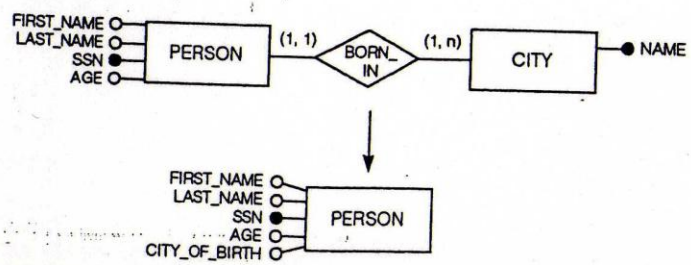


Figure 6.10 Elimination of a dangling entity

temporarily represented as an entity, but it can be synthesized as a simple attribute of the entity PERSON.

The minimal and maximal cardinalities of a new attribute A (with respect to the principal entity) can be easily computed as combinations of previous cardinalities of the dangling entity E in the relationship R and of the original attribute A within the entity. For example, if the max-card of E is 1 and the max-card of A is n, the max-card of the new attribute is n.

Creation of a Generalization. This transformation is applied when we discover two distinct entities with similar properties, which in fact belong to the same generalization hierarchy. We have seen an example of this transformation in Figure ~~6.1~~. Adding a generalization yields compactness and simplicity to the resulting schema through the use of the inheritance property. → 6.2 → no, hay más elem.

Creation of a New Subset. This transformation emphasizes the role of an entity. It can be applied to entities with a min-card equal to 0 in a relationship; this means that the relationship is applied to a subset of the entity instances. This transformation should be applied every time such a subset has a clear identity and is meaningful for the design. For example, in Figure 6.11 the entity EMPLOYEE may or may not be a car driver. This can be modeled by introducing a special subtype of employees, called DRIVER.

6.5 Transformations for Achieving Normalization

In the relational model, normalization of relations is a process of applying progressive transformations to achieve the intended normal form. The process is driven by *functional dependencies*. In this section we analyze how the concepts of functional dependency and normal forms can be applied to the ER model.

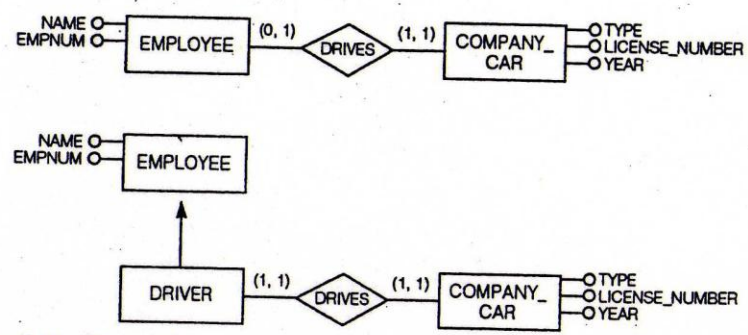


Figure 6.11 Creation of a new subset

6.5.1 *Functional Dependencies*

A functional dependency (FD) exists between two single-valued attributes A_1 and A_2 of an entity E or of a relationship R if each value of A_1 corresponds to precisely one value of A_2. This property is better discussed by giving examples of instances. Let A_1, A_2 be two attributes of an entity E; assume that there exists an entity instance e_1 of E having values a_1 for A_1 and a_2 for A_2:

$$<e_1: \ldots a_1 \ldots a_2 \ldots>$$

The functional dependency between A_1 and A_2 implies that if another entity instance e_2 exists, with A_1 taking value a_1, then A_2 must take value a_2:

$$<e_2: \ldots a_1 \ldots a_2 \ldots>$$

We say that A_1 functionally determines A_2, also denoted as $A_1 \rightarrow A_2$; the attribute on the left side of the FD is called the **determinant.** FDs are similarly established between sets of attributes; for instance, $A_1, A_2 \rightarrow A_3$ (the pair of attributes $[A_1, A_2]$ are in this case the determinant), or $A_1 \rightarrow A_2, A_3$. When the right side of an FD is a set S of *n* attributes, then the original FD is equivalent to *n* individual FDs, each one having a single attribute of S as the right side. For instance, the dependency $A_1 \rightarrow A_2, A_3$ is equivalent to the two FDs $A_1 \rightarrow A_2$ and $A_1 \rightarrow A_3$ separately. Functional dependencies are also established between attributes of relationships, with exactly the same interpretation.

The careful reader should be aware at this point that *in a correct schema* all internal identifiers of entities functionally determine the other single-valued attributes. The example of Figure 6.12 shows the entity PERSON with two internal identifiers: SOC_SEC_NUM, and the pair (NAME, BIRTH_DATE). Other attributes are ADDRESS, CITY, STATE, and ZIP_CODE. Thus, we have the following FDs:

SOC_SEC_NUM → NAME, BIRTH_DATE, ADDRESS, CITY, STATE, ZIP_CODE

NAME, BIRTH_DATE → ADDRESS, CITY, SOC_SEC_NUM, STATE, ZIP_CODE

These dependencies tell us that if we assign the value of determinant attributes, then we find in the database one value of determined attributes; this follows trivially from the fact that determinants are also identifiers. We also have one additional FD,

ZIP_CODE → CITY, STATE

expressing the property that persons having the same ZIP_CODE live in the same CITY and STATE.

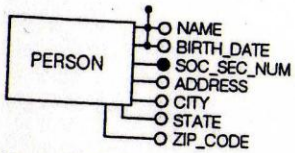


Figure 6.12 Entity PERSON

6.5.2 *Update Anomalies*

While the dependencies corresponding to identifiers do not cause problems, the other dependencies that may exist in an entity may cause what are called *update anomalies*, which we informally present in this section. Consider the example in Figure 6.13, describing ORDER in terms of order numbers, ordered parts, date of order, cost of each part, and quantity ordered of each part. Figure 6.13b shows some entity instances.

The identifier of the entity is given by the pair (ORD_NUM, PART_NAME); hence we deduce the following dependency:

ORD_NUM, PART_NAME → COST, QUANTITY, DATE

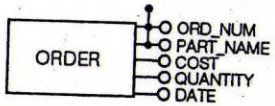
However, the cost of one part is uniquely determined by its part number; this is asserted by the following functional dependency:

PART_NAME → COST

This is the source of redundancy shown in the entity instances; for example, the cost of pencils is repeated three times. Further, we recognize the following anomalies:

1. **Insertion anomaly:** We cannot indicate the cost of a part unless it has some pending orders.
2. **Deletion anomaly:** When we delete the last pending order for a part, we also delete the information about its cost.
3. **Update anomaly:** If the cost of some parts changes, all orders referring to those parts also change; the update operation is propagated to several entity instances.

These anomalies are all related to the presence of an *undesired* functional dependency, namely, PART_NAME → COST. A similar anomaly is due to the dependency ORD_NUM → DATE.



(a) The entity ORDER

⟨01 : 1518, PEN, 1, 12, 3-8-90⟩
⟨02 : 1518, PENCIL, 0.5, 15, 3-8-90⟩
⟨03 : 1521, PENCIL, 0.5, 18, 2-9-89⟩
⟨04 : 1407, PEN, 1, 15, 2-6-89⟩
⟨05 : 1407, ERASER, 0.2, 28, 2-6-89⟩
⟨06 : 1407, BOARD, 5, 3, 2-6-89⟩
⟨07 : 1729, ERASER, 0.2, 1, 3-1-90⟩
⟨08 : 1729, DISKETTE, 2, 10, 3-1-90⟩
⟨09 : 1729, PENCIL, 0.5, 15, 3-1-90⟩

(b) Instances of the entity ORDER

Figure 6.13 Update anomalies

The process of normalization is a progressive detection and elimination of such undesired dependencies.

6.5.3 *Relevant Attributes for Entities and Relationships*

The main difficulties in extending the treatment of normalization from the relational to the ER model arise from the different means of expressing identification. External identifiers of an entity provide functional dependencies among entity attributes. Similarly, given a relationship among several entities, entity identifiers provide functional dependencies among attributes of the relationship. For these reasons, we need to define, for each entity or relationship, the set of attributes that are *relevant* to the normalization process; this set contains the original attributes of the entity or relationship, but it is usually larger. We start by considering external identification.

Relevant Attributes for an Entity. Let us consider an entity E; for each external identification from an entity E_1, we incorporate in the set of relevant attributes of E one internal identifier of E_1. This process is iterated if in turn E_1 has external identifiers. For instance, consider the entity ORDER_DETAIL in Figure 6.14a, with attributes LINE_NUM, QUANTITY, UNIT_PRICE and the external identifiers provided by the entity ORDER. According to the above definition, the relevant attributes of ORDER_DETAIL are ORD_NUM, LINE_NUM, QUANTITY, and UNIT_PRICE

Note that, as an effect of including ORD_NUM among the relevant attributes, we can now consider the pair (ORD_NUM, LINE_NUM) as an identifier of the entity ORDER.

In the rest of this chapter we will simply denote as identifier of an entity any subset of relevant attributes that uniquely determines entity instances and disregard the distinction between external and internal identification.

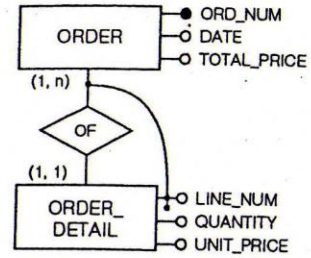
Relevant Attributes for a Relationship. Let R be a binary relationship among entities E_1 and E_2; let m_1 and m_2 denote the max-cards of E_1 and E_2 respectively in R. We consider three cases:

1. *One-to-one relationship* ($m_1 = 1$ and $m_2 = 1$). We incorporate into relevant attributes of R one identifier arbitrarily selected either from E_1 or from E_2.
2. *One-to-many relationship* ($m_1 = 1$ and $m_2 = n$). We incorporate into relevant attributes of R one identifier of E_1.
3. *Many-to-many-relationship* ($m_1 = n$ and $m_2 = n$). We incorporate into relevant attributes of R one identifier of both E_1 and E_2.

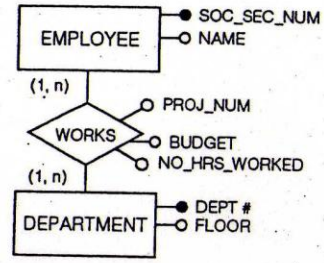
These rules extend easily to n-ary relationships.

Consider the example of Figure 6.14b, describing entities EMPLOYEE and DEPARTMENT and the many-to-many relationship WORKS between them. In this case, identifiers SOC_SEC_NUM for EMPLOYEE and DEPT# for DEPARTMENT are incorporated into relevant attributes of WORKS, which are: SOC_SEC_NUM, DEPT#, PROJ_NUM, BUDGET, and NO_HRS_WORKED.

We now have the appropriate tools for applying normalization to extended attributes of entities and relationships. In the relational theory of normalization, several progressive normal forms are defined (first, second, third, Boyce-Codd, fourth, and others). The first



(a) Relevant attributes for an entity



(b) Relevant attributes for a relationship

Figure 6.14 Relevant attributes

normal form is automatically achieved by any *flat* relation; that is, one without any nested entries and one where every value is atomic, implying that repeating values or lists are not allowed. Each subsequent normalization enforces the elimination of certain types of anomalies. We deal with the first normal form in Chapter 12 in the discussion of mapping from the ER model to the relational model; in particular, we deal with multivalued attributes, having max-card > 1. In this chapter we restrict our attention to single-valued attributes. Each entity or relationship, properly extended with relevant attributes and considered as isolated from the rest of the schema, is structurally equivalent to a flat relation and can be normalized.

6.5.4 *Second Normal Form*

Before introducing second normal form, we need to define prime attributes: An attribute is not **prime (nonprime)** if it does not belong to any identifier; it is **prime** otherwise. An entity or relationship is in **second normal form** if no FD exists whose determinant is properly contained in an identifier and whose right-side attribute is not prime. As an example of an entity that violates the second normal form, consider the entity ORDER in Figure 6.15a, in which PART_NUM replaces PART_NAME from Figure 6.13. Here, the pair

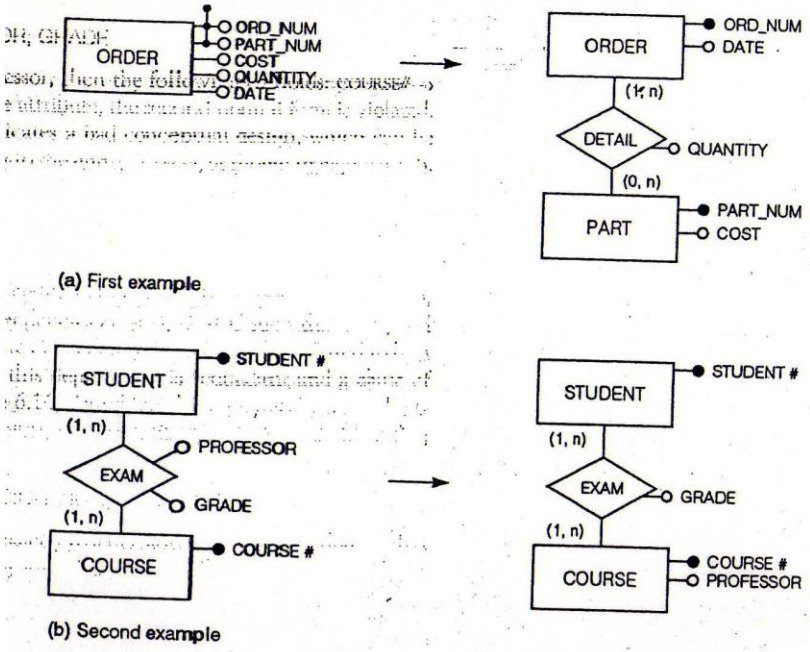


Figure 6.15 Achieving second normal form

ORD_NUM, PART_NUM) is the only identifier. COST, QUANTITY, and DATE are nonprime attributes. The following FDs are defined:

d_1: ORD_NUM, PART_NUM → QUANTITY
d_2: PART_NUM → COST
d_3: ORD_NUM → DATE

he entity ORDER is not in second normal form because dependencies d_2 and d_3 have a eterminant that is properly contained in the identifier and COST and DATE are nonprime.

The violation of second normal form indicates a bad conceptual design; in fact, we ave represented in the entity ORDER three different concepts: orders, parts, and the relaonship among them. We achieve second normal form by introducing exactly one oncept of the schema for each object of the application domain. The resulting schema is own on the right in Figure 6.15a. The introduction of an entity for PART causes the troduction of a many-to-many relationship DETAIL between ORDER and PART.

An example of second normal form for relationships is shown in Figure 6.15b. In this se, we have the two entities STUDENT and COURSE and one relationship EXAM between them; evant attributes for EXAM are STUDENT#, COURSE#, PROFESSOR, and GRADE. The pair TUDENT#, COURSE#) is the only identifier. Hence,

STUDENT#, COURSE# → PROFESSOR, GRADE

However, if each course has just one professor, then the following FD holds: COURSE# → PROFESSOR. Because PROFESSOR is a nonprime attribute, the second normal form is violated. Again, the violation of normalization indicates a bad conceptual design, which can be improved by moving the attribute PROFESSOR to the entity COURSE, as shown in Figure 6.15b.

6.5.5 *Third Normal Form*

Before introducing third normal form, we need to define transitive dependencies. An FD, A → C, is **transitive** if there exist two dependencies A → B, B → C such that A, B, and C are different groups of attributes. Then the dependency A → C can be inferred to be a combination of A → B, B → C; as such, this dependency is redundant and a cause of anomalies. Consider the example of Figure 6.16, describing the entity EMPLOYEE with attributes EMPNUM, NAME, DEPT_NUM, DIVISION_NUM, MANAGER. EMPNUM is the only identifier; hence we deduce the FD

EMPNUM → NAME, DEPT_NUM, DIVISION_NUM, MANAGER

We observe that each department belongs exactly to a division and that each division has one manager; hence we know the following FDs exist:

d_1: DEPT_NUM → DIVISION_NUM
d_2: DIVISION_NUM → MANAGER

We then deduce several transitive dependencies:

EMPNUM → DIVISION_NUM, MANAGER
DEPT_NUM → MANAGER

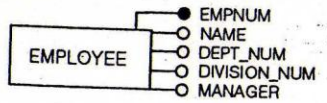
An entity or relationship is in **third normal form** if it is in second normal form and has no transitive FD. Clearly, the entity EMPLOYEE of the above example violates third normal form. Once again, violation of normalization indicates a bad design, since the entity EMPLOYEE incorporates the concepts of DEPARTMENT and DIVISION. The progressive normalization of EMPLOYEE is carried out in Figures 6.16b and 6.16c. We introduce two new entities, each one corresponding to a different real-world concept; in this way we eliminate transitive dependencies.

As another example of a relationship that violates third normal form, consider the relationship WORKS of Figure 6.17. Since the pair (SS#, DEPT#) is the identifier of works, we deduce the following dependencies:

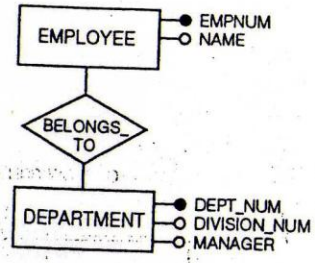
d_1: SS#, DEPT# → BUDGET
d_2: SS#, DEPT# → PROJ_NUM
d_3: SS#, DEPT# → NO_HRS_WORKED

However, each project (with its associated PROJ_NUM) has a unique budget; hence we deduce the dependency

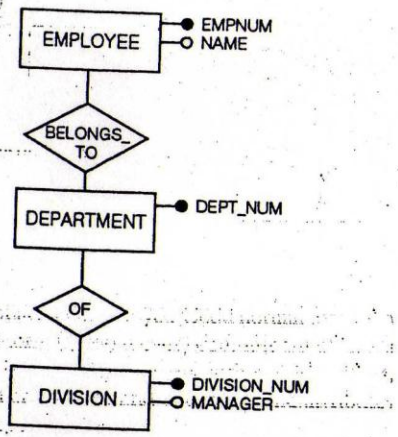
d_4: PROJ_NUM → BUDGET



(a) First schema



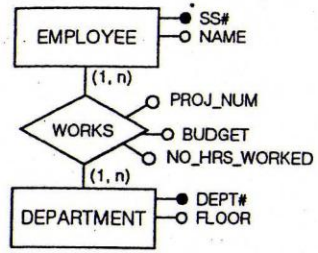
(b) Second schema



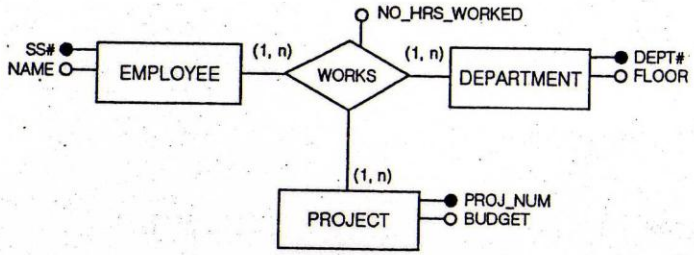
(c) Third schema

Figure 6.16 Achieving third normal form: first case

Therefore, dependency d_1 is transitive, and the relationship is not in third normal form. We may achieve third normal form by introducing the new entity PROJECT and transforming the relationship WORKS into a ternary relationship (see Figure 6.17b). Notice that once again normalization is achieved by introducing a separate concept for each object of the application domain.



(a) First schema



(b) Second schema

Figure 6.17 Achieving third normal form: second case

6.5.6 *Boyce-Codd Normal Form*

Another well-known normal form in the relational model is Boyce-Codd normal form. An entity or relationship is in **Boyce-Codd normal form** (BCNF) if every determinant of its FDs is an identifier. An example of a BCNF relation is the entity STUDENT with two identifiers, social security number (SS#) and student number (STUD#), and two nonprime attributes NAME and ADDRESS. In this case we have the dependencies

d_1: SS# → STUD#, NAME, ADDRESS
d_2: STUD# → SS#, NAME, ADDRESS

These FDs do not violate BCNF.

Consider now the schema in Figure 6.18, representing students, courses, and professors; assume that every professor teaches exactly one course, that the same course can be taught by several professors, and that each student attends a given course with a unique professor. The above constraints can be expressed in terms of the following FDs:

d_1: STUDENT, COURSE# → PROFESSOR
d_2: PROFESSOR → COURSE#

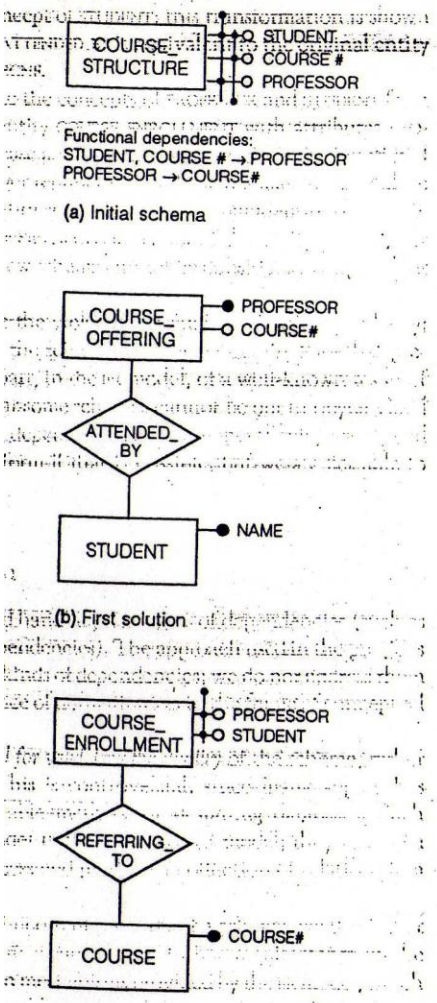


Figure 6.18 Trying to achieve Boyce-Codd normal form

The entity COURSE_STRUCTURE has two alternative identifiers: the pair (STUDENT, COURSE#) and the pair (STUDENT, PROFESSOR). Then the FD d_2 violates BCNF, since PROFESSOR is a determinant but not an identifier.

Since the violation to BCNF follows from the fact that professor is not a determinant, we might try to separate the concept of PROFESSOR and the associated COURSE#, together

referred to as a course offering, from the concept of STUDENT; this transformation is shown in Figure 6.18b. However, the relationship ATTENDED_BY is equivalent to the original entity COURSE_STRUCTURE and so in turn violates BCNF.

As a second solution, we could separate the concepts of PROFESSOR and STUDENT from the concept of COURSE by introducing an entity COURSE_ENROLLMENT with attributes PROFESSOR and STUDENT (Figure 6.18c). In this case a new problem arises, since the functional dependency PROFESSOR → COURSE# is no longer represented in the schema. We can deduce it from neither the internal entities and relationships, nor the external identifiers. Indeed, d_2 is neither a dependency of the entity COURSE_ENROLLMENT, nor of the entity COURSE, nor of the relationship REFERRING_TO. Thus the new schema cannot be considered as equivalent to the old one, since a dependency is lost.

All possible solutions either reproduce the violation or the loss of one dependency; therefore, we have no way of transforming the schema into a new one in BCNF that preserves the dependency. This is the counterpart, in the ER model, of a well-known result of relational normalization theory, namely, that some schemas cannot be put in Boyce-Codd normal form and still preserve functional dependencies. As a general rule, we should always try to achieve Boyce-Codd normal form if that is possible, but we are *not* able to achieve it in all cases.

6.5.7 *Reasoning about Normalization*

Further normal forms require identifying and handling new types of dependencies (such as multivalued, domain, join, and inclusion dependencies). The approach used in the previous sections can be extended to deal with these kinds of dependencies; we do not address them in this book. Rather, we evaluate the relevance of normalization in the frame of conceptual database design.

In our approach, normalization is *a tool for validating the quality* of the schema, rather than *a method for designing* the schema. This is controversial, since many approaches propose normalization as the only acceptable method for designing databases. Such approaches suggest using the relational model itself as the design model; they assume a collection of FDs as input to the design process and produce a collection of relations in a given normal form as a result.

While we believe that normality is an important property of a schema, we think that dependencies are inappropriate and obscure means for capturing requirements in the application domain. We prefer the abstraction mechanisms provided by the ER model, which are easy to understand and can be represented by means of diagrams naturally.

At the same time, we stress that the ER model and the design methods described in Chapter 3 *tend naturally to produce normalized schemas*. The objective of normalization up to Boyce-Codd normal form is to keep each functional dependency separated by associating to each set of homogeneous FDs an element of the model (entity or relationship) that has the determinants of the FDs as identifiers. Thus, each concept of the application domain is mapped to exactly one concept of the schema.

This *concept separation* is the natural result of using classification, aggregation, and generalization abstractions and applying a top-down methodology for producing a schema via subsequent refinements. In all examples that violate normalization, we have noticed

that the corresponding schema was badly designed. This is consistent with our view of normalization as a validation tool.

6.6 An Example of Schema Restructuring

Let us consider again the schema obtained at the end of Chapter 5 (Figure 5.8) as a result of the integration activity and check its quality.

Considering minimality first, we discover a redundant cycle of relationships (see Figure 6.19): each book belongs to a series of a given publishing company; hence we can deduce the relationship PUBLISHED_BY between BOOK and PUBLISHING_COMPANY as the combination of the two relationships BELONGS_TO and OF. Keeping PUBLISHED_BY in the schema depends on its importance in the application: we have here a trade-off between expressiveness and minimality. In the absence of strong arguments, we can either eliminate the source of redundancy or simply indicate it in the conceptual schema, postponing the decision to the logical design phase (see Chapter 11).

We then consider expressiveness. We may apply two transformations. First, consider the schema fragment in Figure 6.20. We note that BOOK and PAPER share the same relationships with AUTHOR and SCIENTIST. The global expressiveness is improved by introducing a more general concept than BOOK and PAPER, called AUTHORED_WORK, that inherits the above relationships. Note that in this way BOOK is a subset entity of both PUBLICATION and AUTHORED_WORK (technically this situation is often called *multiple inheritance*). This transformation enhances self-explanation as well: in the previous schema, the participation of the entity AUTHOR was optional in both the relationship with entity BOOK and the relationship with entity PAPER; however, an additional constraint was needed, stating that every author had to be connected either to a book or to a paper. This constraint was not explicitly expressed in the schema. After the restructuring, we change to 1 the min-card value of the entity AUTHOR in the relationship WRITTEN_BY.

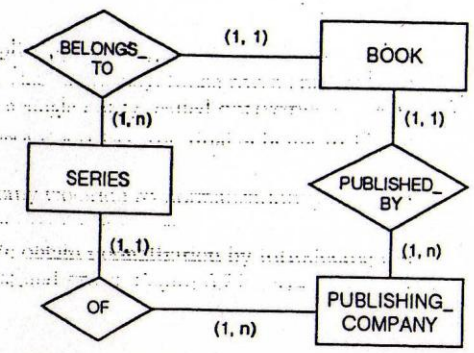


Figure 6.19 Redundancy in the example of Figure 5.8

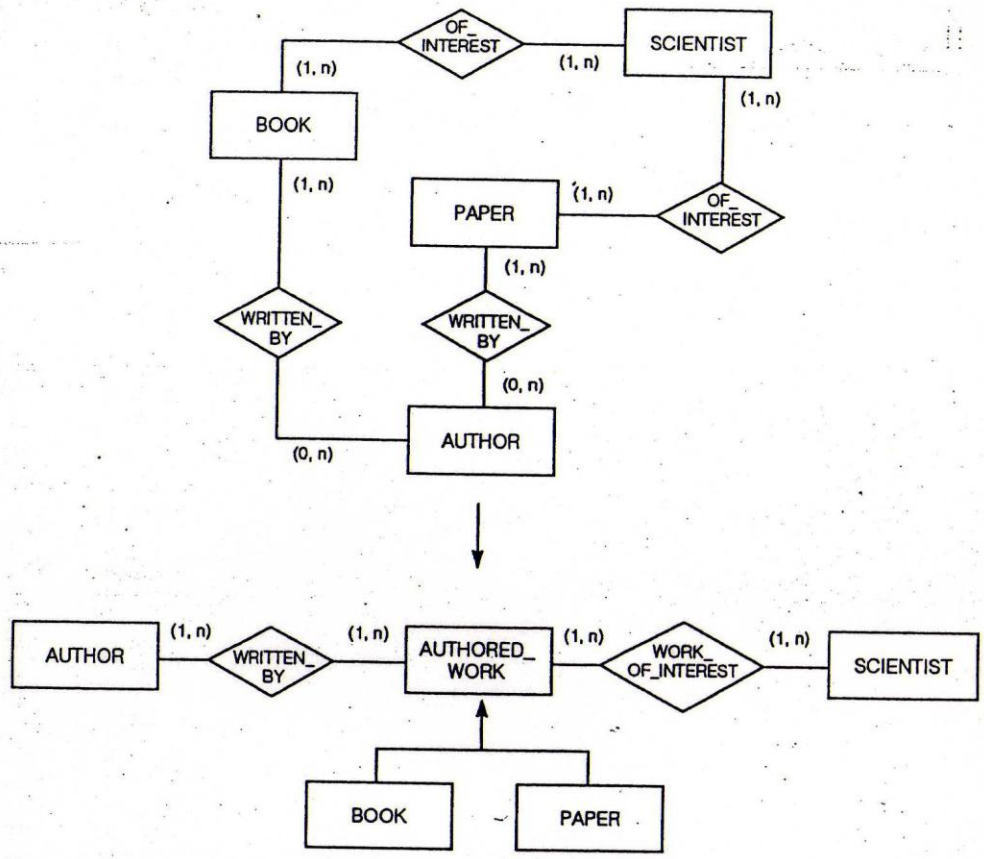
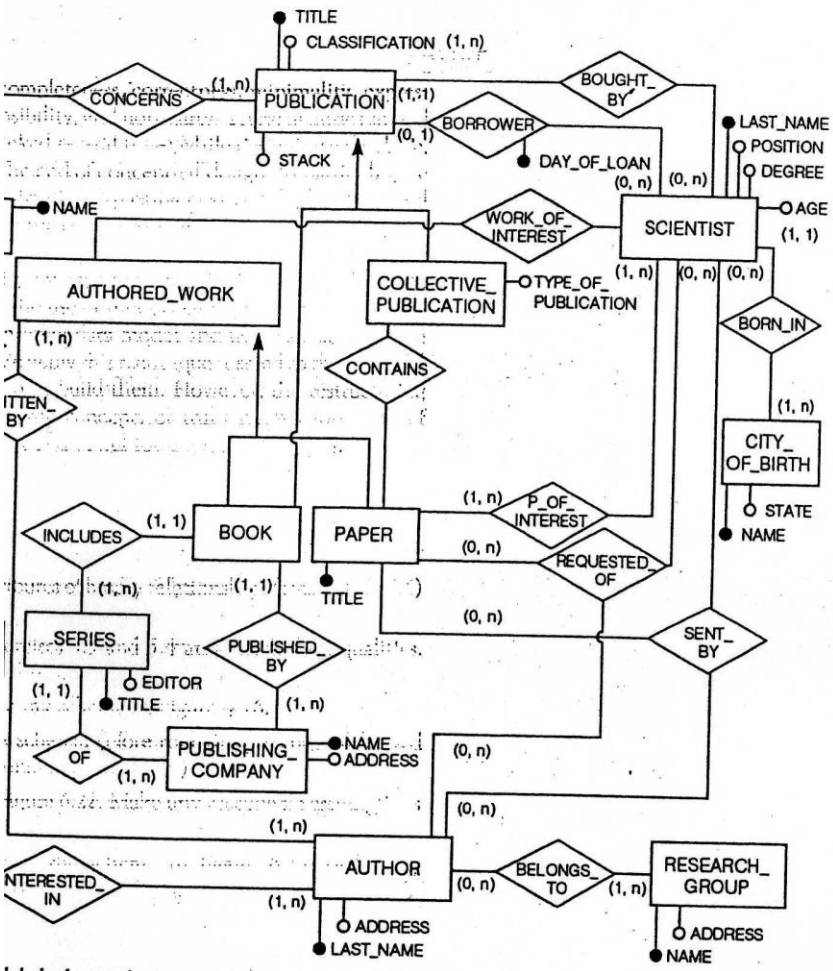


Figure 6.20 Improving expressiveness and self-explanation in the example

Another restructuring action can be applied to the fragment of the schema that includes the entities JOURNAL and PROCEEDINGS (see Figure 5.8). Since these entities have the same properties, they can be merged into a single entity, called COLLECTIVE_PUBLICATION, with the additional attribute TYPE_OF_PUBLICATION (see the final schema in Figure 6.21).

Finally, we consider normalization. The only violation to normalization is found in the entity SCIENTIST: if we assume, as usual, that CITY_OF_BIRTH determines STATE_OF_BIRTH, then the entity is not in third normal form. We obtain normalization by introducing the new entity CITY_OF_BIRTH, with attributes NAME and STATE. Figure 6.21 shows the final conceptual schema after restructuring.



lobal schema after restructuring

6.7 Summary

The qualities of a conceptual schema are completeness, correctness, minimality, expressiveness, readability, self-explanation, extensibility, and normality. These qualities should drive the design process; they should be checked several times while designing a database, and they should be carefully considered at the end of conceptual design. In particular, we present several techniques for improving minimality, expressiveness, self-explanation, and normality; these techniques are based on the use of transformations that must be applied to schemas.

Transformations for achieving normality are applied to entities in order to eliminate update anomalies. This process is based on the use of functional dependencies. Normalization was originally developed for the relational data model and used as a stand-alone technique for building relational schemas. We apply this technique instead to the ER model and use it to restructure schemas rather than to build them. However, the restructuring operations that focus on normalization and the concepts of third normal form and of Boyce-Codd normal form are quite important and useful for database designers.

Exercises

6.1. Extend the rules for finding relevant attributes of binary relationships (see Section 6.6) to n-ary relationships.

6.2. Consider the schemas produced for Exercises 5.3 and 5.4 and check their qualities. Write an evaluation for each of the qualities.

6.3. Check the expressiveness of the income tax schema in Figure 4.16.

6.4. Check the normality of the income tax schema. (Note that the presence of derived data produces a huge number of functional dependencies.)

6.5. Check the normality of the schema in Figure 6.22. Make any reasonable assumptions you need to make and state them.

6.6. Check the redundancy and normality of the schema in Figure 6.23. Make any reasonable assumptions you need to make and state them.

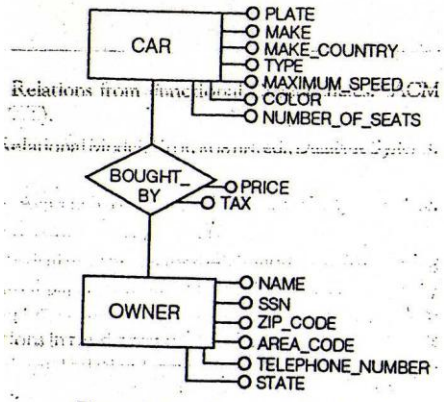
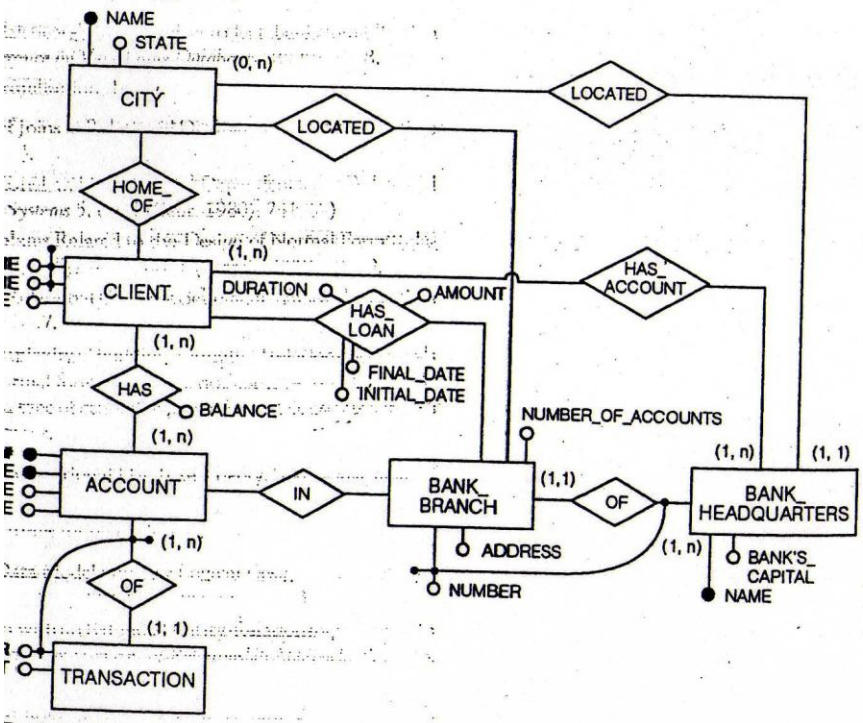


Figure 6.22 Car-ownership database



Bank database

Annotated Bibliography

P. BERNSTEIN. "Synthesizing Third Normal Form Relations from Functional Dependencies." *ACM Transactions on Database Systems* 1, no. 4 (1976).

E. F. CODD. "Further Normalization of the Database Relational Model." In R. RUSTIN, ed., *Database Systems*. Prentice-Hall, 1972.

R. FAGIN. "The Decomposition versus the Synthetic Approach to Relational Database Design." In *Proc. Third International Conference on Very Large Databases*. Tokyo. 1977.

These papers are mentioned here both for historical reasons and for their importance in providing a formal treatment of normal forms. CODD's seminal paper introduces normal forms. The paper by BERNSTEIN describes a synthesis procedure. Starting from a set of attributes and functional dependencies, it provides an algorithm that synthesizes relations in third normal form. The paper by FAGIN starts from the same input and decomposes an initial universal relation formed by all attributes. Decomposition rules are driven by functional dependencies.

Both BERNSTEIN's and FAGIN's approaches can be classified as bottom-up design strategies, since they assume as a starting point the entire set of attributes and functional dependencies. A long debate has been focused on the pros and cons of conceptual modeling versus the use of normalization on top of the relational model. Experience in database design for large databases has shown that conceptual modeling is in fact much more effective; many modern design methodologies, including the one in this book, use normalization as a verification technique for conceptual schemas.

C. BEERI, P. A. BERNSTEIN, and N. GOODMAN. "A Sophisticate's Introduction to Database Normalization Theory." In *Proc. Fourth International Conference on Very Large Databases*. Berlin. 1978.

This paper is a comprehensive survey of normalization theory.

A. V. AHO, C. BEERI, and J. D. ULLMAN. "The Theory of Joins in Relational Databases." *ACM Transactions on Database Systems* 4, no. 3, (1979): 297–314.

C. BEERI. "On the Membership Problem for Functional and Multivalued Dependencies in Relational Databases." *ACM Transactions on Database Systems* 5, no. 3 (Sept. 1980): 241–59)

C. BEERI and P. A. BERNSTEIN. "Computational Problems Related to the Design of Normal Form Relational Schemas." *ACM Transactions on Database Systems* 4, no. 1 (March 1979): 30–59.

C. ZANIOLO and M. A. MELKANOFF. "On the Design of Relational Database Schemata." *ACM Transactions on Database Systems* 6, no. 1 (March 1981): 1–47.

These papers describe the computational complexity of building normalized relations. BEERI deals with multivalued dependencies and the fourth normal form, which are not considered in this book. AHO, BEERI, and ULLMAN study join dependencies, a type of general dependency not considered in this book, and the lossless join property of decompositions.

W. KENT. "A Simple Guide to Five Normal Forms in Relational Database Theory." *Communications of the ACM* 26, no. 2 (Feb. 1983): 120–25.

This paper gives an introduction to normal forms for practitioners.

R. BROWN and D. S. PARKER. "LAURA: A Formal Data Model and Her Logical Design Methodology." In *Proc. Ninth International Conference on Very Large Databases*. Florence. 1983.

D. EMBLEY and T. LING. "Synergistic Database Design with an Extended Entity-Relationship Model." In F. LOCHOVSKY, ed., *Proc. Eighth International Conference on Entity-Relationship Approach*. Toronto. North-Holland, 1988.

These papers present an approach to normalization similar to the one followed in Section 6.6. The ER schema, instead of being transformed to a relational schema and then normalized, is first