Nithya BN
Assistant Professor
Department of Computer Applications
M S Ramaiah Institute of Technology
nithyabn@msrit.edu
Ph:9900087291

MCA23: Database Systems: Unit III

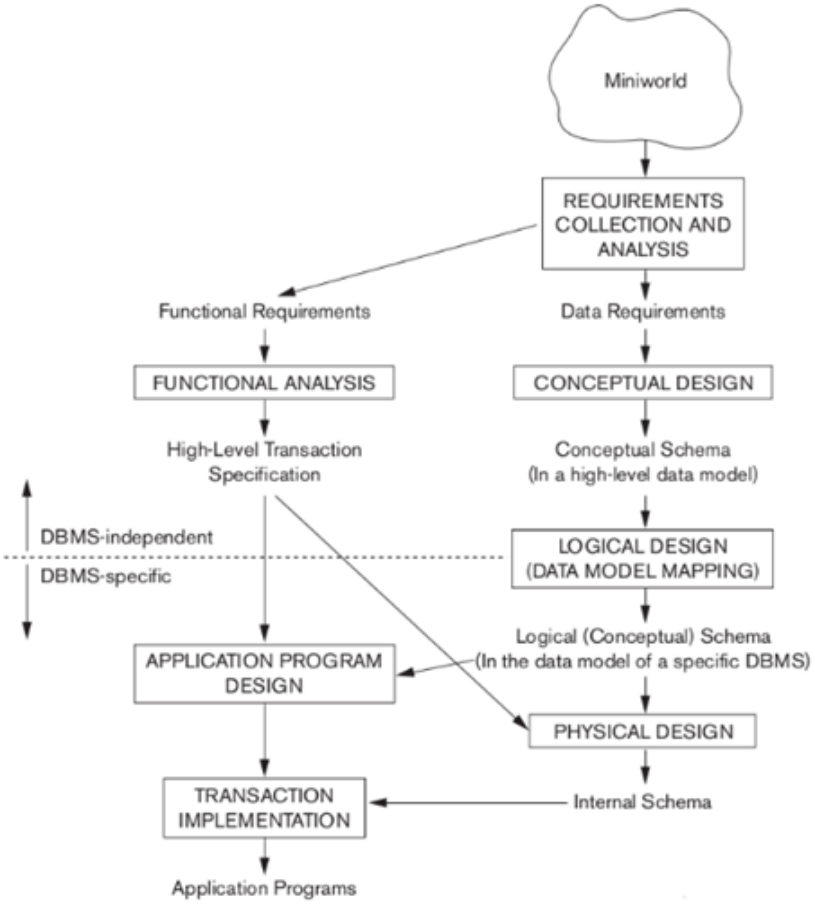
Contents

UNIT III

Entity - Relationship Model Using High-level Conceptual Data Models for Database Design, Entity Types, Entity Sets, Attributes and Keys, Relationship Types, Relationship Sets, Roles and Structural Constraints, Weak Entity Types, Refining the ER Design, ER Diagrams.

Database Design Functional Dependencies, Normal Forms Based on Primary Keys

1 Using High-Level Conceptual Data Models for Database Design



In the above simplified overview of the database design process, the first step shown is requirement collection and analysis. During this step, the database designer interview prospective database user to understand and document their data requirements. The result of this step is a concisely written set of user's requirements. These requirements should be specified in a detailed and complete form as possible.

In parallel with specifying the data requirements, it is useful to specify the known functional requirements of the application. These consist of the user defined operations (or transactions) that will be applied to the database, including both retrievals and updates. In software design, it is common to use data flow diagrams, sequence diagrams, scenarios and other techniques to specify functional requirements.

Once the requirements have been collected and analyzed, the next step is to create a conceptual schema for the database, using a high – level conceptual data model. This step is called conceptual design. The conceptual schema is a concise description of the data requirements of the users and includes detailed descriptions of the entity types, relationships and constraints; these are expressed using the concepts provided by the high level data model. The high level conceptual schema can also be used as a reference to ensure that all user data requirements are met and that the requirements do not conflict. This approach enables database designers to concentrate on specifying the properties of the data, without being concerned with storage and implementation details, which make it easier to create a good conceptual database design.

The next step in database design is the actual implementation of the database, using a commercial DBMS. Most current commercial DBMSs use an implementation data model – such as the relational (SQL) model – so the conceptual schema is transformed from the high – level data model into the implementation data model. This step is called logical design or data model mapping; its result is a database schema in the implementation data model of the DBMS. Data model mapping is often automated or semi-automated within the database design tools. The last step is the physical design phase, during which the internal storage structure, file organization, indexes, access paths and physical design patterns for the database file are specified. In parallel with this activities, application programs are designed and implemented as database transactions corresponding to the high – level transaction specifications.

1.1 A Sample Database Application

Once the information is collected for the database application, the tables are then created. A brief description like given under taking the example of the COMPANY database would be very helpful for understanding the concept.

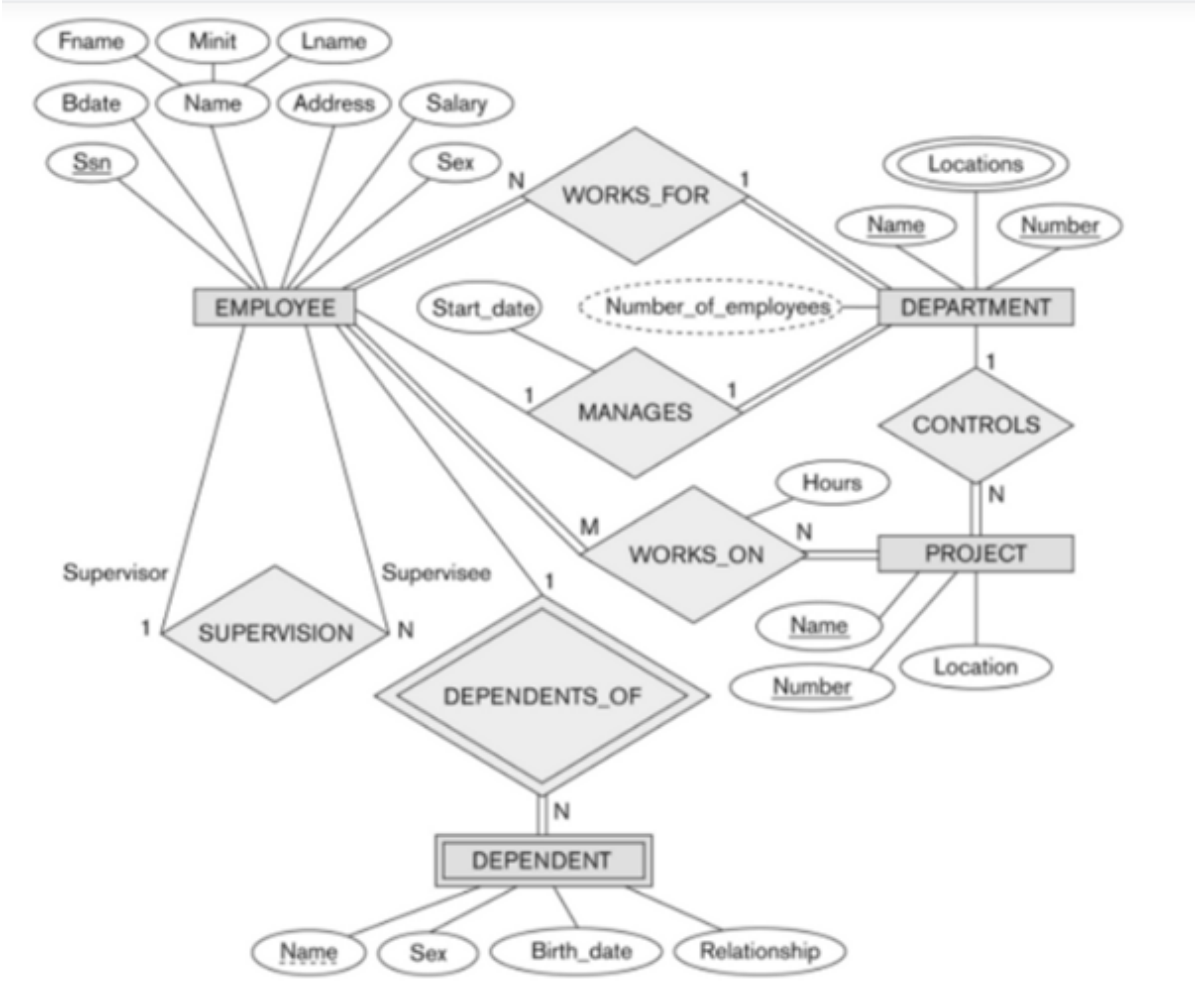
- The company is organized into departments. Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. The department may have several locations.
- A department controls a number of projects, each of which has a unique name, a unique number and a single location.
- The database will store, each employees name, social security number, address, salary, sex (gender) and birth date. An employee is assigned to one department, but may work on several projects, which are not necessarily controlled by the same department. It is required to keep track of the current number of hours per week that an employee works on each project, as well as the direct supervisor of each employee (who is another employee)

- The database will keep track of the dependents of each employee for insurance purposes, including each dependent first name, sex, birth date and relationship to the employee.

1.2 Entity Types, Entity Sets, Attributes and Keys

Definition 1.1 (Entity)**.** An entity may be an object with a physical existence (e.g., a particular person, car, house or employee) or it may be an object with a conceptual existence (e.g., a company, a job or a university course).

Definition 1.2 (Attribute)**.** Each entity has attributes – the particular properties that describe it. A particular entity will have a value for each of its attributes. The attribute values that describe each entity become a major part of the data stored in the database.



Definition 1.3 (Composite versus Simple (atomic) attributes)**.** Composite attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings. Attributes that are not divisible are called

simple or atomic attributes. The value of a composite attribute is the concatenation of the values of its component simple attributes.

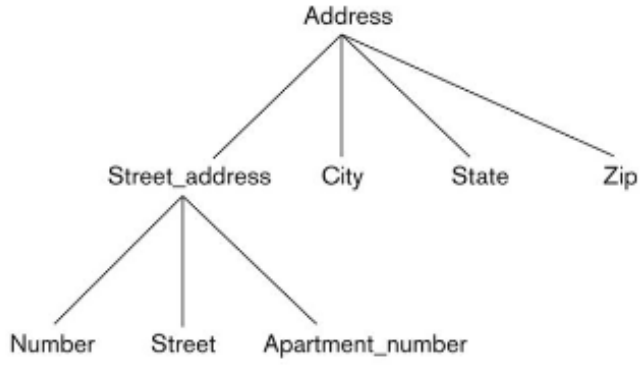


Figure 3.4
A hierarchy of composite attributes.

Definition 1.4 (Single valued versus multivalued attributes)**.** Most attributes have a single value for a particular entity; such attributes are called single – valued. For example, age is a single valued attribute for a person.in some cases an attribute can have a set of values for the same entity. For instance, a colors attribute for a car, or a college degree attribute for a person.

Definition 1.5 (Stored versus derived attributes)**.** In some cases, two (or more) attribute values are related – for example, the age and birth date attributes of a person. The age attribute is called a derived attribute and is said to be derivable from the birthdate attribute which is called a stored attribute. Attribute of number of employees in a company is also derivable.

Definition 1.6 (NULL Values)**.** In some cases, a particular entity may not have an applicable value for an attribute. NULL can able be used if we do not know the value of an attribute for a particular entity. The meaning of the former type of NULL is not applicable, whereas the meaning of the latter is unknown. The unknown category of NULL can be further classified into two cases. The first case arises when it is known that the attribute value exists but is missing (e.g., if the height attribute of a person is listed as NULL). The second case arises when it is not known whether the attribute value exists (if the home_phone attribute of a person is NULL).

Definition 1.7 (Complex Attributes)**.** In general, composite and multivalued attributes can be nested arbitrarily. We can represent arbitrary nesting by grouping components of a composite attribute between parenthesis () and separating the components with commas, and by displaying multivalued attributes between the braces . Such attributes are called complex attributes.

Example - AddressPhone
{AddressPhone(
{Phone(AreaCode,PhoneNumber)},Address
(StreetAddress(Number,Street,ApartmentNumb
er),City,State,Zip)) }

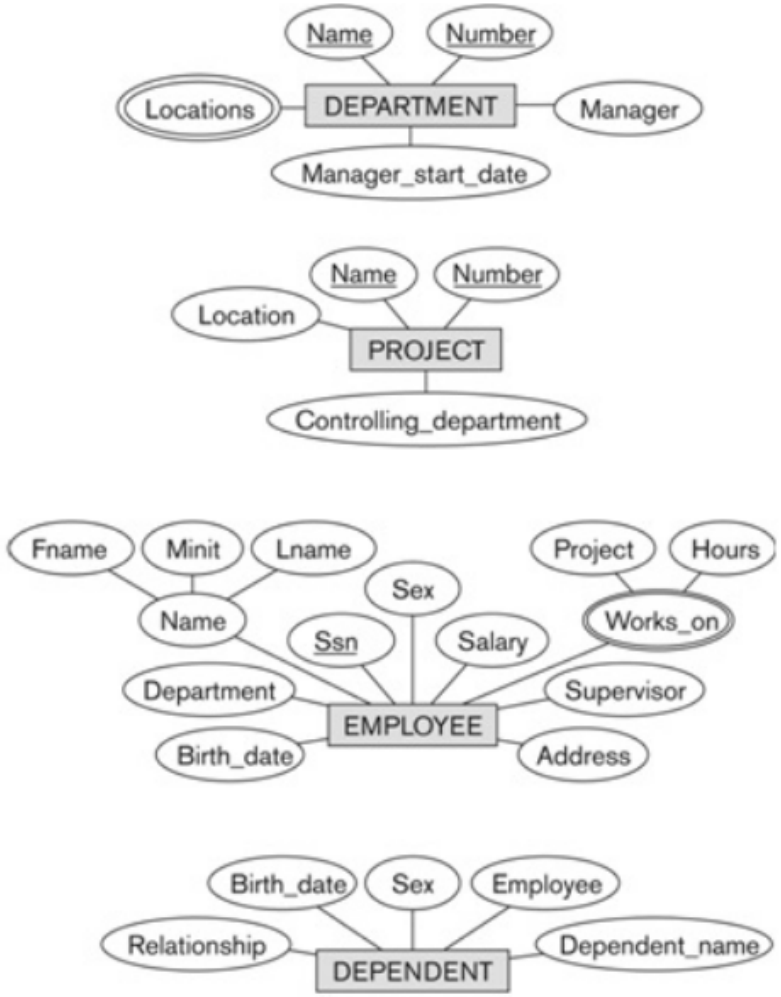
2 Entity types, Entity Sets, Keys and Value Sets

An entity type defines a collection (or set) of entities that have the same attribute. The collection of all entities of a particular entity type in the database at any point in time is called an entity set or entity collection; the entity set is usually referred to using the same name as the entity type, even though they are two separate concepts. An entity type describes the schema or intension for a set of entities that share the same structure. The collection of entities of a particular entity type is grouped into an entity set, which is also called the extension of the entity type.

Definition 2.1 (Key attributes of an Entity Type)**.** An important constraint on the entities of an entity type is the Key or Uniqueness constraint on attributes. An entity type usually has one or more attributes whose values are distinct for each individual entity in the entity set. Such an attribute is called a key attribute, and its values can be used to identify each entity uniquely. In ER diagrammatic notation, each key attribute has its name underlined inside the oval. An entity type may also have no key, in which case it is called a weak entity type.

Definition 2.2 (Value Sets (Domains) of attributes)**.** Each simple attribute of an entity type is associated with a value set (or domain of values), which specifies the set of values that may be assigned to that attribute for each individual entity.

3 Initial Conceptual Design of the COMPANY database

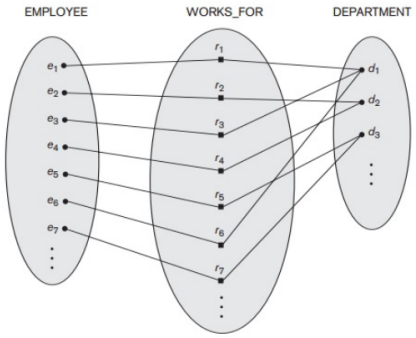


Preliminary design of entity types for the COMPANY database.

Another requirement is that an employee can work on several projects, and the database has to store the number of hours per week an employee works on each project. This can be represented by a multivalued composite attribute of employee called works_on with the simple components (project, hours). Alternatively, it can be represented as a multivalued composite attribute of project called workers with the simple components (employee, hours)

4 Relationship types, relationship sets, roles and structural constraints

A relationship type R among n entity types $E_1, E_2, \cdots E_n$ defines a set of associations – or a relationship set – among entities from these entity types. Mathematically, the relationship set R is a set of relationship instances r_i, where each r_i associates n individual entities $(e_1, e_2, \cdots, e_n)$, and each entity e_j in r_i is a member of entity set $E_j, 1 \leq j \leq n$.



Each of the entity types $E_1, E_2, \cdots E_n$ is said to participate in the relationship type R; similarly, each of the individual entities $e_1, e_2, \cdots e_n$ is said to participate in the relationship instance $r_i = (e_1, e_2, \cdots, e_n)$.

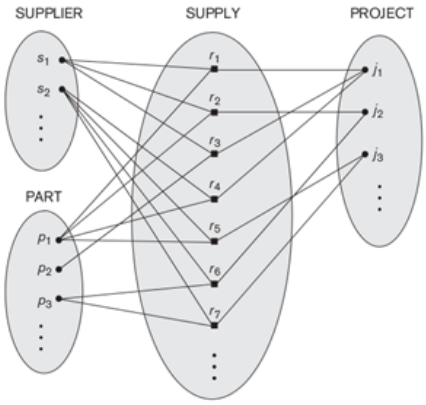
In ER diagrams, relationship types are displayed as diamond shaped boxes, which are connected by straight lines to the rectangular boxes representing the participating entity types.

4.1 Relationship Degree, Role names and Recursive Relationships

The degree of a relationship type is the number of participating entity types. A relationship type of degree two is called binary, and one of degree three is called ternary.

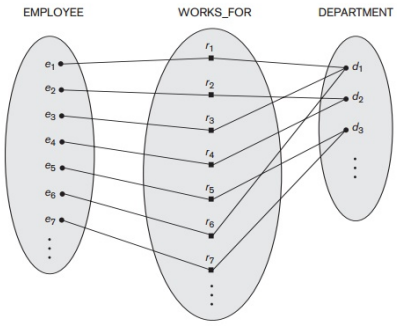
E.g., of Binary Relationship

E.g., of ternary relationship



4.2 Relationships as attributes

It is sometimes convenient to think of a binary relationship type in terms of attributes. Consider the following example of works for relationship type in the following diagram.

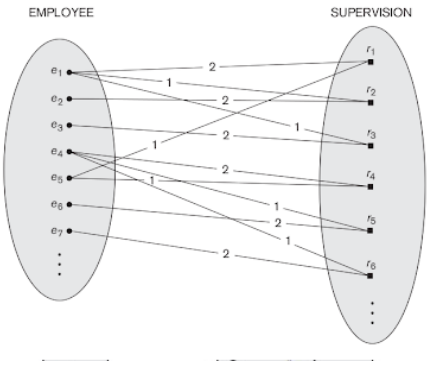


Employees of the entity type DEPARTMENT whose value for each department entity is he set of EMPLOYEE entities who works for that department. The value set of this Employees attribute is the power set of the EMPLOYEE entity set. Either of these two attributes – department of EMPLOYEE or employees of DEPARTMENT can represent the WORKS_FOR relationship type. If both are represented, they are constrained to be inverses of each other.

4.3 Role Names and Recursive Relationships

- Each entity type that participates in a relationship type plays a particular role in the relationship. The role name signifies the role that a participating entity from the entity type plays in each relationship instance, and it helps to explain what the relationship means. For example, in the WORKS_FOR relationship type, EMPLOYEE plays the role of employee or worker and DEPARTMENT plays the role of department or employer.

Role names are not technically necessary in relationship types where all the participating entity types are distinct, since each participating entity type name can be used as the role name. however, in some cases the same entity type participates more than once in a relationship types in different roles. In such cases the role name becomes essential for distinguishing the meaning of the role that each participating entity plays. Such relationship types are called recursive relationships or self-referencing relationships.



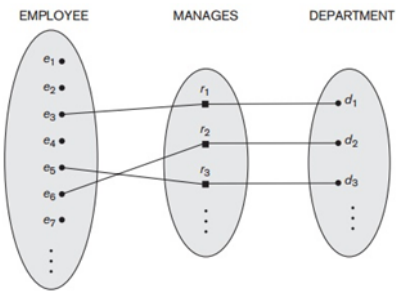
4.4 Constraints on Binary Relationship Types

Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set. We can distinguish two main types of binary relationship constraints: cardinality ratio and participation.

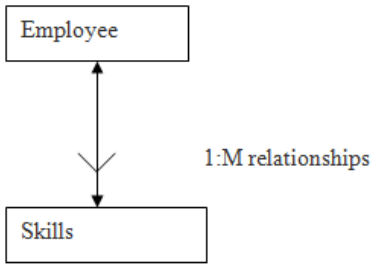
Definition 4.1 (Cardinality Ratio)**.** The cardinality ratio for a binary relationship specifies the maximum number of relationship instances that an entity can participate in.

Types of cardinality ratio are: -

- 1:1 (one to one) e.g., A 1:1 relationship MANAGES

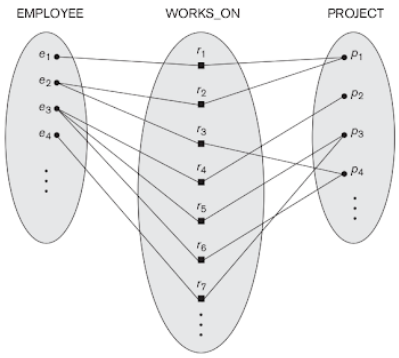


- 1:M (one to many)



- M:1 (many to one)

- M: N (many to many) e.g., An M:N relationship WORKS_ON



Cardinality ratios for binary relationships are represented on ER diagrams by displaying 1, M and N on the diamonds

4.5 Participation constraints and existence dependencies

The participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type. This constraint specifies the minimum number of relationship instances that each entity can participate in and is sometimes called the minimum cardinality constraint. There are two types of participation constraints – total and partial – that we illustrate by example. If a company policy states that every employee must work for a department, then an employee entity can exist only if it participates in atleast one WORKS_FOR relationship instance.

Thus, the participation of EMPLOYEE in WORKS_FOR is called total participation, meaning that every entity in the total set of employee entities

must be related to a department entity via WORKS_FOR. Total participation is also called existence dependency. for example, if we do not expect every employee to manage a department, so the participation of EMPLOYEE in the MANAGES relationship type is partial, meaning that some or part of the set of employee entities are related to some department entity via MANAGES but not necessarily all.

In ER diagrams, total participation (or existence dependency) is displayed as a double line connecting the participating entity type to the relationship, whereas partial participation is represented by a single line.

4.6 Attributes of Relationship Types

Relationship types can also have attributes, similar to those of entity types. For example, to record the number of hours per week that a particular employee works on a particular project, we can include an attribute Hours for the WORKS_ON relationship type.

5 Weak Entity Types

Definition 5.1 (Weak Entity)**.** Entity types that do not have key attributes of their own are called weak entity types. A weak entity always has a total participation constraint (existence dependency) with respect to its identifying relationship because a weak entity cannot be identified without an owner entity. A weak entity type normally has a partial key, which is the attribute that can uniquely identify weak entities that are related to the same owner entity.

In ER diagrams, both a weak entity type and its identifying relationship are distinguished by surrounding their boxes and diamonds with double lines. The partial key attribute is underlined with a dashed or dotted line.

5.1 Refining the ER design

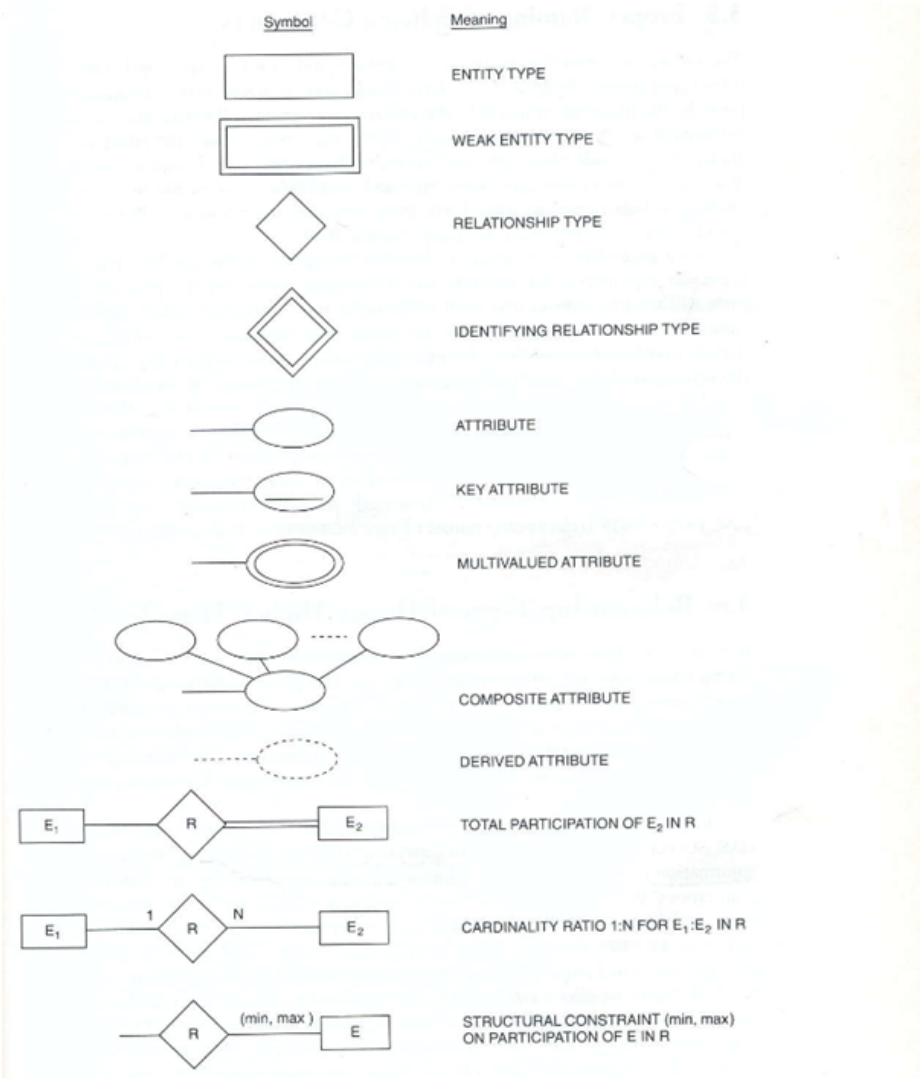
- MANAGES, a 1:1 relationship type between EMPLOYEE and DEPARTMENT. EMPLOYEE participation is partial. DEPARTMENT participation is not clear from the requirements. We question the users, who say that a department must have a manager at all times, which implies total participation.13 The attribute Start_date is assigned to this relationship type.
- WORKS_FOR, a 1: N relationship type between DEPARTMENT and EMPLOYEE. Both participation are total.
- CONTROLS, a 1: N relationship type between DEPARTMENT and PROJECT. The participation of PROJECT is total, whereas that of DEPARTMENT is determined to be partial, after consultation with the users indicates that some departments may control no projects.
- SUPERVISION, a 1: N relationship type between EMPLOYEE (in the supervisor role) and EMPLOYEE (in the supervisee role). Both participation are determined to be partial, after the users indicate that not every employee is a supervisor and not every employee has a supervisor.

- WORKS_ON, determined to be an M: N relationship type with attribute Hours, after the users indicate that a project can have several employees working on it. Both participation are determined to be total.
- DEPENDENTS_OF, a 1: N relationship type between EMPLOYEE and DEPENDENT, which is also the identifying relationship for the weak entity type DEPENDENT. The participation of EMPLOYEE is partial, whereas that of DEPENDENT is total.

After specifying the above six relationship types, we remove from the entity types in Figure 7.8 all attributes that have been refined into relationships.

These include Manager and Manager_start_date from DEPARTMENT; Controlling_department from PROJECT; Department, Supervisor, and Works_on from EMPLOYEE; and Employee from DEPENDENT. It is important to have the least possible redundancy when we design the conceptual schema of a database.

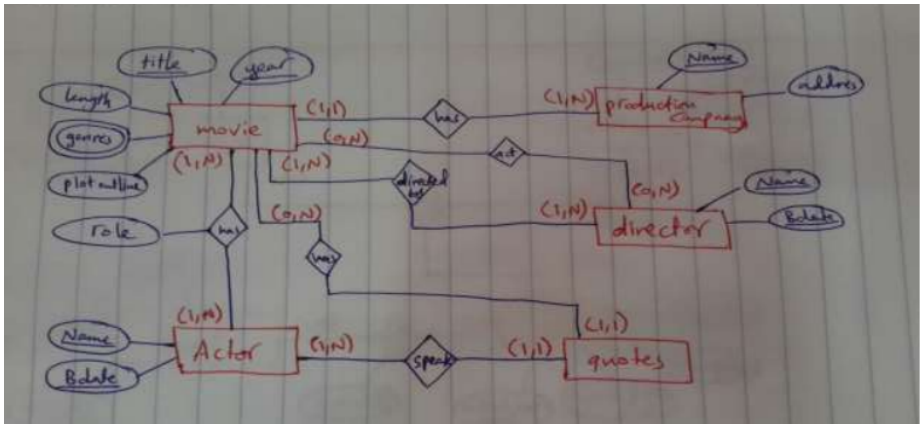
6 ER Diagrams, Naming Conventions, and Design Issues



Example 1. Consider a MOVIE database in which data is recorded about the movie industry.The data requirements are summarized as follows:

- Each movie is identified by title and year of release. Each movie has a length in minutes. Each has a production company, and each is classified under one or more genres (such as horror, action, drama, and so forth). Each movie has one or more directors and one or more actors appear in it. Each movie also has a plot outline. Finally, each movie has zero or more quotable quotes, each of which is spoken by a particular actor appearing in the movie.

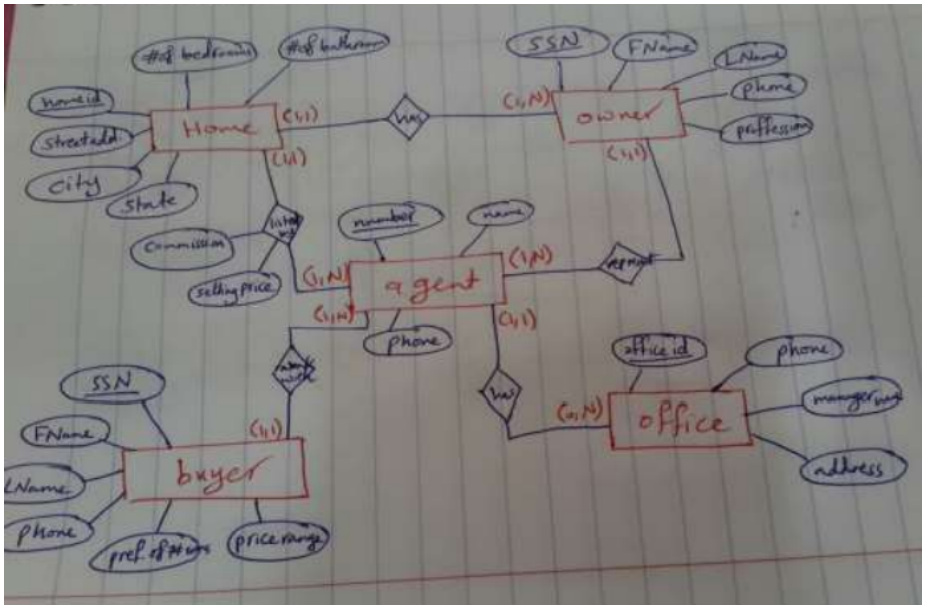
- Actors are identified by name and date of birth and appear in one or more movies. Each actor has a role in the movie.
- Directors are also identified by name and date of birth and direct one or more movies. It is possible for a director to act in a movie (including one that he or she may also direct).
- Production companies are identified by name and each has an address. A production company produces one or more movies.



Example 2. Draw an ER diagram to model the application with the following assumptions.Specify key attributes of each entity type and (min, max) constraints on each relationship type.

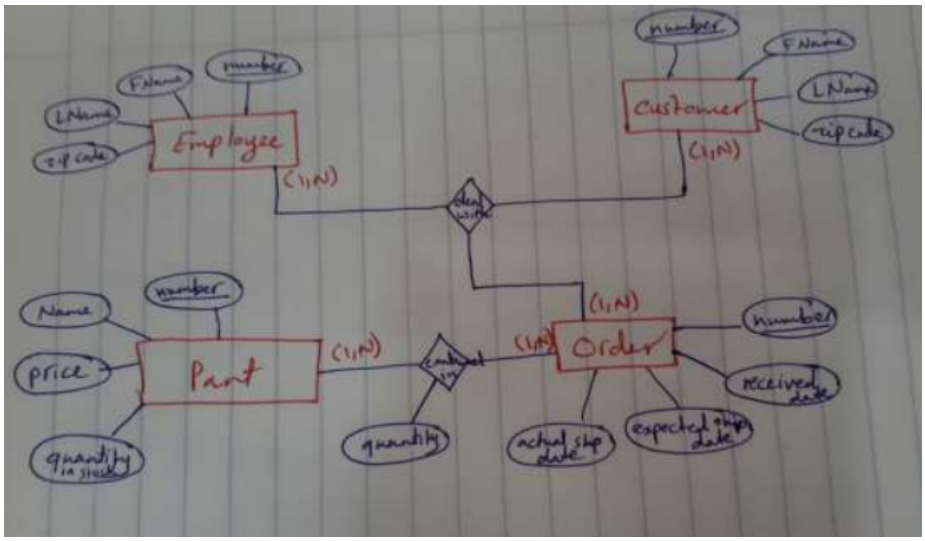
- Each home uniquely defined by home identifier, street address, city, state, a number of bedrooms and a number of bathrooms and an associated owner.
- Each owner has a Social Security Number, first name, last name, phone, and profession.
- An owner can spouse one or more homes.
- Agents represent owners in the sale of a home. An agent can list many homes, but only one agent can list a home.
- An agent has a unique agent number, name, phone number and an associated office.
- When an owner agrees to list a home with an agent, a commission and a selling price are determined.
- An office has office identifier, phone number, the manager name, address and an optional agent number.
- Many agents can work at one office.

- A buyer entity type has a Social Security Number, first name, last name, phone, preferences for the number of bedrooms and bathrooms, and a price range.
- An agent can work with many buyers, but a buyer works with only one agent.



Example 3. Consider a mail order database in which employees take orders for parts from customers.Design an Entity-Relationship diagram for the mail order database.The data requirements are summarized as follows:

- The mail order company has employees identified by a unique employee number, their first and last names, and a zip code where they are located.
- The customers of the company are identified by a unique customer number, their first and last names, and a zip code where they are located.
- The parts being sold by the company are identified by a unique part number, a part name, their price, and quantity in stock.
- Orders placed by customers are taken by employees and are given a unique order number. Each order may contain certain quantities of one or more parts.
- Each order has a received date as well as an expected ship date. The actual ship date is also recorded.



7 Database Design

7.1 Functional Dependencies

If there exists a functional dependency between two attributes of a table, it means that if I have the left hand column value I can uniquely find out the right hand side value of a different column.

Definition 7.1 (Functional Dependencies)**.** A functional dependency, denoted by $X \to Y$, between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation state r of R. The constraint is that, for any two tuples t_1 and t_2 in r that have $t_1[X] = t_2[X]$, they must also have $t_1[Y] = t_2[Y]$.

7.2 Closure set of attributes

$R(ABC)$

$A \to B; B \to C$

$A \to B \to C$: then if we know a then we can find out c
$A \to BC$ This is the closure set of attributes.

Example 1. $R(ABCDEFG)$
$A \to B$
$BC \to DE$
$AEG \to G$
Find the closure of AC i.e, $(AC)+$

Example 2. $R(ABCDE)$
$A \rightarrow BC$
$CD \rightarrow E$
$B \rightarrow D$
$E \rightarrow A$
Find the closure of B i.e, $(B)+$

Example 3. $R(ABCDE)$
$AB \rightarrow C$
$BC \rightarrow AD$
$D \rightarrow E$
$CF \rightarrow B$
Find the closure of AB i.e, $(AB)+$

Example 4. $R(ABCDEFGH)$
$A \rightarrow BC$
$CD \rightarrow E$
$E \rightarrow C$
$D \rightarrow AEH$
$ABH \rightarrow BD$
$DH \rightarrow BC$
$DoesBCD \rightarrow H$?

7.3 Equivalence of Functional Dependencies

Example 5. $R(ACDEH)$
$F : A \rightarrow C$
$AC \rightarrow D$
$E \rightarrow AH$
$G : A \rightarrow CD$
$E \rightarrow AD$
$E \rightarrow H$

We will first try to get both functionalities of F and G similar. To do that we would find the closure of F using G. we would try to find the closure of $(A)+(AC)+(E)+$ Using the Rules of G function
$(A)+ = ACD; (AC)+ = ACD; (D)+ = EAHCD$ Now we compare the above closures to track f. We find that both are same.

7.4 Irreducible set of functional dependencies (canonical form)

If the functional dependencies have a repeating dependency we need to remove that repetition.

Example 6. Consider R (WXYZ) $X \rightarrow W$
$WZ \rightarrow XY$
$Y \rightarrow WXY$ Using decomposition rule: $X \rightarrow W$
$WZ \rightarrow X$
$WZ \rightarrow Y$

$Y \rightarrow W$
$Y \rightarrow X$
$Y \rightarrow Z$
Using the 1st functional dependency we try to find $(x)+ = xw$ Now ignoring 1st functional dependency we try to again find $(x)+$ which is not possible. So this is important Like this it is realized that 2 and 4 are not important dependencies so removing that we get $X \rightarrow W$
$WZ \rightarrow Y$
$Y \rightarrow X$
$Y \rightarrow Z$
Now we have to find if there are any repetitions on the left hand side now for doing so we compute $(WZ)+ = WZXY$
$(W)+$
$(Z)+$
We find that there is no redundancy here so we checked for both left and right.
$X \rightarrow W$
$WZ \rightarrow Y$
$Y \rightarrow XZ$

8 Normal Forms based on Primary Keys

Having introduced functional dependencies, we are now ready to use them to specify some aspects of the semantics of relation schemas. We assume that a set of functional dependencies is given for each relation, and that each relation has a designated primary key; this information combined with the tests (conditions) for normal forms drives the normalization process for relational schema design. Most practical relational design projects take one of the following two approaches:

- Perform a conceptual schema design using a conceptual model such as ER or EER and map the conceptual design into a set of relations
- Design the relations based on external knowledge derived from an existing implementation of files or forms or reports

Following either of these approaches, it is then useful to evaluate the relations for goodness and decompose them further as needed to achieve higher normal forms, using the normalization theory presented in this chapter and the next. We focus in this section on the first three normal forms for relation schemas and the intuition behind them, and discuss how they were developed historically. More general definitions of these normal forms, which take into account all candidate keys of a relation rather than just the primary key.

A relation $R(A, B, C, D)$ with its extension.

A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a2	b2	c2	d3
a3	b3	c4	d3

We start by informally discussing normal forms and the motivation behind their development, as well as reviewing some definitions from Chapter 3 that are needed here. Then we discuss the first normal form (1NF) and present the definitions of second normal form (2NF) and third normal form (3NF), which are based on primary keys

8.1 Normalisation of Relations

The normalization process, as first proposed by Codd (1972a), takes a relation schema through a series of tests to certify whether it satisfies a certain **normal form**. The process, which proceeds in a top-down fashion by evaluating each relation against the criteria for normal forms and decomposing relations as necessary, can thus be considered as relational design by analysis. Initially, Codd proposed three normal forms, which he called first, second, and third normal form. A stronger definition of 3NF—called Boyce-Codd normal form (BCNF)—was proposed later by Boyce and Codd. All these normal forms are based on a single analytical tool: the functional dependencies among the attributes of a relation. Later, a fourth normal form (4NF) and a fifth normal form (5NF) were proposed, based on the concepts of multivalued dependencies and join dependencies.

Normalization of data can be considered a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of

- minimizing redundancy
- minimizing the insertion, deletion, and update anomalies

It can be considered as a "filtering" or "purification" process to make the design have successively better quality. Unsatisfactory relation schemas that do not meet certain conditions—the normal form tests—are decomposed into smaller relation schemas that meet the tests and hence possess the desirable properties. Thus, the normalization procedure provides database designers with the following:

- A formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes.
- A series of normal form tests that can be carried out on individual relation schemas so that the relational database can be normalized to any desired degree.

Definition 8.1 (Normal Form)**.** The normal form of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized.

Normal forms, when considered in isolation from other factors, do not guarantee a good database design. It is generally not sufficient to check separately that each relation schema in the database is, say, in BCNF or 3NF. Rather, the process of normalization through decomposition must also confirm the existence of additional properties that the relational schemas, taken together, should possess. These would include two properties:

- The **nonadditive join or lossless join property**, which guarantees that the spurious tuple generation problem does not occur with respect to the relation schemas created after decomposition.
- The **dependency preservation property**, which ensures that each functional dependency is represented in some individual relation resulting after decomposition.

The nonadditive join property is extremely critical and must be achieved at any cost, whereas the dependency preservation property, although desirable, is some-times sacrificed

8.2 Practical Use of Normal Forms

Most practical design projects acquire existing designs of databases from previous designs, designs in legacy models, or from existing files.

Normalization is carried out in practice so that the resulting designs are of high quality and meet the desirable properties stated previously. Although several higher normal forms have been defined, such as the 4NF and 5NF, the practical utility of these normal forms becomes questionable when the constraints on which they are based are rare, and hard to understand or to detect by the database designers and users who must discover these constraints. Thus, database design as practiced in industry today pays particular attention to normalization only up to 3NF, BCNF, or at most 4NF.

Another point worth noting is that the database designers need not normalize to the highest possible normal form. Relations may be left in a lower normalization status, such as 2NF, for performance reasons. Doing so incurs the corresponding penalties of dealing with the anomalies.

Definition 8.2 (Denormalization)**.** Denormalization is the process of storing the join of higher nor-mal form relations as a base relation, which is in a lower normal form.

8.3 Definitions of Keys and Attributes Participating in Keys

Before proceeding further, let's look again at the definitions of keys of a relation schema

Definition 8.3 (Superkey)**.** A superkey of a relation schema $R = \{A_1, A_2, \cdots A_n\}$ is a set of attributes $S \subseteq R$ with the property that no two tuples t1 and t2 in any legal relation state r of R will have $t_1[S] = t_2[S]$.

Definition 8.4 (Key)**.** A key K is a superkey with the additional property that removal of any attribute from K will cause K not to be a superkey any more.

The difference between a key and a superkey is that a key has to be minimal; that is,

If we have a key$K = \{A_1, A_2, \cdots, A_k\}$ of R, then $K - \{A_i\}$ is not a key of R for any $A_i, 1 \leq i \leq k$.

If a relation schema has more than one key, each is called a **candidate key**. One of the candidate keys is arbitrarily designated to be the **primary key**, and the others are called secondary keys. In a practical relational database, each relation schema must have a primary key. If no candidate key is known for a relation, the entire relation can be treated as a default superkey.

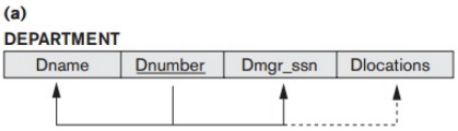
Definition 8.5. An attribute of relation schema R is called a **prime attribute** of R if it is a member of some candidate key of R. An attribute is called **non-prime** if it is not a prime attribute—that is, if it is not a member of any candidate key

We now present the first three normal forms: 1NF, 2NF, and 3NF. These were proposed by Codd (1972a) as a sequence to achieve the desirable state of 3NF relations by progressing through the intermediate states of 1NF and 2NF if needed. As we shall see, 2NF and 3NF attack different problems. However, for historical reasons, it is customary to follow them in that sequence; hence, by definition a 3NF relation already satisfies 2NF.

8.4 First Normal Form

First normal form (1NF) is now considered to be part of the formal definition of a relation in the basic (flat) relational model; historically, it was defined to disallow multivalued attributes, composite attributes, and their combinations. It states that the domain of an attribute must include only atomic (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute. Hence, 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple. In other words, 1NF dis-allows relations within relations or relations as attribute values within tuples. The only attribute values permitted by 1NF are single **atomic (or indivisible)** values. Consider the DEPARTMENT relation schema shown in the figure below, whose primary key is Dnumber, and suppose that we extend it by including the Dlocations attribute as shown in Figure (a). We assume that each department can have a number of locations. The DEPARTMENT schema and a sample relation state are shown in Figure 15.9. As we can see, this is not in 1NF because Dlocations is not an atomic attribute, as illustrated by the first tuple in Figure (b). There are two ways we can look at the Dlocations attribute:

The domain of Dlocations contains atomic values, but some tuples can have a set of these values. In this case, Dlocations is not functionally dependent on the primary key Dnumber.



(a)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations

(b)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Figure 15.9
Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Sample state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

The domain of Dlocations contains sets of values and hence is nonatomic. In this case, Dnumber → Dlocations because each set is considered a single member of the attribute domain.

In either case, the DEPARTMENT relation in the above figure is not in 1NF; in fact, it does not even qualify as a relation according to our definition of relation. There are three main techniques to achieve first normal form for such a relation:

- Remove the attribute Dlocations that violates 1NF and place it in a separate relation DEPT_LOCATIONS along with the primary key Dnumber of DEPARTMENT. The primary key of this relation is the combination Dnumber, Dlocation. A distinct tuple in DEPT_LOCATIONS exists for each location of a department. This decomposes the non-1NF relation into two 1NF relations.

Expand the key so that there will be a separate tuple in the original DEPARTMENT relation for each location of a DEPARTMENT, as shown in Figure 15.9(c). In this case, the primary key becomes the combination Dnumber, Dlocation. This solution has the disadvantage of introducing redundancy in the relation. If a maximum number of values is known for the attribute—for example, if it is known that at most three locations can exist for a department—replace the Dlocations attribute by three atomic attributes: Dlocation1, Dlocation2, and Dlocation3. This solution has the disadvantage of introducing NULL values if most departments have fewer than three locations. It further introduces spurious semantics about the ordering among the location values that is not originally intended. Querying on this attribute becomes more difficult; for example, consider how you would write the query: List the departments that have 'Bellaire' as one of their locations in this design.

Of the three solutions above, the first is generally considered best because it does not suffer from redundancy and it is completely general, having no limit placed on a maximum number of values. In fact, if we choose the second solution, it will be decomposed further during subsequent normalization steps into the first solution.

First normal form also disallows multivalued attributes that are themselves composite. These are called nested relations because each tuple can have a relation within it. Figure 15.10 shows how the EMP_PROJ relation could appear if nesting is allowed. Each tuple represents an employee entity, and a relation PROJS(Pnumber, Hours) within each tuple represents the employee's projects and the hours per week that employee works on each project. The schema of this EMP_PROJ relation can be represented as follows:
EMP_PROJ(Ssn, Ename, PROJS(Pnumber, Hours))

The set braces identify the attribute PROJS as multivalued, and we list the component attributes that form PROJS between parentheses ( ).

Notice that Ssn is the primary key of the EMP_PROJ relation in Figures 15.10(a) and (b), while Pnumber is the partial key of the nested relation; that is, within each tuple, the nested relation must have unique values of Pnumber. To normalize this into 1NF, we remove the nested relation attributes into a new relation and propagate the primary key into it; the primary key of the new relation will combine the partial key with the primary key of the original relation. Decomposition and primary key propagation yield the schemas EMP_PROJ1 and EMP_PROJ2, as shown in Figure 15.10(c).

This procedure can be applied recursively to a relation with multiple-level nesting to unnest the relation into a set of 1NF relations. This is useful in converting an unnormalized relation schema with many levels of nesting into 1NF relations.

(a)

EMP_PROJ

		Projs	
Ssn	Ename	Pnumber	Hours

(b)

EMP_PROJ

Ssn	Ename	Pnumber	Hours
123456789	Smith, John B.	1	32.5
		2	7.5
666884444	Narayan, Ramesh K.	3	40.0
453453453	English, Joyce A.	1	20.0
		2	20.0
333445555	Wong, Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya, Alicia J.	30	30.0
		10	10.0
987987987	Jabbar, Ahmad V.	10	35.0
		30	5.0
987654321	Wallace, Jennifer S.	30	20.0
		20	15.0
888665555	Borg, James E.	20	NULL

Figure 15.10
Normalizing nested relations into 1NF. (a) Schema of the EMP_PROJ relation with a *nested relation* attribute PROJS. (b) Sample extension of the EMP_PROJ relation showing nested relations within each tuple. (c) Decomposition of EMP_PROJ into relations EMP_PROJ1 and EMP_PROJ2 by propagating the primary key.

(c)

EMP_PROJ1

Ssn	Ename

EMP_PROJ2

Ssn	Pnumber	Hours

The existence of more than one multivalued attribute in one relation must be handled carefully. As an example, consider the following non-1NF relation:

$\{PERSON\,(Ss\#, \{Car\_lic\#\}, \{Phone\#\})\}$

This relation represents the fact that a person has multiple cars and multiple phones. If strategy 2 above is followed, it results in an all-key relation:

$\{PERSON\_IN\_1NF\,(Ss\#, Car\_lic\#, Phone\#)\}$

To avoid introducing any extraneous relationship between Car_lic# and Phone#, all possible combinations of values are represented for every Ss#, giving rise to redundancy. This leads to the problems handled by multivalued dependencies and 4NF. The right way to deal with the two multivalued attributes in PERSON shown previously is to decompose it into two separate relations, using strategy 1 discussed above: P1(Ss#, Car_lic#) and P2(Ss#, Phone#).

9 Second Normal Form

Second normal form (2NF) is based on the concept of full functional dependency. A functional dependency X→Y is a full functional dependency if removal of any attribute A from X means that the dependency does not hold any more; that is, for any attribute A εX, $(X - \{A\})$does not functionally determine Y. A functional dependency X→ Y is a partial dependency if some attribute A ε X can be removed from X and the dependency still holds; that is, for some A ε X, $(X - \{A\}) \rightarrow Y$. In Figure 15.3(b), $\{Ssn, Pnumber\} \rightarrow Hours$ is a full dependency ($neither\ Ssn \rightarrow Hours\ nor\ Pnumber \rightarrow Hours\ holds$). However, the dependency $\{Ssn, Pnumber\} \rightarrow Ename$ is partial because Ssn → Ename holds.

Definition 9.1. A relation schema R is in 2NF if every nonprime attribute A in R is fully functionally dependent on the primary key of R.

The test for 2NF involves testing for functional dependencies whose left-hand side attributes are part of the primary key. If the primary key contains a single attribute, the test need not be applied at all. The EMP_PROJ relation in Figure 15.3(b) is in 1NF but is not in 2NF. The nonprime attribute Ename violates 2NF because of FD2, as do the nonprime attributes Pname and Plocation because of FD3. The functional dependencies FD2 and FD3 make Ename, Pname, and Plocation partially dependent on the primary key $\{Ssn, Pnumber\}$ of EMP_PROJ, thus violating the 2NF test.

If a relation schema is not in 2NF, it can be second normalized or 2NF normalized into a number of 2NF relations in which nonprime attributes are associated only with the part of the primary key on which they are fully functionally dependent. Therefore, the functional dependencies FD1, FD2, and FD3 in Figure 15.3(b) lead to the decomposition of EMP_PROJ into the three relation schemas EP1, EP2, and EP3 shown in Figure 15.11(a), each of which is in 2NF.

10 Third Normal Form

Third normal form (3NF) is based on the concept of transitive dependency. A functional dependency X→Y in a relation schema R is a transitive dependency if there exists a set of attributes Z in R that is neither a candidate key nor a subset of any key of R,10 and both X→Z and Z→Y hold. The dependency Ssn→Dmgr_ssn is transitive through Dnumber in EMP_DEPT in Figure 15.3(a), because both the dependencies Ssn→Dnumber and Dnumber→Dmgr_ssn hold

and Dnumber is neither a key itself nor a subset of the key of EMP_DEPT. Intuitively, we can see that the dependency of Dmgr_ssn on Dnumber is undesirable in since Dnumber is not a key of EMP_DEPT.

Definition 10.1. According to Codd's original definition, a relation schema R is in 3NF if it satisfies 2NF and no nonprime attribute of R is transitively dependent on the primary key.

The relation schema EMP_DEPT in Figure 15.3(a) is in 2NF, since no partial dependencies on a key exist. However, EMP_DEPT is not in 3NF because of the transitive dependency of Dmgr_ssn (and also Dname) on Ssn via Dnumber. We can normalize EMP_DEPT by decomposing it into the two 3NF relation schemas ED1 and ED2 shown in Figure 15.11(b). Intuitively, we see that ED1 and ED2 represent independent entity facts about employees and departments. A NATURAL JOIN operation on ED1 and ED2 will recover the original relation EMP_DEPT without generating spurious tuples.

Intuitively, we can see that any functional dependency in which the left-hand side is part (a proper subset) of the primary key, or any functional dependency in which the left-hand side is a nonkey attribute, is a problematic FD. 2NF and 3NF normalization remove these problem FDs by decomposing the original relation into new relations. In terms of the normalization process, it is not necessary to remove the partial dependencies before the transitive dependencies, but historically, 3NF has been defined with the assumption that a relation is tested for 2NF first before it is tested for 3NF. Table 15.1 informally summarizes the three normal forms based on primary keys, the tests used in each case, and the corresponding remedy or normalization performed to achieve the normal form.

11 General Definitions of Second and Third Normal Forms

In general, we want to design our relation schemas so that they have neither partial nor transitive dependencies because these types of dependencies cause the update anomalies. The steps for normalization into 3NF relations that we have discussed so far disallow partial and transitive dependencies on the primary key. The normalization procedure described so far is useful for analysis in practical situations for a given database where primary keys have already been defined. These definitions, however, do not take other candidate keys of a relation, if

Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
Third (3NF)	Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes). That is, there should be no transitive dependency of a nonkey attribute on the primary key.	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s).

any, into account. In this section we give the more general definitions of 2NF and 3NF that take all candidate keys of a relation into account. Notice that this does not affect the definition of 1NF since it is independent of keys and functional dependencies. As a general definition of prime attribute, an attribute that is part of any candidate key will be considered as prime. Partial and full functional dependencies and transitive dependencies will now be considered with respect to all candidate keys of a relation.

12 General Definition of Second Normal Form

Definition 12.1. A relation schema R is in second normal form (2NF) if every non-prime attribute A in R is not partially dependent on any key of R.

The test for 2NF involves testing for functional dependencies whose left-hand side attributes are part of the primary key. If the primary key contains a single attribute, the test need not be applied at all. Consider the relation schema LOTS shown in Figure 15.12(a), which describes parcels of land for sale in various counties of a state. Suppose that there are two candidate keys: Property_id# and County_name, Lot#; that is, lot numbers are unique only within each county, but Property_id# numbers are unique across counties for the entire state.
Based on the two candidate keys Property_id# and County_name, Lot#, the functional dependencies FD1 and FD2 in Figure 15.12(a) hold. We choose Property_id# as the primary key, so it is underlined in Figure 15.12(a), but no special consideration will be given to this key over the other candidate key. Suppose that the following two additional functional dependencies hold in LOTS:

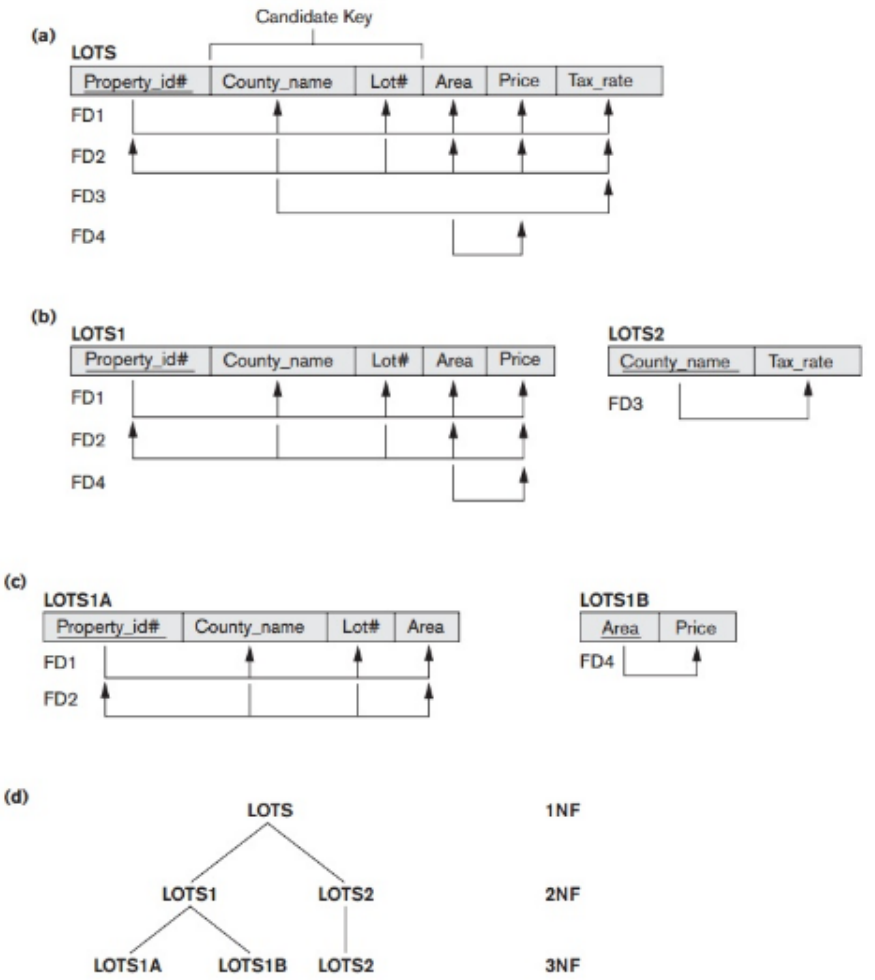
$$FD3: County\_name \rightarrow Tax\_rate$$

$$FD4: Area \rightarrow Price$$

In words, the dependency FD3 says that the tax rate is fixed for a given county (does not vary lot by lot within the same county), while FD4 says that the price of a lot is determined by its area regardless of which county it is in. (Assume that this is the price of the lot for tax purposes.)
The LOTS relation schema violates the general definition of 2NF because Tax_rate

is partially dependent on the candidate key $\{County\_name, Lot\#\}$, due to FD3. To normalize LOTS into 2NF, we decompose it into the two relations LOTS1 and LOTS2, shown in Figure 15.12(b). We construct LOTS1 by removing the attribute Tax_rate that violates 2NF from LOTS and placing it with County_name (the left-hand side of FD3 that causes the partial dependency) into another relation LOTS2. Both LOTS1 and LOTS2 are in 2NF. Notice that FD4 does not violate 2NF and is carried over to LOTS1.

Figure 15.12
Normalization into 2NF and 3NF. (a) The LOTS relation with its functional dependencies FD1 through FD4. (b) Decomposing into the 2NF relations LOTS1 and LOTS2. (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B. (d) Summary of the progressive normalization of LOTS.



13 General Definition of Third Normal Form

Definition 13.1. A relation schema R is in third normal form (3NF) if, whenever a nontrivial functional dependency X $\rightarrow$A holds in R, either (a) X is a

superkey of R, or (b) A is a prime attribute of R.

According to this definition, LOTS2 (Figure 15.12(b)) is in 3NF. However, FD4 in LOTS1 violates 3NF because Area is not a superkey and Price is not a prime attribute in LOTS1. To normalize LOTS1 into 3NF, we decompose it into the relation schemas LOTS1A and LOTS1B shown in Figure 15.12(c). We construct LOTS1A by removing the attribute Price that violates 3NF from LOTS1 and placing it with Area (the left-hand side of FD4 that causes the transitive dependency) into another relation LOTS1B. Both LOTS1A and LOTS1B are in 3NF.

Two points are worth noting about this example and the general definition of 3NF:

- LOTS1 violates 3NF because Price is transitively dependent on each of the candidate keys of LOTS1 via the nonprime attribute Area.

This general definition can be applied directly to test whether a relation schema is in 3NF; it does not have to go through 2NF first. If we apply the above 3NF definition to LOTS with the dependencies FD1 through FD4, we find that both FD3 and FD4 violate 3NF. Therefore, we could decompose LOTS into LOTS1A, LOTS1B, and LOTS2 directly. Hence, the transitive and partial dependencies that violate 3NF can be removed in any order.

14 Interpreting the General Definition of Third Normal Form

Definition 14.1. A relation schema R violates the general definition of 3NF if a functional dependency X→A holds in R that does not meet either condition—meaning that it violates both conditions (a) and (b) of 3NF. This can occur due to two types of problematic functional dependencies:

- A nonprime attribute determines another nonprime attribute. Here we typically have a transitive dependency that violates 3NF.
- A proper subset of a key of R functionally determines a nonprime attribute. Here we have a partial dependency that violates 3NF (and also 2NF).

Therefore, we can state a general alternative definition of 3NF as follows:

Definition 14.2 (Alternative Definition)**.** A relation schema R is in 3NF if every nonprime attribute of R meets both of the following conditions:

- It is fully functionally dependent on every key of R.
- It is nontransitively dependent on every key of R.

15 Boyce-Codd Normal Form

Boyce-Codd normal form (BCNF) was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF. That is, every relation in BCNF is

also in 3NF; however, a relation in 3NF is not necessarily in BCNF. Intuitively, we can see the need for a stronger normal form than 3NF by going back to the LOTS relation schema in Figure 15.12(a) with its four functional dependencies FD1 through FD4. Suppose that we have thousands of lots in the relation but the lots are from only two counties DeKalb and Fulton. Suppose also that lot sizes in DeKalb County are only 0.5, 0.6, 0.7, 0.8, 0.9, and 1.0 acres, whereas lot sizes in Fulton County are restricted to 1.1, 1.2, $\cdots$, 1.9, and 2.0 acres. In such a situation we would have the additional functional dependency FD5: Area$\rightarrow$ County_name. If we add this to the other dependencies, the relation schema LOTS1A still is in 3NF because County_name is a prime attribute.

The area of a lot that determines the county, as specified by FD5, can be represented by 16 tuples in a separate relation R(Area, County_name), since there are only 16 possible Area values (see Figure 15.13). This representation reduces the redundancy of repeating the same information in the thousands of LOTS1A tuples. BCNF is a stronger normal form that would disallow LOTS1A and suggest the need for decomposing it.

Definition 15.1. A relation schema R is in BCNF if whenever a nontrivial functional dependency X$\rightarrow$A holds in R, then X is a superkey of R.

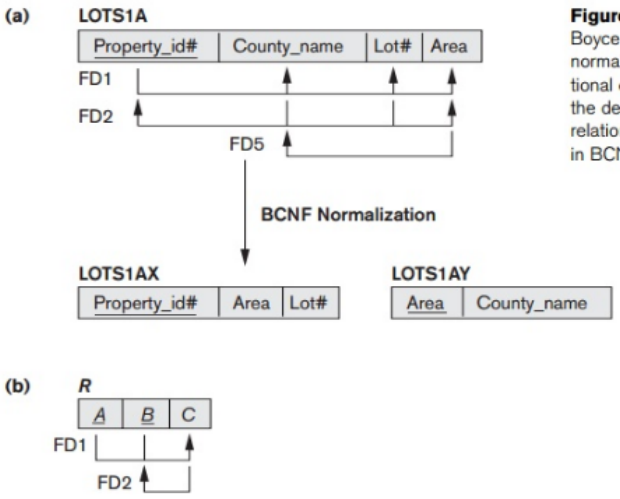


Figure 15.13
Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF.

The formal definition of BCNF differs from the definition of 3NF in that condition (b) of 3NF, which allows A to be prime, is absent from BCNF. That makes BCNF a stronger normal form compared to 3NF. In our example, FD5 violates BCNF in LOTS1A because AREA is not a superkey of LOTS1A. Note that FD5 satisfies 3NF in LOTS1A because County_name is a prime attribute (condition b), but this condition does not exist in the definition of BCNF. We can decompose LOTS1A into two BCNF relations LOTS1AX and LOTS1AY, shown in Figure 15.13(a). This decomposition loses the functional dependency FD2 because its attributes no longer coexist in the same relation after decomposition.

In practice, most relation schemas that are in 3NF are also in BCNF. Only if X→A holds in a relation schema R with X not being a superkey and A being a prime attribute will R be in 3NF but not in BCNF. The relation schema R shown in Figure 15.13(b) illustrates the general case of such a relation. Ideally, relational database design should strive to achieve BCNF or 3NF for every relation schema. Achieving the normalization status of just 1NF or 2NF is not considered adequate, since they were developed historically as stepping stones to 3NF and BCNF. As another example, consider Figure 15.14, which shows a relation TEACH with the following dependencies:

$FD1: \{Student, Course\} \rightarrow Instructor$

$FD2 :^{12} Instructor \rightarrow Course$

Note that Student, Course is a candidate key for this relation and that the dependencies shown follow the pattern in Figure 15.13(b), with Student as A, Course as B, and Instructor as C. Hence this relation is in 3NF but not BCNF. Decomposition of this relation schema into two schemas is not straightforward because it may be

Figure 15.14
A relation TEACH that is in 3NF but not BCNF.

TEACH

Student	Course	Instructor
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe
Narayan	Operating Systems	Ammar

decomposed into one of the three following possible pairs:

- $\{Student, Instructor\}$ *and* $\{Student, Course\}$
- $\{Course, Instructor\}$ *and* $\{Course, Student\}$
- $\{Instructor, Course\}$ *and* $\{Instructor, Student\}$

All three decompositions lose the functional dependency FD1. The desirable decomposition of those just shown is 3 because it will not generate spurious tuples after a join.
A test to determine whether a decomposition is nonadditive (or lossless) is discussed in Section 16.2.4 under Property NJB. In general, a relation not in BCNF should be decomposed so as to meet this property.
We make sure that we meet this property, because nonadditive decomposition

is a must during normalization. We may have to possibly forgo the preservation of all functional dependencies in the decomposed relations, as is the case in this example. Algorithm 16.5 does that and could be used above to give decomposition 3 for TEACH, which yields two relations in BCNF as:

$(\underline{Instructor}, Course)$ *and* $(\underline{Instructor}, \underline{Student})$

Note that if we designate (Student, Instructor) as a primary key of the relation TEACH, the FD Instructor→ Course causes a partial (non-full-functional) depend-ency of Course on a part of this key. This FD may be removed as a part of second normalization yielding exactly the same two relations in the result. This is an example of a case where we may reach the same ultimate BCNF design via alternate paths of normalization.