Database Management System (MDS 505)

Jagdish Bhatta

Unit-2

The Relational Languages and Relational Model: Relational Model and Constraints

Relational Model

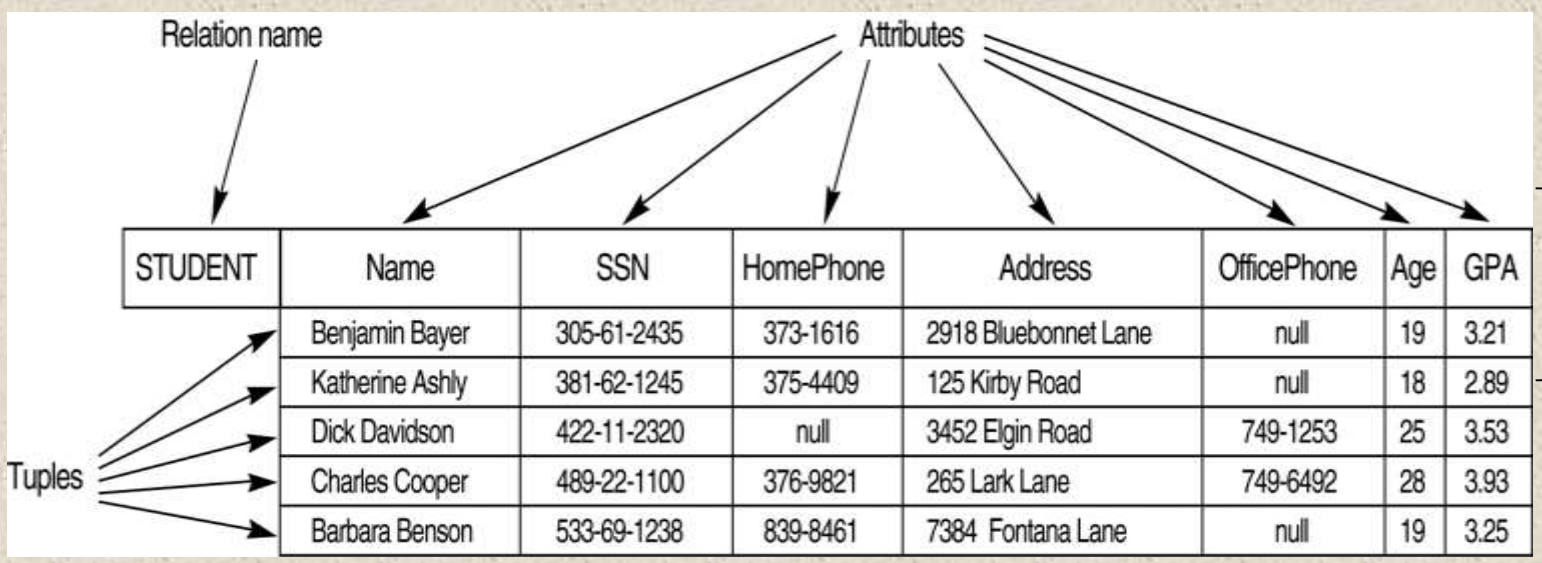
- Relational model represents the database as a collection of relations where each relation resembles a table of values with rows and columns. Major advantages of relational model over the older data models are the simple data representation & the ways complex queries can be expressed easily.
- When a relation is thought of as a **table** of values, each row in the table represents a collection of related data values. A row represents a fact that typically corresponds to a real-world entity or relationship. The table name and column names are used to help to interpret the meaning of the values in each row.

Relational Model

- In the formal relational model terminology, a row is called a *tuple*, a column header is called an *attribute*, and the table is called a *relation.* The data type describing the types of values that can appear in each column is represented by a *domain* of possible values.

Relational Model

Eg: Student (Name: string, SSN: varchar, HomePhone: varchar,..) } Rel^n Schema



STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25

} Rel^n Instance

Domains

- A **domain** *D* is a set of atomic values. By **atomic** we mean that each value in the domain is indivisible as far as the formal relational model is concerned. A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn. It is also useful to specify a name for the domain, to help in interpreting its values.
- **Data type** or **format** is also specified for each domain.
- Some examples of domains follow:
  - Names: The set of character strings that represent names of persons.
  - Grade_point_averages. Possible values of computed grade point averages; each must be a real (floating-point) number between 0 and 4.

Attributes

- A **relation schema** *R*, denoted by *R*(*A*1, *A*2, … , *An*), is made up of a relation name *R* and a list of attributes, *A*1, *A*2, … , *An*. Each **attribute** *Ai* is the name of a role played by some domain *D* in the relation schema *R*. *D* is called the **domain** of *Ai* and is denoted by **dom**(***A****i*). A relation schema is used to *describe* a relation; *R* is called the **name** of this relation. The **degree** (or **arity**) of a relation is the number of attributes *n* of its relation schema.
- A relation of degree seven, which stores information about university students, would contain seven attributes describing each student as follows:

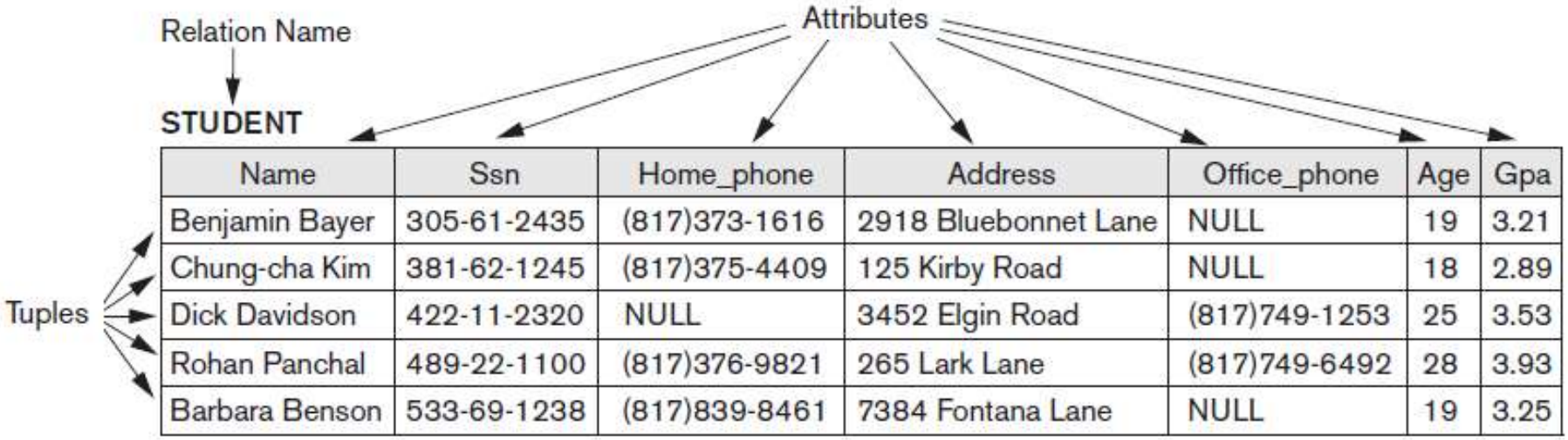
  STUDENT(Name, Ssn, Home_phone, Address, Office_phone, Age, Gpa)
- Using the data type of each attribute, the definition is sometimes written as:

  STUDENT(Name: string, Ssn: string, Home_phone: string, Address: string, Office_phone: string, Age: integer, Gpa: real)

Tuples and Relations

- **Relation** (or **relation state**) *r* of the relation schema *R*(*A*1, *A*2, … , *An*), also denoted by *r*(*R*), is a set of *n*-tuples *r* = {*t*1, *t*2, … , *tm*}. Each ***n*-tuple** *t* is an ordered list of *n* values *t* =<*v*1, *v*2, ... , *vn*>, where each value *vi*, $1 \leq i \leq n$, is an element of dom (*Ai*) or is a special NULL value.
- The *i*th value in tuple *t*, which corresponds to the attribute *Ai*, is referred to as *t*[*Ai*] or *t.Ai* (or *t*[*i*] if we use the positional notation). The terms **relation intension** for the schema *R* and **relation extension** for a relation state *r*(*R*) are also commonly used.
- Figure 5.1 in the next slide shows an example of a STUDENT relation, which corresponds to the STUDENT schema just specified. Each tuple in the relation represents a particular student entity (or object). We display the relation as a table, where each tuple is shown as a *row* and each attribute corresponds to a *column header* indicating a role or interpretation of the values in that column. *NULL values* represent attributes whose values are unknown
- or do not exist for some individual STUDENT tuple.

Attributes, Tuples, and Relations



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25

Figure 5.1
The attributes and tuples of a relation STUDENT.

Relations

- The earlier definition of a relation can be *restated* more formally using set theory concepts as follows. A relation (or relation state) $r(R)$ is a **mathematical relation** of degree n on the domains dom($A1$), dom($A2$), … , dom(An), which is a **subset** of the **Cartesian product** (denoted by ×) of the domains that define R:

$$r(R) \subseteq (\text{dom}(A1) \times \text{dom}(A2) \times \ldots \times (\text{dom}(An))$$

- The Cartesian product specifies all possible combinations of values from the underlying domains. Hence, if we denote the total number of values, or **cardinality,** in a domain D by $|D|$ (assuming that all domains are finite), the total number of tuples in the Cartesian product is

$$|\text{dom}(A1)| \times |\text{dom}(A2)| \times \ldots \times |\text{dom}(An)|$$

- This product of cardinalities of all domains represents the total number of possible instances or tuples that can ever exist in any relation state $r(R)$. Of all these possible combinations, a relation state at a given time—the **current relation state**—reflects only the valid tuples that represent a particular state of the real world. In general, as the state of the real world changes, so does the relation state, by being transformed into another relation state. However, the schema R is relatively static and changes *very* infrequently—for example, as a result of adding an attribute to represent new information that was not originally stored in the relation.

Characteristics of relations:

- **Ordering of Tuples in a Relation:** A relation is defined as a *set* of tuples. Mathematically, elements of a set have *no order* among them; hence, tuples in a relation do not have any particular order. In other words, a relation is not sensitive to the ordering of tuples. However, in a file, records are physically stored on disk (or in memory), so there always is an order among the records. This ordering indicates first, second, *i*th, and last records in the file. Similarly, when we display a relation as a table, the rows are displayed in a certain order. Tuple ordering is not part of a relation definition because a relation attempts to represent facts at a logical or abstract level.
- **Ordering of Values within a Tuple and an Alternative Definition of a Relation:** A relation schema $R = \{A1, A2, \ldots, An\}$ is a *set* of attributes (instead of an ordered list of attributes), and a relation state $r(R)$ is a finite set of mappings $r = \{t1, t2, \ldots, tm\}$, where each tuple ti is a **mapping** from R to D, and D is the **union** (denoted by $\cup$) of the attribute domains; that is, $D = \text{dom}(A1) \cup \text{dom}(A2) \cup \ldots \cup \text{dom}(An)$. In this definition, $t[Ai]$ must be in $\text{dom}(Ai)$ for $1 \leq i \leq n$ for each mapping t in r. Each mapping ti is called a tuple. According to this definition of tuple as a mapping, a **tuple** can be considered as a **set** of (<attribute>, <value>) pairs, where each pair gives the value of the mapping from an attribute Ai to a value vi from $\text{dom}(Ai)$.

Characteristics of relations:

- **Values and NULLs in the Tuples:** Each value in a tuple is an **atomic** value; that is, it is not divisible into components within the framework of the basic relational model. Hence, composite and multivalued attributes are not allowed. An important concept is that of NULL values, which are used to represent the values of attributes that may be unknown or may not apply to a tuple. A special value, called NULL, is used in these cases. In general, we can have several meanings for NULL values, such as ***value unknown***, **value** exists but is ***not available***, or ***attribute does not apply*** to this tuple *(*also known as ***value undefined****)*.
- **Interpretation (Meaning) of a Relation.** The relation schema can be interpreted as a declaration or a type of **assertion**. For example, the schema of the STUDENT relation of Figure 5.1, in slide number 9, asserts that, in general, a student entity has a Name, Ssn, Home_phone, Address, Office_phone, Age, and Gpa. Each tuple in the relation can then be interpreted as a **fact** or a particular instance of the assertion. For example, the first tuple in Figure 5.1 asserts the fact that there is a STUDENT whose Name is Benjamin Bayer, Ssn is 305-61-2435, Age is 19, and so on. An alternative interpretation of a relation schema is as a **predicate**; in this case, the values in each tuple are interpreted as values that *satisfy* the predicate. For example, the predicate STUDENT (Name, Ssn, …) is true for the five tuples in relation STUDENT of Figure 5.1.

Relational Model Notations

- **A** relation schema *R* of degree *n* is denoted by *R*(*A*1, *A*2, … , *An*).
- The uppercase letters *Q*, *R*, *S* denote relation names.
- The lowercase letters *q*, *r*, *s* denote relation states.
- The letters *t*, *u*, *v* denote tuples.
- In general, the name of a relation schema such as STUDENT also indicates the current set of tuples in that relation—the *current relation state*—whereas STUDENT(Name, Ssn, …) refers *only* to the relation schema.
- An attribute *A* can be qualified with the relation name *R* to which it belongs by using the dot notation *R.A*—for example, STUDENT.Name or STUDENT.Age. This is because the same name may be used for two attributes in different relations. However, all attribute names *in a particular relation* must be distinct.

Relational Model Notations

- An *n*-tuple *t* in a relation *r*(*R*) is denoted by *t* = <*v*1, *v*2, … , *vn*>, where *vi* is the value corresponding to attribute *Ai*. The following notation refers to **component values** of tuples:
  - Both *t*[*Ai*] and *t.Ai* (and sometimes *t*[*i*]) refer to the value *vi* in *t* for attribute *Ai*.
  - Both *t*[*Au*, *Aw*, … , *Az*] and *t*.(*Au*, *Aw*, … , *Az*), where *Au*, *Aw*, … , *Az* is a list of attributes from *R*, refer to the subtuple of values <*vu*, *vw*, … , *vz*> from *t* corresponding to the attributes specified in the list.
- As an example, consider the tuple *t* = <'Barbara Benson', '533-69-1238', '(817)839-8461', '7384 Fontana Lane', NULL, 19, 3.25> from the STUDENT relation in Figure 5.1, in slide number 9; we have *t*[Name] = <'Barbara Benson'>, and *t*[Ssn, Gpa, Age] = <'533-69-1238', 3.25, 19>.

Relational Model Constraints

- There are generally many restrictions or **constraints** on the actual values in a database state.
- Constraints on databases can generally be divided into three main categories:
  - Constraints that are inherent in the data model. We call these **inherent model-based constraints** or **implicit constraints**.
  - Constraints that can be directly expressed in the schemas of the data model, typically by specifying them in the DDL. We call these **schema-based constraints** or **explicit constraints**.
  - Constraints that *cannot* be directly expressed in the schemas of the data model, and hence must be expressed and enforced by the application programs or in some other way. We call these **application-based** or **semantic constraints** or **business rules**.

Relational Model Constraints

- There are generally many restrictions or **constraints** on the actual values in a database state.
- Constraints on databases can generally be divided into three main categories:
  - Constraints that are inherent in the data model. We call these **inherent model-based constraints** or **implicit constraints**.
  - Constraints that can be directly expressed in the schemas of the data model, typically by specifying them in the DDL. We call these **schema-based constraints** or **explicit constraints**.
  - Constraints that *cannot* be directly expressed in the schemas of the data model, and hence must be expressed and enforced by the application programs or in some other way. We call these **application-based** or **semantic constraints** or **business rules**.

Relational Model Constraints

- The schema-based constraints include domain constraints, key constraints, constraints on NULLs, entity integrity constraints, and referential integrity constraints.
- **Domain Constraints:** Domain constraints specify that within each tuple, the value of each attribute *A* must be an atomic value from the domain dom(*A*). The data types associated with domains typically include standard numeric data types for integers (such as short integer, integer, and long integer) and real numbers (float and double-precision float). Characters, Booleans, fixed-length strings, and variable-length strings are also available, as are date, time, timestamp, and other special data types. Domains can also be described by a subrange of values from a data type or as an enumerated data type in which all possible values are explicitly listed.

Relational Model Constraints

- **Key Constraints and Constraints on NULL Values:** In the formal relational model, a *relation* is defined as a *set of tuples*. By definition, all elements of a set are distinct; hence, all tuples in a relation must also be distinct.
- This means that no two tuples can have the same combination of values for *all* their attributes. Usually, there are other **subsets of attributes** of a relation schema *R* with the property that no two tuples in any relation state *r* of *R* should have the same combination of values for these attributes. Suppose that we denote one such subset of attributes by SK; then for any two *distinct* tuples *t*1 and *t*2 in a relation state *r* of *R*, we have the constraint that:

$$t1[\text{SK}] \neq t2[\text{SK}].$$

Relational Model Constraints

- Any such set of attributes SK is called a **superkey** of the relation schema *R*. A superkey SK specifies a *uniqueness constraint* that no two distinct tuples in any state *r* of *R* can have the same value for SK. Every relation has at least one default superkey— the set of all its attributes. A superkey can have redundant attributes, however, so a more useful concept is that of a *key*, which has no redundancy. A **key** *k* of a relation schema *R* is a superkey of *R* with the additional property that removing any attribute *A* from *K* leaves a set of attributes *K′* that is not a superkey of *R* any more. Hence, a key satisfies two properties:
  - Two distinct tuples in any state of the relation cannot have identical values for (all) the attributes in the key. This *uniqueness* property also applies to a superkey.
  - It is a *minimal superkey*—that is, a superkey from which we cannot remove any attributes and still have the uniqueness constraint hold. This *minimality* property is required for a key but is optional for a superkey.

Relational Model Constraints

- Any such set of attributes SK is called a **superkey** of the relation schema *R*. A superkey SK specifies a *uniqueness constraint* that no two distinct tuples in any state *r* of *R* can have the same value for SK. Every relation has at least one default superkey— the set of all its attributes. A superkey can have redundant attributes, however, so a more useful concept is that of a *key*, which has no redundancy. A **key** *k* of a relation schema *R* is a superkey of *R* with the additional property that removing any attribute *A* from *K* leaves a set of attributes *K′* that is not a superkey of *R* any more. Hence, a key satisfies two properties:
  - Two distinct tuples in any state of the relation cannot have identical values for (all) the attributes in the key. This *uniqueness* property also applies to a superkey.
  - It is a *minimal superkey*—that is, a superkey from which we cannot remove any attributes and still have the uniqueness constraint hold. This *minimality* property is required for a key but is optional for a superkey.

Relational Model Constraints

- Hence, a key is a superkey but not vice versa. A superkey may be a key (if it is minimal) or may not be a key (if it is not minimal). Consider the STUDENT relation of Figure 5.1, in slide number 9. The attribute set {Ssn} is a key of STUDENT because no two student tuples can have the same value for Ssn.8 Any set of attributes that includes Ssn—for example, {Ssn, Name, Age}—is a superkey. However, the superkey {Ssn, Name, Age} is not a key of STUDENT because removing Name or Age or both from the set still leaves us with a superkey. In general, any superkey formed from a single attribute is also a key. A key with multiple attributes must require *all* its attributes together to have the uniqueness property.

Relational Model Constraints

- The value of a key attribute can be used to identify uniquely each tuple in the relation.
- Notice that a set of attributes constituting a key is a property of the relation schema; it is a constraint that should hold on *every* valid relation state of the schema.
- A key is determined from the meaning of the attributes, and the property is *time-invariant:* It must continue to hold when we insert new tuples in the relation. For example, we cannot and should not designate the Name attribute of the STUDENT relation in Figure 5.1 as a key because it is possible that two students with identical names will exist at some point in a valid state.

Relational Model Constraints

- In general, a relation schema may have more than one key. In this case, each of the keys is called a **candidate key**. For example, the CAR relation in Figure 5.4, in slide number 23, has two candidate keys: License_number and Engine_serial_number. It is common to designate one of the candidate keys as the **primary key** of the relation. This is the candidate key whose values are used to *identify* tuples in the relation.
- The convention that the attributes that form the primary key of a relation schema are underlined is followed in general.
- When a relation schema has several candidate keys, the choice of one to become the primary key is somewhat arbitrary; however, it is usually better to choose a primary key with a single attribute or a small number of attributes. The other candidate keys are designated as **unique keys** and are not underlined

Relational Model Constraints

CAR

License_number	Engine_serial_number	Make	Model	Year
Texas ABC-739	A69352	Ford	Mustang	02
Florida TVP-347	B43696	Oldsmobile	Cutlass	05
New York MPO-22	X83554	Oldsmobile	Delta	01
California 432-TFY	C43742	Mercedes	190-D	99
California RSK-629	Y82935	Toyota	Camry	04
Texas RSK-629	U028365	Jaguar	XJS	04

Figure 5.4
The CAR relation, with two candidate keys: License_number and Engine_serial_number.

Relational Model Constraints

- Another constraint on attributes specifies whether NULL values are or are not permitted.

- For example, if every STUDENT tuple must have a valid, non-NULL value for the Name attribute, then Name of STUDENT is constrained to be NOT NULL.

Relational Databases and Relational Database Schemas

- A **relational database schema** *S* is a set of relation schemas *S* = {*R*1, *R*2, … , *Rm*} and a set of **integrity constraints** IC. A **relational database state** DB of *S* is a set of relation states DB = {*r*1, *r*2, … , *rm*} such that each *ri* is a state of *Ri* and such that the *ri* relation states satisfy the integrity constraints specified. Figure 5.5, in slide number 27, shows a relational database schema that we call COMPANY = {EMPLOYEE, DEPARTMENT, DEPT_LOCATIONS, PROJECT, WORKS_ON, DEPENDENT}. In each relation schema, the underlined attribute represents the primary key. The figure 5.6 shows a relational database state corresponding to the COMPANY schema.
- When we refer to a relational database, we implicitly include both its schema and its current state. A database state that does not obey all the integrity constraints is called **not valid**, and a state that satisfies all the constraints in the defined set of integrity constraints is called a **valid state.**

Relational Databases and Relational Database Schemas

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Figure 5.5
Schema diagram for the COMPANY relational database schema.

Relational Databases and Relational Database Schemas

- In some early versions of the relational model, an assumption was made that the same real-world concept, when represented by an attribute, would have *identical* attribute names in all relations. This creates problems when the same real-world concept is used in different roles (meanings) in the same relation. For example, the concept of Social Security number appears twice in the EMPLOYEE relation of Figure 5.5, in previous slide: once in the role of the employee's SSN, and once in the role of the supervisor's SSN. We are required to give them distinct attribute names—Ssn and Super_ssn, respectively—because they appear in the same relation and in order to distinguish their meaning.

Relation e Schemas

Figure 5.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Entity Integrity, Referential Integrity, and Foreign Keys

- The **entity integrity constraint** states that no primary key value can be NULL. This is because the primary key value is used to identify individual tuples in a relation. Having NULL values for the primary key implies that we cannot identify some tuples. For example, if two or more tuples had NULL for their primary keys, we may not be able to distinguish them if we try to reference them from other relations.
- Key constraints and entity integrity constraints are specified on individual relations. The **referential integrity constraint** is specified between two relations and is used to maintain the consistency among tuples in the two relations. Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an *existing tuple* in that relation. For example, in Figure 5.6, in the previous slide, the attribute Dno of EMPLOYEE gives the department number for which each employee works; hence, its value in every EMPLOYEE tuple must match the Dnumber value of some tuple in the DEPARTMENT relation.

Entity Integrity, Referential Integrity, and Foreign Keys

- To define *referential integrity* more formally, first we define the concept of a *foreign key*. The conditions for a foreign key, given below, specify a referential integrity constraint between the two relation schemas $R1$ and $R2$. A set of attributes FK in relation schema $R1$ is a **foreign key** of $R1$ that **references** relation $R2$ if it satisfies the following rules:
  - The attributes in FK have the same domain(s) as the primary key attributes PK of $R2$; the attributes FK are said to **reference** or **refer to** the relation $R2$.
  - A value of FK in a tuple $t1$ of the current state $r1(R1)$ either occurs as a value of PK for some tuple $t2$ in the current state $r2(R2)$ *or is NULL*. In the former case, we have $t1[FK] = t2[PK]$, and we say that the tuple $t1$ **references** or **refers to** the tuple $t2$.
- In this definition, $R1$ is called the **referencing relation** and $R2$ is the **referenced relation**. If these two conditions hold, a **referential integrity constraint** from $R1$ to $R2$ is said to hold. In a database of many relations, there are usually many referential integrity constraints.

Entity Integrity, Referential Integrity, and Foreign Keys

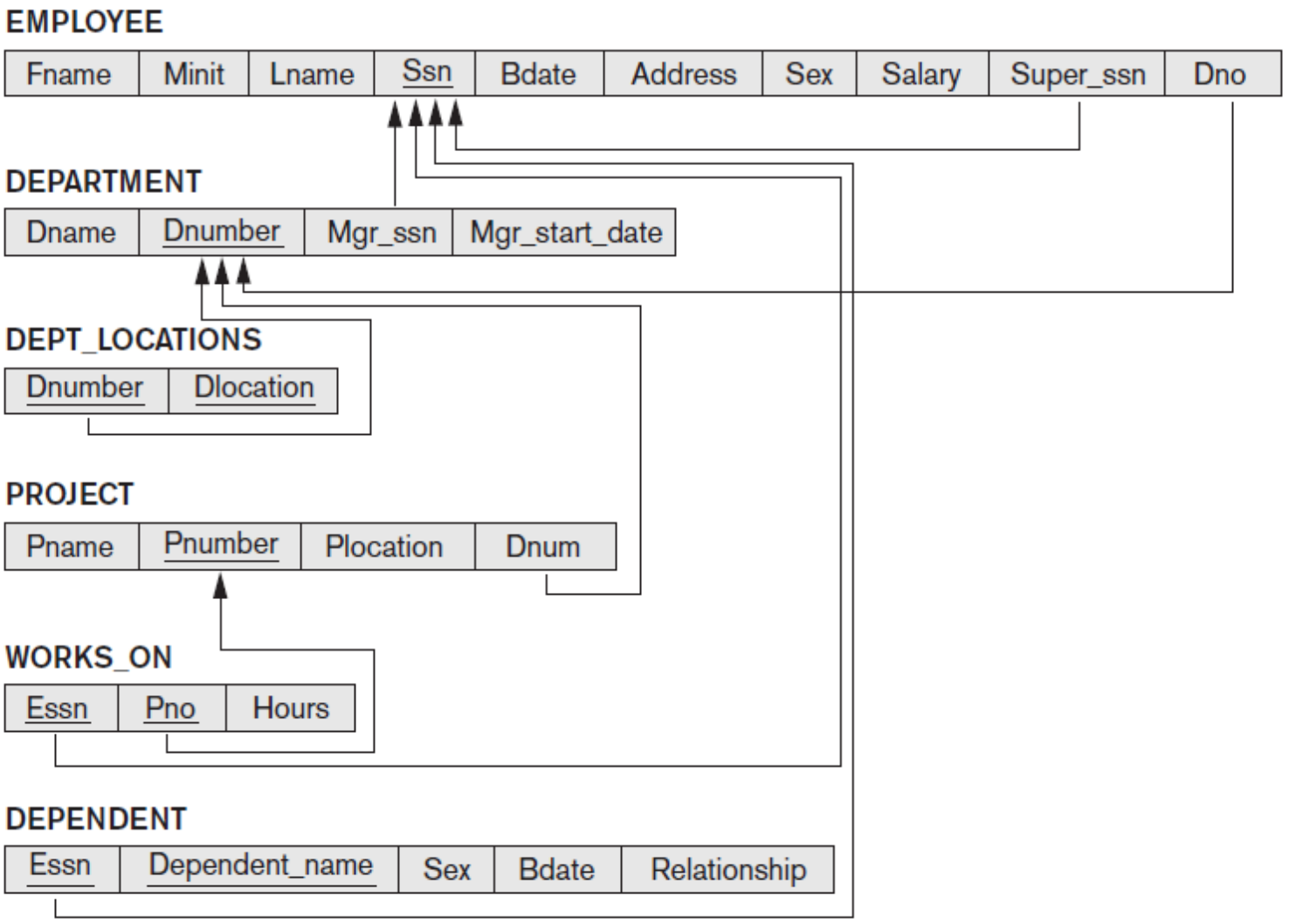
- Referential integrity constraints typically arise from the *relationships among the entities* represented by the relation schemas. For example, consider the database shown in Figure 5.6, slide number 29. In the EMPLOYEE relation, the attribute Dno refers to the department for which an employee works; hence, we designate Dno to be a foreign key of EMPLOYEE referencing the DEPARTMENT relation. This means that a value of Dno in any tuple *t*1 of the EMPLOYEE relation must match a value of the primary key of DEPARTMENT—the Dnumber attribute—in some tuple *t*2 of the DEPARTMENT relation, or the value of Dno *can be NULL* if the employee does not belong to a department or will be assigned to a department later. For example, in Figure 5.6, in slide number 29, the tuple for employee 'John Smith' references the tuple for the 'Research' department, indicating that 'John Smith' works for this department.

Entity Integrity, Referential Integrity, and Foreign Keys

- Notice that a foreign key can *refer to its own relation.* For example, the attribute Super_ssn in EMPLOYEE refers to the supervisor of an employee; this is another employee, represented by a tuple in the EMPLOYEE relation. Hence, Super_ssn is a foreign key that references the EMPLOYEE relation itself. In Figure 5.6, in the slide number 29, the tuple for employee 'John Smith' references the tuple for employee 'Franklin Wong,' indicating that 'Franklin Wong' is the supervisor of 'John Smith'.
- We can *diagrammatically display referential integrity constraints* by drawing a directed arc from each foreign key to the relation it references. For clarity, the arrowhead may point to the primary key of the referenced relation. Figure 5.7, in next slide, shows the schema in Figure 5.5, in slide number 27, with the referential integrity constraints displayed in this manner.

Entity Integrity, Referential Integrity, and Foreign Keys

Figure 5.7
Referential integrity constraints displayed on the COMPANY relational database schema.



Entity Integrity, Referential Integrity, and Foreign Keys

- All integrity constraints should be specified on the relational database schema (that is, specified as part of its definition) if we want the DBMS to enforce these constraints on the database states. Hence, the DDL includes provisions for specifying the various types of constraints so that the DBMS can automatically enforce them.
- In SQL, the CREATE TABLE statement of the SQL DDL allows the definition of primary key, unique key, NOT NULL, entity integrity, and referential integrity constraints, among other constraints

Other Types of Constraints

- The preceding integrity constraints are included in the data definition language because they occur in most database applications. Another class of general constraints, sometimes called *semantic integrity constraints*, are not part of the DDL and have to be specified and enforced in a different way. Examples of such constraints are *the salary of an employee should not exceed the salary of the employee's supervisor* and *the maximum number of hours an employee can work on all projects per week is 56*. Such constraints can be specified and enforced within the application programs that update the database, or by using a general-purpose **constraint specification language**. Mechanisms called **triggers** and **assertions** can be used in SQL, through the CREATE ASSERTION and CREATE TRIGGER statements, to specify some of these constraints. It is more common to check for these types of constraints within the application programs than to use constraint specification languages because the latter are sometimes difficult and complex to use.

Other Types of Constraints

- The types of constraints we discussed so far may be called **state constraints** because they define the constraints that a *valid state* of the database must satisfy.
- Another type of constraint, called **transition constraints**, can be defined to deal with state changes in the database. An example of a transition constraint is: “the salary of an employee can only increase.” Such constraints are typically enforced by the application programs or specified using active rules and triggers.

Database Modification and Dealing with Constraint Violations

- There are three basic operations that can change the states of relations in the database: Insert, Delete, and Update (or Modify). They insert new data, delete old data, or modify existing data records, respectively.
- **Insert** is used to insert one or more new tuples in a relation.
- **Delete** is used to delete tuples.
- **Update** (or **Modify**) is used to change the values of some attributes in existing tuples.
- Whenever these operations are applied, the integrity constraints specified on the relational database schema should not be violated.

Database Modification and Dealing with Constraint Violations

- **The Insert Operation:**
- The **Insert** operation provides a list of attribute values for a new tuple *t* that is to be inserted into a relation *R*. Insert can violate any of the four types of constraints.
  - Domain constraints can be violated if an attribute value is given that does not appear in the corresponding domain or is not of the appropriate data type.
  - Key constraints can be violated if a key value in the new tuple *t* already exists in another tuple in the relation *r*(*R*).
  - Entity integrity can be violated if any part of the primary key of the new tuple *t* is NULL. Referential integrity can be violated if the value of any foreign key in *t* refers to a tuple that does not exist in the referenced relation.

Database Modification and Dealing with Constraint Violations

- **The Insert Operation:**
- If an insertion violates one or more constraints, the default option is to *reject the insertion.* In this case, it would be useful if the DBMS could provide a reason to the user as to why the insertion was rejected.

Database Modification and Dealing with Constraint Violations

- **The Delete Operation:**
- The **Delete** operation can violate only referential integrity. This occurs if the tuple being deleted is referenced by foreign keys from other tuples in the database. To specify deletion, a condition on the attributes of the relation selects the tuple (or tuples) to be deleted.
- Several options are available if a deletion operation causes a violation. The first option, called **restrict,** is to *reject the deletion.* The second option, called **cascade,** is to *attempt to cascade (or propagate) the deletion* by deleting tuples that reference the tuple that is being deleted. A third option, called **set null** or **set default,** is to *modify the referencing attribute values* that cause the violation; each such value is either set to NULL or changed to reference another default valid tuple. Notice that if a referencing attribute that causes a violation is *part of the primary key*, it *cannot* be set to NULL; otherwise, it would violate entity integrity.

Database Modification and Dealing with Constraint Violations

- **The Update Operation:**
- The **Update** (or **Modify**) operation is used to change the values of one or more attributes in a tuple (or tuples) of some relation *R*. It is necessary to specify a condition on the attributes of the relation to select the tuple (or tuples) to be modified.
- Updating an attribute that is *neither part of a primary key nor part of a foreign key* usually causes no problems; the DBMS need only check to confirm that the new value is of the correct data type and domain. Modifying a primary key value is similar to deleting one tuple and inserting another in its place because we use the primary key to identify tuples. Hence, the issues discussed earlier for Insert and Delete come into play.

Relational Database Design Using ER-to-Relational Mapping

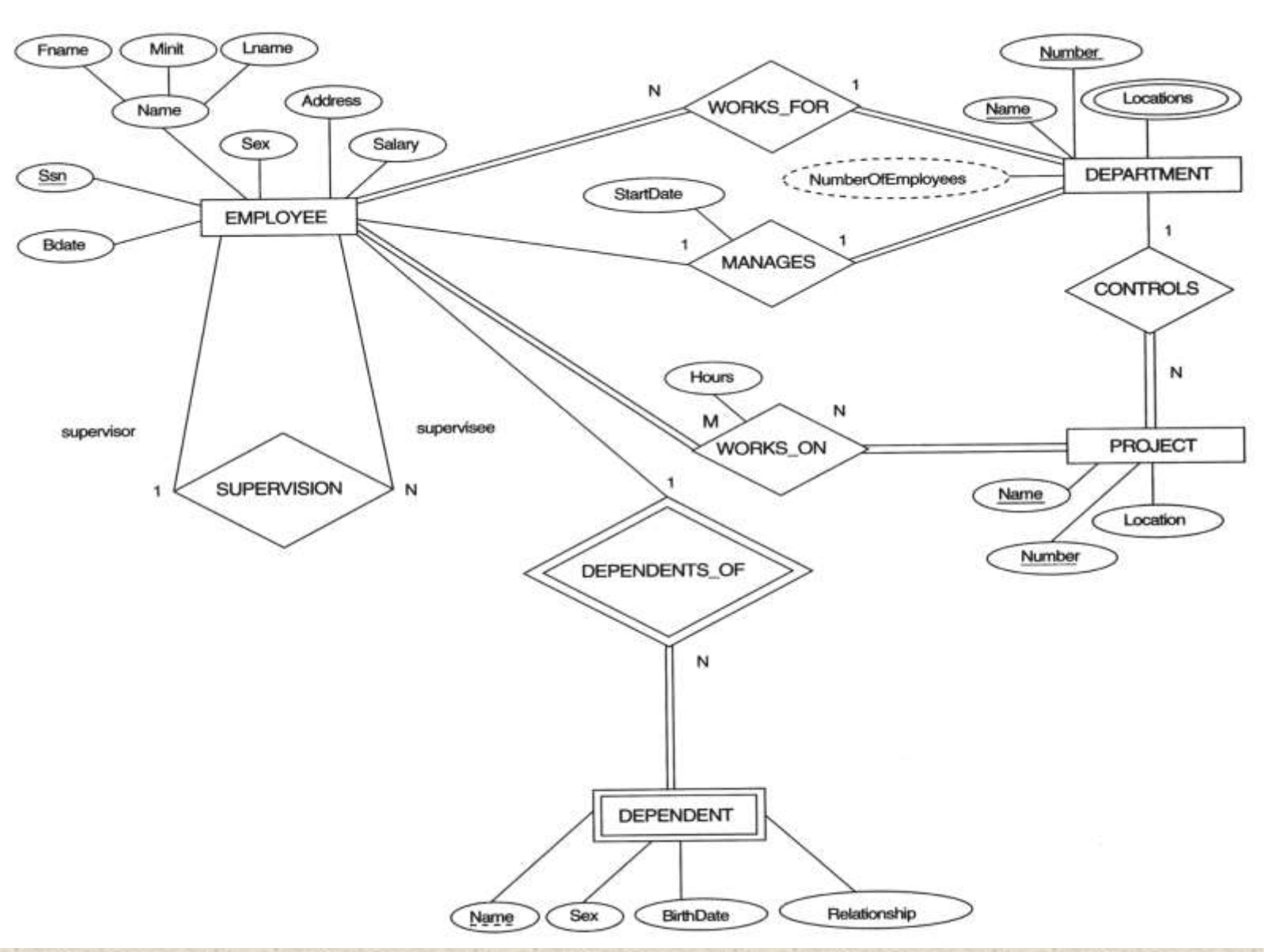
- It includes how to *design a relational database schema* based on a conceptual schema design. Here, we focus on the **logical database design** step of database design, which is also known as **data model mapping**. We present the procedures to create a relational schema from an entity–relationship (ER).
- It includes a seven-step algorithm to convert the basic ER model constructs—entity types (strong and weak), binary relationships (with various structural constraints), *n*-ary relationships, and attributes (simple, composite, and multivalued)—into relations

Reduction of E-R Schema to Tables

- As a part of database design, representing the logical structure of database with the relational tables needs conversion of the conceptual ER-representation into the tables. This conversion is the mapping of ER-schema into the tables.
- Reduction of E-R schema into tables include following algorithmic steps:

Step 1: Mapping of Regular Entity Types
Step 2: Mapping of Weak Entity Types
Step 3: Mapping of Binary 1:1 Relation Types
Step 4: Mapping of Binary 1:N Relationship Types.
Step 5: Mapping of Binary M:N Relationship Types.
Step 6: Mapping of Multivalued attributes.
Step 7: Mapping of N-ary Relationship Types.

ER to Relational Mapping: An Example



Step1: Mapping of Regular Entity Types:

- For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E.
- Include only simple component attributes of a composite attribute.
- Choose one of the key attributes of E as the primary key for R. If the chosen key of E is composite, the set of simple attributes that form it will together form the primary key of R.

Step2: Mapping of Weak Entity Types:

- For each weak entity type W in the ER schema with owner entity type E, create a relation R and include all simple attributes (or simple components of composite attributes) of W as attributes of R.
- In addition, include as foreign key attributes of R the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s).
- The primary key of R is the *combination of* the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any. If there is a weak entity type *E*2 whose owner is also a weak entity type *E*1, then *E*1 should be mapped before *E*2 to determine its primary key first.

Step3: Mapping of binary 1:1 Relationship Types:

- For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R. There are three possible approaches:

  (1) Foreign Key approach: Choose one of the relations, say, S and include the primary key of T as a foreign key in S. To choose role of S, it is better to choose an entity type with *total participation* in the relationship R. Include attributes of the relationship type, if any, as attributes of S. This approach is the most useful and should be followed unless special conditions exist.

  Example: 1:1 relation MANAGES is mapped by choosing the participating entity type DEPARTMENT to serve in the role of S, because its participation in the MANAGES relationship type is total. We include the primary key of the EMPLOYEE relation as foreign key in the DEPARTMENT relation and rename it to Mgr_ssn. We also include the simple attribute Start_date of the MANAGES relationship type in the DEPARTMENT relation and rename it Mgr_start_date

Step3: Mapping of binary 1:1 Relationship Types:

(2) Merged relation option: An alternate mapping of a 1:1 relationship type is possible by merging the two entity types and the relationship into a single relation. This may be appropriate when *both participations are total*.

(3) Cross-reference or relationship relation option: The third alternative is to set up a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types. As we will see, this approach is required for binary M:N relationships. The relation *R* is called a **relationship relation** (or sometimes a **lookup table**), because each tuple in *R* represents a relationship instance that relates one tuple from *S* with one tuple from *T*. The relation *R* will include the primary key attributes of *S* and *T* as foreign keys to *S* and *T*. The primary key of *R* will be one of the two foreign keys, and the other foreign key will be a unique key of *R*. The drawback is having an extra relation, and requiring extra join operations when combining related tuples from the tables.

Step4: Mapping of binary 1:N Relationship Types:

- There are two possible approaches: (1) the foreign key approach and (2) the cross-reference or relationship relation approach. The first approach is generally preferred as it reduces the number of tables.
- **The foreign key approach:** For each regular binary 1:N relationship type *R*, identify the relation *S* that represents the participating entity type at the *N-side* of the relationship type. Include as foreign key in *S* the primary key of the relation *T* that represents the other entity type participating in *R*; we do this because each entity instance on the N-side is related to at most one entity instance on the 1-side of the relationship type. Include any simple attributes (or simple components of composite attributes) of the 1:N relationshiptype as attributes of *S*.

Step4: Mapping of binary 1:N Relationship Types:

- **The relationship relation approach:** An alternative approach is to use the **relationship relation** (cross-reference) option as in the third option for binary 1:1 relationships. We create a separate relation *R* whose attributes are the primary keys of *S* and *T*, which will also be foreign keys to *S* and *T*. The primary key of *R* is the same as the primary key of *S*. This option can be used if few tuples in *S* participate in the relationship to avoid excessive NULL values in the foreign key.
- For a 1:N relationship, the primary key of the relationship relation will be the foreign key that references the entity relation on the N-side.

Step5: Mapping of binary M:N Relationship Types:

- In the traditional relational model with no multivalued attributes, the only option for M:N relationships is the **relationship relation (cross-reference) option**. For each binary M:N relationship type *R*, create a new relation *S* to represent *R*. Include as foreign key attributes in *S* the primary keys of the relations that represent the participating entity types; their *combination* will form the primary key of *S*. Also include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of *S*. Notice that we cannot represent an M:N relationship type by a single foreign key attribute in one of the participating relations (as we did for 1:1 or 1:N relationship types) because of the M:N cardinality ratio; we must create a separate *relationship relation S*.

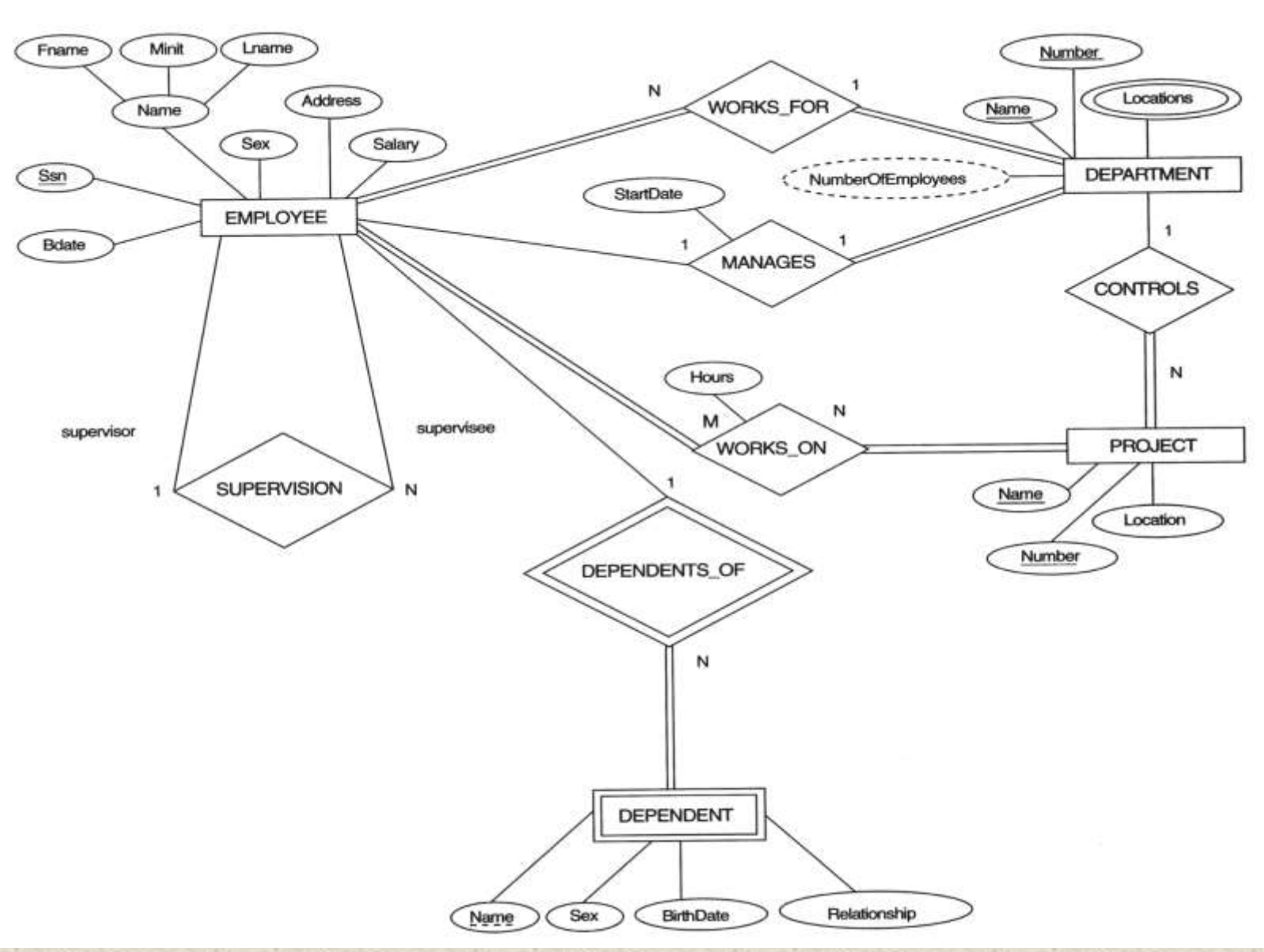
Step6: Mapping of Multivalued Attributes:

- For each multivalued attribute A, create a new relation R. This relation R will include an attribute corresponding to A, as well as the primary key attribute K of the relation that represents the entity type of relationship type that has A as an attribute as a foreign key in R.
- The primary key of R is the combination of A and K. If the multivalued attribute is composite, we include its simple components.

Step7: Mapping of N-ary Relationship Types:

- We use the **relationship relation option**. For each *n*-ary relationship type *R*, where $n > 2$, create a new relationship relation *S* to represent *R*. Include as foreign key attributes in *S* the primary keys of the relations that represent the participating entity types. Also include any simple attributes of the *n*-ary relationship type (or simple components of composite attributes) as attributes of *S*. The primary key of *S* is usually a combination of all the foreign keys that reference the relations representing the participating entity types. However, if the cardinality constraints on any of the entity types *E* participating in *R* is 1, then the primary key of *S* should not include the foreign key attribute that references the relation *E′* corresponding to *E*

ER to Relational Mapping: An Example



ER to Relational Mapping: An Example

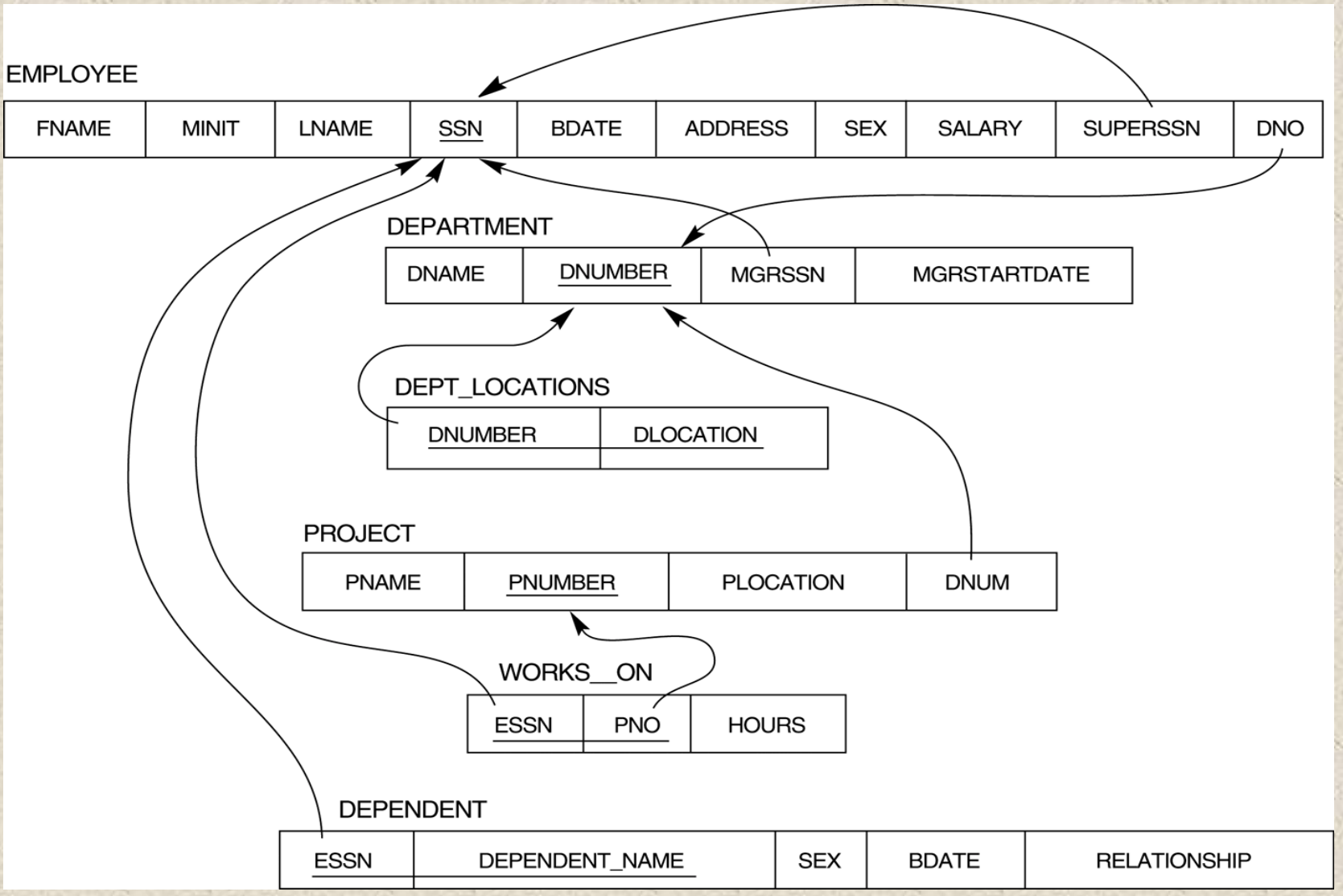


Fig: Result of mapping the COMPANY ER schema into a relational schema.

Mapping of N-ary relationship

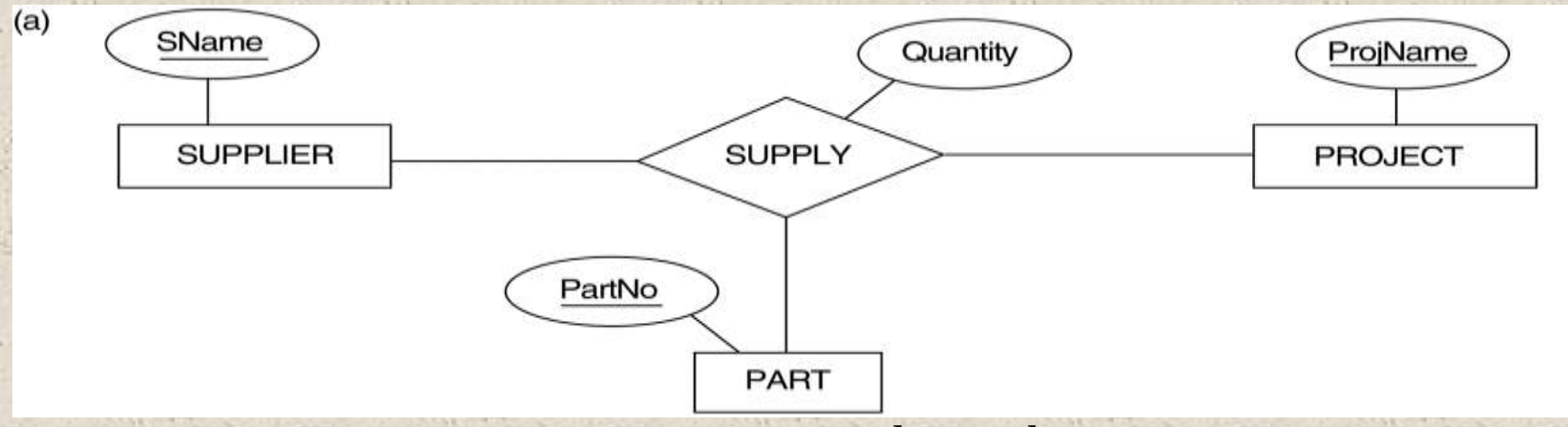


Fig: A Ternary relationship

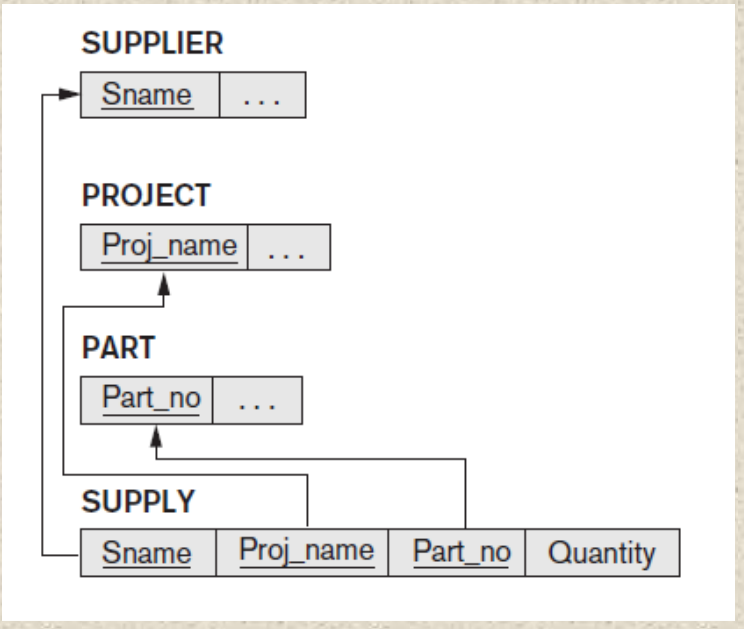


Fig: Result of ER to Relational Mapping of the n-ary relationship type

Table 9.1 Correspondence between ER and Relational Models

ER MODEL	RELATIONAL MODEL
Entity type	*Entity* relation
1:1 or 1:N relationship type	Foreign key (or *relationship* relation)
M:N relationship type	*Relationship* relation and *two* foreign keys
n-ary relationship type	*Relationship* relation and *n* foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Value set	Domain
Key attribute	Primary (or secondary) key