

MODULE II

RELATIONAL MODEL

STRUCTURE OF RELATIONAL DATABASES

The relational model represents the database as a collection of relations.

When a relation is thought of as a table of values, each row in the table represents a collection of related data values.

In the relational model, each row in the table represents a fact that typically corresponds to a real-world entity or relationship.

Domains, Attributes, Tuples, and Relations

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 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. Some examples of domains follow:

- Social_security_numbers: The set of valid nine-digit social security numbers.
- Names: The set of character strings that represent names of persons.
- Employee_ages: Possible ages of employees of a company; each must be a value between 15 and 80 years old.

A **data type or format** is also specified for each domain. For example, the data type for the domain phone_numbers can be declared as a character string of the form (ddd)ddd-dddd, where

each d is a numeric (decimal) digit and the first three digits form a valid telephone area code. The data type for Employee_ages is an integer number between 15 and 80.

A relation schema/ R , denoted by $R(A_1, A_2, \dots, A_n)$ is made up of a relation name R and a list of attributes A_1, A_2, \dots, A_n .

Each attribute A_i is the name of a role played by some domain D in the relation schema R . D is called the domain of A_i and is denoted by $\text{dom}(A_i)$. 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.

An example of a relation schema for a relation of degree seven, which describes university students, is the following:

STUDENT(Name, SSN, HomePhone, Address, OfficePhone, Age, GPA)

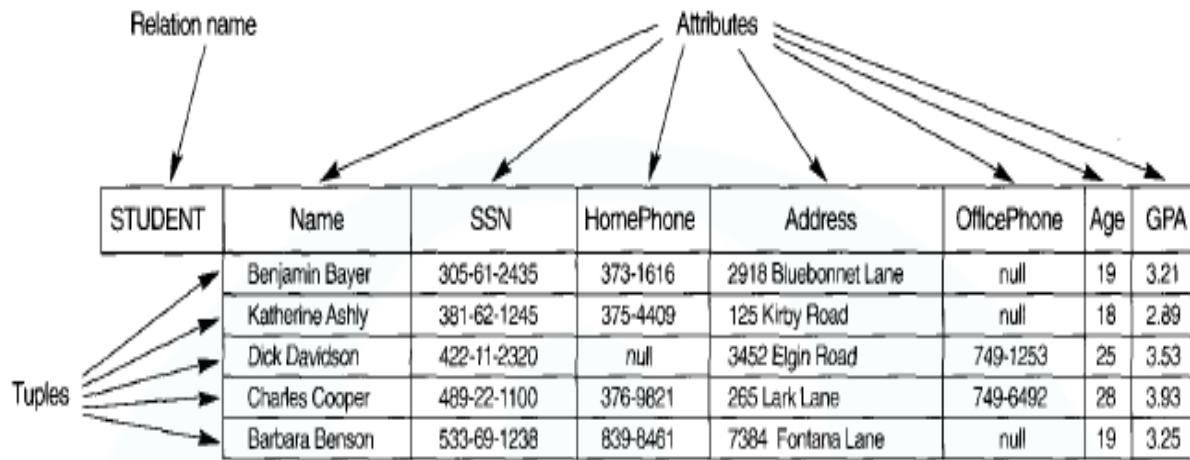
Using the **data type** of each attribute, the definition is sometimes written as:

STUDENT(Name: string, SSN: string, HomePhone: string, Address: string, OfficePhone: string, Age: integer, GPA: real)

For this relation schema, STUDENT is the name of the relation, which has seven attributes.

A relation (or relation state) r of the relation schema $R(A_1, A_2, \dots, A_n)$ also denoted by $r(R)$, is a set of n -tuples $r = \{t_1, t_2, \dots, t_m\}$. Each n -tuple t is an ordered list of n values $t = \langle v_1, v_2, \dots, v_n \rangle$.

The terms **relation intension** for the schema R and **relation extension** for a relation state $r(R)$.



Characteristics of Relations

(1) 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.

(2) Ordering of Values within a Tuple, and an Alternative Definition of a Relation.

According to the preceding definition of a relation, an n-tuple is an ordered list of n values, so the ordering of values in a tuple-and hence of attributes in a relation schema-is important

According to this definition of tuple as a mapping, a tuple can be considered as a set of ($\langle \text{attribute} \rangle$, $\langle \text{value} \rangle$) pairs, where each pair gives the value of the mapping from an attribute A_i to a value V_i from $\text{dom}(A_i)$.

(3) 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 nulls, 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 for these cases. For example, some student tuples have null for their office phones because they do not have an office number.

INTEGRITY CONSTRAINTS

Constraints on databases can generally be divided into three main categories:

1. Constraints that are inherent in the data model. We call these **inherent model based constraints**.

The characteristics of relations that we discussed above are the inherent constraints of the relational model .

2. 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**.

3. Constraints that cannot be directly expressed in the schemas of the data model, and hence must be expressed and enforced by the application programs. We call these **application-based constraints**.

The **schema-based constraints** include domain constraints, key constraints, constraints on nulls, entity integrity constraints, and referential integrity constraints.

(a)Domain Constraints

Domain constraints specify that within each tuple, the value of each attribute A must be an atomic value from the domain $\text{dom}(A)$.

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, in some cases, money data types.

(b) Key Constraints and Constraints on Null Values

A key satisfies two constraints:

1. Two distinct tuples in any state of the relation cannot have identical values for (all) the attributes in the key.
2. It is a minimal superkey-that is, a superkey from which we cannot remove any attributes and still have the uniqueness constraint in condition 1 hold.

Any superkey formed from a single attribute is also a key. Eg: SSN alone from {ssn,Name,Age} is a superkey as well as a key.

A key with multiple attributes must require all its attributes to have the uniqueness property hold.

Eg: {Name,Age}.(i.e:On removing Name, Age alone can't act as a key.)

Another constraint on attributes specifies whether null values are or are not permitted. For example, if every STUDENT tuple must have a valid, nonnull value for the Nameattribute, then Name of STUDENT is constrained to be NOT NULL.

(c) The entity integrity constraint

It 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 might not be able to distinguish them if we tried to reference them from other relations.

(d) Referential integrity constraint

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, 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.

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	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	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

To define referential integrity more formally, we first 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 (Foreign Key) in relation schema R1 is a foreign key of R1 that references relation R2 if it satisfies the following two rules:

1. The attributes in FK have the **same domains** as the **primary key attributes PK of R2**; the attributes FK are said to reference or refer to the relation R2.

2. A **value of FK in a tuple t1** of the current state $r1(R1)$ either **occurs as a value of PK(Primary Key) 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.

EXAMPLE-

Consider 2 relations "stu" and "stu_1" Where "Stu_id " is the primary key in the "stu" relation and foreign key in the "stu_1" relation.

Relation "stu"

Stu_id	Name	Branch
11255234	Aman	CSE
11255369	Kapil	EcE
11255324	Ajay	ME
11255237	Raman	CSE
11255678	Aastha	ECE

Relation "stu_1"

Stu_id	Course	Duration
11255234	B TECH	4 years
11255369	B TECH	4 years
11255324	B TECH	4 years
11255237	B TECH	4 years
11255678	B TECH	4 years

Rule 1. You can't **delete** any of the rows in the "stu" relation that are visible since all the "stu" are in use in the "stu_1" relation.

Rule 2. You can't **change** any of the "Stu_id" in the "stu" relation since all the "Stu_id" are in use in the "stu_1" relation.

Rule 3 You can enter a null value in the "stu_1" relation if the records are unrelated.

Domain constraints can be violated if an attribute value is given that does not appear in the corresponding domain.

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 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.

SYNTHESIZING ER DIAGRAM TO RELATIONAL SCHEMA

ER-to-Relational Mapping Algorithm

We will use the COMPANY database example to illustrate the mapping procedure. The COMPANY ER schema is shown in Figure 7.1, and the corresponding COMPANY relational database schema is shown in Figure 7.2 to illustrate the mapping steps.

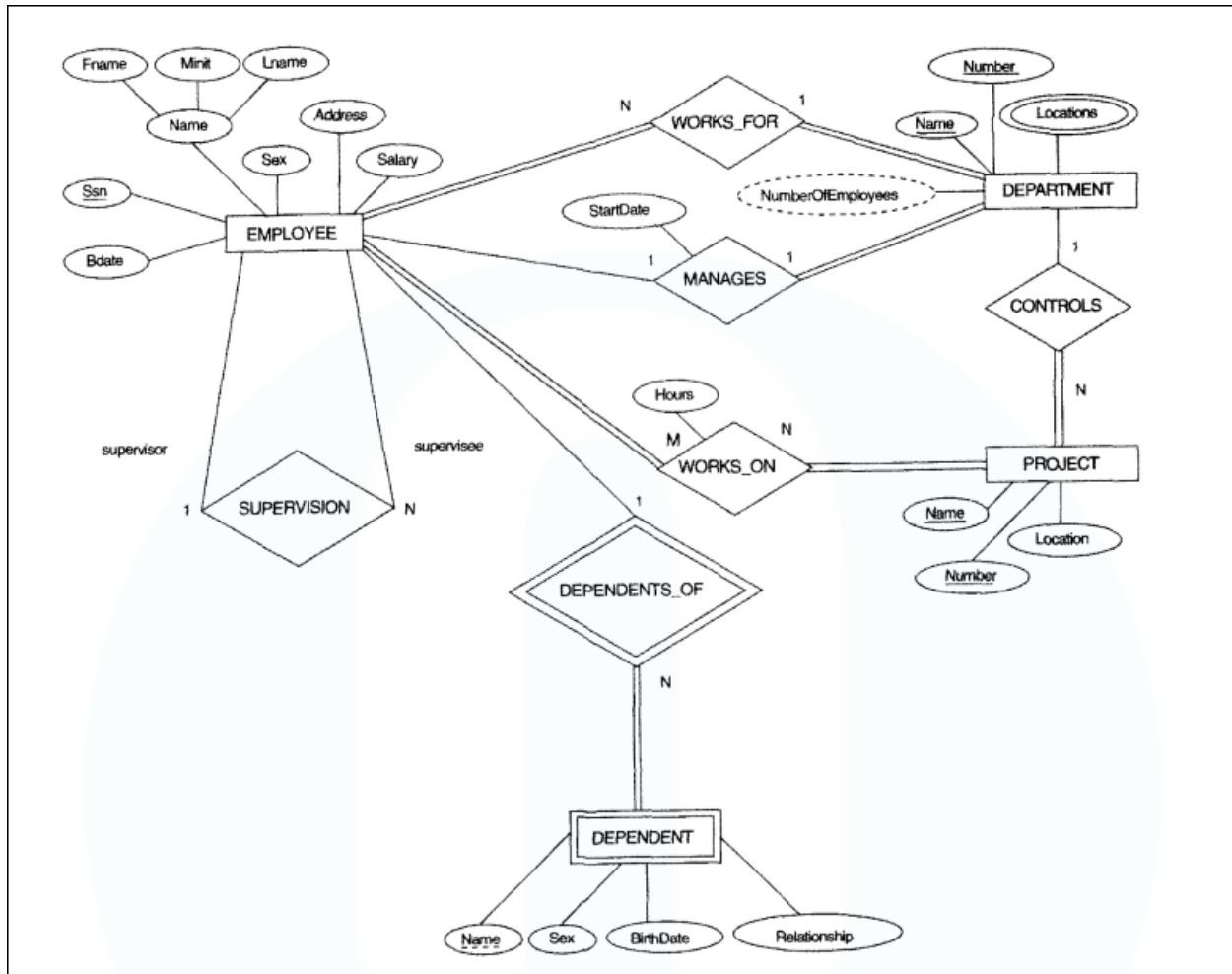
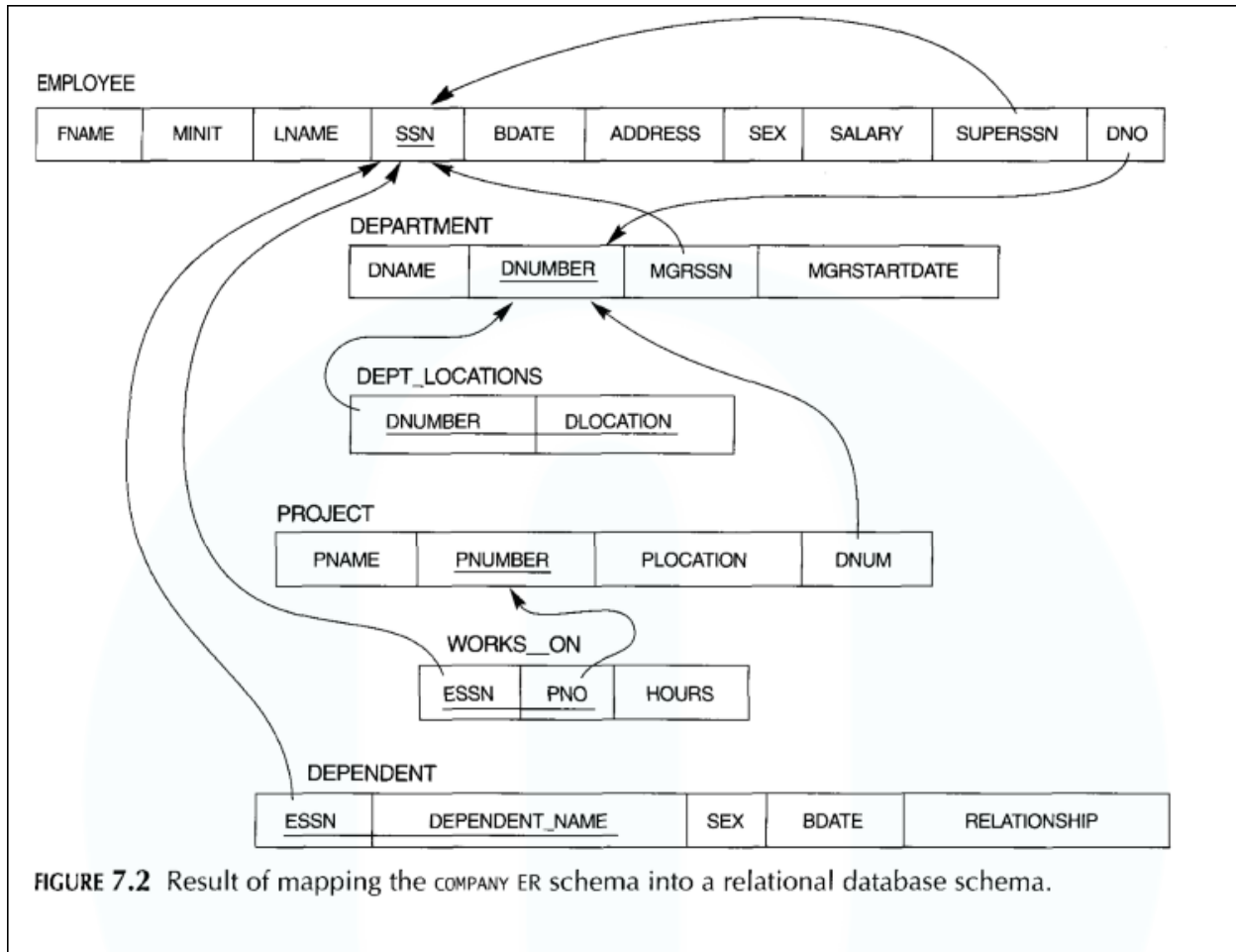


FIGURE 7.1 The ER conceptual schema diagram for the COMPANY database



Step 1: Mapping of Regular Entity Types.

Step 2: Mapping of Weak Entity Types

Step 3: Mapping of Binary 1:1 Relationship 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.

Steps in detail

Step 1: 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 the simple component attributes of a composite attribute.** Choose one of the **key attributes of E as 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.

In our example, we create the relations EMPLOYEE, DEPARTMENT, and PROJECT in Figure 7.2 to correspond to the regular entity types EMPLOYEE, DEPARTMENT, and PROJECT from Figure 7.1. The foreign key and relationship attributes, if any, are not included yet; they will be added during subsequent steps. These include the attributes SUPERSSN and DNO of EMPLOYEE, MGRSSN and MGRSTARTDATE of DEPARTMENT, and DNUM of PROJECT.

In our example, we choose SSN, DNUMBER, and PNUMBER as primary keys for the relations EMPLOYEE, DEPARTMENT, and PROJECT, respectively.

Step 2: 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 attribute of W as attributes of R.** In addition, **include as foreign key attributes of R the primary key attribute of the relation that correspond to the owner entity type:** this takes care of the identifying relationship type of W **The primary key of R is the combination of the primary key of the owner and the partial key of the weak entity type W.**

In our example, we create the relation DEPENDENT in this step to correspond to the weak entity type DEPENDENT. We include the primary key SSN of the EMPLOYEE relation-which corresponds to the owner entity type-as a foreign key attribute of DEPENDENT; we renamed it ESSN, although this is not necessary. The primary key of the DEPENDENT relation is the

combination {ESSN, DEPENDENT_NAME} because DEPENDENT_NAME is the partial key of DEPENDENT.

Step 3: 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.

Choose one of the relations-S, say-and include as a **foreign key in S the primary key of T**. It is better to choose an entity type with total participation in R in the role of S. Include all the simple attributes (or simple components of composite attributes) of the 1:1 relationship type R as attributes of S.

In our example, we map the 1:1 relationship type MANAGES from Figure 7.1 by choosing the participating entity type DEPARTMENT to serve in the role of S, because its participation in the MANAGES relationship type is total (every department has a manager). We include the primary key of the EMPLOYEE relation as foreign key in the DEPARTMENT relation and rename it MGRSSN. We also include the simple attribute STARTDATE of the MANAGES relationship type in the DEPARTMENT relation and rename it MGRSTARTDATE.

Step 4: Mapping of Binary 1 :N Relationship Types.

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; this is done 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 relationship type as attributes of S.

In our example, we now map the 1:N relationship types WORKS_FOR, CONTROLS, and SUPERVISION from Figure 7.1. For WORKS_FOR we include the primary key DNUMBER of the DEPARTMENT relation as foreign key in the EMPLOYEE relation and call it DNO. For

SUPERVISION we include the primary key of the EMPLOYEE relation as foreign key in the EMPLOYEE relation itself because the relationship is recursive-and call it SUPERSSN. The CONTROLS relationship is mapped to the foreign key attribute DNUM of PROJECT, which references the primary key DNUMBER of the DEPARTMENT relation.

Step 5: Mapping of Binary M:N Relationship Types.

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.

In our example, we map the M:N relationship type WORKS_ON from Figure 7.1 by creating the relation WORKS_ON in Figure 7.2. We include the primary keys of the PROJECT and EMPLOYEE relations as foreign keys in WORKS_ON and rename them PNO and ESSN, respectively. We also include an attribute HOURS in WORKS_ON to represent the HOURS attribute of the relationship type. The primary key of the WORKS_ON relation is the combination of the foreign key attributes {ESSN, PNO}.

Step 6: Mapping of Multivalued Attributes.

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

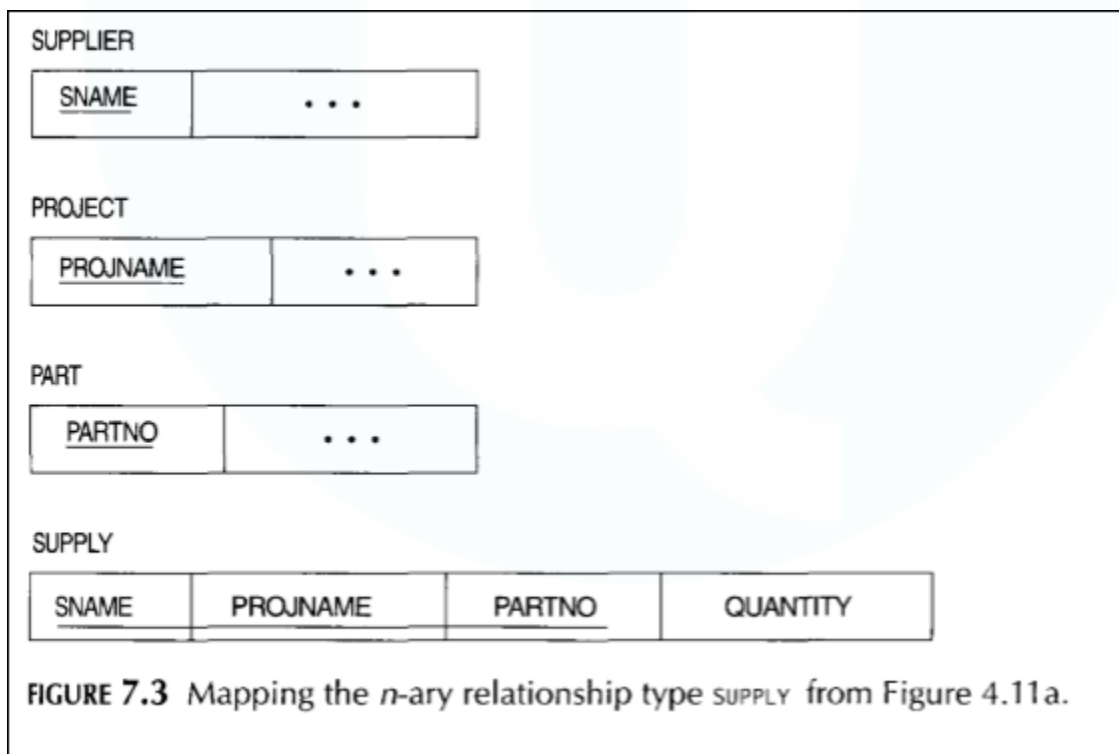
In our example, we create a relation DEPT_LOCATIONS. The attribute DLOCATION represents the multivalued attribute LOCATIONS of DEPARTMENT, while DNUMBER-as

foreign key represents the primary key of the DEPARTMENT relation. The primary key of DEPT_LOCATIONS is the combination of {DNUMBER, DLOCATION}. A separate tuple will exist in DEPT_LOCATIONS for each location that a department has.

Step 7: Mapping of N-ary Relationship Types.

For each n-ary relationship type R, where $n > 2$, 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. 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.

For example, consider the relationship type SUPPLY of Figure 4.11a. This can be mapped to the relation SUPPLY shown in Figure 7.3, whose primary key is the combination of the three foreign keys {SNAME, PARTNO, PROJNAME}.



RELATIONAL ALGEBRA**Figure 3.6**

One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	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	<u>Dnumber</u>	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

<u>Dnumber</u>	<u>Dlocation</u>
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

<u>Essn</u>	<u>Pno</u>	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

<u>Pname</u>	<u>Pnumber</u>	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

<u>Essn</u>	<u>Dependent_name</u>	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

RELATIONAL ALGEBRA

The SELECT Operation

The SELECT operation is used to select a subset of the tuples from a relation that satisfy a selection condition. One can consider the SELECT operation to be a filter that keeps only those tuples that satisfy a qualifying condition. The SELECT operation can also be visualized as a horizontal partition of the relation into two sets of tuples-those tuples that satisfy the condition and are selected, and those tuples that do not satisfy the condition and are discarded.

For example, to select the EMPLOYEE tuples whose department is 4,

or those whose salary is greater than \$30,000, we can individually specify each of these two conditions with a SELECT operation as follows:

$$\sigma_{DNO=4}(EMPLOYEE)$$

$$\sigma_{SALARY>30000}(EMPLOYEE)$$

In general, the SELECT operation is denoted by

$$\sigma_{\langle \text{selection condition} \rangle}(R)$$

where the symbol σ (sigma) is used to denote the SELECT operator, and the selection condition is a Boolean expression specified on the attributes of relation R. Notice that R is generally a relational algebra expression whose result is a relation-the simplest such expression is just the name of a database relation

The Boolean expression specified in $\langle \text{selection condition} \rangle$ is made up of a number of clauses of the form

$\langle \text{attribute name} \rangle \langle \text{comparison op} \rangle \langle \text{constant value} \rangle,$

or

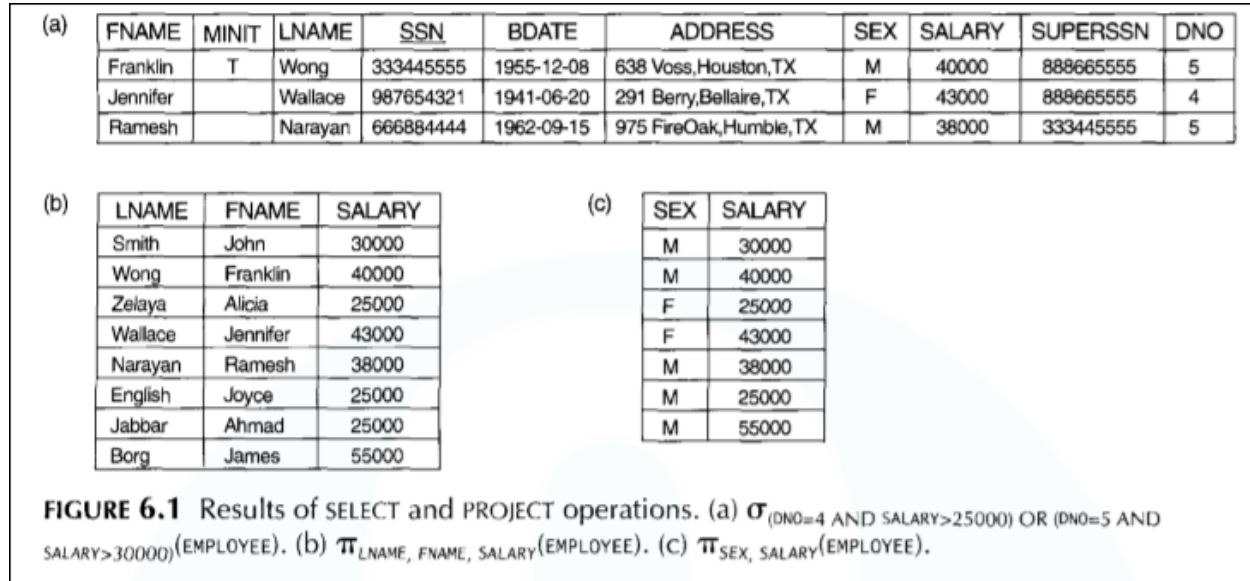
$\langle \text{attribute name} \rangle \langle \text{comparison op} \rangle \langle \text{attribute name} \rangle$

where

$\langle \text{attribute name} \rangle$ is the name of an attribute of R,

$\langle \text{comparison op} \rangle$ is normally one of the operators $\{=, <, >, \leq, \geq\}$, and

$\langle \text{constant value} \rangle$ is a constant value from the attribute domain



The PROJECT Operation

The SELECT operation selects some of the rows from the table while discarding other rows. The PROJECT operation, on the other hand, selects certain columns from the table and discards the other columns. If we are interested in only certain attributes of a relation, we use the PROJECT operation to project the relation over these attributes only. The result of the PROJECT operation can hence be visualized as a vertical partition of the relation into two relations.

For example, to list each employee's first and last name and salary, we can use the PROJECT operation as follows:

$\pi_{LNAME, FNAME, SALARY}(EMPLOYEE)$

LNAME	FNAME	SALARY
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

The general form of the PROJECT operation is

$$\pi_{\langle \text{attribute list} \rangle}(R)$$

where π (pi) is the symbol used to represent the PROJECT operation, and $\langle \text{attribute list} \rangle$ is the desired list of attributes from the attributes of relation R.

The result of the PROJECT operation has only the attributes specified in $\langle \text{attribute list} \rangle$ in the same order as they appear in the list. Hence, its degree is equal to the number of attributes in $\langle \text{attribute list} \rangle$.

The PROJECT operation removes any duplicate tuples, so the result of the PROJECT operation is a set of tuples, and hence a valid relation.' This is known as duplicate elimination. For example, consider the following PROJECT operation:

$$\pi_{\text{SEX, SALARY}}(\text{EMPLOYEE})$$

SEX	SALARY
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

RENAME Operation

For example, to retrieve the first name, last name, and salary of all employees who work in department number 5, we must apply a SELECT and a PROJECT operation. We can write a single relational algebra expression as follows:

$$\pi_{\text{FNAME, LNAME, SALARY}}(\sigma_{\text{DNO}=5}(\text{EMPLOYEE}))$$

the result of this relational algebra expression. Alternatively, we can explicitly show the sequence of operations, giving a name to each intermediate relation:

$$\begin{aligned} \text{DEP5_EMPS} &\leftarrow \sigma_{\text{DNO}=5}(\text{EMPLOYEE}) \\ \text{RESULT} &\leftarrow \pi_{\text{FNAME, LNAME, SALARY}}(\text{DEP5_EMPS}) \end{aligned}$$

(a)

FNAME	LNAME	SALARY
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

(b)

TEMP	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	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
	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

$$TEMP \leftarrow \sigma_{DNO=5}(EMPLOYEE)$$

$$R(FIRSTNAME, LASTNAME, SALARY) \leftarrow \pi_{FNAME, LNAME, SALARY}(TEMP)$$

Renaming the relation name as “R” and attributes as FIRSTNAME, LASTNAME, SALARY

TEMP	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	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
	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

R	FIRSTNAME	LASTNAME	SALARY
	John	Smith	30000
	Franklin	Wong	40000
	Ramesh	Narayan	38000
	Joyce	English	25000

The general RENAME operation when applied to a relation R of degree n is denoted by any of the following three forms

$$\rho_{S(B_1, B_2, \dots, B_n)}(R) \text{ or } \rho_S(R) \text{ or } \rho_{(B_1, B_2, \dots, B_n)}$$

where the symbol ρ (rho) is used to denote the RENAME operator, S is the new relation name, and B_1, B_2, \dots, B_n are the new attribute names.

SET OPERATIONS

Several set theoretic operations are used to merge the elements of two sets in various ways, including UNION, INTERSECTION, and SET DIFFERENCE (also called MINUS). These are binary operations; that is, each is applied to two sets (of tuples).

When these operations are adapted to relational databases, the two relations on which any of these three operations are applied must have the same type of tuples; this condition has been called **union compatibility**.

Two relations $R(A_1, A_2, \dots, A_n)$ and $S(B_1, B_2, \dots, B_n)$ are said to be **union compatible** if they have the same degree n and if $\text{dom}(A_i) = \text{dom}(B_i)$ for $1 \leq i \leq n$. This means that the two relations have the same number of attributes, and each corresponding pair of attributes has the same domain.

- **union:** The result of this operation, denoted by $R \cup S$, is a relation that includes all tuples that are either in R or in S or in both R and S . Duplicate tuples are eliminated.
- **intersection:** The result of this operation, denoted by $R \cap S$, is a relation that includes all tuples that are in both R and S .
- **set difference (or MINUS):** The result of this operation, denoted by $R - S$, is a relation that includes all tuples that are in R but not in S .

(a)

STUDENT	FN	LN
	Susan	Yao
	Ramesh	Shah
	Johnny	Kohler
	Barbara	Jones
	Amy	Ford
	Jimmy	Wang
	Ernest	Gilbert

INSTRUCTOR	FNAME	LNAME
	John	Smith
	Ricardo	Browne
	Susan	Yao
	Francis	Johnson
	Ramesh	Shah

(b)

FN	LN
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c)

FN	LN
Susan	Yao
Ramesh	Shah

(d)

FN	LN
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)

FNAME	LNAME
John	Smith
Ricardo	Browne
Francis	Johnson

The set operations UNION, INTERSECTION, and MINUS.

(a) Two union compatible relations. (b) $STUDENT \cup INSTRUCTOR$. (c) $STUDENT \cap INSTRUCTOR$. (d) $STUDENT - INSTRUCTOR$. (e) $INSTRUCTOR - STUDENT$.

THE CARTESIAN PRODUCT (OR CROSS PRODUCT) OPERATION

Next we discuss the CARTESIAN PRODUCT operation-also known as CROSS PRODUCT or CROSS JOIN-which is denoted by \times . This is also a binary set operation, but the relations on which it is applied do not have to be union compatible.

In general, the result of $R(A_1, A_2, \dots, A_n) \times S(B_1, B_2, \dots, B_m)$ is a relation Q with degree $n + m$ attributes $Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$, in that order. The resulting relation Q has one tuple for each combination of tuples—one from R and one from S .

For example, suppose that we want to retrieve a list of names of each female employee's dependents. We can do this as follows:

```

FEMALE_EMPS ←  $\sigma_{SEX='F'}(EMPLOYEE)$ 
EMPNAMES ←  $\pi_{FNAME, LNAME, SSN}(FEMALE_EMPS)$ 
EMP_DEPENDENTS ← EMPNAMES  $\times$  DEPENDENT
ACTUAL_DEPENDENTS ←  $\sigma_{SSN=ESSN}(EMP\_DEPENDENTS)$ 
RESULT ←  $\pi_{FNAME, LNAME, DEPENDENT\_NAME}(ACTUAL\_DEPENDENTS)$ 

```

FEMALE_EMPS	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5


EMPNAMES	FNAME	LNAME	SSN
	Alicia	Zelaya	999887777
	Jennifer	Wallace	987654321
	Joyce	English	453453453

EMP_DEPENDENTS	FNAME	LNAME	SSN	ESSN	DEPENDENT_NAME	SEX	BDATE	...
	Alicia	Zelays	999887777	333445555	Alice	F	1986-04-05	...
	Alicia	Zelays	999887777	333445555	Theodore	M	1983-10-25	...
	Alicia	Zelays	999887777	333445555	Joy	F	1958-05-03	...
	Alicia	Zelays	999887777	987654321	Abner	M	1942-02-28	...
	Alicia	Zelays	999887777	123456789	Michael	M	1988-01-04	...
	Alicia	Zelays	999887777	123456789	Alice	F	1988-12-30	...
	Alicia	Zelays	999887777	123456789	Elizabeth	F	1967-05-05	...
	Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	...
	Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	...
	Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	...
	Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...
	Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	...
	Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	...
	Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	...
	Joyce	English	453453453	333445555	Alice	F	1986-04-05	...
	Joyce	English	453453453	333445555	Theodore	M	1983-10-25	...
	Joyce	English	453453453	333445555	Joy	F	1958-05-03	...
	Joyce	English	453453453	987654321	Abner	M	1942-02-28	...
	Joyce	English	453453453	123456789	Michael	M	1988-01-04	...
	Joyce	English	453453453	123456789	Alice	F	1988-12-30	...
	Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	...

ACTUAL_DEPENDENTS	FNAME	LNAME	SSN	ESSN	DEPENDENT_NAME	SEX	BDATE	...
	Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...

RESULT	FNAME	LNAME	DEPENDENT_NAME
	Jennifer	Wallace	Abner

JOIN OPERATION

The JOIN operation, denoted by , is used to combine related tuples from two relations into single tuples.

Consider the example we gave earlier to illustrate CARTESIAN PRODUCT, which included the following sequence of operations:

$$\text{EMP_DEPENDENTS} \leftarrow \text{EMP_NAMES} \times \text{DEPENDENT}$$

$$\text{ACTUAL_DEPENDENTS} \leftarrow \sigma_{\text{SSN}=\text{ESSN}}(\text{EMP_DEPENDENTS})$$

These two operations can be replaced with a single JOIN operation as follows:

$$\text{ACTUAL_DEPENDENTS} \leftarrow \text{EMP_NAMES} \bowtie_{\text{SSN}=\text{ESSN}} \text{DEPENDENT}$$

The general form of a JOIN operation on two relations⁴ $R(A_1, A_2, \dots, A_n)$ and $S(B_1, B_2, \dots, B_m)$ is

$$R \bowtie_{\langle \text{join condition} \rangle} S$$

In JOIN, only combinations of tuples satisfying the join condition appear in the result, whereas in the CARTESIAN PRODUCT all combinations of tuples are included in the result.

DEPT_MGR	DNAME	DNUMBER	MGRSSN	...	FNAME	MINIT	LNAME	SSN	...
	Research	5	333445555	...	Franklin	T	Wong	333445555	...
	Administration	4	987654321	...	Jennifer	S	Wallace	987654321	...
	Headquarters	1	888665555	...	James	E	Borg	888665555	...

FIGURE 6.6 Result of the JOIN operation $\text{DEPT_MGR} \leftarrow \text{DEPARTMENT} \bowtie_{\text{MGRSSN}=\text{SSN}} \text{EMPLOYEE}$.

A general join condition is of the form:

$\langle \text{condition} \rangle \text{ AND } \langle \text{condition} \rangle \text{ AND } \dots \text{ AND } \langle \text{condition} \rangle$

where each condition is of the form $A_i \theta B_j$, A_i is an attribute of R , B_j is an attribute of S , A_i and B_j have the same domain, and θ (theta) is one of the comparison operators $\{=, <, \leq, >, \geq, \neq\}$. A JOIN operation with such a general join condition is called a **THETA JOIN**. Tuples whose join attributes are null *do not* appear in the result. In that sense, the JOIN operation does not necessarily preserve all of the information in the participating relations.

The EQUIJOIN and NATURAL JOIN Variations of JOIN

The most common use of JOIN involves join conditions with equality comparisons only. Such a JOIN, where the only comparison operator used is $=$, is called an **EQUIJOIN**.

Because one of each pair of attributes with identical values is superfluous, a new operation called **NATURAL JOIN**-denoted by $*$ was created to get rid of the second (superfluous) attribute in an EQUIJOIN condition.

The standard definition of **NATURAL JOIN** requires that the two join attributes (or each pair of join attributes) have the same name in both relations. If this is not the case, a renaming operation is applied first. In the following example, we first rename the **DNUMBER** attribute of

DEPARTMENT to DNUM so that it has the same name as the DNUM attribute in PROJECT- and then apply NATURAL JOIN:

$$\text{PROJ_DEPT} \leftarrow \text{PROJECT} * \rho_{(\text{DNAME}, \text{DNUM}, \text{MGRSSN}, \text{MGRSTARTDATE})}(\text{DEPARTMENT})$$

The same query can be done in two steps by creating an intermediate table DEPT as follows:

$$\text{DEPT} \leftarrow \rho_{(\text{DNAME}, \text{DNUM}, \text{MGRSSN}, \text{MGRSTARTDATE})}(\text{DEPARTMENT})$$

$$\text{PROJ_DEPT} \leftarrow \text{PROJECT} * \text{DEPT}$$

The attribute DNUM is called the **join attribute**.

The resulting relation is illustrated below. In the PROJ_DEPT relation, each tuple combines a PROJECT tuple with the DEPARTMENT tuple for the department that controls the project, but only one join attribute is kept.

(a)

PROJ_DEPT	PNAME	<u>PNUMBER</u>	PLOCATION	DNUM	DNAME	MGRSSN	MGRSTARTDATE
	ProductX	1	Bellaire	5	Research	333445555	1988-05-22
	ProductY	2	Sugarland	5	Research	333445555	1988-05-22
	ProductZ	3	Houston	5	Research	333445555	1988-05-22
	Computerization	10	Stafford	4	Administration	987654321	1995-01-01
	Reorganization	20	Houston	1	Headquarters	888665555	1981-06-19
	Newbenefits	30	Stafford	4	Administration	987654321	1995-01-01

The DIVISION Operation

This means that, for a tuple t to appear in the result T of the DIVISION, the values in t must appear in R in combination with every tuple in S .

(b)

R	A	B
a1	b1	
a2	b1	
a3	b1	
a4	b1	
a1	b2	
a3	b2	
a2	b3	
a3	b3	
a4	b3	
a1	b4	
a2	b4	
a3	b4	

S	A
a1	
a2	
a3	

T	B
b1	
b4	

$$T \leftarrow R \div S$$

SSN_PNOS	ESSN	PNO
	123456789	1
	123456789	2
	666884444	3
	453453453	1
	453453453	2
	333445555	2
	333445555	3
	333445555	10
	333445555	20
	999887777	30
	999887777	10
	987987987	10
	987987987	30
	987654321	30
	987654321	20
	888665555	20

SMITH_PNOS	PNO
	1
	2

Dividing SSN_PNOS by SMITH_PNOS we get below result

SSNS	SSN
	123456789
	453453453

AGGREGATE FUNCTIONS AND GROUPING

Common functions applied to collections of numeric values include SUM, AVERAGE, MAXIMUM, and MINIMUM. The COUNT function is used for counting tuples or values.

We can define an AGGREGATE FUNCTION operation, using the symbol \mathcal{F} (pronounced “script F”),⁶ to specify these types of requests as follows:

$$\langle \text{grouping attributes} \rangle \mathcal{F} \langle \text{function list} \rangle (R)$$

where $\langle \text{grouping attributes} \rangle$ is a list of attributes of the relation specified in R, and $\langle \text{function list} \rangle$ is a list of $\langle \text{function} \rangle \langle \text{attribute} \rangle$ pairs. In each such pair, $\langle \text{function} \rangle$ is one of the allowed functions—such as SUM, AVERAGE, MAXIMUM, MINIMUM, COUNT—and $\langle \text{attribute} \rangle$ is an attribute of the relation specified by R.

For example, to retrieve each department number, the number of employees in the department, and their average salary, while renaming the resulting attributes as indicated below, we write:

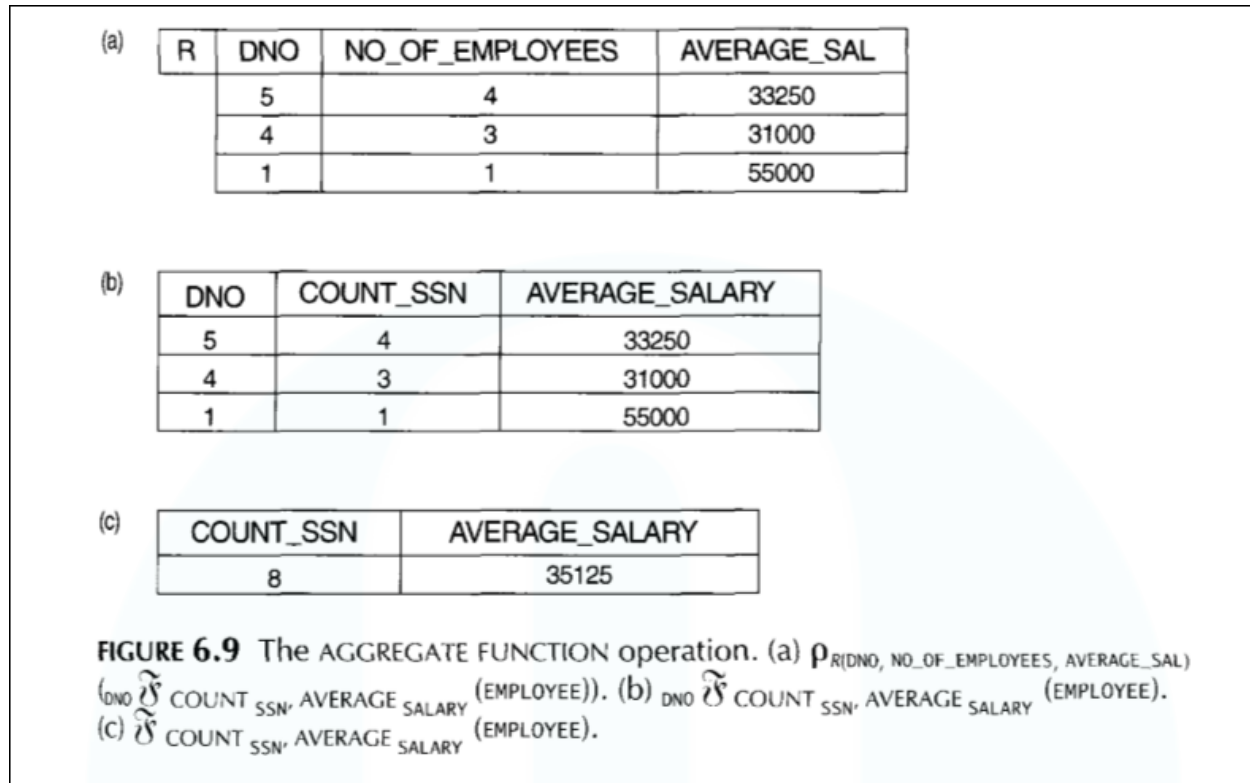
$$\rho R_{(DNO, NO_OF_EMPLOYEES, AVERAGE_SAL)}(DNO \mathcal{F} COUNT_{SSN}, AVERAGE_{SALARY}(EMPLOYEE))$$

If no renaming is applied, then the attributes of the resulting relation that correspond to the function list will each be the concatenation of the function name with the attribute name in the form $\langle \text{function} \rangle \langle \text{attribute} \rangle$. Below example, shows the result of the following operation:

$$DNO \mathcal{F} COUNT_{SSN}, AVERAGE_{SALARY}(EMPLOYEE)$$

If no grouping attributes are specified, the functions are applied to all the tuples in the relation, so the resulting relation has a single tuple only.

$$\mathcal{F} COUNT_{SSN}, AVERAGE_{SALARY}(EMPLOYEE)$$



OUTER JOIN

The JOIN operations described earlier match tuples that satisfy the join condition. For example, for a NATURAL JOIN operation $R * S$, only tuples from R that have matching tuples in S -and vice versa-appear in the result. Hence, tuples without a matching (or related) tuple are eliminated from the JOIN result.

Tuples with null values in the join attributes are also eliminated. This amounts to loss of information, if the result of JOIN is supposed to be used to generate a report based on all the information in the component relations.

A set of operations, called **outer joins**, can be used when we want to keep all the tuples in R , or all those in S , or all those in both relations in the result of the JOIN, regardless of whether or not they have matching tuples in the other relation.

This satisfies the need of queries in which tuples from two tables are to be combined by matching corresponding rows, but without losing any tuples for lack of matching values.

The join operations, where only matching tuples are kept in the result, are called **inner joins**.

For example, suppose that we want a list of all employee names and also the name of the departments they manage if they happen to manage a department; if they do not manage any, we can so indicate with a null value. We can apply an operation LEFT OUTER JOIN, denoted by



, to retrieve the result as follows:

```
TEMP ← (EMPLOYEE ⋈SSN=MGRSSN DEPARTMENT)
RESULT ← πFNAME, MINIT, LNAME, DNAME(TEMP)
```

The LEFT OUTER JOIN operation keeps every tuple in the *first*, or *left*, relation R in $R \bowtie S$; if no matching tuple is found in S , then the attributes of S in the join result are filled or “padded” with null values. The result of these operations is shown in Figure 6.11.

A similar operation, RIGHT OUTER JOIN, denoted by $\bowtie<$, keeps every tuple in the *second*, or *right*, relation S in the result of $R \bowtie< S$. A third operation, FULL OUTER JOIN, denoted by $\bowtie\ltimes$ keeps all tuples in both the left and the right relations when no matching tuples are found, padding them with null values as needed. The three outer join

EXAMPLES OF QUERIES IN RELATIONAL ALGEBRA

QUERY 1

Retrieve the name and address of all employees who work for the ‘Research’ department.

```
RESEARCH_DEPT ← σDNAME='RESEARCH'(DEPARTMENT)
RESEARCH_EMPS ← (RESEARCH_DEPT ⋈DNUMBER=DNOEMPLOYEE)
RESULT ← πFNAME, LNAME, ADDRESS(RESEARCH_EMPS)
```

This query could be specified in other ways; for example, the order of the JOIN and SELECT operations could be reversed, or the JOIN could be replaced by a NATURAL JOIN after renaming one of the join attributes.

QUERY 2

For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

```
STAFFORD_PROJS ←  $\sigma_{PLOCATION='STAFFORD'}$ (PROJECT)
CONTR_DEPT ← (STAFFORD_PROJS  $\bowtie_{DNUM=DNUMBER}$  DEPARTMENT)
PROJ_DEPT_MGR ← (CONTR_DEPT  $\bowtie_{MGRSSN=SSN}$  EMPLOYEE)
RESULT ←  $\pi_{PNUMBER, DNUM, LNAME, ADDRESS, BDATE}$ (PROJ_DEPT_MGR)
```

QUERY 3

Find the names of employees who work on *all* the projects controlled by department number 5.

```
DEPT5_PROJS(PNO) ←  $\pi_{PNUMBER}(\sigma_{DNUM=5}(\text{PROJECT}))$ 
EMP_PROJ(SSN, PNO) ←  $\pi_{ESSN, PNO}(\text{WORKS\_ON})$ 
RESULT_EMP_SSNS ← EMP_PROJ  $\div$  DEPT5_PROJS
RESULT ←  $\pi_{LNAME, FNAME}(\text{RESULT\_EMP\_SSNS} * \text{EMPLOYEE})$ 
```

QUERY 4

Make a list of project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

```
SMITHS(ESSN) ←  $\pi_{SSN}(\sigma_{LNAME='SMITH'}(\text{EMPLOYEE}))$ 
SMITH_WORKER_PROJ ←  $\pi_{PNO}(\text{WORKS\_ON} * \text{SMITHS})$ 
MGRS ←  $\pi_{LNAME, DNUMBER}(\text{EMPLOYEE} \bowtie_{SSN=MGRSSN} \text{DEPARTMENT})$ 
SMITH_MANAGED_DEPTS(DNUM) ←  $\pi_{DNUMBER}(\sigma_{LNAME='SMITH'}(\text{MGRS}))$ 
SMITH_MGR_PROJS(PNO) ←  $\pi_{PNUMBER}(\text{SMITH\_MANAGED\_DEPTS} * \text{PROJECT})$ 
RESULT ← (SMITH_WORKER_PROJS  $\cup$  SMITH_MGR_PROJS)
```

QUERY 5

List the names of all employees with two or more dependents.

Strictly speaking, this query cannot be done in the *basic (original) relational algebra*. We have to use the AGGREGATE FUNCTION operation with the COUNT aggregate function. We assume that dependents of the *same* employee have *distinct* DEPENDENT_NAME values.

$$T1(SSN, NO_OF_DEPTS) \leftarrow \text{ESSN} \ \tilde{\Sigma} \ \text{COUNT} \ \text{DEPENDENT_NAME} \ (\text{DEPENDENT})$$

$$T2 \leftarrow \sigma_{NO_OF_DEPS \geq 2}(T1)$$

$$\text{RESULT} \leftarrow \pi_{LNAME, FNAME}(T2 * \text{EMPLOYEE})$$
QUERY 6

Retrieve the names of employees who have no dependents.

This is an example of the type of query that uses the MINUS (SET DIFFERENCE) operation.

$$\text{ALL_EMPS} \leftarrow \pi_{SSN}(\text{EMPLOYEE})$$

$$\text{EMPS_WITH_DEPS}(SSN) \leftarrow \pi_{ESSN}(\text{DEPENDENT})$$

$$\text{EMPS_WITHOUT_DEPS} \leftarrow (\text{ALL_EMPS} - \text{EMPS_WITH_DEPS})$$

$$\text{RESULT} \leftarrow \pi_{LNAME, FNAME}(\text{EMPS_WITHOUT_DEPS} * \text{EMPLOYEE})$$
QUERY 7

List the names of managers who have at least one dependent.

$$\text{MGRS}(SSN) \leftarrow \pi_{MGRSSN}(\text{DEPARTMENT})$$

$$\text{EMPS_WITH_DEPS}(SSN) \leftarrow \pi_{ESSN}(\text{DEPENDENT})$$

$$\text{MGRS_WITH_DEPS} \leftarrow (\text{MGRS} \cap \text{EMPS_WITH_DEPS})$$

$$\text{RESULT} \leftarrow \pi_{LNAME, FNAME}(\text{MGRS_WITH_DEPS} * \text{EMPLOYEE})$$

Database languages(Concept of DDL and DML relational algebra)

1.DML (Data Manipulation Language)

DML statements affect records in a table. These are basic operations we perform on data such as selecting a few records from a table, inserting new records, deleting unnecessary records, and updating/modifying existing records.

DML statements include the following:

SELECT – select records from a table

INSERT – insert new records

UPDATE – update/Modify existing records

DELETE – delete existing records

2.DDL (Data Definition Language)

DDL statements are used to alter/modify a database or table structure and schema. These statements handle the design and storage of database objects.

DDL statements include the following:

CREATE – create a new Table, database, schema

ALTER – alter existing table, column description

DROP – delete existing objects from database

3.DCL (Data Control Language)

DCL statements control the level of access that users have on database objects.

GRANT – allows users to read/write on certain database objects

REVOKE – keeps users from read/write permission on database objects

4.TCL (Transaction Control Language)

TCL statements allow you to control and manage transactions to maintain the integrity of data within SQL statements.

BEGIN Transaction – opens a transaction

COMMIT Transaction – commits a transaction

ROLLBACK Transaction – ROLLBACK a transaction in case of any error

