

Module 2 Notes

Relational Model Concepts

- 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.
- 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.
- 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*.

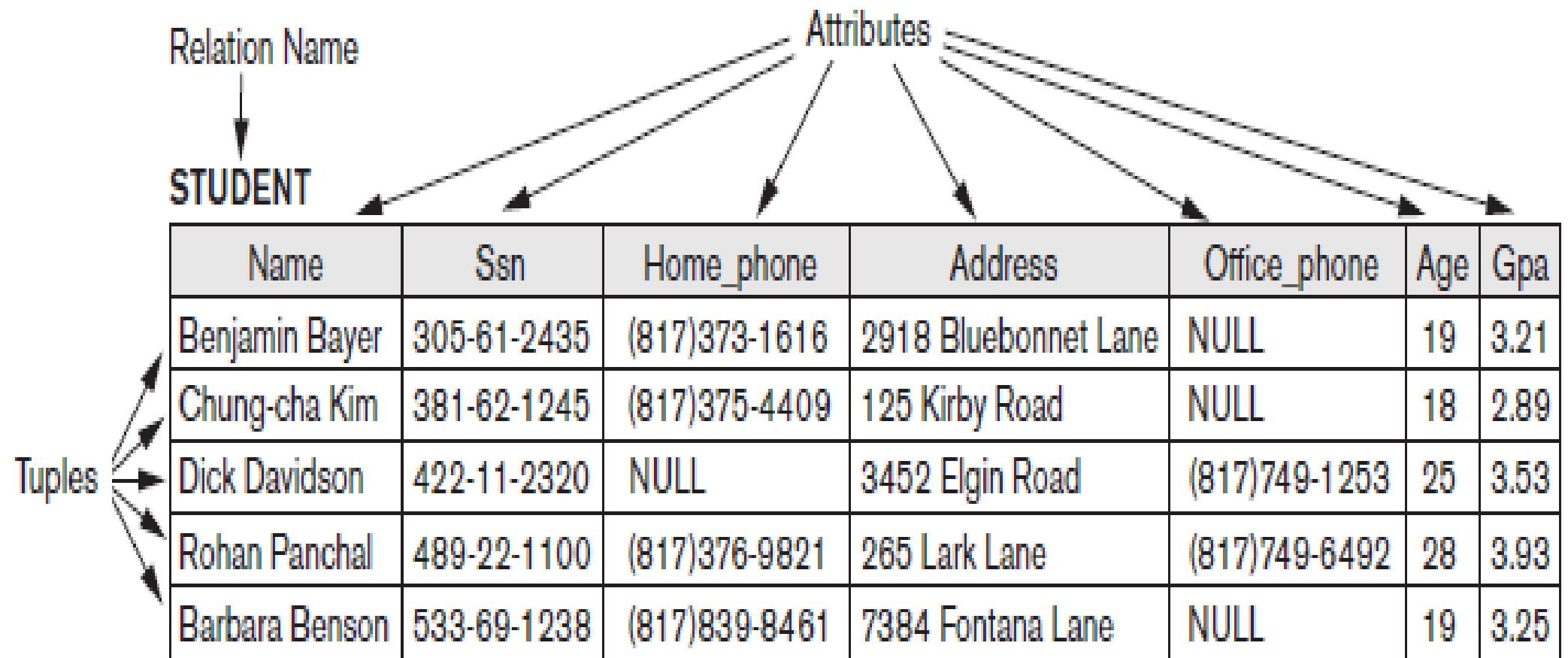


Figure 5.1

The attributes and tuples of a relation **STUDENT**.

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.
- Some examples of domains follow:
- `Usa_phone_numbers`. The set of ten-digit phone numbers valid in the United States.
- `Names`: The set of character strings that represent names of persons.
- `Employee_ages`. Possible ages of employees in a company; each must be an integer value between 15 and 80.
- The preceding are called *logical* definitions of domains. A **data type** or **format** is also specified for each domain. For example, the data type for the domain
- `Usa_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.

Relation schema

- 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 . 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.
- **Example:** 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)

Relation state

- 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 $r(R)$ are also commonly used.
- 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

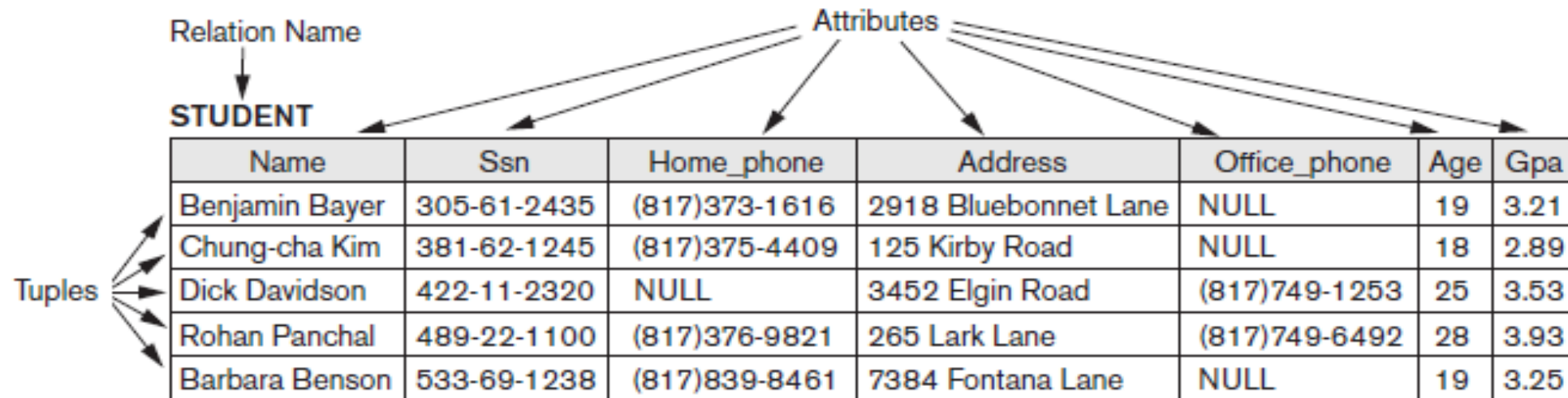


Figure 5.1

The attributes and tuples of a relation STUDENT.

Characteristics of Relations

- Ordering of Tuples in a Relation.
- Ordering of Values within a Tuple.
- Values and NULLs in the Tuples
- Interpretation (Meaning) of a 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.
- There is *no preference* for one ordering over another.
- For example, tuples in the STUDENT relation in Figure 5.1 could be ordered by values of Name, Ssn, Age, or some other attribute. The definition of a relation does not specify any order:

Characteristics of Relations

- **Ordering of Values within a Tuple**
- Order of attributes and their values is *not* that important as long as the correspondence between attributes and values is maintained
- An **alternative definition** of a relation can be given, making the ordering of values in a tuple *unnecessary*.
- 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 A_i to a value v_i from $\text{dom}(A_i)$. The ordering of attributes is *not* important, because the *attribute name* appears with its *value*.

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.
- 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.
- student has a NULL for home phone, because he does not have a home phone
- 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
- An example of the last type of NULL will occur if we add an attribute Visa_status to the STUDENT relation that applies only to tuples representing foreign students

Characteristics of Relations

- **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 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.

Relational Model Notation

- We will use the following notation in our presentation:
 - A relation schema R of degree n is denoted by $R(A_1, A_2, \dots, A_n)$.
 - 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.
 - 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 Constraints and Relational Database Schemas

- Constraints on databases can generally be divided into three main categories:
 - Constraints that are inherent in the data model are called **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**.
- 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.
- This means multivalued attributes and composite attributes are not allowed.

- **Key Constraints and Constraints on NULL Values.**

- *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:*
 - $t_1[SK] \neq t_2[SK]$
 - *SK means super key.*
 - *STUDENT(name, age, usn, address, phone, sem)*
 - *{USN, age, name}*

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.
- 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, 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.

Figure 5.6
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

- 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:
- **1. 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 $R2$.***
- **2. 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.**

Referential integrity constraints displayed on the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno
-------	-------	-------	------------	-------	---------	-----	--------	-----------	-----

DEPARTMENT

Dname	<u>Dnumber</u>	Mgr_ssn	Mgr_start_date
-------	----------------	---------	----------------

DEPT_LOCATIONS

<u>Dnumber</u>	<u>Dlocation</u>
----------------	------------------

PROJECT

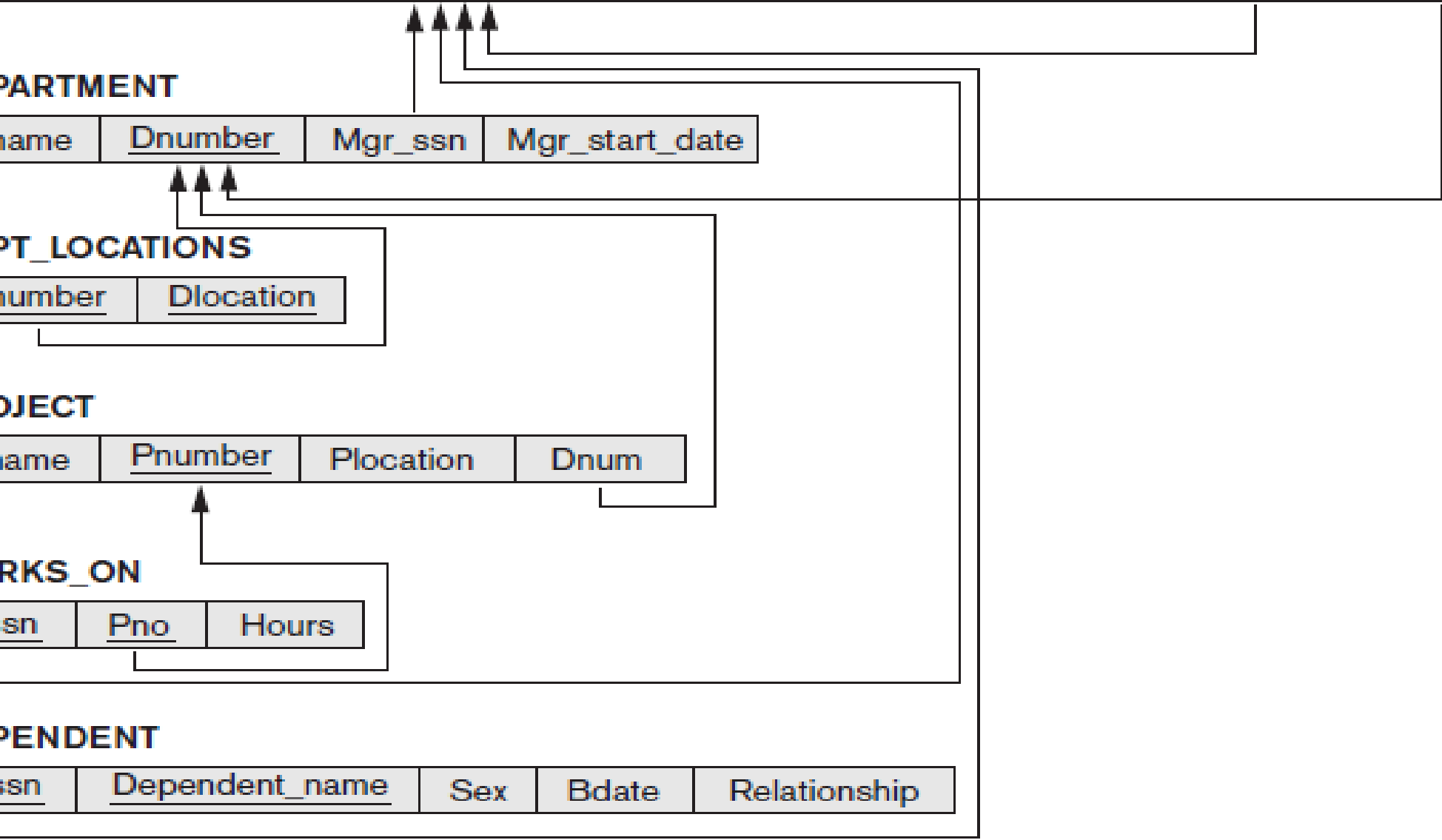
Pname	<u>Pnumber</u>	Plocation	Dnum
-------	----------------	-----------	------

WORKS_ON

<u>Essn</u>	<u>Pno</u>	Hours
-------------	------------	-------

DEPENDENT

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship
-------------	-----------------------	-----	-------	--------------



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.*

- **Insert into EMPLOYEE VALUES('Cecilia', 'F', 'Kolonsky', NULL, '1960-04-05', '6357 Windy Lane, Katy, TX', F, 28000, NULL, 4).**
- *Result: This insertion violates the entity integrity constraint (NULL for the primary key Ssn), so it is rejected*
- **Insert into EMPLOYEE VALUES**
- **('Alicia', 'J', 'Zelaya', '999887777', '1960-04-05', '6357 Windy Lane, Katy, TX', F, 28000, '987654321', 4)**
- *Result: This insertion violates the key constraint because another tuple with the same Ssn value already exists in the EMPLOYEE relation, and so it is rejected.*
- **Insert into EMPLOYEE VALUES('Cecilia', 'F', 'Kolonsky', '677678989', '1960-04-05', '6357 Windswept, Katy, TX', F, 28000, '987654321', 7);**
- *Result: This insertion violates the referential integrity constraint specified on Dno in EMPLOYEE because no corresponding referenced tuple exists in DEPARTMENT with Dnumber = 7.*

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.**
- *Operation:*
- **Delete FROM WORKS_ON**
- **WHERE Essn = '999887777' and Pno = 10.**
- *Result: This deletion is acceptable and deletes exactly one tuple.*
- *Operation:*
- **Delete FROM EMPLOYEE**
- **WHERE Ssn = '999887777'.**
- *Result: This deletion is not acceptable, because there are tuples in WORKS_ON that refer to this tuple. Hence, if the tuple in EMPLOYEE is deleted, referential integrity violations will result.*

- *Operation:*
- Delete FROM EMPLOYEE
- WHERE Ssn = '333445555'.
- *Result: This deletion will result in even worse referential integrity violations, because the tuple involved is referenced by tuples from the EMPLOYEE, DEPARTMENT, WORKS_ON, and DEPENDENT relations.*

Options are available if a deletion operation causes a violation.

- **Restrict:** 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. For example, in operation 2, the DBMS could automatically delete the offending tuples from WORKS_ON with Essn = '999887777'. A



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.*
- *Operation:*
- Update EMPLOYEE
- SET salary= 28000 WHERE Ssn = '999887777'
- *Result: Acceptable.*
- *Operation:*
- Update the Dno of the EMPLOYEE tuple with Ssn = '999887777' to 1.
- *Result: Acceptable.*
- *Operation:*
- Update the Dno of the EMPLOYEE tuple with Ssn = '999887777' to 7.
- *Result: Unacceptable, because it violates referential integrity.*

The Transaction Concept

- A **transaction is an executing program that includes** some database operations, such as reading from the database, or applying insertions, deletions, or updates to the database.
- At the end of the transaction, it must leave the database in a valid or consistent state that satisfies all the constraints specified on the database schema.
- A single transaction may involve any number of retrieval operations and any number of update operations.
- For example, a transaction to apply a bank withdrawal will typically read the user account record, check if there is a sufficient balance, and then update the record by the withdrawal amount.
- A large number of commercial applications running against relational databases in **online transaction processing (OLTP) systems are executing transactions at rates** that reach several hundred per second.

The Relational algebra.

- It is a procedural query language.
- It is a set of operations on relations.
- It is a set of algebraic expressions
- INPUT  ONE OR MORE RELATIONS
- OUTPUT  A RELATION
- SQL IS DEVELOPED BASED ON RELATIONAL ALGEBRA.
- A sequence of relational algebra operations forms a **relational algebra expression**, whose result will also be a relation that represents the result of a database query (or retrieval request).

- The relational algebra is very important for several reasons.
 1. it provides a formal foundation for relational model operations.
 2. It is used as a basis for implementing and optimizing queries in the query processing and optimization modules that are integral parts of relational database management systems (RDBMSs)
- OR
- It helps us to write OPTIMIZED QUERIES.

RELATIONAL ALGEBRA OPERATIONS

- FUNDAMENTAL OPERATIONS.

1. *SELECT*
2. *PROJECT*
3. *UNION*
4. *SET DIFFERENCES*
5. *CARTESIAN PRODUCT*
6. *RENAME*

ADDITIONAL RELATIONAL ALGEBRA OPERATIONS

- *SET INTERSECTION*
- *ASSIGNMENT OPERATIONS*
- *NATURAL JOIN*
- *DIVISION OPERATION*
- *OUTER JOIN*
 - *LEFT OUTER JOIN*
 - *RIGHT OUTER JOIN*
 - *FULL OUTER JOIN*

UNARY RELATIONAL ALGEBRA: The SELECT Operation

Here UNARY indicates-- $\square > 1$ operand or single input. It operates on single relation.

- The SELECT operation is used to choose a *subset of the tuples from a relation that satisfies a selection condition*.
- **We can consider the SELECT operation to be a *filter*** that keeps only those tuples that satisfy a qualifying condition.
- Alternatively, we can consider the SELECT operation to *restrict the tuples in a relation to only those tuples that satisfy the 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 filtered out*.

- 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 (condition) specified on the attributes of relation R .
- The Boolean expression specified in <selection condition> is made up of a number of **clauses of the form**
- <attribute name> <comparison op> <constant value>
- where <attribute name> is the name of an attribute of R , <comparison op> is *normally* one of the operators $\{=, <, \leq, >, \geq, \neq\}$, and <constant value> is a constant value from the attribute domain.
- For example, to select the tuples for all employees who either work in department 4 and make over \$25,000 per year, or work in department 5 and make over \$30,000, we can specify the following SELECT operation:
- $\sigma(\text{Dno}=4 \text{ AND Salary}>25000) \text{ OR } (\text{Dno}=5 \text{ AND Salary}>30000)(\text{EMPLOYEE})$

- ***Properties of SELECT OPERATION***

- The SELECT operator is **unary**; that is, it is applied to a single relation
- The **degree of the relation resulting from** a SELECT operation—its number of attributes—is the same as the degree of *R*.
- The number of tuples in the resulting relation is always *less than or equal to the number of tuples in R*.
- Notice that the SELECT operation is **commutative**; that is,
$$\sigma_{\langle \text{cond1} \rangle}(\sigma_{\langle \text{cond2} \rangle}(R)) = \sigma_{\langle \text{cond2} \rangle}(\sigma_{\langle \text{cond1} \rangle}(R))$$

Hence, a sequence of SELECTs can be applied in any order.

- In SQL, the SELECT condition is typically specified in the *WHERE clause of a query*. For example, the following operation:
- **$\sigma_{\text{Dno}=4 \text{ AND Salary} > 25000}(\text{EMPLOYEE})$**
- would correspond to the following SQL query:
- **SELECT ***
- **FROM EMPLOYEE**
- **WHERE Dno=4 AND Salary>25000;**

UNARY RELATIONAL ALGEBRA:

The PROJECT Operation

- 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.
- the result of the PROJECT operation can 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:
- **$\pi_{Lname, Fname, Salary}(EMPLOYEE)$**
- 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 sublist of attributes from the attributes of relation R .

- The result of the PROJECT operation has only the attributes specified in <attribute list> *in the same order as they appear in the list. Hence, its **degree is equal to the number of attributes** in <attribute list>.*
- **Properties of project operation:**
- The PROJECT operator is **unary; that is, it is applied to a single relation**
- The PROJECT operation *removes any duplicate tuples, so the result of the PROJECT operation is a set of distinct tuples,*
- The number of tuples in a relation resulting from a PROJECT operation is always less than or equal to the number of tuples in R .
- It is also noteworthy that commutativity *does not hold* on PROJECT.
- In SQL, the PROJECT attribute list is specified in the *SELECT clause of a query. For*
- example, the following operation:

$\pi_{\text{Gender, Salary}}(\text{EMPLOYEE})$

would correspond to the following SQL query:

**SELECT DISTINCT gender, Salary
FROM EMPLOYEE**

Sequences of Operations & RENAME Operation

- In general, for most queries, we need to apply several relational algebra operations one after the other. Either we can write the operations as a single **relational algebra expression by nesting the operations, or we can apply one operation** at a time and create intermediate result relations.
- we must give names to the relations that hold the intermediate results.
- 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, also known as an **in-line expression, as follows:**
- $\pi_{\text{Fname, Lname, Salary}}(\sigma_{\text{Dno}=5}(\text{EMPLOYEE}))$
- Alternatively, we can explicitly show the sequence of operations, giving a name to each intermediate relation, and using the **assignment operation, denoted by \leftarrow (left arrow)**, as follows:
- $\text{DEP5_EMPS} \leftarrow \sigma_{\text{Dno}=5}(\text{EMPLOYEE})$
- $\text{RESULT} \leftarrow \pi_{\text{Fname, Lname, Salary}}(\text{DEP5_EMPS})$

Rename operations

- $\rho S(B1, B2, \dots, Bn)(R)$
- or $\rho S(R)$ or
- $\rho(B1, B2, \dots, Bn)(R)$
- where the symbol ρ (rho) is used to denote the RENAME operator, *S is the new relation name*, and *B1, B2, ... , Bn are the new attribute names*.
- *The first expression* renames both the relation and its attributes,
- The second renames the relation only,
- Third renames the attributes only.

- Consider the student table given below –
- | | Regno | Branch | Section |
|-----|-------|--------|---------|
| • 1 | | CSE | A |
| • 2 | | ECE | B |
| • 3 | | CIVIL | B |
| • 4 | | IT | A |
- Example 1
- The student table is renamed with newstudent with the help of the following command –
- `pnewstudent (student)`

Employee

$\rho_{\text{EmployeeName/Name}}(\text{Employee})$

Name	EmployeeId
Harry	3415
Sally	2241

EmployeeName	EmployeeId
Harry	3415
Sally	2241

Binary Relational Operations:

JOIN and DIVISION

- The **JOIN** operation, denoted by \bowtie , is used to combine *related tuples from two relations* into single “longer” tuples.
- This operation is very important for any relational database with more than a single relation because it allows us to process relationships among relations.
- To illustrate JOIN, suppose that we want to retrieve the name of the manager of each department. To get the manager’s name, we need to combine each department tuple with the employee tuple whose Ssn value matches the Mgr ssn value in the department tuple.

DEPT_MGR \leftarrow DEPARTMENT \bowtie Mgr_ssn=Ssn EMPLOYEE

RESULT \leftarrow $\pi_{Dname, Lname, Fname}(DEPT_MGR)$

Variations of JOIN: The EQUIJOIN and NATURAL JOIN

- **EQUIJOIN:**

- A JOIN, where the only comparison operator used is =, is called an **EQUIJOIN**.
- previous examples were EQUIJOINS

$\text{DEPT_MGR} \leftarrow \text{DEPARTMENT} \bowtie_{\text{Mgr_ssn}=\text{Ssn}} \text{EMPLOYEE}$

$\text{RESULT} \leftarrow \pi_{\text{Dname, Lname, Fname}}(\text{DEPT_MGR})$

- **NATURAL JOIN:**

- 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.
- It is denoted by * symbol
- Suppose we want to combine each PROJECT tuple with the DEPARTMENT tuple that controls the project. In the following example, first we rename the Dnumber attribute of DEPARTMENT to Dnum—so that it has the same name as the Dnum attribute in PROJECT—and then we apply NATURAL JOIN:
- $\text{PROJ_DEPT} \leftarrow \text{PROJECT} \bowtie \rho_{\text{Dname, Dnum, Mgr_ssn, Mgr_start_date}}(\text{DEPARTMENT})$

- The same query can be done in two steps by creating an intermediate table DEPT
- as follows:
- $DEPT \leftarrow \rho(Dname, Dnum, Mgr_ssn, Mgr_start_date)(DEPARTMENT)$
- $PROJ_DEPT \leftarrow PROJECT * DEPT$
- The attribute Dnum is called the **join attribute for the NATURAL JOIN operation**, because it is the only attribute with the same name in both relations.

- **The DIVISION Operation**

- The DIVISION operation, denoted by \div , is useful for a special kind of query that sometimes occurs in database applications.
- An example is *Retrieve the names of employees who work on **all the projects that 'John Smith' works on.*** To express this query using the DIVISION operation, proceed as follows.
- First, retrieve the list of project numbers that 'John Smith' works on in the intermediate relation SMITH_PNOS:
- $SMITH \leftarrow \sigma_{Fname='John' \text{ AND } Lname='Smith'}(EMPLOYEE)$
- $SMITH_PNOS \leftarrow \pi_{Pno}(WORKS_ON \text{ Essn}=SsnSMITH)$

- Next, create a relation that includes a tuple $\langle Pno, Essn \rangle$ whenever the employee whose Ssn is Essn works on the project whose number is Pno in the intermediate relation SSN_PNOS:
- $SSN_PNOS \leftarrow \pi_{Essn, Pno}(WORKS_ON)$
- Finally, apply the DIVISION operation to the two relations, which gives the desired employees' Social Security numbers:
- $SSNS(Ssn) \leftarrow SSN_PNOS \div SMITH_PNOS$
- $RESULT \leftarrow \pi_{Fname, Lname}(SSNS * EMPLOYEE)$

(a)

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

SSNS

Ssn
123456789
453453453

(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

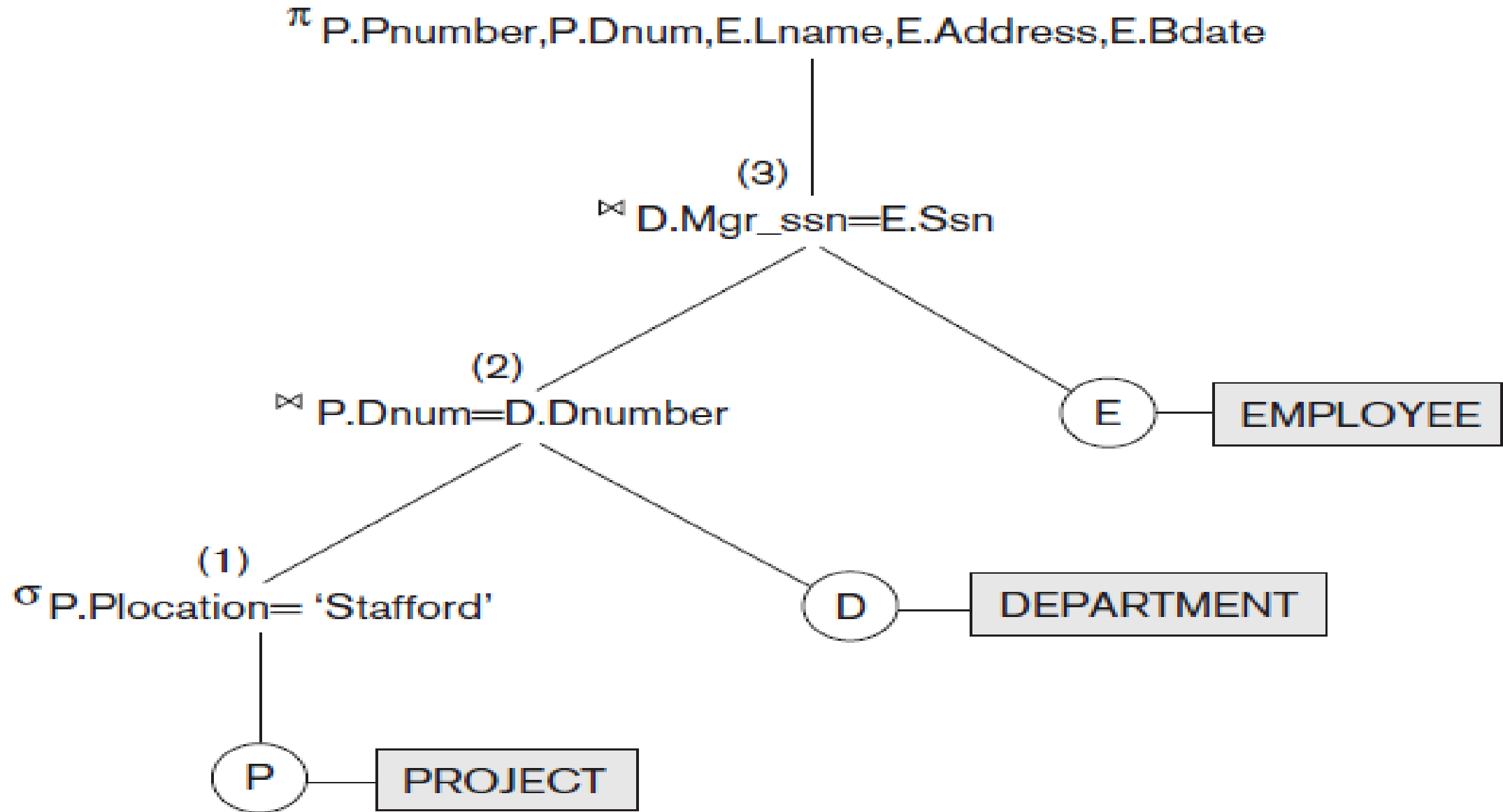
A
a1
a2
a3

T

B
b1
b4

- **Notation for Query Trees.**
- **A query tree is a tree data structure that corresponds to a relational algebra expression.**
- It represents the input relations of the query as *leaf nodes of the tree*, and represents the relational algebra operations as internal nodes.
- An execution of the query tree consists of executing an internal node operation whenever its operands (represented by its child nodes) are available, and then replacing that internal node by the relation that results from executing the operation.
- The execution terminates when the root node is executed and produces the result relation for the query

- *EX: 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.*
- π Pnumber, Dnum, Lname, Address,
Bdate(((σ Plocation='Stafford'(PROJECT))
- Dnum=Dnumber(DEPARTMENT)) Mgr_ssn=Ssn(EMPLOYEE))



- **Generalized Projection**

The generalized projection operation extends the projection operation by allowing functions of attributes to be included in the projection list. The generalized form can be expressed as:

$\pi_{F1, F2, \dots, F_n}(R)$

where $F1, F2, \dots, F_n$ are functions over the attributes in relation R and may involve arithmetic operations and constant values.

- As an example, consider the relation
EMPLOYEE (Ssn, Salary, Deduction, Years of service)

-

A report may be required to show
Net Salary = Salary – Deduction,
Bonus = 2000 * Years of service, and
Tax = 0.25 * Salary

-

Then a generalized projection combined with renaming may be used as follows:

REPORT $\leftarrow \rho(\text{Ssn, Net_salary, Bonus, Tax})(\pi_{\text{Ssn, Salary - Deduction, 2000 * Years_service, 0.25 * Salary}}(\text{EMPLOYEE}))$

Aggregate Functions and Grouping

- These functions are used in simple statistical queries that summarize information from the database tuples. Common functions applied to collections of numeric values include
 - SUM,
 - AVERAGE,
 - MAXIMUM,
 - MINIMUM.
- And COUNT- function is used for counting tuples or values.
- Examples of such functions include retrieving the average or total salary of all employees or the total number of employee tuples
- We can define an AGGREGATE FUNCTION operation, using the symbol \mathcal{F} (pronounced *script F*), to specify these types of requests as follows:
 - **<grouping attributes> \mathcal{F} <function list> (R)**
 - where <grouping attributes> is a list of attributes of the relation specified in R
- The <function list> is a list of (<function> <attribute>) pairs.
- In each such pair, <function> is one of the allowed functions—such as SUM, AVERAGE, MAXIMUM, MINIMUM, COUNT—
- and <attribute> is an attribute of the relation specified by R.

- **For example 1:**

- 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:
- $\rho R(\text{Dno}, \text{No_of_employees}, \text{Average_sal}) (\text{Dno} \tilde{\text{J}} \text{COUNT Ssn}, \text{AVERAGE Salary} (\text{EMPLOYEE}))$
- The result of this query is as follows

R

(a)

Dno	No_of_employees	Average_sal
5	4	33250
4	3	31000
1	1	55000

- **Example 2:**

- 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 <function>_<attribute>.
- For example, Figure 8.10(b) shows the result of the following operation:

Dno ⋈ COUNT Ssn, AVERAGE Salary(EMPLOYEE)

(b)

Dno	Count_ssn	Average_salary
5	4	33250
4	3	31000
1	1	55000

Examples of Queries in Relational Algebra

- **Query 1.** Retrieve the name and address of all employees who work for the 'Research' department.
- **RESEARCH_DEPT** $\leftarrow \sigma_{\text{Dname}='Research'}(\text{DEPARTMENT})$
- **RESEARCH_EMPS** $\leftarrow (\text{RESEARCH_DEPT} \bowtie_{\text{Dnumber}=\text{Dno}} \text{EMPLOYEE})$
- **RESULT** $\leftarrow \pi_{\text{Fname}, \text{Lname}, \text{Address}}(\text{RESEARCH_EMPS})$
- As a single in-line expression, this query becomes:
 $\pi_{\text{Fname}, \text{Lname}, \text{Address}}(\sigma_{\text{Dname}='Research'}(\text{DEPARTMENT} \bowtie_{\text{Dnumber}=\text{Dno}} \text{EMPLOYEE}))$
- 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.
- **ALL_EMPS** $\leftarrow \pi_{\text{Ssn}}(\text{EMPLOYEE})$
EMPS_WITH_DEPS(Ssn) $\leftarrow \pi_{\text{Essn}}(\text{DEPENDENT})$
EMPS_WITHOUT_DEPS $\leftarrow (\text{ALL_EMPS} - \text{EMPS_WITH_DEPS})$
RESULT $\leftarrow \pi_{\text{Lname}, \text{Fname}}(\text{EMPS_WITHOUT_DEPS} \bowtie \text{EMPLOYEE})$

- List the names of all employees with two or more dependents.
- 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.
- $T1(\text{Ssn}, \text{No_of_dependents}) \leftarrow \text{Essn} \bowtie \text{COUNT Dependent_name}(\text{DEPENDENT})$
 $T2 \leftarrow \sigma_{\text{No_of_dependents} > 2}(T1)$
 $\text{RESULT} \leftarrow \pi_{\text{Lname}, \text{Fname}}(T2 * \text{EMPLOYEE})$

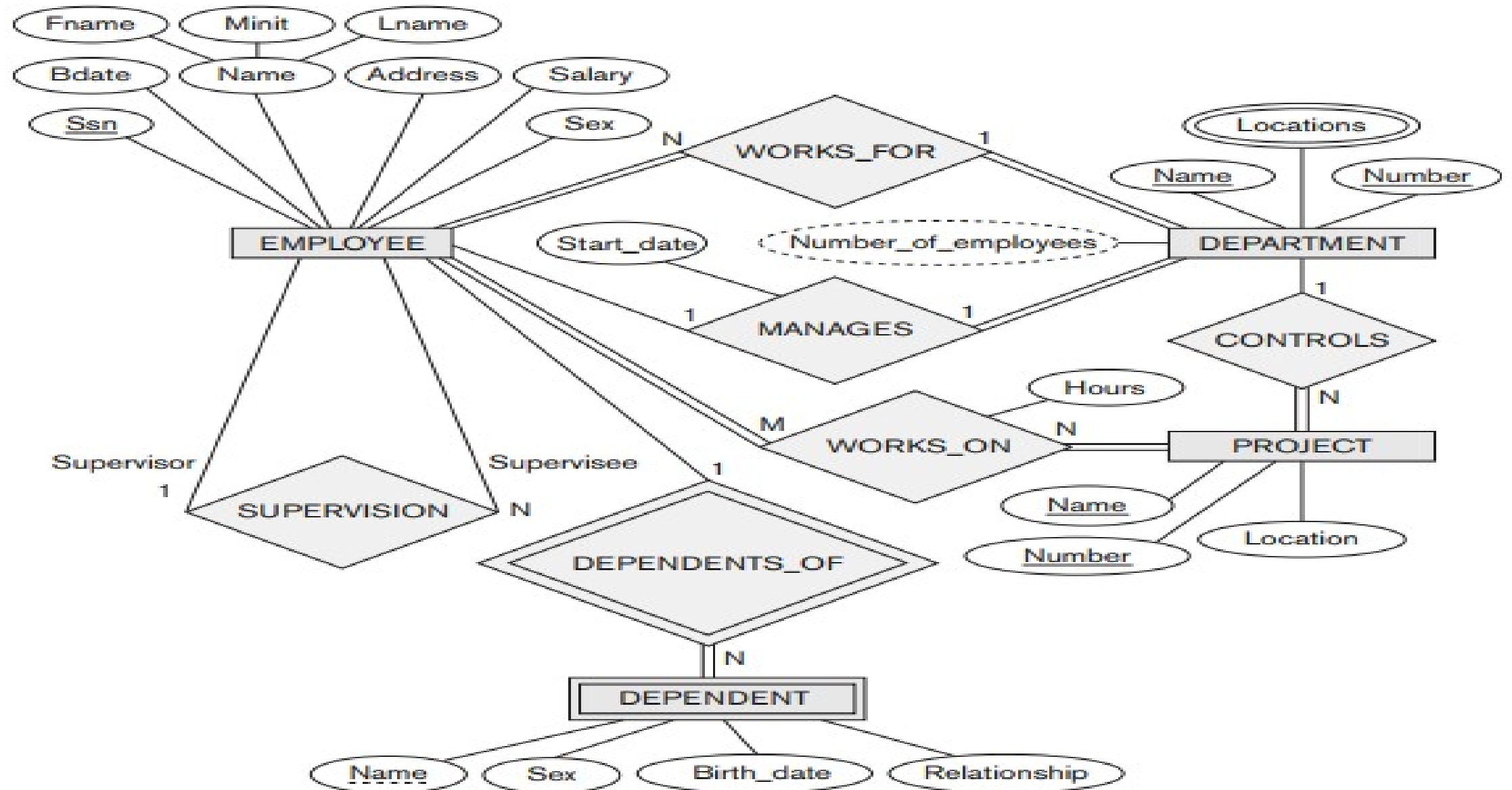
Relational Database Design Using ER-to-Relational Mapping

- **ER-to-Relational Mapping Algorithm**

We use the COMPANY database example to illustrate the mapping procedure. The COMPANY ER schema is shown again in Figure 9.1, and the corresponding COMPANY relational database schema is shown in Figure 9.2 to illustrate the mapping steps

Figure 9.1

The ER conceptual schema diagram for the COMPANY database.



EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno
-------	-------	-------	------------	-------	---------	-----	--------	-----------	-----

DEPARTMENT

Dname	<u>Dnumber</u>	Mgr_ssn	Mgr_start_date
-------	----------------	---------	----------------

DEPT_LOCATIONS

<u>Dnumber</u>	<u>Dlocation</u>
----------------	------------------

PROJECT

Pname	<u>Pnumber</u>	<u>Plocation</u>	Dnum
-------	----------------	------------------	------

WORKS_ON

<u>Essn</u>	<u>Pno</u>	Hours
-------------	------------	-------

DEPENDENT

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship
-------------	-----------------------	-----	-------	--------------

Figure 9.2

Result of mapping the COMPANY ER schema into a relational database schema.

- **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 the primary key for R .
- In our example, we create the relations EMPLOYEE, DEPARTMENT, and PROJECT in Figure 9.2 to correspond to the regular entity types EMPLOYEE, DEPARTMENT, and PROJECT from Figure 9.1.
- The result after this mapping step is shown in Figure 9.3(a).

(a) **EMPLOYEE**

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary
-------	-------	-------	------------	-------	---------	-----	--------

DEPARTMENT

Dname	<u>Dnumber</u>
-------	----------------

PROJECT

Pname	<u>Pnumber</u>	Plocation
-------	----------------	-----------

- **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 attributes) of W as attributes of R .

- In our example, we create the relation DEPENDENT in this step to correspond to the weak entity type DEPENDENT (see Figure 9.3(b)). 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 rename it Essn,
- The primary key of the DEPENDENT relation is the combination {Essn, Dependent_name},

(b) DEPENDENT

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship
-------------	-----------------------	-----	-------	--------------

- **Step 3: Mapping of Binary 1:1 Relationship Types.**

There are three possible approaches:

- (1) the foreign key approach,
- (2) the merged relationship approach,
- and (3) the cross reference or relationship relation approach.
- **Foreign key approach:** 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 .
- Example, we map the 1:1 relationship type MANAGES from Figure 9.1 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 to Mgr_ssn.

- **Merged Relation Approach:** An alternative mapping of a 1:1 relationship type is to merge the two entity types and the relationship into a single relation. This is possible when *both participations are total*, as this would indicate that the two tables will have the exact same number of tuples at all times.
Cross-reference or relationship relation approach: The third option 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.

- **Step 4: 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.
- **Step 5: 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 .
- In our example, we map the M:N relationship type WORKS_ON from Figure 9.1 by creating the relation WORKS_ON in Figure 9.2. We include the primary keys of the PROJECT and EMPLOYEE relations as foreign keys in WORKS_ON and rename them Pno and Essn, respectively (renaming is *not required*; it is a design choice). 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}. This **relationship relation** is shown in Figure 9.3(c).

- **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 a multivalued 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 (see Figure 9.3(d)). The attribute Dlocation represents the multivalued attribute LOCATIONS of DEPARTMENT, whereas Dnumber—as foreign key—represents the primary key of the DEPARTMENT relation. The primary key of DEPT_LOCATIONS is the combination of {Dnumber, Dlocation}.
- **Step 7: 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 .

6.1 SQL Data Definition and Data Types

- **6.1.1 Schema and Catalog Concepts in SQL**

An SQL Schema is identified by a **schema name** and includes an **authorization identifier** to indicate the user or account who owns the schema, as well as **descriptors** for *each element* in the schema.

- A schema is created via the CREATE SCHEMA statement, which can include all the schema elements' definitions. Alternatively, the schema can be assigned a name and authorization identifier, and the elements can be defined later. For example, the following statement creates a schema called COMPANY owned by the user with authorization identifier 'Jsmith'.

Note that each statement in SQL ends with a semicolon.

-

CREATE SCHEMA COMPANY AUTHORIZATION 'Jsmith';

- We can explicitly attach the schema name to the relation name, separated by a period. For example, by writing

CREATE TABLE COMPANY.EMPLOYEE

-

rather than

CREATE TABLE EMPLOYEE

- **Attribute Data Types and Domains in SQL**

The basic **data types** available for attributes include numeric, character string, bit string, Boolean, date, and time.

■ **Numeric** data types include integer numbers of various sizes (INTEGER or INT, and SMALLINT) and floating-point (real) numbers of various precision (FLOAT or REAL, and DOUBLE PRECISION). Formatted numbers can be declared by using DECIMAL(*i*, *j*)—or DEC(*i*, *j*) or NUMERIC(*i*, *j*)—where *i*, the *precision*, is the total number of decimal digits and *j*, the *scale*, is the number of digits after the decimal point

- **Character-string** data types are either fixed length—CHAR(*n*) or CHARACTER(*n*), where *n* is the number of characters—or varying length—VARCHAR(*n*) or CHAR VARYING(*n*) or CHARACTER VARYING(*n*), where *n* is the maximum number of characters.
- **Bit-string** data types are either of fixed length *n*—BIT(*n*)—or varying length—BIT VARYING(*n*), where *n* is the maximum number of bits.
- **A Boolean** data type has the traditional values of TRUE or FALSE. In SQL, because of the presence of NULL values, a three-valued logic is used, so a third possible value for a Boolean data type is UNKNOWN

- The **DATE** data type has ten positions, and its components are YEAR, MONTH, and DAY in the form YYYY-MM-DD. The TIME data type has at least eight positions, with the components HOUR, MINUTE, and SECOND in the form HH:MM:SS.

Specifying Constraints in SQL

- Given in LAB PPTs

Basic Retrieval Queries in SQL

- **Query 0.** Retrieve the birth date and address of the employee(s) whose name is 'John B. Smith'.
- **Query 1.** Retrieve the name and address of all employees who work for the 'Research' department.

- **Query 12.** Retrieve all employees whose address is in Houston, Texas.
- To retrieve all employees who were born during the 1970s.
- **Query 13.** Show the resulting salaries if every employee working on the 'ProductX' project is given a 10% raise.
- **Query 14.** Retrieve all employees in department 5 whose salary is between \$30,000 and \$40,000

- **The INSERT Command**

In its simplest **First form**, INSERT is used to add a single tuple (row) to a relation (table). We must specify the relation name and a list of values for the tuple.

- INSERT INTO EMPLOYEE
- VALUES ('Richard', 'K', 'Marini', '653298653', '1962-12-30', '98
- Oak Forest, Katy, TX', 'M', 37000, '653298653', 4);

- A **second form** of the INSERT statement allows the user to specify explicit attribute names that correspond to the values provided in the INSERT command.

- INSERT INTO EMPLOYEE
- VALUES ('Richard', 'K', 'Marini', '653298653', '1962-12-30', '98
- Oak Forest, Katy, TX', 'M', 37000, '653298653', 4);

- **The DELETE Command**

The DELETE command removes tuples from a relation. It includes a WHERE clause, similar to that used in an SQL query, to select the tuples to be deleted. Tuples are explicitly deleted from only one table at a time.

- DELETE FROM EMPLOYEE

- WHERE Lname = 'Brown';

- DELETE FROM EMPLOYEE

- WHERE Ssn = '123456789';

- DELETE FROM EMPLOYEE

- WHERE Dno = 5;

- DELETE FROM EMPLOYEE; // This deletes all tuples from employee table

- **The UPDATE Command**

The **UPDATE** command is used to modify attribute values of one or more selected tuples.

- UPDATE PROJECT
- SET Plocation = 'Bellaire', Dnum = 5
- WHERE Pnumber = 10;

- UPDATE EMPLOYEE
- SET Salary = Salary * 1.1
- WHERE Dno = 5;

- SQL has various techniques for writing programs in various programming languages that include SQL statements to access one or more databases. These include embedded (and dynamic) SQL, SQL/CLI (Call Level Interface) and its predecessor ODBC (Open Data Base Connectivity), and SQL/PSM (Persistent Stored Modules)
- Each commercial RDBMS will have, in addition to the SQL commands, a set of commands for specifying physical database design parameters, file structures for relations, and access paths such as indexes. We called these commands a *storage definition language (SDL)*
- SQL has transaction control commands. These are used to specify units of database processing for concurrency control and recovery purposes.
- SQL has language constructs for specifying the *granting and revoking of privileges* to users. Privileges typically correspond to the right to use certain SQL commands to access certain relations. Each relation is assigned an owner, and either the owner or the DBA staff can grant to selected users the privilege to use an SQL statement.
- SQL has language constructs for creating triggers. These are generally referred to as **active database** techniques, since they specify actions that are automatically triggered by events such as database updates.
- SQL and relational databases can interact with new technologies such as XML