Database Management Systems (BCS403) – 2023-24 Module 2

Dr. Narender M
Department of CS&E
The National Institute of Engineering

Topics

Relational Model

- Relational Model Concepts, Relational Model Constraints and relational database schemas, Update operations, transactions, and dealing with constraint violations.

Relational Algebra

- Unary and Binary relational operations, additional relational operations (aggregate, grouping, etc.) Examples of Queries in relational algebra.

Mapping Conceptual Design into a Logical Design

- Relational Database Design using ER-to-Relational mapping

Module 2 – Chapter 1

Relational Model Concepts

- The relational model represents the database as a collection of relations.
- Informally, each relation resembles a table of values or, to some extent, a flat file of records
- 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 Concepts

- In the formal relational model terminology, a row → a tuple, a column header →an attribute, and the table →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 Concepts

Domains, Attributes, Tuples, and Relations

- A domain D is a set of atomic values.
- By atomic means each value in the domain is indivisible in formal relational model.
- A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn.
- Some examples of domains follow:
  - USA_phone_number: string of digits of length ten
  - SSN: string of digits of length nine
  - Name: string of characters beginning with an upper-case letter
  - GPA: a real number between 0.0 and 4.0
  - Sex: a member of the set { female, male }
  - Dept_Code: a member of the set { CMPS, MATH, ENGL, PHYS, PSYC, … }

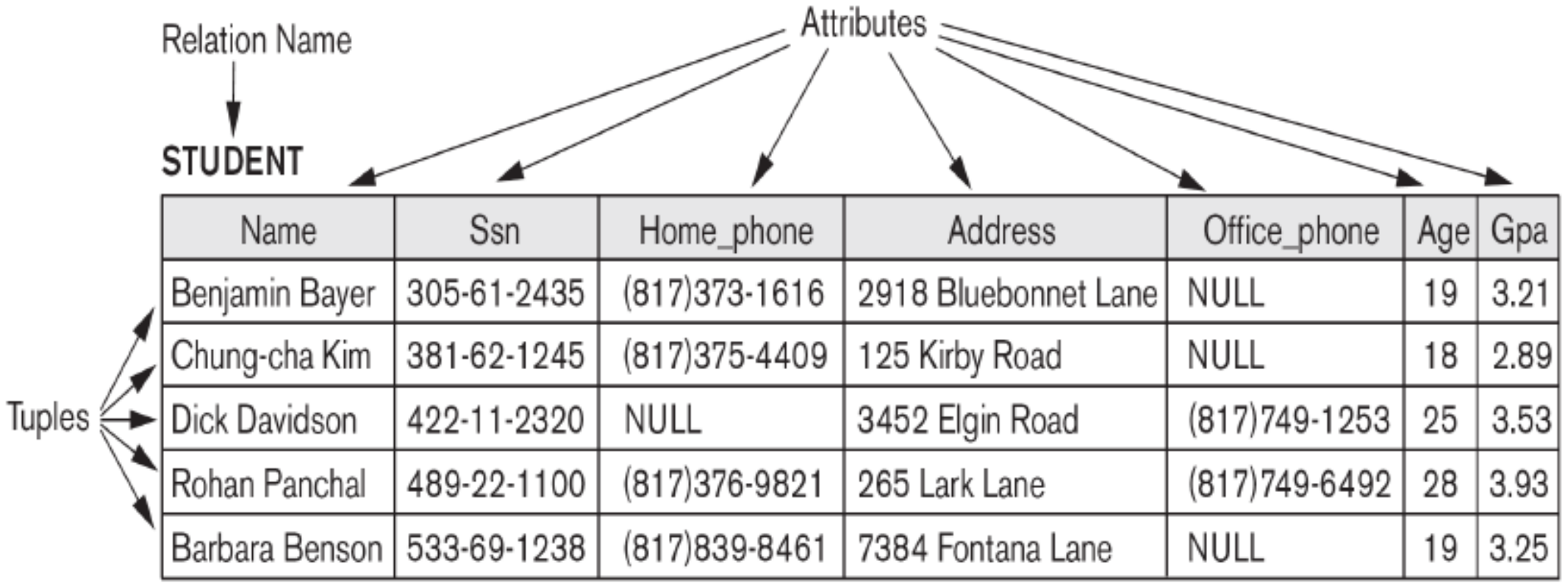
Relational Model Concepts

- A relation schema R, denoted by $R(A_1, A_2, \ldots, A_n)$, is made up of a relation name R and a list of attributes, $A_1, A_2, \ldots, A_n$.
- 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 $dom(A_i)$.
- Tuple: A tuple is a mapping from attributes to values drawn from the respective domains of those attributes.
- A tuple is intended to describe some entity (or relationship between entities) in the miniworld.
- R is called the name of this relation.

Relational Model Concepts

- 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)
- Relational Database: A collection of relations, each one consistent with its specified relational schema.
- A relation (or relation state) r of the relation schema $R(A_1, A_2, \ldots, A_n)$, also denoted by r(R), is a set of n-tuples $r = \{t_1, t_2, \ldots, t_m\}$. Each n-tuple t is an ordered list of n values $t = <v_1, v_2 \ldots, v_n>$

Relational Model Concepts



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

Relational Model Concepts

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.
- When tuples are represented on a storage device, they must be organized in some fashion, and it may be advantageous, from a performance standpoint, to organize them in a way that depends upon their content.

Relational Model Concepts

2. Ordering of Values within a Tuple

- The order of attributes and their values is not that important as long as the correspondence between attributes and values is maintained.
- 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 dom(Ai).
- The ordering of attributes is not important, because the attribute name appears with its value.

Relational Model Concepts

3. Values and NULLs in the Tuples

- Each value in a tuple is an atomic value; that is, it is not divisible into components.
- An important concept is NULL values, which are used to represent the values of attributes that may be unknown or may not apply to a tuple.
- NULL values has several meanings, such as value unknown, value exists but is not available, or attribute does not apply to this tuple.

Relational Model Concepts

4. Interpretation (Meaning) of a Relation

- Each tuple in the relation can then be interpreted as a fact or a particular instance of the assertion.
- Each relation can be viewed as a predicate and each tuple in that relation can be viewed as an assertion for which that predicate is satisfied (i.e., has value true) for the combination of values in it.
- Example: There exists a student having name Benjamin Bayer, having SSN 305-61-2435, having age 19, etc.

Relational Model Concepts

Relational Model Notation: The following notation are used for presentation:

- A relation schema R of degree n is denoted by R(A1, A2, ... , 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.

Relational Model Concepts

- 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.
- An n-tuple ‘t’ in a relation r(R) is denoted by t =<v1,v2, ..., 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 from t corresponding to the attributes specified in the list.

Relational Model Constraints and Relational Database Schemas

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

1. Constraints that are inherent in the **data model** known as **inherent model-based constraints** or implicit constraints.
2. Constraints that can be directly expressed in the **schemas** of the data model, typically by specifying them in the **DDL** is known as **schema-based constraints** or explicit 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** or in some other way known as **application-based** or semantic constraints or business rules.

Relational Model Constraints and Relational Database Schemas

- 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 and real numbers. **Characters, Booleans**, fixed-length strings, and variable-length strings are also available, as are **date, time, timestamp**, and **other special data types**.

Relational Model Constraints and Relational Database Schemas

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.

Relational Model Constraints and Relational Database Schemas

- Suppose that we denote one such subset of attributes by SK; then for any two distinct tuples t1 and t2 in a relation state r of R, we have the constraint that: t1[SK] ≠ t2[SK].
- 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.
- 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.

Relational Model Constraints and Relational Database Schemas

- Hence, a key satisfies two properties:

1. 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.
2. 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 and Relational Database Schemas

Relational Databases and Relational Database Schemas

- A relational database is a collection of many relations.
- A relational database schema S is a set of relation schemas S = {R1, R2, … , Rm} and a set of integrity constraints IC.
- A relational database state DB of S is a set of relation states DB = {r1, r2, …, rm} such that each ri is a state of Ri and such that the ri relation states satisfy the integrity constraints specified in IC.

Relational Model Constraints and Relational Database Schemas

- A relational database schema that we call COMPANY = {EMPLOYEE, DEPARTMENT, DEPT_LOCATIONS, PROJECT, WORKS_ON, DEPENDENT}.
- 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 IC is called a valid state.

Relational Model Constraints 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 Model Constraints and Relational Database Schemas

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.
- 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.

Relational Model Constraints and Relational Database Schemas

- 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.
- 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:

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.

Relational Model Constraints and Relational Database Schemas

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. In the former case, we have t1[FK] = t2[PK], and we say that the tuple t1 references or refers to the tuple t2.

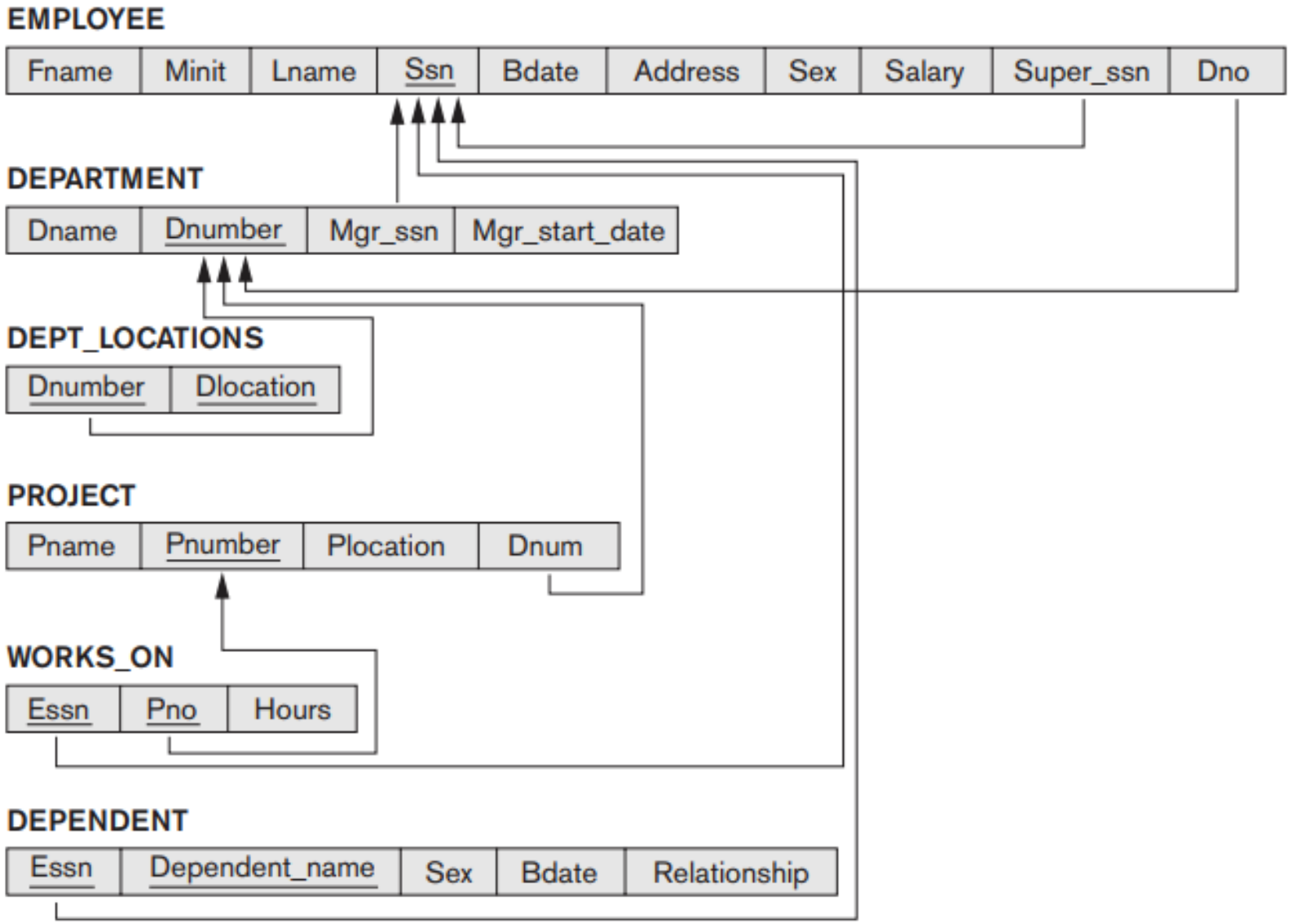
- In this definition, R1 is called the referencing relation and R2 is the referenced relation. If these two conditions hold, a referential integrity constraint from R1 to R2 is said to hold.
- In the EMPLOYEE relation, the attribute Dno refers to the department for which an employee works; hence, it is designated Dno to be a foreign key of EMPLOYEE referencing the DEPARTMENT relation.

Relational Model Constraints and Relational Database Schemas

- This means that a value of Dno in any tuple t1 of the EMPLOYEE relation must match a value of Constraints the primary key of DEPARTMENT—the Dnumber attribute—in some tuple t2 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.

Relational Model Constraints and Relational Database Schemas

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



Relational Model Constraints and Relational Database Schemas

Other types of constraints

- 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. Sometimes called as Semantic Integrity constraint.

Update operations, Transactions, 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).
  - 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.

Update operations, Transactions, 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.

1. 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.
2. Key constraints can be violated if a key value in the new tuple t already exists in another tuple in the relation r(R).
3. Entity integrity can be violated if any part of the primary key of the new tuple t is NULL.
4. 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.

Update operations, Transactions, and Dealing with Constraint Violations

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

Update operations, Transactions, and Dealing with Constraint Violations

- If an insertion violates one or more constraints, the default option is to reject the insertion.
- Another option is to attempt to correct the reason for rejecting the insertion, but this is typically not used for violations caused by Insert; rather, it is used more often in correcting violations for Delete and Update.

Update operations, Transactions, 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.

Update operations, Transactions, and Dealing with Constraint Violations

- 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 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.

Update operations, Transactions, and Dealing with Constraint Violations

- *Operation*:
  Delete the WORKS_ON tuple with Essn = '999887777' and Pno = 10.
  Result: This deletion is acceptable and deletes exactly one tuple.
- *Operation*:
  Delete the EMPLOYEE tuple with 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 the EMPLOYEE tuple with 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.

Update operations, Transactions, 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.

Update operations, Transactions, and Dealing with Constraint Violations

- If a foreign key attribute is modified, the DBMS must make sure that the new value refers to an existing tuple in the referenced relation (or is set to NULL).
- When a referential integrity constraint is specified in the DDL, the DBMS will allow the user to choose separate options.

Update operations, Transactions, and Dealing with Constraint Violations

- *Operation*:
  Update the salary of the EMPLOYEE tuple with Ssn = '999887777' to 28000.
  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.
- *Operation*:
  Update the Ssn of the EMPLOYEE tuple with Ssn = '999887777' to '987654321'.
  Result: Unacceptable, because it violates primary key constraint by repeating a value that already exists as a primary key in another tuple; it violates referential integrity constraints because there are other relations that refer to the existing value of Ssn.

Update operations, Transactions, and Dealing with Constraint Violations

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.
- These retrievals and updates will together form an atomic unit of work against the database.

Update operations, Transactions, and Dealing with Constraint Violations

- 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.

Module 2 – Chapter 2

Unary and Binary relational operations

The SELECT Operation

- The SELECT operation is used to choose a subset of the tuples from a relation that satisfies a ‘selection condition’.
- It restricts the tuples in a relation to only those tuples that satisfy the condition.
- It 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

Unary and Binary relational operations

- In general, the SELECT operation is denoted by $\sigma_{<\text{selection condition}>}(R)$
- 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 is made up of a number of clauses of the form : <attribute name><comparison op><constant value> Or <attribute name><comparison op><attribute name>
- Clauses can be connected by the standard Boolean operators and, or, and not to form a general selection condition.

Unary and Binary relational operations

- $\sigma_{<\text{selection condition}>}(R)$
- $\sigma_{Dno=4}(EMPLOYEE)$
- $\sigma_{Salary>30000}(EMPLOYEE)$
- $\sigma_{(Dno=4\ AND\ Salary>25000)\ OR\ (Dno=5\ AND\ Salary>30000)}(EMPLOYEE)$
- The Boolean conditions AND, OR, and NOT have their normal interpretation, as follows:
  - (cond1 AND cond2) is TRUE if both (cond1) and (cond2) are TRUE; otherwise,it is FALSE.
  - (cond1 OR cond2) is TRUE if either (cond1) or (cond2) or both are TRUE; otherwise, it is FALSE.
  - (NOT cond) is TRUE if cond is FALSE; otherwise, it is FALSE.

Unary and Binary relational operations

- The SELECT operator is unary; that is, it is applied to a single relation. Hence, selection conditions cannot involve more than one tuple.
- The degree of the relation resulting from a SELECT operation—its number of attributes—is the same as the degree of R.
- The SELECT operation is commutative; that is,

$$\sigma_{(cond1)}(\sigma_{(cond2)}(R)) = \sigma_{(cond2)}(\sigma_{(cond1)}(R))$$

Unary and Binary relational operations

The PROJECT Operation

- The PROJECT operation, selects certain columns from the table and discards the other columns.
- The result of the PROJECT operation can be visualized as a vertical partition of the relation into two relations: one has the needed columns (attributes) and contains the result of the operation, and the other contains the discarded columns.
- For example, to list each employee's first and last name and salary, we can use the PROJECT operation as follows:

$$\pi_{Lname,\ Fname,\ Salary}(EMPLOYEE)$$

Unary and Binary relational operations

- The general form of the PROJECT operation is $\pi_{<\text{attribute list}>}(R)$
- where (pi) is the symbol used to represent the PROJECT operation, and is the desired sublist of attributes from the attributes of relation R.
- The result of the PROJECT operation has only the attributes specified in in the same order as they appear in the list. Hence, its degree is equal to the number of attributes in <attribute list>.
- The PROJECT operation removes any duplicate tuples, so the result of the PROJECT operation is a set of distinct tuples, and hence a valid relation. This is known as duplicate elimination.

Unary and Binary relational operations

Sequences of Operations and the RENAME Operation

- The relations shown above depict operation results do not have any names.
- 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.
- In the latter case, 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, apply a SELECT and a PROJECT operation.

$$\pi_{Fname,\ Lname,\ Salary}(\sigma_{Dno=5}(EMPLOYEE))$$

Unary and Binary relational operations

- Alternatively, we can explicitly show the sequence of operations, giving a name to each intermediate relation, and using the assignment operation, denoted by ← (left arrow), as follows:

  $DEP5\_EMPS \leftarrow \sigma_{Dno=5}(EMPLOYEE)$

  $RESULT \leftarrow \pi_{Fname,\ Lname,\ Salary}(DEP5\_EMPS)$

- It is sometimes simpler to break down a complex sequence of operations by specifying intermediate result relations than to write a single relational algebra expression.
- We can also use this technique to rename the attributes in the intermediate and result relations.

Unary and Binary relational operations

- To rename the attributes in a relation, we simply list the new attribute names in parentheses, as in the following example:

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

  $R(First\_name, Last\_name, Salary) \leftarrow \pi_{Fname, Lname, Salary}(TEMP)$

- The formal RENAME operation—which can rename either the relation name or the attribute names, or both—as a unary operator.

Unary and Binary relational operations

- The general RENAME operation when applied to a relation R of degree n is denoted by any of the following three forms: ρS(B1, B2, … , Bn)(R) or ρS(R) or ρ(B1, B2, … , 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, and the third renames the attributes only.

Unary and Binary relational operations

Relational Algebra Operations from Set Theory (The UNION, INTERSECTION, and MINUS Operations)

- UNION: The result of this operation, denoted by R ∪ 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 ∩ 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.
- These are binary operations; that is, each is applied to two sets (of tuples).

Unary and Binary relational operations

- 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 or type compatibility.
- Two relations R(A1, A2, ... , An) and S(B1, B2, ... , Bn) are said to be union compatible (or type compatible) if they have the same degree n and if dom(Ai) = dom(Bi) 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.

Unary and Binary relational operations

- For example, to retrieve the Social Security numbers of all employees who either work in department 5 or directly supervise an employee who works in department 5.
- DEP5_EMPS $\leftarrow \sigma_{Dno=5}(EMPLOYEE)$
- RESULT1 $\leftarrow \pi_{Ssn}(DEP5\_EMPS)$
- RESULT2(Ssn) $\leftarrow \pi_{Super\_ssn}(DEP5\_EMPS)$
- RESULT $\leftarrow$ RESULT1 $\cup$ RESULT2
- Both UNION and INTERSECTION are commutative operations; that is,

  $R \cup S = S \cup R$ and $R \cap S = S \cap R$

Unary and Binary relational operations

- Both UNION and INTERSECTION can be treated as n-ary operations applicable to any number of relations because both are also associative operations; that is,
  $R \cup (S \cup T) = (R \cup S) \cup T$ and $(R \cap S) \cap T = R \cap (S \cap T)$
- The MINUS operation is not commutative; that is, in general,
  $R - S \neq S - R$
- The INTERSECTION can be expressed in terms of union and set difference as follows: $R \cap S = ((R \cup S) - (R - S)) - (S - R)$

Unary and Binary relational operations

The CARTESIAN PRODUCT (CROSS PRODUCT) Operation

- CARTESIAN PRODUCT operation—also known as CROSS PRODUCT or CROSS JOIN— which is denoted by ×.
- This is also a binary set operation, but the relations on which it is applied do not have to be union compatible.
- This set operation produces a new element by combining every member (tuple) from one relation (set) with every member (tuple) from the other relation (set).
- In general, the result of R(A1, A2, ..., An) × S(B1, B2, ..., Bm) is a relation Q with degree n + m attributes Q(A1, A2, ..., An, B1, B2, ..., Bm), in that order.

Unary and Binary relational operations

- The resulting relation Q has one tuple for each combination of tuples—one from R and one from S.
- If R has m tuples and S has n tuples, then R × S will have m*n tuples.
- Example, suppose that we want to retrieve a list of names of each female employee's dependents. We can do this as follows:

$$\text{FEMALE\_EMPS} \leftarrow \sigma_{\text{Sex='F'}}(\text{EMPLOYEE})$$

$$\text{EMPNAMES} \leftarrow \pi_{\text{Fname, Lname, Ssn}}(\text{FEMALE\_EMPS})$$

$$\text{EMP\_DEPENDENTS} \leftarrow \text{EMPNAMES} \times \text{DEPENDENT}$$

$$\text{ACTUAL\_DEPENDENTS} \leftarrow \sigma_{\text{Ssn=Essn}}(\text{EMP\_DEPENDENTS})$$

$$\text{RESULT} \leftarrow \pi_{\text{Fname, Lname, Dependent\_name}}(\text{ACTUAL\_DEPENDENTS})$$

Unary and Binary relational operations

Binary Relational Operations: JOIN and DIVISION (The JOIN Operation)

- The JOIN operation, denoted by ⋈, 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, as follows:

$$DEPT\_MGR \leftarrow DEPARTMENT \bowtie_{Mgr\_ssn=Ssn} EMPLOYEE$$

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

Unary and Binary relational operations

- The general form of a JOIN operation on two relations R(A1, A2, … , An) and S(B1, B2, … , Bm) is $R \bowtie_{\text{<join condition>}} S$
- The result of the JOIN is a relation Q with n + m attributes Q(A1, A2, … , An, B1, B2, … , Bm) in that order; Q has one tuple for each combination of tuples—one from R and one from S—whenever the combination satisfies the join condition.
- The main difference between CARTESIAN PRODUCT and JOIN are, 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.

Unary and Binary relational operations

- The join condition is specified on attributes from the two relations R and S and is evaluated for each combination of tuples.
- Each tuple combination for which the join condition evaluates to TRUE is included in the resulting relation Q as a single combined tuple.
- A general join condition is of the form <condition>AND<condition> AND ... AND<condition> where each <condition> is of the form Ai θ Bj , Ai is an attribute of R, Bj is an attribute of S, Ai and Bj have the same domain, and θ (theta) is one of the comparison operators {=, <,>,< , ≥, ≠}.
- A JOIN operation with such a general join condition is called a THETA JOIN.

Unary and Binary relational operations

Variations of JOIN:

- **The EQUIJOIN and NATURAL 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.
- Notice that in the result of an EQUIJOIN we always have one or more pairs of attributes that have identical values in every tuple.
- For example, the values of the attributes Mgr_ssn and Ssn are identical in every tuple of DEPT_MGR (the EQUIJOIN result) because the equality join condition specified on these two attributes requires the values to be identical in every tuple in the result.
- 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.

Unary and Binary relational operations

- 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.
- 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:

PROJ_DEPT ← PROJECT * ρ(Dname, Dnum, Mgr_ssn, Mgr_start_date)(DEPARTMENT)

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

DEPT ← ρ(Dname, Dnum, Mgr_ssn, Mgr_start_date)(DEPARTMENT)

PROJ_DEPT ← PROJECT * DEPT

Unary and Binary relational operations

- 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.
- In general, the join condition for NATURAL JOIN is constructed by equating each pair of join attributes that have the same name in the two relations and combining these conditions with AND.
- A single JOIN operation is used to combine data from two relations so that related information can be presented in a single table. These operations are also known as inner joins.
- A more general, but nonstandard definition for NATURAL JOIN is Q ← R *(list1),(list2)S

Unary and Binary relational operations

- In this case,<list1> specifies a list of i attributes from R, and <list2>specifies a list of i attributes from S.
- The NATURAL JOIN or EQUIJOIN operation can also be specified among multiple tables, leading to an n-way join.
- For example, consider the following three-way join:

$((\text{PROJECT} \bowtie_{\text{Dnum=Dnumber}} \text{DEPARTMENT}) \bowtie_{\text{Mgr\_ssn=Ssn}} \text{EMPLOYEE})$

Unary and Binary relational operations

A Complete Set of Relational Algebra Operations

- It has been shown that the set of relational algebra operations {σ, π, ∪, ρ, –, ×} is a complete set; that is, any of the other original relational algebra operations can be expressed as a sequence of operations from this set.
- For example, the INTERSECTION operation can be expressed by using UNION and MINUS as follows: $R \cap S \equiv (R \cup S) - ((R - S) \cup (S - R))$
- JOIN operation can be specified as a CARTESIAN PRODUCT followed by a SELECT operation: $R \bowtie_{<condition>} S \equiv \sigma_{<condition>}(R \times S)$
- A NATURAL JOIN can be specified as a CARTESIAN PRODUCT preceded by RENAME and followed by SELECT and PROJECT operations.

Unary and Binary relational operations

The DIVISION Operation

- The DIVISION operation, denoted by ÷, is useful for a special kind of query that sometimes occurs in database applications.
- 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:
- $SMITH \leftarrow \sigma_{Fname='John' \text{ AND } Lname='Smith'}(EMPLOYEE)$
- $SMITH\_PNOS \leftarrow \pi_{Pno}(WORKS\_ON \bowtie_{Essn=Ssn} SMITH)$

Unary and Binary relational operations

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

Unary and Binary relational operations

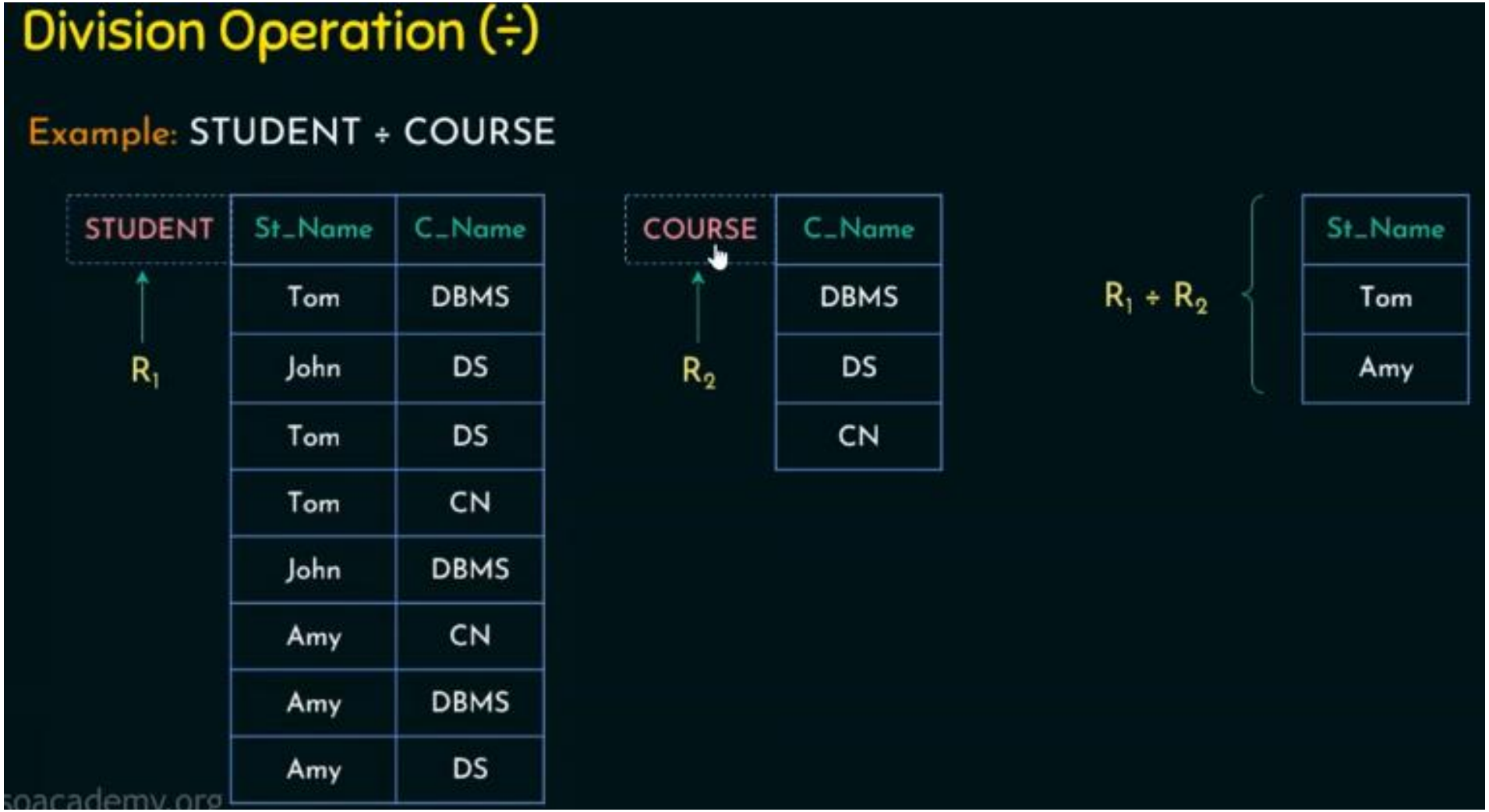
- In general, the DIVISION operation is applied to two relations R(Z) ÷ S(X), where the attributes of R are a subset of the attributes of S; that is, X ⊆ Z. Let Y be the set of attributes of R that are not attributes of S.
- The DIVISION operation is defined for convenience for dealing with queries that involve universal quantification or the all condition.

Unary and Binary relational operations

Division Operation (÷)

- ☆ Denoted by ÷ or /.
- ☆ Used for queries which involve the 'all' or 'every'.
- ☆ $R_1 \div R_2$ = Tuples of R_1 associated with all tuples of R_2.
- ☆ $R_1 \div R_2$ is possible, if and only if $R_2 \subset R_1$.
- ☆ $R_2 \subset R_1 \rightarrow R_1 \neq R_2$ and every attribute of R_2 should be present in R_1.

Unary and Binary relational operations



Unary and Binary relational operations

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\text{<selection condition>}}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\text{<attribute list>}}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\text{<join condition>}} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\text{<join condition>}} R_2$, OR $R_1 \bowtie_{\text{(<join attributes 1>), (<join attributes 2>)}} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{\text{<join condition>}} R_2$, OR $R_1 *_{\text{(<join attributes 1>), (<join attributes 2>)}} R_2$ OR $R_1 * R_2$

Unary and Binary relational operations

UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Unary and Binary relational operations

Notation for Query Trees

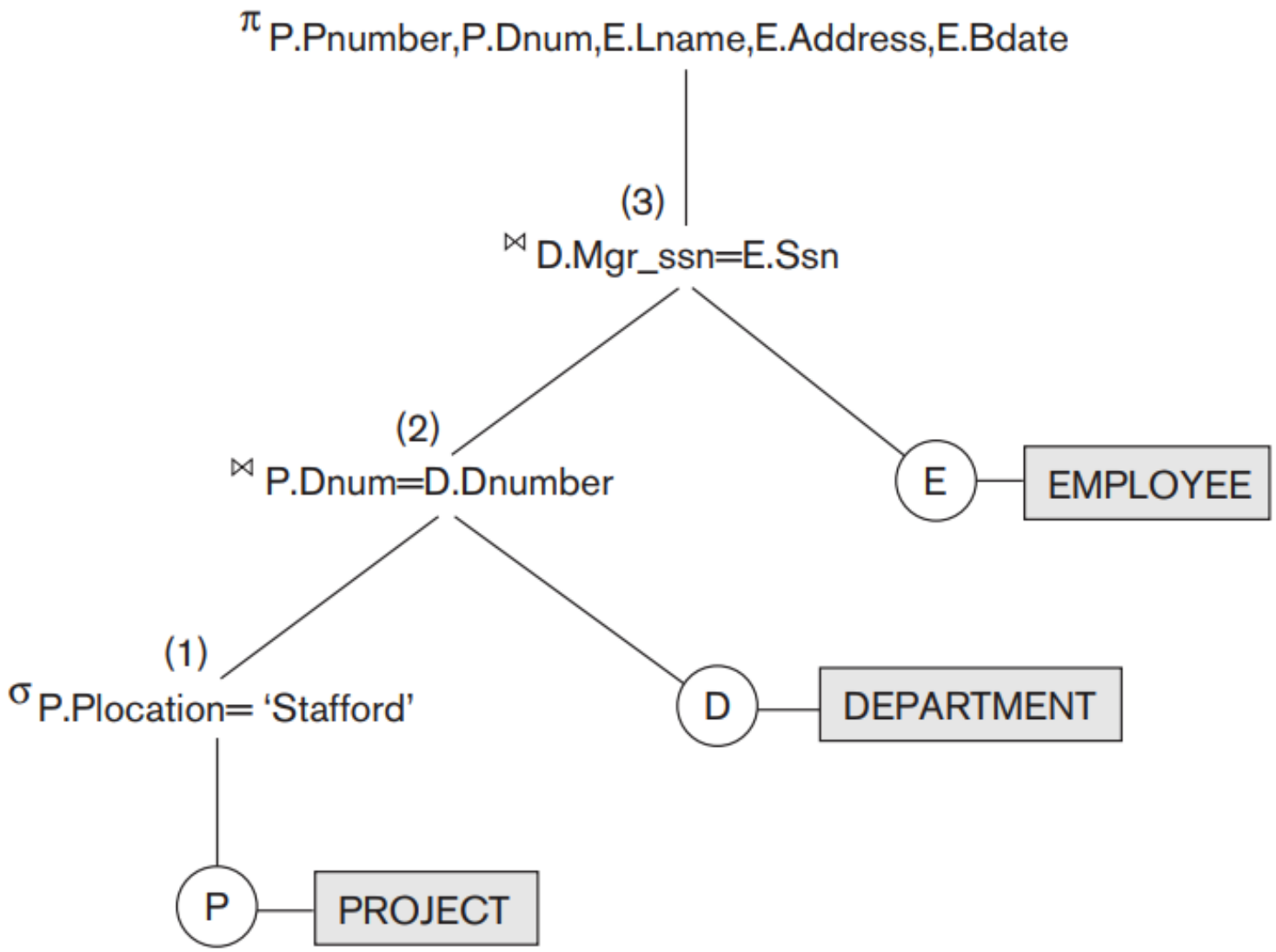
- A query tree is a tree data structure that corresponds to a relational algebra expression.
- It represents the input relations of the query as leaf nodes of the tree and represents the relational algebra operations as internal nodes.
- It is known as a query evaluation tree or query execution tree.
- It includes the relational algebra operations being executed and is used as a possible data structure for the internal representation of the query in an RDBMS.

$$\pi_{\text{Pnumber, Dnum, Lname, Address, Bdate}}(((\sigma_{\text{Plocation='Stafford'}}(\text{PROJECT})) \bowtie_{\text{Dnum=Dnumber}}(\text{DEPARTMENT})) \bowtie_{\text{Mgr\_ssn=Ssn}}(\text{EMPLOYEE}))$$

Unary and Binary relational operations

- 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.

Unary and Binary relational operations



$$\pi_{\text{Pnumber, Dnum, Lname, Address, Bdate}}(((\sigma_{\text{Plocation='Stafford'}}(\text{PROJECT})) \bowtie_{\text{Dnum=Dnumber}}(\text{DEPARTMENT})) \bowtie_{\text{Mgr\_ssn=Ssn}}(\text{EMPLOYEE}))$$

Additional relational operations (aggregate, grouping, etc.)

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: $\pi_{F1, F2, ..., Fn}$ (R), where F1, F2, … , Fn are functions over the attributes in relation R and may involve arithmetic operations and constant values.
- Helpful when developing reports where computed values have to be produced in the columns of a query result.

Additional relational operations (aggregate, grouping, etc.)

- Consider the relation EMPLOYEE (Ssn, Salary, Deduction, Years_service)
- A report may be required to show
  Net Salary = Salary – Deduction,
  Bonus = 2000 * Years_service, and
  Tax = 0.25 * Salary
- Then a generalized projection combined with renaming may be used as follows:
- REPORT ← $\rho_{(Ssn,\ Net\_salary,\ Bonus,\ Tax)}(\pi_{Ssn,\ Salary - Deduction,\ 2000 * Years\_service,\ 0.25 * Salary}(EMPLOYEE))$

Additional relational operations (aggregate, grouping, etc.)

Aggregate Functions and Grouping

- Another type of request that cannot be expressed in the basic relational algebra is to specify mathematical aggregate functions on collections of values from the database.
- Examples of such functions include retrieving the average or total salary of all employees or the total number of employee tuples.
- Common functions applied to collections of numeric values include SUM, AVERAGE, MAXIMUM, and MINIMUM.
- The COUNT function is used for counting tuples or values.

Additional relational operations (aggregate, grouping, etc.)

- Grouping the tuples in a relation by the value of some of their attributes and then applying an aggregate function independently to each group.
- We can define an AGGREGATE FUNCTION operation, using the symbol I (pronounced script F), to specify these types of requests as follows: $_{\text{<grouping attributes>}}\mathfrak{I}_{\text{<function list>}}(R)$
- where <grouping attributes> is a list of attributes of the relation specified in R, and <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

Additional relational operations (aggregate, grouping, etc.)

- To retrieve each department number, the number of employees in the department, and their average salary, while renaming the resulting attributes
- $\rho_{R(Dno,\ No\_of\_employees,\ Average\_sal)}\ (_{Dno}\ \Im\ _{COUNT\ Ssn,\ AVERAGE\ Salary}\ (EMPLOYEE))$
- 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 <function>_<attribute>.

Additional relational operations (aggregate, grouping, etc.)

(a) R

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

(b)

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

(c)

Count_ssn	Average_salary
8	35125

Figure 8.10
The aggregate function operation.

a. $\rho_{R(\text{Dno, No\_of\_employees, Average\_sal})}(_{\text{Dno}}\Im_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE}))$.

b. $_{\text{Dno}}\Im_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE})$.

c. $\Im_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE})$.

Additional relational operations (aggregate, grouping, etc.)

- 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.

Additional relational operations (aggregate, grouping, etc.)

Recursive Closure Operations

- Another type of operation that, in general, cannot be specified in the basic original relational algebra is recursive closure.
- This operation is applied to a recursive relationship between tuples of the same type, such as the relationship between an employee and a supervisor.
- This relationship is described by the foreign key Super_ssn of the EMPLOYEE relation.

Additional relational operations (aggregate, grouping, etc.)

- An example of a recursive operation is to retrieve all supervisees of an employee e at all levels—that is, all employees e' directly supervised by e, all employees e' directly supervised by each employee e', all employees e''' directly supervised by each employee e'', and so on.
- For example, to specify the Ssns of all employees e' directly supervised—at level one—by the employee e whose name is ‘James Borg’.

  $BORG\_SSN \leftarrow \pi_{Ssn}(\sigma_{Fname='James' \text{ AND } Lname='Borg'}(EMPLOYEE))$

  $SUPERVISION(Ssn1, Ssn2) \leftarrow \pi_{Ssn,Super\_ssn}(EMPLOYEE)$

  $RESULT1(Ssn) \leftarrow \pi_{Ssn1}(SUPERVISION\ _{Ssn2=Ssn}BORG\_SSN)$

Additional relational operations (aggregate, grouping, etc.)

- To retrieve all employees supervised by Borg at level 2—that is, all employees e'' supervised by some employee e' who is directly supervised by Borg—we can apply another JOIN to the result of the first query, as follows:
- RESULT2(Ssn) ← π_{Ssn1}(SUPERVISION $\bowtie_{Ssn2=Ssn}$ RESULT1)
- To get both sets of employees supervised at levels 1 and 2 by ‘James Borg’, we can apply the UNION operation to the two results, as follows: RESULT ← RESULT2 ∪ RESULT1

Additional relational operations (aggregate, grouping, etc.)

- We cannot specify a query such as “retrieve the supervisees of ‘James Borg’ at all levels” without utilizing a looping mechanism unless we know the maximum number of levels.
- An operation called the transitive closure of relations has been proposed to compute the recursive relationship as far as the recursion proceeds.

Additional relational operations (aggregate, grouping, etc.)

SUPERVISION

(Borg's Ssn is 888665555)

(Ssn)	(Super_ssn)
Ssn1	Ssn2
123456789	333445555
333445555	888665555
999887777	987654321
987654321	888665555
666884444	333445555
453453453	333445555
987987987	987654321
888665555	null

RESULT1

Ssn
333445555
987654321

(Supervised by Borg)

RESULT2

Ssn
123456789
999887777
666884444
453453453
987987987

(Supervised by Borg's subordinates)

RESULT

Ssn
123456789
999887777
666884444
453453453
987987987
333445555
987654321

(RESULT1 ∪ RESULT2)

Additional relational operations (aggregate, grouping, etc.)

OUTER JOIN Operations

- 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 type of join, where tuples with no match are eliminated, is known as an inner join.
- This amounts to the loss of information if the user wants the result of the JOIN to include all the tuples in one or more of the component relations.
- A set of operations, called outer joins, were developed for the case where the user wants 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.

Additional relational operations (aggregate, grouping, etc.)

- 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.
- Suppose that we want a list of all employee names as well as the name of the departments they manage if they happen to manage a department; if they do not manage one, we can indicate it with a NULL value.

RESULT

Fname	Minit	Lname	Dname
John	B	Smith	NULL
Franklin	T	Wong	Research
Alicia	J	Zelaya	NULL
Jennifer	S	Wallace	Administration
Ramesh	K	Narayan	NULL
Joyce	A	English	NULL
Ahmad	V	Jabbar	NULL
James	E	Borg	Headquarters

Additional relational operations (aggregate, grouping, etc.)

Example of Left Outer Join

$$TEMP \leftarrow (EMPLOYEE \ ⟕_{Ssn=Mgr\_ssn} DEPARTMENT)$$

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

Additional relational operations (aggregate, grouping, etc.)

- A similar operation, RIGHT OUTER JOIN, denoted by, keeps every tuple in the second, or right, relation S in the result of R S.
- A third operation, FULL OUTER JOIN, denoted by , keeps all tuples in both the left and the right relations when no matching tuples are found, padding them with NULL values as needed.

Additional relational operations (aggregate, grouping, etc.)

The OUTER UNION Operation

- The OUTER UNION operation was developed to take the union of tuples from two relations that have some common attributes but are not union (type) compatible.
- This operation will take the UNION of tuples in two relations R(X, Y) and S(X, Z) that are partially compatible, meaning that only some of their attributes, say X, are union compatible.
- The attributes that are union compatible are represented only once in the result, and those attributes that are not union compatible from either relation are also kept in the result relation T(X, Y, Z).
- It is therefore the same as a FULL OUTER JOIN on the common attributes.

Examples of Queries in relational algebra

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

Examples of Queries in relational algebra

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

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

Examples of Queries in relational algebra

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

$$RESEARCH\_DEPT \leftarrow \sigma_{Dname='Research'}(DEPARTMENT)$$

$$RESEARCH\_EMPS \leftarrow (RESEARCH\_DEPT \bowtie_{Dnumber=Dno} EMPLOYEE)$$

$$RESULT \leftarrow \pi_{Fname,\ Lname,\ Address}(RESEARCH\_EMPS)$$

As a single in-line expression, this query becomes:

$$\pi_{Fname,\ Lname,\ Address}\ (\sigma_{Dname='Research'}(DEPARTMENT \bowtie_{Dnumber=Dno}\ (EMPLOYEE))$$

Examples of Queries in relational algebra

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

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

Examples of Queries in relational algebra

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

$DEPT5\_PROJS \leftarrow \rho_{(Pno)}(\pi_{Pnumber}(\sigma_{Dnum=5}(PROJECT)))$

$EMP\_PROJ \leftarrow \rho_{(Ssn,\ Pno)}(\pi_{Essn,\ Pno}(WORKS\_ON))$

$RESULT\_EMP\_SSNS \leftarrow EMP\_PROJ \div DEPT5\_PROJS$

$RESULT \leftarrow \pi_{Lname,\ Fname}(RESULT\_EMP\_SSNS * EMPLOYEE)$

Examples of Queries in relational algebra

List the names of all employees with two or more dependents

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

Examples of Queries in relational algebra

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

$T1(Ssn, No\_of\_dependents) \leftarrow {}_{Essn}\Im_{COUNT\ Dependent\_name}(DEPENDENT)$

$T2 \leftarrow \sigma_{No\_of\_dependents>2}(T1)$

$RESULT \leftarrow \pi_{Lname,\ Fname}(T2 * EMPLOYEE)$

Examples of Queries in relational algebra

Retrieve the names of employees who have no dependents.

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

Examples of Queries in relational algebra

Retrieve the names of employees who have no dependents.

$ALL\_EMPS \leftarrow \pi_{Ssn}(EMPLOYEE)$

$EMPS\_WITH\_DEPS(Ssn) \leftarrow \pi_{Essn}(DEPENDENT)$

$EMPS\_WITHOUT\_DEPS \leftarrow (ALL\_EMPS - EMPS\_WITH\_DEPS)$

$RESULT \leftarrow \pi_{Lname,\ Fname}(EMPS\_WITHOUT\_DEPS * EMPLOYEE)$

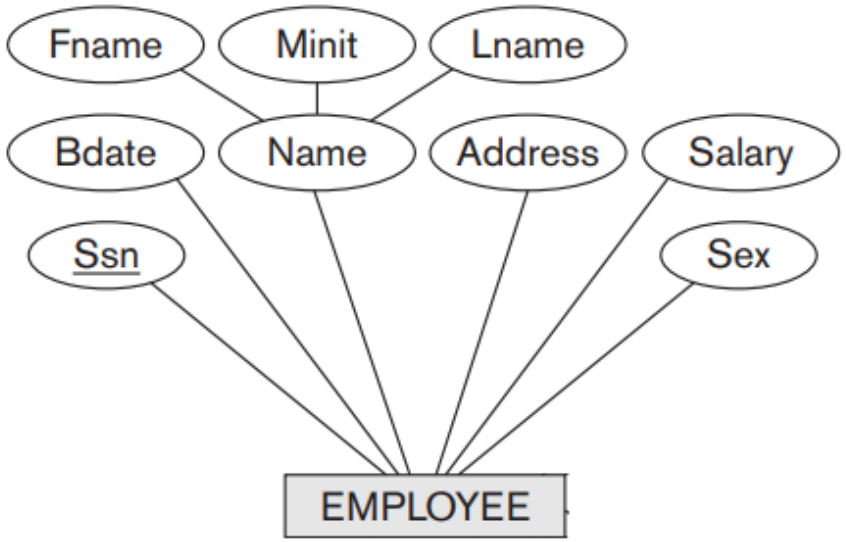
Module 2 – Chapter 3

Relational Database Design using ER-to-Relational mapping

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. If the chosen key of E is a composite, then the set of simple attributes that form it will together form the primary key of R.

Relational Database Design using ER-to-Relational mapping



- If multiple keys were identified for E during the conceptual design, the information describing the attributes that form each additional key is kept to specify additional (unique) keys of relation R.
- Knowledge about keys is also kept for indexing purposes and other types of analyses.

EMPLOYEE

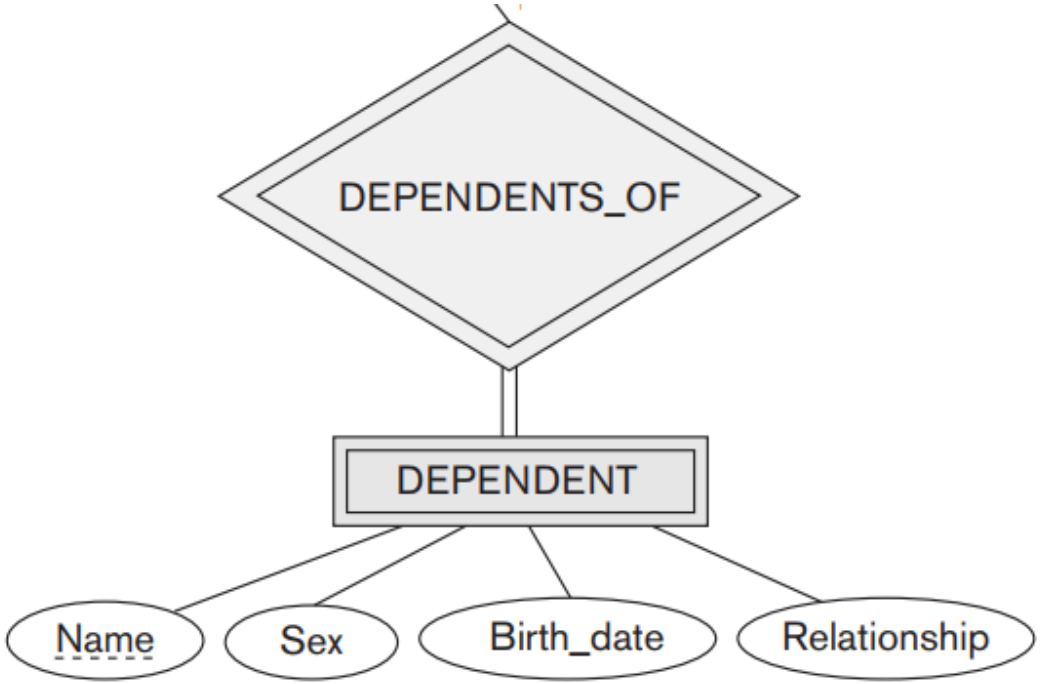
Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

Relational Database Design using ER-to-Relational mapping

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.
- 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.

Relational Database Design using ER-to-Relational mapping



DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Relational Database Design using ER-to-Relational mapping

Step 3: Mapping of Binary 1:1 Relationship Types

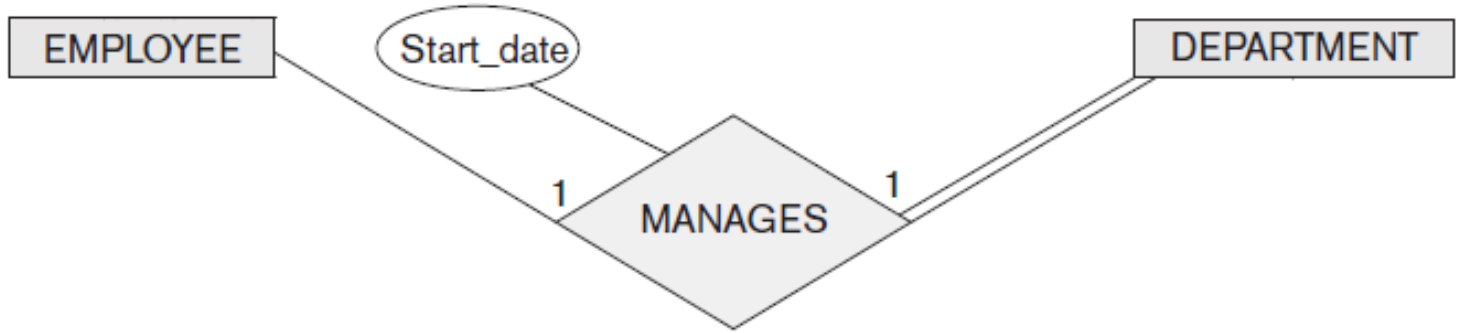
- For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R.
- There are three possible approaches:
  (1) the foreign key approach
  (2) the merged relationship approach
  (3) the cross reference or relationship relation approach

Relational Database Design using ER-to-Relational mapping

Foreign key approach:

- Choose one of the relations—S, 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.
- We map the 1:1 relationship type MANAGES 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.

Relational Database Design using ER-to-Relational mapping



DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

Relational Database Design using ER-to-Relational mapping

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.

Relational Database Design using ER-to-Relational mapping

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.
- This approach is required for binary M:N relationships. The relation R is called a relationship relation (lookup table).

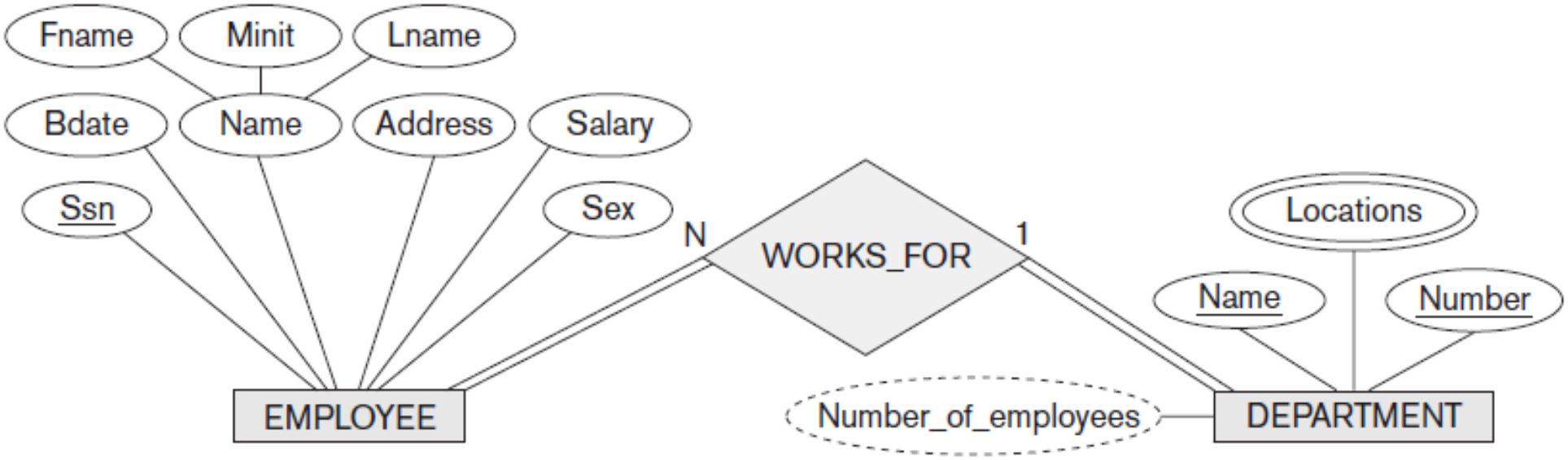
Relational Database Design using ER-to-Relational mapping

Step 4: Mapping of Binary 1:N Relationship Types

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.
- For WORKS_FOR we include the primary key Dnumber of the DEPARTMENT relation as foreign key in the EMPLOYEE relation and call it Dno.

Relational Database Design using ER-to-Relational mapping



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

Relational Database Design using ER-to-Relational mapping

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.
- This is only recommended when few relationship instances exist, in order to avoid NULL values in foreign keys.

Relational Database Design using ER-to-Relational mapping

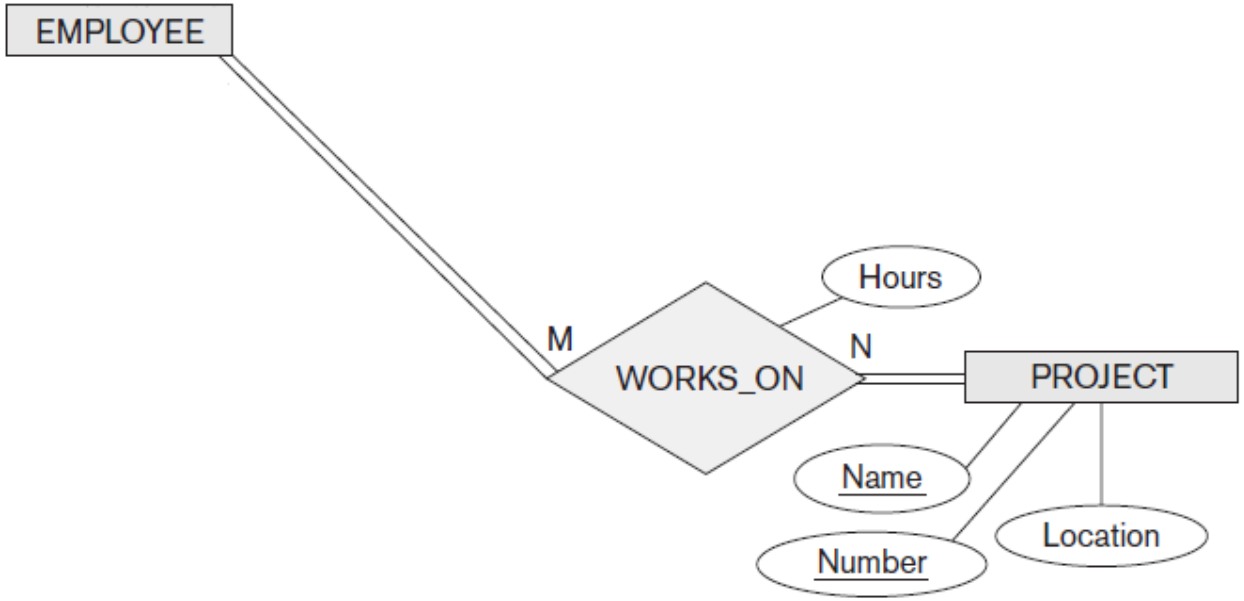
Step 5: Mapping of Binary M:N Relationship Types

- 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.

Relational Database Design using ER-to-Relational mapping

- Map the M:N relationship type WORKS_ON by creating the relation WORKS_ON.
- We include the primary keys of the PROJECT and EMPLOYEE relations as foreign keys in WORKS_ON and rename them Pno and Essn, respectively.
- The primary key of the WORKS_ON relation is the combination of the foreign key attributes {Essn, Pno}.

Relational Database Design using ER-to-Relational mapping



WORKS_ON

Essn	Pno	Hours

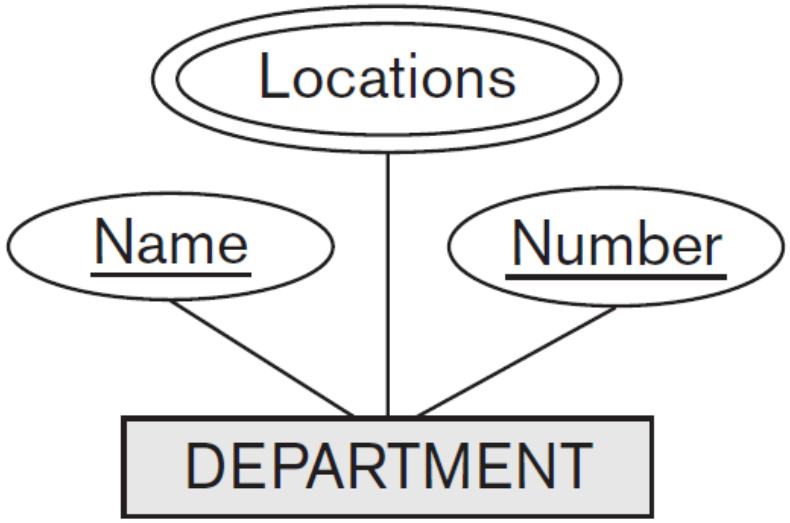
Relational Database Design using ER-to-Relational mapping

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.

Relational Database Design using ER-to-Relational mapping

- We create a relation DEPT_LOCATIONS.
- 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}.



DEPT_LOCATIONS

Dnumber	Dlocation

Relational Database Design using ER-to-Relational mapping

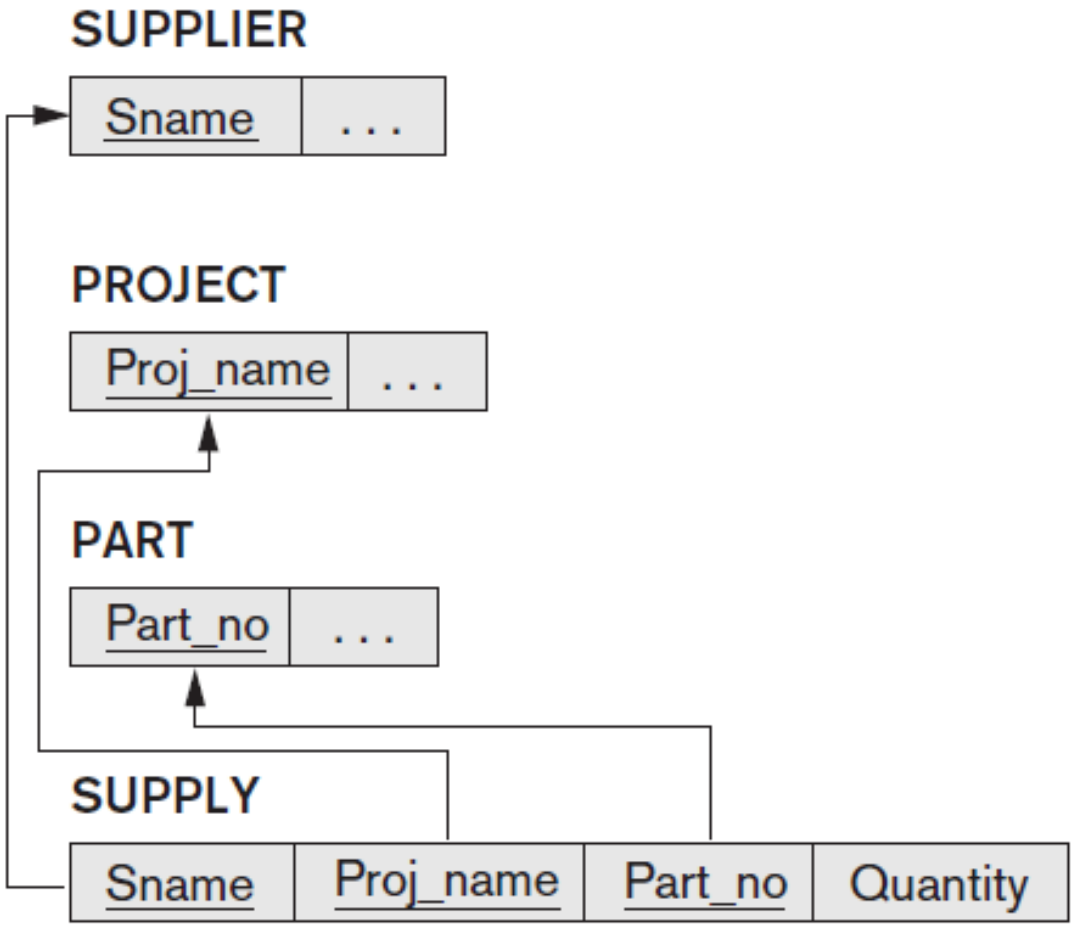
Relational Database Design using ER-to-Relational mapping

Step 7: Mapping of N-ary Relationship Types

- 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.
- The primary key of S is usually a combination of all the foreign keys that reference the relations representing the participating entity types.

Relational Database Design using ER-to-Relational mapping

- Ternary relationship type SUPPLY, which relates a SUPPLIER s, PART p, and PROJECT j.
- Primary key is the combination of the three foreign keys {Sname, Part_no, Proj_name}.



End of Module 2