

Part 1

Introduction

- Traditional Database Applications
- Basic Definitions
- DBMS functionality
- Database System
- Example of a Database
- Database Users
- Advantages of using DBMS
- Data Models and 3 Schema Architecture
- Database Schema vs Database State
- DB System Architectures

Prepared By: Dr. Osama Al-Haj Hassan

Reference to: Fundamentals of Database Systems , Ramez Elmasri and Shamkant B. Navathe, Fifth Edition.

Traditional Database Applications

Traditional Database Applications are DB applications in which accessed or retrieved data is either textual or numeric

Examples:

- Bank Systems
- Hotel Reservations
- Airline Reservations
- Computerized Library Catalog
- E-commerce websites

Basic Definitions

- **Database (DB):** A collection of related data
- **Mini World (Universe of Discourse (UoD)):** Part of the world about which a database stores data.
 - Example: A database that stores data about students, instructors, departments. This database is concerned about university related information. In this case, the mini world of this database is “University Environment”.

Basic Definitions

- **Database Management System (DBMS):** A collection of programs that enable users to create and maintain a database. Examples:
 - Oracle (Popular with most programming languages)
 - SQL Server (Most popular when working with .Net languages)
 - MySQL (Most popular when working with php or java languages)
 - MS Access (Most popular with .Net language. It is better to use it for simple applications)
- **Database System:** DBMS + DB
- **Meta-Data:** is a description of database

Typical DBMS Functionality

- 1) **Defining a database** (data types, structure, constraints).
 - Eg. student ID + address + major are all related together.
 - Eg. Student id should be numeric.
 - Eg. Student grade > 0 and < 100 .
- 2) **Constructing a database:** storing actual data on a storage medium controlled by the DBMS.
 - Eg. Stores data on secondary storage medium such as hard disk.
- 3) **Manipulating a database:**
 - Querying (retrieving) data from DB.
 - Inserting data in DB.
 - Updating data in DB.
 - Generating reports from data.

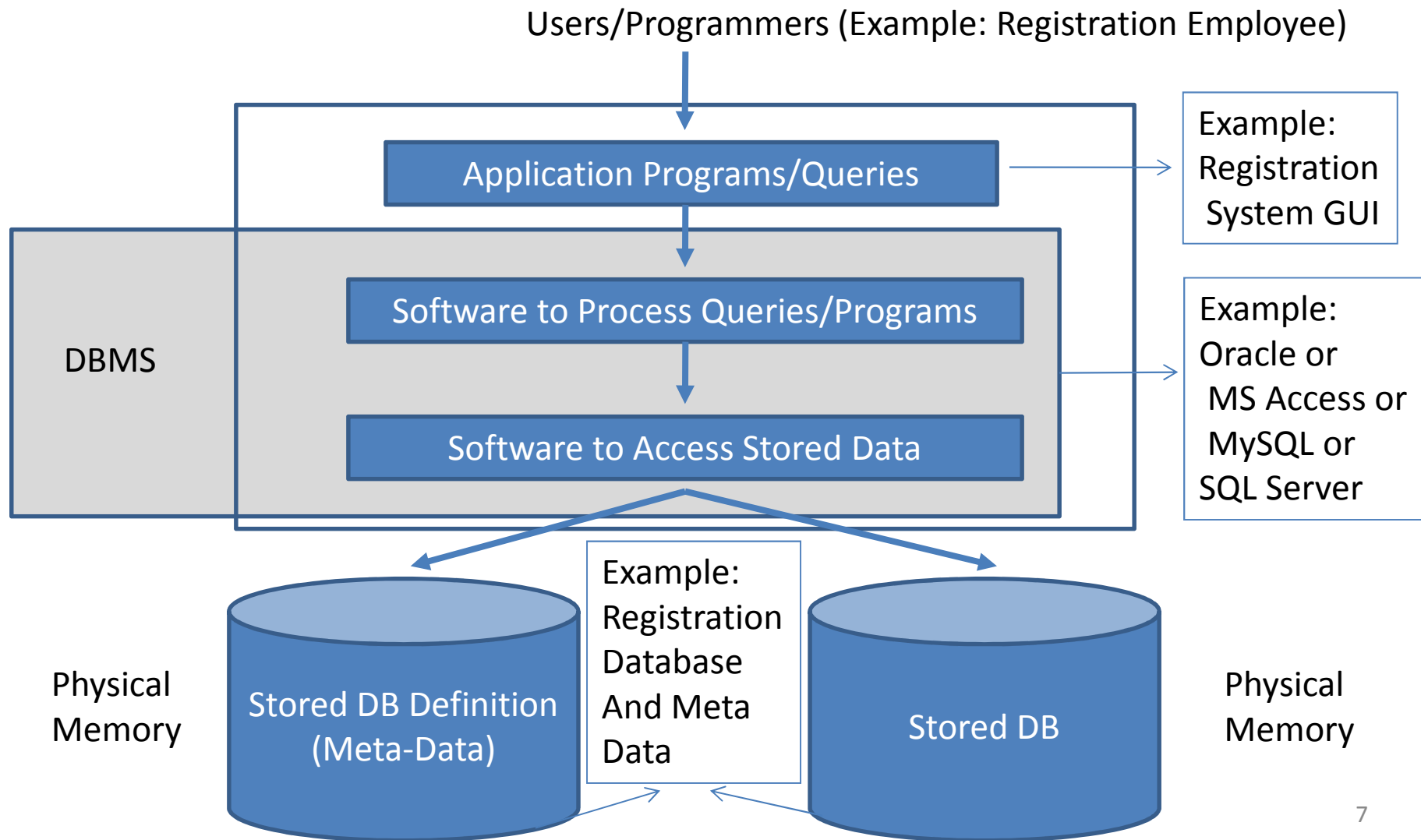
Typical DBMS Functionality

- 4) **Sharing a database:** allows multiple users and programs to access the DB “concurrently”. At the same time data should remain valid and consistent.

- 5) **Protection:** protecting the DB from:
 - Software or Hardware malfunction or crashes.
 - Unauthorized or malicious access.

- 6) **Presentation and visualization of data**
 - Presents data for users in away they can easily understand.

A simplified DB System Environment



Example of a DB

- Mini world of the DB: University Environment
- Mini world entities in the Example:
 - Students
 - Courses
 - Sections (of courses)
 - Academic Departments
 - Instructors

Example of a DB

- Some mini world relationships:
 - Courses **have** sections
 - Students **register for** a section
 - Instructors **teach** sections
 - Courses **have** prerequisite courses
 - Courses are **offered by** departments
 - Students **major in** departments

Note: This Conceptual data model can be translated into entity-relationship data model (will be explained later).

Example of a DB

Student	Name	Student Number	Class	Major
	Ahmad	45	1	CS
	Kamal	76	2	CIS

Course	Course Name	Course Number	Credit Hours	Department
	Data Structures	CS456	3	CS
	Database 1	CS467	3	CS
	Calculus 1	Math547	3	Math

Section	Section Identifier	Course Number	Semester	Year	Instructor
	3	CS456	First	2010	Dr. Ahmad
	1	Math547	Summer	2008	Dr. Sami

Example of a DB

Grade Report	Student Number	Section Identifier	Grade
	45	1	90
	45	3	78

Prerequisite	Course Number	Prerequisite Number
	CS456	CS783
	CS467	CS542
	Math547	Math339

Database Users

1) Database Administrator (DBA)

- Authorizes access to the DB.
- Coordinates and monitors DB usage.
- Acquires software and hardware resources as needed.

2) Database Designer

- Identifies data to be stored in the DB.
- Chooses appropriate structure to represent and store data.
- Communicates with different DB users to come up with the best design for the DB.

3) End user

- Users who access the DB to query, update, and generate reports from the DB.

Advantages of using DBMS Approach

- 1) Controlling Redundancy: Data is stored once in most cases and multiple times in few cases.
- 2) Restricting unauthorized access
- 3) Providing Persistent Storage
- 4) Providing Backup and Recovery

Problems of Redundancy

- a) Repeating actions multiple times (duplication of effort).
 - Eg. Registration people add new student and
 - Accounting people add the same new student
- b) Waste of storage space
- c) Inconsistent Data: different value for the data that is supposed to be identical
 - Eg. Registration people enter a student birth date as Jan-29-1999 and
 - Accounting people enter the same date as Jan-19-1999

Categories of Data Models

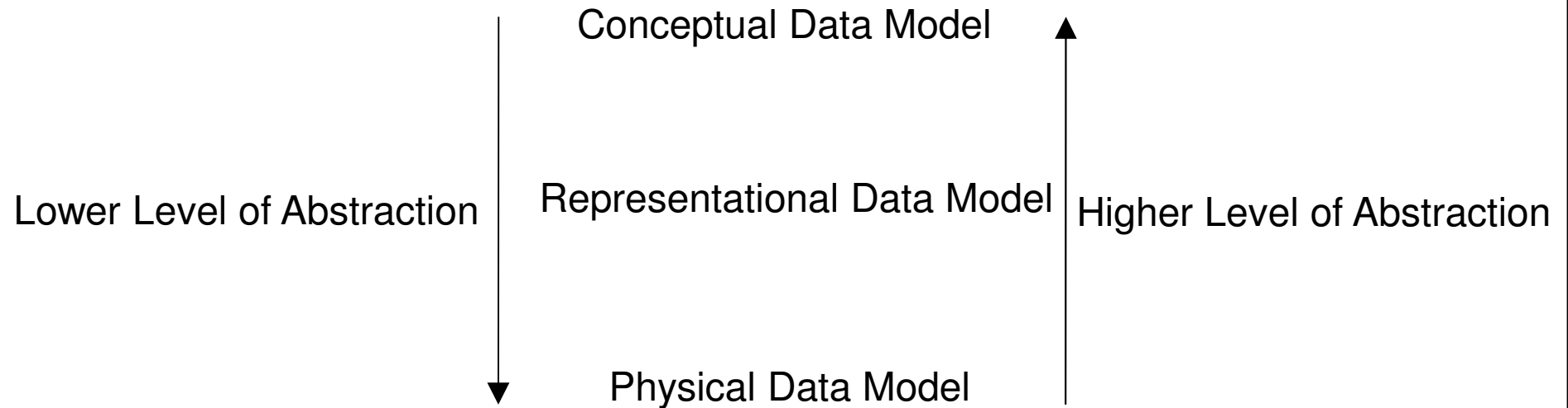
3 Schema Architecture

Data Model	Corresponding Schema
Conceptual (High Level)	External Schema
Representational (Implementation)	Conceptual Schema
Physical (Low Level)	Internal Schema

Data Models

- **Conceptual Model:** eg. Registration database consists of information related students, instructors, courses, ... etc
- **Representational Model:** eg. implementing registration database as tables using DBMS such as Oracle
- **Physical Model:** eg. Defining an index on an attribute to make search more efficient

Abstraction in Data Models



If we are hiding more information, level of data abstraction increases

Schema

- DB Schema:
 - Description of the DB.
 - It is usually specified during **design stage** and is **not expected to change** frequently.

Schema

Entity (Schema Construct)

STUDENT

Name	StudentNumber	Class	Major
------	---------------	-------	-------

Table

COURSE

CourseName	CourseNumber	CreditHours	Department
------------	--------------	-------------	------------

PREREQUISITE

CourseNumber	PrerequisiteNumber
--------------	--------------------

SECTION

SectionIdentifier	CourseNumber	Semester	Year	Instructor
-------------------	--------------	----------	------	------------

GRADE_REPORT

StudentNumber	SectionIdentifier	Grade
---------------	-------------------	-------

Attribute (field),(Column)

Schema

FIGURE 2.1 Schema diagram for the database in Figure 1.2.

Database State

- Database State (Snapshot): It is the data in the DB at a particular moment in time.
- Database state changes frequently while DB schema does not change frequently.
- Database state contains instances (occurrences) of data.

Database State

STUDENT	Name	StudentNumber	Class	Major
	Smith	17	1	CS
	Brown	8	2	CS

Record (tuple) (instance) (occurrence)

COURSE	CourseName	CourseNumber	CreditHours	Department
	Intro to Computer Science	CS1310	4	CS
	Data Structures	CS3320	4	CS
	Discrete Mathematics	MATH2410	3	MATH
	Database	CS3380	3	CS

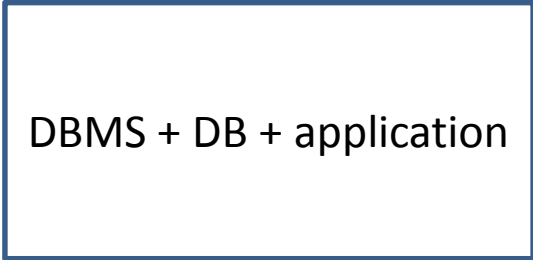
- Every time we perform an update operation, we get a new DB state.

DBMS Architectures

(Centralized Architecture)

- The DBMS, DB, and applications to access DB all exist on one machine.

One Computer



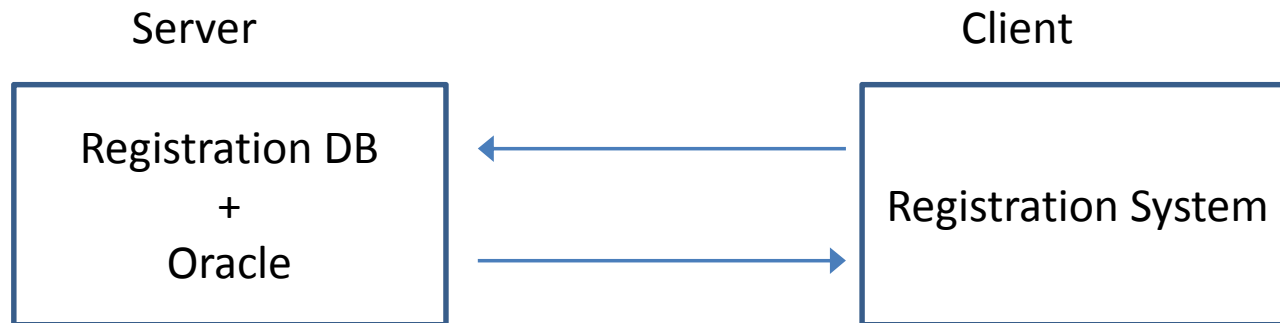
DBMS + DB + application

The diagram illustrates a centralized architecture. It features a blue rectangular box containing the text 'DBMS + DB + application'. Above this box, the text 'One Computer' is centered, indicating that all database components are housed on a single machine.

DBMS Architectures

(2-tier Architecture)

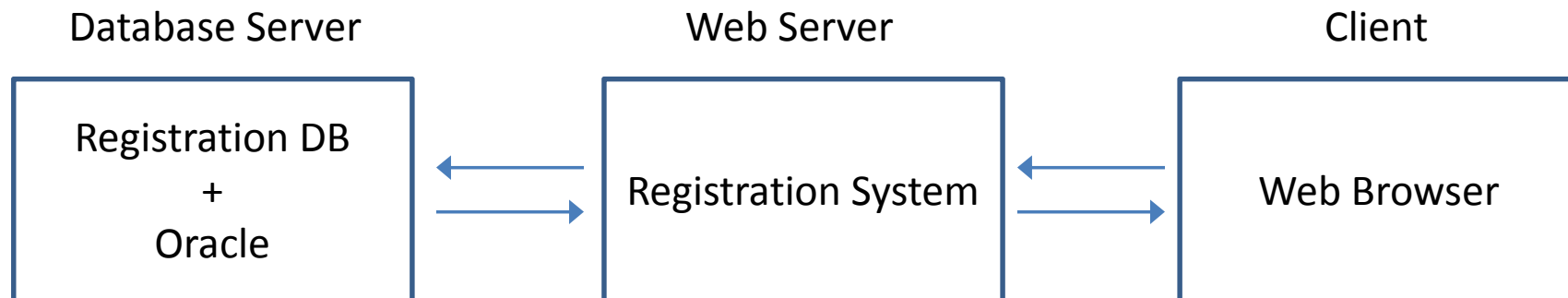
- The DBMS and DB exist on Server.
- The application exists on Client
- This is very popular when the application is a desktop application (can be developed using .Net, java, ... etc)



DBMS Architectures

(3-tier Architecture)

- The DBMS and DB exist on Database Server.
- The application exists on Web Server (Application Server)
- The Client uses a browser (eg. Google Chrome) to access system
- This is very popular with Web Applications (It can be designed using ASP.Net, php, JSP+Servlets, Ruby on Rails, ... etc)



Why tier architectures are good design to follow

- Separation between user interface, business logic rules, and data (DB).
- The ability to change any tier without having to change the other tiers.
- Enhancing security because the client is not directly connected to database.

Part 2

Entity Relationship Diagram (ERD)

- Example: Company Database
- Entity, Attribute, Relationship
- Structural constraints
- Weak entity types
- More ER Examples (Registration DB)
- More ER Examples (Bank DB)
- In class exercise 1: Zoo Database
- In class exercise 2: Library Database
- Alternative Notations for structural constraints (min,max) notation

Prepared By: Dr. Osama Al-Haj Hassan

Reference to: Fundamentals of Database Systems , Ramez Elmasri and Shamkant B. Navathe, Fifth Edition.

Example Database (Company DB)

- Suppose the outcome of requirements collection is as follows.
 - The company is organized into departments. Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.

Example Database (Company DB)

- Suppose the outcome of requirements collection is as follows
 - A department controls a number of projects, each of which has a unique name, a unique number, and a single location.

Example Database (Company DB)

- Suppose the outcome of requirements collection is as follows
 - We store each employee's name, social security number, address, salary, gender, and birth date. An employee is assigned to one department but may work on several projects, which are not necessarily controlled by the same department. We keep track of the number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee.

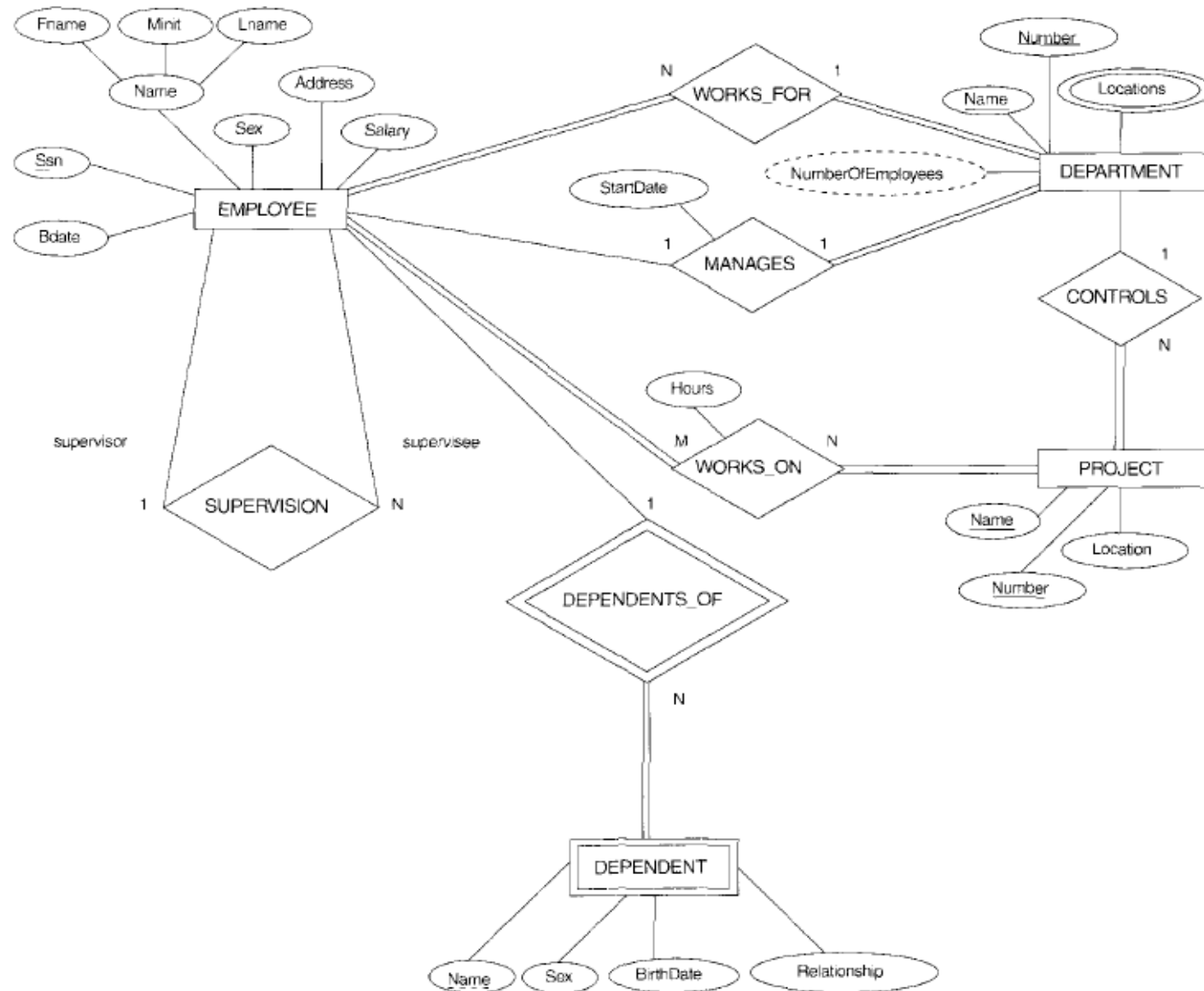
Example Database (Company DB)

- Suppose the outcome of requirements collection is as follows
 - We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's first name, gender, birth date, and relationship to the employee.

Example Database (Company DB)

- The previous requirements can be translated into the following schema represented as “Entity-Relationship (ER) Schema Diagram”.
- This ER diagram is shown in the next slide.

ER Schema diagram for the Company Database



Entities

- Entity:
 - Is a "thing" in the real world with an independent existence (eg. Department, employee, project, ... etc). (This is called Strong Entity)

- It's ER Diagram notation:



ENTITY

- Example: Department Entity



Entity Instance

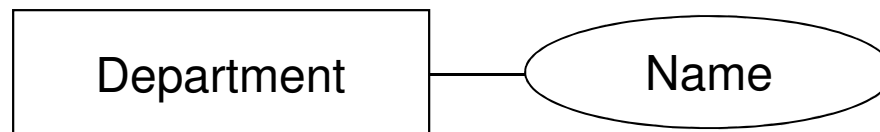
- An Entity is a general type that includes all instances
- An instance is an example of the entity
- Example:
 - Employee is an entity
 - Ahmad is an instance of entity Employee
- Example:
 - Department is an entity
 - CS Department is an instance of entity Department

Attributes

- Attribute:
 - Properties that describe entities (eg. Employee name, department name, ...etc).
 - It's ER Diagram notation



- Attribute “Value Set” or “Domain”
 - Data type associated with the attribute.
 - Example: EmployeeID is an integer.
 - Example: Grade can take letters “A”, “B”, “C”, “D”, and “F”.
- Attribute Example: Name is an attribute of Department



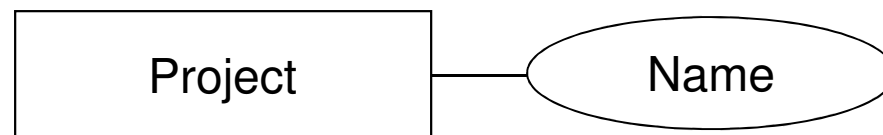
Attribute Types

- Simple (Atomic) Attribute
 - Attributes that are not divisible.
 - It's ER Diagram notation is the same as the general attribute you saw in the previous slide.



ATTRIBUTE

- Example: Project Name is a simple attribute of entity Project

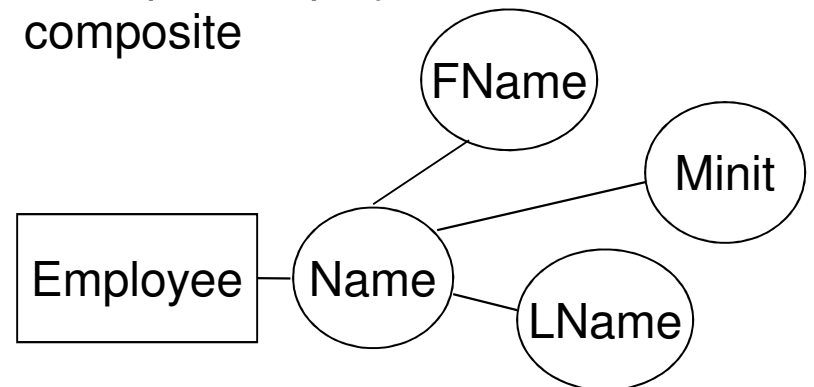


Attribute Types

- Composite Attribute
 - Can be divided into smaller subparts.
 - Example: Address can be derived into “city”, “street”, “building number”, and “apartment number”.
 - It’s ER-Diagram notation:

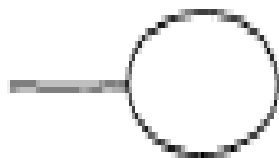


Example: Employee Name can be composite



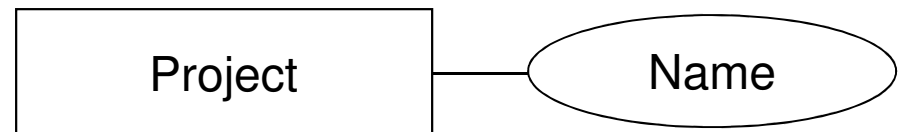
Attribute Types

- Single-Valued Attribute:
 - A single entity has a single value of that attribute.
 - Example: employee ID. A single employee has only one single ID.
 - Its ER Diagram notation is the same as a general attribute .



ATTRIBUTE

A single project has a single name

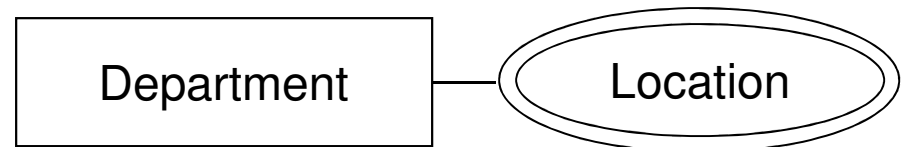


Attribute Types

- Multi-Valued Attribute:
 - A **single** entity has a **multiple** values of that attribute.
 - Example: Employee earned degree. A **single** employee can have **multiple** degrees (B.SC + MS + PhD degrees).
 - Its ER Diagram Notation is **two nested circles**.

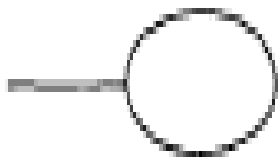


Example: A **single** department can have **multiple** locations



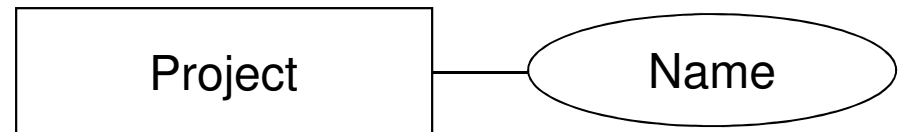
Attribute Types

- Stored Attribute
 - Cannot be derived from any other attribute.
 - Example: Birth Date.
 - Its ER-Diagram Notation is the same as the general attribute notation.



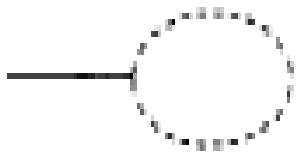
ATTRIBUTE

Example: Name is a stored attribute of project entity



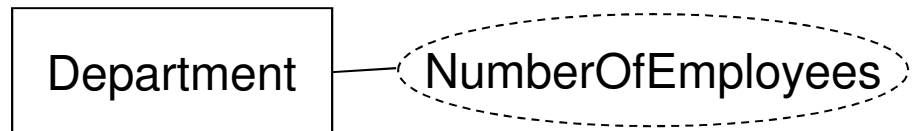
Attribute Types

- Derived Attribute
 - Can be derived from a “Stored Attribute”.
 - Example: employee age can be derived from his birth date and today’s date.
 - Its ER Diagram Notation is **dashed** circle.



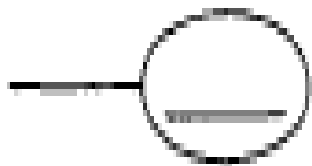
DERIVED ATTRIBUTE

Example: NumberOfEmployees
Is a derived attribute of entity Department.
It is derived from count of records.



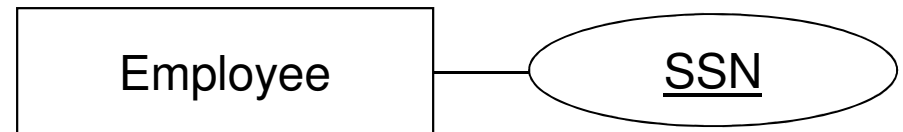
Attribute Types

- Key Attribute:
 - Its value uniquely identifies the entity it belongs to.
 - Example: employee ID uniquely identifies an employee.
 - Its ER Diagram Notation is an **underlined attribute name**.



KEY ATTRIBUTE

Example: SSN is a key attribute of entity Employee



Key Attribute Examples

- What is the key of the following entities?

Entity	Key
Student	Student_id
Employee	Employee_id
Course	Course_number
Section	section_id, course_number, semester, year
Bank branch	bank_id, branch_id
Employee Project	employee_id, project_id

Attribute Types

- Complex Attribute
 - Uses multi-level of nesting in a multi-valued or composite attributes.
 - Example:
 - PreviousDegrees of a STUDENT is a composite multi-valued attribute denoted by {PreviousDegrees(College, Year, Degree, Field)}.
 - Here we used “Multi-value” and “Composite”.
 - “Multi-value”: A single student can have multiple degrees.
 - “Composite”: each degree can be specified by several attributes (college, year, degree, field)

Note

- One attribute can have different types at the same time.
- For example: Project Name is:
 - Stored Attribute.
 - Key Attribute.
 - Single-Valued Attribute.
 - Simple Attribute.

Null Values

- Some attribute values could be optional or may be they are not crucial to have.
- For example, if you have an attribute **Hobbies**. It is OK for the value of this attribute to be missing. In this case, we call it “NULL” value.
- Key Attributes **cannot** be NULL because they uniquely identifies an entity, so they have to have a value.

More about “Key” Attributes

- A key uniquely identifies each entity in Entity Set.
- For example: StudentID is a key for student entity because each student has different (Unique) ID.
- Some times key attributes can be **composite** attributes. This happens when a single attribute cannot satisfy the “Uniqueness Requirement”.
Example:

SectionID, courseID, Semester, Year

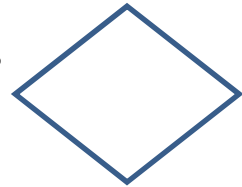
1	,	34234,	first,	2010
1	,	34234,	first,	2009
1	,	44478,	Sum,	2010

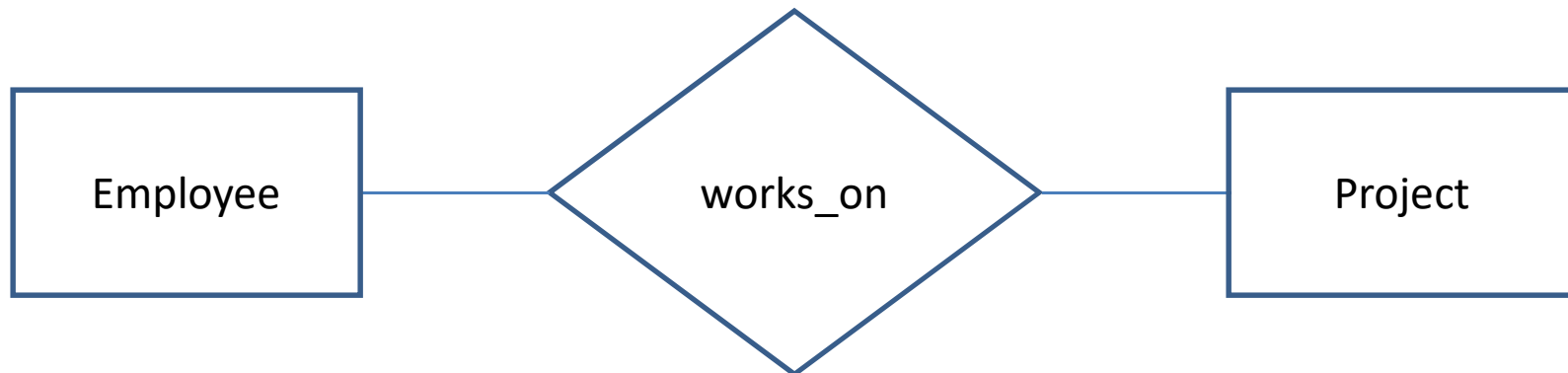
- In this example, Section entity has one key.
- This key consists of combination of 4 attributes.

Keys should be Minimal

- A key is minimal if it cannot be broken into smaller parts that work as a key.
- For example:
 - SectionID, courseID, Semester, Year
 - Is it Minimal?
 - Yes, because none of its smaller parts can work by itself as a key.
- Keys should be minimal.
- “studentID+studentAge”, is it minimal key of Student?
 - No, because “studentID” by itself works as a key.

Relationship

- A Relationship is an association between two or more entities.
- It is represented in ER diagram by using the following shape which is connected to participating entities. 
- Usually, a “verb” is written inside the relationship shape because verbs can express why the entities are connected with each other.

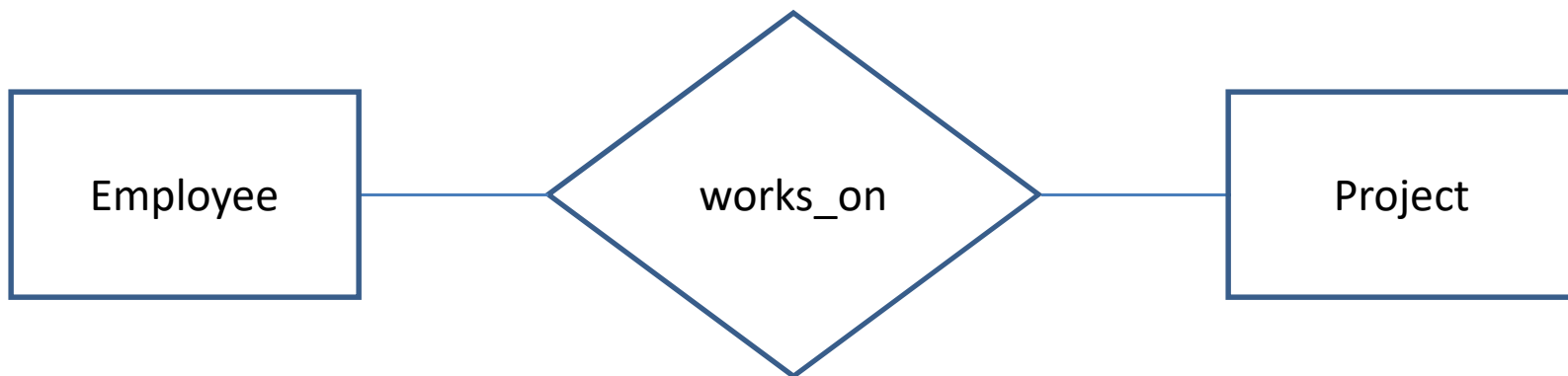


Relationship Degree

- Relationship degree is the number of participating entity types.
- Possible relationship degrees are
 - Binary Relationship: includes 2 entity types.
 - Ternary Relationship: includes 3 entity types.
 - N-ary Relationship: includes N entity types.

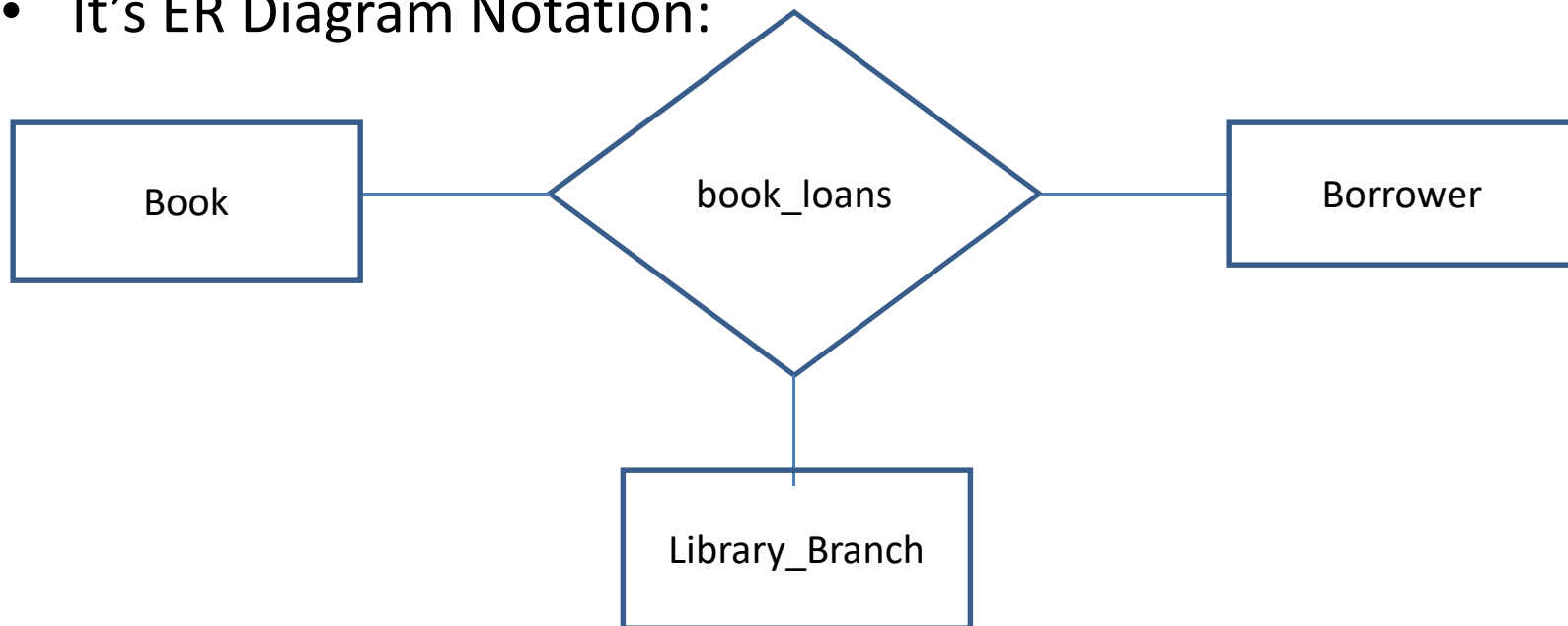
Relationship Degrees

- 1) Binary Relationship: Includes two entity types.
- 2) Example: Employee “works_on” Project.
- 3) Its notation in ER Diagram is as follows.



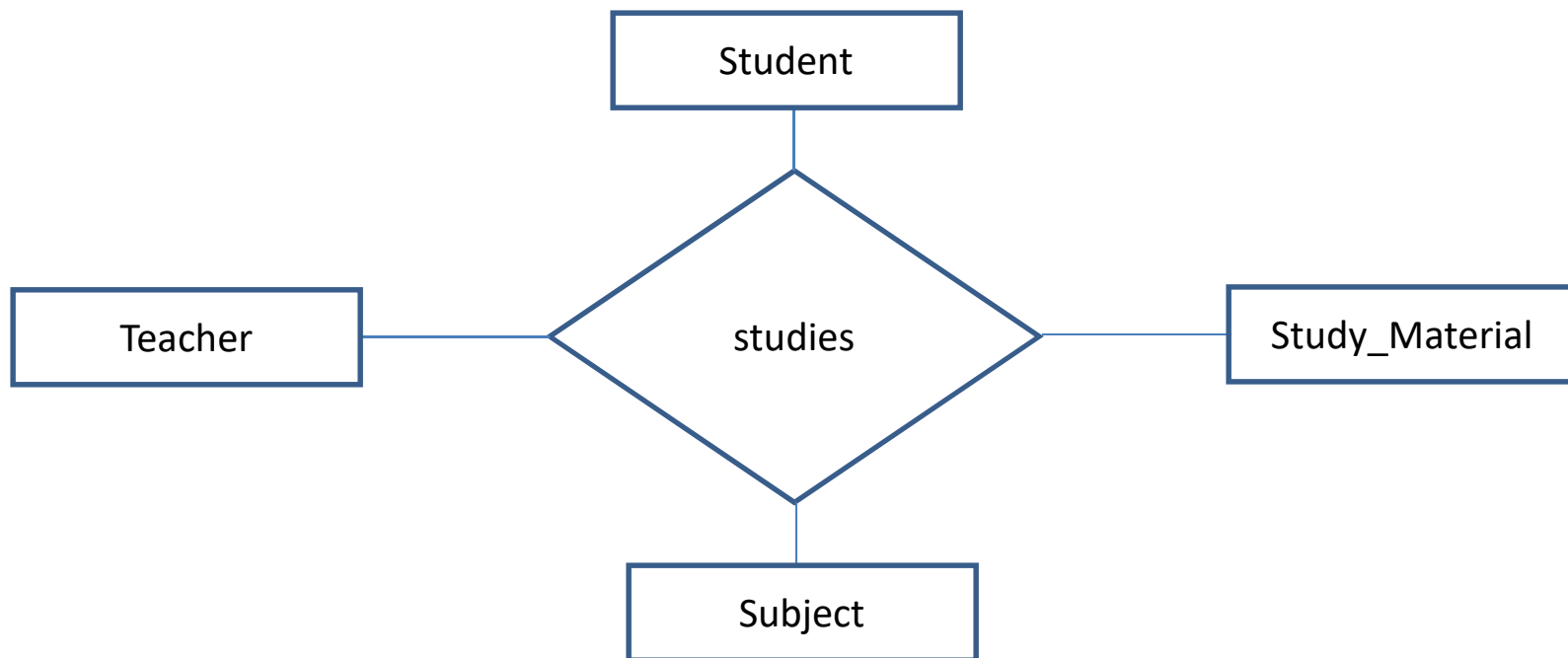
Relationship Degree

- Ternary relationship: includes 3 entity types
- Example: Book_loans: is a relationship that shows:
 - Each borrowed book (**Book**)
 - Who borrowed it (**Borrower**)
 - Which library branch the book was borrowed from (**Library_Branch**)
- It's ER Diagram Notation:



Relationship Degrees

- N-ary Relationship: includes N entity types.
- Example: studies: is a 4-ary relationship that shows that a “student” studies a “subject” with a “teacher” and the help of “study material”.

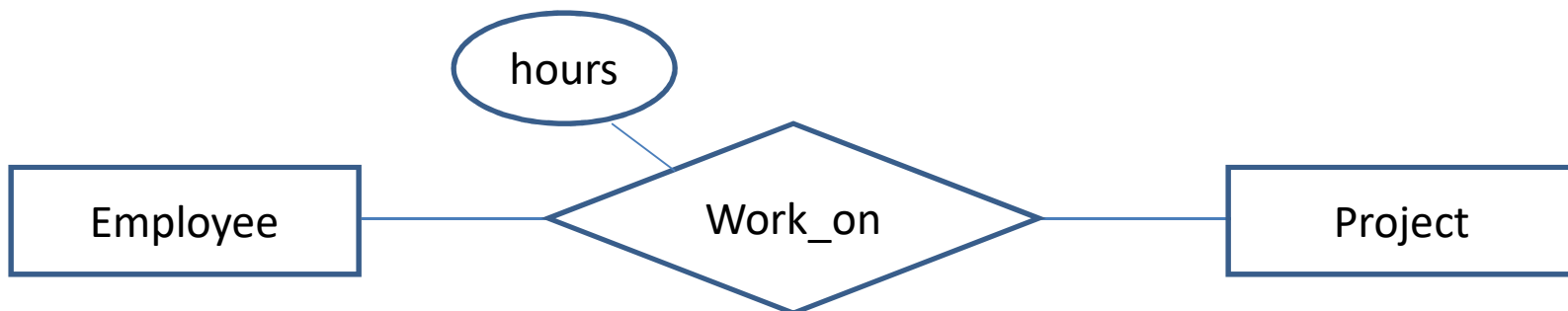


Relationship Attributes

- In some cases, a relationship type can have attributes.
- Usually, in these cases, the attribute does not belong to any of the participating entities (exclusively).
- Because of that, we add the attribute on the relationship.

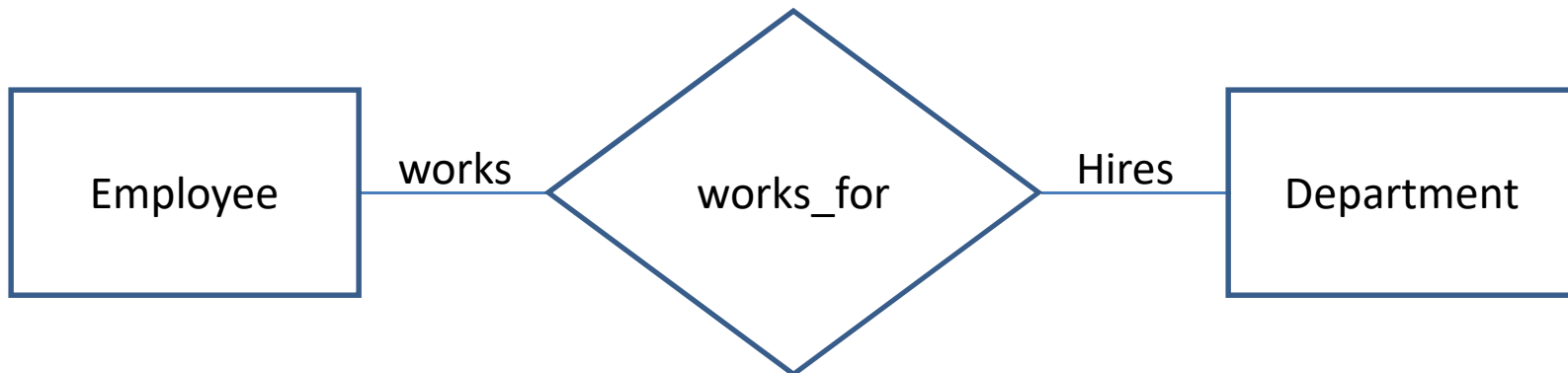
Relationship Attributes

- Examples:
 - start_date: is an attribute that specifies the start date of an employee as a manager of a department. It does not belong to employee or department exclusively. But, it belongs to both of them, therefore we place it on the relationship “manages”.
 - Hours: is an attribute that specifies the number of hours an employee works on a project. It does not belong to employee nor project exclusively. Therefore we place it on relationship “works_on”.



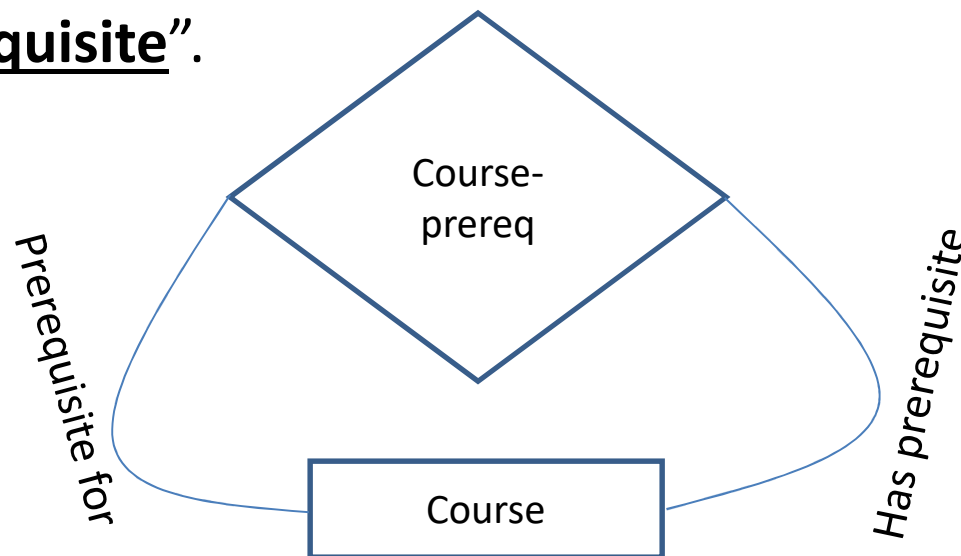
Entity Roles

- In any relationship, entity has a role that specifies what it does in a relationship.
- Example: In Employee “works_for” department relationship:
 - Employee Role: “worker” (works in department)
 - Department Role: “Employer” (employs employee)
- Entity roles can be written on relationship lines in ER Diagram. But they are **implicitly** known, so they are not necessary unless we have a recursive relationship (Look Next Slide).



Recursive Relationship

- Recursive Relationship is a relationship between an entity and itself.
- Example: Course_Prereq is a recursive relationship that shows courses and their prerequisite.
- Notice that entity role is important here because we need to know which course is a “prerequisite for” and which course “has prerequisite”.

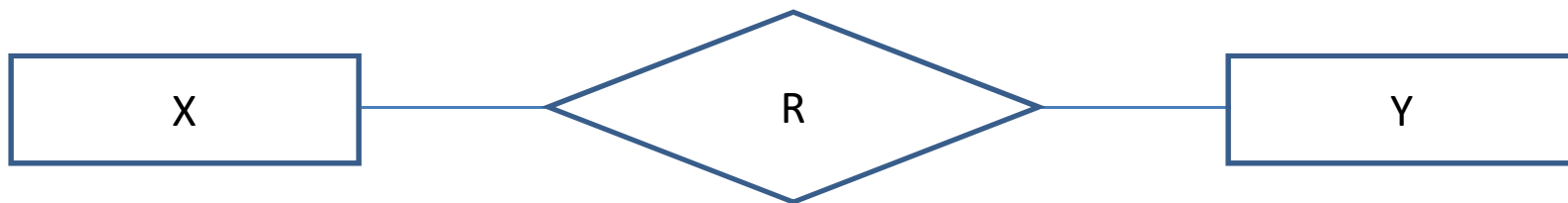


Constraints on Relationships

- Constraints should be reflected in ER diagram.
- They are called **Structural Constraints**.
- For example: Each employee works on “only one project”.
- Types of Structural Constraints:
 - Cardinality Ratio (Maximum Cardinality)
 - Participation (Minimum Cardinality)

Cardinality Ratio (Maximum Cardinality)

- Cardinality Ratio: is the maximum number of “relationship instances” that an entity can participate in. (Maximum Cardinality)
- Types of cardinality ratio :
 - 1:1 --- It is read as (one to one), 1 instance of entity x can be connected to only 1 instance of entity Y via relationship R and vice versa.
 - 1:N --- It is read as (one to many), 1 instance of entity x can be connected to N instances of entity Y via relationship R.
 - M:N --- It is read as (many to many), M instances of entity x can be connected to N instances of entity Y via relationship R and vice versa.



Cardinality Ratio (Maximum Cardinality) (1:1)

Examples:

- One department is managed by only One employee.
- One employee can manage only One department.

Employee ---Manage--- Department

Ahmad ————— CS_Dep

Subresult: 1 to 1

Employee ---Manage--- Department

Ahmad ————— CS_Dep

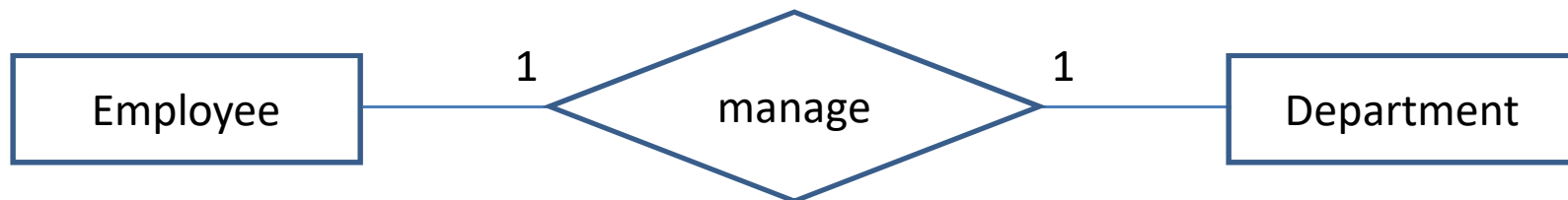
Subresult: 1 to 1

Final Result: (take the max on each column)

1	to	1
1	to	1

1	to	1

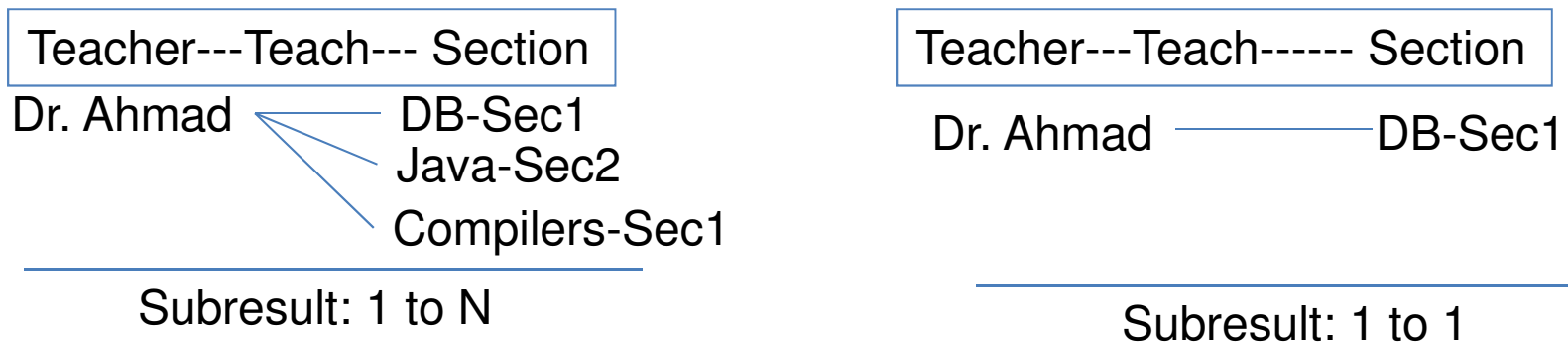
(Final Result)



Cardinality Ratio (Maximum Cardinality) (1:N)

Examples:

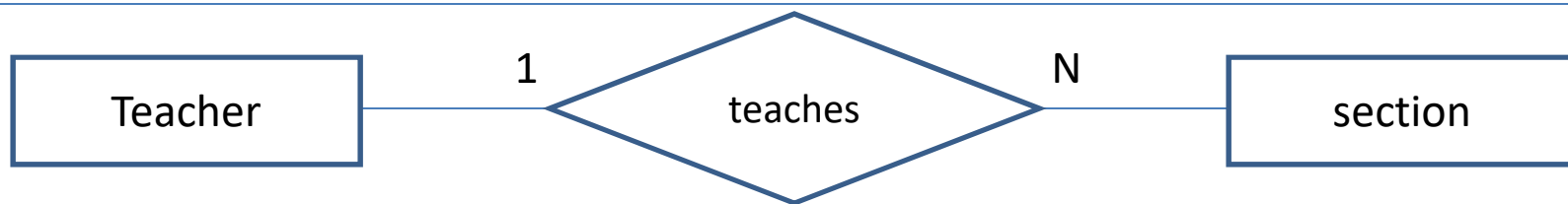
- **One** teacher can teach **Many** sections
- **One** section is only taught by only **One** teacher



Final Result: (take the max on each column)

1 to N
1 to 1

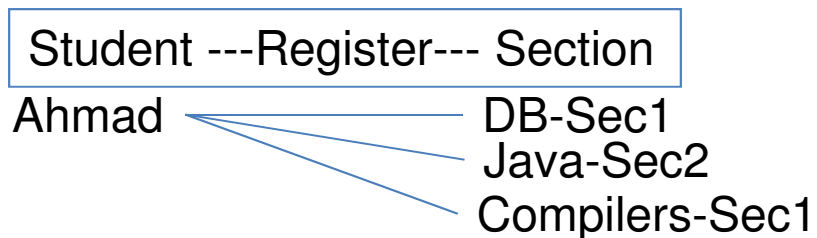
1 to N
(Final Result)



Cardinality Ratio (Maximum Cardinality) (M:N)

Examples:

- **One** student can register for **Many** sections
- **One** section can be registered for by **Many** students



Subresult: 1 to N



Subresult: N to 1

Final Result: (take the max on each column)

1 to N
N to 1

N to M

(Final Result) (switch one N to M to fix ambiguity)



Constraints on Relationships

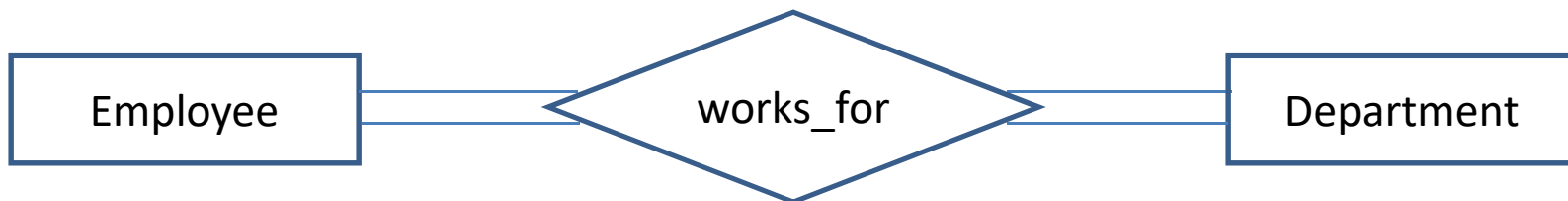
- Types of Structural Constraints:
 - Cardinality Ratio (Maximum Cardinality)
 - We already discussed this one.
 - Participation (Minimum Cardinality)
 - Now, we look into this.

Participation Constraints

- The participation constraint specifies whether the existence of an entity depends on it being related to another entity via the relationship type. This constraint specifies the “minimum” number of relationship instances that each entity can participate in.
- Two types of participation constraints:
 - Total Participation (Existence Dependency)
 - Partial Participation

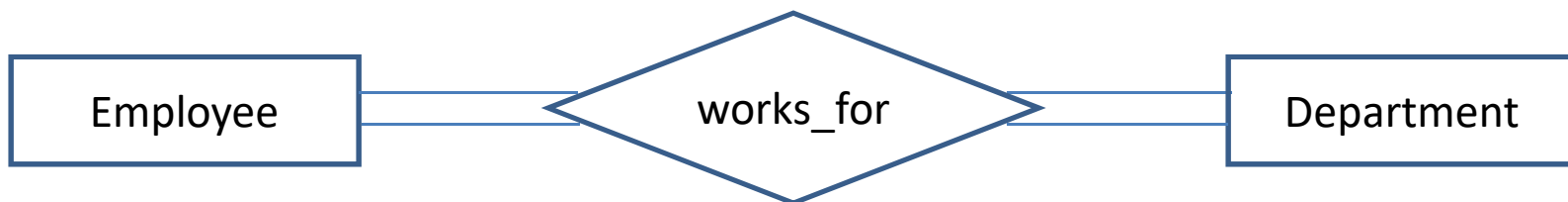
A way to think of participation constraints

- Example 1:
 - Employee “works for” department
 - Total Participation from **Employee side** (how?)
 - Assume that the **company has 3 employees**. Should **all** of the employees belong to at least one department ?
 - If the answer is yes: (Total Participation) (represented by two lines)
 - If the answer is No: (Partial Participation) (represented by one line)
 - In our example, the answer is “yes”



A way to think of participation constraints

- Example 1:
 - Employee “works for” department.
 - Total Participation from department side (how?)
 - Assume that the company has 3 departments. Should all departments have employees working in them ?
 - If the answer is yes: (Total Participation) (represented by two lines)
 - If the answer is No: (Partial Participation) (represented by one line)
 - In our example, the answer is “yes”



A way to think of participation constraints

- Example 2:
 - Employee “manages” department.
 - Partial Participation from **employee side** (how?)
 - Assume that the company has 3 employees. Should **all** employees manage departments ?
 - If the answer is yes: (Total Participation) (represented by two lines)
 - If the answer is No: (Partial Participation) (represented by one line)
 - In our example, the answer is “No” because some employees are not managers.



A way to think of participation constraints

- Example 2:
 - Employee “manages” department
 - Total Participation from **department side** (how?)
 - Assume that **company** has **3 departments**. Should **all** departments be managed by employees?
 - If the answer is yes: (Total Participation) (represented by two lines)
 - If the answer is No: (Partial Participation) (represented by one line)
 - In our example, the answer is “yes”



Weak Entity

- A weak entity: is an entity with a primary key that does not come from its own attributes.
- Strong entity: is an entity that does have a key attribute “from within its own attributes”.
- Weak entity is only identified by being related to another strong entity.
- This kind of relationship is called **identifying relationship**.
- Weak Entities are identified by a combination of:
 - **Partial Key**: Some attributes of weak entity.
 - **Strong Entity Key**: Key of strong entity that defines weak entity.
- **Weak entity key = partial key of weak entity + key of strong entity**

Weak Entity

- Example: Assume that employees can have dependents.
- By dependents we mean “Son”, “Daughter”, “Wife”, ... etc.
- Dependents are only identified through employees they belong to.
- For example: Employee Ahmad has a dependent, his daughter “Sara”. Sara is only identified by being related to Ahmad.
- Dependent can be identified by a combination of:
 - **Partial key**: may be “**First Name**” of dependent, assuming that dependents of the same employee do not have similar first name.
 - **Strong Entity Key**: Key of employee (**employeeID**)
- **Key of dependent is**: dependent name + employee ID

Weak Entity

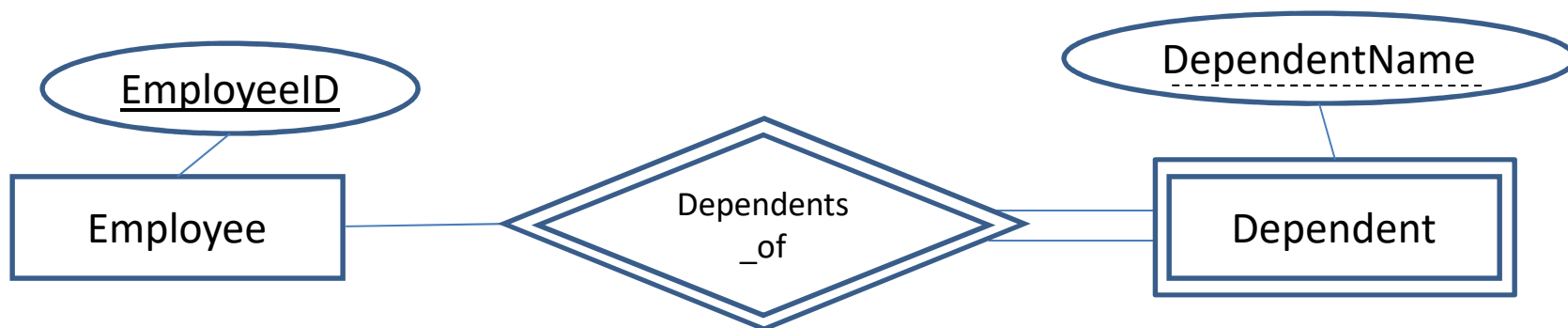
- For example assume employee with ID “**365**” has two dependents, his **daughter “Sara”** and his **son “Kamal”**. Also employee with ID 300 has a son “Kamal”, then dependents are identified as:

<u>EmployeeID</u>	<u>DependentName</u>	Relationship
365	Sara	Daughter
365	Kamal	Son
300	Kamal	Son

- Notice that partial key cannot work as key by itself.
 - in this example, the two employees have a son named “Kamal”. So, “kamal” (dependent name) cannot be a key for dependent. This is why it is called partial key.

Weak Entities

- Weak entities are represented in ER Diagram by a double lines in entity and relationship shapes.
- Also, because weak entities depend on a strong entity in order to exist, then weak entities **always** has “**total participation**” in the **identifying relationship**. (represented by double lines).
- In ER diagram, a **partial key** is underlined with a **dashed** line.



More Examples

(Registration DB)(Entities)

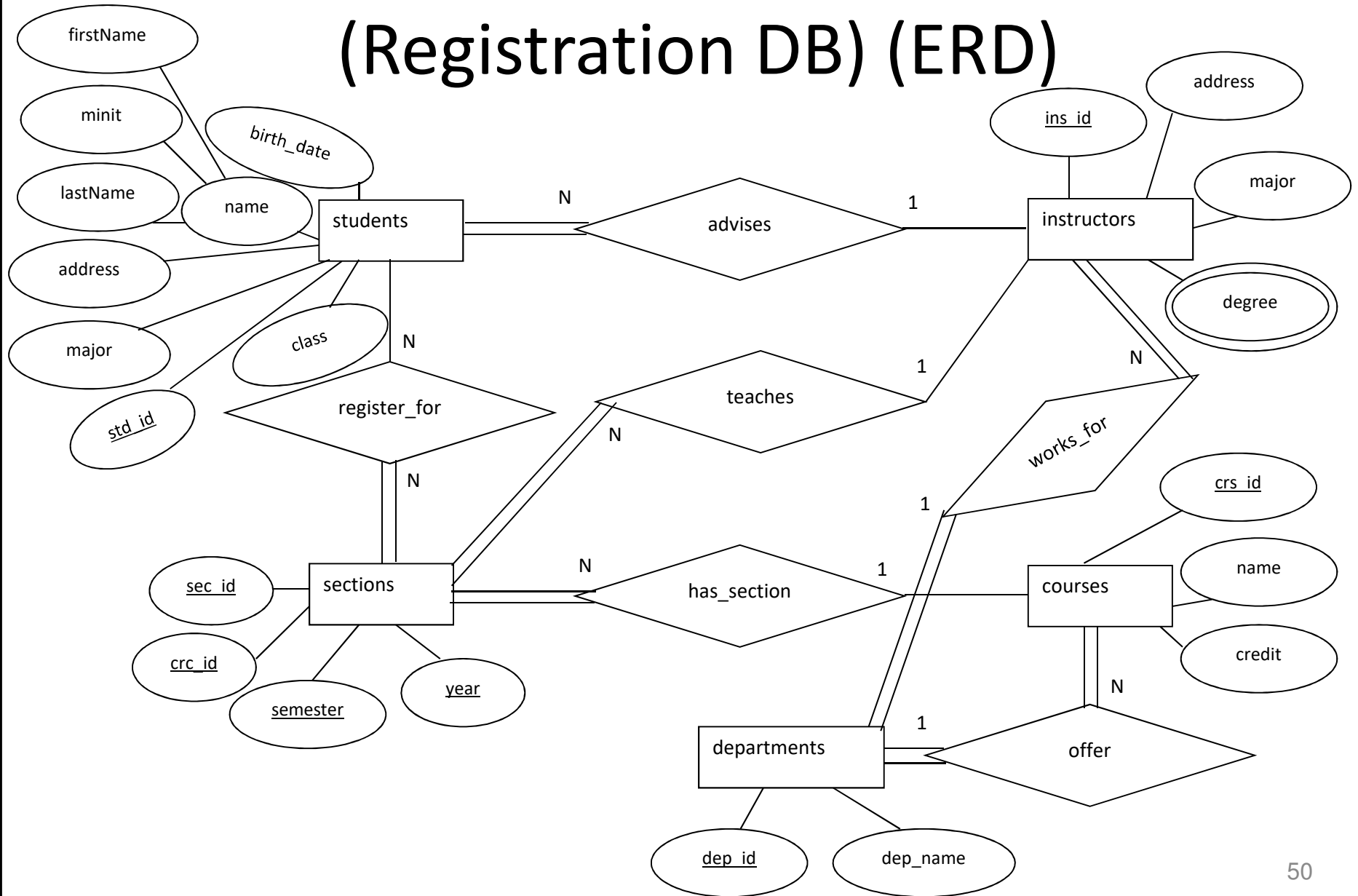
- **Student:** Each student has an id, a name that is composed of a first name, middle initial, and last name. Each student has an address, gender, major, class, and birth date.
- **Course:** Each course has an id, name, and credit hours.
- **Instructor:** Each instructor has an id, a name, address, major, and degree.
- **Department:** Each department has an id, and name.
- **Section:** Each section has an id which is “not” unique among other sections. Each section has a semester and year in which it was offered.

More Examples

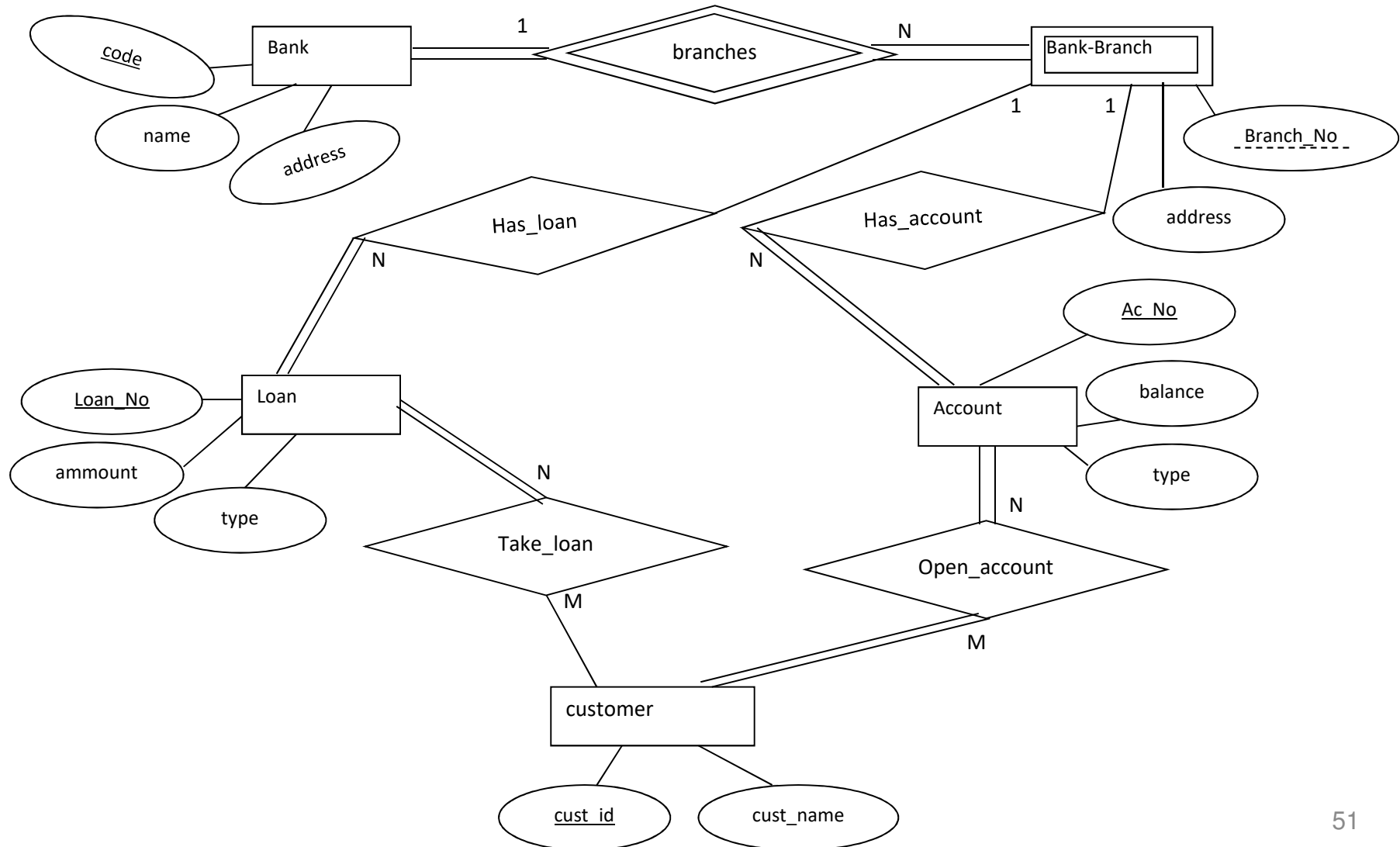
(Registration DB) (Relationship)

- Student “registers-for” Section
- Instructor “teaches” Section
- Course “has” Section
- Course is “offered by” Department
- Student “belongs to” a Department
- Course “belongs to” a Department
- Instructor “belongs to” a Department

More Examples (Registration DB) (ERD)



More Examples (Bank DB) (ERD)



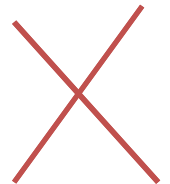
Important Note

- Question: Assume you are designing ERD for one Library. **Do you put entity “Library” in ERD?**

Answer: **No**, remember that the entity is translated into a table in the database. In this case, the table would have only **one record** for the library (**Not Useful**)

Library

LibraryID	LibraryName
1	Abd Alhameed Shoman



In Class Exercises

- Design ERD for one zoo
- Design ERD for one Library
 - Assume each library has branches

Alternative Notation for Structural Constraints

(Min,Max) Notation

- Instead of “cardinality ratio” & “participation constraints”, we can use (min,max) notation.
- For each participating entity type in a relationship, you specify a pair of numbers (min,max).
- (min,max) indicates the minimum and the maximum number of relationship instances an entity can participate in this relationship.

Alternative Notation for Structural Constraints

(Min,Max) Notation

- Example:
 - Employee “manages” department.
 - From the employee side, the notation is (0,1). 0 is the minimum because some employees are **not managers**. 1 is the maximum because 1 given employee can be a manager for **at most 1 department**.
 - From the department side, the notation is (1,1) because each department **must** have a manager, and **at most** it has only **one manager**.



Alternative Notation for Structural Constraints

(Min,Max) Notation

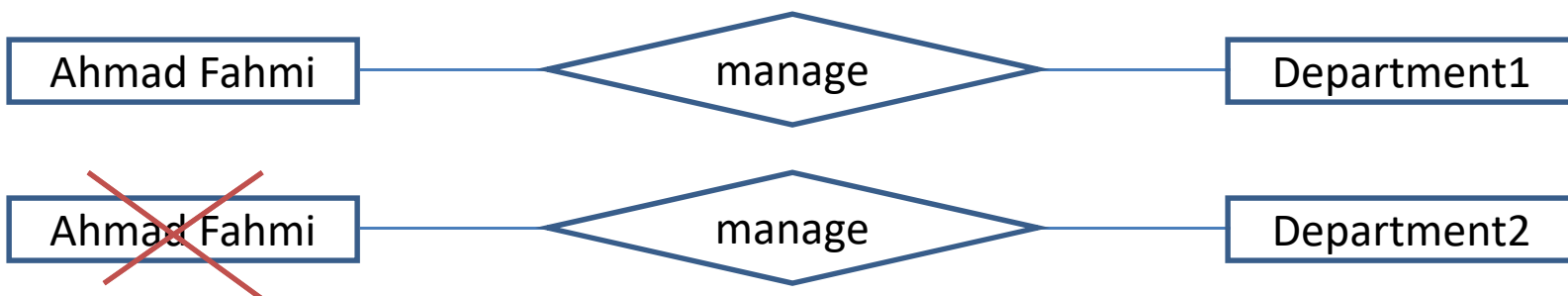
- From “Employee” side:
 - Suppose we have one employee Ahmad Fahmi.
 - What is the minimum number of times Ahmad Fahmi appears in manages relationship?
 - Answer: **zero**, because Ahmad Fahmi might not be a manager.
 - What is the maximum number of times Ahmad Fahmi appears in manages relationship?
 - Answer: **1**, because Ahmad Fahmi can be a manager for only one department.
 - Therefore, the final result from Employee side is (0,1)

Alternative Notation for Structural Constraints (Min,Max) Notation



- **Minimum** number of times for **Ahmad Fahmi** is zero because he might not be a manager

In this slide, we focus only on employee



- **Maximum** number of times for **Ahmad Fahmi** is 1 because he cannot be a manager for more than one department.

Alternative Notation for Structural Constraints

(Min,Max) Notation

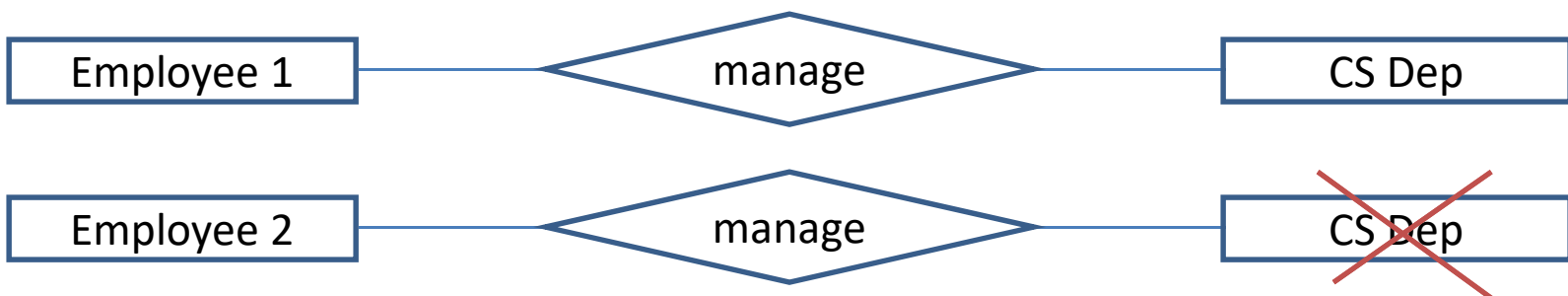
- From “Department” side:
 - Suppose we have one department CS Dep.
 - What is the minimum number of times CS Dep appears in manages relationship?
 - Answer: **1**, because CS Dep must have a manager.
 - What is the maximum number of times CS Dep appears in manages relationship?
 - Answer: **1**, because CS Dep can have only one manager.
 - Therefore, the final result from Department side is (1,1)

Alternative Notation for Structural Constraints (Min,Max) Notation



- **Minimum** number of times for **CS Dep** is 1 because it must have a manager

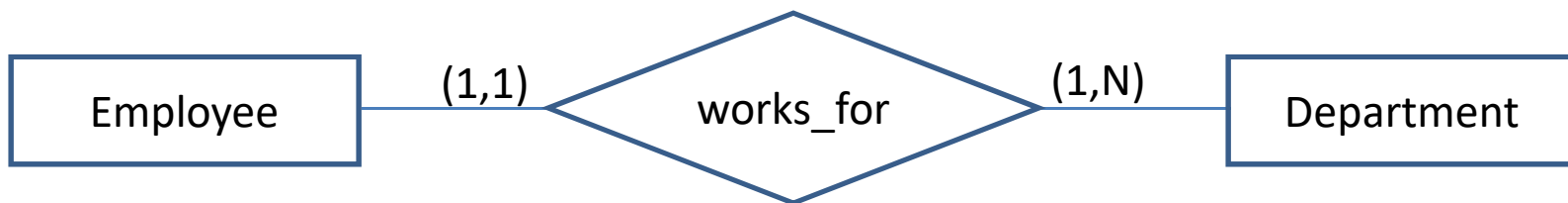
In this slide, we focus only on department



- **Maximum** number of times for **CS Dep** is 1 because it can have at most 1 manager

Alternative Notation for Structural Constraints (Min,Max) Notation

- Example:
 - Employee “works_for” department
 - From the employee side, the notation is (1,1). 1 is the minimum because each employees **must work** in a department. 1 is the maximum because 1 given employee can **work only in 1 department.**
 - From the department side, the notation is (1,n) because each department **must** have **at least** one employee, and **at most** it is can have n employees.

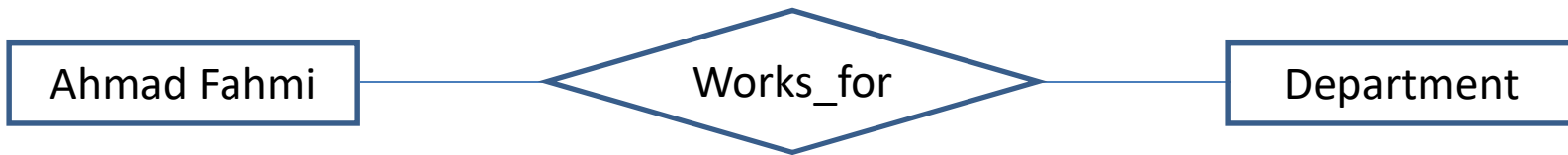


Alternative Notation for Structural Constraints

(Min,Max) Notation

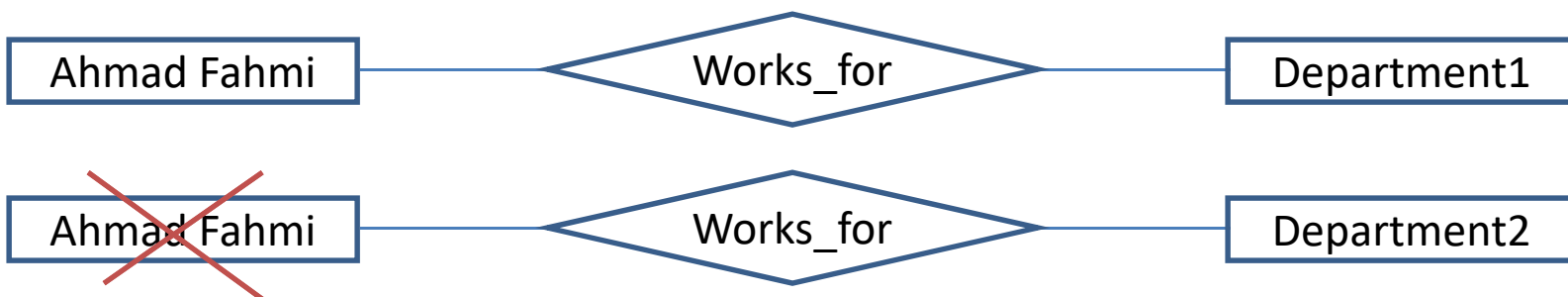
- From “Employee” side:
 - Suppose we have one employee Ahmad Fahmi.
 - What is the minimum number of times Ahmad Fahmi appears in work for relationship?
 - Answer: **1**, because Ahmad Fahmi must work in some department.
 - What is the maximum number of times Ahmad Fahmi appears in works for relationship?
 - Answer: **1**, because Ahmad Fahmi can can work for only one department.
 - Therefore, the final result from Employee side is (1,1)

Alternative Notation for Structural Constraints (Min,Max) Notation



- **Minimum** number of times for **Ahmad Fahmi** is 1 because he must work in at least 1 department

In this slide, we focus only on employee



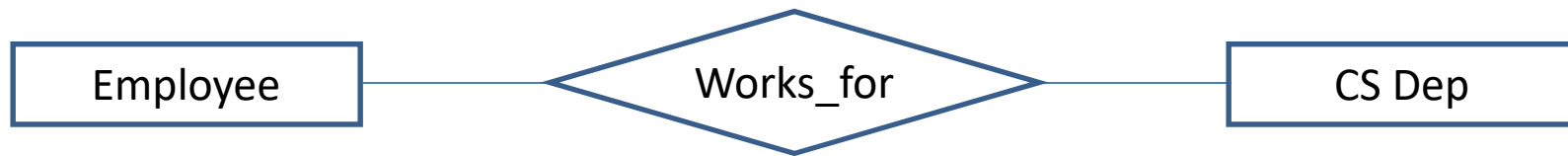
- **Maximum** number of times for **Ahmad Fahmi** is 1 because he can work for at most one department

Alternative Notation for Structural Constraints

(Min,Max) Notation

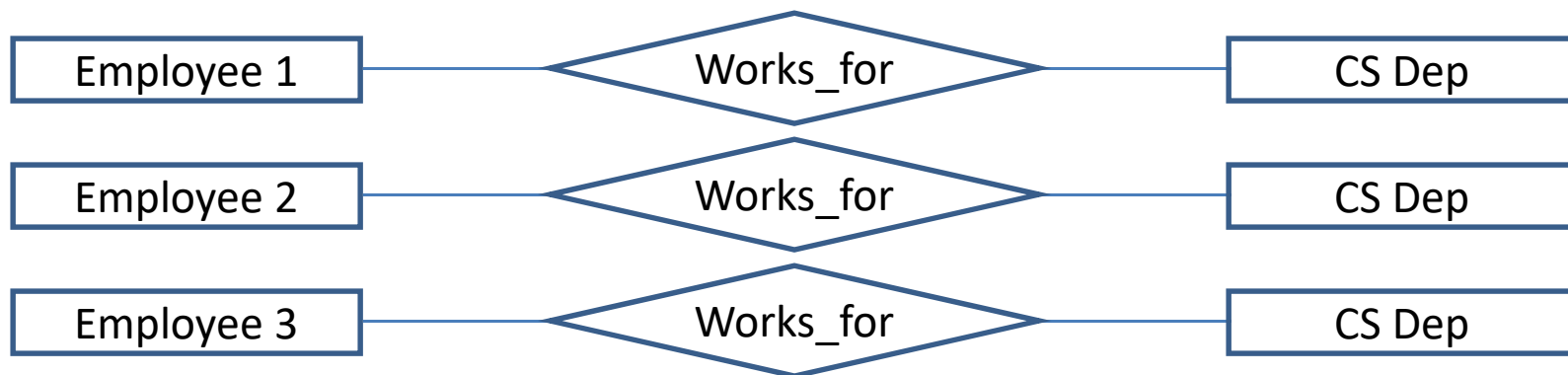
- From “Department” side:
 - Suppose we have one department CS Dep.
 - What is the minimum number of times CS Dep appears in works_for relationship?
 - Answer: **1**, because CS Dep must have at least one employee.
 - What is the maximum number of times CS Dep appears in works_for relationship?
 - Answer: **N**, because CS Dep can have many employees.
 - Therefore, the final result from Department side is (1,N)

Alternative Notation for Structural Constraints (Min,Max) Notation



- **Minimum** number of times for **CS Dep** is 1 because it must have at least one employee.

In this slide, we focus only on department



- **Maximum** number of times for **CS Dep** is "N" because at most "N" employees can work in CS Dep.

Part 3

Enhanced ER Diagram

- Enhanced ER Diagram (EER)
 - Subclasses, superclasses, and inheritance
 - Specialization
 - Generalization
 - Constraints on Specialization and Generalization
 - Specialization and Generalization Hierarchies and Lattices

Prepared By: Dr. Osama Al-Haj Hassan

Reference to: Fundamentals of Database Systems , Ramez Elmasri and Shamkant B. Navathe, Fifth Edition.

What is EER

- It is Enhanced ER Diagram (EER) or
- It is Extended ER Diagram (EER).
- It is an ER diagram + enhancement.
- Enhancements are:
 - Subclasses, superclasses, and inheritance.
 - Specialization and generalization.
 - Shared subclasses (Multiple Inheritance).
- Some times EER diagram is called E2R Diagram.

Subclasses, Superclasses, and Inheritance

- In ER diagram, an entity type represents all entities underneath it.
 - Example, “Employee” represents all entities of employees.
- Sometimes, entities underneath an entity type can be classified to specific entities which have its own characteristics.
 - Example, SalariedEmployee, Hourly Employee, Manager, Secretary, Engineer, Technician.
 - All the previous entity types are considered employees, but each has its own unique attributes.

Subclasses, Superclasses, and Inheritance

- SalariedEmployee, Hourly Employee, Manager, Secretary, Engineer, and Technician are considered subclasses of “Employee”.
- “Employee” is the superclass of all the previous types of employees.
- We have a set of subclass/superclass connections
 - Example: Manager/Employee
 - Example: Secretary/Employee

Subclasses, Superclasses, and Inheritance

- The connection between subclass and superclass is called inheritance.
- A subclass inherits all attributes of its superclass.
 - Example:
 - Employee has a name, address, and birth_date.
 - Each subclass of Employee such as manager, secretary, ... etc has also name, address, and birth_date which they inherit from the superclass Employee.

Subclasses, Superclasses, and Inheritance

- A subclass inherits all relationships of its superclass
 - Example: Employee “works_for” Department
 - All subclasses such as Manager, Secretary, ... etc also “work_for” Department.

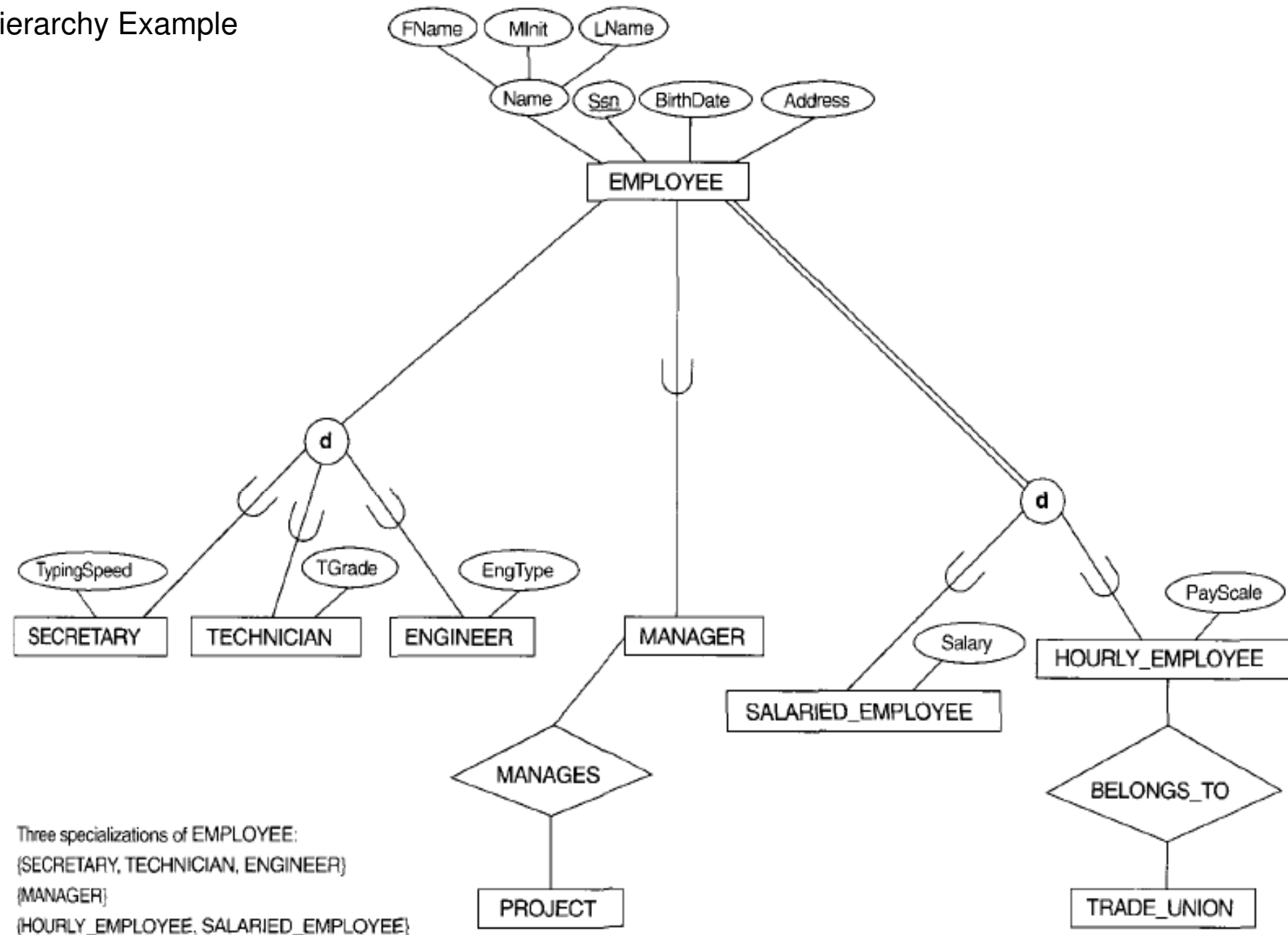
Subclasses, Superclasses, and Inheritance

- We classify an entity type into subclasses because each subclass has its own unique attributes. (Local or Specific Attributes).
- Example:
 - Secretary has “typing_speed” attribute.
 - Salaried_Employee has “salary” attribute.
 - Hourly Employee has “payScale” attribute.
 - Engineer has “eng_type” attribute.
 - Technician has “Tech_grade” attribute.

Subclasses, Superclasses, and Inheritance

- Subclasses can also have their unique relationships with other entities. (Specific Relationships).
- Example 1:
 - In company ER Diagram we had:
 - Employee “manages” Department
 - But, this relationship does not include all employees. (Only Managers)
 - Now, in EER, instead of the above relationship. We have:
 - Manager “manages” Department
- Example 2:
 - HourlyEmployee “belongs_to” Trade_Union
 - This relationship applies only to HourlyEmployee subclass

Hierarchy Example




Specialization

- The process of defining subclasses of an entity type is called **specialization**.
- This entity type is called “superclass” of the specialization.
- Specialization distinguishes between subclasses based on a certain method:
 - Example: Salaried Employee and Hourly Employee are grouped together because they are **classified based on paying method**.

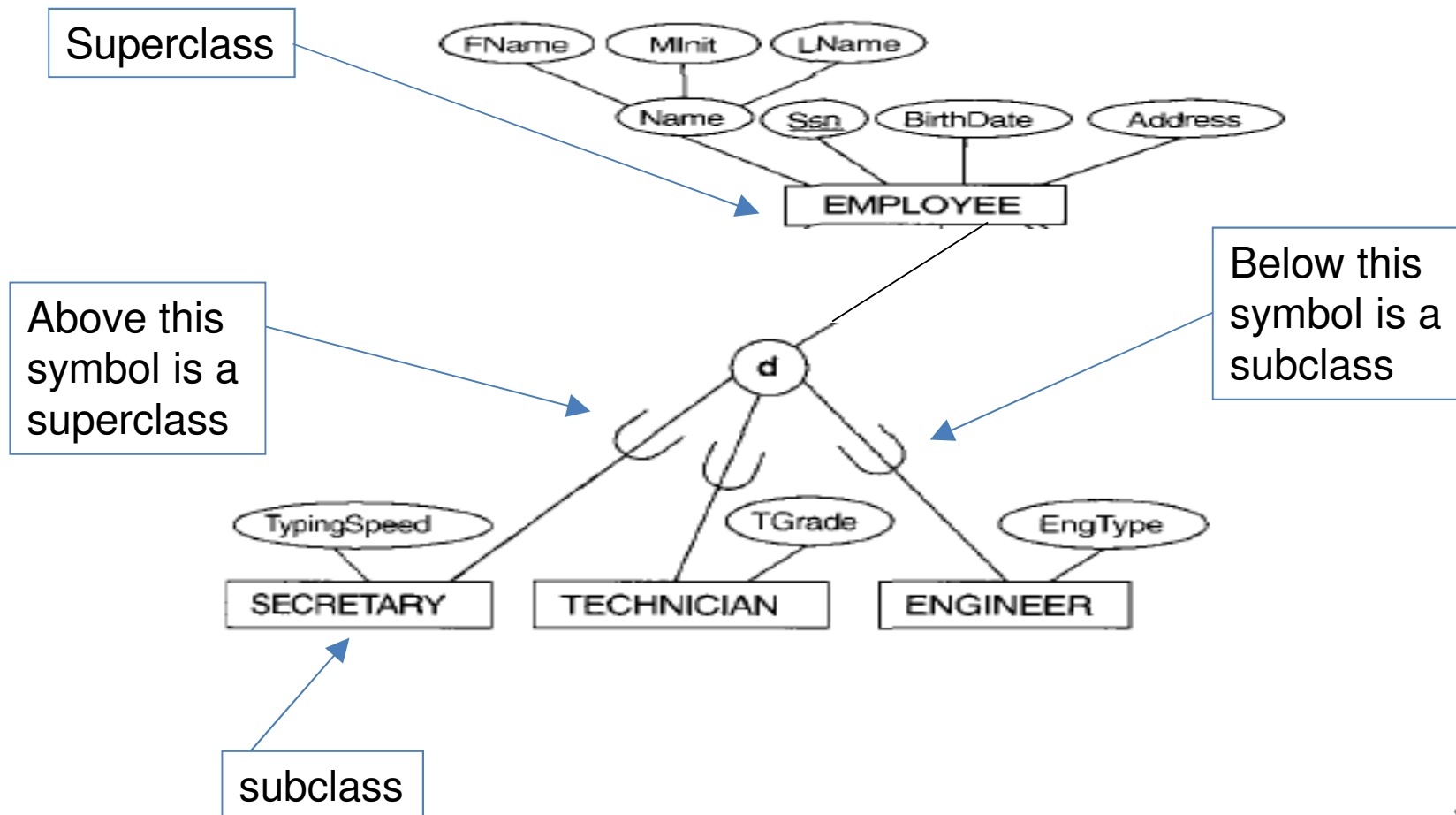
Specialization

- The Employee entity type has 3 specializations:
 - 1) {SalariedEmployee, HourlyEmployee}
 - Classified based on **paying method**.
 - 2) {Manager}
 - Has its own **unique role** (managing).
 - 3) {Secretary, Technician, Engineer}
 - Classified based on **job type**.

Specialization

- How specialization is represented in EER:
 - Subclasses that define a specialization are attached by lines to a circle that represents the specialization, which is connected to the superclass.
- The *subset symbol* “U” on each line connecting a subclass to the circle indicates the direction of the superclass/subclass relationship.
- If the specialization contains only one subclass, we do not use the symbol  which is used for grouping subclasses.

Specialization



Specialization

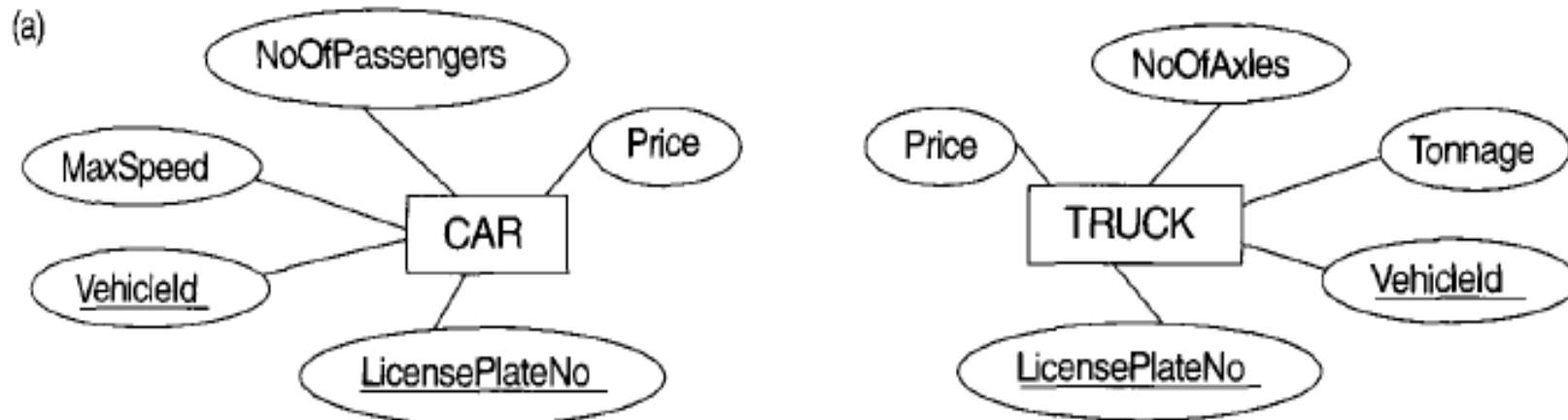
- In summary, specialization allows us to:
 - Define subclasses of entity types.
 - Define specific attributes for subclasses.
 - Define specific relationships between subclasses and other entities or subclasses.

Generalization

- Generalization is the reverse process of specialization.
- In generalization, you generalize a set of entity types into one superclass entity type. Therefore, the generalized entity types are considered subclasses.
- If you find a set of classes with many common attributes, they can be considered subclasses and generalized to a common entity (superclass).

Generalization

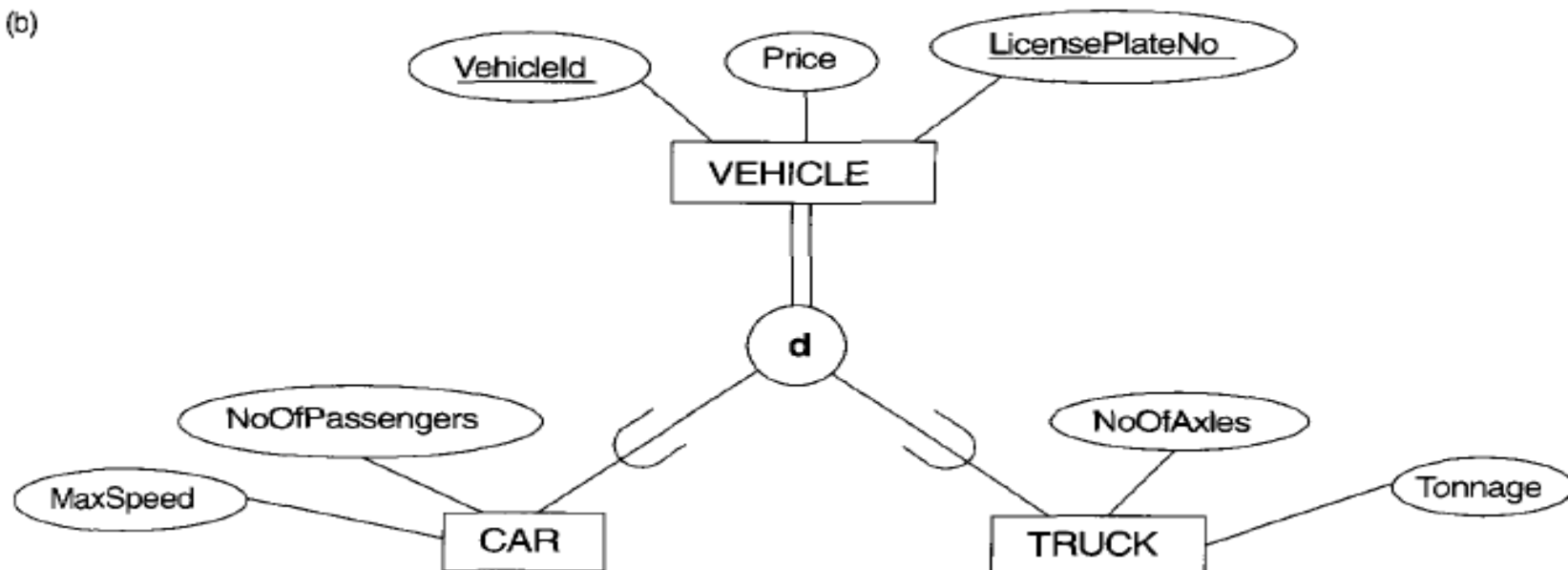
- Example, in an ER diagram you might have two entity types:
 - Car
 - Truck
- How can we generalize this?



Generalization

- The common attributes go to the superclass and the current entity types (Car and Truck) become subclasses.

(b)



Constraints on Specialization and Generalization

- Same constraints apply to specialization and generalization.
- From now on, we will discuss constraints on specialization.
But, keep in mind that they also apply to generalization.

Constraints on Specialization and Generalization

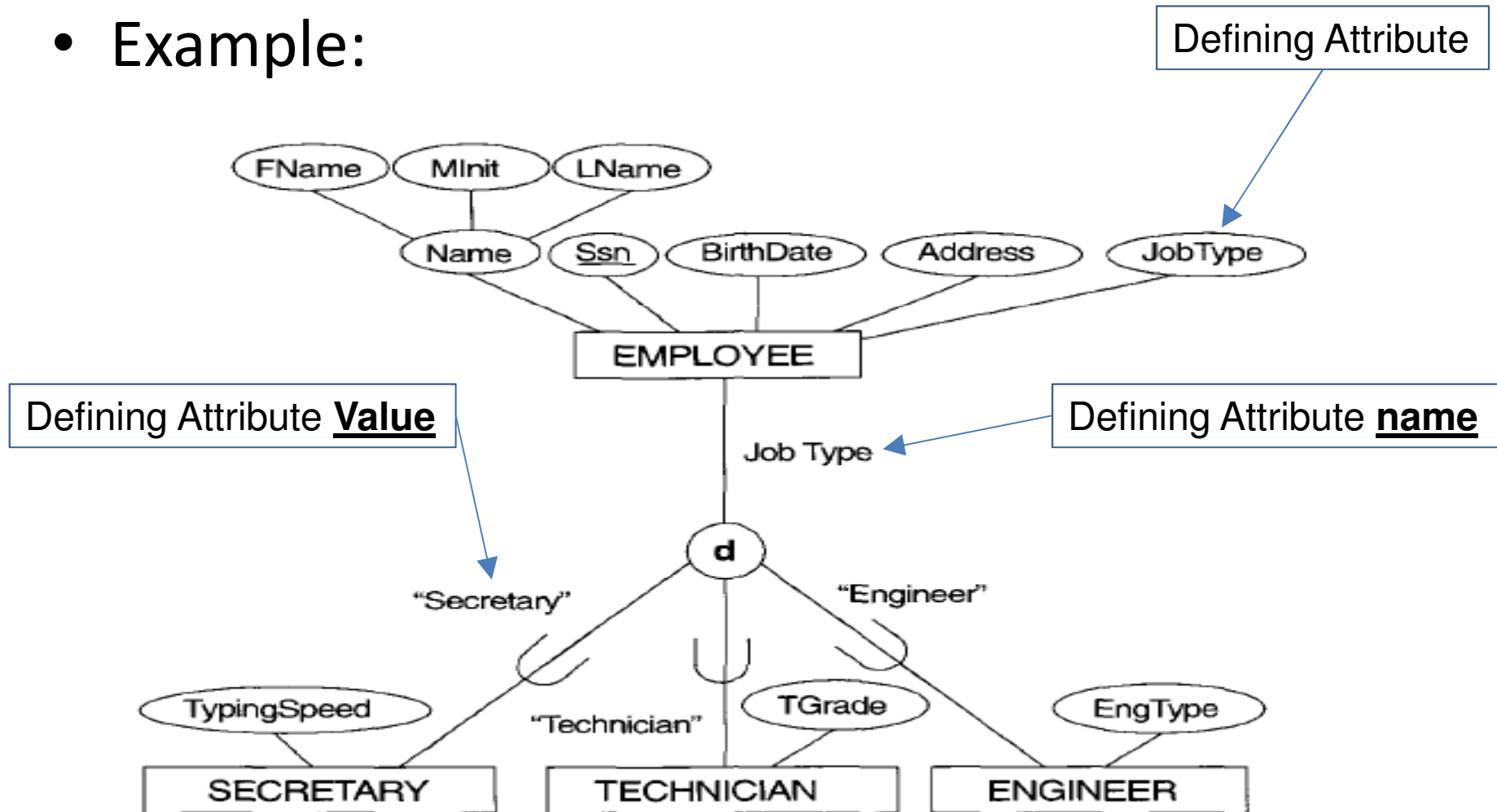
- Based on Definition:
 - 1) Predicate Defined Subclasses
 - 2) User Defined Subclasses
- Based on membership
 - 1) Disjoint Subclasses
 - 2) Overlapping Subclasses
- Based on completeness of participation
 - 1) Total Specialization
 - 2) Partial Specialization

Based on Definition (Predicate Defined Subclasses)

- It is also called Condition Defined Subclasses.
- We can determine to which subclass an entity belongs by placing a condition on some attribute in the superclass.
- We call this attribute “Defining Attribute”.
- Example:
 - JobType is an attribute of superclass “Employee”, based on the value of this attribute, we can determine if an employee belongs to “Secretary”, “Technician”, or “Engineer”.

Based on Definition (Predicate Defined Subclasses)

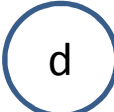
- Example:



Based on Definition (User Defined Subclasses)

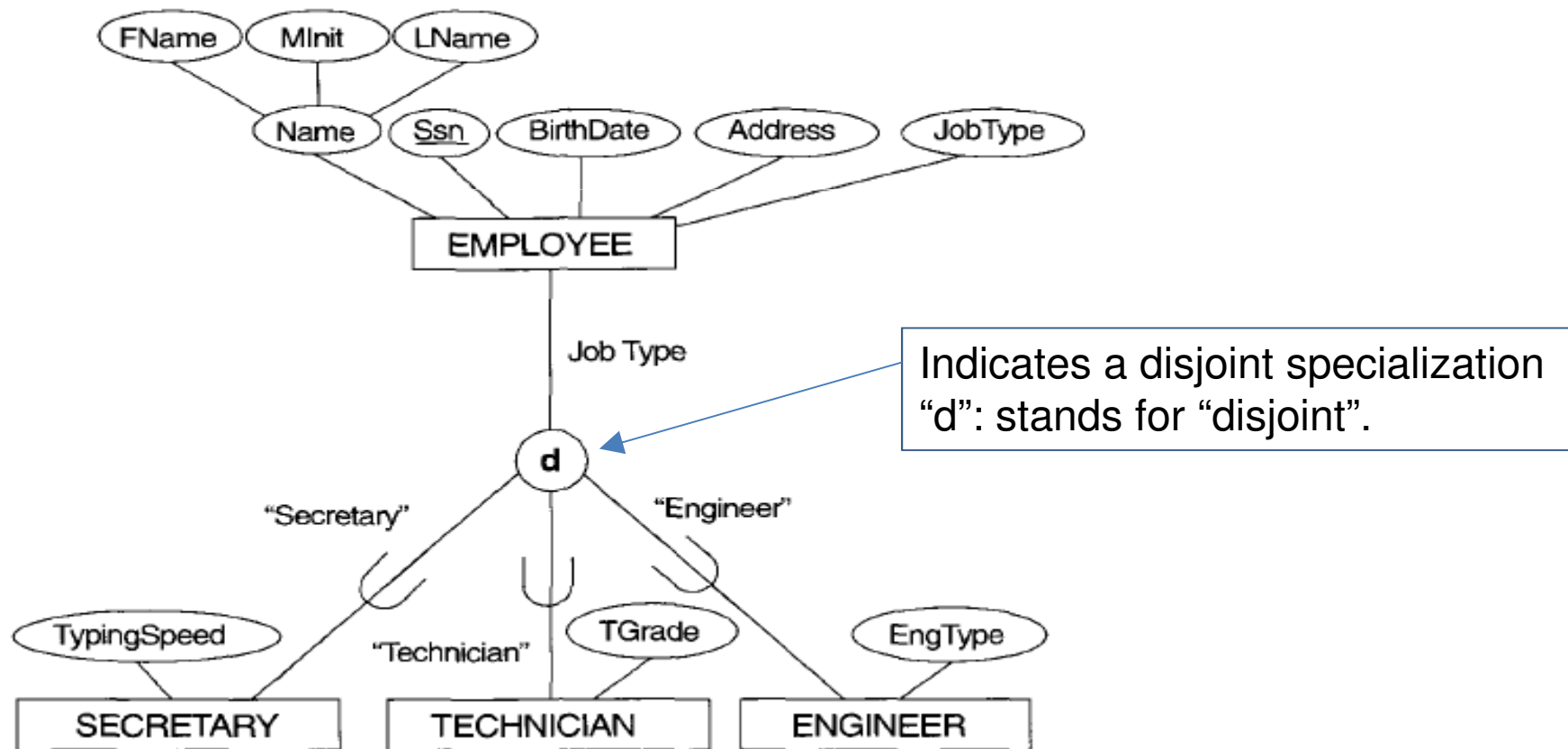
- If the membership in a subclass is not defined based on an attribute, then this subclass is called “User Defined Classes”.
- If a DB user has an entity “A” to add to the database, then he can use his understanding of the mini-world to determine to which subclass entity “A” should be added.

Based on Membership (Disjointness Constraint)

- Specifies that all subclasses of a specialization must be disjoint.
- In other words, if an entity belongs to one subclass of a specialization, then it **cannot belong to another subclass** of the **same** specialization.
- Example:
 - {Secretary,Technicien,Engineer} is a disjoint specialization.
 - If an employee is a secretary, then she cannot be a technician
- The disjointness is represented in EER using  symbol.

Disjointness Constraint

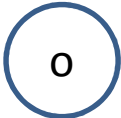
- Example



Disjointness Constraint

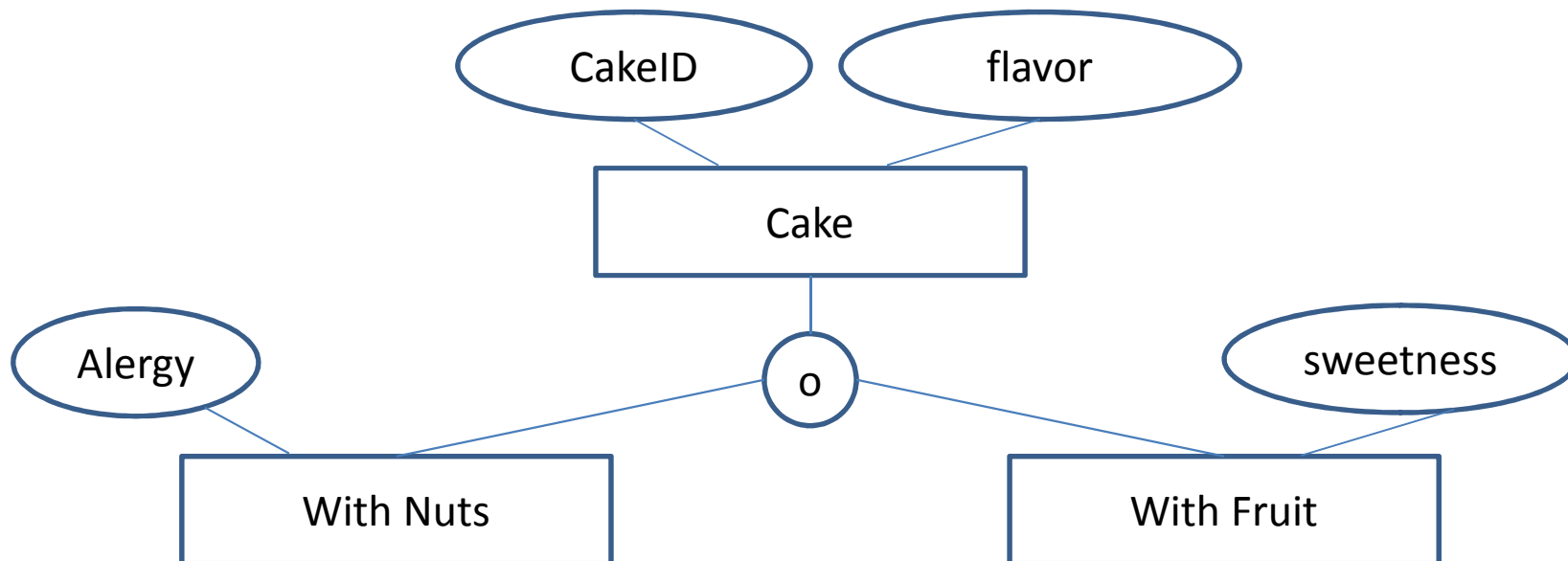
- Question: If a specialization is “attribute-defined”, does this mean that it is a “disjoint” specialization?
- Answer: If the defining attribute is:
 - Single-Valued: then, the answer is “yes”.
 - Multi-valued: then, the answer is “No”.

Based on Membership (Overlapping Subclasses)

- Sometimes an entity can belong to more than one subclass of the same specialization, this means that subclasses overlap with each other.
- In EER, this is represented by using  notation.

Based on Membership (Overlapping Subclasses)

- In a bakery, cake can be made with nuts or with fruits. However, a cake can also be made with nuts and fruits at the same time



Based on Participation (Completeness Constraint)

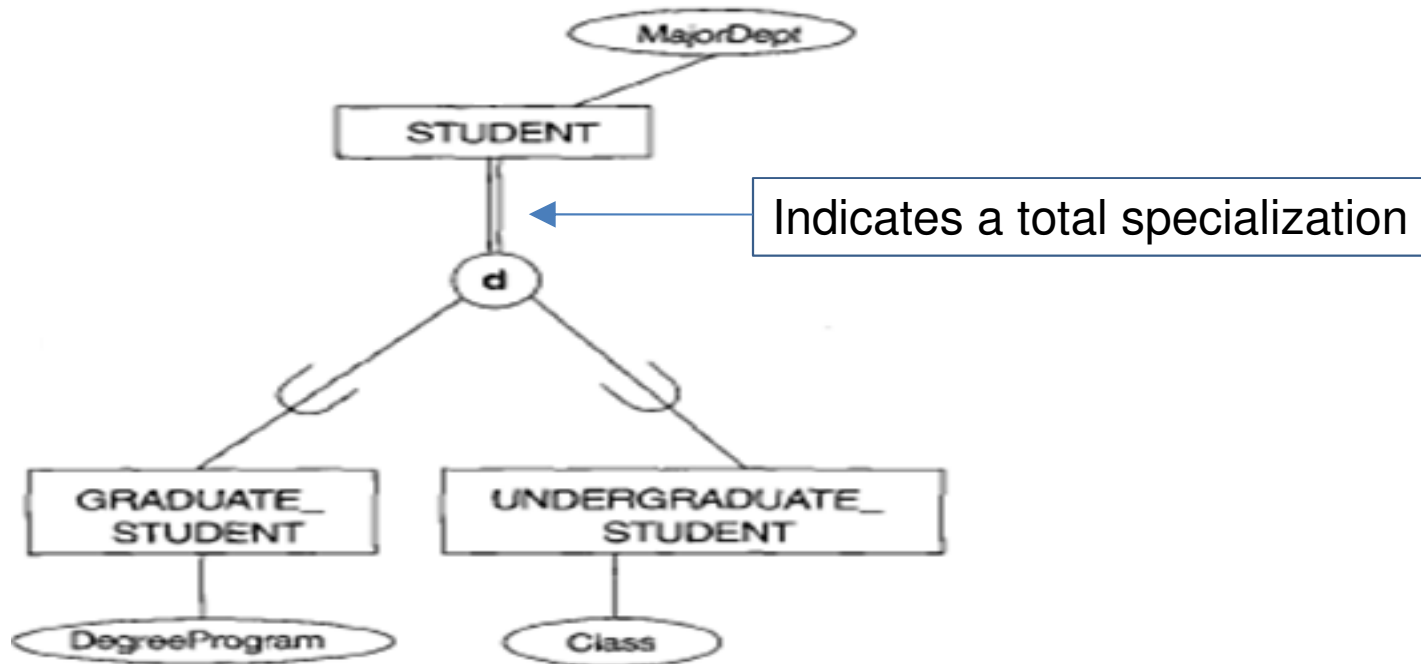
- Completeness constraint states that a specialization can be:
 - Total Specialization or
 - Partial Specialization

Based on Participation (Completeness Constraint)

- Total Specialization:
 - Every entity that belongs to a superclass must belong to at least one subclass of the specialization
- Example: If “every” employee must be either an HourlyEmployee or a SalaryEmployee then the specialization {HourlyEmployee,SalaryEmployee} is a total specialization of Employee.
- In EER, a total specialization is represented by using a double lines that is going out of the superclass.

Based on Participation (Completeness Constraint)

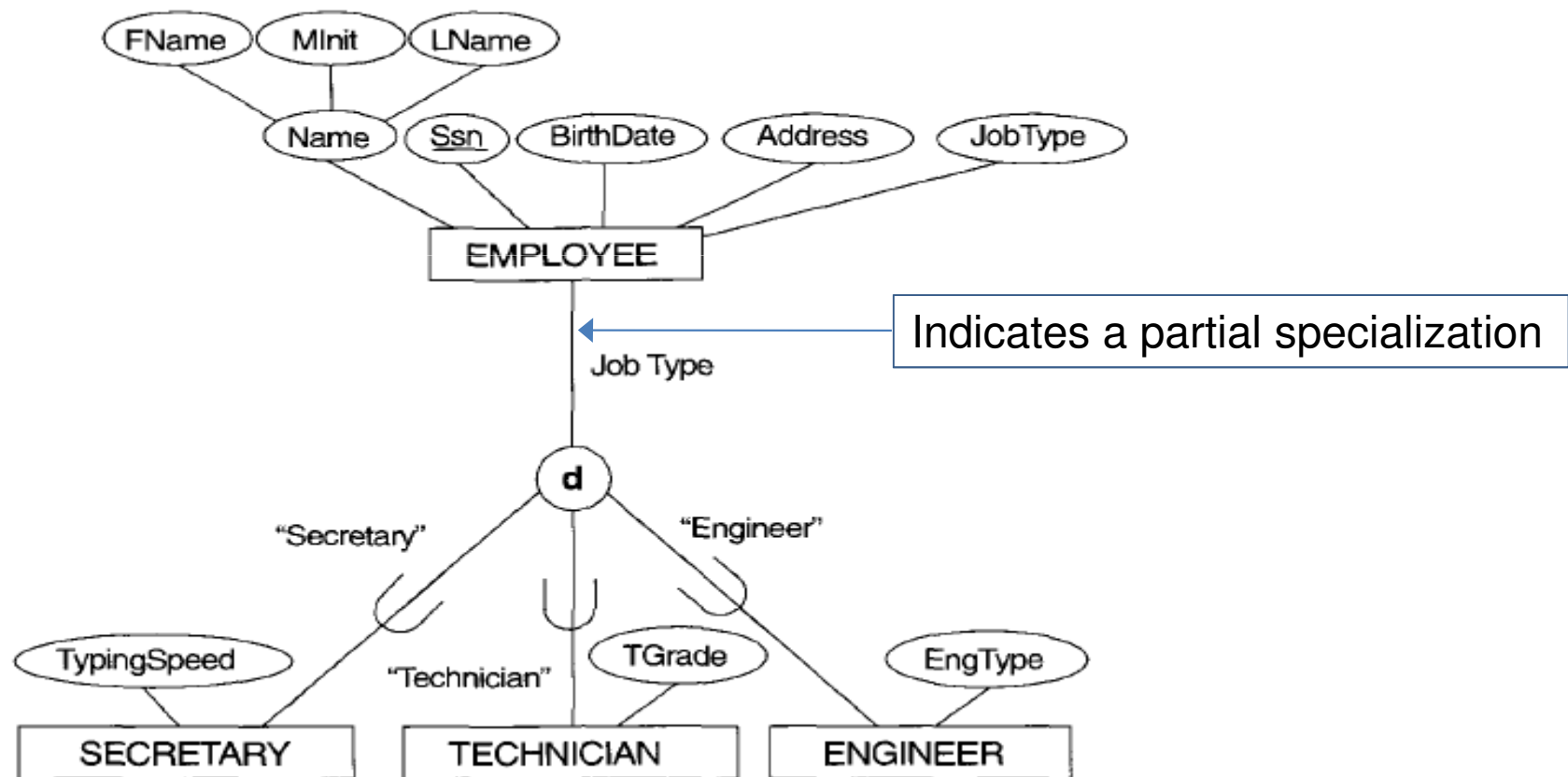
- Example: A student is specialized into:
 - Graduate Student: Master or PhD student
 - Undergraduate Student: B.Sc. Student
- It is clear that there is no other type of student. So, all students will have a place in the subclasses.



Based on Participation (Completeness Constraint)

- Partial Specialization:
 - Some entities might not belong to any of the subclasses of the specialization
 - In EER, it is represented as one line going out of the superclass.
- Example:
 - If an employee is not a secretary, technician, or engineer, then he does not belong to any of these subclasses of this specialization.
 - So, the specialization {secretary, technician, engineer} is a partial specialization.

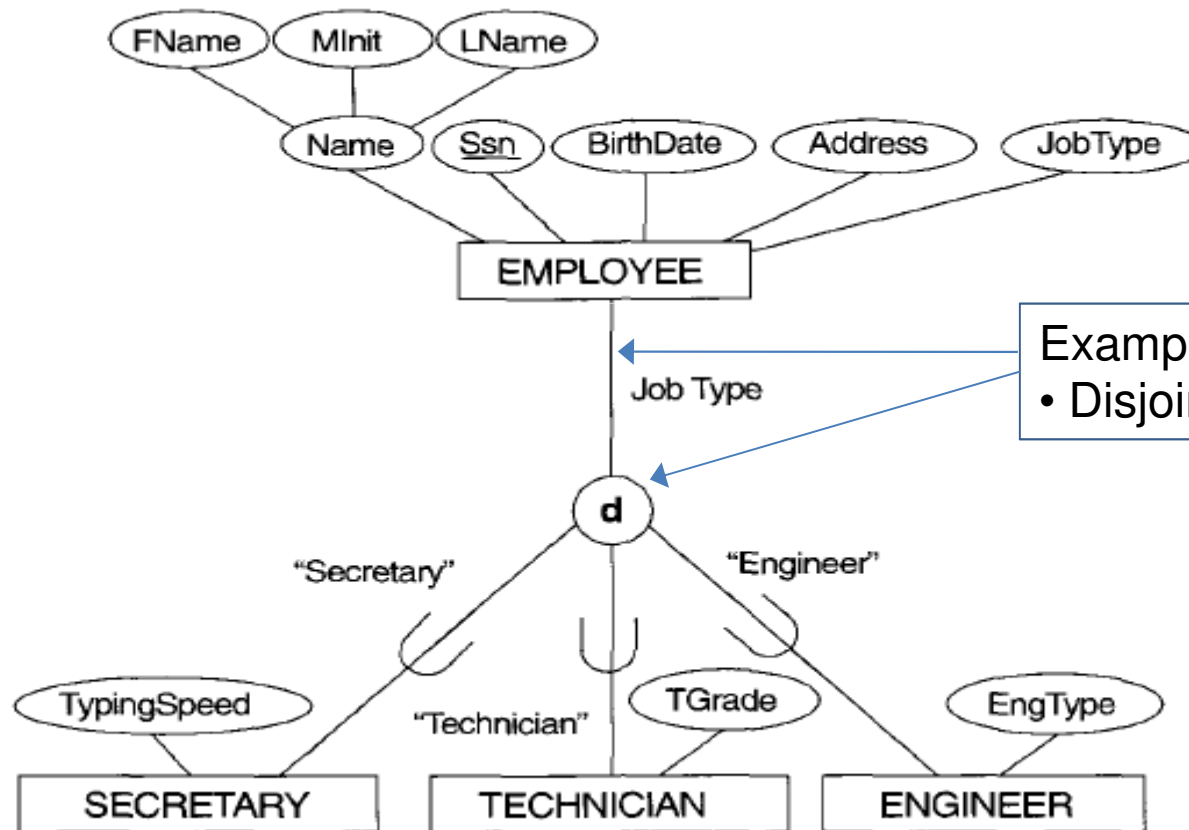
Based on Participation (Completeness Constraint)



Note

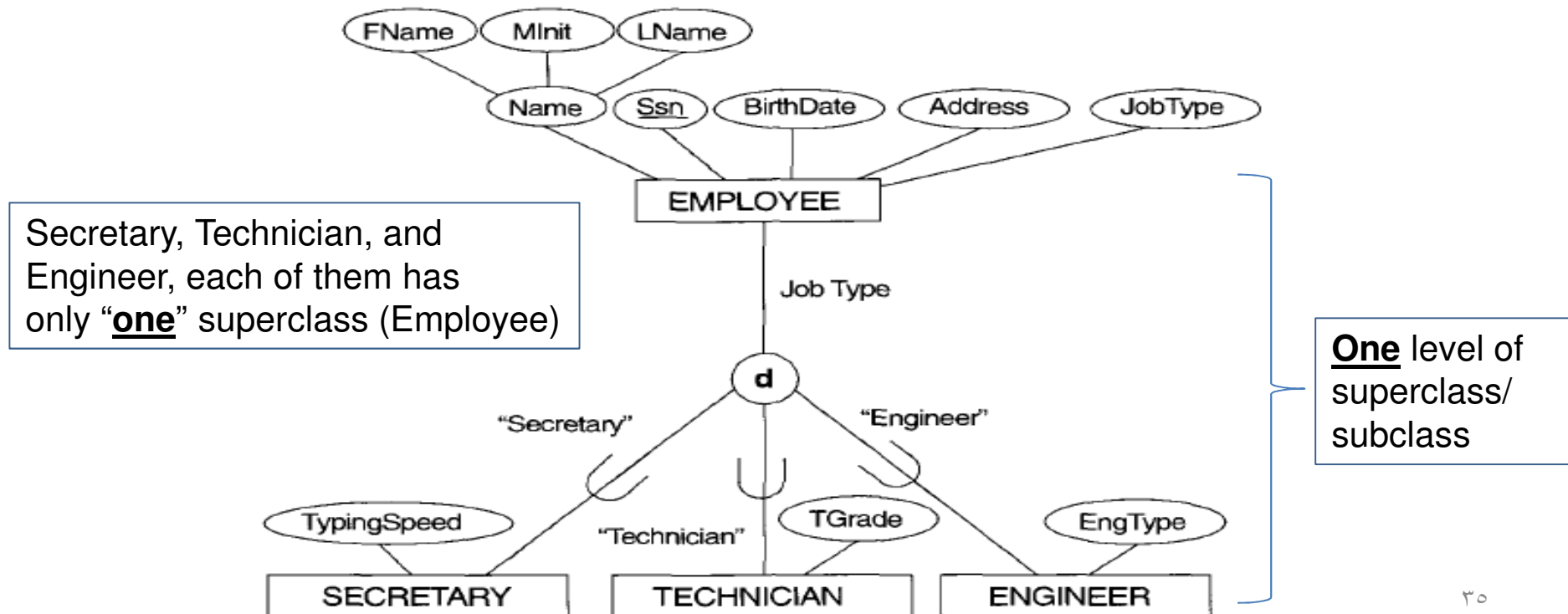
- Disjointness and Completeness constraints are independent, so you might have the following combinations of specializations:
 - Disjoint, total
 - Disjoint, partial
 - Overlapping, total
 - Overlapping, partial

Example



Specialization and Generalization Hierarchies and Lattices

- In the specialization hierarchy that you have seen so far:
 - There is one superclass for each subclass (One superclass/subclass connection).
 - There is only one level of superclass/subclass connections.



Specialization and Generalization

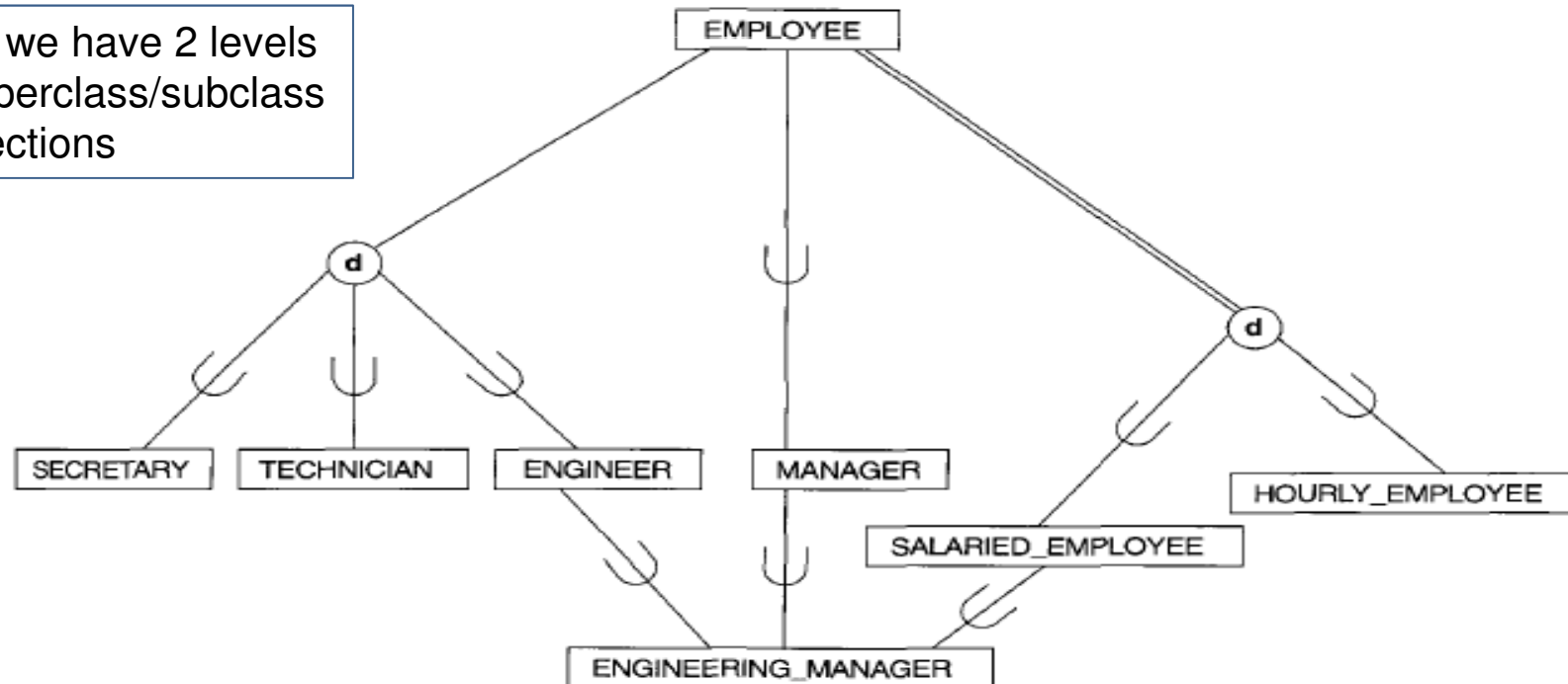
Hierarchies and Lattices

- If we have a subclass that has multiple parents (superclasses), then the hierarchy is called a “specialization lattice”.
 - Multiple superclass/subclass connections for the same subclass.
- Having more than one superclass for the same subclass is also called “**Multiple Inheritance**”.

Specialization and Generalization Hierarchies and Lattices

- Example:
 - Engineer_Manager is an engineer, a manager, and a salaried employee at the same time.
 - Engineer_Manager has 3 superclasses(Manager,Engineer,Salaried_Employee).

Here, we have 2 levels
Of superclass/subclass
connections

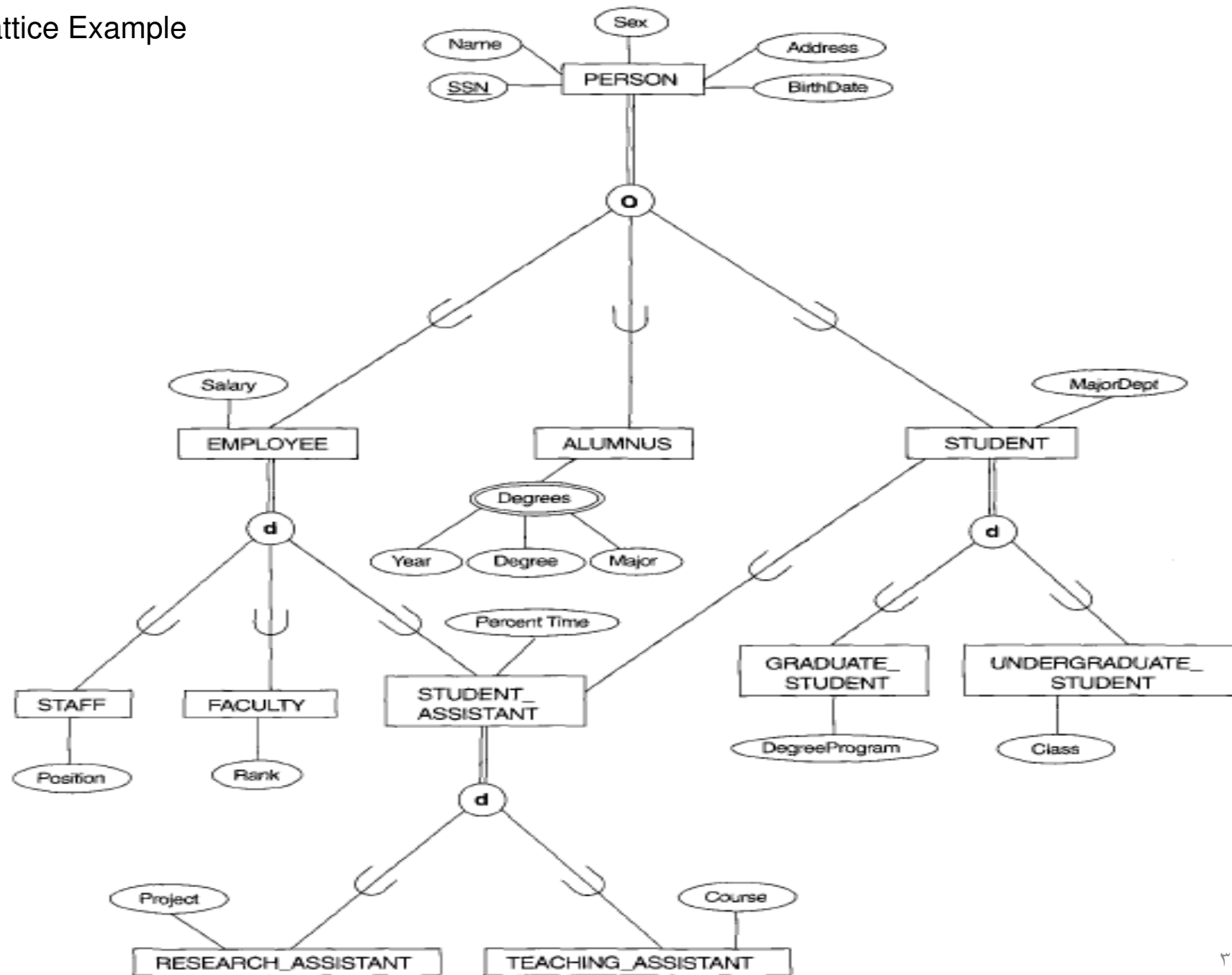


Specialization and Generalization

Hierarchies and Lattices

- Shared Subclass: A subclass with more than one superclass.
- Leaf Node: A class that has no subclasses of its own.
- In the previous example:
 - Engineer is one of the “direct” superclasses of subclass Engineer_Manager.
 - Employee is an “indirect” superclass of subclass Engineer_Manager.

Lattice Example



Specialization and Generalization

Hierarchies and Lattices

- Person entity type is specialized into the subclasses {Employee, Alumnus, Student}.
- Question: Is this specialization overlapping?
- Answer: Yes, because an employee in a university can be an **Alumni** of the same university and at the same time **studying** a different degree while he is **working** in the university.

Part 4

Mapping

- Relational Database Design
 - Foreign Keys
 - Mapping ER Diagram to Relations
 - Mapping EER Diagram to Relations

Foreign Key

- Assume you have 2 entity types, E1 and E2.
- Assume the primary key of E1 is A3.
- If you create an attribute (call it A3') and put it in E2, such that the value of A3' is in the values of A3, then A3' is called a foreign key.

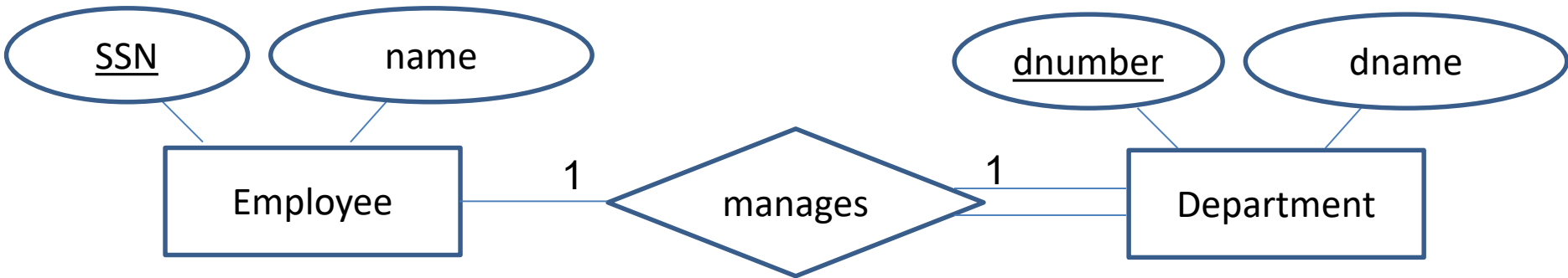
E1 Entity Type
A1, A2, A3, A4

E2 Entity Type
B1, B2, B3, B4, B5, A3'

Primary Key of E1

One Foreign Key in E2

Example 1 (1:1 Relationship)



- Why is it done this way? We will explain later.

Employee

<u>SSN</u>	name
1	Ahmad
2	Kamal
3	Sara

Primary Key

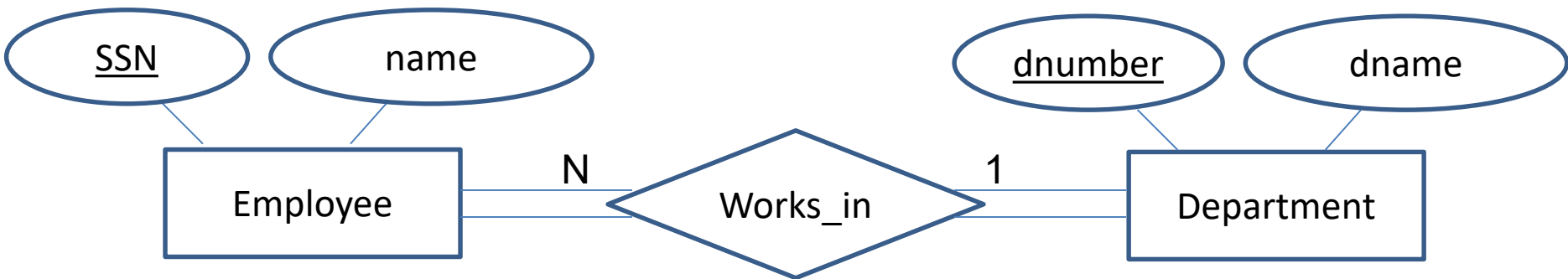
Department

<u>dnumber</u>	dname	mgrSSN
1	CS	3
2	QA	2

Primary Key

Foreign Key

Example 2 (1:N Relationship)



- Why is it done this way? We will explain later.

Employee

<u>SSN</u>	name	dno
1	Ahmad	2
2	Kamal	1
3	Sara	1

Primary Key

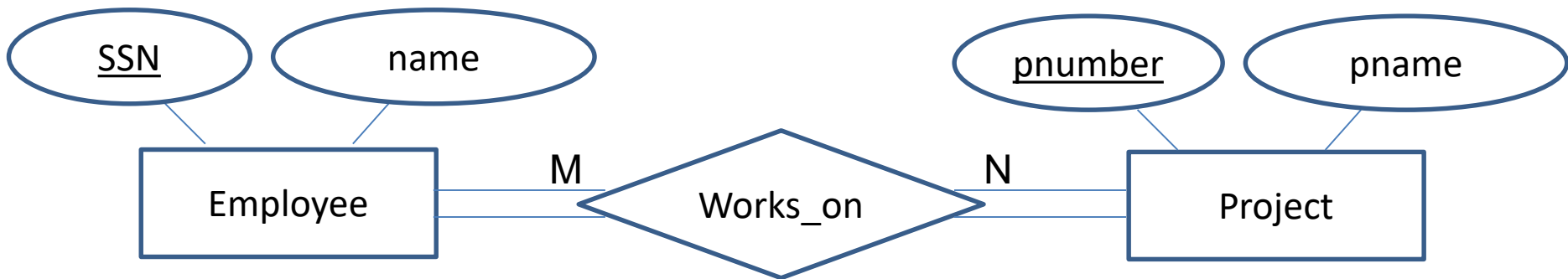
Foreign Key

Department

<u>dnumber</u>	dname
1	CS
2	QA

Primary Key

Example 3 (M:N Relationship)



- Why is it done this way? We will explain later.

Employee

<u>SSN</u>	name
1	Ahmad
2	Kamal
3	Sara

Primary Key

Works_on

<u>SSN</u>	<u>pno</u>
1	10
1	30
2	20
3	30

Foreign Key

Foreign Key

Project

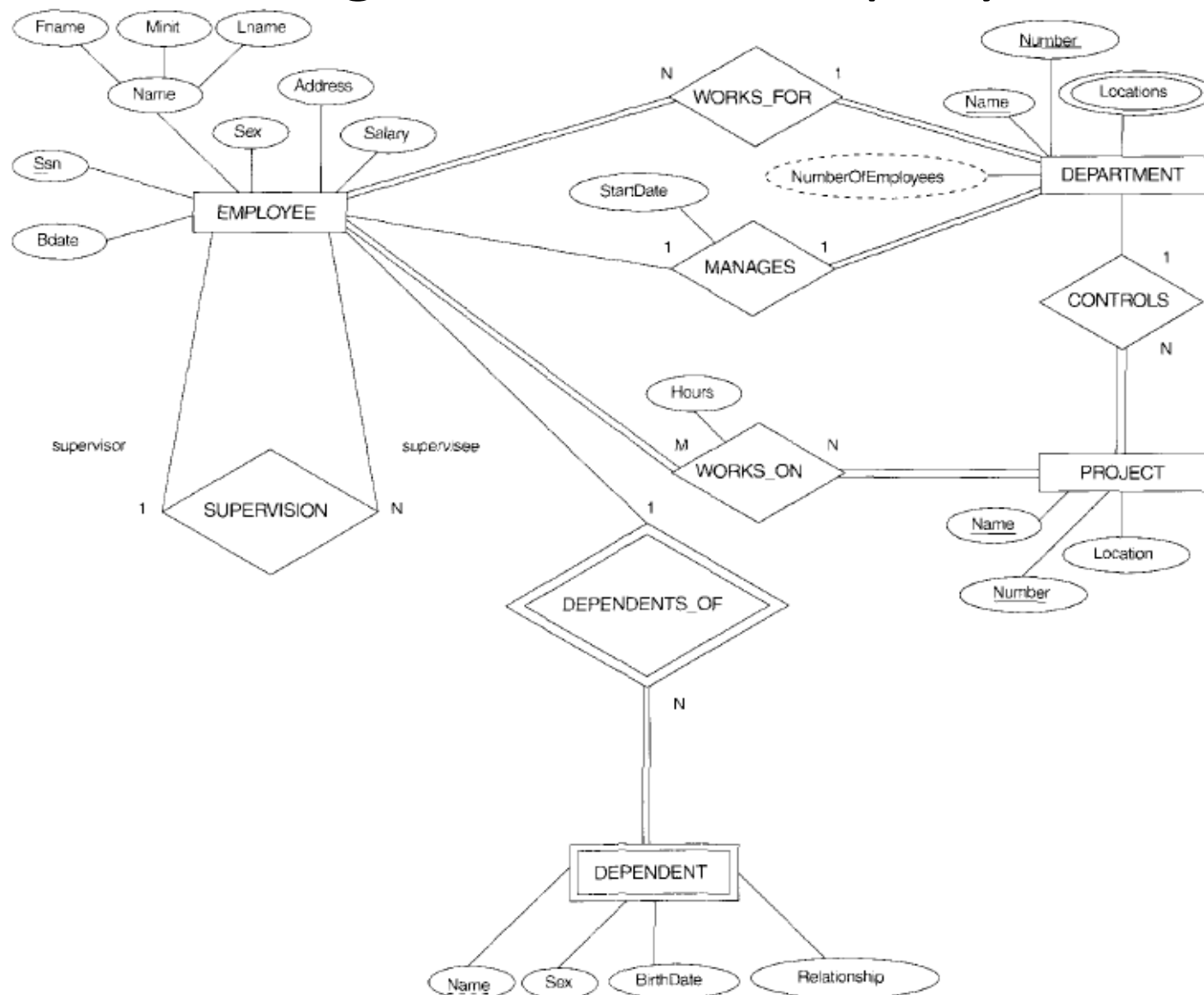
<u>pnumber</u>	pname
10	Abdali
20	Jaleel
30	Shaheq

Primary Key

Relations

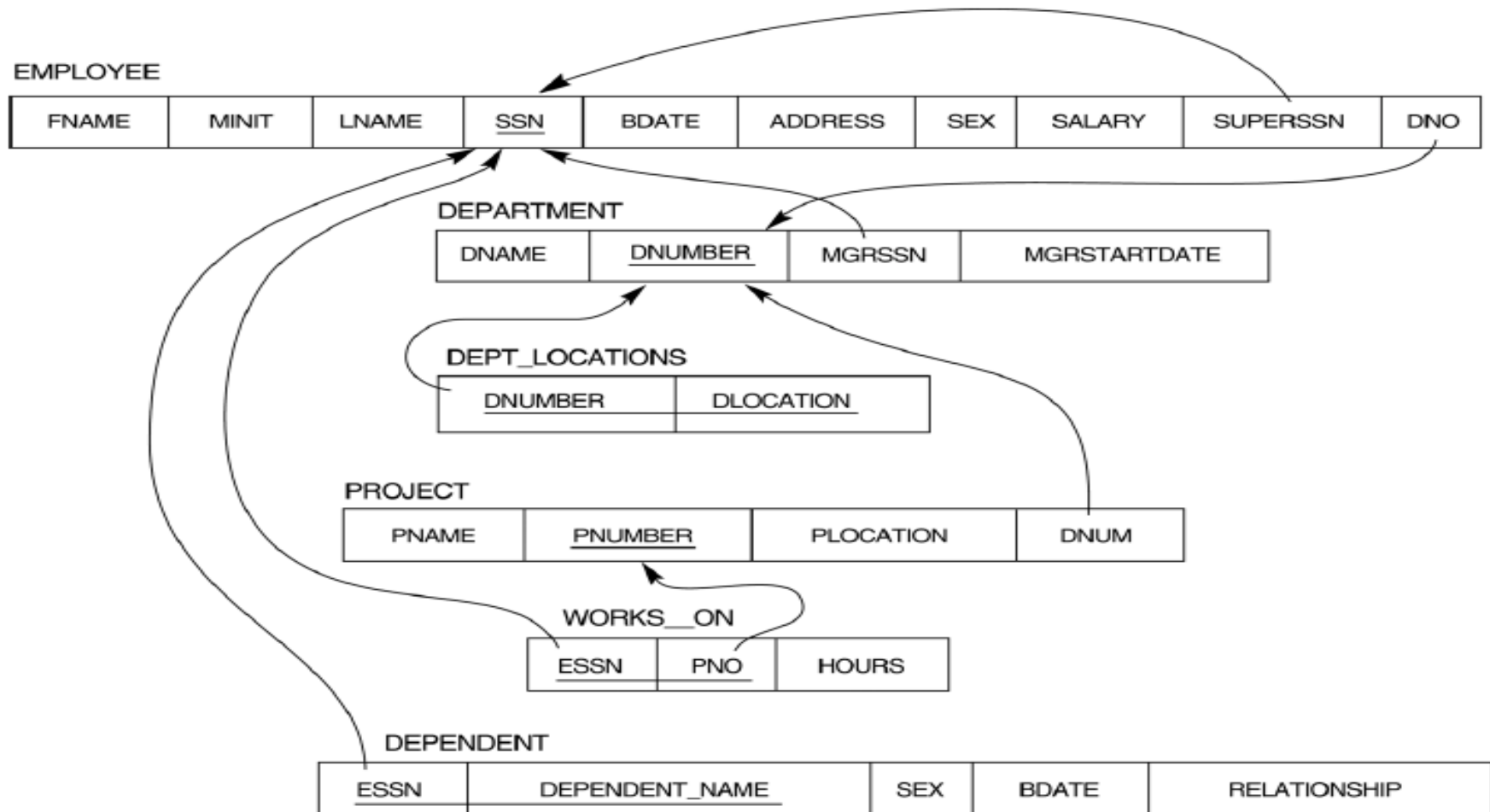
- In relational model, tables or entity types are called relations.
- In relational model, a relationship between two entity types is a relationship between two relations.
- What we do in relational model is:
 - Map Entity types into relations.
 - Map relationships into connections between relations.

ER Schema diagram for the Company Database



Company DB represented in Relational Model

This schema is the result of mapping company ER diagram



Mapping ER Diagram to Relations

- Steps in “ER” to relational model mapping:
 - 1) Mapping of Regular Entity Types
 - 2) Mapping of Weak Entity Types
 - 3) Mapping of Binary 1:1 Relation Types
 - 4) Mapping of Binary 1:N Relationship Types.
 - 5) Mapping of Binary M:N Relationship Types.
 - 6) Mapping of N-ary Relationship Types.
 - 7) Mapping of Multi-valued attributes.
- Complete mapping process requires completing **all** the previous steps.
- **Derived attributes are excluded from mapping because they are not stored in DB.**

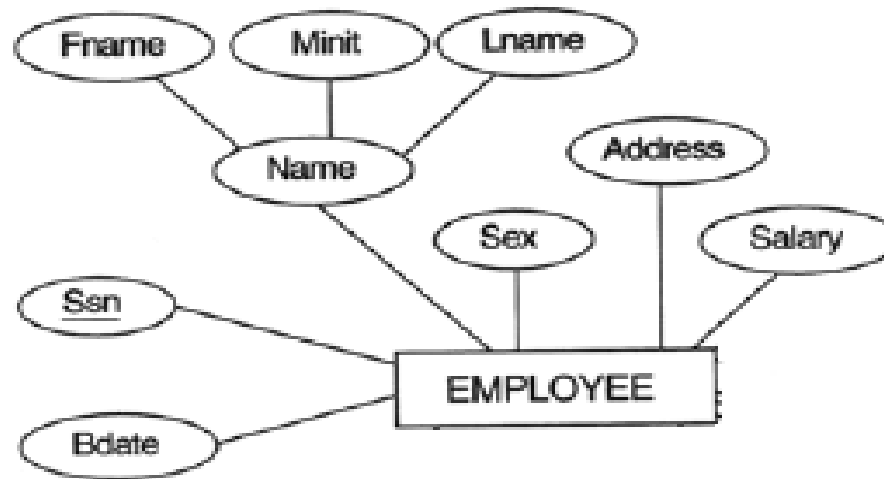
Note

- The mapping process is accumulative. In other words, tables might change many times until the mapping process is complete.

Mapping ER Diagram to Relations

- Step 1: Mapping of Regular (Strong) Entity Types
 - A regular entity type is mapped into one relation.
 - The relation contains:
 - All simple attributes
 - The primary key of the relation is the same as the entity type.
 - Put a line under all attributes that form a primary key of new relation.

Example



EMPLOYEE

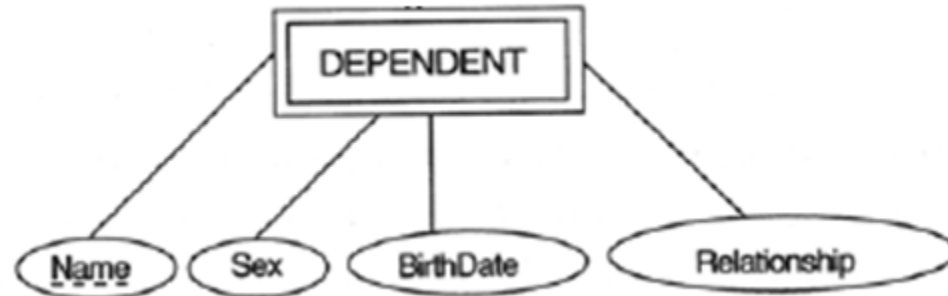
FNAME	MINIT	LNAME	<u>SSN</u>	BDATE	ADDRESS	SEX	SALARY
-------	-------	-------	------------	-------	---------	-----	--------

Notice that in case we have a composite attribute (eg. name), we only include the simple attributes that form the composite attribute into the relation.

Mapping ER Diagram to Relations

- Step 2: Mapping Weak Entity Types
 - A weak entity type is mapped into one relation.
 - The relation contains:
 - All **simple** attributes +
 - A **foreign key** that represents the primary key of the owner (strong) entity type it is connected to.
 - The primary key of the new relation is:
 - **Partial key of weak entity type + foreign key**
 - Put a line under all attributes that form a primary key of new relation.

Example



Remember that “Dependent” is connected to “Employee” via “dependent_of” relationship:

- Weak Entity: Dependent
- Strong Entity: Employee
- Weak entity partial key: Name
- Strong Entity primary key: SSN

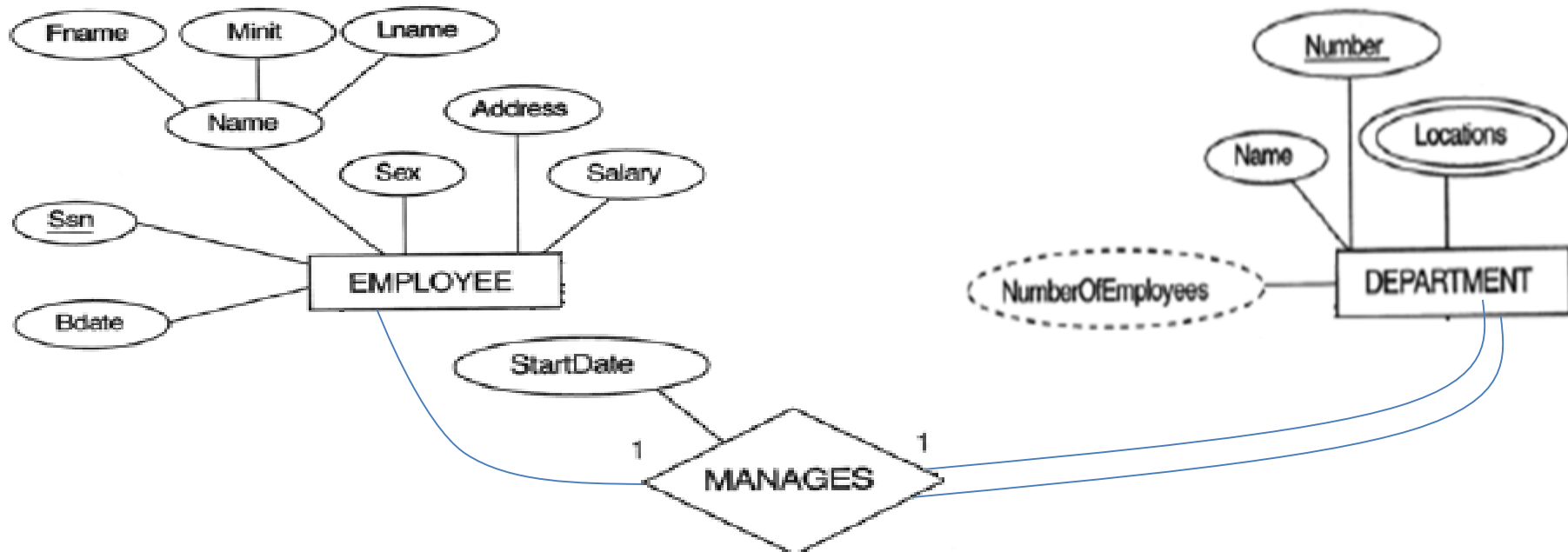
DEPENDENT				
<u>ESSN</u>	<u>DEPENDENT_NAME</u>	SEX	BDATE	RELATIONSHIP

Foreign Key in Dependent

Mapping ER Diagram to Relations

- Step 3: Mapping of Binary 1:1 Relation Types
 - Suppose E1 entity type is connected to E2 entity type via relationship r1.
 - Remember that E1 is mapped into relation R1.
 - Remember that E2 is mapped into relation R2.
 - We can use foreign keys to map “1:1” relations
 - Put the primary key of R1 into R2 or vice versa.
 - If the relationship has simple attributes, put them in the relation in which you added the foreign key.
 - It is better to have the foreign key on the relation that has a total participation, why? (No null values).
 - Example: Employee “manages” Department

Example



DEPARTMENT

DNAME	<u>DNUMBER</u>	MGRSSN	MGRSTARTDATE
-------	----------------	--------	--------------

SSN of Employee is used as a foreign key in Department relation


startDate of Employee is put in Department relation

Locations is a multi-valued attribute. It will be mapped separately later on

Mapping ER Diagram to Relations

- It is better to have the **foreign key** on the relation that has a **total participation**, why? (No null values) (Saving Space). Some employees are not managers.

department



<u>dnumber</u>	dname	mgrssn	mgrstartdate
1	CS	10	21-Apr-80
2	CIS	13	18-Jan-82

employee

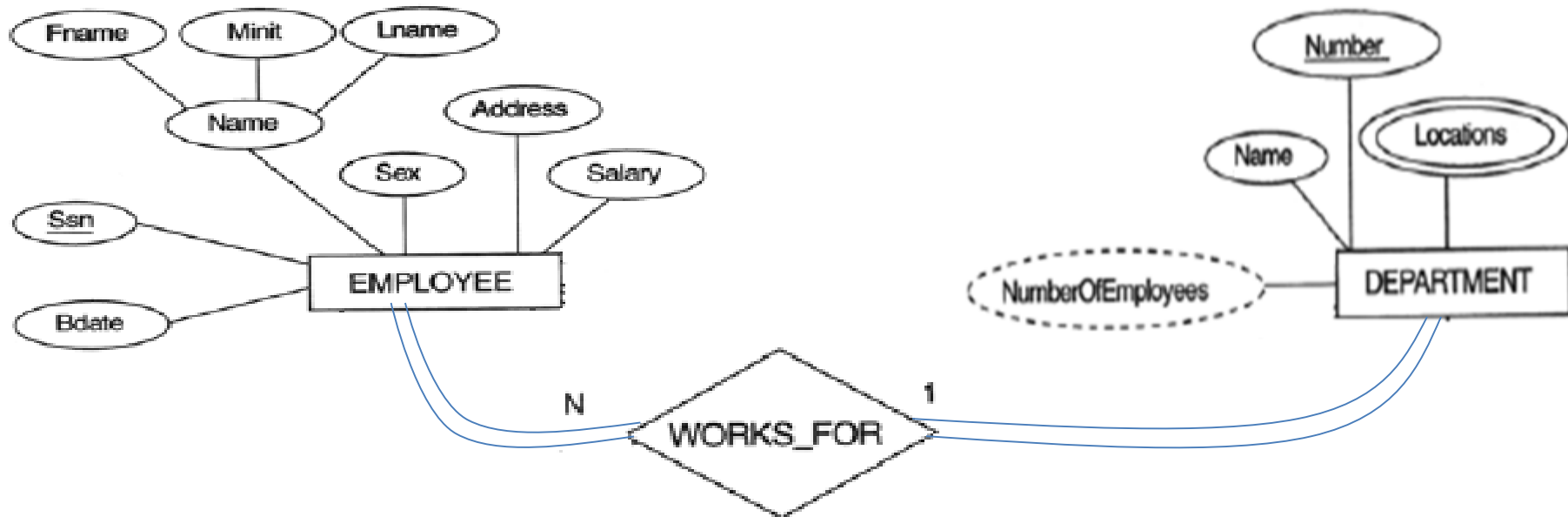


fname	minit	lname	<u>ssn</u>	bdate	address	sex	salary	DepNumber	mgrstartdate
Ahmad	M	Mazen	10	1	21-Apr-80
Sara	S	Salam	11	Null	Null
Farah	G	Talal	12	Null	Null
Kamal	R	Lafi	13	2	18-Jan-82

Mapping ER Diagram to Relations

- Step 4: Mapping of 1:N Relation Types
 - Suppose E1 entity type is connected to E2 entity type via relationship r1.
 - The relationship r1 is “1” from E1 side and “N” from E2 side.
 - Remember that E1 is mapped into relation R1.
 - Remember that E2 is mapped into relation R2.
 - The mapping steps are:
 - Put the primary key of R1 (The “1” side) as a foreign key in relation R2 (The “N” side).
 - Put any simple attributes of relationship r in relation R2 (The “N” side).

Example 1



EMPLOYEE

FNAME	MINIT	LNAME	<u>SSN</u>	BDATE	ADDRESS	SEX	SALARY	DNO
-------	-------	-------	------------	-------	---------	-----	--------	-----


“number” of a department is a primary key in Department, it is put as a foreign key in Employee relation (renamed as DNO)

Question

- In mapping 1:N relationship, why do you put the foreign key on the N side? (eg. Employee works_for Department)

Foreign Key on the N side

Employee




<u>SSN</u>	name	...	dno
1	Kamal	...	1
2	Hala	...	1
3	Sameh	...	2

Foreign Key on the 1 side

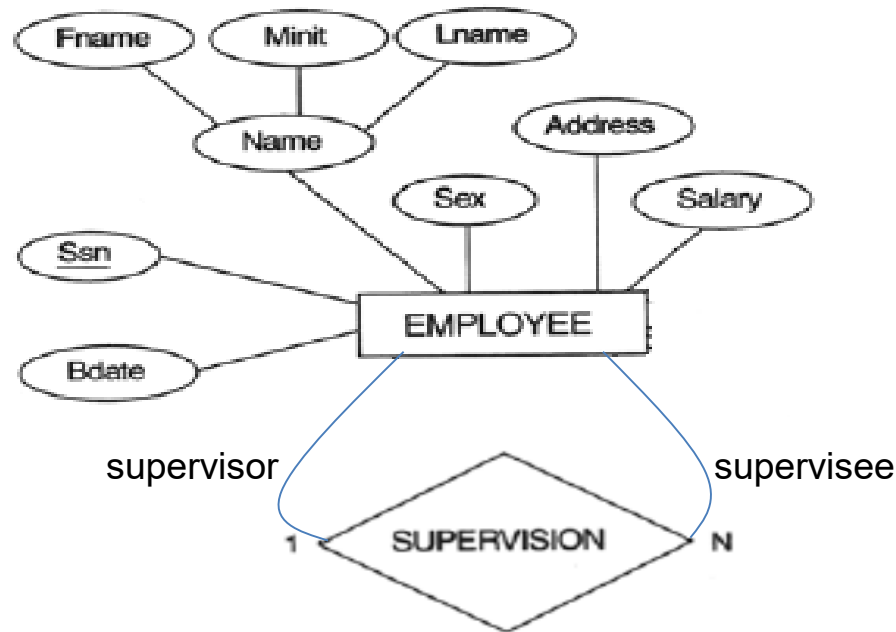
Department

<u>dnumber</u>	dname	SSN
1	CS	1
1	CS	2
2	Math	3



The ordinary primary key “dnumber” is not a primary key any more.

Example 2

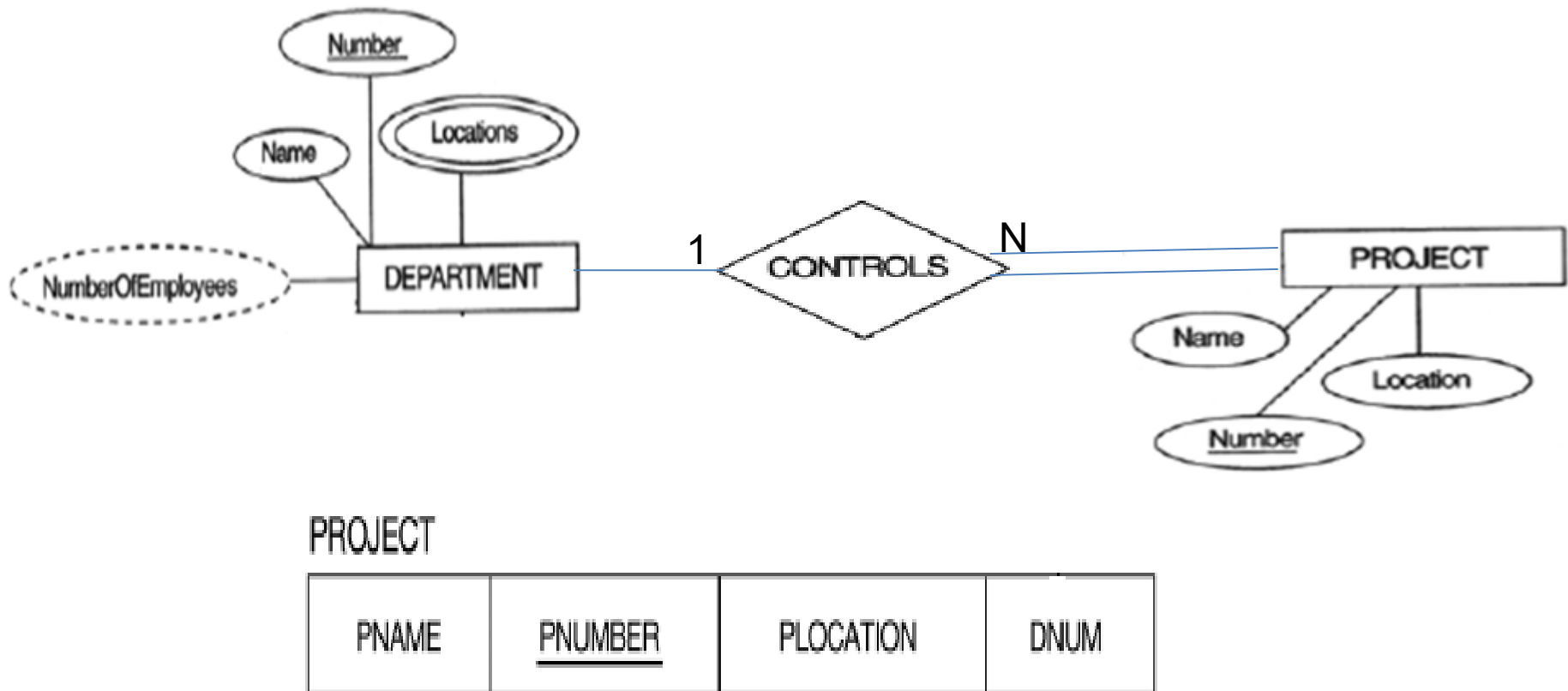


EMPLOYEE

FNAME	MINIT	LNAME	<u>SSN</u>	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
-------	-------	-------	------------	-------	---------	-----	--------	----------	-----

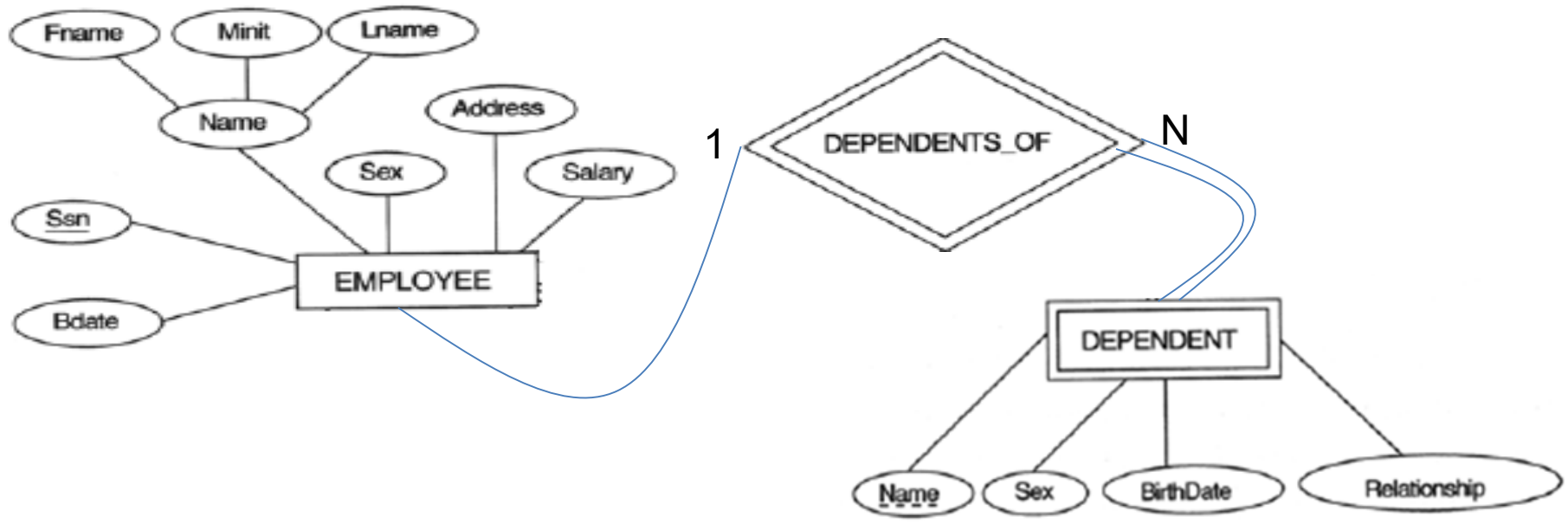
“SSN” of Employee supervisor is a primary key in Employee, it is put as a foreign key in Employee relation itself(renamed as SUPERSSN). This is an example of a reflexive relationship.

Example 3



“number” of department is a primary key in Department, it is put as a foreign key in Project relation (renamed as DNUM).

Example 4



DEPENDENT

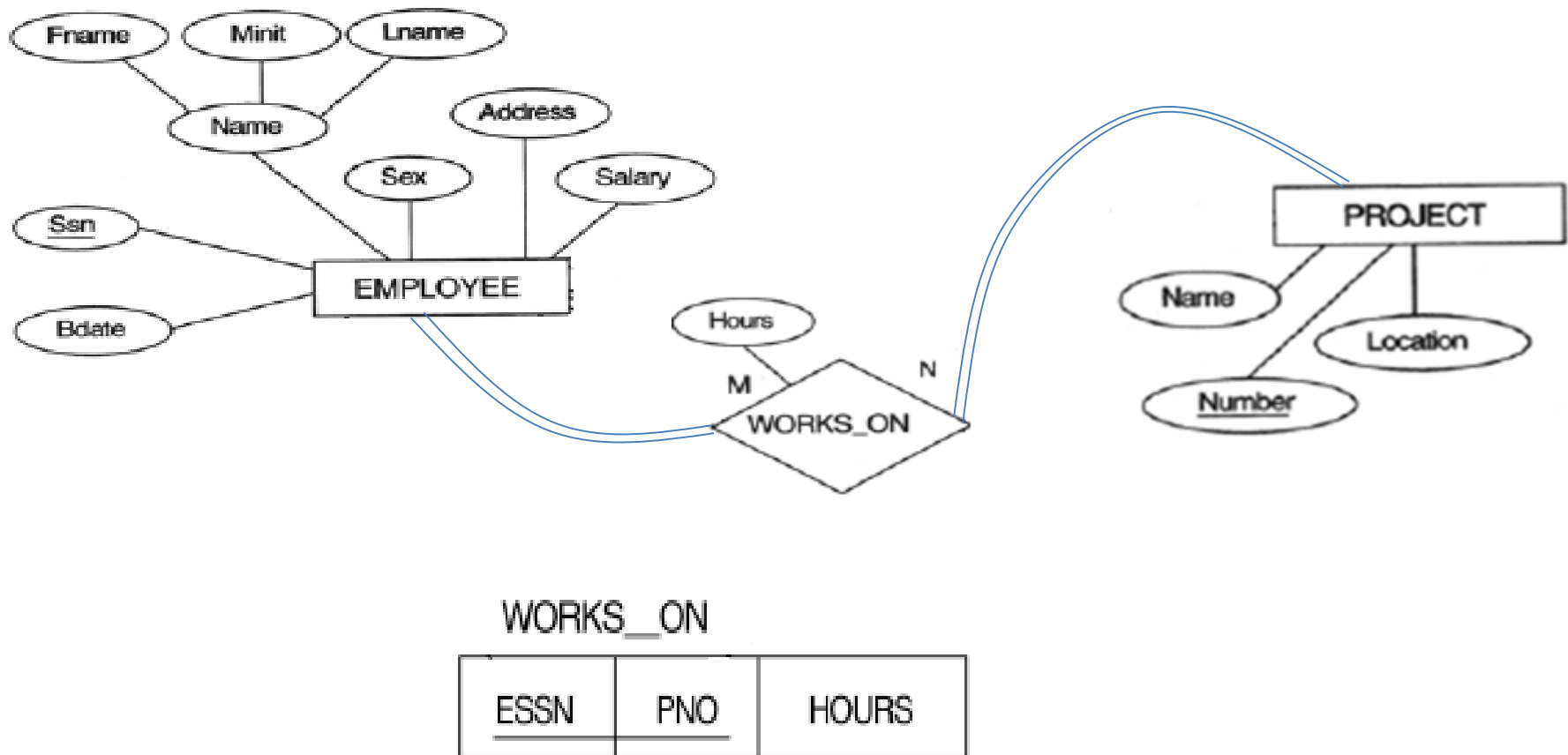
<u>ESSN</u>	<u>DEPENDENT_NAME</u>	SEX	BDATE	RELATIONSHIP
-------------	-----------------------	-----	-------	--------------

- “SSN” of Employee is a primary key in Employee, it is put as a foreign key in Dependent relation (renamed as ESSN).
- **Remember that we already did this when we mapped Dependent weak entity type. So, you should not do this again.**

Mapping ER Diagram to Relations

- Step 5: Mapping of M:N Relation Types
 - Suppose E1 entity type is connected to E2 entity type via relationship r1.
 - The relationship r1 is “M” from E1 side and “N” from E2 side
 - Remember that E1 is mapped into relation R1.
 - Remember that E2 is mapped into relation R2.
 - The mapping steps are:
 - Put the primary key of R1 as a foreign key in new relation R3.
 - Put the primary key of R2 as a foreign key in new relation R3.
 - Put any simple attributes of relationship r into relation R3.
 - The primary key of R3 is a combination of primary key of R1 and primary key of R2.

Example

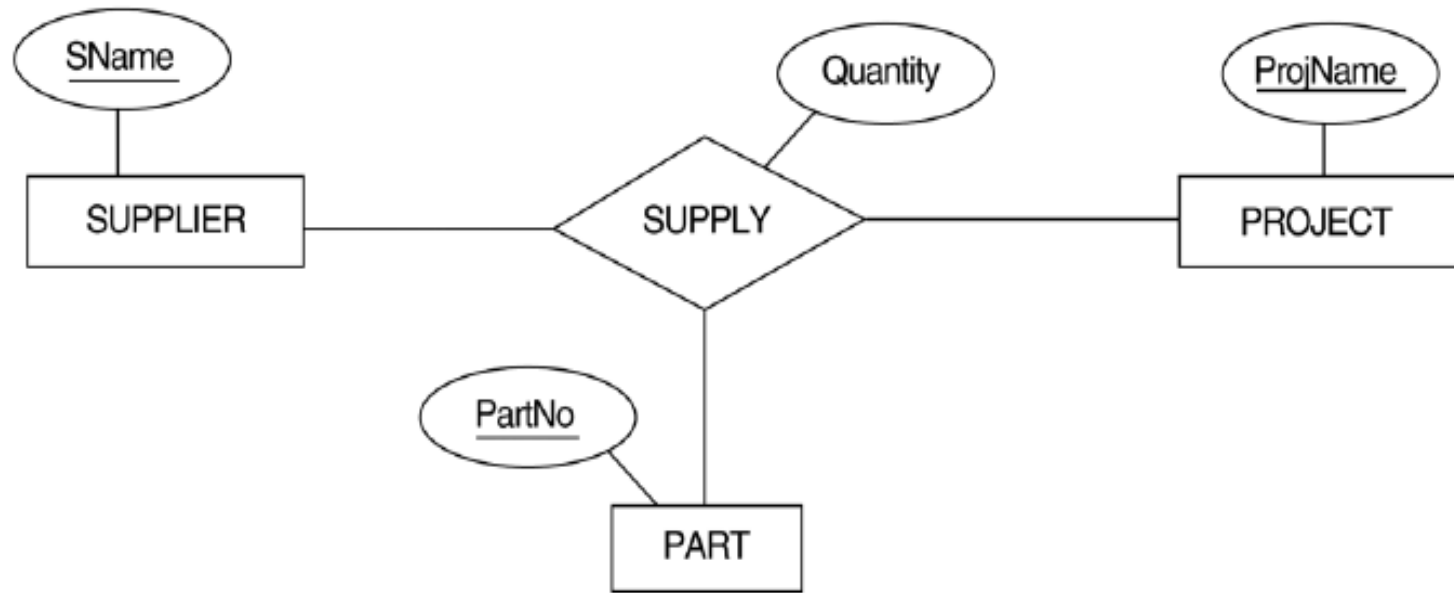


- “SSN” of Employee (renamed as ESSN) and Number of project (renamed as PNO) and attribute “hours” of the relationship are put in a new relation “works_for”.
- ESSN is a foreign key. PNO is also a foreign key.

Mapping ER Diagram to Relations

- Step 6: Mapping of N-arry Relation Types
 - It is done in the same way we mapped M:N relationship type:
 - Put primary keys of all relations into a new relation.
 - Put simple attributes of relationship (if any) in the new relation.
 - The primary key of the new relation is the combination of all keys of all relations.

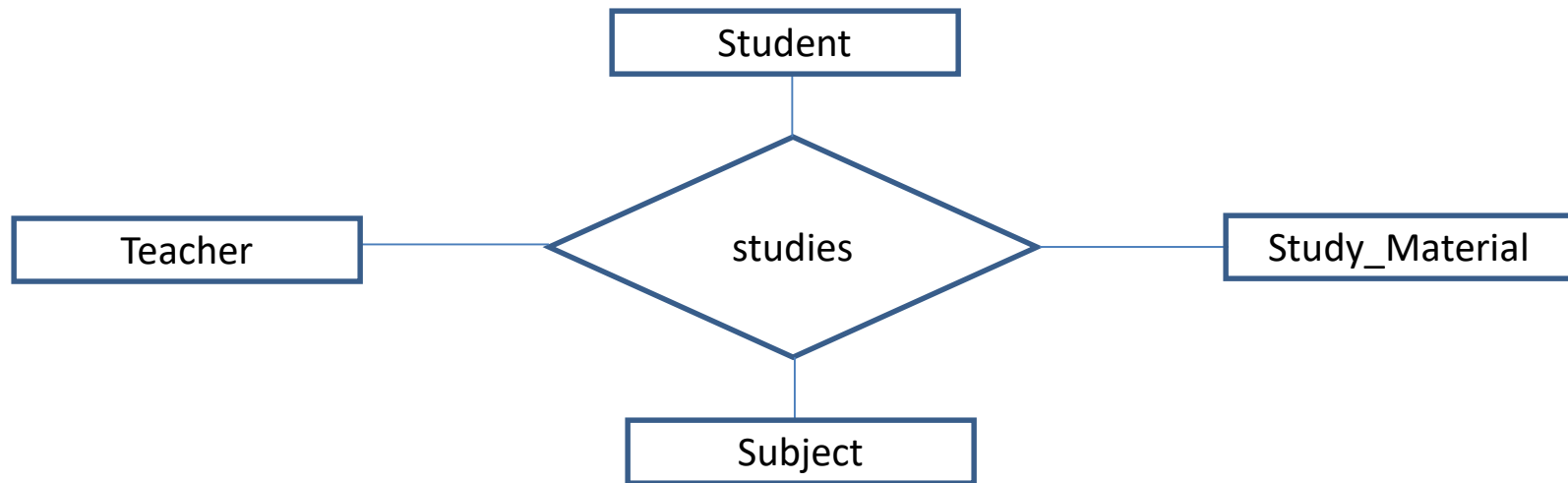
Example 1



SUPPLY

<u>SNAME</u>	PROJNAME	<u>PARTNO</u>	QUANTITY
--------------	----------	---------------	----------

Example 2



Studies

<u>studentID</u>	teacherID	materialID	<u>subjectID</u>
------------------	-----------	------------	------------------

The primary keys of all 4 relations are put as foreign keys in Studies relation

Mapping ER Diagram to Relations

- Step 7: Mapping of Multi-valued attributes
 - Suppose entity type “E1” has a multi-valued attribute “A”.
 - Remember that E1 is mapped into relationship R1.
 - Mapping steps:
 - Put primary key of R1 as a foreign key in new relation.
 - Put the multi-valued attribute in the new relation.
 - The primary key of new relation is a combination of:
 - Primary key of R1 +
 - Multi-valued attribute

Example



DEPT_LOCATIONS


<u>DNUMBER</u>	<u>DLOCATION</u>

- The primary key of Department is put as a foreign key in “Dept_Locations” relation (renamed as DNUMBER).
- The multi-valued attribute “Locations” is put in “Dept_Locations” relation.

Question

- Why do not we map the multi-valued attribute on the same entity type relation that it belongs to?

Department



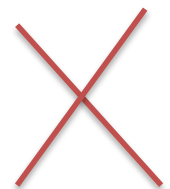
<u>dnumber</u>	dname
1	CS
2	CIS

Dept_locations

<u>dnumber</u>	<u>location</u>
1	Abdoun
1	Swaifiyeh
2	Abdoun

Department

<u>dnumber</u>	dname	location
1	CS	Abdoun
1	CS	Swaifiyeh
2	CIS	Abdoun



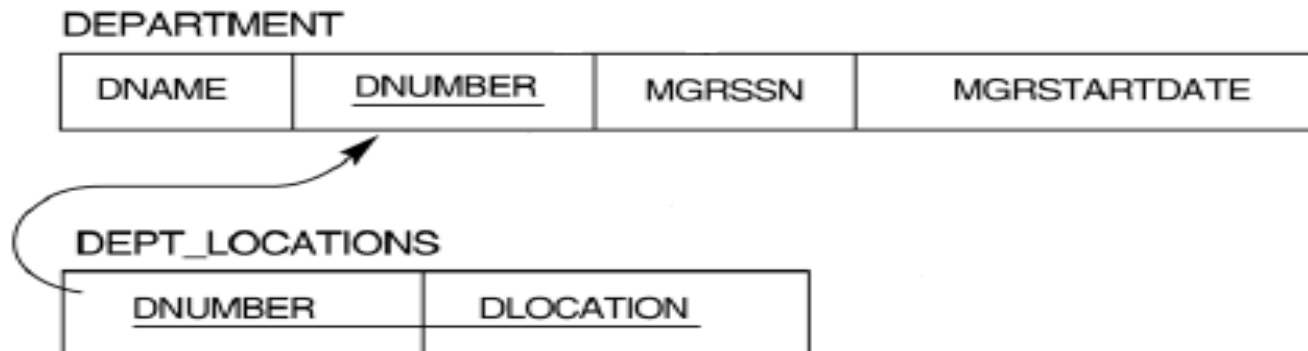
Dnumber is not primary key any more.
But, it should be a primary key for
Department. So, we must map the multi-
valued attribute into separate relation.

Note about relationship attributes

- As you have seen from previous examples, relationship attributes always go to the same table in which you put foreign keys resulted from mapping that relationship.

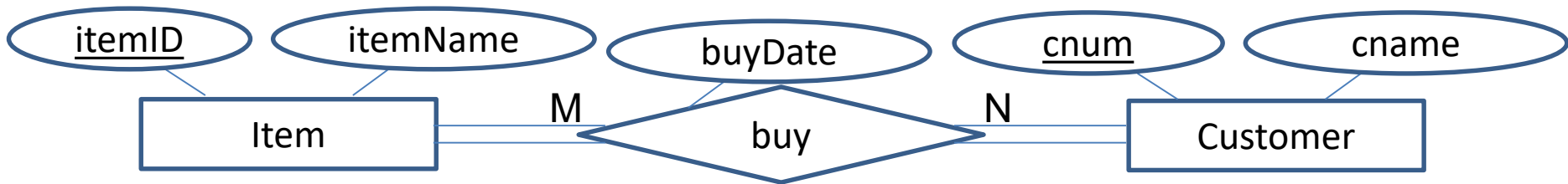
Notes

- In all new relations, underline the attributes that form primary keys.
- In a primary/foreign key relationship:
 - **Draw an arrow that connects each foreign key to the primary key attribute that it came from. The arrow points to the primary key.**
 - Example



Important Note

- It is very important to check the primary key of tables. If primary key of a table is not correct, then you need to change the design of your ERD
- Example: In ERD for supermarket. According to rules, “buy” primary key is itemID + cnum. **But this is wrong**. So, we need to redesign ERD (Next Slide)



Item	
<u>itemID</u>	itemName
1	Milk



buy		
<u>itemID</u>	<u>cnum</u>	buyDate
1	5	12-Apr-80 12:30:04
1	5	13-Apr-80 12:30:04
1	6	13-Apr-80 12:30:04



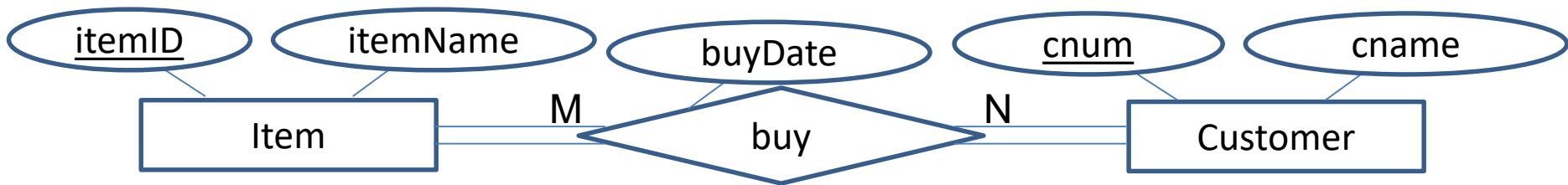
Customer	
<u>cnum</u>	cname
5	Ahmad



Correct Solution in next slide

Important Note

- **Solution 1**: Leave ERD design as is and consider buyDate to be part of the primary key.
 - Now primary key is complex

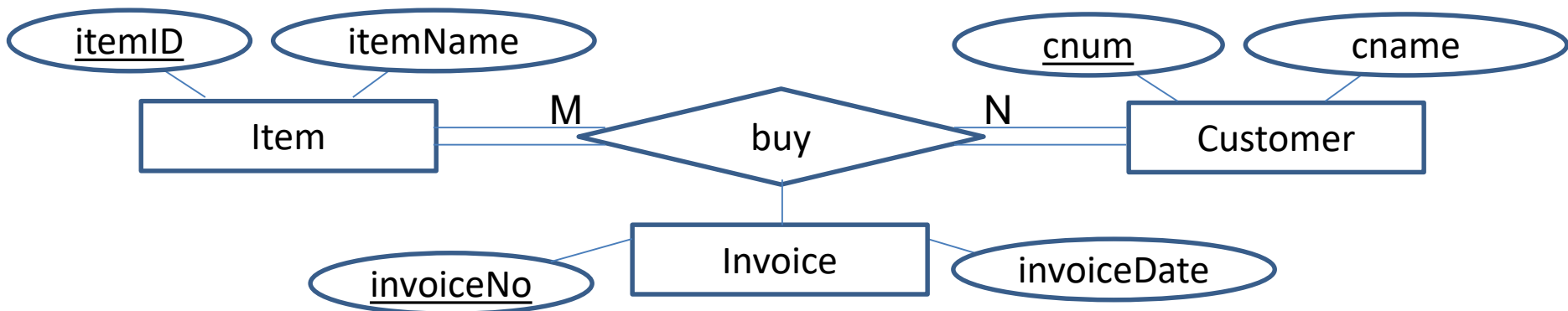


buy

<u>itemID</u>	<u>cnum</u>	<u>buyDate</u>
1	5	12-Apr-80 12:30:04
1	5	13-Apr-80 12:30:04
1	6	13-Apr-80 12:30:04

Important Note

- **Solution 2**: Change ERD design by adding “Invoice” entity
 - Simple Key
 - Each buy operation has a different invoice number



buy

<u>invoiceID</u>	itemID	cnum	invoiceDate
1	1	5	12-Apr-80 12:30:04
2	1	5	13-Apr-80 12:30:04
3	1	6	13-Apr-80 12:30:04

In Class Exercise 1

- Previously, during classes, we designed ER diagram for one zoo
- Map it into relations (Schema)

In Class Exercise 2

- Previously, during classes, we designed ER diagram for a several interconnected libraries
- Map it into relations (Schema)

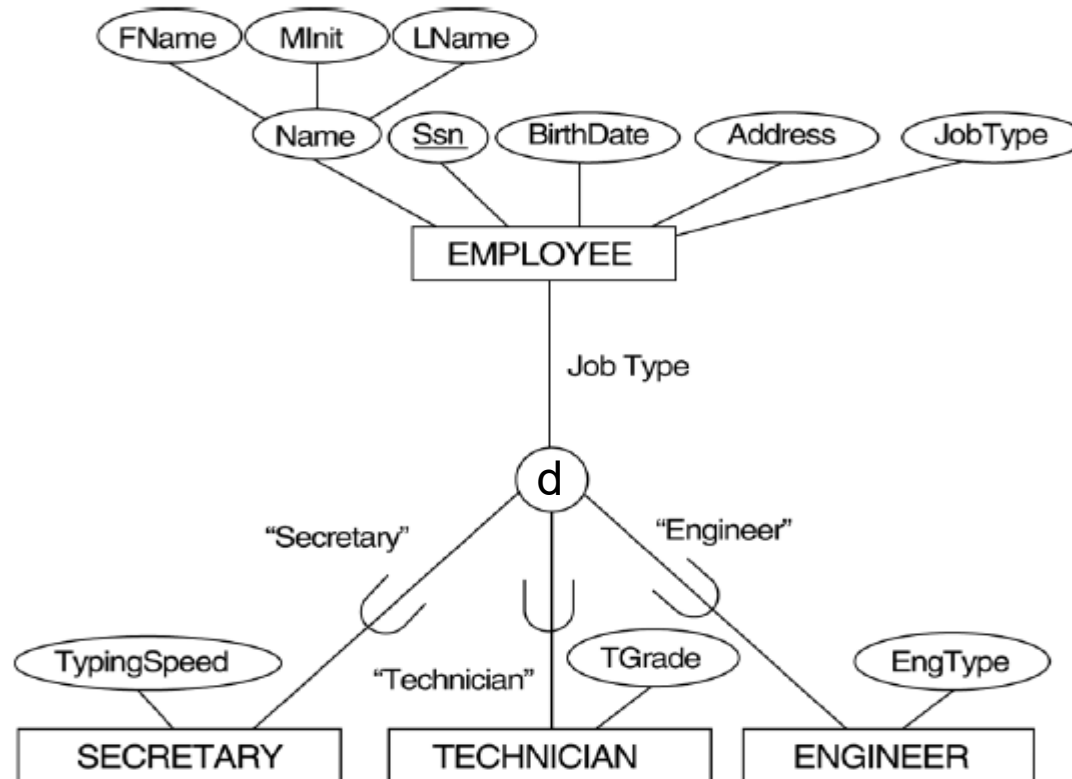
Mapping EER Diagram to Relations

- Options for mapping Specialization/Generalization:
 - **Option 1:** Multiple relations-superclass and subclasses.
 - **Option 2:** Multiple relations-subclass relations only
 - **Option 3:** Single relation with one type attribute
 - **Option 4:** Single relation with multiple type attributes.

Mapping EER Diagram to Relations

- **Option 1: Multiple relations-superclass and subclasses.**
 - Suppose a class C is specialized into $\{S_1, S_2, \dots, S_m\}$ subclasses .
 - Suppose attributes of C are $\{k, a_1, \dots, a_n\}$ and k is the (primary) key of C .
 - Mapping steps:
 - **Create a relation for superclass C and put all attributes of C in the relation.**
 - **Create a relation for each subclass and put all attributes of each subclass in its corresponding relation.**
 - Add the primary key of C in each subclass relation.
 - The primary key of superclass relation is K .
 - The primary key of each subclass is K .
 - **Option1 works for any kind of specialization (Total, Disjoint, ...).**

Example



EMPLOYEE

<u>SSN</u>	FName	MInit	LName	BirthDate	Address	JobType
------------	-------	-------	-------	-----------	---------	---------

SECRETARY

<u>SSN</u>	TypingSpeed
------------	-------------

TECHNICIAN

<u>SSN</u>	TGrade
------------	--------

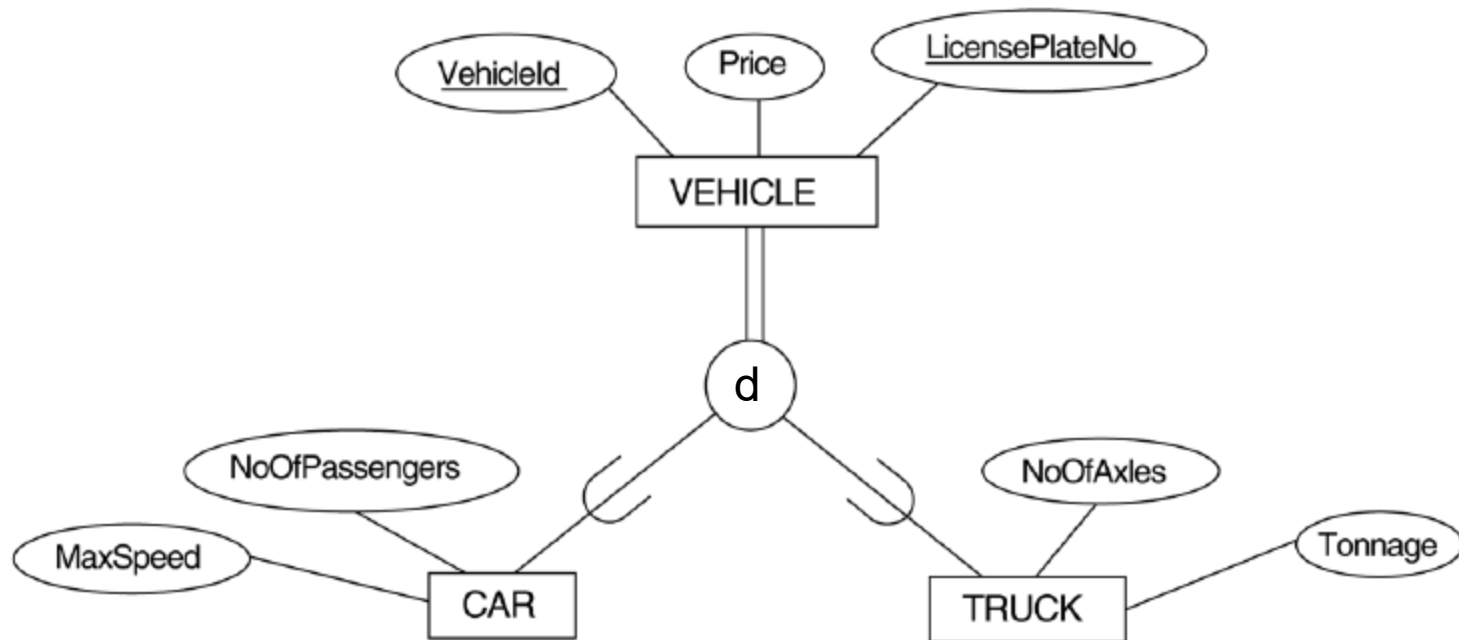
ENGINEER

<u>SSN</u>	EngType
------------	---------

Mapping EER Diagram to Relations

- **Option 2: Multiple relations-Subclass relations only.**
 - Suppose a class C is specialized into $\{S_1, S_2, \dots, S_m\}$ subclasses.
 - Suppose attributes of C are $\{k, a_1, \dots, a_n\}$ and k is the (primary) key of C .
 - Mapping steps:
 - **Create a relation for each subclass and put all attributes of each subclass in its corresponding relation.**
 - **Put all attributes of superclass C in each subclass.**
 - Here, the primary key for each subclass is K .
 - **Option2 works for only total specialization.**

Example



CAR

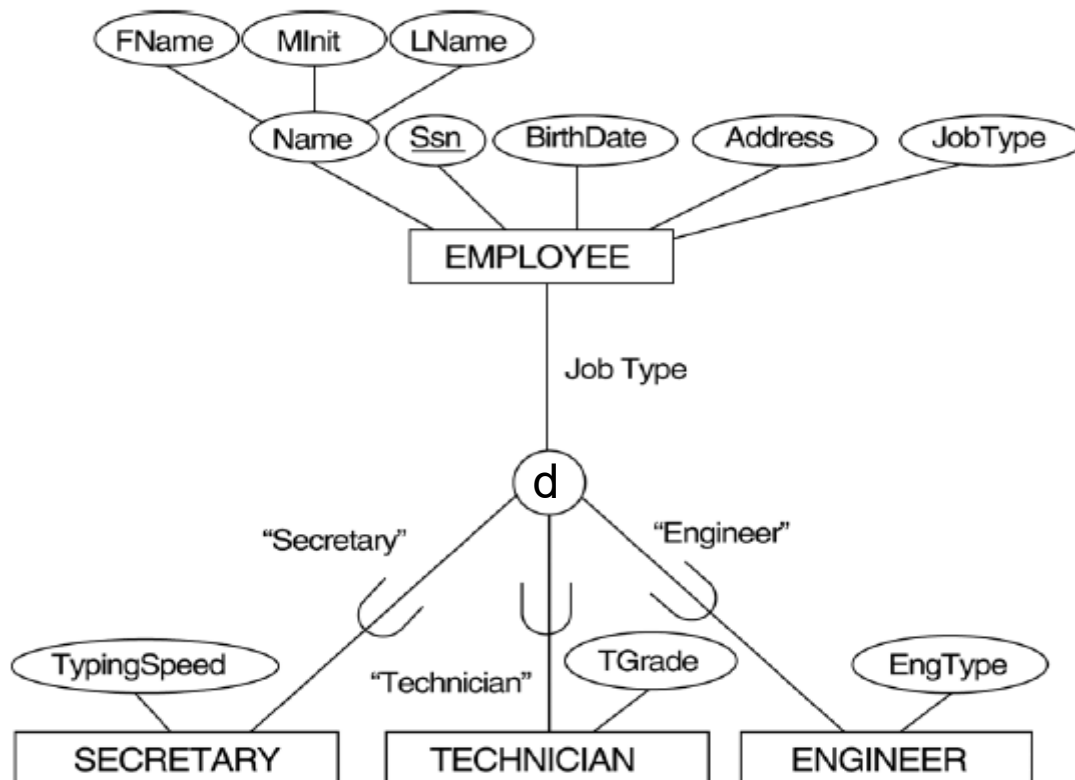
<u>VehicleId</u>	LicensePlateNo	Price	MaxSpeed	NoOfPassengers
------------------	----------------	-------	----------	----------------

TRUCK

<u>VehicleId</u>	LicensePlateNo	Price	NoOfAxes	Tonnage
------------------	----------------	-------	----------	---------

Question

- Why does “option2” work only in the case of total participation specialization?



If we use option 2, the outcome is:

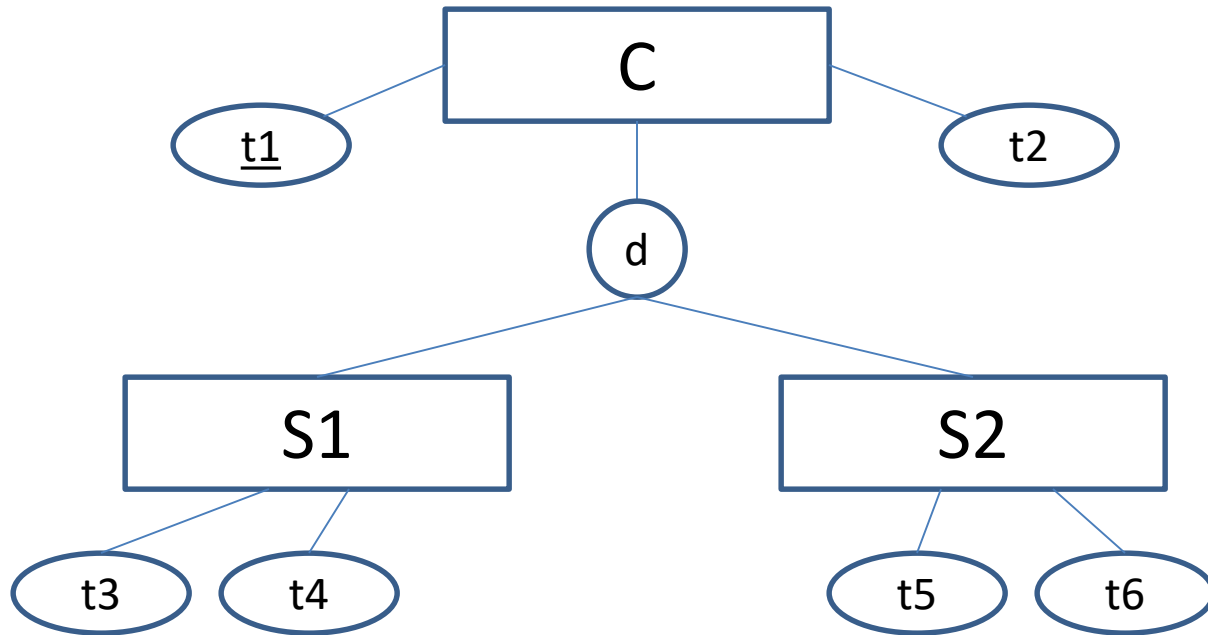
- Relation “Secretary”
- Relation “Technician”
- Relation “Engineer”

- Suppose you want to insert a new Employee in DB. He works as a Accountant.
- In which relation his information is stored??!!!

Mapping EER Diagram to Relations

- **Option 3: Single relation with one type attribute.**
 - Suppose a class C is specialized into {S1, S2,...,Sm} subclasses.
 - Suppose attributes of C are {k,a1,...an} and k is the (primary) key of C.
 - Mapping steps:
 - **Create “only one” relation. Put all attributes of superclass C in the relation.**
 - **Put all attributes of each subclass in the same new relation.**
 - **Add a new attribute to the relation, this attribute is used to discriminate between different entity types.**
 - Here, the primary key for our single relation is K.
 - **Option 3 works only for disjoint specialization.**
 - This option means that we can have many null values.

Example 1

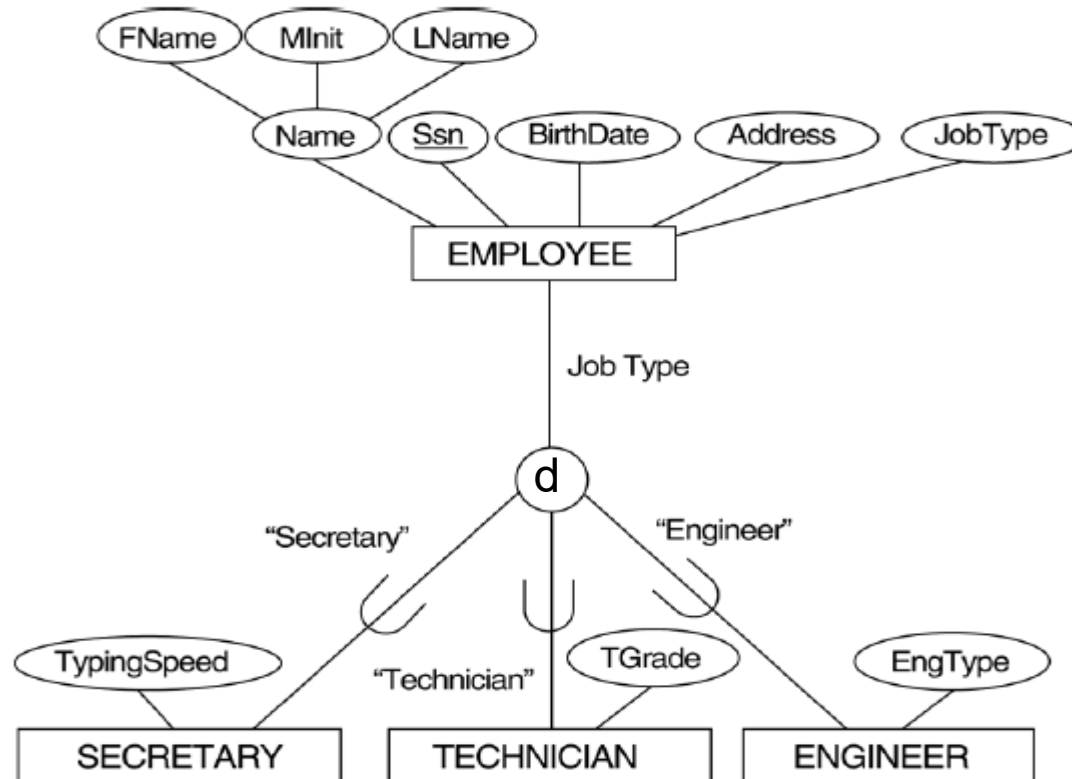


C

<u>t1</u>	t2	t3	t4	t5	t6	type
-----------	----	----	----	----	----	------

The value of “type” is either “S1” or “S2” or null, it can be null because it is a partial Specialization.

Example 2



EMPLOYEE

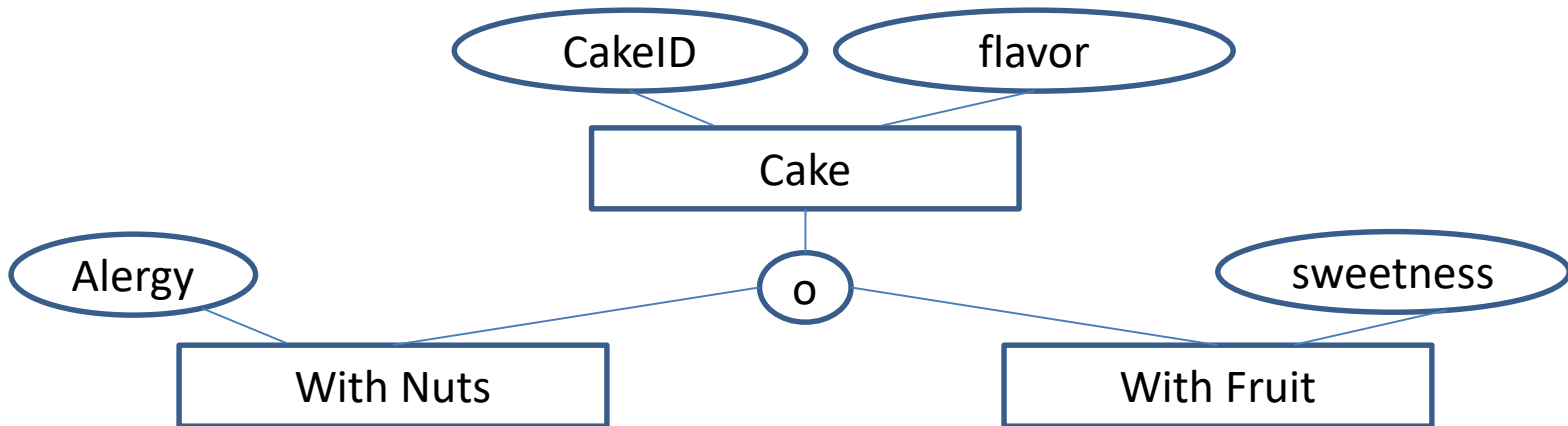
<u>SSN</u>	FName	MInit	LName	BirthDate	Address	JobType	TypingSpeed	TGrade	
------------	-------	-------	-------	-----------	---------	---------	-------------	--------	--



Here, we do not need to add a new attribute, because we already use **jobType** to discriminate between attributes. 47

Question

- Why does option3 work only for disjoint specialization?



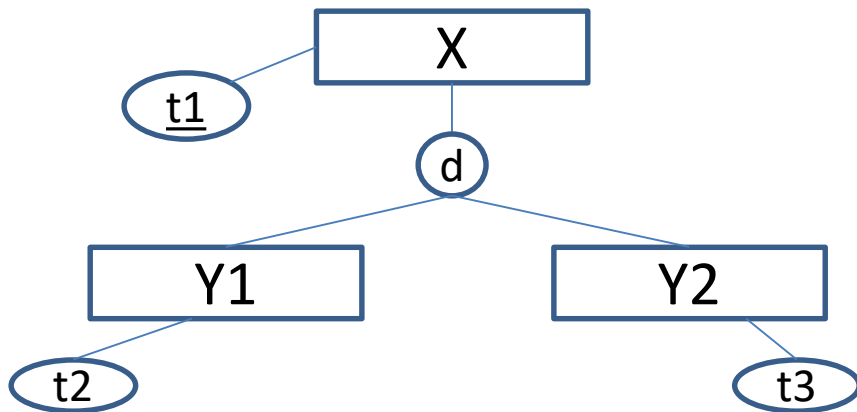
<u>cakeID</u>	flavor	alergy	sweetness	type
1	N
2	F
3	N & F

- N stands for “With Nuts”.
- F stands for “With Fruit”.
- type only fits one value
- Cake 3 is made with fruit and nuts



Question

- Why does option 3 generate lots of null values?
- Answer by example



X

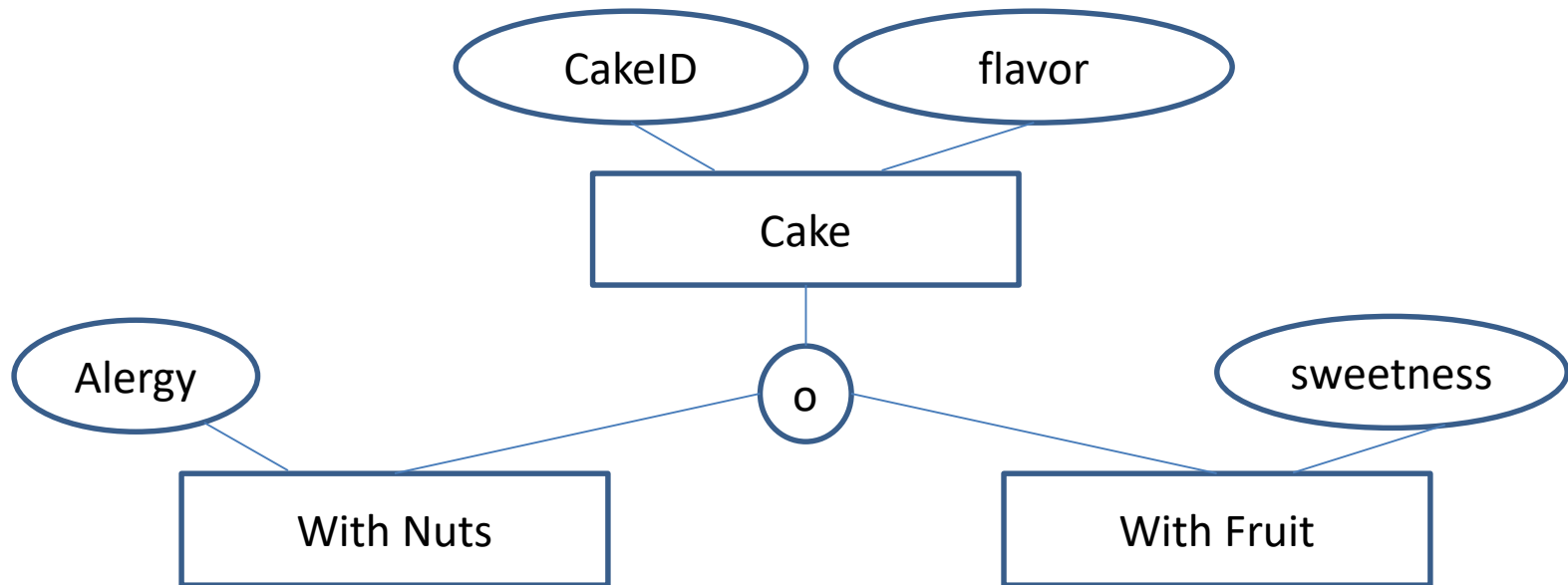
<u>t1</u>	t2	t3	X-type
1	val1	null	Y1
2	null	val4	Y2
3	null	null	null

Because it is partial, then we can have many instances that are not Y1 or Y2.

Mapping EER Diagram to Relations

- **Option 4: Single relation with multiple type attributes.**
 - Suppose a class C is specialized into $\{S_1, S_2, \dots, S_m\}$ subclasses .
 - Suppose attributes of C are $\{k, a_1, \dots, a_n\}$ and k is the (primary) key of C .
 - Mapping steps:
 - **Create “only one” relation. Put all attributes of superclass C in the relation.**
 - **Put all attributes of each subclass in the same new relation.**
 - **Add new attributes to the relation, each attribute corresponds to one subclass type. Each attribute is a Boolean attribute used to discriminate between different entity types.**
 - Here, the primary key for our single relation is K .
 - **Option 4 works for overlapped specialization (but also works for disjoint).**

Example



Cake

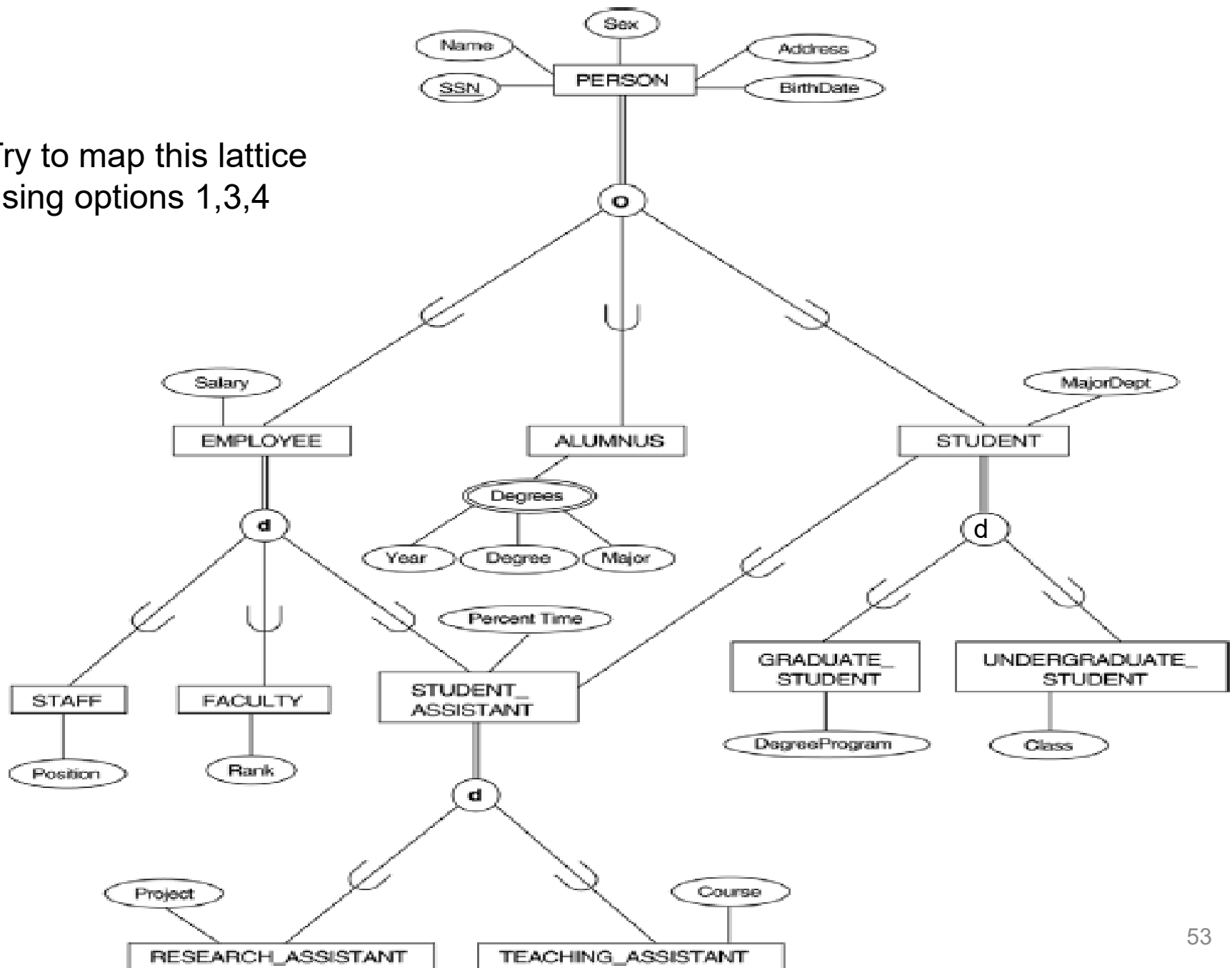
<u>cakeID</u>	flavor	alergy	sweetness	FruitFlag	NutsFlag
---------------	--------	--------	-----------	-----------	----------

For example, if we are adding a new “Cake” that is made with nuts, then:
FruitFlag = false and NutsFlag=true

Mapping EER Diagram to Relations

- Mapping of shared subclasses (Multiple Inheritance)
 - You can use any of the options 1,2,3,4 to map shared classes taking into consideration the restrictions of each option.

Try to map this lattice using options 1,3,4



Example

- On the previous lattice: (Try mappings in a bottom-up) fashion:
 - 1) Student_Assistant is specialized into{Research_Assistant, Teaching_Assistant}. (Map it using **option 4**).
 - 2) Student is specialized into{Graduate_student, Undergraduate_Student}. (Map it using **option 4**).
 - 3) Employee is specialized into{Staff, Faculty, Student_Assistant}. (Map it using **option 3**).
 - 4) Person is specialized into {Employee, Alumnus, Student}. (Map it using **option 1**).

Example (Continue)

- 1) Student_Assistant is specialized into {Research_Assistant, Teaching_Assistant}. (Map it using **option 4**).

Student_Assistant

project	course	Percent_time	TA_Flag	RA_Flag
---------	--------	--------------	---------	---------

Example (Continue)

2) Student is specialized into {Graduate_student, Undergraduate_Student}. (Map it using **option 4**).

Student

Major_Dep	Degree_Program	class	Grad_Flag	Undergrad_Flag	studAssistantFalg
-----------	----------------	-------	-----------	----------------	-------------------

Example (Continue)

- 3) Employee is specialized into {Staff, Faculty, Student_Assistant}.
(Map it using **option 3**).

Employee

project	course	Percent_time	TA_Flag	RA_Flag	rank	position	salary	etype
---------	--------	--------------	---------	---------	------	----------	--------	-------

~~Student_Assistant~~

project	course	Percent_time	TA_Flag	RA_Flag
--------------------	-------------------	-------------------------	--------------------	--------------------

Employee will take all attributes in Student_Assistant relation. So, we do not need Student_Assistant any more (Delete Student_Assistant relation)

Example (Continue)

4) Person is specialized into {Employee, Alumnus, Student}. (Map it using option1).

Person

<u>SSN</u>	name	Birth_date	sex	address
------------	------	------------	-----	---------

Employee

<u>SSN</u>	project	course	Percent_time	TA_Flag	RA_Flag	rank	position	salary	etype
------------	---------	--------	--------------	---------	---------	------	----------	--------	-------

Alumnus

<u>SSN</u>

Student

<u>SSN</u>	MajorDep	DegreeProgram	class	Grad_Flag	Undergrad_Flag	studAssistantFalg
------------	----------	---------------	-------	-----------	----------------	-------------------

Example (Continue)

- 5) Alumnus has a multi-valued attribute. It should be mapped in a separate relation.

Alumnus_Degrees

<u>SSN</u>	degree	year	<u>major</u>
------------	--------	------	--------------

Example (Continue)

- Make sure you have the correct keys for each relation (Below is the **final solution of mapping**)

Person

<u>SSN</u>	name	Birth_date	sex	address
------------	------	------------	-----	---------

Employee

<u>SSN</u>	project	course	Percent_time	TA_Flag	RA_Flag	rank	position	salary	etype
------------	---------	--------	--------------	---------	---------	------	----------	--------	-------

Alumnus

Alumnus_Degrees

<u>SSN</u>

<u>SSN</u>	degree	year	<u>major</u>
------------	--------	------	--------------

Student

<u>SSN</u>	MajorDep	DegreeProgram	class	Grad_Flag	Undergrad_Flag	studAssistantFalg
------------	----------	---------------	-------	-----------	----------------	-------------------

Part 5

Relation Constraints

- Characteristics of Relations
- Types of Keys
- Relation Integrity Constraints
- Violations to Relation Integrity Constraints
- Actions Performed after Violations
- Examples

Relation

A relation looks like a table of values

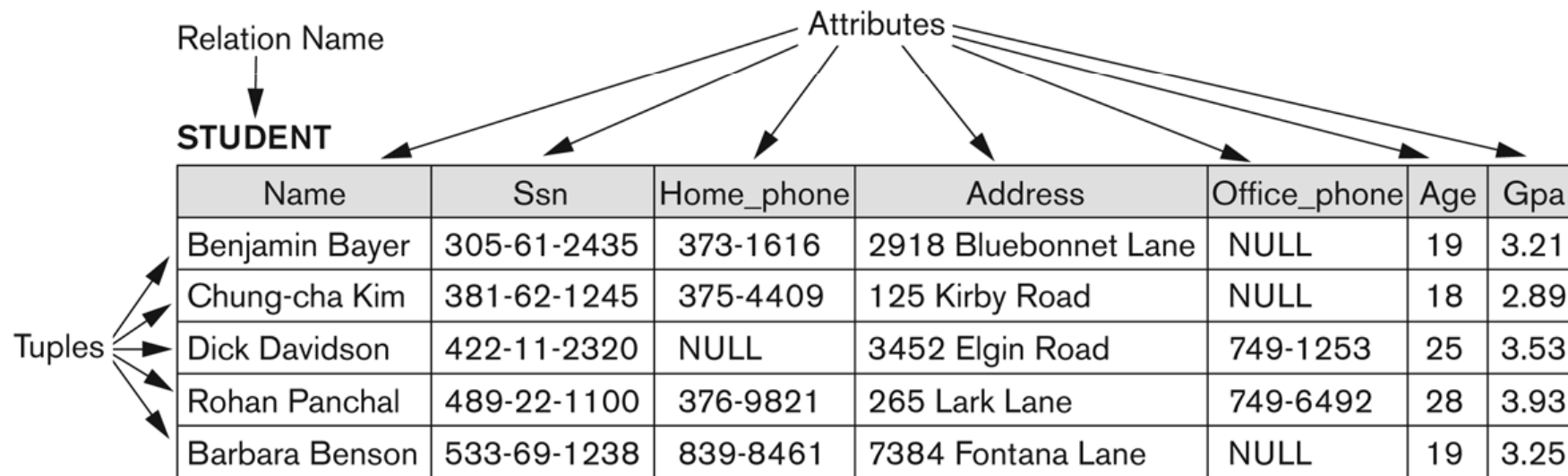


Figure 5.1

The attributes and tuples of a relation STUDENT.

Terms

<u>Informal Terms</u>		<u>Formal Terms</u>
Table		Relation
Column Header		Attribute
All possible Column Values		Domain
Row		Tuple
Table Definition		Schema of a Relation
Populated Table		State of the Relation

Characteristics of Relations

- Ordering of tuples is not important
 - For example, It is OK if you switch the position of tuple 1 and tuple 5.
- Ordering of attributes(columns) is not important
 - For example, you can switch the entire column 2 with the entire column 1.
 - Switching has to be applied for the whole column.
- Values of attributes should be indivisible. This is why we put only 'simple' attributes in relations.

Types of Keys

- Key: Should be minimal
 - Cannot be divided into smaller parts that work as a key for the same relation.
 - Examples:
 - SSN: is a key for employee.
 - studentID: is a key for student.
 - SectionID,CourseID,semester,year is a key for section.
- Super Key: A minimal key + Attribute(s)
 - It works as a key for the relation, but it is not minimal.
 - Any key is a super key.
 - Examples:
 - SSN,name is a super key for employee.
 - studentID,address is a super key for student.

Types of Keys

- **Candidate Key**: When a relation can have several different keys, each of them is called a 'Candidate Key'.
 - Relation 'Student' has 2 candidate keys, studentID and nationalNumber.
- **Primary Key**: If a relation has several candidate keys, one of them is chosen as 'Primary Key'.
 - studentID can be chosen as primary key for relation 'Student'.
- **Secondary(Alternative) Key**: If a relation has several candidate keys, all of them except the primary key can be chosen as 'Alternative Keys'.
 - nationalNumber can be chosen as a secondary key for relation 'Student'.

Types of Keys

- **Artificial (Surrogate) key**: In case no attribute(s) in a relation work as a key, we use a **sequential number** attribute that gives each record a unique number.
- This sequential number attribute is called an artificial or surrogate key.

Note

- In General: Try to make the primary key as small as possible (fewer attributes). Why?
 - Simpler Design.
 - More efficient. DBMS does not have to keep a lot of information related to primary keys and indexes.

Relational Integrity Constraints

- These constraints must hold on relations at all times:
 - Key Constraints
 - Entity Integrity Constraints
 - Referential Integrity Constraints
- Other Constraints:
 - Domain Constraint
 - Semantic Integrity Constraint

Key Constraints

- No two tuples in a relation has the same key value.

Entity Integrity Constraints

- A key of an entity in a relation (record) cannot have the value 'null'.

Referential Integrity Constraint

- A foreign key value can have “only” one of the following
 - A value that exists in the primary key it is connected to.
 - A ‘null’ value.
- Referential Integrity: making sure that the above constraint holds all the time.
- It is the responsibility of the DBMS to make sure Referential Integrity holds at all times.

Domain Constraint

- Domain Constraint: It is an implicit constraint which simply says that the value of an attribute must fall within the domain of the attribute.
- For example, gpa is a number and its value is in the range >0 and <100 .

Semantic Integrity Constraint

- It is based on the semantics of the DB.
- It cannot be expressed by a model.
- Example:
 - A student can register for a max of 18 hours in a semester.
 - The max number of hours an employee can work on a ‘all’ projects is 40 hours.
- In SQL, they are represented as ‘Triggers’ and ‘Assertions’.

Violations To Relation Integrity Constraints

- Insert, update, and delete operations can cause violations to the previously mentioned constraints.
- An 'Action' has to be taken in case of a violation.

Actions Performed After Violations

- **Reject**: Tell the user that the operation cannot be done (rejected).
- **Cascade**: Propagate the operation to other tuples and tables.
- **Set Null**: Change value of certain attributes to null.
- **Execute the operation** that caused violation and inform the user about the violation.
- **Execute a user specific routine** after violation.

Examples of Violations

Employee

<u>SSN</u>	Phone_Number
1	515A245

Domain Violation: Phone number should include only digits

Employee

<u>SSN</u>	Phone_Number
1	5157364332
1	3432425144

Key Violation: Both records have same primary key value

Employee

<u>SSN</u>	Phone_Number	dno
1	5157364332	1
4	3432425144	2

Department

<u>Dnumber</u>	dname
1	HR

* Referential Integrity Violation: Dno is a foreign key which references the primary key dnumber.

* In record 2, dno=2 but it does not exist in dnumber.

Employee

<u>SSN</u>	Phone_Number	dno
1	5157364332	1
null	3432425144	2

Entity Integrity Violation: primary key value in second record equals null.

Example of Actions Performed After Violation

Foreign Key / Primary Key Relationship

Option 1 (Reject): Update of dnumber in record 2 from Department is rejected because dno is a foreign key in Employee that references dnumber.

Employee

<u>SSN</u>	Phone_Number	dno
1	5157364332	1
5	3432425144	2

Department

<u>Dnumber</u>	dname
1	HR
2	QA

Option 2 (Cascade): Update is accepted but we have to update record 2 from employee as well, we set its dno to 3.

Employee

<u>SSN</u>	Phone_Number	dno
1	5157364332	1
5	3432425144	2 3

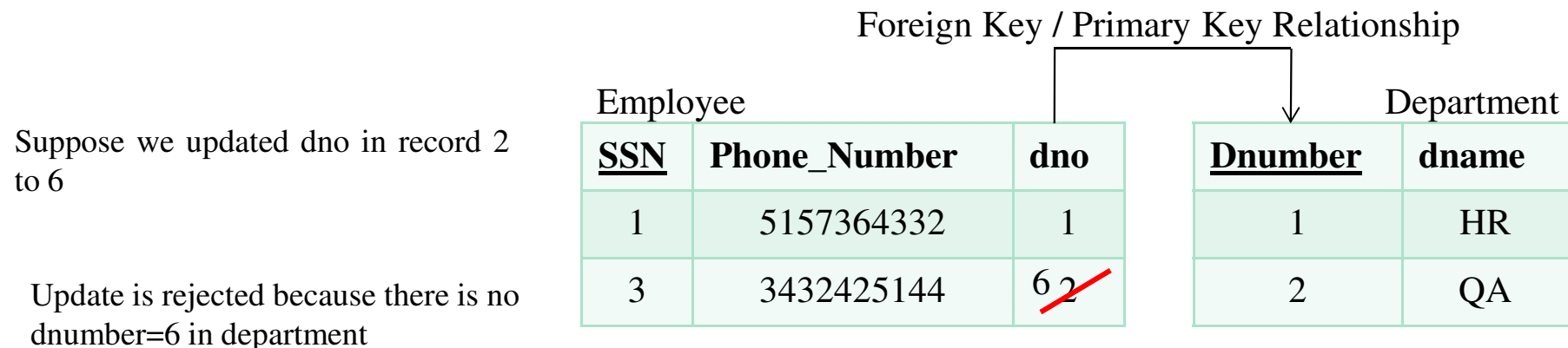
Suppose we want to update Record 2 in Relation Department such that we set dnumber to 3. (referential integrity violation)

Option 3 (set null): Update is accepted but we have to set dno in record 2 from employee to null

Employee

<u>SSN</u>	Phone_Number	dno
1	5157364332	1
5	3432425144	null

Example of Actions Performed After Violation



It is not recommended to use cascade or set null options here. Why?

1- Most likely, the update from dno=2 to dno=6 is a result of a mistake in data entry. So, it is logically to change the original department number in table department.

2- Assume option cascade is used. Then, if we have 100 employees who work in dno = 2, then we need to change all their dno to 6. Many unnecessary updates because most likely the initial update operation is a mistake in data entry

Part 6

Relational Algebra

- Relational Algebra
- Unary Relational Algebra Operators
- Binary Relational Algebra Operators
- Recursive Closure Operation
- Relational Algebra Expression
- Single Statement Expression vs Multiple Statement Expression
- Query Tree
- Query Efficiency
- Relational Algebra Examples

Prepared By: Dr. Osama Al-Haj Hassan

Reference to: Fundamentals of Database Systems , Ramez Elmasri and Shamkant B. Navathe, Fifth Edition.

Relational Algebra

- **Relational Algebra** is a set of operators used to retrieve (query) data from relations.
- When you combine several relational operators in one expression, then you call it **Relational Algebra Expression**.
- Relational Algebra was first introduced by “**Alkhwarismi**”.

Relational Algebra

- The input to a relational algebra operator is relations and the output of a relational algebra operator is a relation (This is why algebra operators are closed).
- Relational algebra operators do not change the content of a relation.

Unary Relational Operators

- They take one relation as input and produces one relation as output.
- Unary Operators are:
 - Select
 - Project
 - Rename

Unary Relational Operators (Select)

- Select Operator: Retrieves all tuples(records)(rows) that satisfy a certain condition(predicate) from a relation.
- This operator works on 'records'.
- Its symbol is sigma σ
- General Form: σ Relation
condition

Unary Relational Operators (Select)

Student

<u>SSN</u>	name	address
1	Kamal	Amman
2	Sara	Irbed
3	Majed	Karaq
4	Lama	Amman
5	Ali	Aqaba

σ Student
address='Amman'

<u>SSN</u>	name	address
1	Kamal	Amman
4	Lama	Amman

Select operator work as a **filter** operator

Unary Relational Operators (Project)

- Project Operator: Retrieves the specified columns from a relation.
- Its symbol is Π
- Duplicates among retrieved data are eliminated.
- This operator works on 'columns'.
- General Format: Π Relation
column_names

Unary Relational Operators (Project)

Student

<u>SSN</u>	name	address	gpa
1	Kamal	Amman	90
2	Sara	Irbed	80
3	Majed	Karaq	67
4	Lama	Amman	87
5	Ali	Aqaba	99

Π Student
name,address

name	address
Kamal	Amman
Sara	Irbed
Majed	Karaq
Lama	Amman
Ali	Aqaba

Unary Relational Operators (Project)

StudentGrades

<u>SSN</u>	name	courseNo	grade
1	Kamal	304	90
1	Kamal	307	80
1	Kamal	402	67
2	Sara	304	90
2	Sara	307	99

Π StudentGrades
courseNo,grade

courseNo	grade
304	90
307	80
402	67
307	99

Notice that (340,90) appeared **twice** in relation “StudentGrades”. But it appeared only **once** in the output. (**Duplicates are removed**).

Unary Relational Operators (Rename)

- Rename Operator: Rename a relation, attributes in relation, or both.
- Note: The original relation is not changed.
- Its symbol is Rho ρ

Unary Relational Operators (Rename)

- General Format 1: ρ Relation1
Relation2 (a_1, a_2, \dots, a_n)
 - Rename relation1 to Relation2
 - And rename attributes of relation to a_1, a_2 and so on.
- General Format 2: ρ Relation1
Relation2
 - Rename relation1 to Relation2
- General Format 3: ρ Relation1
(a_1, a_2, \dots, a_n)
 - Rename attributes of relation1 to a_1, a_2 and so on.

Unary Relational Operators (Rename)

Example 1

ρ Student
AllStudents(id, stname, staddress)

Student

<u>SSN</u>	name	address
1	Kamal	Amman
2	Sara	Irbed

AllStudents

<u>id</u>	stname	staddress
1	Kamal	Amman
2	Sara	Irbed

Example 2

ρ Student
(id, stname, staddress)

Student

<u>SSN</u>	name	address
1	Kamal	Amman
2	Sara	Irbed

Student

<u>id</u>	stname	staddress
1	Kamal	Amman
2	Sara	Irbed

Binary Relational Operators

- Binary Operators (from set theory) are:
 - Union
 - Intersection
 - Set Difference
 - Cartesian Product
- They take two relations as input and produce one relation as output.

Binary Relational Operators (Union)

- Union Operator: merges tuples of two relations R1 and R2 into one relation.
- Its symbol is \cup
- Duplicate tuples in the output are **eliminated**.
- R1 and R2 has to be **union compatible** ???
- The resulting relation has the same name and attributes name of the first relation in the union expression.

Binary Relational Operators (Union)

- Union Compatibility (Type Compatibility):
 - R1 and R2 are union compatible if:
 - R1 and R2 has the same number of attributes.
 - Each pair of corresponding attributes has the same domain.

Binary Relational Operators (Union)

Student

<u>SSN</u>	name	address
1	Kamal	Amman
2	Sara	Irbed
3	Lama	Amman

ForeignStudent

<u>id</u>	stname	staddress
11	Jack	Sparo
13	Robin	Hood

Student U ForeignStudent

Student

SSN	name	address
1	Kamal	Amman
2	Sara	Irbed
3	Lama	Amman
11	Jack	Sparo
13	Robin	Hood

Binary Relational Operators (Intersection)

- Intersection Operator: extracts tuples that exists in both R1 and R2.
- Its symbol is \cap
- Duplicate tuples in the output are **eliminated**.
- R1 and R2 has to be **union compatible** ???
- The resulting relation has the same name and attributes name of the first relation in the intersection expression.

Binary Relational Operators (Intersection)

Find computer science students who are also distinguished students.

ComputerStudent

<u>SSN</u>	name	address
1	Kamal	Amman
2	Sara	Irbed
3	Lama	Amman

DistinguishedStudent

<u>id</u>	stname	staddress
2	Sara	Irbed
3	Lama	Amman
6	Ahmad	Irbed

ComputerStudent \cap DistinguishedStudent

ComputerStudent

SSN	name	address
2	Sara	Irbed
3	Lama	Amman

Binary Relational Operators

(Set Difference)

- Set Difference Operator: $R1 - R2$ means, extract tuples that exist in $R1$ but not in $R2$.
- Its symbol is $-$
- $R1$ and $R2$ has to be union compatible ???
- The resulting relation has the same name and attributes name of the first relation in the intersection expression.

Binary Relational Operators

(Set Difference)

Find computer science students who are **not** distinguished students.

ComputerStudent

<u>SSN</u>	name	address
1	Kamal	Amman
2	Sara	Irbed
3	Lama	Amman

DistinguishedStudent

<u>id</u>	stname	staddress
2	Sara	Irbed
3	Lama	Amman
6	Ahmad	Irbed

ComputerStudent - DistinguishedStudent

ComputerStudent

SSN	name	address
1	Kamal	Amman

Binary Relational Operators

(Cartesian Product)

- Cartesian Product (Cross Product): The operation of pairing records of two relations.
- The symbol for Cartesian Product is “X”
- (Relation1 X Relation2) is performed as follows
 - Number of attributes in the result of Cartesian Product is the sum of number of attributes in “Relation1” and “Relation2”.
 - Number of tuples in the result of Cartesian Product is the multiplication of number of tuples in “Relation1” and “Relation2”
 - Take each tuple from “Relation1” and pair it with each tuple in “Relation2”

A	B	C		X	Y	Z		A	B	C	X	Y	Z
A1	B1	C1	X	X1	Y1	Z1	=	A1	B1	C1	X1	Y1	Z1
A2	B2	C2		X2	Y2	Z2		A1	B1	C1	X2	Y2	Z2
				X3	Y3	Z3		A1	B1	C1	X3	Y3	Z3
								A2	B2	C2	X1	Y1	Z1
								A2	B2	C2	X2	Y2	Z2
								A2	B2	C2	X3	Y3	Z3

2 tuples 3 tuples 6 tuples

Binary Relational Operators

(Cartesian Product)

- Cartesian product produces **logically incorrect** records for tables which have relationships.
- Example: Employee X Dependent

Employee

<u>Ssn</u>	name	salary	Bdate	superssn	dno
1	Ahmad	356	12-JAN-80	2	2
2	Salem	500	30-APR-85	3	1
3	Maya	333	10-FEB-70	4	3
4	Sara	405	06-JAN-90	5	2
5	Malik	734	15-MAR-86	1	1

Dependent

<u>essn</u>	<u>dep_name</u>	relationship
1	Sofi	Daughter
1	Basem	Son
3	Bana	Daughter

Employee X Dependent

Ssn	name	salary	Bdate	superssn	dno	essn	dep_name	relationship
1	Ahmad	356	12-JAN-80	2	2	1	Sofi	Daughter
1	Ahmad	356	12-JAN-80	2	2	1	Basem	Son
1	Ahmad	356	12-JAN-80	2	2	3	Bana	Daughter
2	Salem	500	30-APR-85	3	1	1	Sofi	Daughter
2	Salem	500	30-APR-85	3	1	1	Basem	Son
2	Salem	500	30-APR-85	3	1	3	Bana	Daughter
3	Maya	333	10-FEB-70	4	3	1	Sofi	Daughter
3	Maya	333	10-FEB-70	4	3	1	Basem	Son
3	Maya	333	10-FEB-70	4	3	3	Bana	Daughter
4	Sara	405	06-JAN-90	5	2	1	Sofi	Daughter
4	Sara	405	06-JAN-90	5	2	1	Basem	Son
4	Sara	405	06-JAN-90	5	2	3	Bana	Daughter
5	Malik	734	15-MAR-86	1	1	1	Sofi	Daughter
5	Malik	734	15-MAR-86	1	1	1	Basem	Son
5	Malik	734	15-MAR-86	1	1	3	Bana	Daughter



Some tuples are **not logically correct**.

Binary Relational Operators

(Join)

- join = Cartesian product + predicate
- To fix the previous Cartesian product problem, we make Cartesian Product based on a predicate that relates primary key to foreign key.
- Example:

Employee ⋈ Dependent
 Employee.ssn = Dependent.essn
- In this case, the operation is called “Join” and the predicate called “join predicate” or “join condition”.
- Because of that, the incorrect records will be deleted. They will be deleted because (ssn is not equal to essn)

Employee ⋈ Dependent
 Employee.ssn = Dependent.essn

Ssn	name	salary	Bdate	superssn	dno	essn	dep_name	relationship
1	Ahmad	356	12-JAN-80	2	2	1	Sofi	Daughter
1	Ahmad	356	12-JAN-80	2	2	1	Basem	Son
3	Maya	333	10-FEB-70	4	3	3	Bana	Daughter

- Now, the result is **logically correct**

Binary Relational Operators (Join)

- Find information related to departments and their managers.

Employee


<u>Ssn</u>	name	salary	Bdate	superssn	dno
1	Ahmad	356	12-JAN-80	2	2
2	Salem	500	30-APR-85	3	1
3	Maya	333	10-FEB-70	4	3
4	Sara	405	06-JAN-90	5	2
5	Malik	734	15-MAR-86	1	1

Department

<u>dnumber</u>	dname	mgrssn
1	CS	2
2	CIS	1
3	SE	4

Solution is in next slide

Binary Relational Operators (Join)

Employee  Department
ssn = mgrssn

Ssn	name	salary	Bdate	superssn	dno	dnumber	dname	mgrssn
1	Ahmad	356	12-JAN-80	2	2	2	CIS	1
2	Salem	500	30-APR-85	3	1	1	CS	2
4	Sara	405	06-JAN-90	5	2	3	SE	4

Binary Relational Operators (Natural Join)

- It is used to implicitly use a join condition that performs equality between attributes which has the **same name**.
- In the result, attributes with the same name appear **only once**.
- Its symbol is *

Binary Relational Operators (Natural Join)

Find information of animals and the cages they live in.

Animal

<u>id</u>	name	type	cageNum
1	Semo	Cat	1
2	Fofo	Dog	3
3	Spiky	Dog	3
4	Fishi	Fish	2
5	Shippo	Cat	1

Cage

<u>cageNum</u>	location	size
1	East-loc1	4
2	West-loc2	3
3	East-loc2	1

Solution is in next slide


Binary Relational Operators (Natural Join)

- Join should be based on equality between primary key and foreign key.
- In this case, primary key and foreign key have the same name (cageNum)

Animal * Cage

id	name	type	cageNum	location	size
1	Semo	Cat	1	East-loc1	4
2	Fofo	Dog	3	East-loc2	1
3	Spiky	Dog	3	East-loc2	1
4	Fishi	Fish	2	West-loc2	3
5	Shippo	Cat	1	East-loc1	4

Equivalent To:

Animal  Cage
Animal.cageNum = cage.cageNum

- Notice that cageNum attribute appeared **only once** in the join result.

Binary Relational Operators (Natural Join)

R1

a1	a2	a3	a4
1	A1	B1	C1
2	A2	B2	C2
3	A3	B3	C3

R2

a2	b1	a4
A2	Z1	C2
A1	Z23	C8
A6	Z5	C1

R1 * R2

a1	a2	a3	a4	B1
2	A2	B2	C2	z1

Equivalent To:

$R1 \bowtie R2$
 $R1.a2=R2.a2$ and $R1.a4=R2.a4$

Complete Set of Relational Operators

- Select, Project, Rename, Union, Intersection, Set Difference, and Cartesian Product are called “complete” because:
 - If you have a given relational algebra expression, you can rewrite it using a combination of the previous operators.
- Example:

$$\text{Employee} \bowtie_{\text{ssn} = \text{mgrssn}} \text{Department} = \sigma_{\text{Ssn}=\text{mgrssn}} (\text{Employee} \times \text{Department})$$

Binary Relational Operators (Division)

- It is applied to two relations R1, R2.
- Example: $R1 \div R2$
- It means: Finds records in R1 that is **related to all** records in R2
 - Related is measured based on attributes with same name.

Binary Relational Operators (Division)

SSN	name	Hoby
1	Kamal	Swimming
1	Kamal	Tennis
1	Kamal	Ping-Pong
2	Ahmad	Swimming
3	Salah	Ping-Pong
3	Salah	Swimming
3	Salah	Tennis
4	Sara	Tennis

÷

Hoby
Swimming
Tennis
Ping-Pong

=

SSN	name
1	Kamal
3	Salah

Meaning: It means who are the employees who likes **all hobbies**

Binary Relational Operators (Division)

SSN	name	Hobby	Level		Hobby	Level		SSN	name
1	Kamal	Swimming	Professional	÷	Swimming	Beginner	=	3	Salah
1	Kamal	Tennis	Professional		Tennis	Professional			
1	Kamal	Ping-Pong	Professional		Ping-Pong	Professional			
2	Ahmad	Swimming	Intermediate						
3	Salah	Ping-Pong	Professional						
3	Salah	Swimming	Beginner						
3	Salah	Tennis	Professional						
4	Sara	Tennis	Beginner						

Meaning: Who are the employees who are:

- professional in ping-pong and tennis. And
- they are beginners in swimming

Binary Relational Operators

(Aggregate Functions and Grouping)

- Are functions which operate on groups of tuples.
- Tuples are classified into groups based on the value of a given attribute(s).
- Popular examples: count, sum, avg, max, min.

Binary Relational Operators

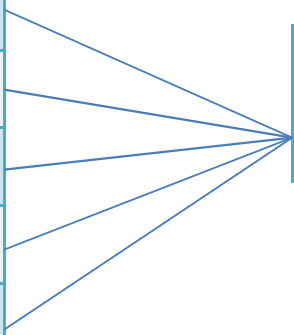
(Aggregate Functions and Grouping)

- Find count of all employees.
- Here, the group we care about is “All Employees”

Employee

<u>Ssn</u>	name	salary	Bdate	superssn	dno
1	Ahmad	356	12-JAN-80	2	2
2	Salem	500	30-APR-85	3	1
3	Maya	333	10-FEB-70	4	3
4	Sara	405	06-JAN-90	5	2
5	Malik	734	15-MAR-86	1	1

\mathcal{F} Employee
count ssn



count
5

Binary Relational Operators

(Aggregate Functions and Grouping)

- Find count of employees who work in each department.
- Here, the groups we care about are all employees working in same department.
- In this example, we have 3 groups (Based on 3 departments) (based on dno)

Employee

<u>Ssn</u>	name	salary	Bdate	superssn	dno
1	Ahmad	356	12-JAN-80	2	2
2	Salem	500	30-APR-85	3	1
3	Maya	333	10-FEB-70	4	3
4	Sara	405	06-JAN-90	5	2
5	Malik	734	15-MAR-86	1	1

dno \mathcal{F} Employee
count ssn

dno	count
1	2
2	2
3	1

Binary Relational Operators

(Aggregate Functions and Grouping)

- Find count of employees and sum of salaries for each department
- Here, the groups we care about are all employees working in same department
- In this example, we have 3 groups (Based on 3 departments) (based on dno)

Employee

<u>Ssn</u>	name	salary	Bdate	superssn	dno
1	Ahmad	356	12-JAN-80	2	2
2	Salem	500	30-APR-85	3	1
3	Maya	333	10-FEB-70	4	3
4	Sara	405	06-JAN-90	5	2
5	Malik	734	15-MAR-86	1	1

dno \mathcal{F} Employee
count ssn, sum salary

Dno	count	sum
1	2	1234
2	2	761
3	1	333

Binary Relational Operators

(Aggregate Functions and Grouping)

- Find count of animals of the same type and live in the same cage.
- Here, the groups we care about are all animals living in same cage and have same type
- In this example, we have 4 groups (Based on type and cageNum)

Animal

<u>id</u>	name	type	cageNum
1	Semo	Cat	4
2	Fofo	Dog	1
3	Spiky	Dog	1
4	Fishi	Fish	3
5	Shippo	Cat	4
6	zefo	Dog	6

type,cageNum \mathcal{F} Animal
count id

type	cageNum	count
cat	4	2
dog	1	2
fish	3	1
dog	6	1

Relational Algebra Expressions

- The power of relational algebra comes when we combine multiple operators into one relational algebra expression.
- Example: Find ssn,name of employees who live in Amman.

Employee

<u>Ssn</u>	name	salary	Address
1	Ahmad	356	Amman
2	Salem	500	Salt
3	Maya	333	Amman
4	Sara	405	Irbed
5	Malik	734	Qarak

Select Operator

Project Operator

$\Pi \left(\sigma \left(\text{Employee} \right)_{\text{Address}='Amman'} \right)_{\text{Ssn,name}}$

Ssn	name
1	Ahmad
3	Maya

Single Statement Expression

VS

Multiple Statements Expression

- A relational algebra expression can be written as one single statement

$$\pi_{\text{Ssn,name}} \left(\sigma_{\text{Ssn=mgrssn}} \left(\text{Employee} \times \text{Department} \right) \right)$$

- The same relational algebra expression can be written as multiple statements

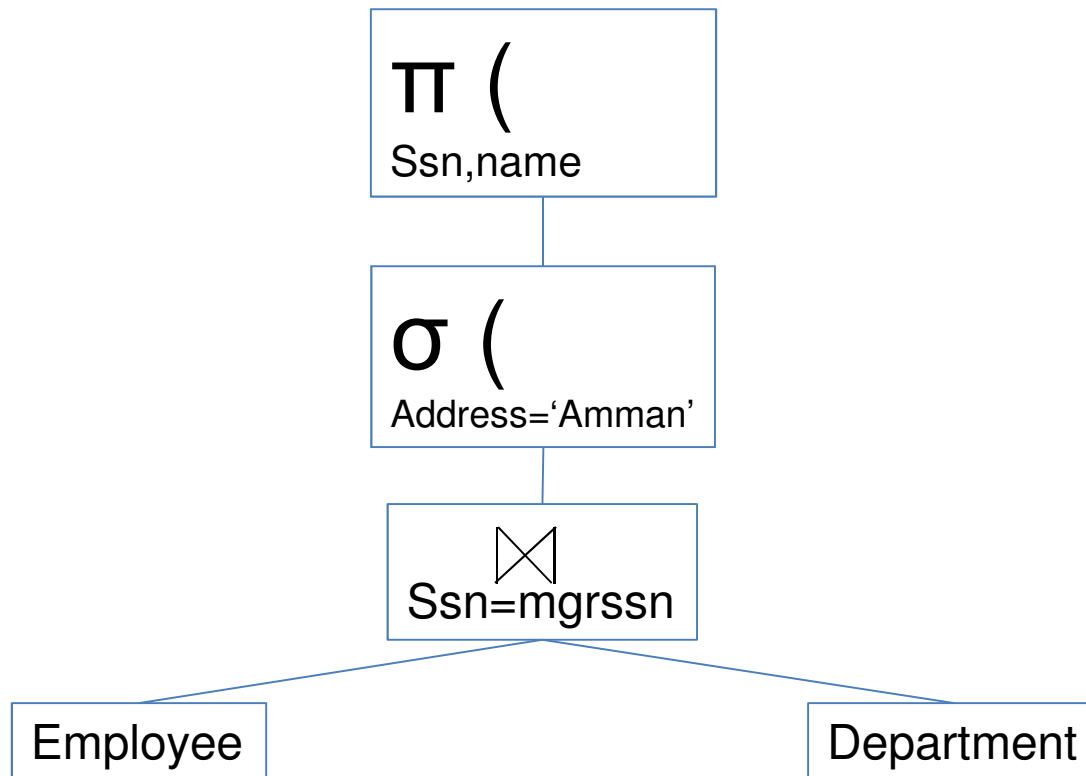
EmpDep	←	Employee X Department
CEmpDep	←	$\sigma_{\text{Ssn=mgrssn}} (\text{EmpDep})$
FinalRes	←	$\pi_{\text{Ssn,name}} (\text{CEmpDep})$

Query Tree

- Each relational algebra expression can be represented as a tree.
- The operations that are executed first are placed in the bottom of tree (Leafs).
- The operations that are executed after that are placed in the middle of the tree.
- The operation that is executed last is placed on the top of the tree (Root).

Query Tree

Π (σ (Employee \bowtie Department)
Address='Amman' Ssn=mgrssn)
Ssn,name



Query Efficiency

- One query can be designed in multiple ways.
- Which query design is the best? (More Efficient to execute)
- Example: Find employees who live in 'Amman' and work in 'CS' department.

Design 1

σ (Employee \bowtie Department
dno=dnumber)

Address='Amman'
and dname='CS'

Design 2

$(\sigma_{\text{Address='Amman'}} (\text{Employee})) \bowtie (\sigma_{\text{dname='CS'}} (\text{Department}))$

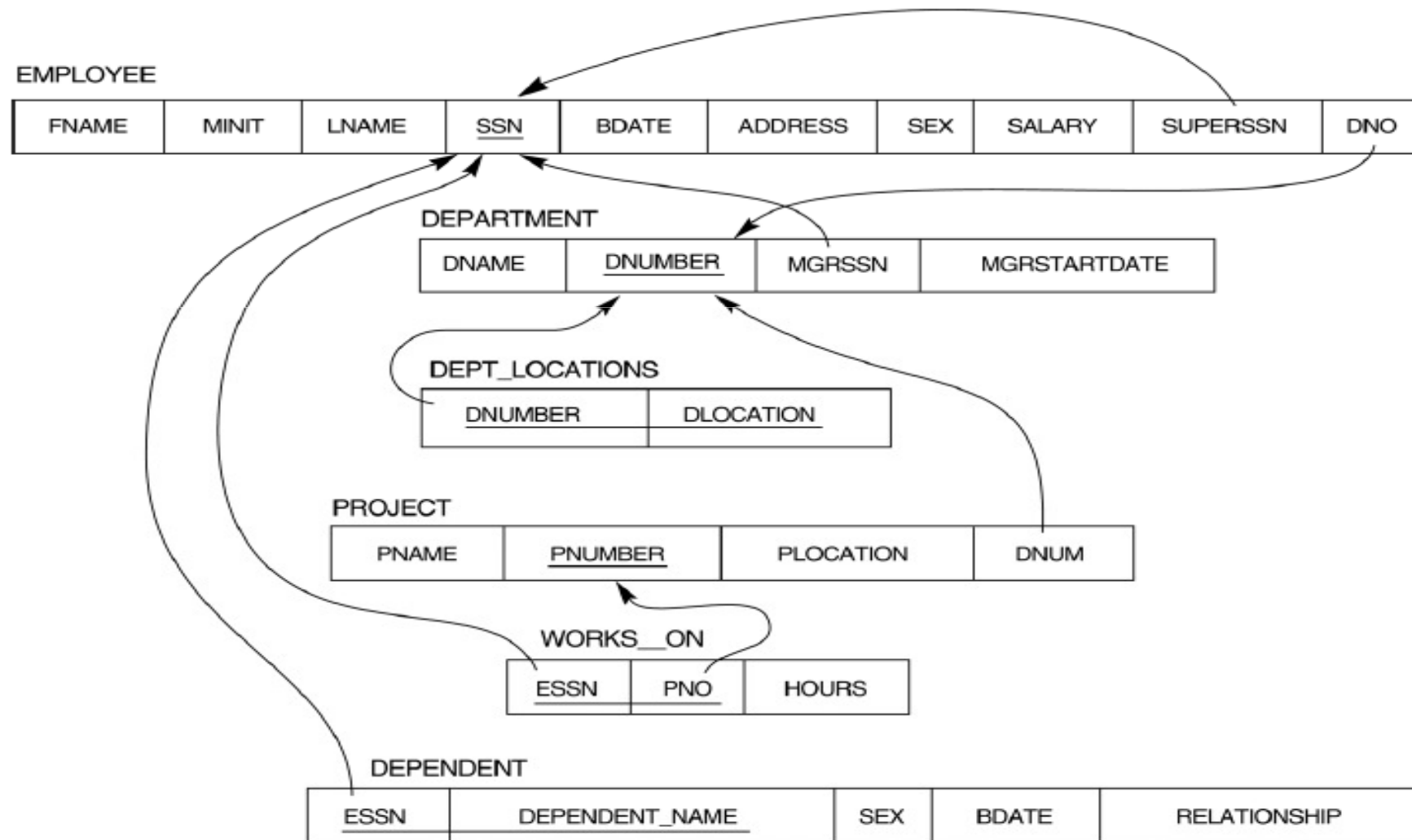
dno=dnumber

Which design is more efficient? (solution is on next slide)

Query Efficiency

- Assume relation 'Employee' has 100 employees. 60 of them live in 'Amman'
- Assume relation 'Department' has 10 departments. 1 of them is 'CS' department
- In design 1:
 - First, Join is executed, number of steps = $100 * 10 = 1000$ step
 - Second, Select is executed on join result, number of steps = 1000 step
 - **Total number of steps = $1000 + 1000 = 2000$ step**
- In design 2:
 - First, select on employee is executed, number of steps = 1000 step, the output is 60 employees.
 - Second, select on department is executed, number of steps = 10 step, the output is 1 department
 - Third, join on result of two selects is executed, number of steps = $60 * 1 = 60$
 - **Total number of steps = $1000 + 10 + 60 = 1070$ step (So, design 2 is better)**

All Following Examples are based on Company DB



Relational Algebra Examples

(Example 1)

- Find employee name, ssn who has salary > 200 and supervised by employee 10
 - employee Name, ssn, salary, supervisor number exist in employee relation

$$\pi_{\text{Ssn,name}} \left(\sigma_{\text{Employee}} \begin{array}{l} \text{Salary} > 200 \text{ and } \text{superssn} = 10 \end{array} \right)$$

Relational Algebra Examples

(Example 2)

- For all department, find number, name, and locations.
 - Department number and name exist in relation department
 - Department location exist in relation dept_location

$$\pi_{\text{dnumber,dname,dlocation}} \left(\text{Department} * \text{Dept_location} \right)$$

Relational Algebra Examples

(Example 3)

- Find department number and count of projects under control of that department.
 - Department number and projects it controls exist in relation “Project”
 - Department number and project number also exist in other tables, but you need to choose the most efficient and easier solution.

Dnum \mathcal{F} Project
count(pnumber)

Relational Algebra Examples

(Example 4)

- Find name of employee and count of projects he is working on.
 - Employee name exists in relation “Employee”
 - The projects an employee works on exist in relation “works_on”

$$\text{name } \mathcal{F}_{\text{Count(pno)}} \left(\begin{array}{c} \text{Employee} \bowtie \text{works_on} \\ \text{ssn=essn} \end{array} \right)$$

Relational Algebra Examples (Example 5)

- Find name of employee and count of projects he is working on in each department
 - Employee name exists in relation “Employee”
 - The projects an employee works on exist in relation “works_on”
 - The department that controls project exist in relation “Project”

Employees and projects
they work on

Employees and projects
they work on with department
Number that controls project

Count number of project an
Employee work on in each
department

Emp_Proj \leftarrow Employee $\bowtie_{\text{ssn=essn}}$ works_on

Emp_Proj2 \leftarrow Emp_Proj $\bowtie_{\text{pno=pnumber}}$ Project

Final_Result \leftarrow Name,dnum $\mathcal{F}_{\text{Count(pno)}}(\text{emp_Proj2})$

Relational Algebra Examples

(Example 5)

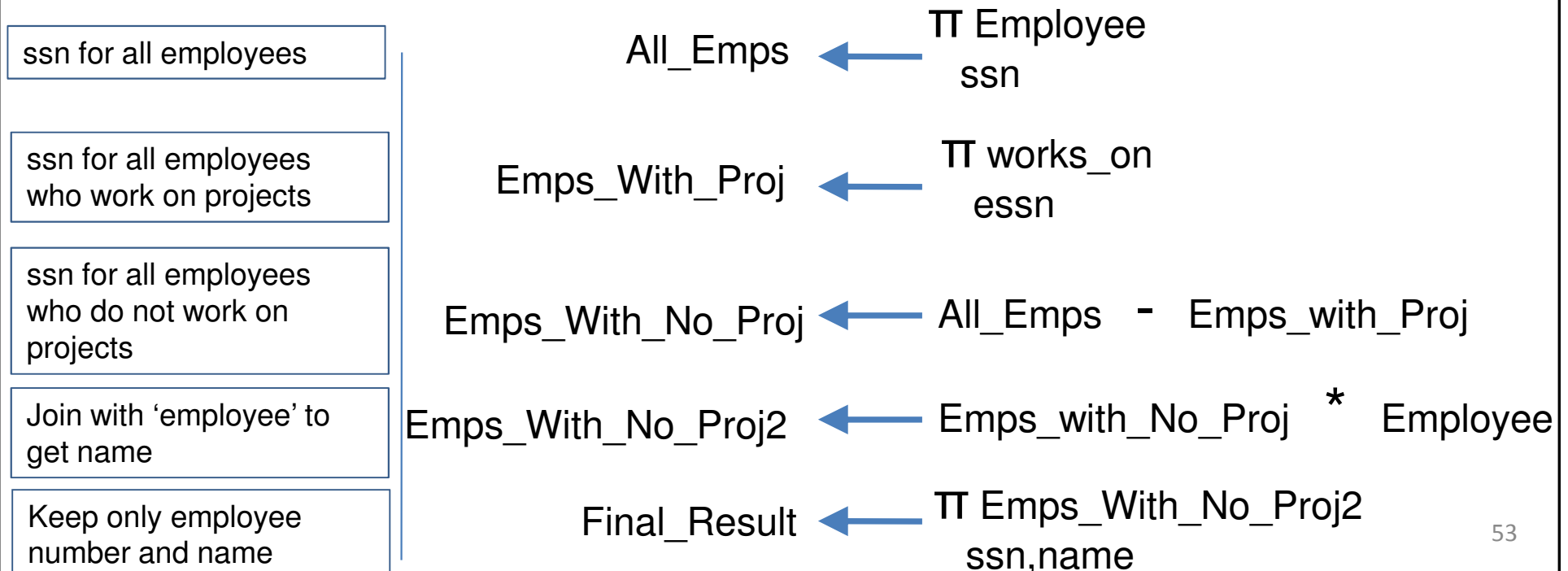
- Solution in one statement

$$\begin{array}{l} \text{Name,dnum} \\ \text{Count(pno)} \end{array} \mathcal{F} \left(\left(\begin{array}{c} \text{Employee} \\ \text{ssn=essn} \end{array} \bowtie \begin{array}{c} \text{works_on} \\ \text{pno=pnumber} \end{array} \right) \bowtie \begin{array}{c} \text{Project} \\ \text{pno=pnumber} \end{array} \right)$$

Relational Algebra Examples

(Example 6)

- Find name,ssn of employees who do not work on any project
 - All employees exist in relation “Employee”
 - Employees working on projects exist in relation “works_on”
 - Employee name exist in relation “Project”



Relational Algebra Examples

(Example 6)

- Solution in one statement

$$\pi_{\text{ssn, name}} \left(\left(\pi_{\text{ssn}} \text{ Employee} \right) - \left(\pi_{\text{essn}} \text{ works_on} \right) \right) * \text{Employee}$$

Relational Algebra Examples

(Example 7)

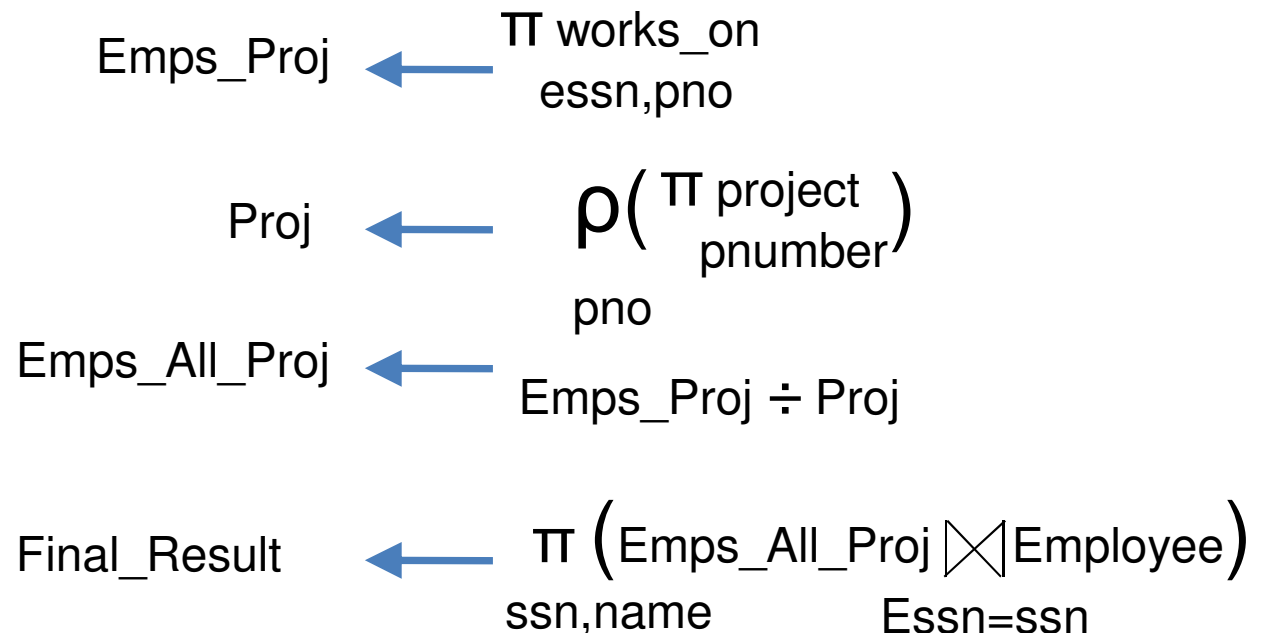
- Find name,ssn of employees who work on all project
 - All employees exist in relation “Employee”
 - Employees working on projects exist in relation “works_on”
 - Employee name exist in relation “Project”

ssn for employees and projects they work on

Numbers of all project, and rename it to pno

ssn for employees who work on all projects

Join with ‘employee’ to get employee name and keep only employee number and name



Relational Algebra Examples

(Example 7)

- Solution in one statement

$$\pi_{\text{ssn, name}} \left(\left(\pi_{\text{works_on}} \right) \div \left(\rho_{\text{pno pnumber}} \left(\pi_{\text{project}} \right) \right) \right) \bowtie_{\text{essn=ssn}} \text{Employee}$$

Relational Algebra Examples

(Example 8)

- Find name, number of departments who have locations in Amman but not in Irbed.
 - Departments name and number exist in relation “Department”
 - Department location exists in relation “dept_location”

Departments
with locations
in amman

Dept_Loc_Amman $\leftarrow \sigma_{\text{dlocation}='Amman'}(\text{department} * \text{dept_location})$

Departments
with locations
in Irbed

Dept_Loc_Irbed $\leftarrow \sigma_{\text{dlocation}='Irbed'}(\text{department} * \text{dept_location})$

Departments with
locations in amman but
not in Irbed, keep only
department number
and name

Final_Result $\leftarrow \pi_{\text{dname}, \text{dnumber}}(\text{Dept_Loc_Amman} - \text{Dept_Loc_Irbed})$

Relational Algebra Examples

(Example 8)

- Solution in one statement

$$\pi_{\text{dname,dnumber}} \left(\left(\sigma_{\text{Department} * \text{dept_location}} \right) - \left(\sigma_{\text{department} * \text{dept_location}} \right) \right)$$

Relational Algebra Examples

(Example 9)

- Find ssn of employees who have more than 3 dependents.
 - Ssn of employees who have dependents exist in relation “dependent”

ssn and number of dependents for each employee

Rename ssn as essn and rename count as cnt

Keep only those records (employees) who have number of dependents > 3

Keep only values of essn

$\text{Emp_Num_Of_Dep} \leftarrow \text{essn} \mathcal{F}_{\text{Count(essn)}}(\text{Dependent})$

$\text{Emp_Num_Of_Dep2} \leftarrow \rho_{\text{essn,cnt}}(\text{emp_Num_Of_Dep})$

$\text{Emp_Num_Of_Dep3} \leftarrow \sigma_{\text{Cnt} > 3}(\text{Emp_Num_Of_Dep2})$

$\text{Final_Result} \leftarrow \pi_{\text{essn}}(\text{Emp_Num_Of_Dep3})$

Relational Algebra Examples (Example 9)

- Solution in one statement

$$\pi_{\text{essn}} \left(\sigma_{\text{Cnt} > 3} \left(\rho_{\text{essn}, \text{cnt}} \left(\mathcal{F}_{\text{Count}(\text{essn})}(\text{Dependent}) \right) \right) \right)$$

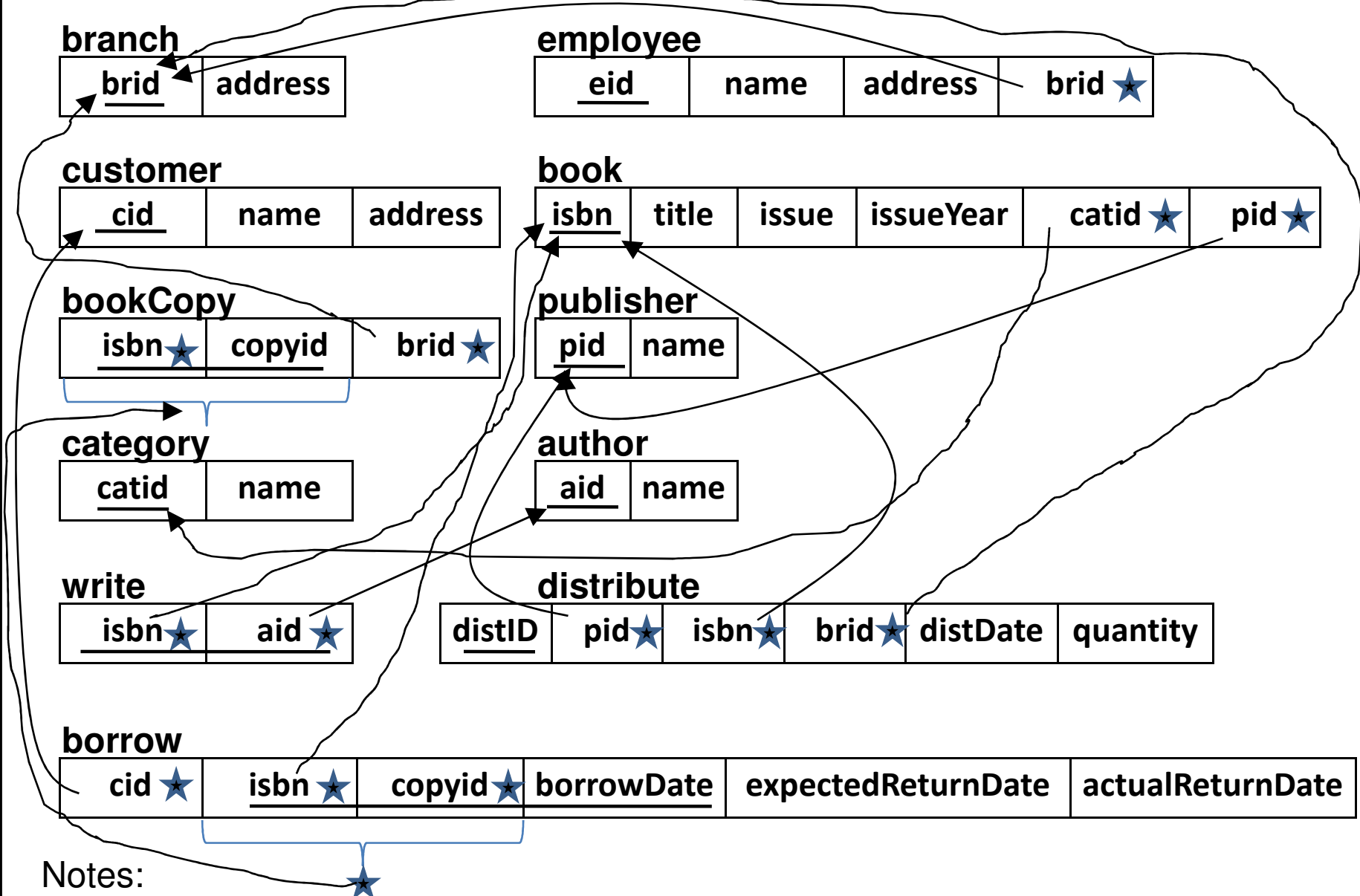
Part 7

Practice on Select Statement

Prepared By: Dr. Osama Al-Haj Hassan

Reference to: Fundamentals of Database Systems , Ramez Elmasri and Shamkant B. Navathe, Fifth Edition.

Library Database Schema



Question 1

- Find customers who live in Amman

Question 2

- Find customer name who borrowed book with isbn 350

Question 3

- Find book title that belong to category “science”

Question 4

- Find book title written by author “Ahmad”

Question 5

- Find book title borrowed by customer “sara”

Question 6

- Find sum of salaries for employees working in branch 2

Question 7

- Find count of copies of book 325

Question 8

- Find book title and count of copies for it.
Order result in descending order based on title.

Question 9

- Find book title and how many times it was borrowed

Question 10

- Find book title and copyID and how many times the copy was borrowed

Question 11

- Find book id and title for books that are currently borrowed.

Question 12

- Find book id and title for books that are currently borrowed and past due return date.

Question 13

- Suppose the fine for each late day is JD10.
Find fine for currently borrowed books

Question 14

- Find book title written by author “ahmad” and borrowed by customer “sara”.

Question 15

- Find book titles that were borrowed more than 3 times

Question 16

- Find book titles that are never borrowed

Part 8

Relational Database Design

- Relational Database Design
 - Informal Concepts for Good DB Design
 - Functional Dependencies
 - X closure
 - Normalization (2NF, 1NF, BCNF, 3NF)

Prepared By: Dr. Osama Al-Haj Hassan

Reference to: Fundamentals of Database Systems , Ramez Elmasri and Shamkant B. Navathe, Fifth Edition.

Informal Concepts for Good DB Design

- 1) Avoid redundant attributes (repetitions) unless they are necessary.
- 2) Design a schema that does not suffer from update, insert, and delete anomalies
- 3) Avoid null values as much as possible.
- 4) Relations should satisfy “Lossless Join” condition.

Informal Concepts for Good DB Design

1) Avoid redundant attributes (repetitions) unless they are necessary

- Redundancy causes:
 - Waste of storage
 - Anomalies (see next slide)

EmployeeInfo1

<u>SSN</u>	name	address
------------	------	---------

EmployeeInfo2

<u>SSN</u>	name	Birth_date
------------	------	------------



Employee

<u>SSN</u>	name	address	Birth_date
------------	------	---------	------------



Informal Concepts for Good DB Design

- 2) Design a schema that does not suffer from update, insert, and delete anomalies
 - Redundant data leads to update, insert, delete anomalies

Update Anomaly

- Update Anomaly: Updating an attribute value leads to many updates that can be avoided.
 - Suppose Employee relation and project relation are merged together
 - Changing pname for project 1 to “Borj” will cause pname to be changed for all employees working on project 1.

Example of Update Anomaly

Emp_Proj

<u>SSN</u>	name	<u>pnumber</u>	pname	hours
1	Ahmad	1	Borj	33
1	Ahmad	2	Shaheq	24
2	Kamal	1	Borj	54
3	Sara	1	Borj	23
4	Lubna	1	Borj	56



One change caused many other changes that can be avoided

Pname is redundant here

Imagine what happens if there are 100 employees working on project 1

Employee

<u>SSN</u>	name
1	Ahmad
1	Ahmad
2	Kamal
3	Sara
4	Lubna

Project

<u>pnumber</u>	pname
1	Borj
2	Shaheq

Works_on

<u>SSN</u>	<u>pnumber</u>	hours
1	1	33
1	2	24
2	1	54
3	1	23
4	1	56



One change caused no changes at all

Insert Anomaly

- Cannot insert an entity because it is tied with another entity in the same relation
 - An example on the previous case: The primary key for Emp_Proj is SSN+pnumber. So, we cannot insert a new project unless at least one employee works on that project

Example of Insert Anomaly

Emp_Proj

<u>SSN</u>	name	<u>pnumber</u>	pname	hours
1	Ahmad	1	Abraj	33
1	Ahmad	2	Shaheq	24
2	Kamal	1	Borj	54
null	null	3	Sabeel	null



Inserting project 3 does not work because no employee is working on project 3 yet. This violates the entity integrity constraint (primary key) . (null,3) does not work as a primary key

Employee

<u>SSN</u>	name
1	Ahmad
1	Ahmad
2	Kamal

Project

<u>pnumber</u>	pname
1	Abraj
2	Shaheq
3	Sabeel

Works_on

<u>SSN</u>	<u>pnumber</u>	hours
1	1	33
1	2	24
2	1	54



Inserting project 3 works fine

Delete Anomaly

- Cannot delete an entity because it is tied with another entity in the same relation
 - An example on the previous case: The primary key for Emp_Proj is SSN+pnumber. Suppose, we delete project 1, this means that *all employees working on that project will be deleted.*
 - Example 2: Suppose only one employee is working on a specific project. Deleting that employee causes the *project to be deleted.*

Example of Delete Anomaly



Emp_Proj

<u>SSN</u>	name	<u>pnumber</u>	pname	hours
1	Ahmad	1	Abraj	33
1	Ahmad	2	Shaheq	24
2	Kamal	3	Sabeel	54

Deleting employee Kamal causes project Sabeel to be deleted

Employee

<u>SSN</u>	name
1	Ahmad
1	Ahmad
2	Kamal

Project

<u>pnumber</u>	pname
1	Abraj
2	Shaheq
3	Sabeel

Works_on

<u>SSN</u>	<u>pnumber</u>	hours
1	1	33
1	2	24
2	1	54



Deleting Kamal only causes record 3 in works_on to be deleted and this is logical

Informal Concepts for Good DB Design

3) Avoid null values as much as possible

- Null values causes waste of storage.
- If you have many null values for an attribute then it is better to move that attribute with a copy of the primary key to a separate table.
- Reasons for null values:
 - Value not applicable
 - Value does not exist
 - Value exists but it is not available now

Example of Delete Anomaly

Employee_Hobby

<u>SSN</u>	name	salary	hobby
1	Ahmad	324	Tennis
2	Samer	424	null
3	Kamal	244	null
4	Sara	234	null

This example assumes that employee can have only one hobby



Many null values. This is a waste of storage

Employee

<u>SSN</u>	name	salary
1	Ahmad	324
2	Samer	424
3	Kamal	244
4	Sara	234

Employee_Hobby

<u>SSN</u>	<u>hobby</u>
1	Tennis



No null values. This is much better design

Informal Concepts for Good DB Design

4) Relations should satisfy “Lossless Join” condition

- A “Lossless Decomposition” is decomposing a relation R into relations R_1 and R_2 such that $(R_1 * R_2)$ satisfy the following:
 - No loss or addition of tuples (In other words, all tuples have to be correct (not spurious)).
 - Preserve functional dependencies (We will discuss it later).

Example of a Lossy Decomposition

Borrow

Customer_id	Customer_name	Loan_amount	<u>loan_id</u>	Bank_name
1	Ahmad	1000	30	Arab
2	Kamal	1000	31	Arab

Decompse
based
on
loan_amount

Amount

Customer_ID	Customer_name	Loan_amount
1	Ahmad	1000
2	Kamal	1000

Loan

Loan_id	Bank_name	Loan_amount
30	Arab	1000
31	Arab	1000

- Lossy decomposition because the result of natural join did not return the original relation
- Lossy decomposition because decomposition is not based on primary key (loan_id)

Borrow = Amount * Loan

Customer_id	Customer_name	Loan_amount	Loan_id	Bank_name
1	Ahmad	1000	30	Arab
1	Ahmad	1000	31	Arab
2	Kamal	1000	30	Arab
2	Kamal	1000	31	Arab



Example of a Lossless Decomposition

Borrow

Customer_id	Customer_name	Loan_amount	<u>loan_id</u>	Bank_name	Decompse based on loan_id
1	Ahmad	1000	30	Arabi	
2	Kamal	1000	31	Arabi	

Amount

Customer_ID	Customer_name	Loan_id
1	Ahmad	30
2	Kamal	31

Loan

Loan_id	Bank_name	Loan_amount
30	Arabi	1000
31	Arabi	1000

- Lossless decomposition because result of ***natural*** join returned the original relation.
- Lossless decomposition happened because decomposition is based on primary key (loan_id).

Borrow = Amount * Loan

Customer_id	Customer_name	Loan_amount	Loan_id	Bank_name
1	Ahmad	1000	30	Arabi
2	Kamal	1000	31	Arabi



Functional Dependencies

- They are constraints derived from the meaning and interconnections between relations.
- Format:
 - $X_1, X_2, \dots \longrightarrow Y_1, Y_2, \dots$
- A set of attributes X functionally determines a set of attributes Y if the value of X determines a unique value for Y
- $X \longrightarrow Y$ holds if whenever two tuples have the same value for X , they *must have* the same value for Y
 - For any two tuples t_1 and t_2 in any relation instance $r(R)$: If $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$


What does a functional dependency really mean?

- Attr1,Attr2 \longrightarrow Attr3, Attr4
- If same value of (Attr1,Attr2) appears in two tuples in the same relation, this means that the value of (Attr3,Attr4) are the same in the two tuples.
- In other word:
 - Attr1,Attr2 (functionally determines) Attr3,Attr4
- Example: SSN \longrightarrow name
- Other examples:
 - SSN \longrightarrow name, salary
 - Dnumber \longrightarrow dname
 - pnumber \longrightarrow pname, plocation
 - SSN,Pnumber \longrightarrow hours

Employee_Hobby

<u>SSN</u>	name	<u>Hobby</u>
1	Ahmad	Swimming
1	Ahmad	Running
2	Sara	Tennis
2	Sara	Swimming

Example

- SSN \rightarrow hobby 
- This FD does not hold because the value of SSN does not determine the value of hobby.


Employee_Hobby

<u>SSN</u>	name	Hobby
1	Ahmad	Swimming
1	Ahmad	Running
2	Sara	Tennis
2	Sara	Swimming

Important Note

- If functional dependency FD1 holds in relation R, this means that FD1 holds on all instances (tuples) of relation R

Important Note

- FDs are a property on the meaning of attributes in a relation.
- A FD always applies on relation even if the relation instances do not currently reflect that FD.
- Example: Teacher \rightarrow course 

TEACH

Teacher	Course	Text
Smith	Data Structures	Bartram
Smith	Data Management	Martin
Hall	Compilers	Hoffman
Brown	Data Structures	Horowitz

- This FD does not hold.
- Suppose that record 2 is deleted
- Now, It **looks like** that the FD holds
- But, this is not true

- A given FD must hold regardless of the content (instance) of relation

Attribute Closure (X^+)

- X^+ : The set of attributes that are functionally determined by X .
- How can we find X^+ : Use the following algorithm (Do this)

```
X+ = X
While X+ changes do
  for each Y → Z in F do
    if Y is subset of X+, then add Z to X+
  end-for
End-while
```

Example

$R = A, B, C$

$F = \{$
 $A \rightarrow B,$
 $B \rightarrow C$
 $\}$

Find B^+

First , $B^+ = \{B\}$

Second, since $B \rightarrow C$,and B is a subset of B^+ , then add C to B^+ . So $B^+ = \{BC\}$

So, $B^+ = \{BC\}$

- Because B^+ does not determine **all attributes**, we know that B is not a **key** for R

Find A^+

First , $A^+ = \{A\}$

Second, since $A \rightarrow B$, and A is a subset of A^+ , then add B to A^+ . So $A^+ = \{AB\}$

Third, since $B \rightarrow C$ and B is a subset of A^+ , then add C to A^+ . So $A^+ = \{ABC\}$

Fourth, no need to continue because A^+ already contains all attributes

So, $A^+ = \{ABC\}$

- Because A^+ determines **all attributes**, we know that A is a **key** for R

Prime Attributes

- Prime Attribute: an attribute which is part of a candidate key.
- Non-Prime Attribute: an attribute which is not a part of a candidate key.

Example

Find all candidate keys for R?

$A^+ = \{A, B\}$

- so A is not a candidate key and

$B^+ = \{B\}$, so B is not a candidate key

$C^+ = \{CD\}$, so C is not a candidate key

$AB^+ = \{AB\}$, so AB is not a candidate key

$AC^+ = \{ABCD\}$, so **AC is a candidate key**

$AD^+ = \{ABCD\}$, so **AD is a candidate key**

$BC^+ = \{BCD\}$, so BC is not a candidate key

$BD^+ = \{BD\}$, so BD is not a candidate key

$R = A, B, C, D$

$F = \{$

$A \rightarrow B,$

$C \rightarrow D,$

$BD \rightarrow C$

$\}$

Note: Since no attribute determines “A”, this means that “A” must be part of a key.
This narrows down the possibilities for which attribute(s) is a key.

Normalization

- Normalization: The process of decomposing relations into smaller ones.
- Denormalization: The process of joining relations into bigger relations.
- Normal Form: The conditions we use to test if a relation satisfies certain rules.

Normal Forms

- 1) First Normal Form
- 2) Second Normal Form
- 3) Boyce-Codd Normal Form (BCNF)
- 4) Third Normal Form (3NF)
- 5) Forth Normal Form (4NF) (We will not cover this)
- 6) Fifth Normal Form (5NF) (We will not cover this)

First Normal Form (1NF)

- A relation R is in 1NF if it satisfies the following:
 - Entities are not merged together (Each entity has its unique key). and
 - Each Column has a single value. and
 - No composite attributes.

Example 1

DEPARTMENT

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

- The problems of multi-valued attribute (multiple values in one column)
- Solution is below

Department

<u>dnumber</u>	dname	Dmgr_ssn
5	Research	333445555
4	Administration	987654321
1	Headquarters	888665555

Dep_Location

<u>dnumber</u>	<u>dlocation</u>
5	Bellaire
5	Sugarland
5	Houston
4	Stafford
1	Houston

Example 2

(b)
EMP_PROJ

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

(c)
EMP_PROJ1

Ssn	Ename
-----	-------

EMP_PROJ2

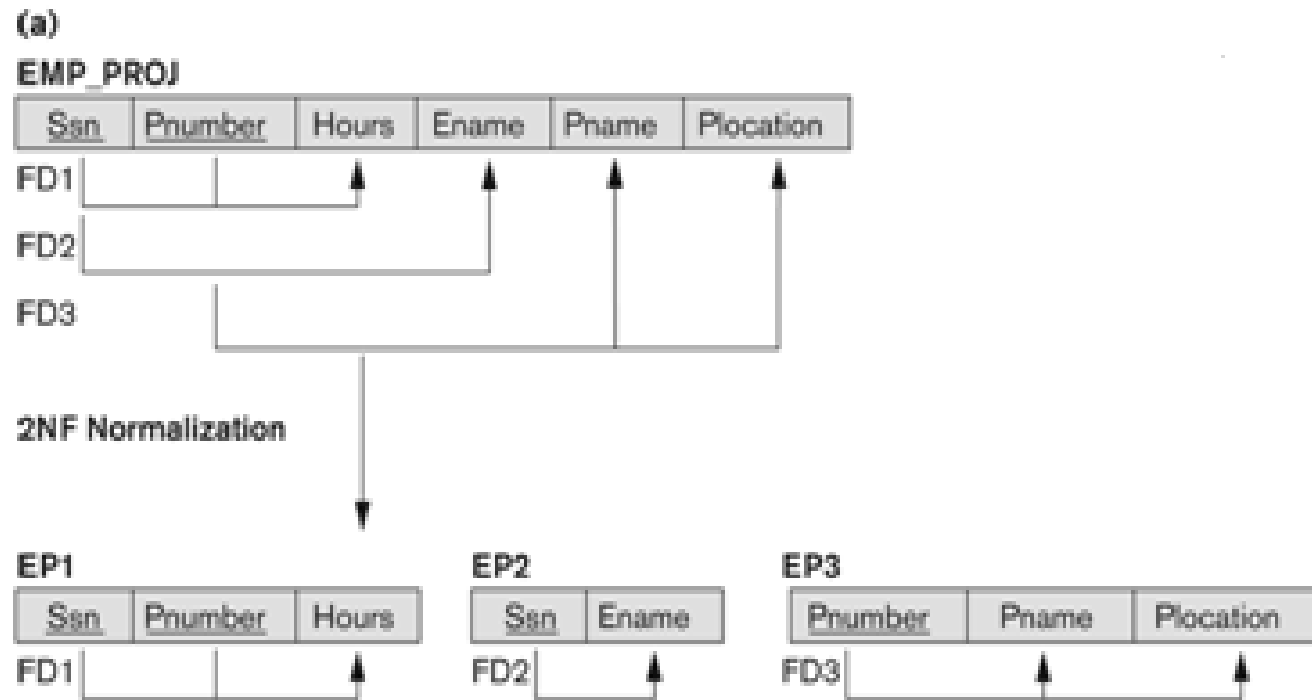
Ssn	Pnumber	Hours
-----	---------	-------

- In b) **Employee information** and **Projects they are working on** are merged in one relation) (1NF Violation). and this causes a problem (multiple values in a single column)
- In c) How normalization in 1NF solves the problem

Second Normal Form (2NF)

- A relation R is in 2NF if:
 - Each non-key attribute is logically determined by the whole Key of R
- Decomposition is performed as follows:
 - Put each key and the attributes that depend on it in a separate relation.

Example 1



- FD1 does not violate 2NF because hours depends on the whole key (SSN,Pnumber)
- FD2 violates 2NF because Ename is determined by SSN (not by SSN,Pnumber)
- FD3 violates 2NF because pname and plocation are not determined by the whole key (SS,Pnumber).

Boyce-Codd Normal Form (BCNF)

- A relation R is in BCNF if:
 - For each non trivial dependency $X \twoheadrightarrow Y$, X is a super key of R.
- Decomposing based on BCNF:
 - For each **violating** FD $X \twoheadrightarrow Y$:
 - You put X,Y in a separate relation.
 - You put R-Y in a separate relation.
 - Each new relation takes its new FDs.
- The decomposition process is repeated until all relations are in BCNF. The final result are in the leaf nodes of decomposition.

Example 1

- Customer = customer-name, customer-street, customer-city

$F = \{\text{customer-name} \rightarrow \text{customer-street, customer-city}\}$

- Branch = branch-name, assets, branch-city

$F = \{\text{branch-name} \rightarrow \text{assets, branch-city}\}$

- Loan = branch-name, customer-name, loan-number, amount

$F = \{\text{loan-number} \rightarrow \text{amount, branch-name}\}$

-
- Customer is in BCNF because customer-name is a super key
 - Branch is in BCNF because branch-name is a super key
 - Loan is **not** in BCNF because customer-name is **not** a super key, (customer-name)+ does not have all attributes in 'Loan'

Loan = branch-name, customer-name, loan-number, amount

Example 1 (Continue)

Loan is decomposed as follows:

Loan = branch-name, customer-name, loan-number, amount
 $F = \{\text{loan-number} \rightarrow \text{amount, branch-name}\}$

Loan-A = loan-number, amount, branch-name
 $F = \{\text{loan-number} \rightarrow \text{amount, branch-name}\}$

Loan-B = loan-number, customer-name
 $F = \{\text{loan-number} \rightarrow \text{customer-name}\}$

- This decomposition is lossless because it is based on a key (loan-number)
- BCNF decomposition is always Lossless Decomposition
- Loan-A is in BCNF, and Loan-B is also in BCNF. So we stop here, no more decompositions are needed

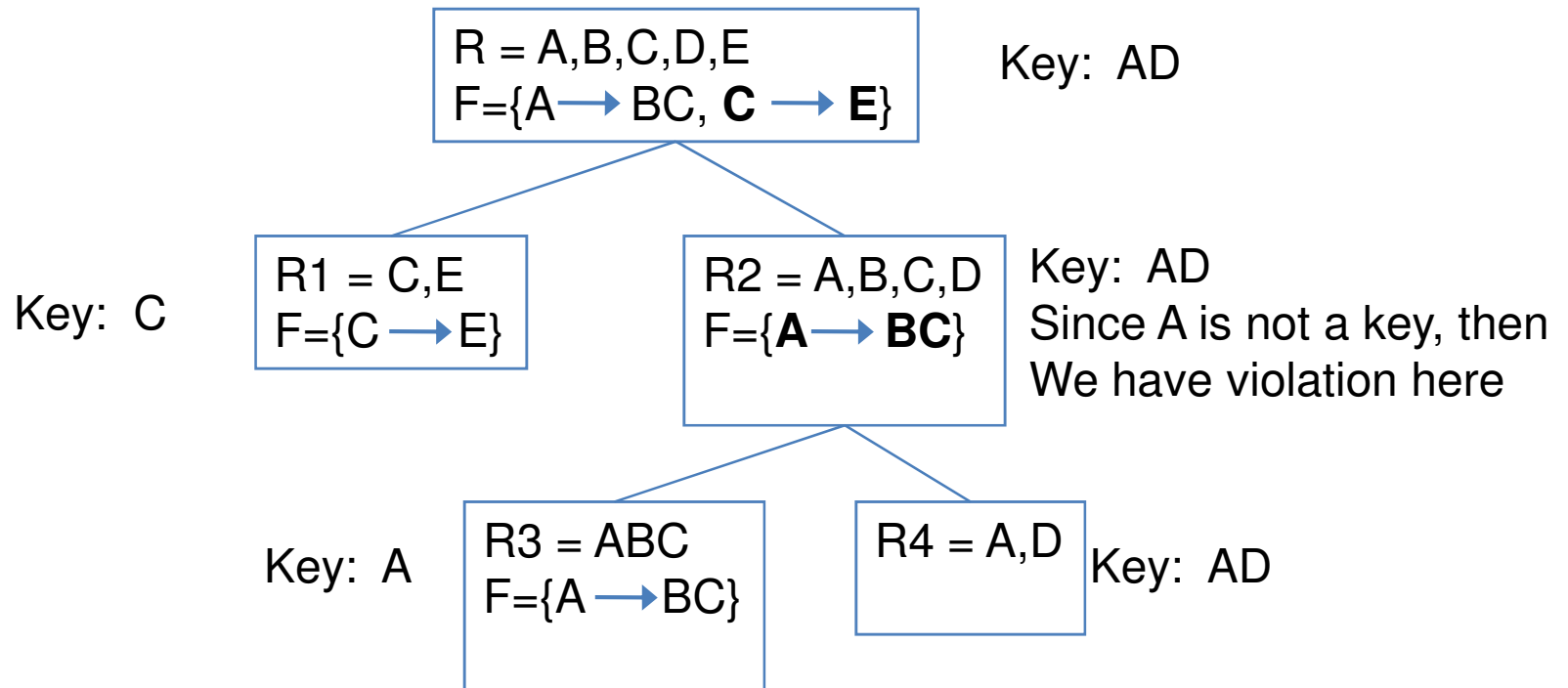
Dependency Preserving Decomposition

- If no FD of original R is lost after decomposition, we say that the decomposition is “Dependency Preserving”.
- Otherwise, it is **not** Dependency Preserving.
- The previous example is a “Dependency Preserving” decomposition because all FDs of original relation are kept after decomposition.
- How does a $X \rightarrow Y$ become “Lost”?
 - After decomposition, if X goes to one relation and Y goes to a different relation, then the FD does not hold anymore.

Example 2

- $R = A, B, C, D, E$
 $F = \{A \rightarrow BC,$
 $C \rightarrow E\}$
- A, D do not appear on the right hand side of any FD, so they should be part of a key of R
- $AD^+ = A, B, C, D, E$ So, AD is a key of R
- $A \rightarrow BC$ is a violation because A is not a superkey
- $C \rightarrow E$ is also a violation because C is not a superkey
- How do you make decomposition? (Next Slide)
 - You can start decomposition using any rule that violates BCNF, I will start with $C \rightarrow E$

Example 2 (Continue)



- Note that R_2 is not in BCNF, this is why we decomposed it
- The result of the whole decomposing process R_1, R_3, R_4 (The leaves)
- This decomposition is dependency preserving because $A \rightarrow BC$ is not lost and $C \rightarrow E$ is not lost

Example 3

- $R = C, T, H, R, S, G$

$F = \{ C \rightarrow T,$

$CS \rightarrow G,$

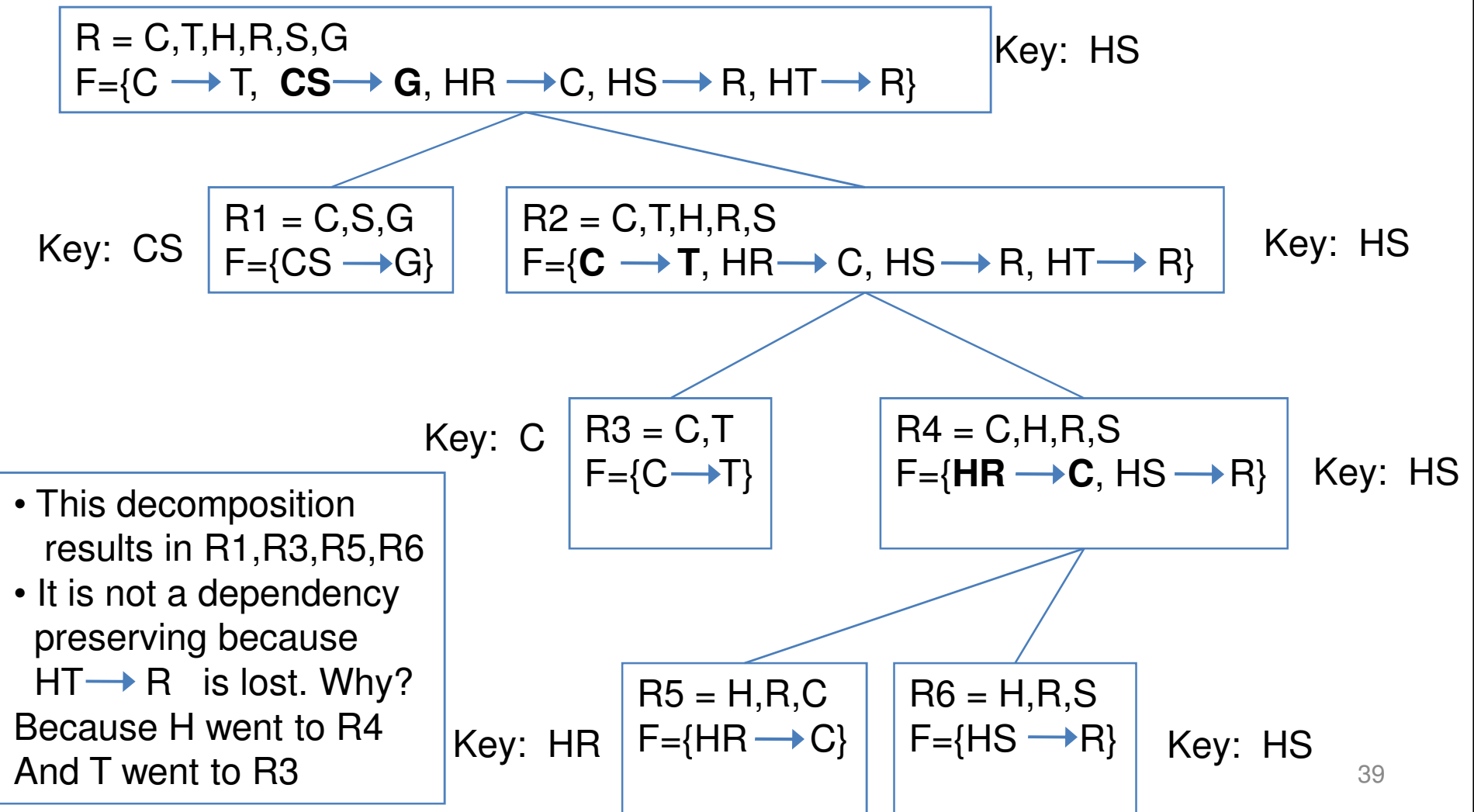
$HR \rightarrow C,$

$HS \rightarrow R,$

$HT \rightarrow R$

- Note that HS are not in the right hand side of any FD, so, they are part of the key for this relation
- $HS^+ = H, S, R, C, G, T$ So, HS is a key of R
- So, all FDs are violating except $HS \rightarrow R$
- I will start decomposition using $CS \rightarrow G$

Example 3 (Continue)



Example 4

R = courseNum, courseName, sectionID, sectionTime, sectionDay, sectionSemester, sectionYear
F={ courseNum \rightarrow courseName
courseNum, sectionID, sectionSemester, sectionYear \rightarrow sectionDay, sectionTime
}

R is not in BCNF

- Candidate Key is: courseNum, sectionID, sectionSemester, sectionYear because its attribute closure gives all attributes in R
- FD1 violates BCNF because left hand side is not a key for R
- FD2 violates BCNF because left hand side is not a key for R

Example 4 (Continue)

R = courseNum, courseName, sectionID, sectionTime, sectionDay, sectionSemester, sectionYear
F={ courseNum \rightarrow courseName
courseNum, sectionID, sectionSemester, sectionYear \rightarrow sectionDay, sectionTime
}

R = courseNum, courseName
F={ courseNum \rightarrow courseName }

Key: courseNum

Now, it is in BCNF

This is actually table course

R=courseNum, sectionID, sectionTime, sectionDay, Semester, Year
F={courseNum, sectionID, semester, Year \rightarrow sectionDay, sectionTime}

Key: courseNum, sectionID, semester, year

Now, it is in BCNF

This is actually table section

Third Normal Form (3NF)

- BCNF is strict
- 3NF relaxes BCNF by adding one relaxing constraint to 3NF:
- A relation R is in 3NF if
 - For each non trivial dependency $X \rightarrow Y$:
 - X is a super key of R. or
 - ***Each attribute in Y is a member of some key of R (Prime Attribute)***
- Decomposition based on 3NF is done in the same way
- Decomposing based on 3NF minimizes the probability of losing FDs but increases redundancy
- In other words, the probability of losing FDs in 3NF is smaller than BCNF

Example

$R = A, B, C, D, E$

$F = \{AB \rightarrow CDE \quad AC \rightarrow BDE \quad B \rightarrow C \quad C \rightarrow B \quad C \rightarrow D \quad B \rightarrow E\}$

Is R in 3NF?

- Candidate Keys are: AB and AC
- $AB \rightarrow CDE$ does not violate 3NF because AB is a superkey
- $AC \rightarrow BDE$ does not violate 3NF because AC is a superkey
- $B \rightarrow C$ does not violate 3NF because C is a prime attribute (part of AC key)
- $C \rightarrow B$ does not violate 3NF because C is a prime attribute (part of AB key)
- $C \rightarrow D$ violates 3NF
- $B \rightarrow E$ violates 3NF

Example (Continue)

$R = A, B, C, D, E$

$F = \{AB \rightarrow CDE \quad AC \rightarrow BDE \quad B \rightarrow C \quad C \rightarrow B \quad \mathbf{C} \rightarrow \mathbf{D} \quad B \rightarrow E\}$

Keys: AB
And AC

$R_1 = C, D$
 $F = \{C \rightarrow D\}$

Key: C

$R_2 = A, B, C, E$

$F = \{B \rightarrow C \quad C \rightarrow B \quad \mathbf{B} \rightarrow \mathbf{E}\}$

Keys: AB
And AC

Key: B

$R_3 = B, E$

$F = \{B \rightarrow E\}$

$R_4 = A, B, C$

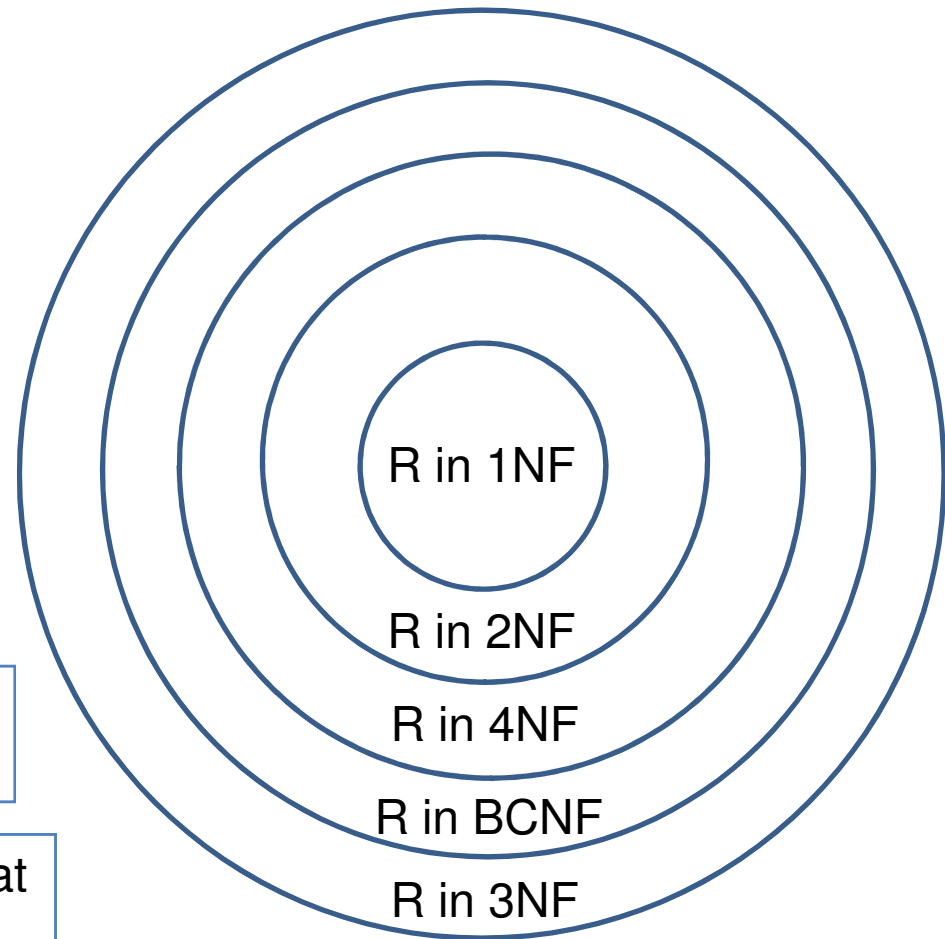
$F = \{B \rightarrow C, C \rightarrow B\}$

Keys: AB
And AC

- This decomposition results in R_1, R_3, R_4
- It is not a dependency preserving because $AC \rightarrow BDE$ is lost and $AB \rightarrow CDE$ is also lost

Keys: Since AB and AC are candidate keys, then A, B, C are prime attributes

Normal Forms Relationship



- If R is in 3NF, does that mean that R is also in BCNF? No.

- If R is in BCNF, does that mean that R is also in 3NF? Yes.