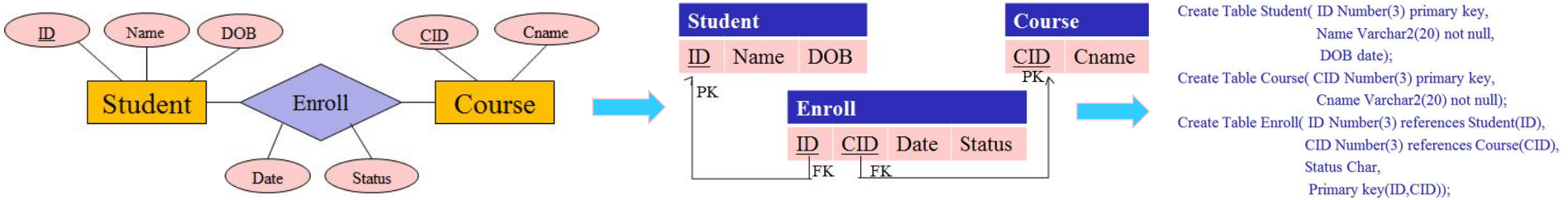
Unit 2 :ER Model and Relational Data Model

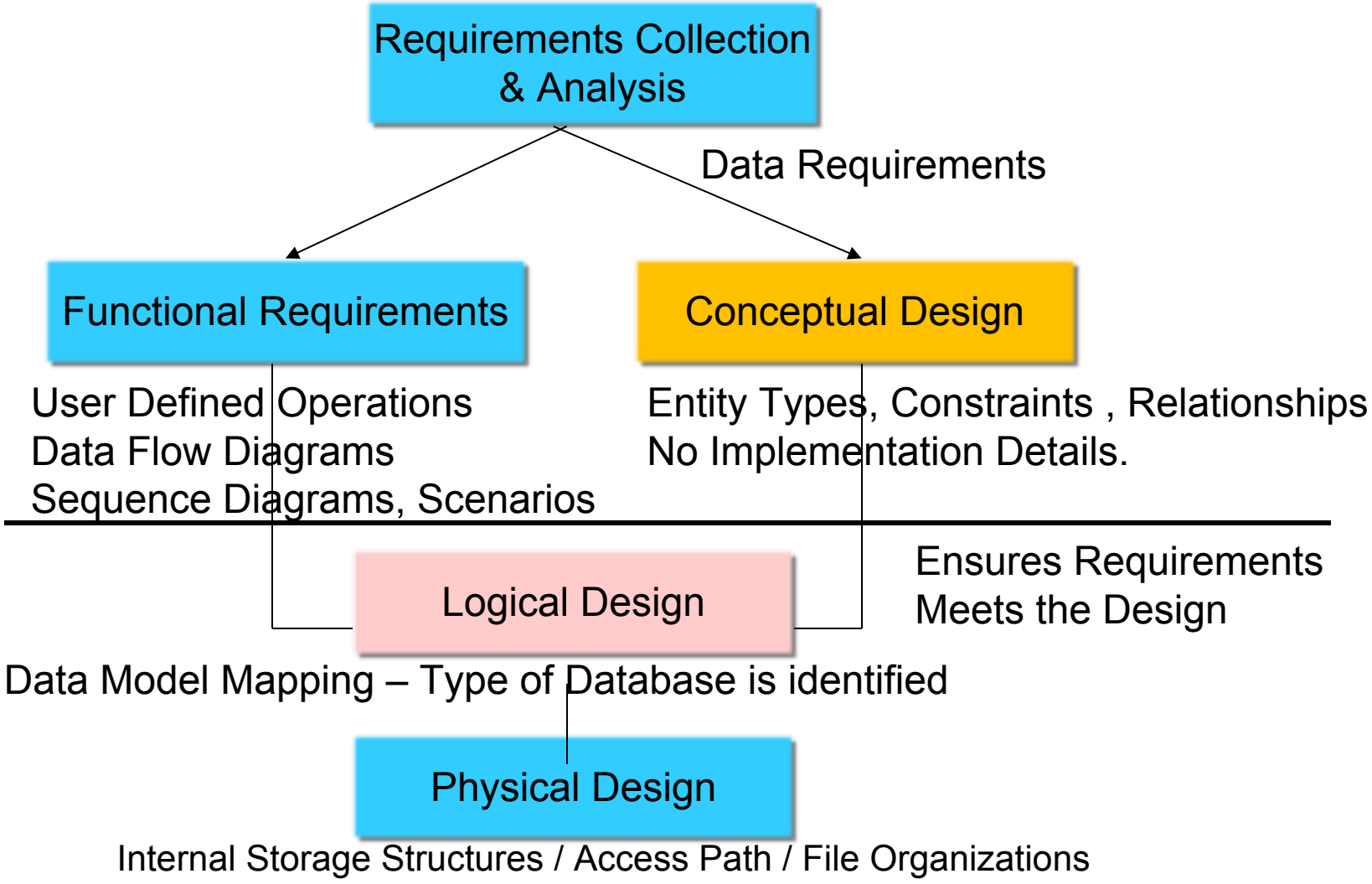


Dr. Aruna M G
Associate Professor
Dept. of AI&ML, DSCE

Contents

- **Data Modelling using the Entity-Relationship(ER) model:**
- Using High-Level Conceptual Data Models for Database Design, A sample Database Application, Entity types, Entity Sets, Attributes, and Keys, Relationship Types, Relationship Sets, Roles and Structural Constraints, Weak Entity types, Refining the ER Design, ER Diagrams, Naming Conventions and Design Issues, Relationship Types of Degree Higher than two, Relational Database Design using ER-to-Relational Mapping.

Database Design Phases…



Overview of Database Design Process

- Requirement Analysis
- Conceptual Design
- Logical Design
- Schema Refinement : (Normalization)
- Physical Database Design and Tuning

Database Design Steps

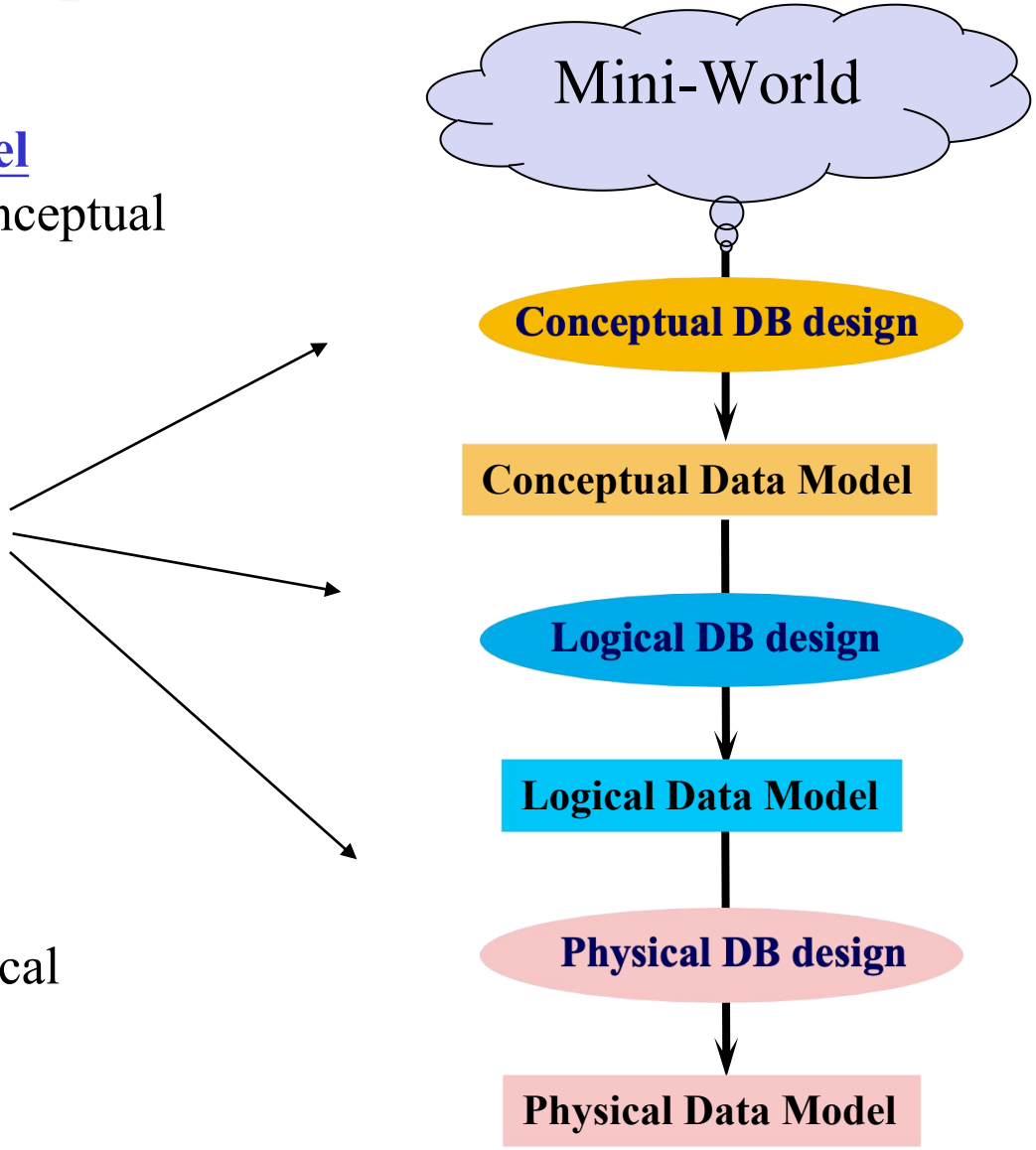
Entity-relationship Model

Typically used for conceptual database design

Three Levels of Modeling

Relational Model

Typically used for logical database design



Conceptual Design Using the Entity- Relationship (ER) Model

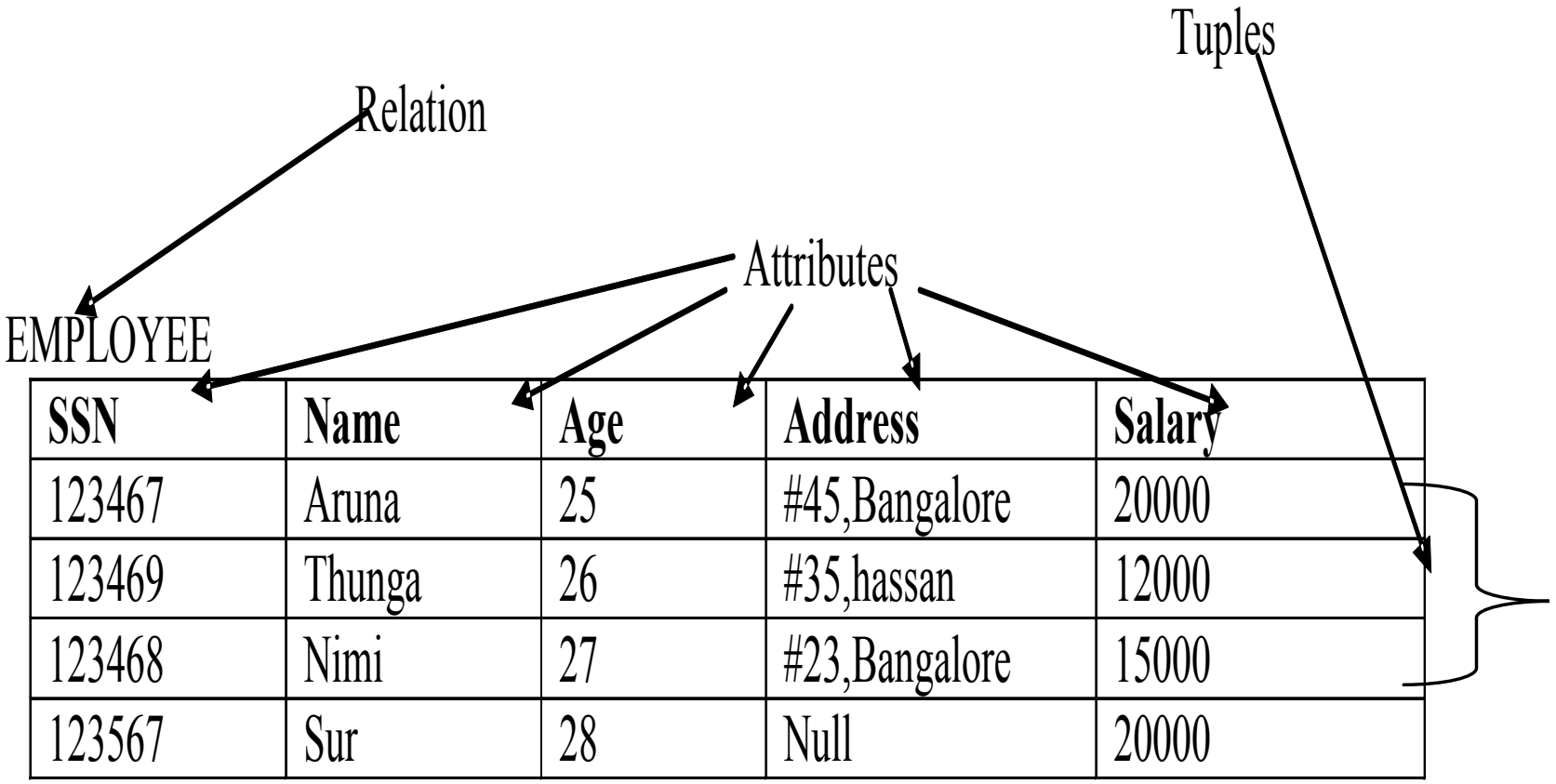
Motivation - Why Conceptual Model?

- If you cannot model, you cannot comprehend, and if you cannot comprehend, you cannot control
- Dual goal:
  - Analysis and conceptualization
  - Presentation
- You've just been hired by Bank of America as their DBA for their online banking web site.
- You are asked to create a database that monitors:
  - customers
  - accounts
  - loans
  - branches
  - transactions, …
- Now what??!!!

Relation

- A relational database is made up of a collection of tables or relation.

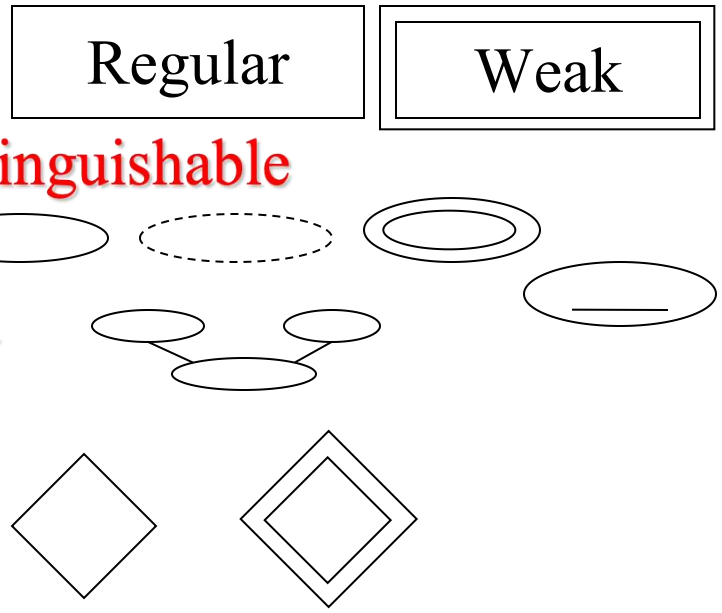
Common terms	DBMS	RDBMS
1. Database	Table	Database
2. Table	Table	Relation
3. Column	Field	Attribute
4. Row	record	tuple



SSN	Name	Age	Address	Salary
123467	Aruna	25	#45,Bangalore	20000
123469	Thunga	26	#35,hassan	12000
123468	Nimi	27	#23,Bangalore	15000
123567	Sur	28	Null	20000

Figure2.3Employee Relation

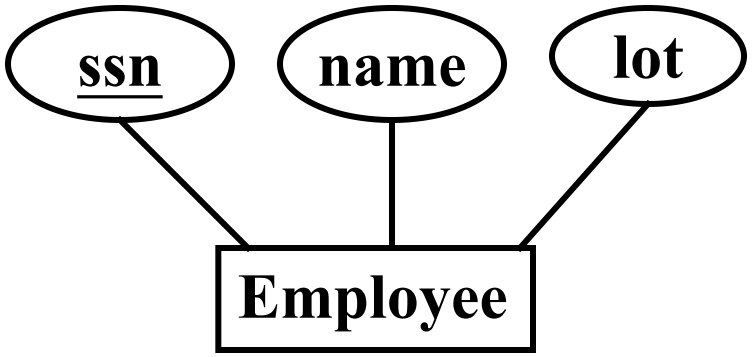
E-R Modeling Concepts



- **Entity**
  - is anything that exists and is distinguishable
- **Attribute**
  - properties that describe an entity
- **Relationship**
  - an association between entities

- **ER Model (Entity Relationship Model)**
- The ER model describes data as entities, relationships, and attributes

Entity, Entity Type, Entity Instance, Entity Set



Entity Type

Entity Instance

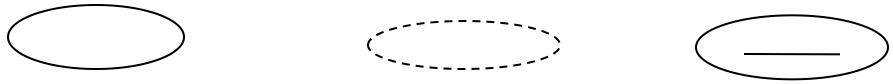
SSN	NAME	LOT
123- 22- 3666	Attishoo	48
231- 31- 5368	Smiley	22
131- 24- 3650	Smethurst	35

Entity Set

```
CREATE TABLE Employees
(ssn CHAR (11),
name CHAR (20),
lot INTEGER,
PRIMARY KEY (ssn))
```

Attributes

- Entity types have ***Attributes*** (or properties) which associate each entity with a value from a *domain* of values for that attribute



- **Attribute Type:**
- It is the property of entity type instance is called attribute type

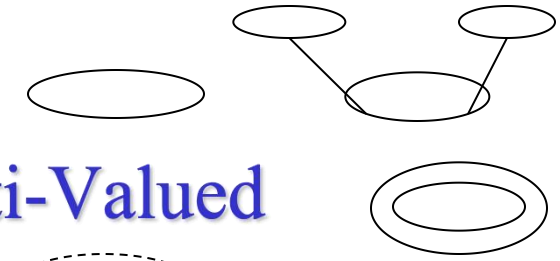
- **Attribute Instance:**
- It is the property of entity instance is called attribute instance.
- –A specific entity will have a value for each of its attributes.

For example a specific employee entity may have

- Name='John Smith', SSN='123456789', Address ='731, Fondren, Houston, TX', Sex='M', BirthDate='09-JAN-55‘

Types of Attributes

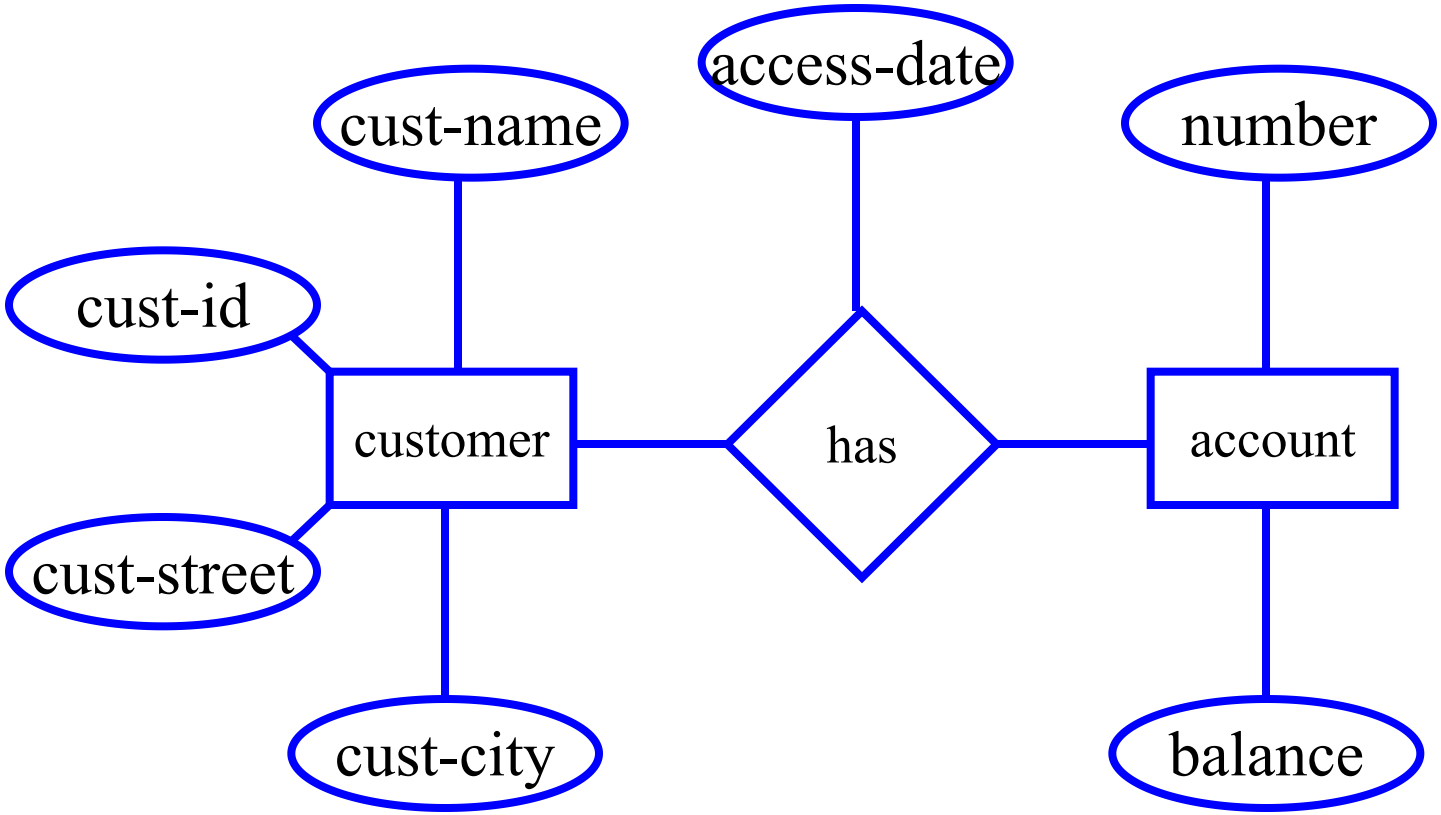
- Attributes can be
  - Simple v/s Composite
  - Single Valued v/s Multi-Valued
  - Stored v/s Derived
  - Key



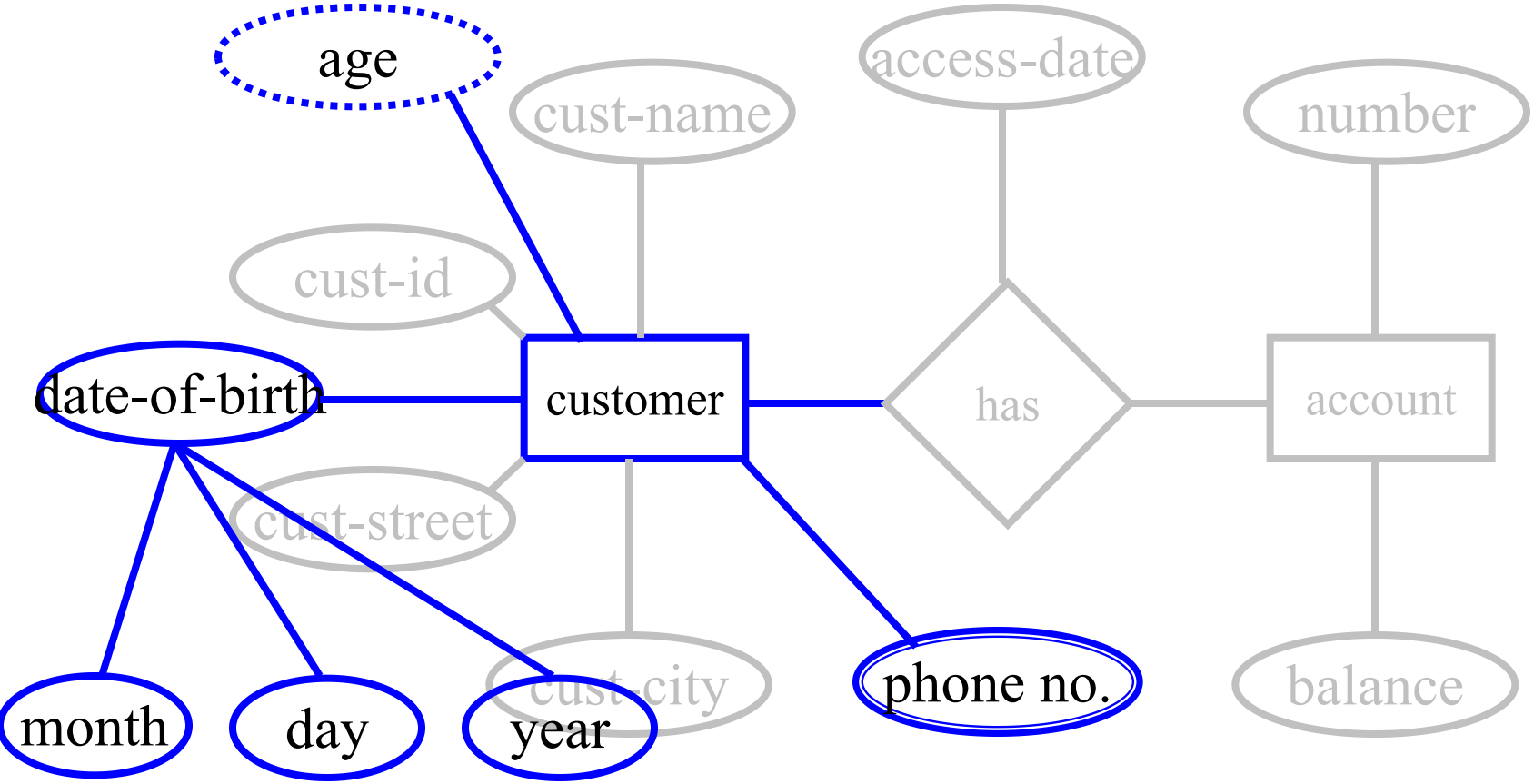
- Example:
  - simple (atomic) e.g. Surname; date of birth
  - composite e.g. address (street, town, postcode)
  - multi-valued e.g. phone number
  - complex nested , multi-valued and composite
  - Stored / derived e.g. D.O.B. ; age
  - Key e.g. SSN
- Relationship types can also have attributes!

Types of Attributes :

Simple, Stored, Single Valued



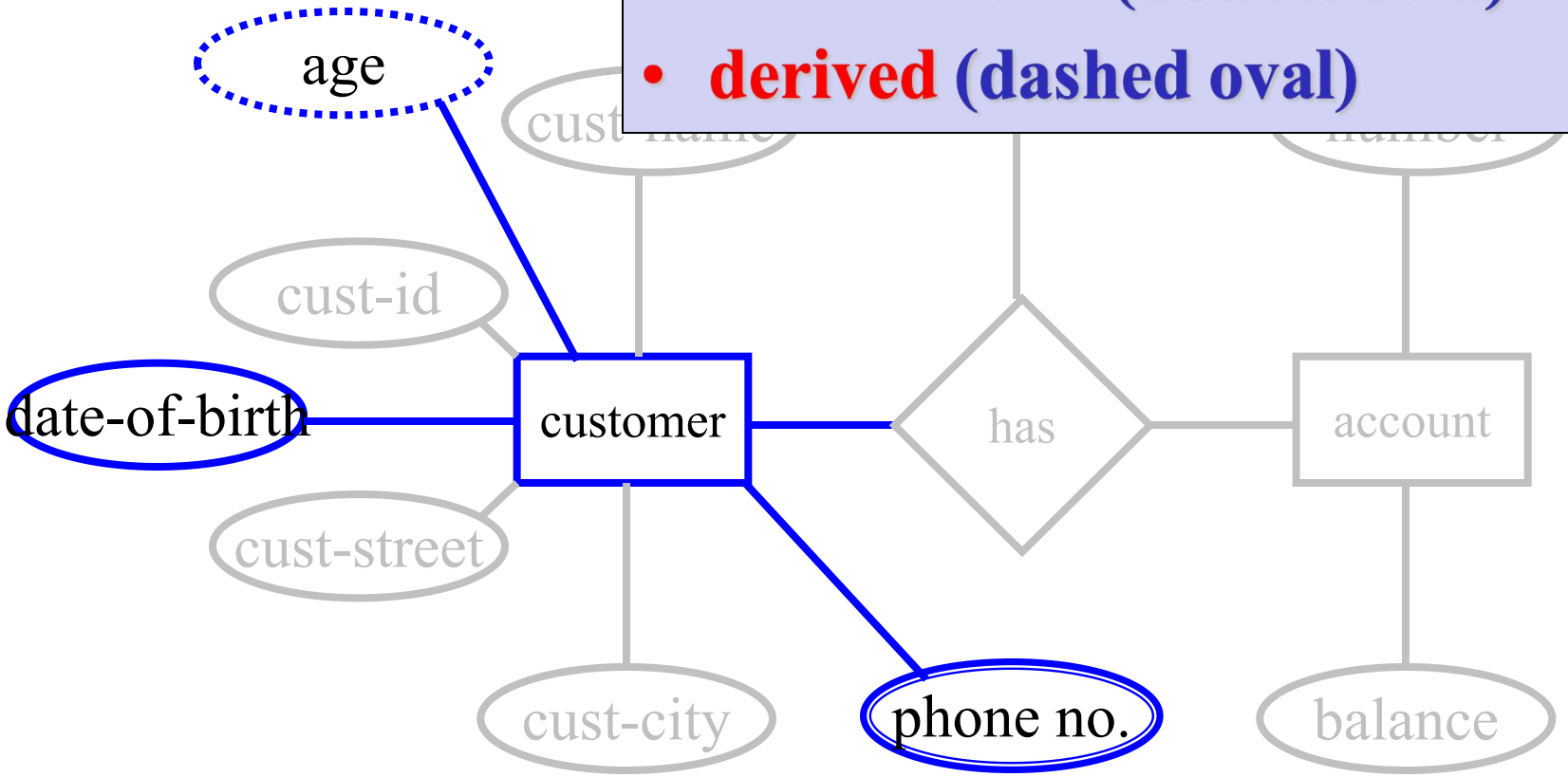
Types of Attributes



Composite Attribute

Types of Attributes

- **multi-valued (double oval)**
- **derived (dashed oval)**



Complex

- The **composite and multi-valued attributes** may be nested arbitrarily to any number of levels although this is rare.
- Or
- The combination of composite and multivalued attribute is also called complex attribute.
- Composite attribute between denoted by parentheses ( ) and multivalued attributes between braces { }.
- Example1: PreviousDegrees of a STUDENT is a composite multi-valued attribute denoted by {PreviousDegrees (College, Year, Degree, Field)}.
- Example2:

{AddressPhone({Phone(AreaCode,PhoneNumber)},
Address(StreetAddress(Number,Street,ApartmentNumber),
City,State,Zip)) }

Null Attirbute

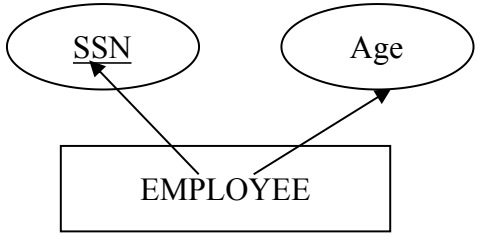
- An entity may not have any applicable value for an attribute. A special value is used in such situation called NULL.
- Example: PhoneNo or Degree attribute of EMPLOYEE entity.

The unknown category of Null is classified into two cases:

1. The first case arises when it is know that the attribute value exists but is missing.
2. The second case, when it is not known whether the attribute value exist or not.

Key Attributes

- Key = set of attributes identifying individual entities or relationships
- An entity type has an attribute whose values can be used to identify each entity uniquely is called a key attribute.
- Keys are one of the basic requirements of a relational database model. It is widely used to identify the tuples(rows) uniquely in the table. We also use keys to set up relations amongst various columns and tables of a relational database.



Different Types of Database Keys

- Candidate Key
- Primary Key
- Super Key
- Alternate Key
- Foreign Key
- Composite Key

Entity Keys

- ***Super Key***
  - any set of attributes that can distinguish entities
- ***Candidate Key***
  - a minimal Super Key
    - Can't remove any attribute and preserve key-ness
      - {cust-id, age} not a super key
      - {cust-name, cust-city, cust-street} is
        - » assuming cust-name is not unique
- ***Primary Key***
  - One of the Candidate Keys chosen as the **key** by DBA
  - Underlined in the ER Diagram

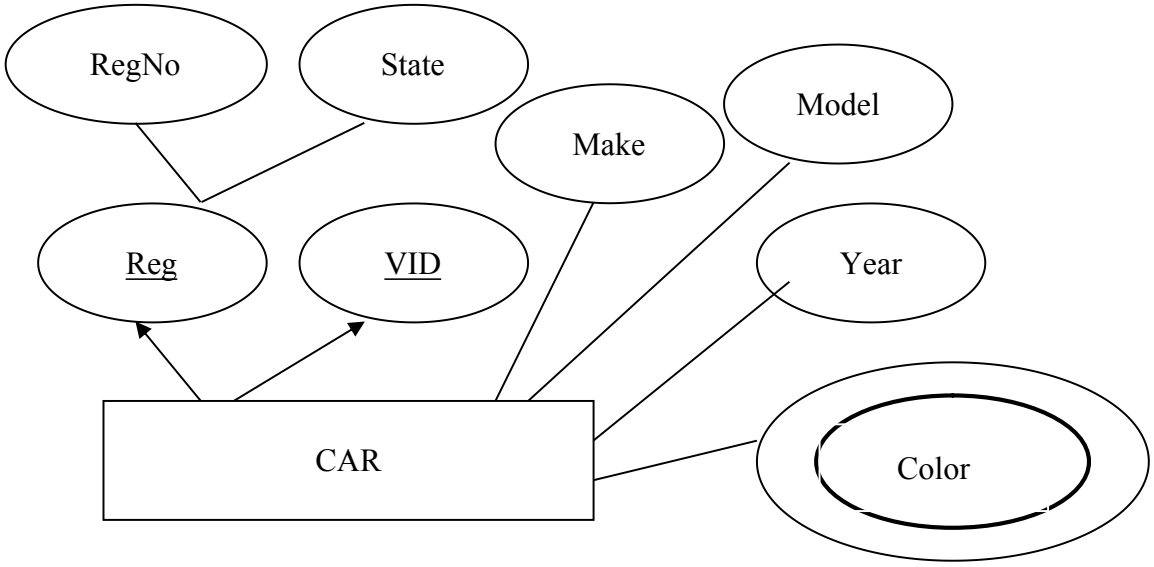
Super Key

- The set of attributes that can uniquely identify a tuple is known as Super Key. For Example, STUD_NO, (STUD_NO, STUD_NAME),
- A super key is a group of single or multiple keys that identifies rows in a table.
- It supports NULL values.
- Adding zero or more attributes to the candidate key generates the super key.
- A candidate key is a super key but vice versa is not true.
- Super Key values may also be NULL.

Candidate key

An entity type may have more than one key called candidate key.
Or

The minimal set of attributes that can uniquely identify a tuple is known as a candidate key.
For example, the CAR entity type may have two keys:
–VehicleIdentificationNumber (popularly called VIN) and
–VehicleTagNumber (Number, State), also known as license_plate number.



Candidate key

For Example, STUD_NO in STUDENT relation.

- It is a minimal super key.
- It is a super key with no repeated data is called a candidate key.
- The minimal set of attributes that can uniquely identify a record.
- It must contain unique values.
- It can contain NULL values.
- Every table must have at least a single candidate key.
- A table can have multiple candidate keys but only one primary key.
- The value of the Candidate Key is unique and may be null for a tuple.
- There can be more than one candidate key in a relationship.

Primary Key

- There can be more than one candidate key in relation out of which one can be chosen as the primary key.
- It is a unique key.
- It can identify only one tuple (a record) at a time.
- It has no duplicate values, it has unique values.
- It cannot be NULL.
- Primary keys are not necessarily to be a single column; more than one column can also be a primary key for a table.

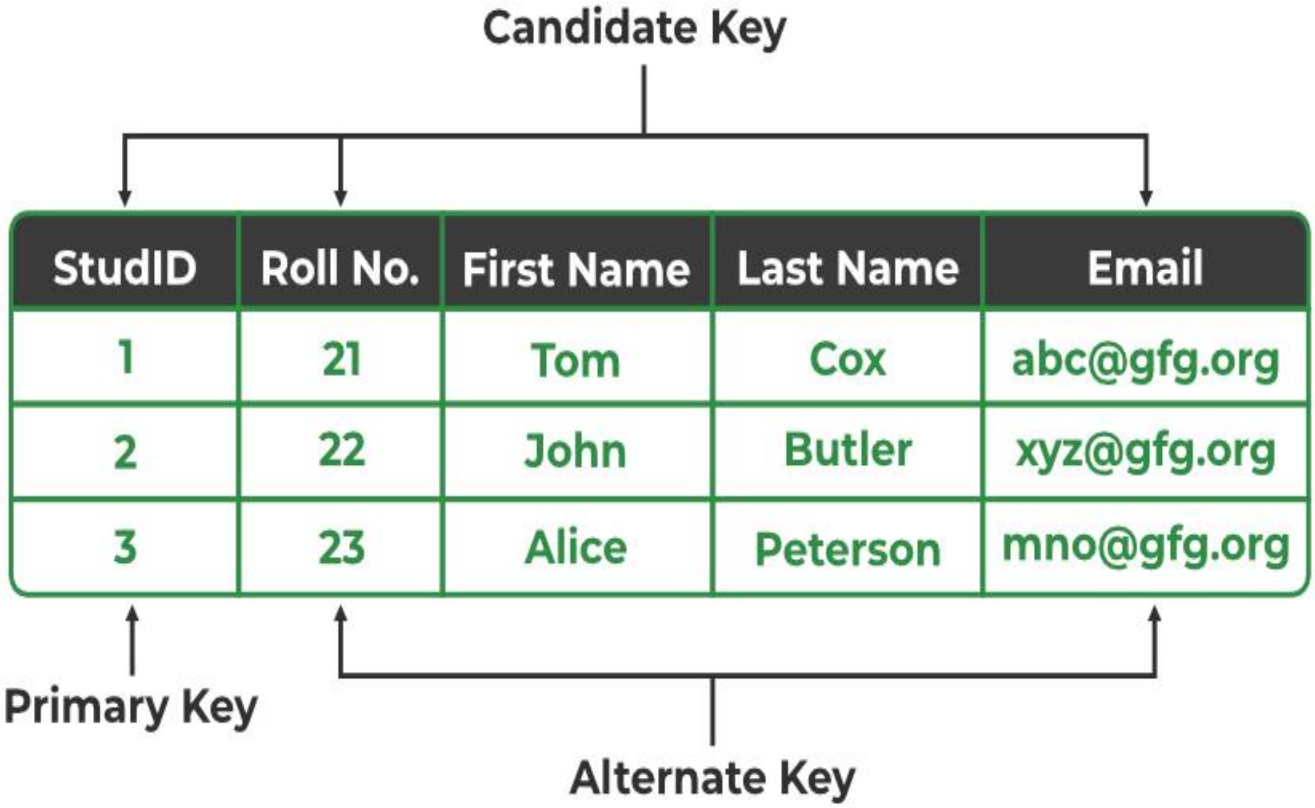
Example:

- STUD_NO, as well as STUD_PHONE, are candidate keys for relation STUDENT but STUD_NO can be chosen as the primary key (only one out of many candidate keys).
- STUDENT table -> Student(STUD_NO, SNAME, ADDRESS, PHONE) , STUD_NO is a primary key.

Table STUDENT

STUD_NO	SNAME	ADDRESS	PHONE
1	Shyam	Delhi	123456789
2	Rakesh	Kolkata	223365796
3	Suraj	Delhi	175468965

Example


Candidate Key
StudID
Roll No.
First Name
Last Name
Email
1
21
Tom
Cox
abc@gfg.org
2
22
John
Butler
xyz@gfg.org
3
23
Alice
Peterson
mno@gfg.org
Primary Key
Alternate Key

Composite Key (CK)

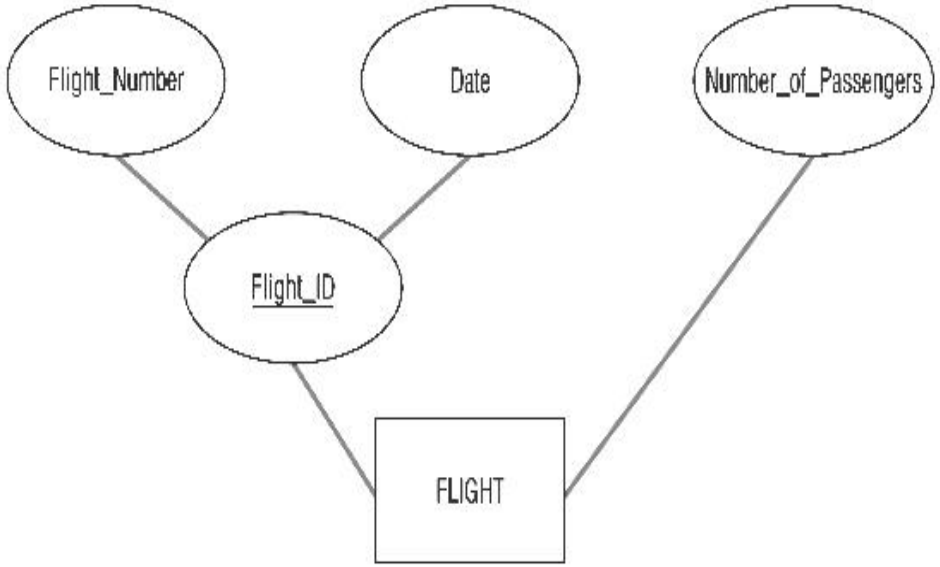
A key attribute may be composite. Such attribute is composite key.

Or

A table might not have a single column/attribute that uniquely identifies all the records of a table. To uniquely identify rows of a table, a combination of two or more columns/attributes can be used.

For example, VehicleTagNumber is a key of the CAR entity type with components (Number, State).

Example2: Flight table Flight_ID is composite key



Continued.

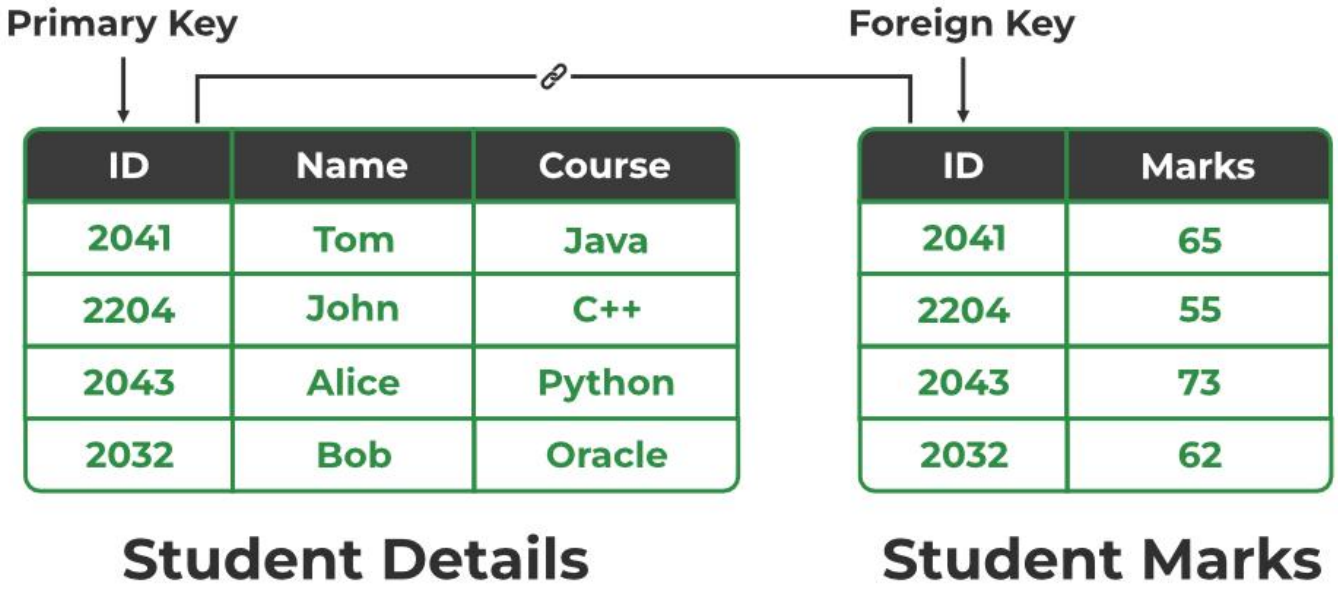
- It still can give duplicate values in rare cases. So, we need to find the optimal set of attributes that can uniquely identify rows in a table.
- It acts as a primary key if there is no primary key in a table
- Two or more attributes are used together to make a composite key.
- Different combinations of attributes may give different accuracy in terms of identifying the rows uniquely.

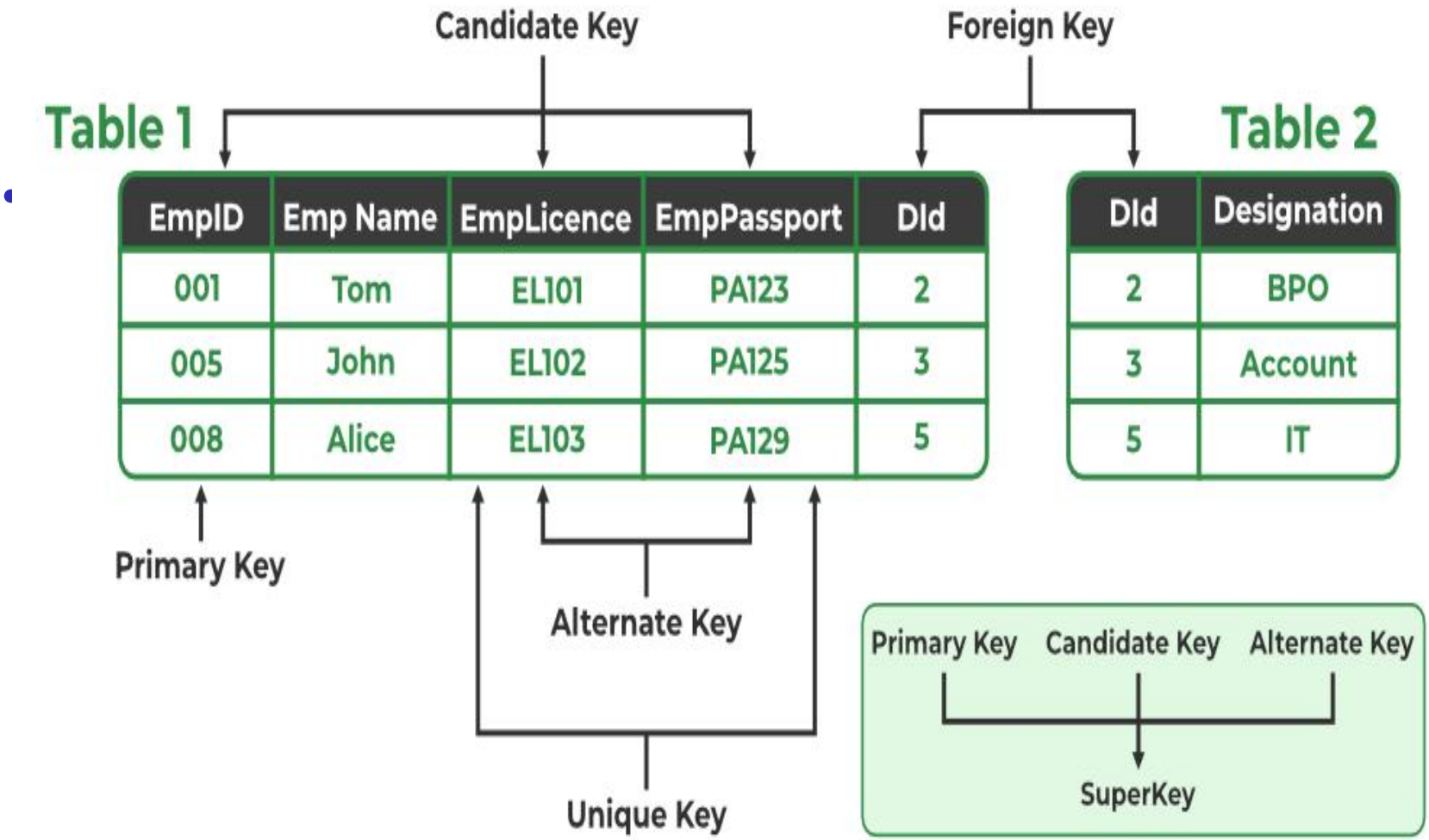
Foreign Key (FK)

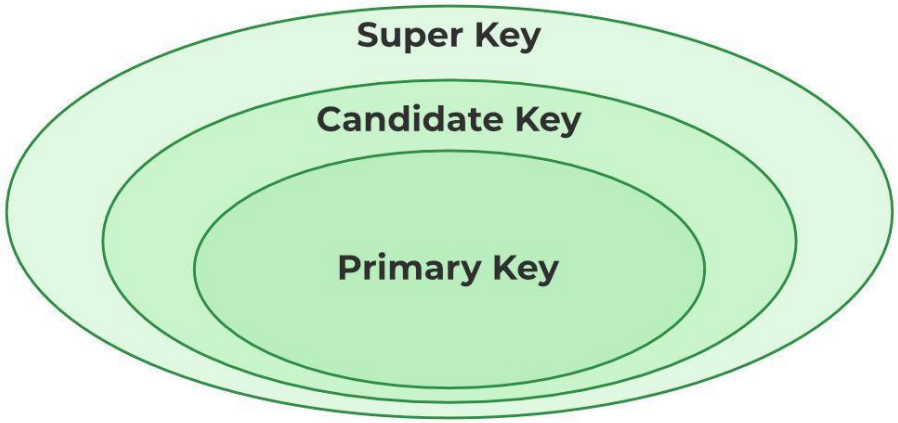
- If an attribute can only take the values which are present as values of some other attribute, it will be a foreign key to the attribute to which it refers.
- The relation which is being referenced is called **referenced relation** and the corresponding attribute is called referenced attribute the relation which refers to the referenced relation is called referencing relation and the corresponding attribute is called referencing attribute.
- The referenced attribute of the referenced relation should be the primary key to it.

FK

- It is a key it acts as a primary key in one table and it acts as secondary key in another table.
- It combines two or more relations (tables) at a time.
- They act as a cross-reference between the tables.

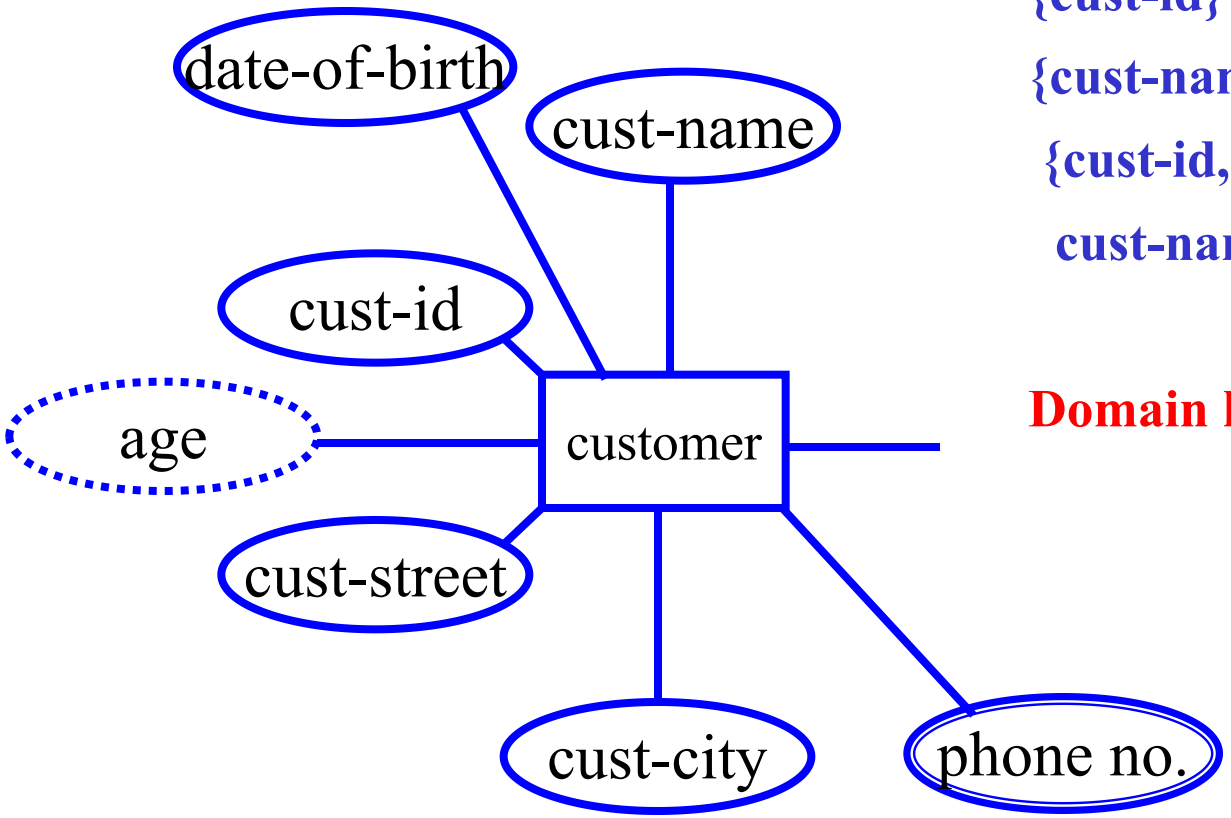



Candidate Key
Foreign Key
Table 1
Table 2
EmpID
Emp Name
EmpLicence
EmpPassport
DId
001
Tom
EL101
PA123
2
005
John
EL102
PA125
3
008
Alice
EL103
PA129
5
DId
Designation
2
BPO
3
Account
5
IT
Primary Key
Alternate Key
Unique Key
Primary Key
Candidate Key
Alternate Key
SuperKey



SCALER Topics

Entity Keys



Possible Keys:

{cust-id}

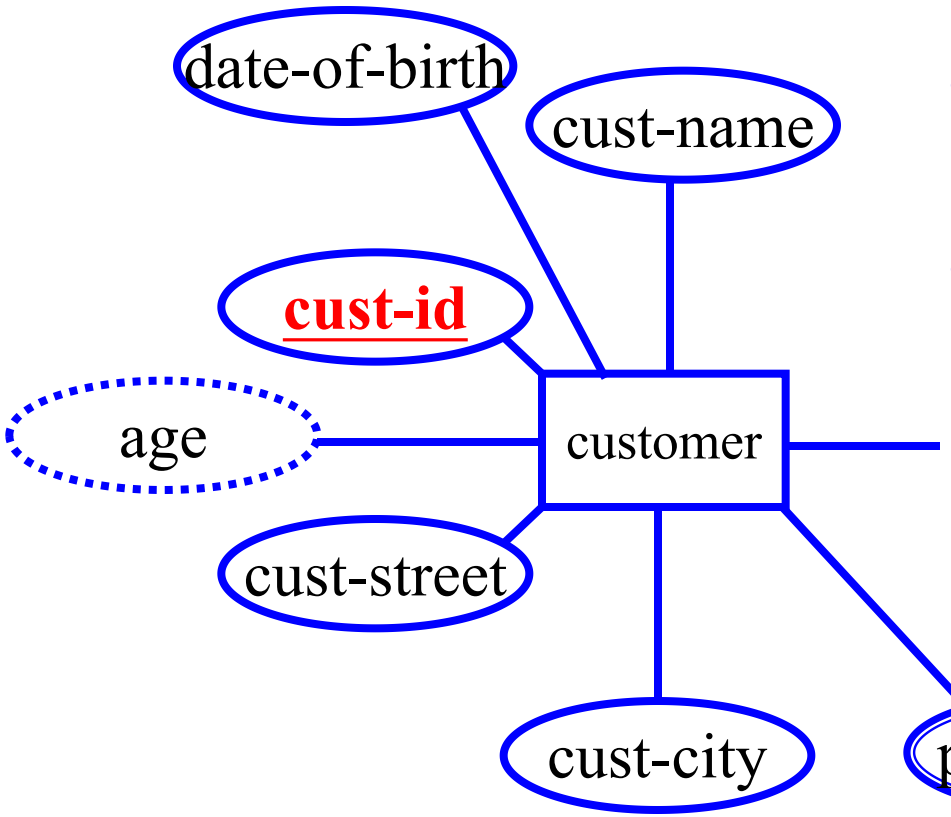
{cust-name, cust-city, cust-street}

{cust-id, age}

cust-name ?? Probably not.

Domain knowledge dependent !!

Entity Keys



- {*cust-id*} is a natural primary key
- Typically, SSN forms a good primary key
- Try to use a candidate key that rarely changes
  - e.g. something involving address not a great idea

Keys in RDBMS

- In EasyShop retail system, how is uniqueness maintained among customers?
- How do you establish the link/association between two relations?

Candidate key	Primary key	Foreign key
• A *minimal* set of columns/attributes that can be used to *uniquely* identify a single row/tuple in a given relation, is called candidate key • Identified during **design time**	• Database designer selects one of the candidate keys as primary key for the purpose of identification of a row of table uniquely • Implented during table creation	• A foreign key is a column or combination of columns that is used to establish and enforce a link between the data in two relations • Implented during table creation

Value set (or Domain) of attributes

- The domain of an attribute is the set of all possible values from which the attributes can take its values.
- Value sets are not displayed in ER diagrams. It is specified during creating entity type.

Example1: Season attribute have the possible values are Spring, Summer, Autumn, Winter.

Example2 :Gender attribute can have Female or Male.

Mathematically Definition of value sets of attribute:

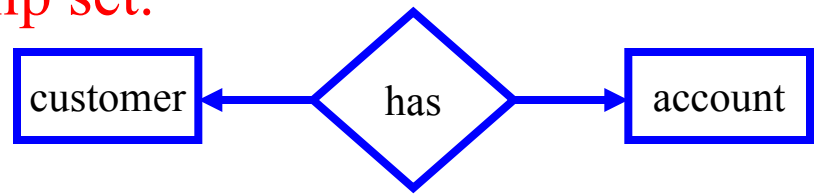
- An attribute A of entity type E whose vale set is V can be defined as a function from E to the power set P(V) of V;
- **A : E→ P(V)**

Relationships

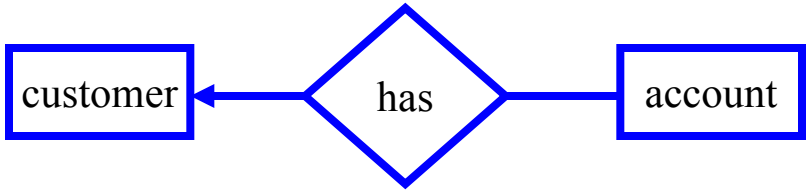
- A relationship is ".. An association among entities (the participants).." . Relationships link entities with each other

Express the number of entities to which another entity can be associated via a relationship set.

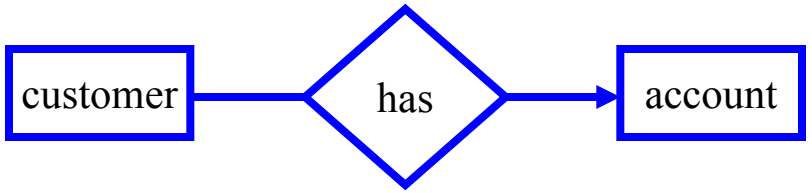
- One-to-One



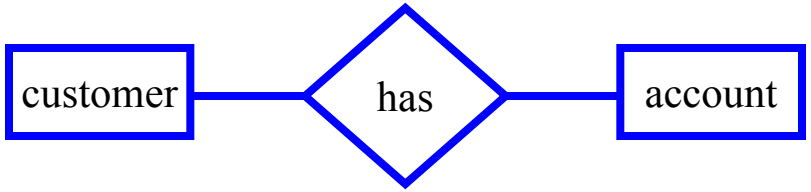
- One-to-Many



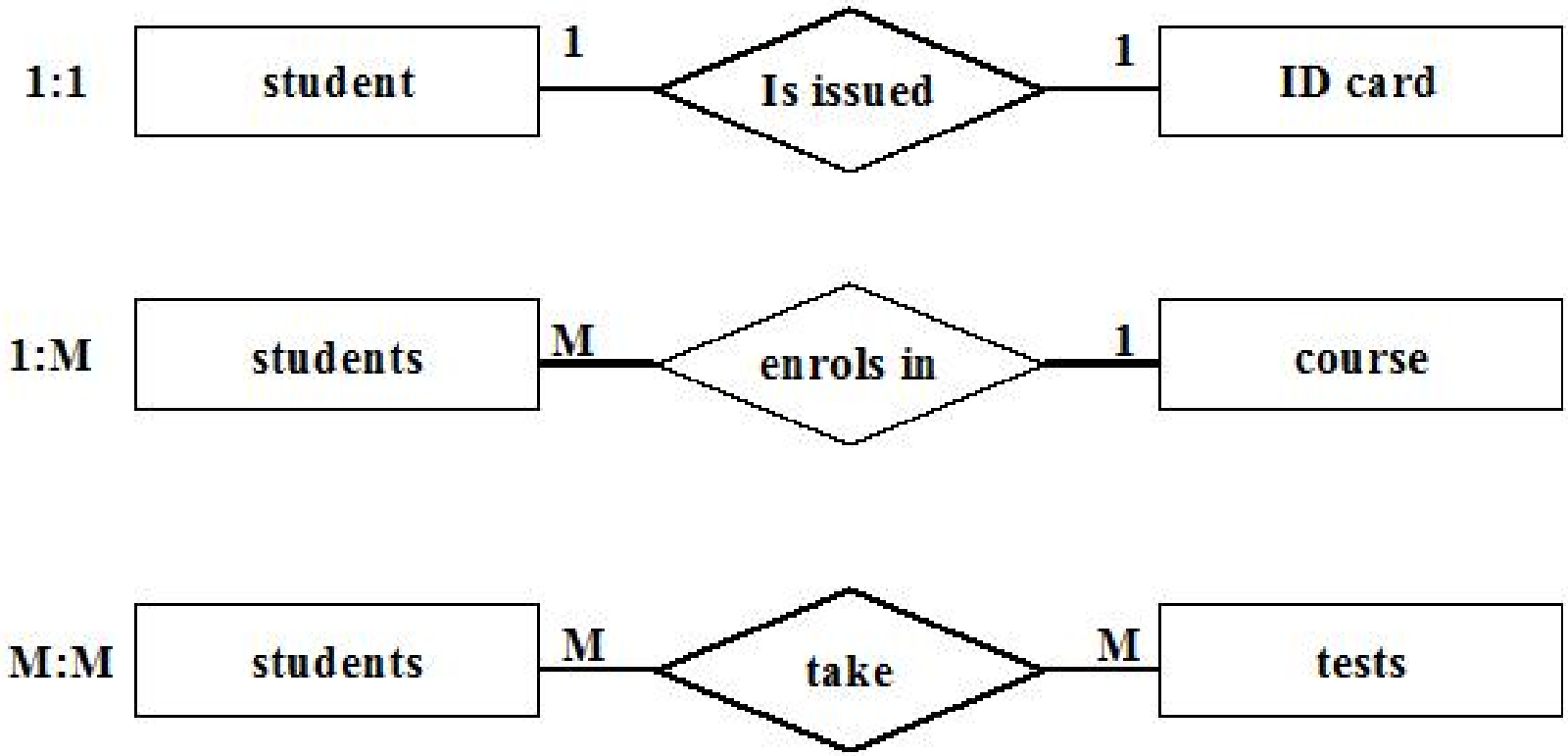
- Many-to-One



- Many-to-Many



Mapping Cardinalities



Relationship Type

- A relationship type between two entity types defines the set of all association between these entity types. A relationship relates two or more distinct entities with a specific meaning.
- or
- A relationship type R among n entity types $E_1, E_2, \ldots\ldots, E_n$ defines a set of associations.

Relationship set

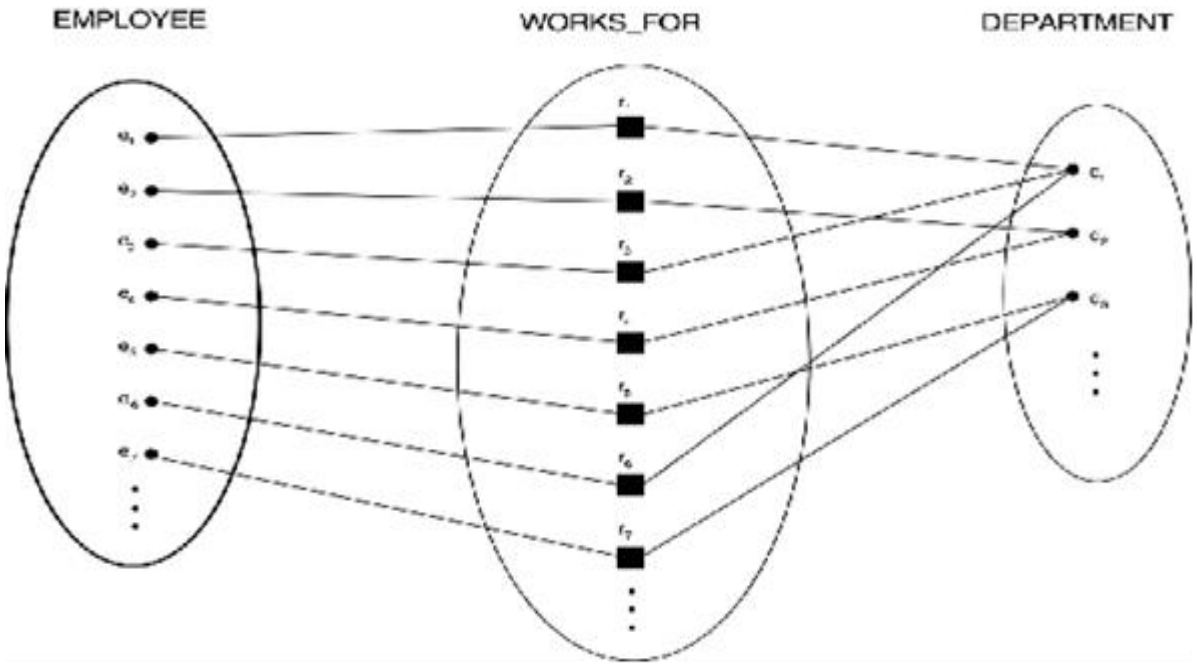
- An instance of a relationship set is a set of relationship.

Relationship instance

- Each instance of the relationship between members of these entity types is called a relationship instance.
- $\mathbf{r_i = (e_1, \ldots, e_n)}$

Mathematically, the relationship set R is a set of relationship instance r_i , where each r_i associates n individual entities ($e_{1,\ldots,}$ e_n) and each entity e_i in r_i is a member of entity type E_j ,1<=j<=n.

Where R be the relationship type. E_1,E_2,….E_n be the entity type. e_1 through e_n, are entity set and r_i is the relationship instances.



Degree of a relationship types

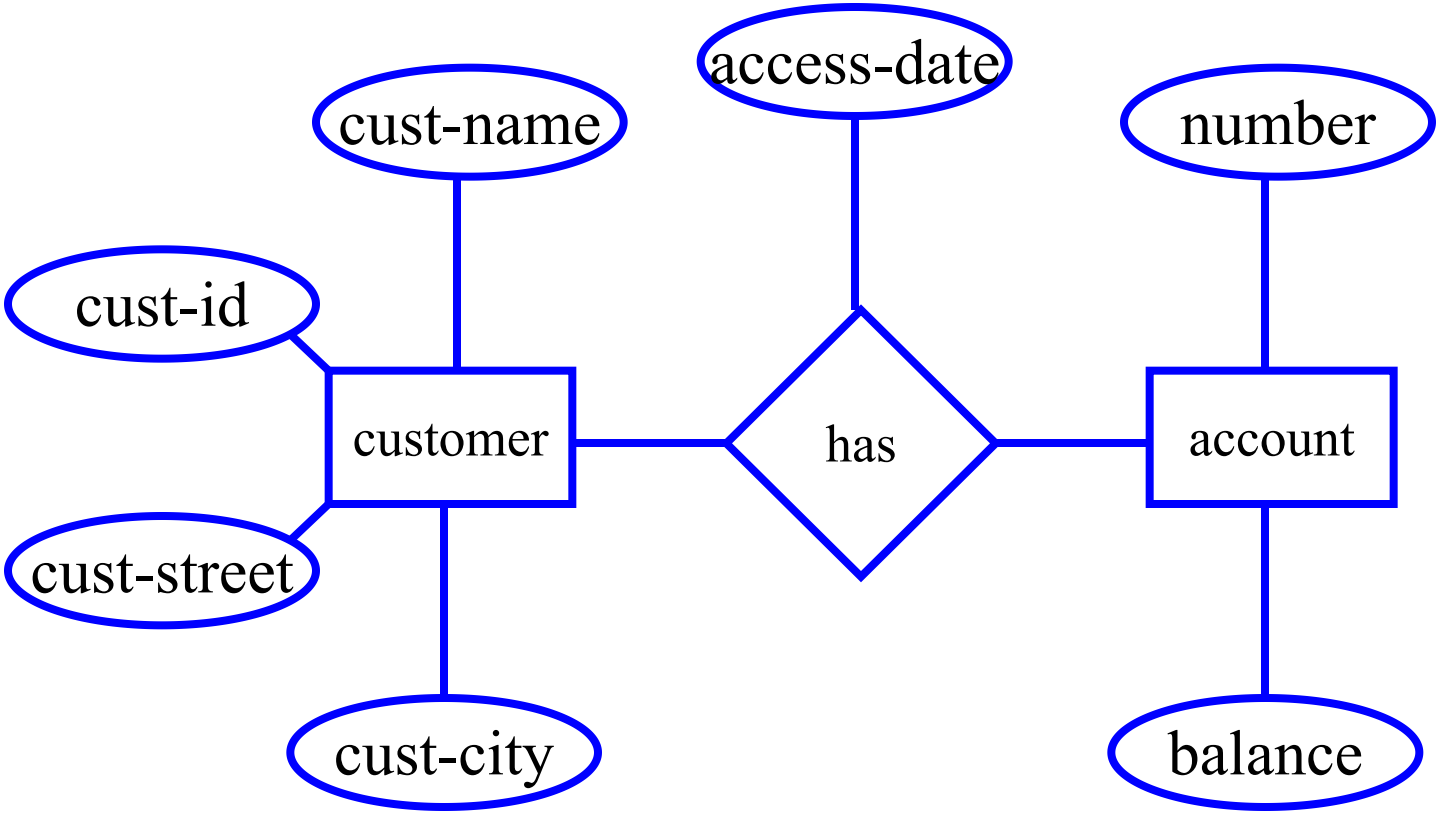
- The ***degree*** of a relationship type number of entity participate in relationship.
  - Binary (connects 2 entity types)
  - Unary/ recursive (connects 1 entity type with itself)
  - n-ary / Complex (connects 3 or more entity types)
    - Ternary (connects 3)

Degree

- Structural constraints - ***cardinality***
  - one to one (1:1)
  - one to many (1:m)
  - many to many (m:n)
- Structural constraints – ***participation***
  - Full(total) / mandatory
  - or partial /optional

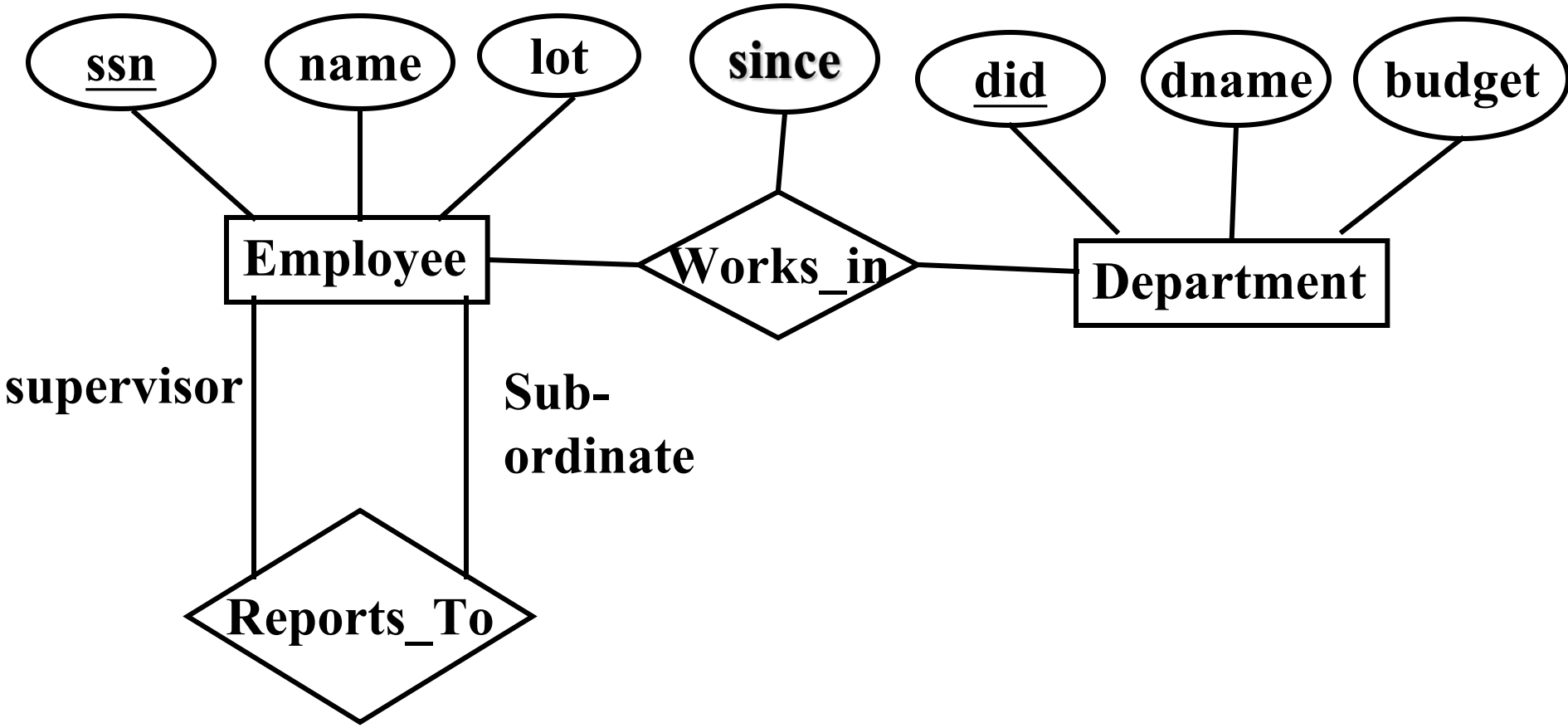
Multiplicity

ER Model: Simple Example

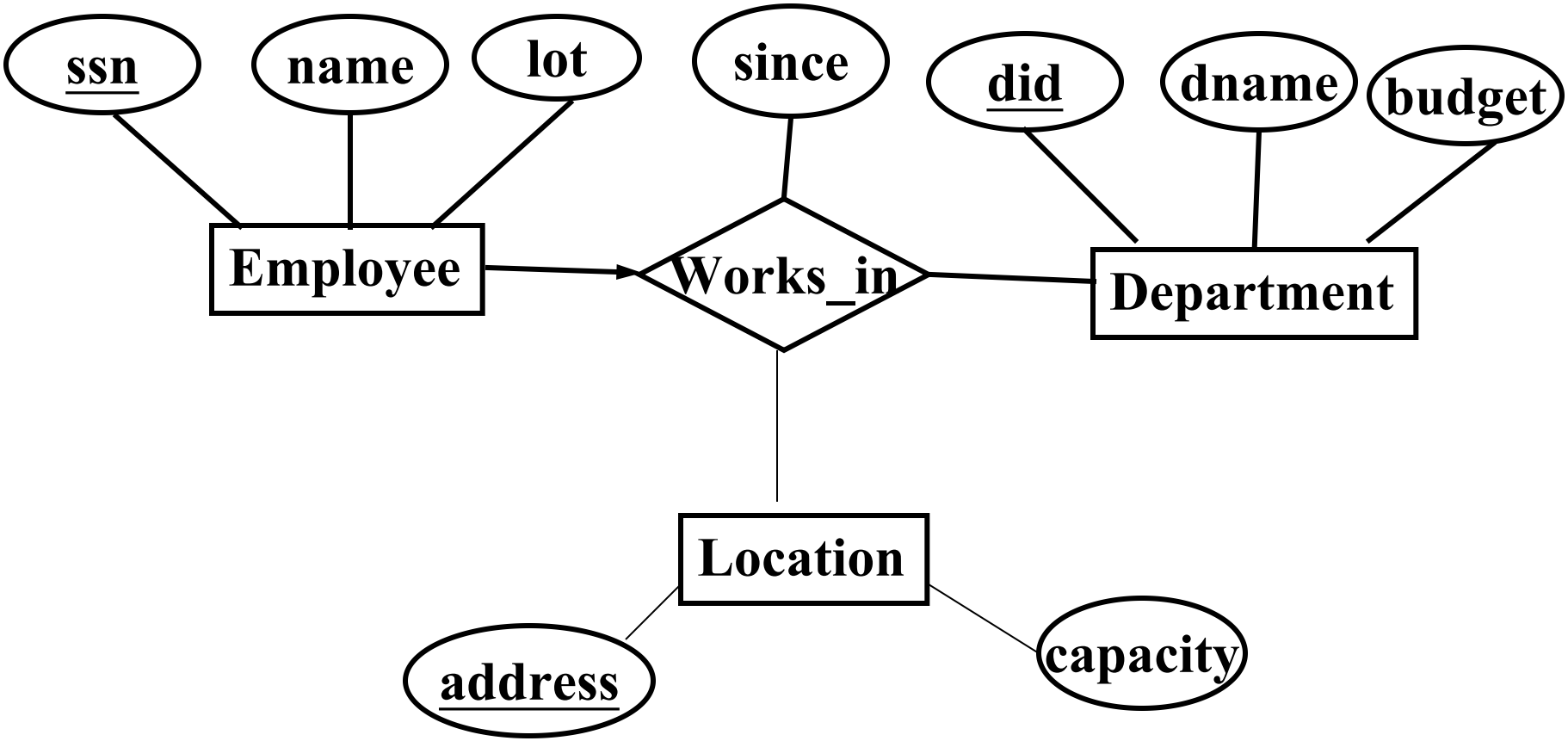


- Rectangles: entity types
- Diamonds: relationship types
- Ovals: attributes

ER Model: Another Example



Ternary Relationships



ER Model (Contd.)

Works_ In

SSN	DID	SINCE
123-22-3666	51	1/1/91
123-22-3666	56	3/3/93
231-31-5368	51	2/2/92

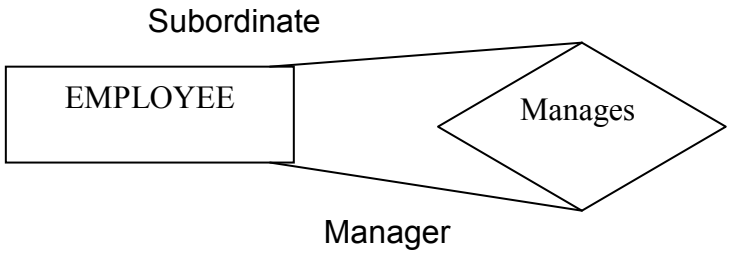
```
CREATE TABLE Works_ In(
ssn CHAR (11),
did INTEGER,
since DATE,
PRIMARY KEY (ssn, did),
FOREIGN KEY (ssn)
REFERENCES Employees,
FOREIGN KEY (did)
REFERENCES Departments)
```

Role names

- Each entity type that participates in a relationship type plays a particular role in the relationship.
- Role names may be added to make the meaning more explicit.

When to use role name?

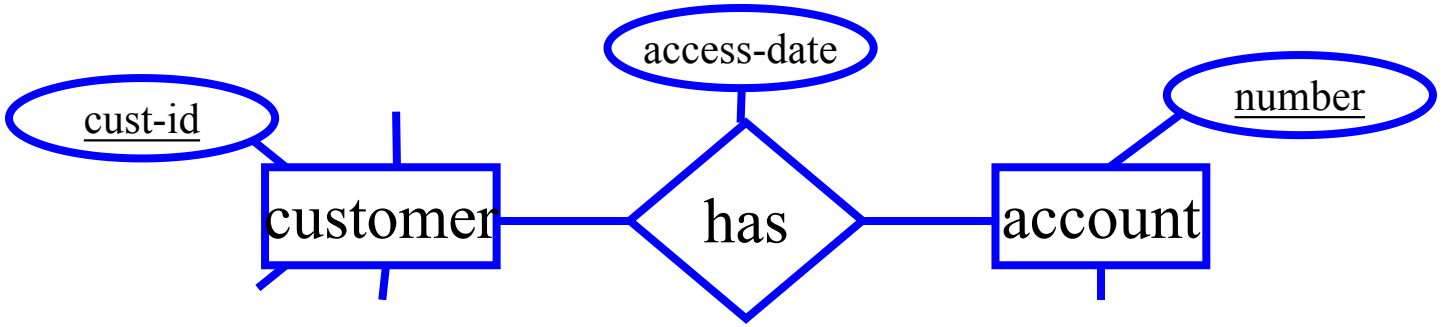
- The participating entity types are distinct, no need to mention role name.
- If the same entity type participates more than once in an r/n type in different roles.
- In such cases the role name becomes essential for distinguishing the meaning of each participation. Such r/n types are called recursive relationship.



Attributes of relationship types

- Relationship types can also have attribute similar to those of entity type.

- What attributes are needed to represent a relationship completely and uniquely ?
  - Union of primary keys of the entities involved, and relationship attributes



  - *{cust-id, access-date, account-number}* describes a relationship completely

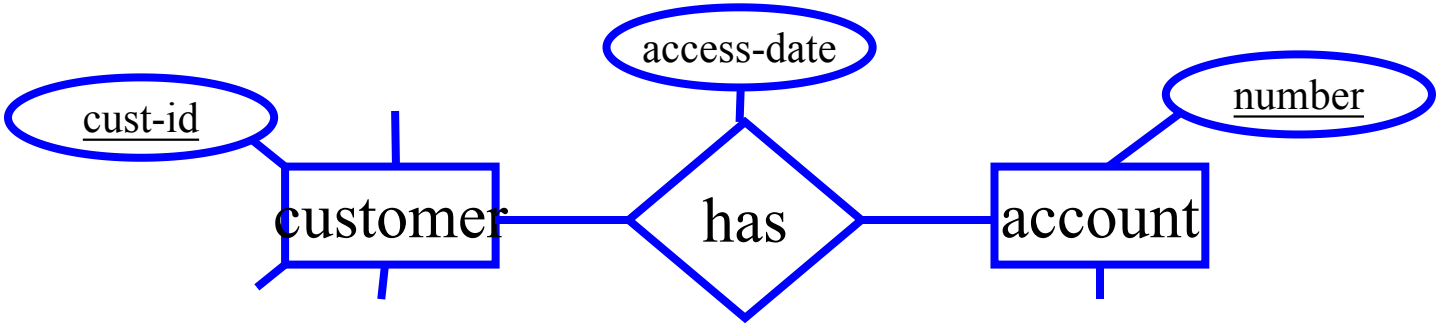
Relationship Set Keys

- ***Note: Placing the attribute in relationship type i.e 1:1, 1:N and M:M . Representing this attribute in one of the entity type.***

- General rule for binary relationships
  - one-to-one: primary key of either entity set
  - one-to-many: primary key of the entity set on the many side
  - many-to-many: union of primary keys of the associate entity sets
- n-ary relationships
  - More complicated rules

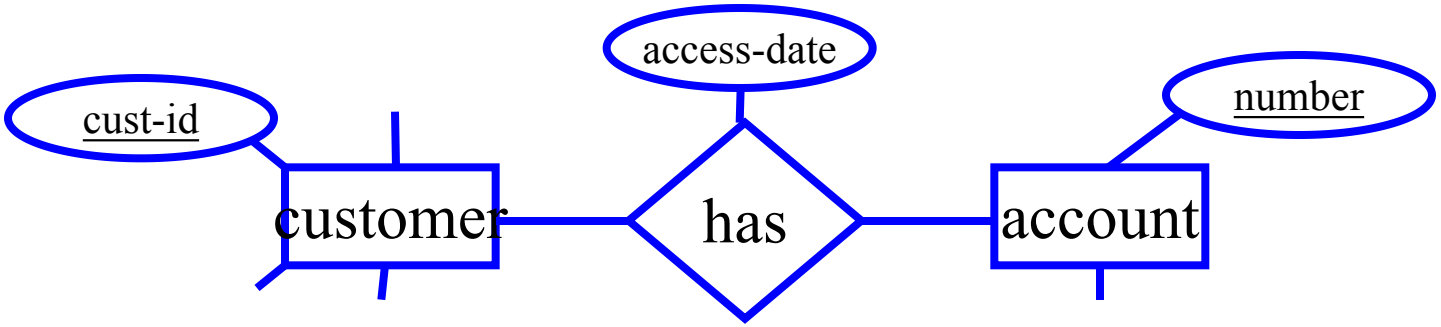
Relationship Set Keys

- Is *{cust-id, access-date, account-number}* a candidate key ?
  - No. Attribute *access-date* can be removed from this set without losing key-ness
  - In fact, union of primary keys of associated entities is always a superkey



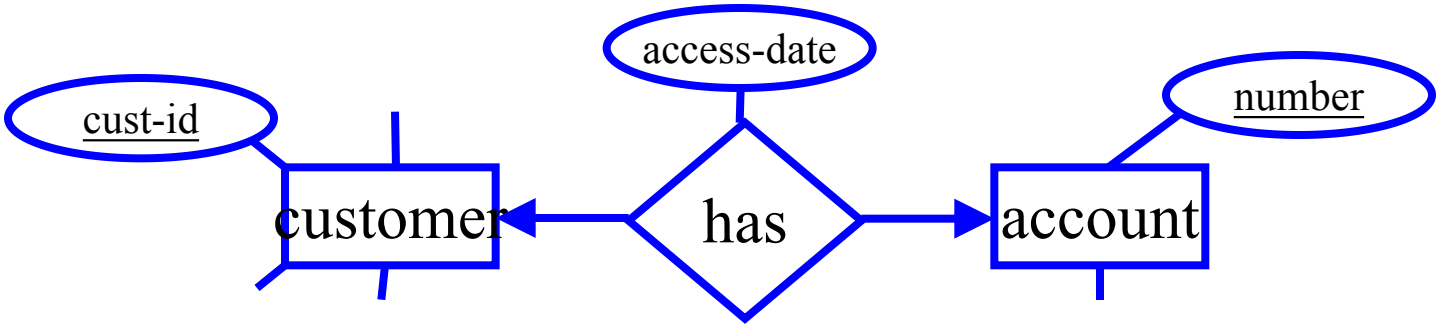
Relationship Set Keys

- Is {cust-id, account-number} a candidate key ?
  - Depends



Relationship Set Keys

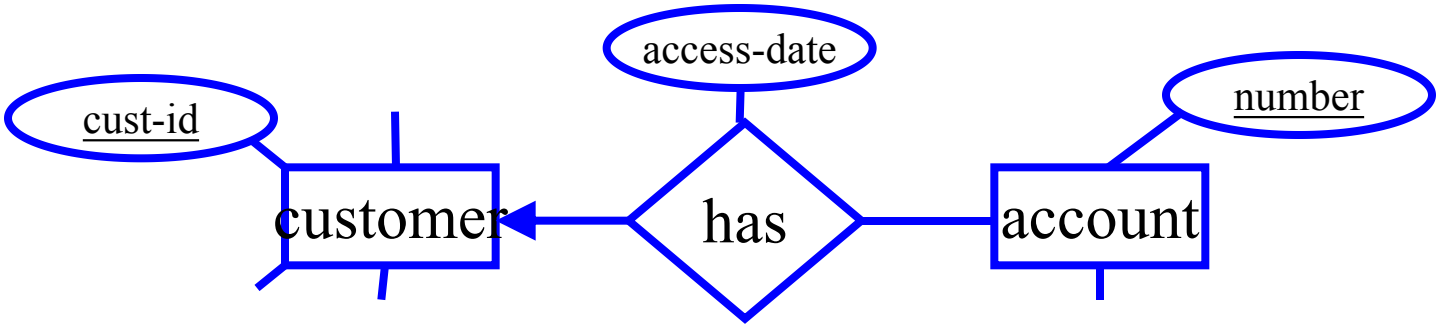
- Is {cust-id, account-number} a candidate key ?
  - Depends



- If one-to-one relationship, either *{cust-id}* or *{account-number}* sufficient
  - Since a given *customer* can only have one *account*, He / She can only participate in one relationship
  - Ditto *account*

Relationship Set Keys

- Is {cust-id, account-number} a candidate key ?
  - Depends



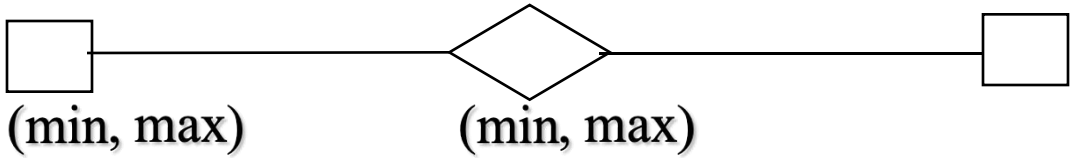
- If one-to-many relationship (as shown), *{account-number}* is a candidate key
  - A given customer can have many accounts, but at most one account holder per account allowed

Constraints on Relationships Types (constraints means restriction or limitation)

- Relationship types have certain constraints that limit the possible combination of entities that may participate in the corresponding relationship set.
- Example: a company has a rule that each employee must work for exactly one department.
- There two main types of relationship constraints
  - Cardinality ratio (Maximum Cardinality constraint)
  - Participation (also called Minimum Cardinality or participation constraint or existence dependency constraints)

Structural Constraints

- Participation
  - Do all entity instances participate in at least one relationship instance?
- Cardinality
  - How many relationship instances can an entity instance participate in?



Participation	Cardinality
0 -- Partial	1 -- one
1 -- Total (Mandatory)	M -- more than one

Constraints ratio for binary Relationship Types

- (Also known as ratio constraints)
- Cardinality ratio (Maximum Cardinality) for a binary relationship specifies the maximum number of relationship instances that an entity can participate in.
    - One-to-one (1:1)
    - One-to-many (1:N)
    - Many-to-one (N:1)
    - Many-to-many (M:N)

Participation constraint

- It specifies whether the existence of an entity depends on its being related to another entity the relationship types.
- This constraint specifies the minimum number of relationship instances that each entity can participate in. This type is called **Minimum Cardinality constraint**.

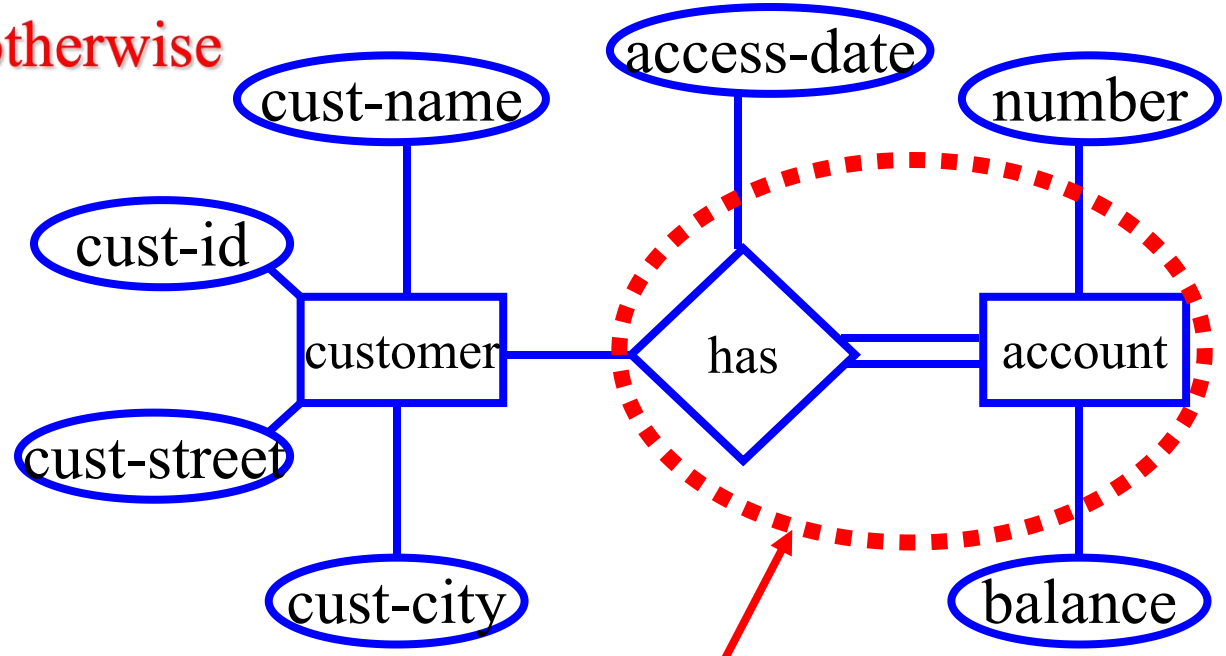
There are two types of Participation constraint

1.Total Participation constraint (or existence dependency):

2.Partial Participation constraint

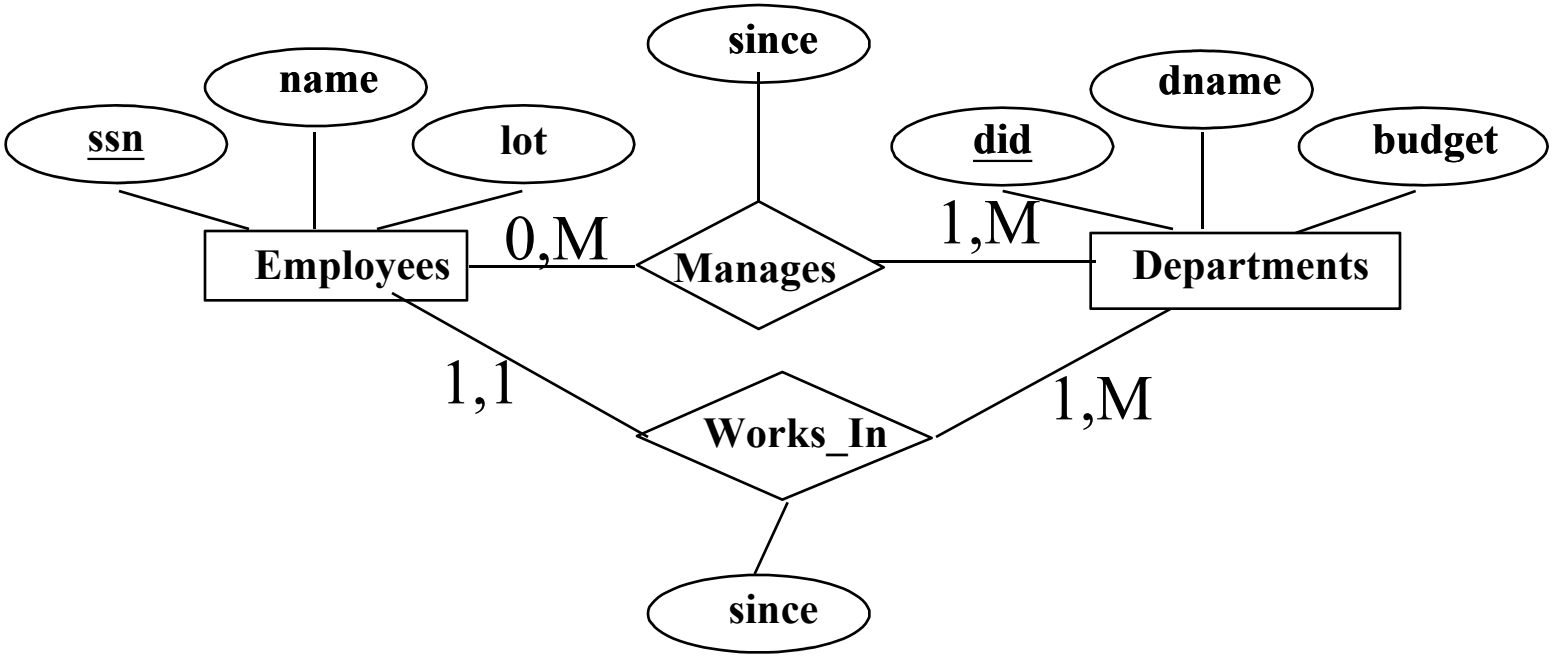
Participation Constraint

- Given an entity set E, and a relationship R it participates in:
  - If every entity in E participates in at least one relationship in R, it is **total participation**
  - partial otherwise



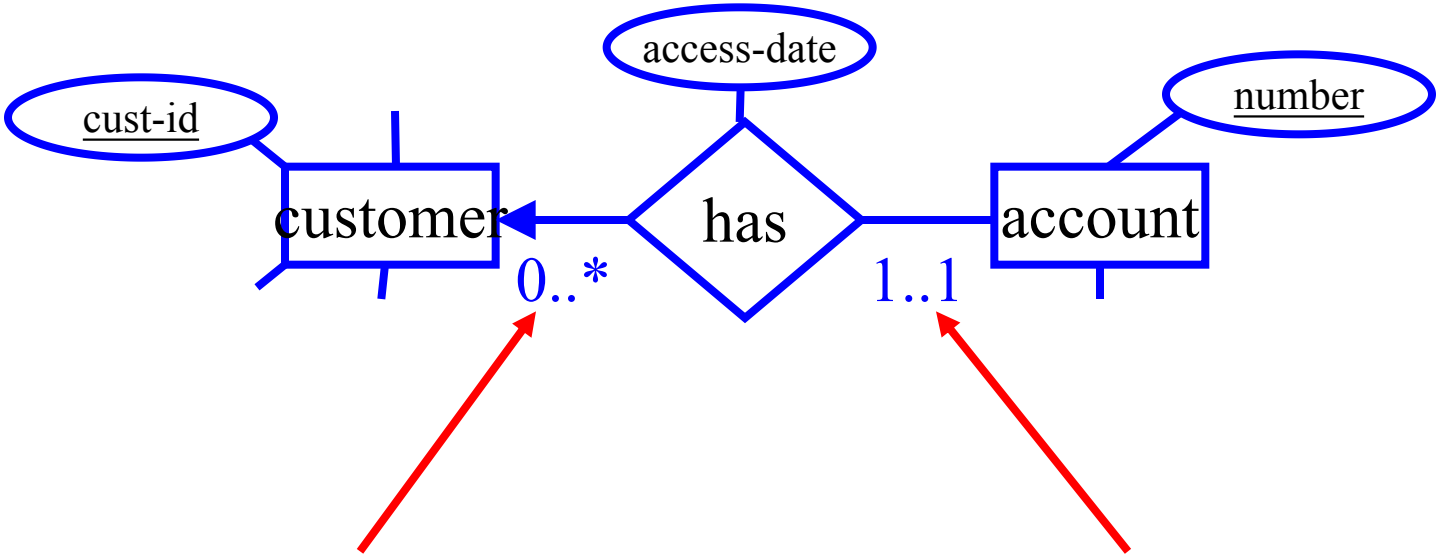
Participation Constraints

- Does every department have a manager?
  - If so, this is a *participation constraint*: the participation of Departments in Manages is said to be *total* (vs. *partial*).
    - Every *did* value in Departments table must appear in a row of the Manages table (with a non-null *ssn* value!)



Cardinality Constraints

How many relationships can an entity participate in ?

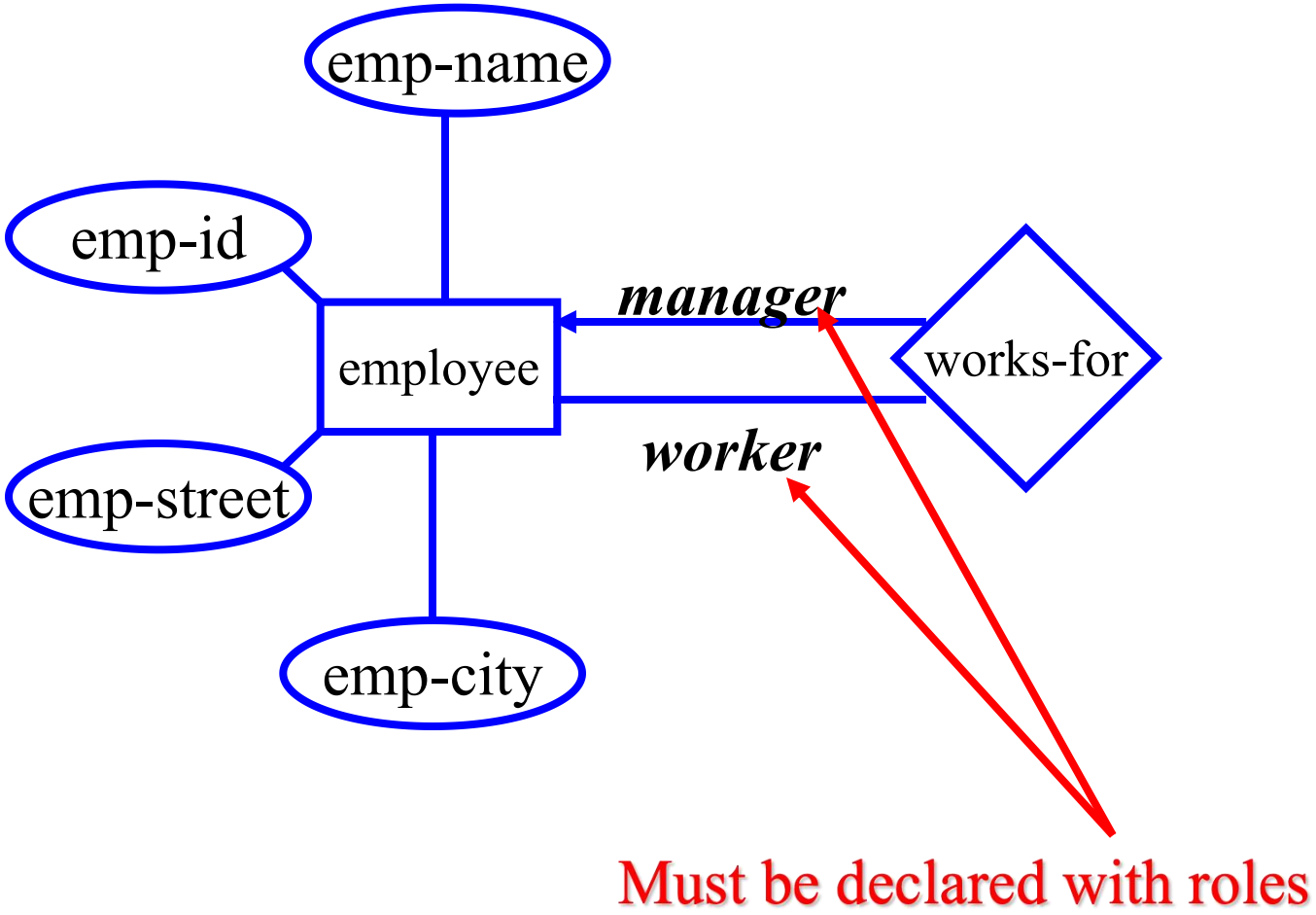


Minimum - 0
Maximum – no limit

Minimum - 1
Maximum - 1

Recursive Relationships

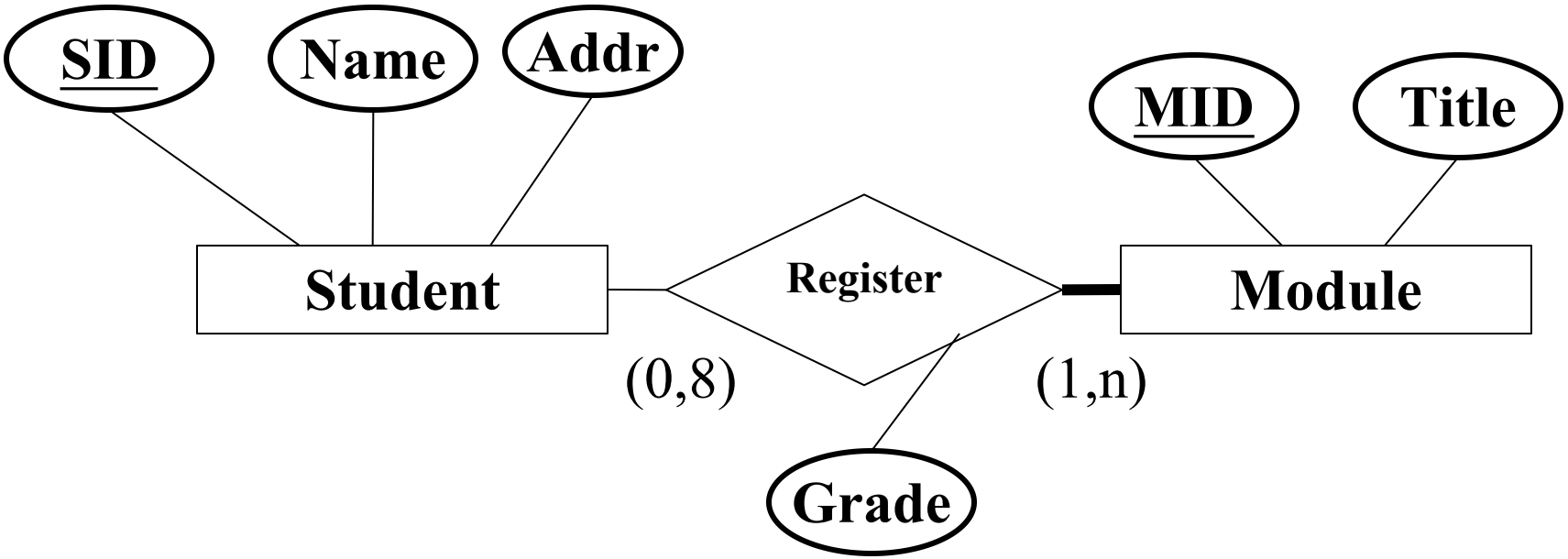
- Sometimes a relationship associates an entity set to itself



Over to You now!

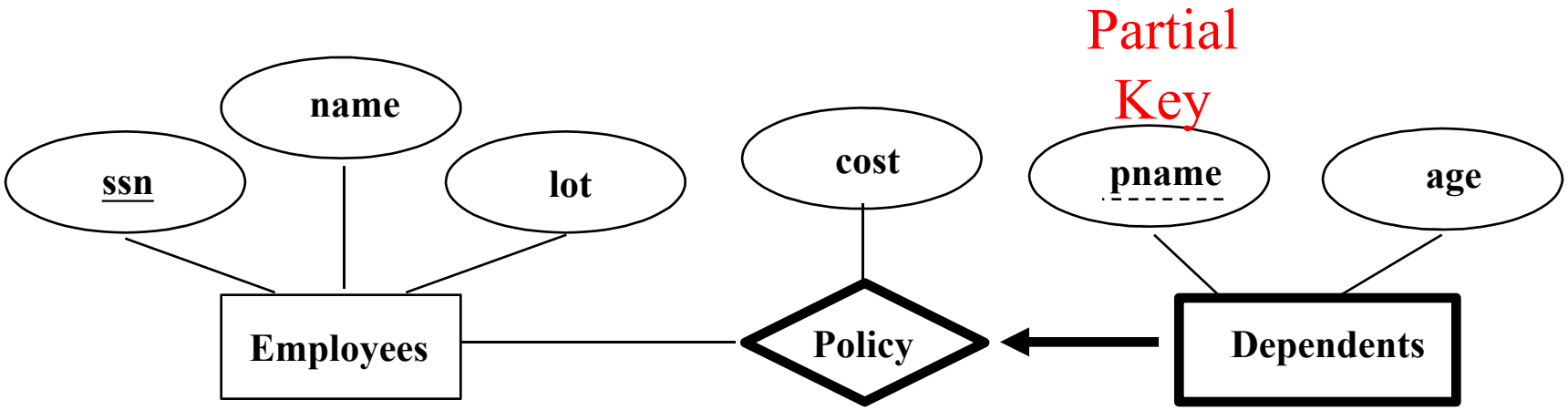
- See if you can draw an E-R diagram for this scenario – you are already familiar with this!
  - "A student registers for up to 8 modules and each module has many students on it. Record the student ID, their full name and address and also each module ID and title. We also want to hold the grade attained by each student for each module"
    - Remember to show in your model:
      - All primary keys,
      - Entities
      - Relationship
      - Attributes

Solution !!!!!



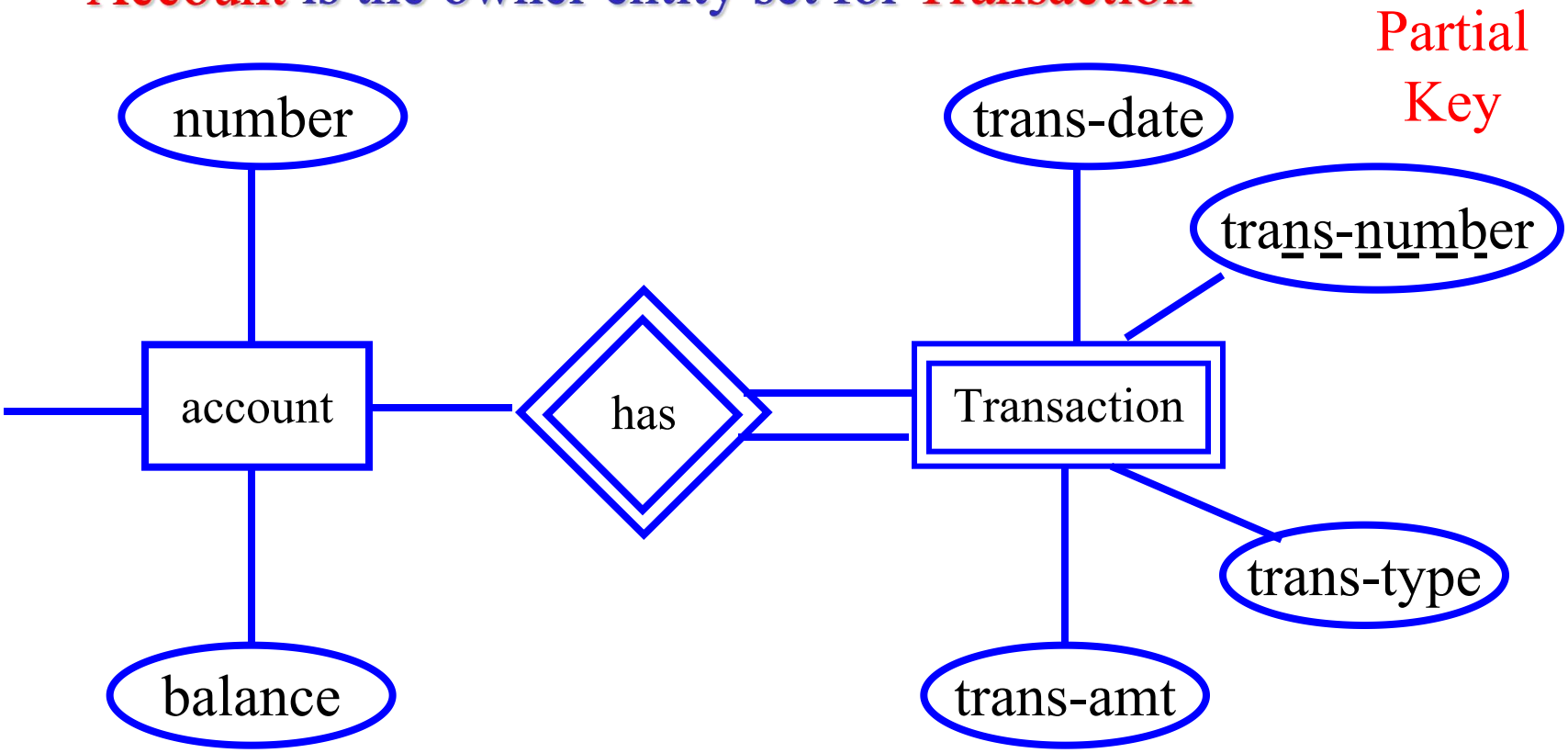
Weak Entities

- A *weak entity* can be identified uniquely only by considering the primary key of another (*owner*) entity.
  - Owner entity set and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities).
  - Weak entity set must have total participation in this *identifying* relationship set.

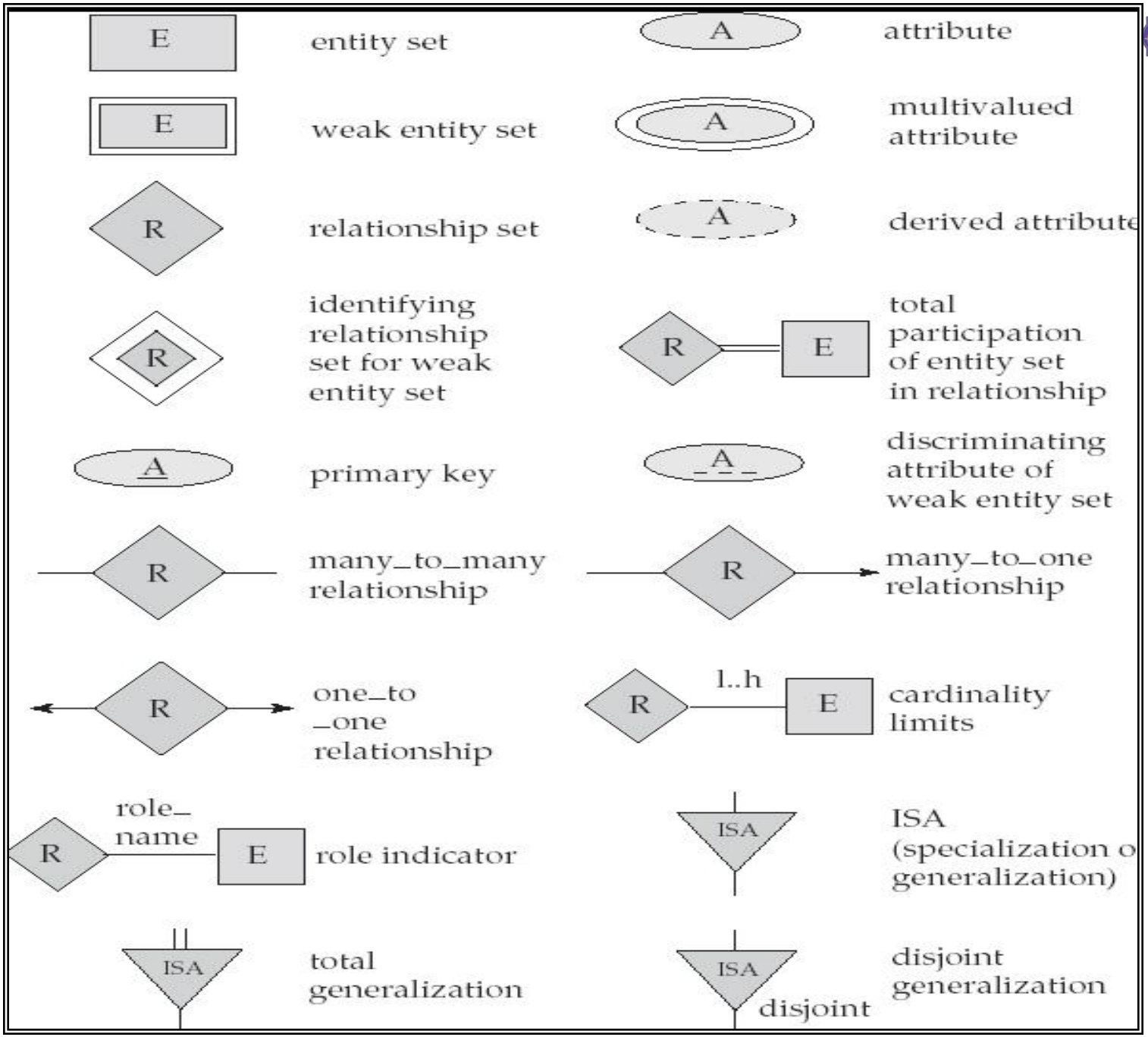


Weak Entity Sets

- A weak entity set must be associated with an identifying or owner entity set
- Account is the owner entity set for Transaction



Summary of Notations in ER Model



ER Model – Design Issues

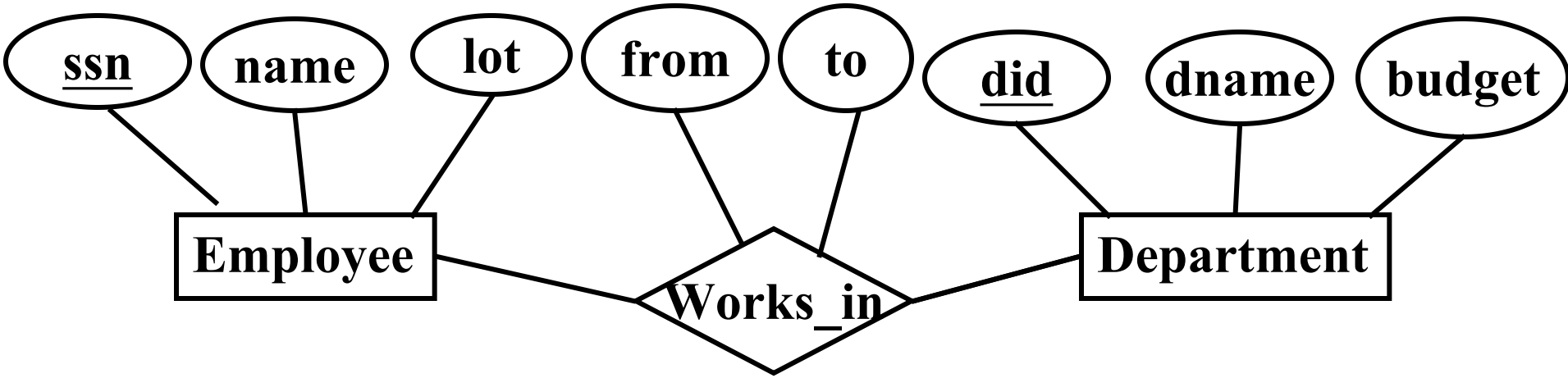
- Design choices:
  - Should a concept be modeled as an entity or an attribute?
  - Should a concept be modeled as an entity or a relationship?
  - Identifying relationships: Binary or ternary? Aggregation?
- Constraints in the ER Model:
  - A lot of data semantics can (and should) be captured.
  - But some constraints cannot be captured in ER diagrams.

Entity vs. Attribute

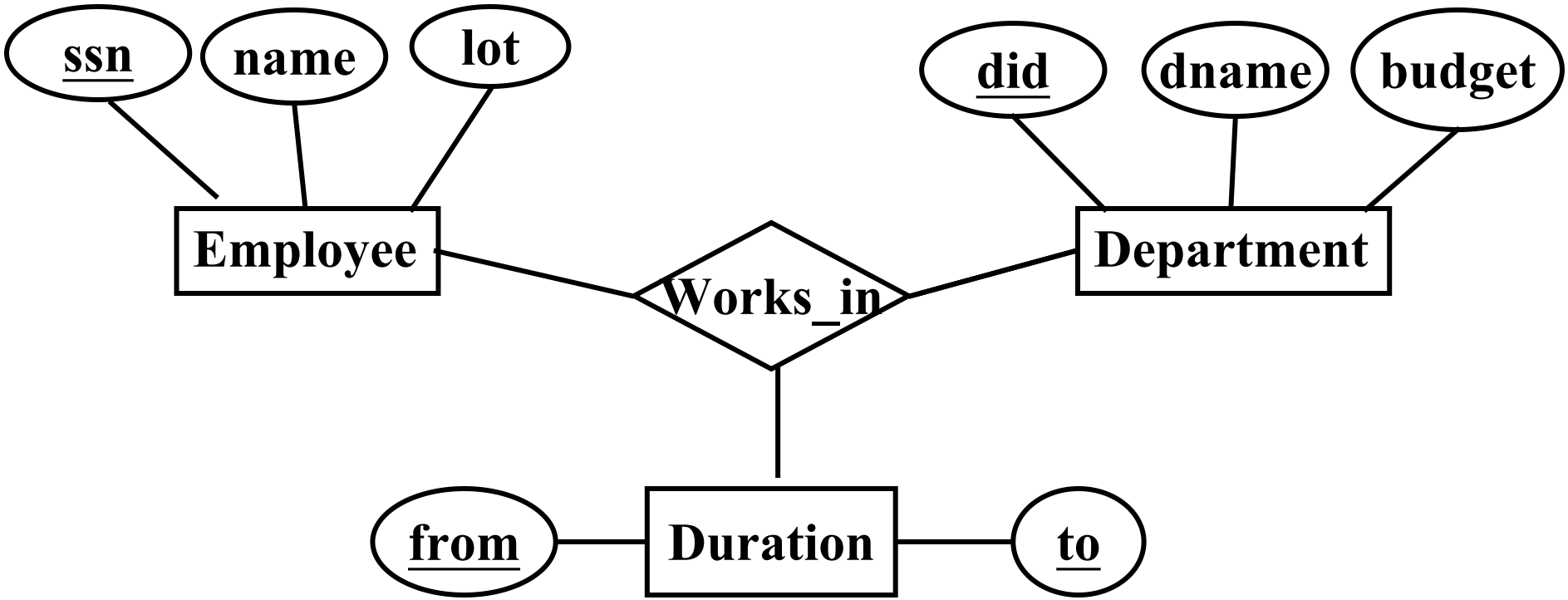
- Should *address* be an attribute of Employees or an entity (connected to Employees by a relationship)?
- Depends upon the use we want to make of address information, and the semantics of the data:
  - If we have several addresses per employee, *address* must be an entity (since attributes cannot be set-valued).
  - If the structure (city, street, etc.) is important, e.g., we want to retrieve employees in a given city, *address* must be modeled as an entity (since attribute values are atomic).

Entity vs. Attribute

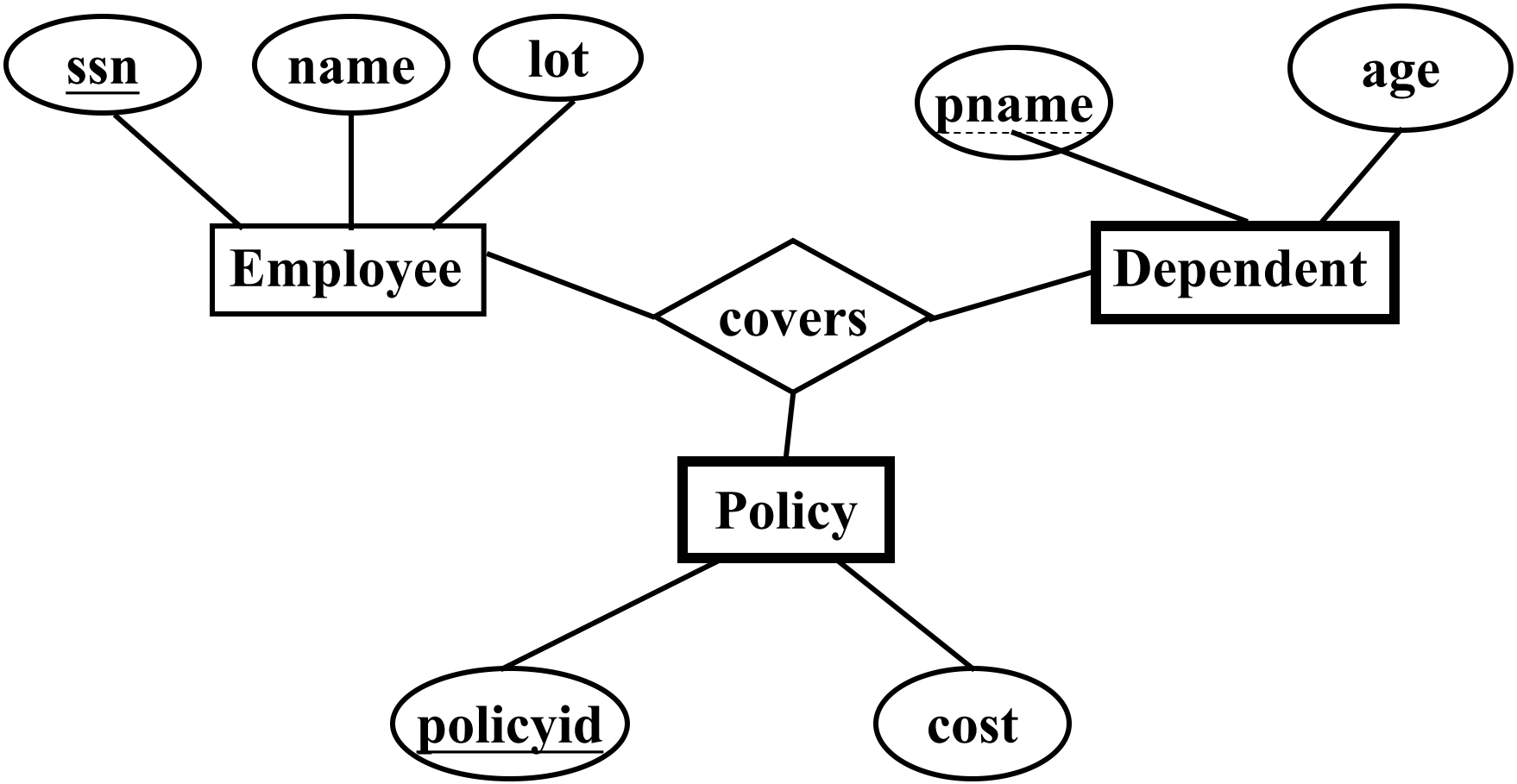
Works_In does not allow an employee to work in a department for two or more periods (why?)



Entity vs. Attribute (Contd.)

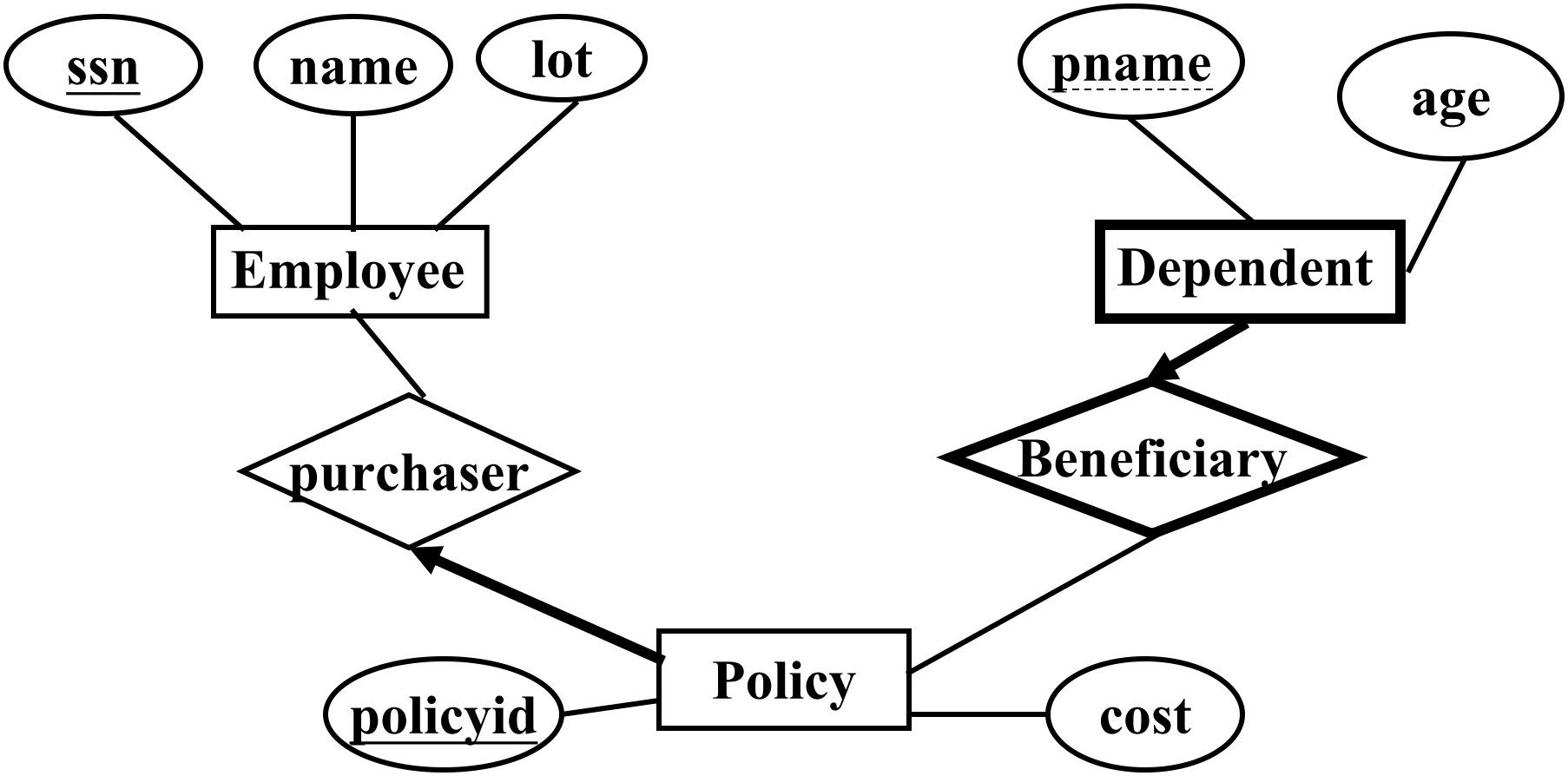


Binary vs. Ternary Relationships



Binary vs. Ternary Relationships

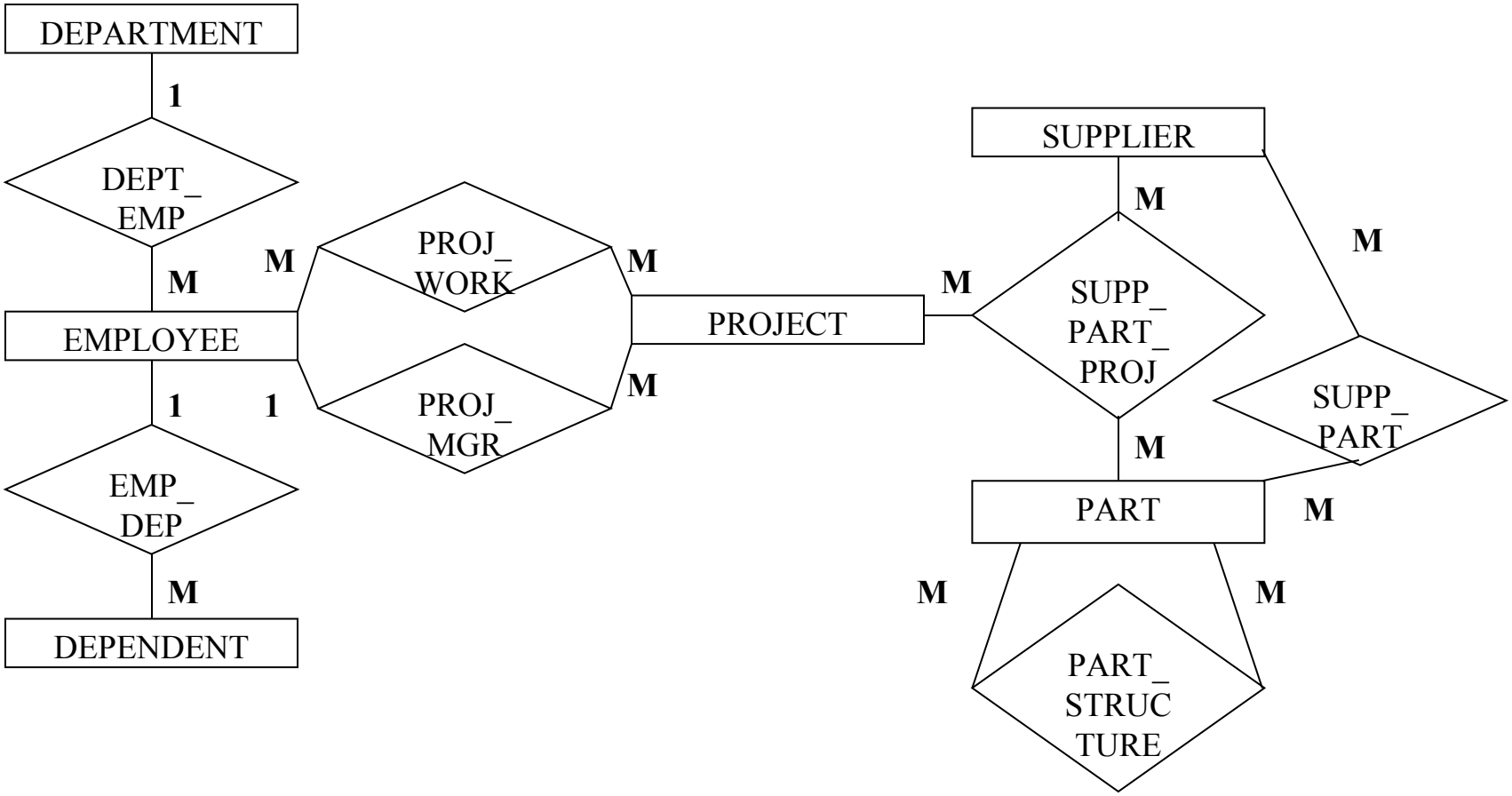
Better Design



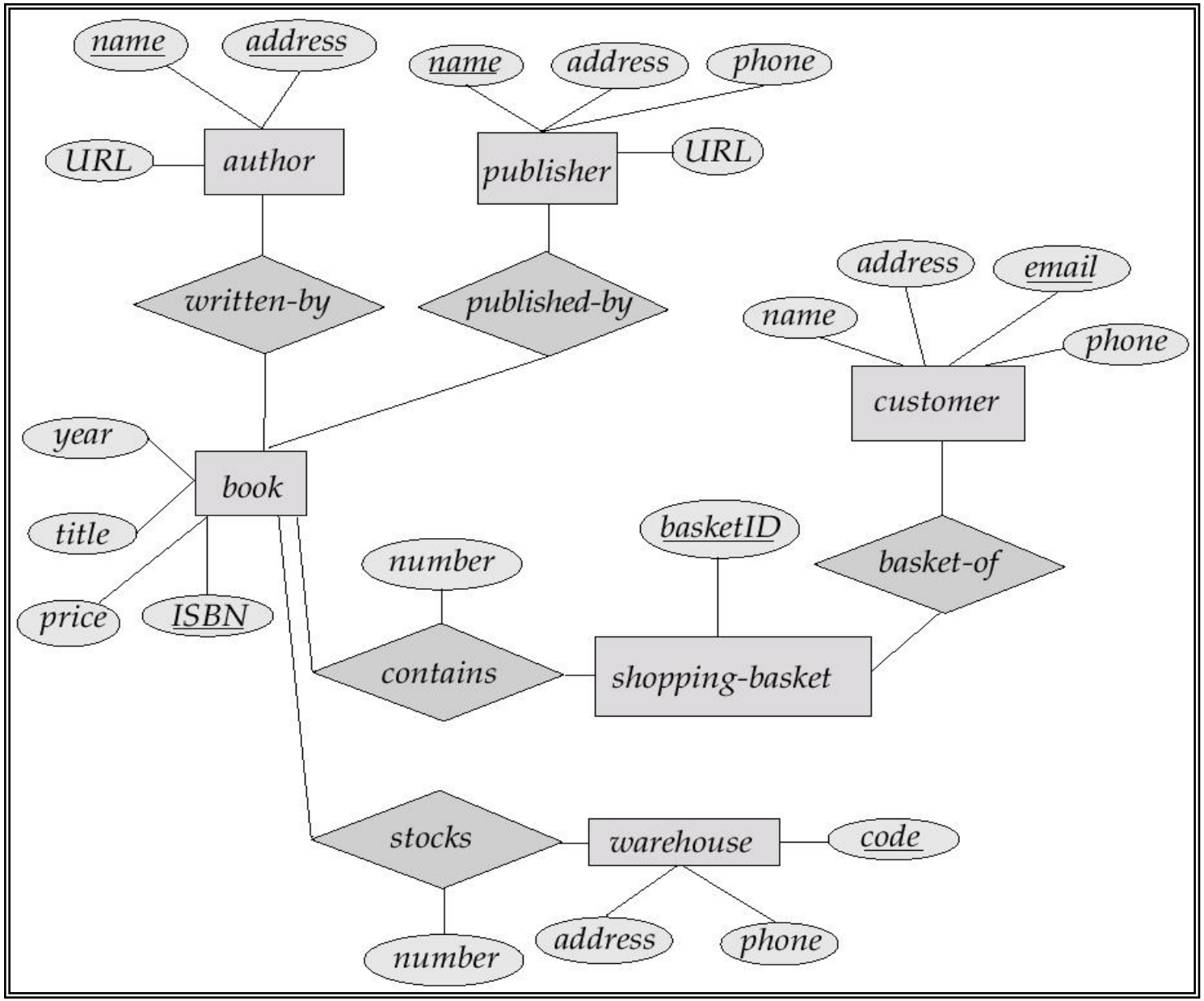
Constraints Beyond the ER Model

- Some constraints cannot be captured in ER diagrams:
  - Functional dependencies
  - Inclusion dependencies
  - General constraints

E-R Diagram : Example



ER Diagram – Internet Book Shop



Exercise – ER Model….

- An Example Database Application called COMPANY which serves to illustrate the ER Model concepts and their schema design.

The following are collection from the Client.

Analysis…

- Company: Organized into Departments, Each Department has a name, no and manager who manages the department. The Company keeps track of the date that employee managing the department. A Department may have a Several locations.

Analysis…

- Department:
  A Department controls a number of Projects each of which has a unique name , no and a single Location.
- Employee:

  Name, Age, Gender, BirthDate, SSN, Address, Salary.
  An Employee is assigned to one department, may work on several projects which are not controlled by the department. Track of the number of hours per week is also controlled.
- Keep track of the dependents of each employee for insurance policies: We keep each dependent first name, gender, Date of birth and relationship to the employee.

Now to our Company…

DEPARTMENT

(Name, Number, { Locations }, Manager, Start Date)

PROJECT

(Name, Number, Location, Controlling Department)

EMPLOYEE

(Name (Fname, Lname), SSN, Gender, Address, Salary
Birthdate, Department, Supervisor, (Workson (Project , Hrs))

DEPENDENT

(Employee, Name, Gender, Birthdate, Relationship)

Example …Relationships

- Manage:
  - Department and Employee
  - Partial Participation
  - Relation Attribute: StartDate.
- Works For:
  - Department and Employee
  - Total Participation

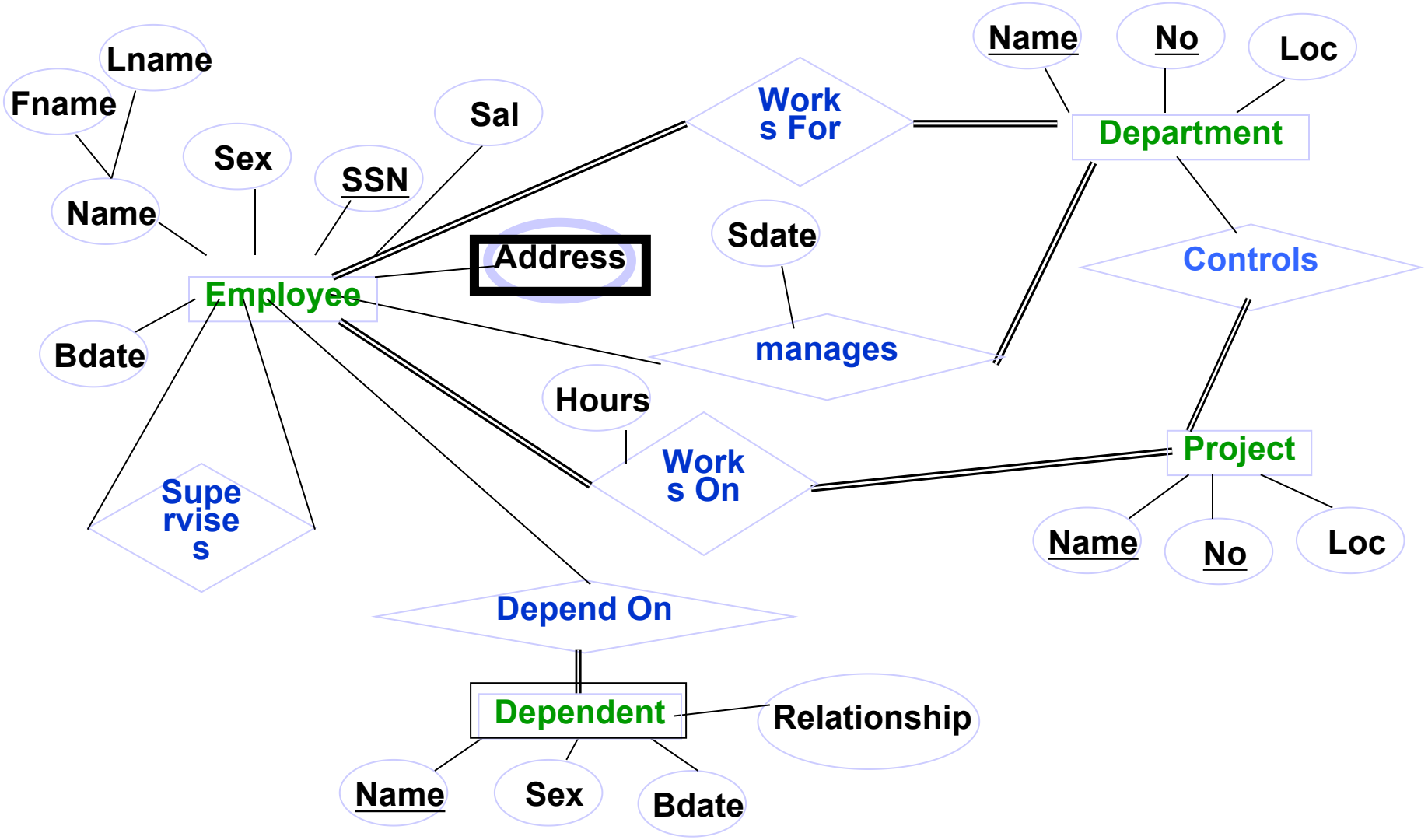
Example…

- Control :
  - Department , Project
  - Partial Participation from Department
  - Total Participation from Project
- Supervisor :
  - Employee, Employee
  - Partial and Recursive

Example …

- Works – On :
  - Project , Employee
  - Total Participation
  - Hours Worked is a RKA.
- Dependants of:
  - Employee, Dependant
  - Dependant is a Weaker entity
  - Dependant is Total , Employee is Partial.

One Possible mapping of the Problem Statement



Company Schema with Structural Constraints

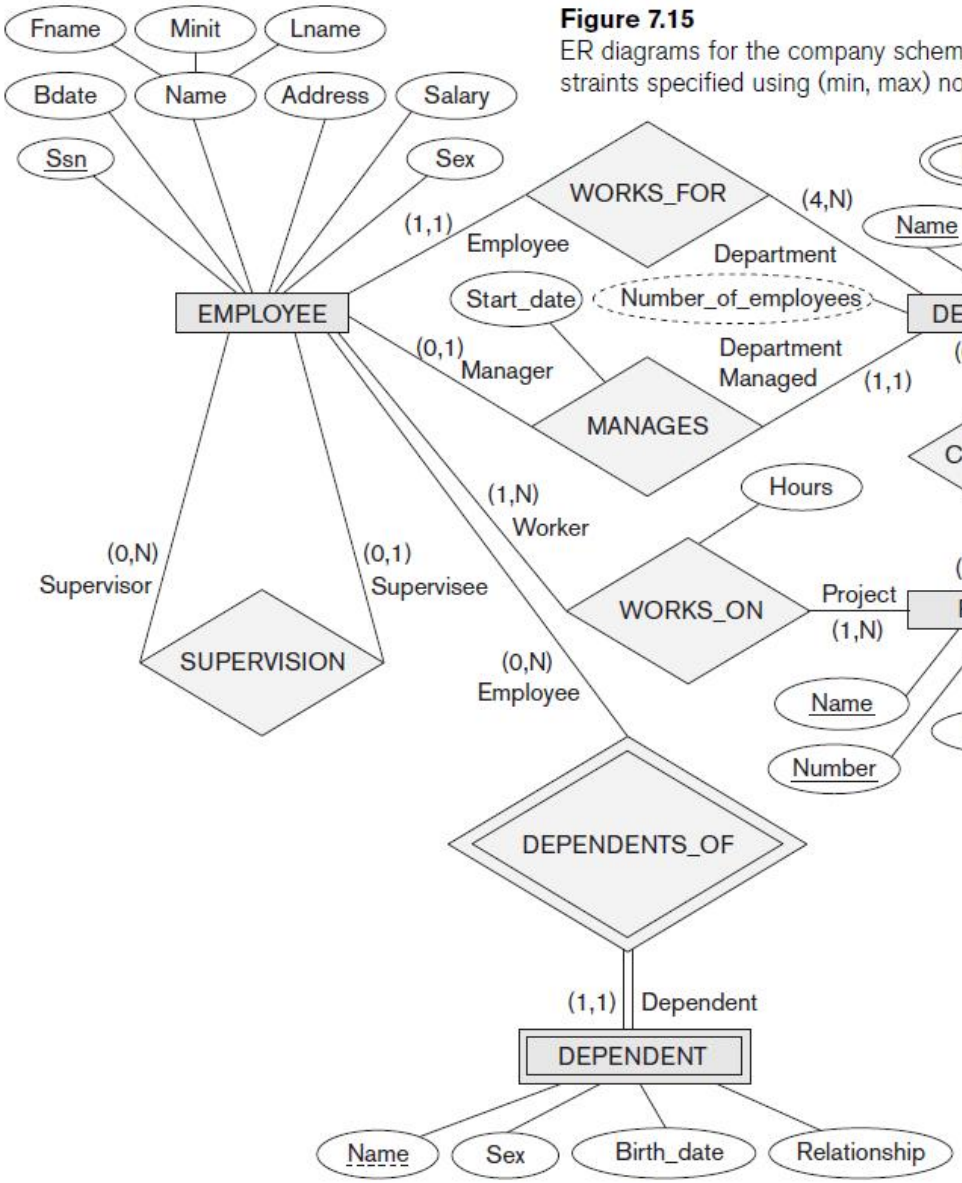


Figure 7.15
ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names.

Algorithm for ER-to-Relational mapping

Step 1: Mapping of Regular Entity Types

Step 2: Mapping of Weak Entity Types

Step 3: Mapping of Binary 1:1 Relationship Types

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

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

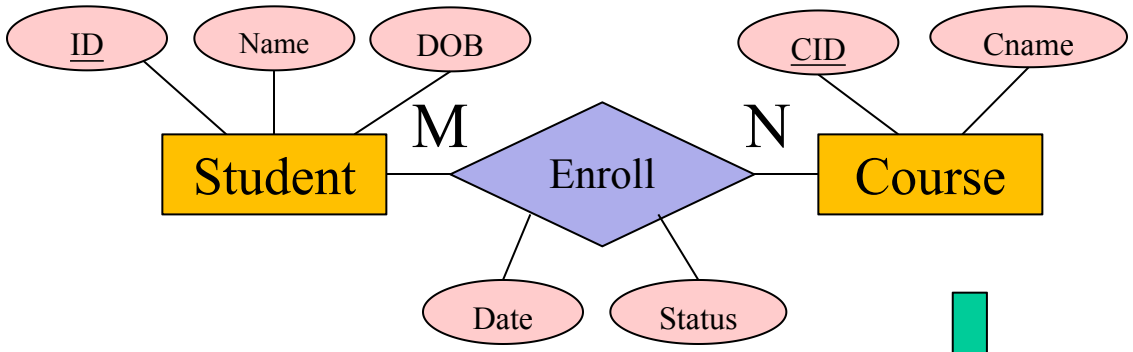
Step 6: Mapping of Multivalued/Composite Attributes

Step 7: Mapping of *N*-ary Relationship Types

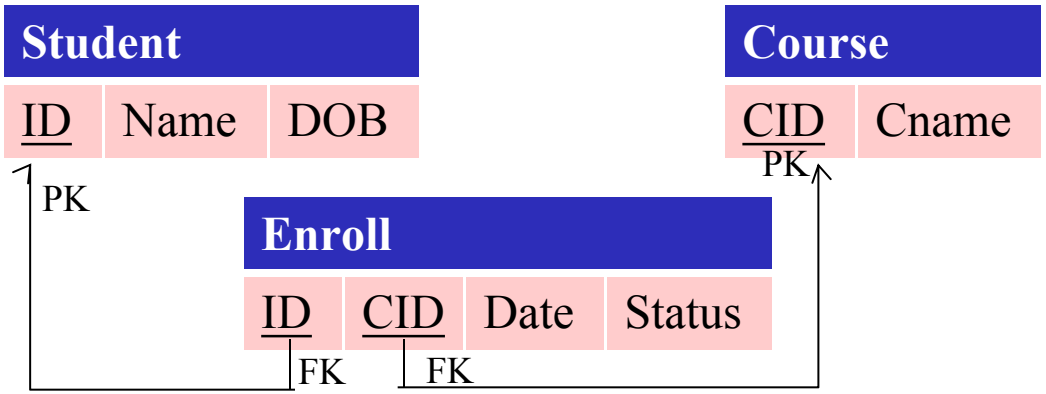
Correspondence between ER and Relational Models

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

ER Model to Relational Model



ER to Relational mapping

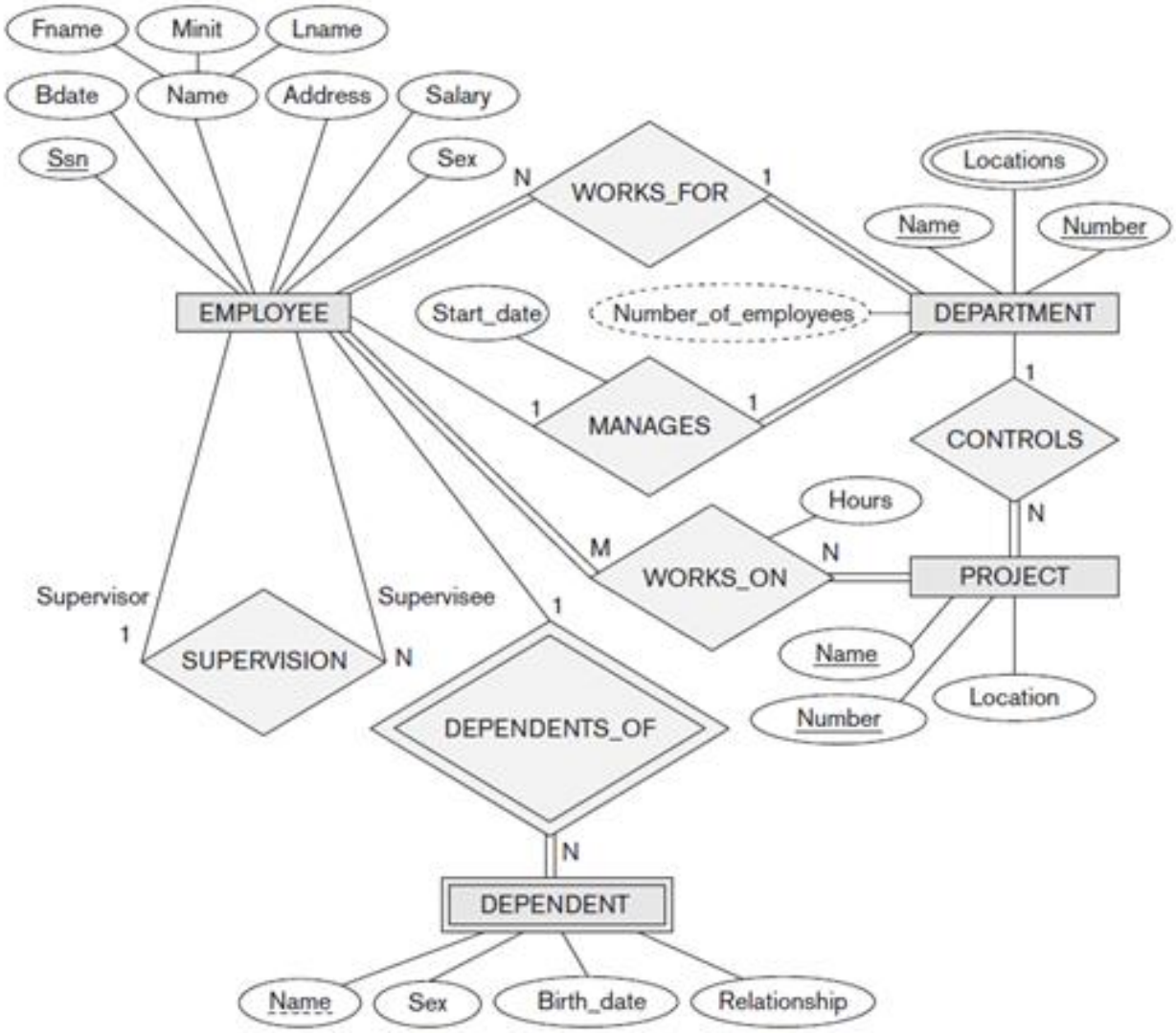


Relational to Physical – low level schema

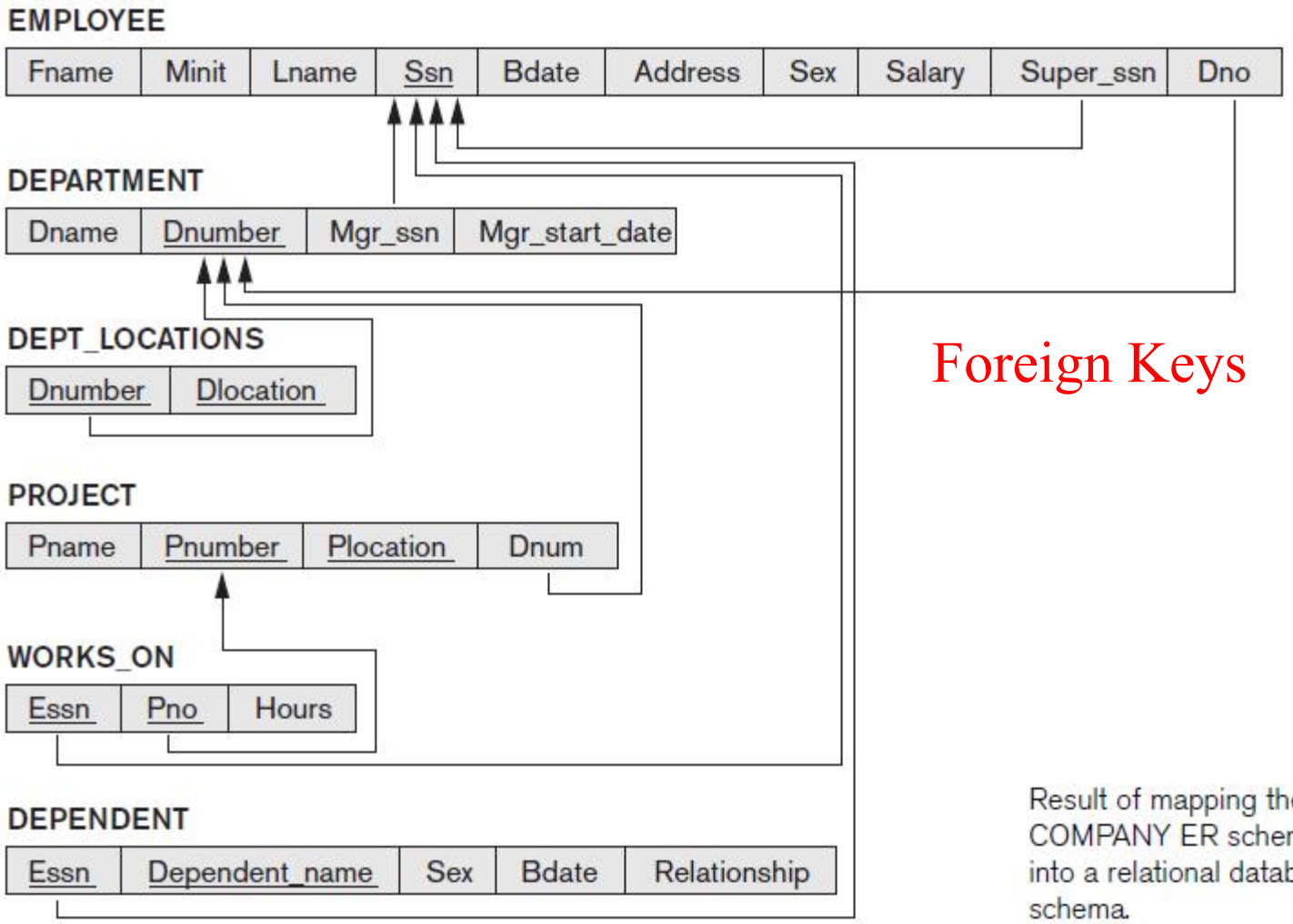
```
Create Table Student( ID Number(3) primary key,
                      Name Varchar2(20) not null,
                      DOB date);

Create Table Course( CID Number(3) primary key,
                     Cname Varchar2(20) not null);

Create Table Enroll( ID Number(3) foreign key references Student(ID),
                     CID Number(3) foreign  references Course(CID),
                     Status Char,
                     Primary key(ID,CID));
```

ER Schema to Relational Schema mapping

The ER conceptual schema diagram for the COMPANY database.



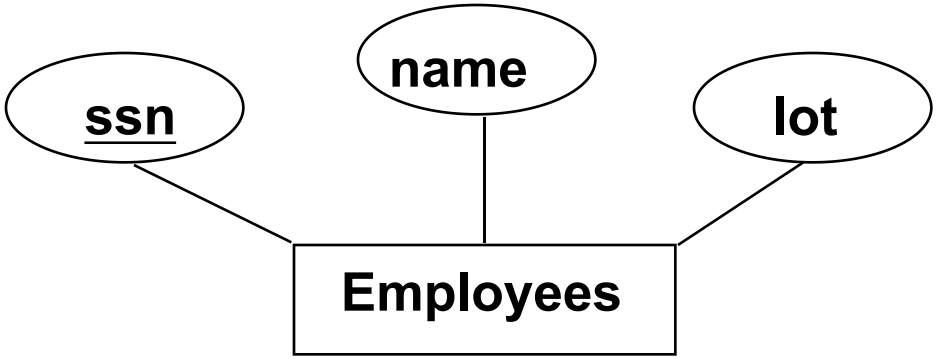
Relational Schema corresponding to Company ER Schema



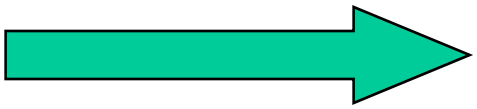
Result of mapping the COMPANY ER schema into a relational database schema.

ER to Relational to Physical

- Entity sets to tables.



ssn	name	lot
123-22-3666	Attishoo	48
231-31-5368	Smiley	22
131-24-3650	Smethurst	35



```
CREATE TABLE Employees
(ssn CHAR(11),
name CHAR(20),
lot  INTEGER,
PRIMARY KEY  (ssn))
```

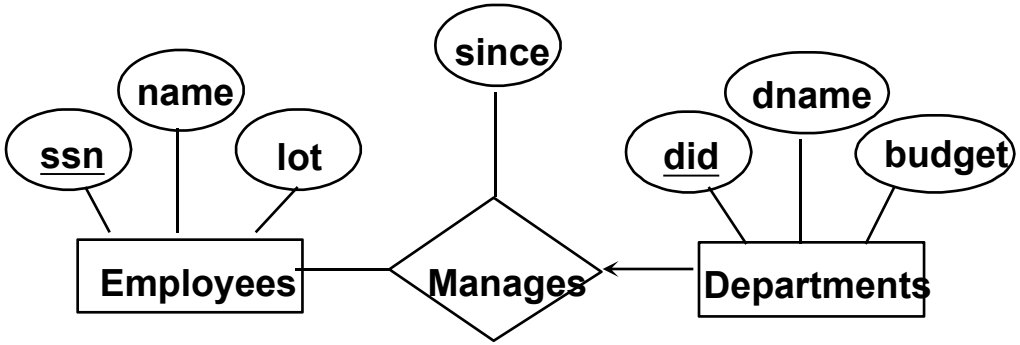
Relationship Sets to Tables

- In translating a many-to-many relationship set to a relation, attributes of the relation must include:
  1) Keys for each participating entity set (as foreign keys). This set of attributes forms a *superkey* for the relation.
  2) All descriptive attributes.

```
CREATE TABLE Works_In(
ssn  CHAR(1),
did  INTEGER,
since  DATE,
PRIMARY KEY (ssn, did),
FOREIGN KEY (ssn)
REFERENCES Employees,
FOREIGN KEY (did)
REFERENCES Departments)
```

ssn	did	since
123-22-3666	51	1/1/91
123-22-3666	56	3/3/93
231-31-5368	51	2/2/92

Translating ER with Key Constraints



- Since each department has a unique manager, we could instead combine Manages and Departments.

```
CREATE TABLE  Manages(
 ssn  CHAR(11),
 did  INTEGER,
 since  DATE,
 PRIMARY KEY  (did),
 FOREIGN KEY (ssn)
REFERENCES Employees,
   FOREIGN KEY (did)
REFERENCES Departments)
```

Vs.

```
CREATE TABLE  Dept_Mgr(
 did  INTEGER,
 dname  CHAR(20),
 budget  REAL,
 ssn  CHAR(11),
 since  DATE,
 PRIMARY KEY  (did),
 FOREIGN KEY (ssn)
  REFERENCES Employees)
```

Translating Weak Entity Sets

- Weak entity set and identifying relationship set are translated into a single table.
  - When the owner entity is deleted, all owned weak entities must also be deleted.

```
CREATE TABLE  Dep_Policy (
   pname  CHAR(20),
   age  INTEGER,
   cost  REAL,
   ssn  CHAR(11) NOT NULL,
   PRIMARY KEY  (pname, ssn),
   FOREIGN KEY  (ssn) REFERENCES Employees,
      ON DELETE CASCADE)
```

Summary of Conceptual Design

- *Conceptual design* follows *requirements analysis*,
  - Yields a high-level description of data to be stored
- ER model popular for conceptual design
  - Constructs are expressive, close to the way people think about their applications.
- Basic constructs: *entities*, *relationships*, and *attributes* (of entities and relationships).
- Some additional constructs: *weak entities*, *ISA hierarchies*, and *aggregation*.
- Note: There are many variations on ER model.

Summary of ER (Contd.)

- Several kinds of integrity constraints can be expressed in the ER model: *key constraints*, *participation constraints*
- Some constraints (notably, *functional dependencies*) cannot be expressed in the ER model.
  - Constraints play an important role in determining the best database design for an enterprise.

Key Points

- Entities (strong, weak)
- Attributes (simple/composite, single valued/ multivalued, stored/derived, key)
- Relationships
  - Degree (1,2,3,..,n)
  - Cardinality(1:1,1:M,M:1,M:N)
  - Participation (Total, Partial)

Relational Data Model

Objectives

- Introduction
- Relational Model Concepts
- Some Important Terms
- Properties of Relations
- Keys
- Referential Integrity
- Relational Algebra
- Example Queries

Introduction

- The Relational Model is based on the concept of a **Relation**.
- Represents Database as a collection of **Relations**
- A **Relation** is a mathematical concept based on the ideas of sets. The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations.
- The model was first proposed by Dr. E F Codd of IBM in 1970 in the following paper:

  "A Relational Model for Large Shared Data Banks," Communications of the ACM, June 1970.

- The above paper caused a major revolution in the field of Database management and earned Ted Codd the coveted ACM Turing Award.

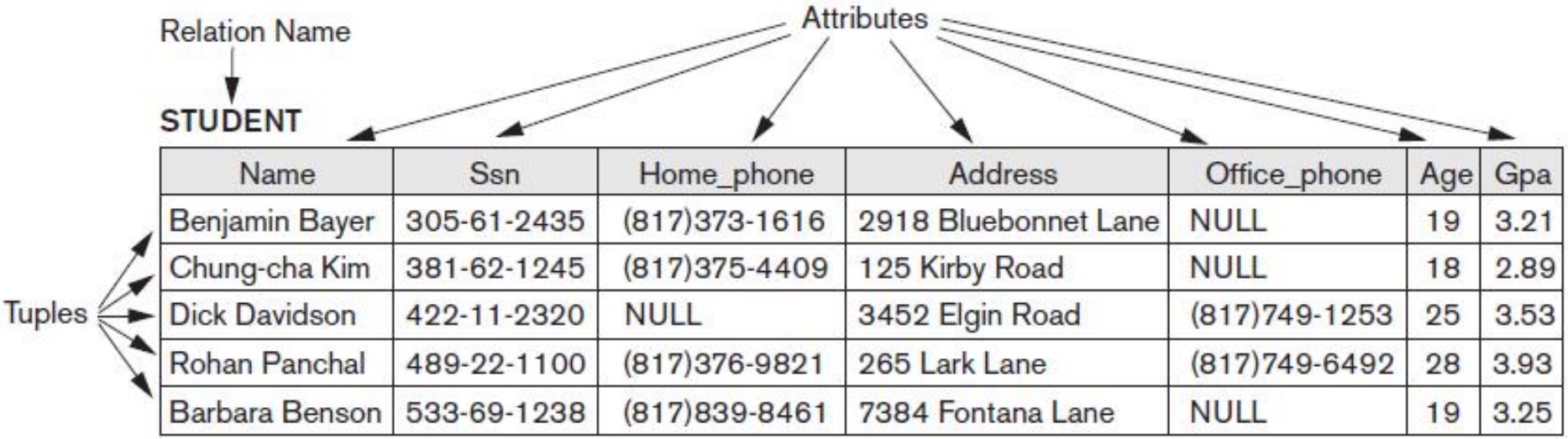
Definitions

- **RELATION: A table of values**
- A **relation** may be thought of as a set of rows OR set of columns.
- Each row represents a fact that corresponds to a real-world entity or relationship. Also called as Tuple
- Each row has a value of an item or set of items that uniquely identifies that row in the table
- Each column typically is called by its column name represents attribute
- Relation schema R is defined over **attributes** R(A1, A2, ...,An)
- **Relational Database Schema:** A set S of relation schemas that belong to the same database. S = {R1, R2, ..., Rn}
- The **degree** of a relation is the number of attributes 'n'

Definitions (contd..)

- **Relation** (or **relation state**) $r(R)$
  - Set of ***n*-tuples** $r = \{t_1, t_2, ..., t_m\}$
  - Each n-tuple t
    - Ordered list of n values $t = <v_1, v_2, ..., v_n>$
    - Each value v_i, $1 \leq i \leq n$, is an element of $\text{dom}(A_i)$ or is a special NULL value
  - **Mathematical relation** of degree n on the domains $\text{dom}(A_1), \text{dom}(A_2), ..., \text{dom}(A_n)$
  - **Subset** of the **Cartesian product** of the domains that define R:
    - $r(R) \subseteq (\text{dom}(A_1) \times \text{dom}(A_2) \times ... \times \text{dom}(A_n))$

The Attributes and Tuples of a relation STUDENT



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

Properties of Relations (Tables)

- There are no duplicate rows (tuples)
- Tuples are unordered, top to bottom
- Attributes are unordered, left to right
- All attribute values are atomic (or scalar)
  - Composite and multi-valued attributes are not allowed.
- Relational databases do not allow ***repeating groups***

Relational Model Constraints

- Constraints are **conditions** that must hold on **all** valid relation instances.
- There are three main types of constraints:
  - **Key** constraints
  - **Entity integrity** constraints
  - **Referential integrity** constraints

Key Constraints

- **Super key** of R:
  - A set of attributes **SK** of R such that no two tuples in any valid relation instance r(R) will have the same value for **SK**. That is, for any distinct tuples **t1** and **t2** in r(R), t1[**SK**] ≠ t2[**SK**].
- **Key** of R:
  - A "minimal" super key; that is, a super key K such that removal of any attribute from K results in a set of attributes that is not a super key.
- Example: The CAR relation schema: CAR(State, Reg#, SerialNo, Make, Model, Year) has two keys Key1 = {State, Reg#}, Key2 = {SerialNo}, which are also super keys. {SerialNo, Make} is a super key but *not* a key.
- Note: If a relation has several **candidate keys (a relation schema may have more than one key)** , one is chosen arbitrarily to be the **primary key**. The primary key attributes are underlined.

Entity Integrity

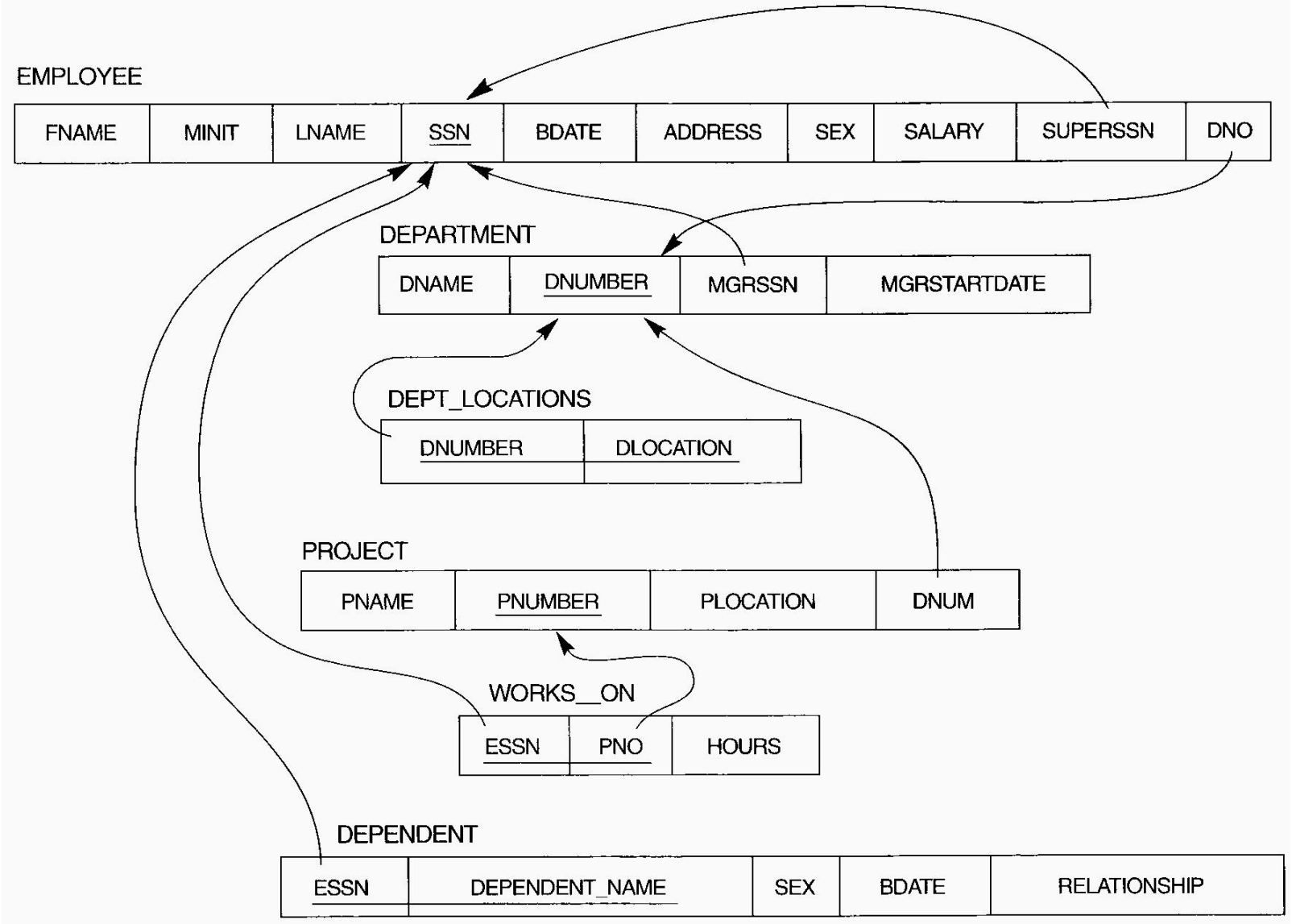
- The *primary key attributes* PK of each relation schema R cannot have null values in any tuple of r(R).
- No primary key value can be NULL.
- t[PK] ≠ null for any tuple t in r(R)
- This is because primary key values are used to *identify* the individual tuples

Referential Integrity (Foreign Key)

- Specified between two relations
- Maintains consistency among tuples in two relations

- Used to specify a relationship among tuples in two relations: The referencing relation and the referenced relation.
- Tuples in the referencing relation R1 have attributes FK (called foreign key attributes) that reference the primary key attributes PK of the referenced relation R2.
- **i.e., A tuple t1 in R1 is said to reference a tuple t2 in R2 if t1[FK] = t2[PK].** The attributes in FK have the same domain(s) as the primary key attributes PK
- A referential integrity constraint is displayed in a relational database schema as a directed arc from R1.FK to R2.PK

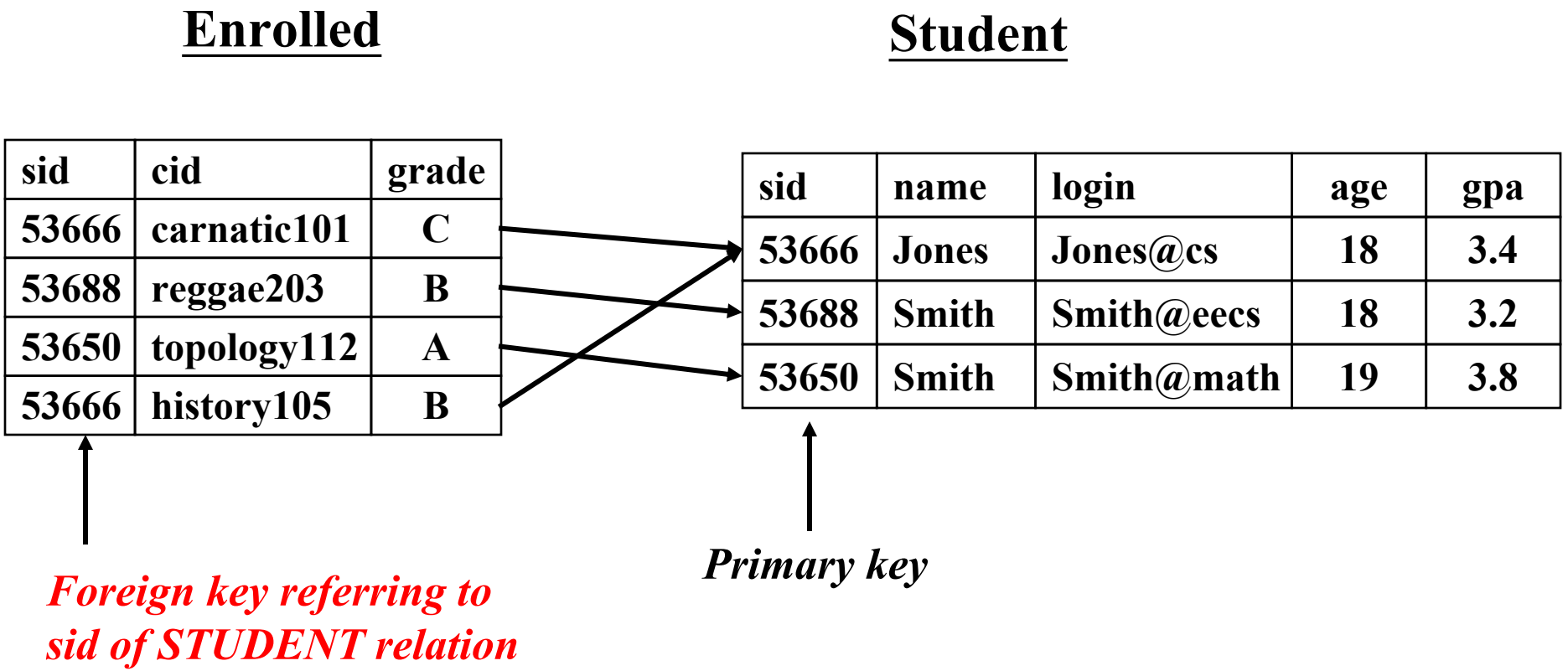
Referential Integrity Constraints on COMPANY Schema



Keys - Summary

- Key
- Super Key
- Candidate Keys
  - Primary Key
  - Alternate Key
- Secondary Keys

Keys and Referential Integrity



Update Operations on Relations

- INSERT a tuple.
  - Ex: Insert < Ceilia, F, Kolonsky,'677678984',…. > into EMPLOYEE
- DELETE a tuple.
  - Ex : Delete the EMPLOYEE tuple with SSN='9987'.
- MODIFY a tuple.
  - Ex : modify SALARY of the EMPLOYEE tuple SSN='998877' to 28000.
- Integrity constraints should not be violated by the update operations.
- Several update operations may have to be grouped together.
- Updates may **propagate** to cause other updates automatically. This may be necessary to maintain integrity constraints.

Exercise

- Consider the following relations for a database that keeps track of student enrollment in courses and the books adopted for each course:

  STUDENT(SSN, Name, Major, Bdate)

  COURSE(Course#, Cname, Dept)

  ENROLL(SSN, Course#, Quarter, Grade)

  BOOK_ADOPTION(Course#, Quarter, Book_ISBN)

  TEXT(Book_ISBN, Book_Title, Publisher, Author)

- **Draw a relational schema diagram specifying the foreign keys for this schema.**

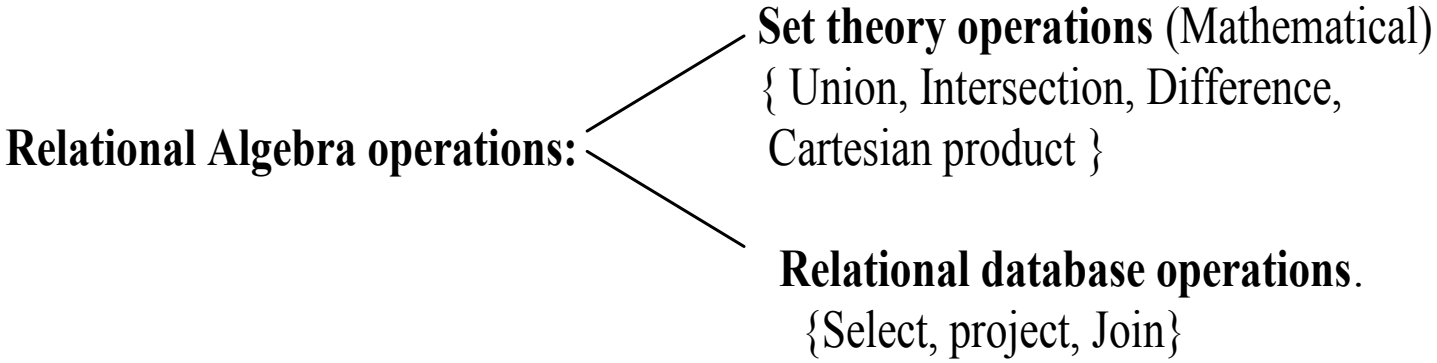
Answer

- The Attribute SSN of relation ENROLL that references relation STUDENT
- The Attribute Course# of relation ENROLL that references relation COURSE
- The Attribute Course# of relation BOOK_ADOPTION that references relation COURSE
- The Attribute Book_ISBN of relation BOOK_ADOPTION that references relation TEXT

Relational Algebra

Relational Algebra

- Relational Algebra is the query language for Relational Data Model enable the user to specify basic retrieval requests (queries).



All operations may operate on one or more relations and the result is in the form of a **relation**.

SELECT Operation (σ)

- Used to select a *subset* of the tuples from a relation that satisfy a ***selection condition***
- Gives the horizontal subset (selected tuples) of a Relation
- Syntax: $\sigma_{<\text{selection condition}>}(R)$ where σ *(sigma)* is used to denote the select operator, and the *selection condition* is a Boolean expression specified on the attributes of relation R
- Example1: To select the EMPLOYEE tuples whose department number is four
  - $\sigma_{DNO = 4}$ (EMPLOYEE)
- Example 2: To select the EMPLOYEE tuples whose salary is greater than $30,000
  - $\sigma_{SALARY > 30,000}$ (EMPLOYEE)

SELECT Operation Properties

- The SELECT operation $\sigma_{<\text{selection condition}>}(R)$ produces a relation S that has the same schema as R
- The SELECT operation s is commutative

$$\sigma_{<\text{condition1}>}(\sigma_{<\text{condition2}>}(R)) = \sigma_{<\text{condition2}>}(\sigma_{<\text{condition1}>}(R))$$

- A cascaded SELECT operation may be applied in any order;

$$\sigma_{<\text{cond1}>}(\sigma_{<\text{cond2}>}(\sigma_{<\text{cond3}>}(R)) = \sigma_{<\text{cond2}>}(\sigma_{<\text{cond3}>}(\sigma_{<\text{cond1}>}(R)))$$

- A cascaded SELECT operation may be replaced by a single selection with a conjunction of all the conditions; i.e.,

$$\sigma_{<\text{cond1}>}(\sigma_{<\text{cond2}>}(\sigma_{<\text{cond3}>}(R)) = \sigma_{<\text{cond1}> \text{ AND } <\text{cond2}> \text{ AND } <\text{cond3}>}(R)))$$

PROJECT Operation (π)

- Projects certain *columns* from the relation/table and discards the other columns.
- Gives the vertical subset (selected columns) of a relation
- Creates a vertical partitioning – one with the needed columns (attributes) containing results of the operation and other containing the discarded Columns.
- Syntax: $\pi_{<\text{attribute list}>}(R)$, where π (pi) is the symbol used to represent the project operation and <attribute list> is the desired list of attributes from the attributes of relation R.
- *removes any duplicate tuples*, so the result of the project operation is a set of tuples and hence a valid relation.
- Example: To list each employee's first and last name and salary, the following is used:
  - $\pi_{\text{LNAME, FNAME, SALARY}}(\text{EMPLOYEE})$

PROJECT Operation Properties

- If the list of attributes includes a key of R, then the number of tuples in the result of projection $\pi_{<list>}(R)$ is always less or equal to the number of tuples in R.

- $\pi_{<list1>}(\pi_{<list2>}(R)) = \pi_{<list1>}(R)$ as long as <list2> contains the attributes in <list1>

Sequence of SELECT and PROJECT operations

- Several operations can be combined to form a *relational algebra expression* (query)
- Example: Retrieve the names and salaries of employees who work in department 4:
  - $\pi_{\text{FNAME, LNAME, SALARY}}(\sigma_{\text{DNO=4}}(\text{EMPLOYEE}))$
- Alternatively, we specify explicit intermediate relations for each step:

$$\text{DEPT4\_EMPS} \leftarrow \sigma_{\text{DNO=4}}(\text{EMPLOYEE})$$

$$R \leftarrow \pi_{\text{FNAME, LNAME, SALARY}}(\text{DEPT4\_EMPS})$$

Set theory Operations

- Binary operations from mathematical set theory:

  UNION: $R_1 \cup R_2$,
  INTERSECTION: $R_1 \cap R_2$,
  SET DIFFERENCE: $R_1 - R_2$,
  CARTESIAN PRODUCT: $R_1 \times R_2$.

- For $\cup$, $\cap$, - , the operand relations $R_1(A_1,A_2,\ldots,A_n)$ and $R_2(B_1,B_2,\ldots,B_n)$ must have the *same number of attributes*, and the *domains* of corresponding attributes must be *compatible*;

  that is, $dom(A_i)=dom(B_i)$ for i=1,2,…, n. This condition is called ***union compatibility***.

- The resulting relation for $\cup$, $\cap$, or - has the same attribute names as the *first* operand R_1 (by convention).

JOIN Operation ⋈

- The sequence of Cartesian product followed by ***select*** is used quite commonly to identify and select related tuples from two relations, a special operation, called JOIN. It is denoted by ⋈
  - This operation is very important for any relational database with more than a single relation, because it allows us to process relationships among relations.
  - The general form of a join operation on two relations $R(A_1, A_2, \ldots, A_n)$ and $S(B_1, B_2, \ldots, B_m)$ is: $R \bowtie_{\text{<join condition>}} S$ where R and S can be any relations that result from general *relational algebra expressions*.
- Types
  - Equijoin
  - Natural Join
  - Theta Join

Example Instances

R1

sid	bid	day
22	101	10/10/99
58	103	11/12/99

S1

sid	sname	rating	age
22	Deepa	7	45.0
31	Laxmi	8	55.5
58	Roopa	10	35.0

S2

sid	sname	rating	age
28	Yamuna	9	35.0
31	Laxmi	8	55.5
44	Geeta	5	35.0
58	Roopa	10	35.0

Relation Algebra Operations - Recap

- *Selection* (σ)
- *Projection* (π)
- *Cross- product* (X)
- *Set- difference* (–)
- *Union* (∪)
- *Intersection* (∩)

Projection

sname	rating
Yamuna	9
Laxmi	8
Geeta	5
Roopa	10

$\pi_{sname,\ rating}(S2)$

age
35.0
55.5

$\pi_{age}(S2)$

Selection

sid	sname	rating	age
28	Yamuna	9	35.0
58	Roopa	10	35.0

$\sigma_{rating > 8}(S2)$

sname	rating
Yamuna	9
Roopa	10

$\pi_{sname,\ rating}(S2)\ (\sigma_{rating > 8}(S2))$

Union, Intersection, Set Difference

sid	sname	rating	age
22	Deepa	7	45.0
31	Laxmi	8	55.5
58	Roopa	10	35.0
44	Geeta	5	35.0
28	Yamuna	9	35.0

$S1 \cup S2$

sid	sname	rating	age
31	Laxmi	8	55.5
58	Roopa	10	35.0

$S1 \cap S2$

sid	sname	rating	age
22	Deepa	7	45.0

$S1 - S2$

Cross-Product

(sid)	sname	rating	age	(sid)	bid	day
22	Deepa	7	45.0	22	101	10/10/99
22	Deepa	7	45.0	58	103	11/12/99
31	Laxmi	8	55.5	22	101	10/10/99
31	Laxmi	8	55.5	58	103	11/12/99
58	Roopa	10	35.0	22	101	10/10/99
58	Roopa	10	35.0	58	103	11/12/99

Joins

Condition Join :

(sid)	sname	rating	age	(sid)	bid	day
22	Deepa	7	45.0	58	103	11/12/99
31	Laxmi	8	55.5	58	103	11/12/99

Equi-Join

(sid)	sname	rating	age	bid	day
22	Deepa	7	45.0	101	10/10/99
58	Roopa	10	35.0	103	11/12/99

Division

- Not supported as a primitive operator, but useful for expressing queries like:
  - *Find sailors who have reserved **all** boats .*

A

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

pno
p2

B1

sno
s1
s2
s3
s4

A/B1

pno
p2
p4

B2

sno
s1
s4

A/B2

pno
p1
p2
p4

B3

sno
s1

A/B3

Additional Operations

- Additional Operations
  - Set intersection
  - Natural join
  - Aggregation
  - Outer Join
  - Division
- All above, other than aggregation, can be expressed using basic operations we have seen earlier

Set-Intersection Operation – Example

- Relation *r*, *s*

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cap s$

A	B
α	2

Natural Join Operation – Example

- Relations r, s:

A	*B*	*C*	*D*
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	*D*	*E*
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ϵ

s

n r $\bowtie$ s

A	*B*	*C*	*D*	*E*
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Natural-Join Operation

n Notation: r ⋈ s

- Let *r* and *s* be relations on schemas *R* and *S* respectively. Then, r ⋈ s is a relation on schema $R \cup S$ obtained as follows:
  - Consider each pair of tuples t_r from *r* and t_s from *s*.
  - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple *t* to the result, where
    - *t* has the same value as t_r on *r*
    - *t* has the same value as t_s on *s*
- Example:

  $R = (A, B, C, D)$

  $S = (E, B, D)$

  - Result schema = (A, B, C, D, E)
  - *r* ⋈ *s* is defined as:

Aggregate Functions and Operations

- **Aggregation function** takes a collection of values and returns a single value as a result.

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

- **Aggregate operation** in relational algebra

$$_{G_1,G_2,\ldots,G_n}\mathcal{G}_{F_1(A_1),F_2(A_2,\ldots,F_n(A_n)}(E)$$

  E is any relational-algebra expression

  - $G_1, G_2 \ldots, G_n$ is a list of attributes on which to group (can be empty)
  - Each F_i is an aggregate function
  - Each A_i is an attribute name

Aggregate Operation – Example

- Relation *r*:

A	*B*	*C*
α	*α*	7
α	*β*	7
β	*β*	3
β	*β*	10

$g_{\text{sum(c)}}(r)$

sum(*c*)
27

Question: Which aggregate operations cannot be expressed using basic relational operations?

Aggregate Operation – Example

- Relation *account* grouped by *branch-name*:

branch_name	*account_number*	*balance*
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

$$branch\_name\ \mathcal{G}\ \mathbf{sum}(balance)\ (account)$$

branch_name	**sum**(*balance*)
Perryridge	1300
Brighton	1500
Redwood	700

Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values:
  - *null* signifies that the value is unknown or does not exist
  - All comparisons involving *null* are (roughly speaking) **false** by definition.
    - We shall study precise meaning of comparisons with nulls later

Outer Join – Example

- Relation *loan*

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

n Relation *borrower*

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

Outer Join – Example

- Join

loan ⋈ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

Left Outer Join

loan ⟕ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*

Outer Join – Example

Right Outer Join

$loan \bowtie\!\!\!\!\sqsubset borrower$

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

Full Outer Join

$loan \sqsupset\!\bowtie\!\sqsubset borrower$

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

Question: can outerjoins be expressed using basic relational algebra operations

Banking Example

branch (branch_name, branch_city, assets)

customer (customer_name, customer_street, customer_city)

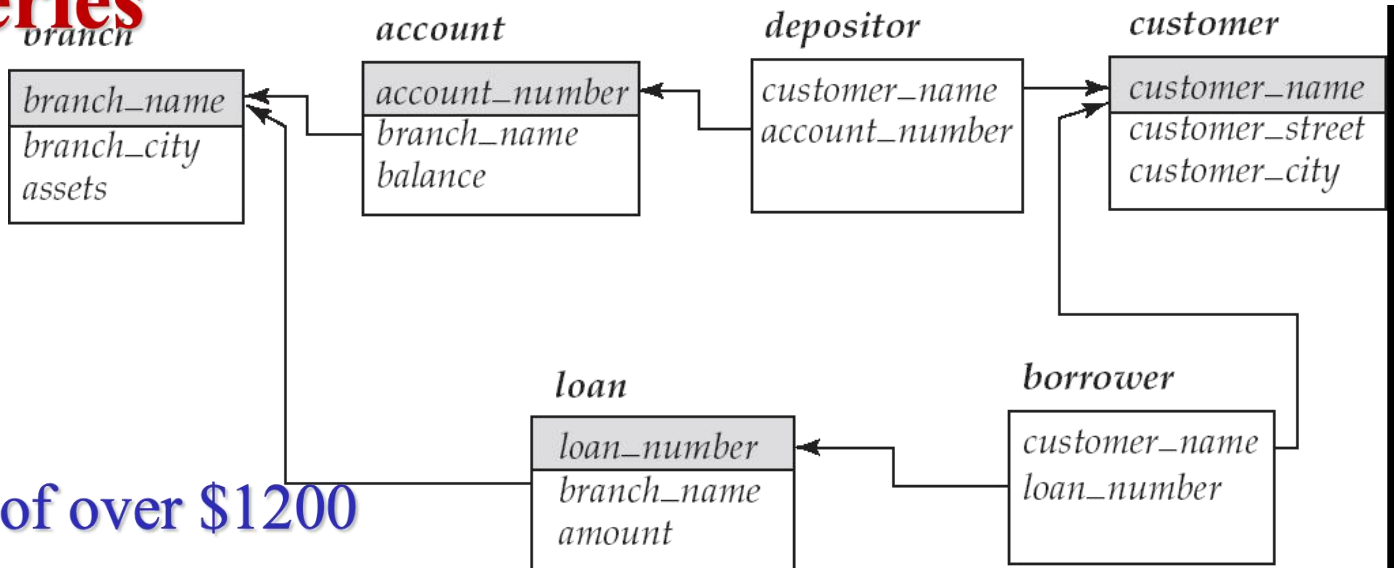
account (account_number, branch_name, balance)

loan (loan_number, branch_name, amount)

depositor (customer_name, account_number)

borrower (customer_name, loan_number)

Example Queries



- Find all loans of over $1200

$$\sigma_{amount > 1200}(loan)$$

Find the loan number for each loan of an amount greater than $1200

$$\prod_{loan\_number}(\sigma_{amount > 1200}(loan))$$

Find the names of all customers who have a loan, an account, or both, from the bank

$$\prod_{customer\_name}(borrower) \cup \prod_{customer\_name}(depositor)$$

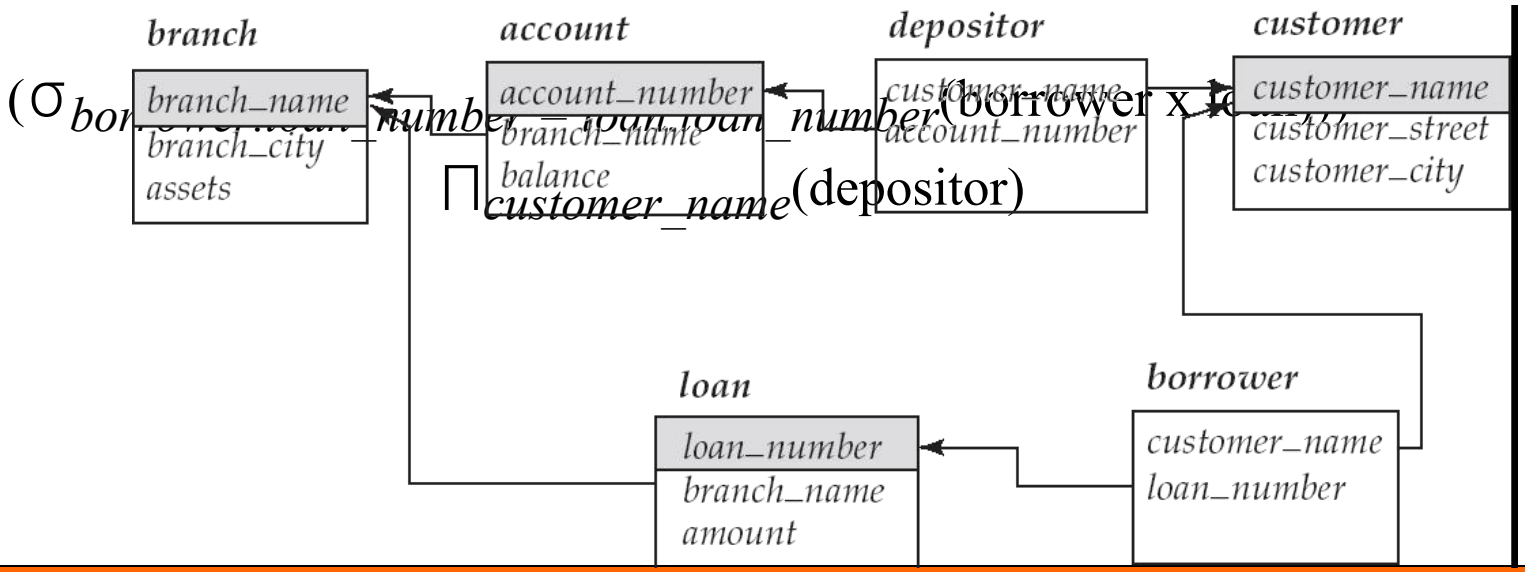
Example Queries

- Find the names of all customers who have a loan at the Perryridge branch.

$$\Pi_{customer\_name}(\sigma_{branch\_name=\text{“Perryridge”}}(\sigma_{borrower.loan\_number = loan.loan\_number}(borrower \times loan)))$$

n Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank.

$$\Pi_{customer\_name}(\sigma_{branch\_name = \text{“Perryridge”}}(\sigma_{borrower.loan\_number = loan.loan\_number}(borrower \times loan))) - \Pi_{customer\_name}(depositor)$$



Example Queries

- Find the names of all customers who have a loan at the Perryridge branch.

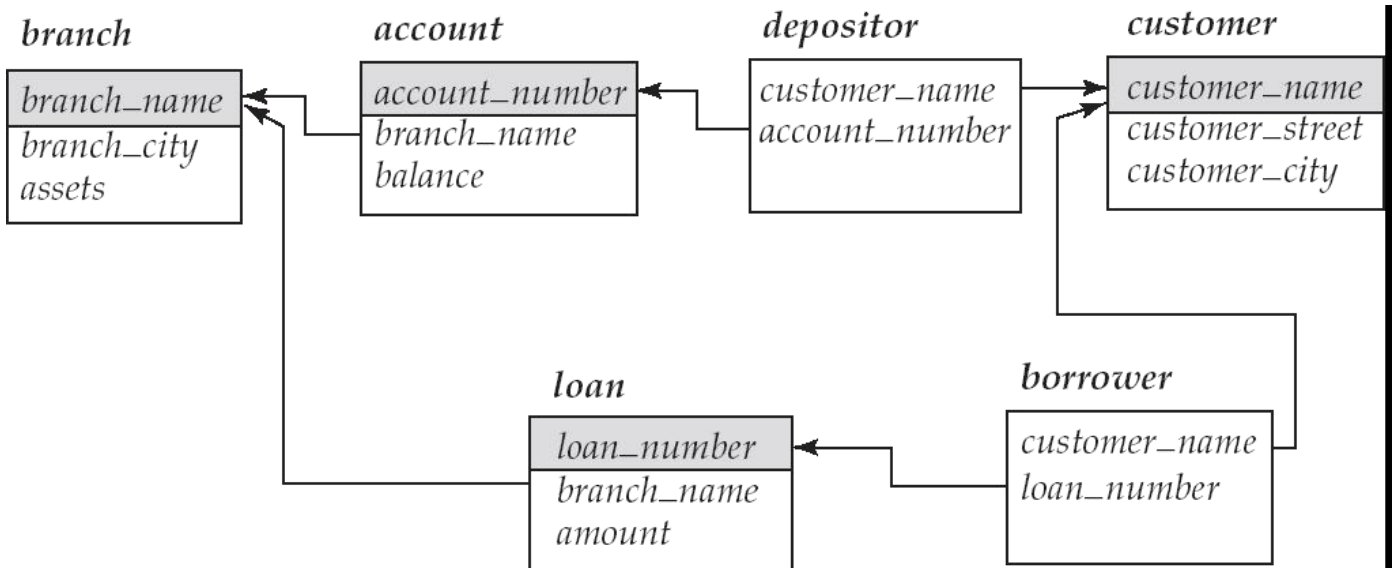
$\Pi_{customer\_name}(\sigma_{branch\_name = \text{“Perryridge”}}($

$\sigma_{borrower.loan\_number = loan.loan\_number}(borrower \times loan)))$

$\Pi_{customer\_name}(\sigma_{loan.loan\_number = borrower.loan\_number}$

$($

$(\sigma_{branch\_name = \text{“Perryridge”}}(loan)) \times borrower))$



Bibliography

1. Database systems Models, Languages, Design and Application Programming, RamezElmasri and Shamkant B. Navathe, 7th Edition, 2017, Pearson.
2. Database Management Systems, Ramakrishnan, and Gehrke, 3rd Edition, 2014, McGraw Hill
3. Database System Concepts, Silberschatz Korth and Sudharshan,, 6th Edition, McGrawHill, 2013.
4. Database Principles Fundamentals of Design, Implementation and Management, Coronel, Morris, and Rob, Cengage Learning, 2012.