

# Entity Relationship Modeling (E R Model)

- Database can be modeled as:
  - Collection of entities
  - Relationship among entities
- **Entity** is an object that exists and is distinguishable from other objects.
  - Example: specific person, company, event, plant
- Entities have attributes
  - Example: people have *names* and *addresses*
- An **entity set** is a set of entities of the same type that share the same properties.
  - Example: set of all persons, companies, trees, holidays

## Entity Sets *customer* and *loan*

customer\_id    customer\_name    customer\_street    customer\_city

|             |          |        |            |
|-------------|----------|--------|------------|
| 321-12-3123 | Jones    | Main   | Harrison   |
| 019-28-3746 | Smith    | North  | Rye        |
| 677-89-9011 | Hayes    | Main   | Harrison   |
| 555-55-5555 | Jackson  | Dupont | Woodside   |
| 244-66-8800 | Curry    | North  | Rye        |
| 963-96-3963 | Williams | Nassau | Princeton  |
| 335-57-7991 | Adams    | Spring | Pittsfield |

*customer*

loan\_number    amount

|      |      |
|------|------|
| L-17 | 1000 |
| L-23 | 2000 |
| L-15 | 1500 |
| L-14 | 1500 |
| L-19 | 500  |
| L-11 | 900  |
| L-16 | 1300 |

*loan*

# Relationship Sets

- A **relationship** is an association among several entities

Example:

|                 |                  |                |
|-----------------|------------------|----------------|
| <u>Hayes</u>    | <u>depositor</u> | <u>A-102</u>   |
| customer entity | relationship set | account entity |

- A **relationship set** is a mathematical relation among  $n \geq 2$  entities, each taken from entity sets

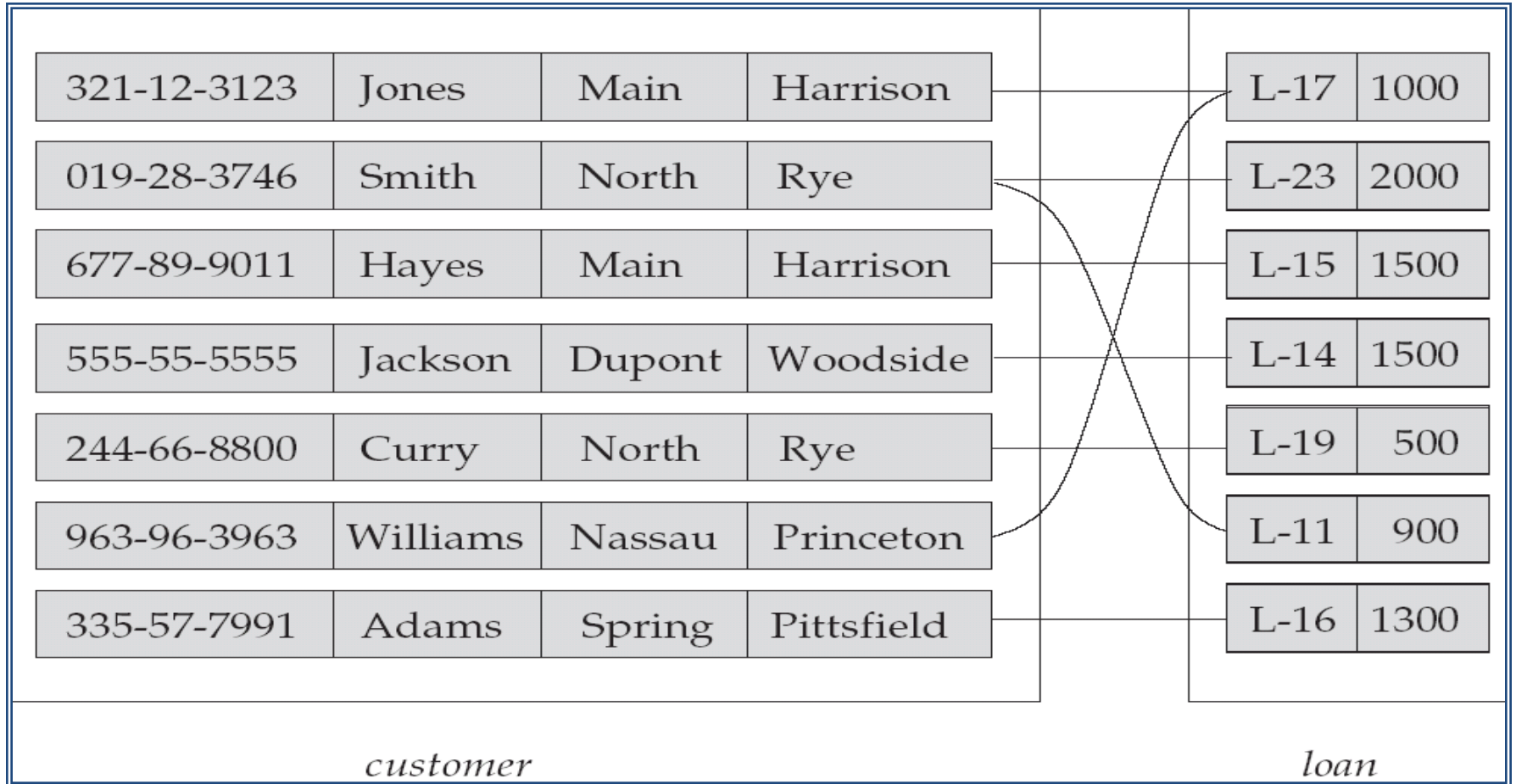
$$\{(e_1, e_2, \dots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \dots, e_n \in E_n\}$$

where  $(e_1, e_2, \dots, e_n)$  is a relationship

Example:

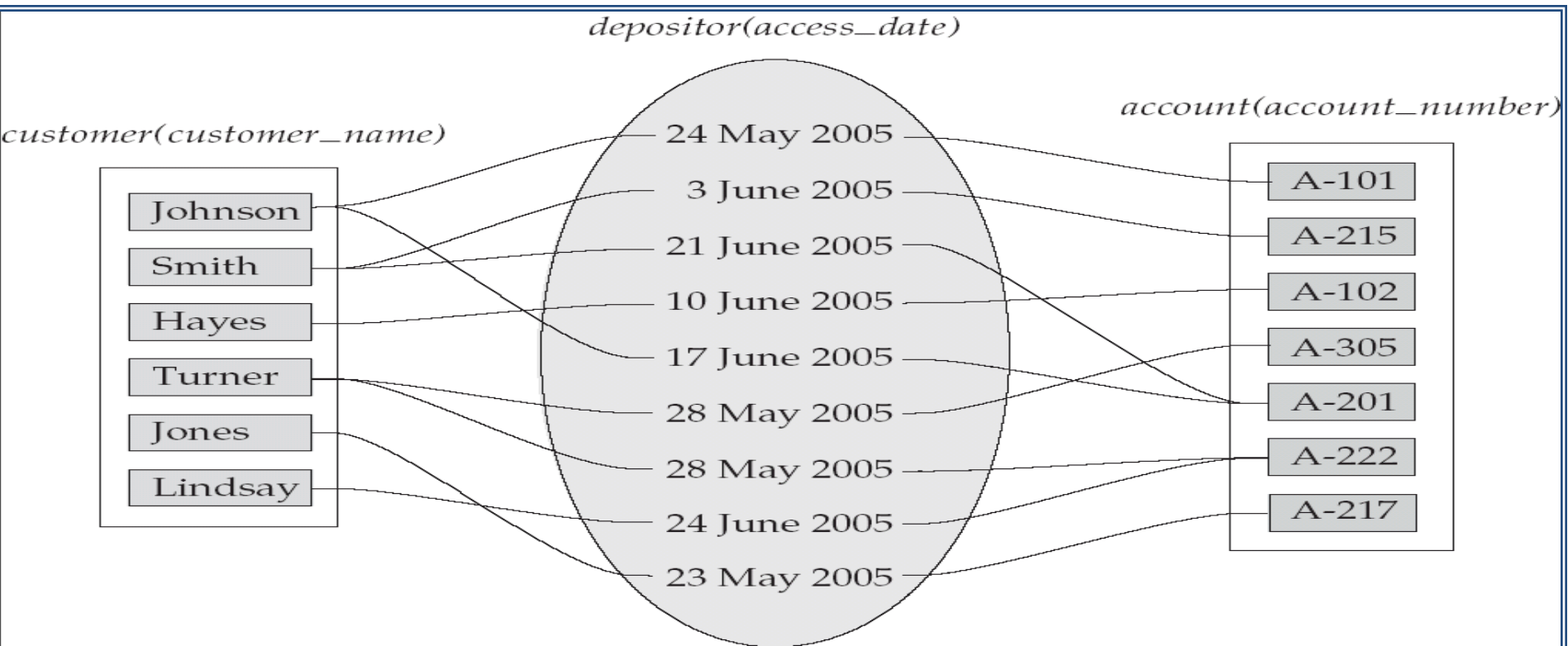
$(\text{Hayes}, \text{A-102}) \in \text{depositor}$

# Relationship Set *borrower*



## Relationship Sets... Contd.

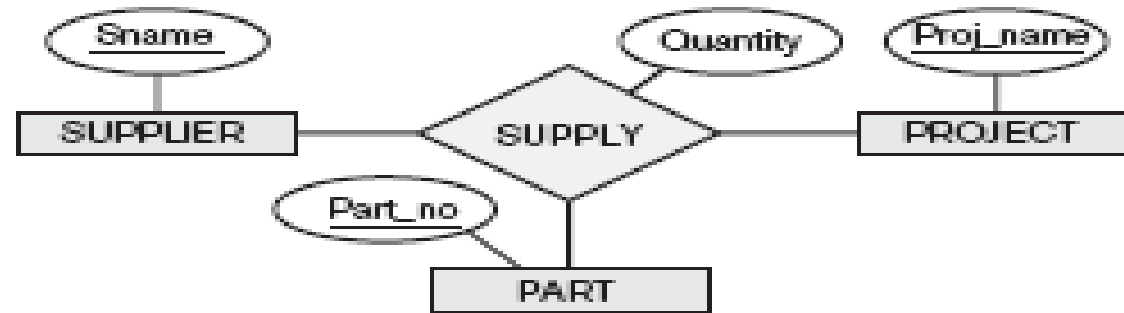
- An **attribute** can also be property of a relationship set.
- For instance, the depositor relationship set between entity sets customer and account may have the attribute access-date



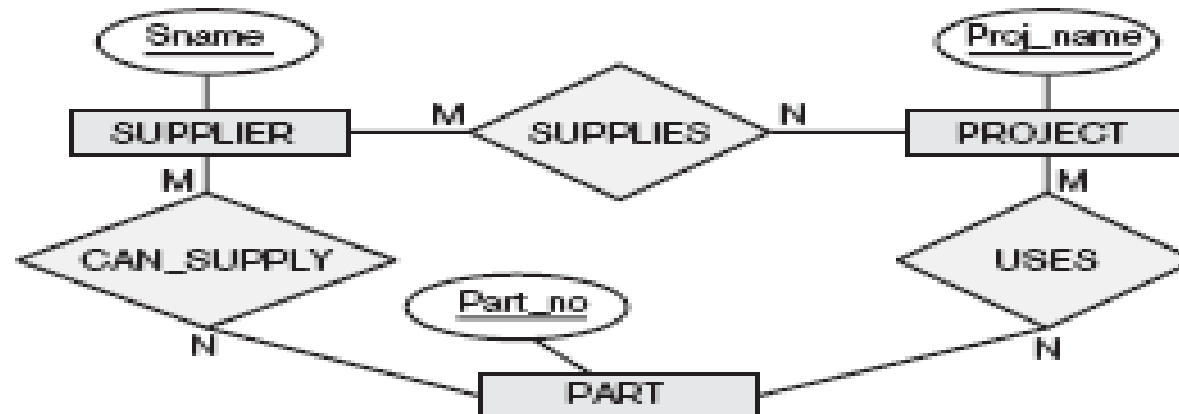
# Degree of a Relationship Set

- Refers to number of entity sets that participate in a relationship set.
- Relationship sets that involve two entity sets are **binary** (or degree two). Generally, most relationship sets in a database system are binary.
- Relationship sets may involve more than two entity sets.
- **Example:** Suppose employees of a bank may have jobs (responsibilities) at multiple branches, with different jobs at different branches. Then there is a ternary relationship set between entity sets *employee*, *job*, and *branch*
- Relationships between more than two entity sets are rare. Most relationships are binary.

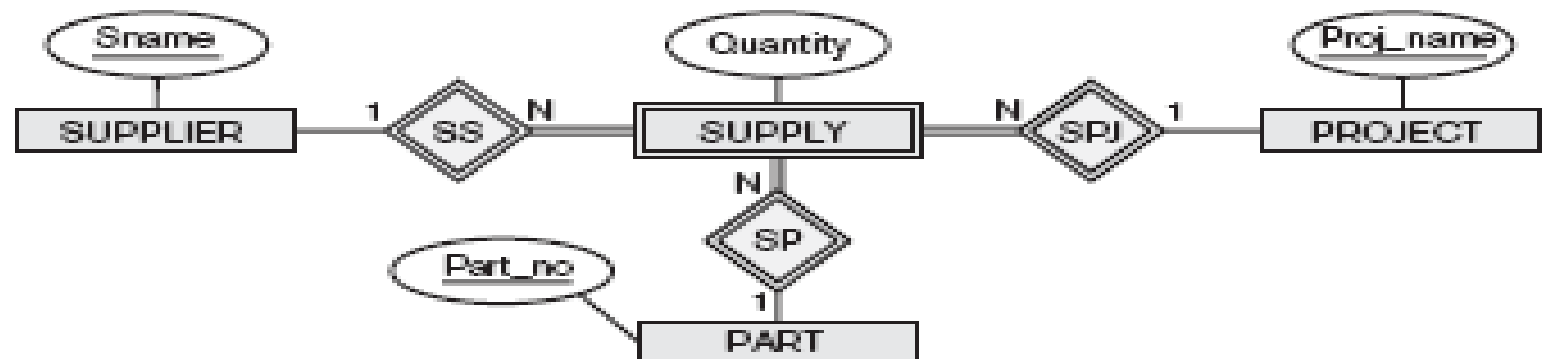
(a)



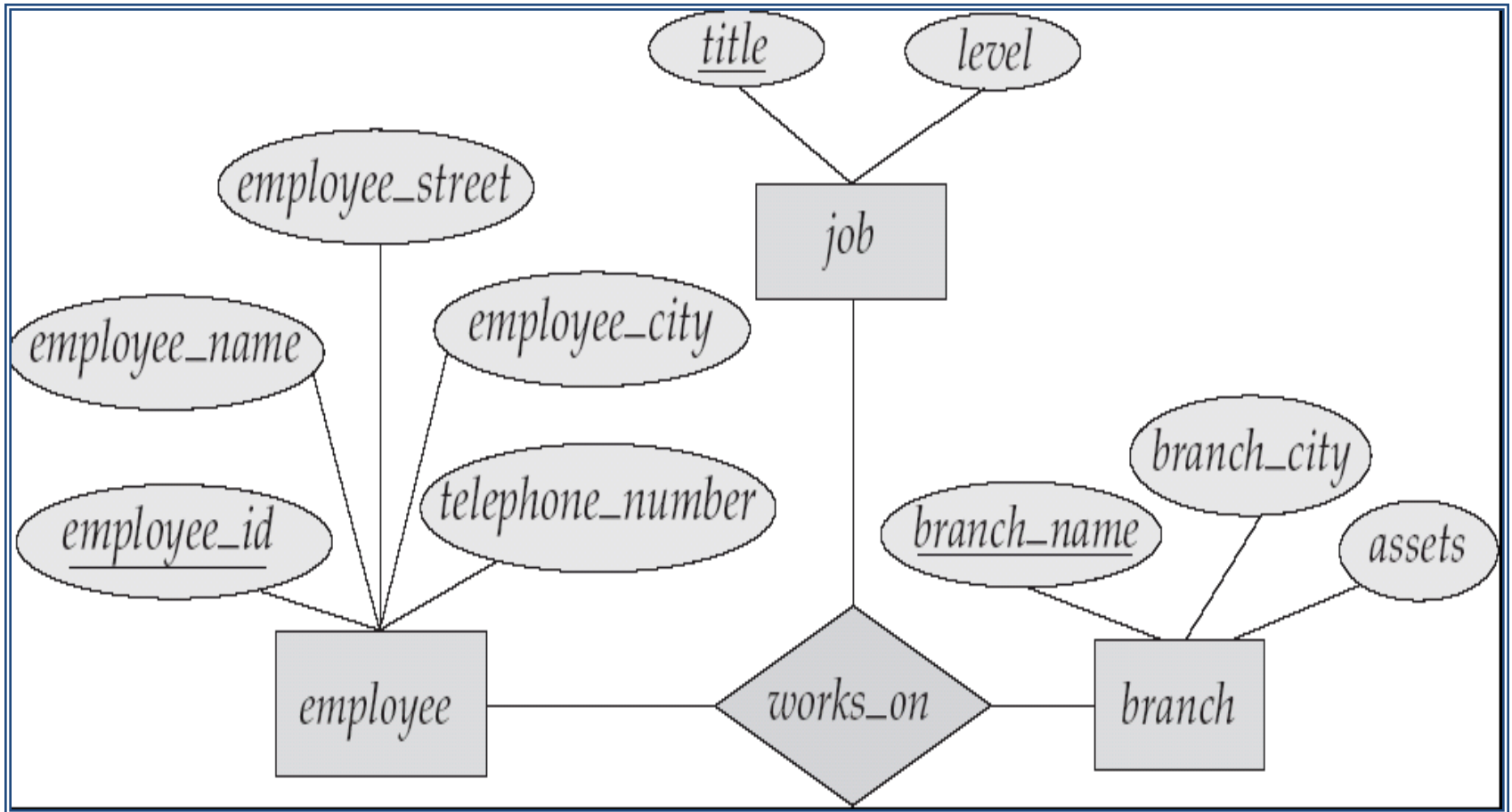
(b)



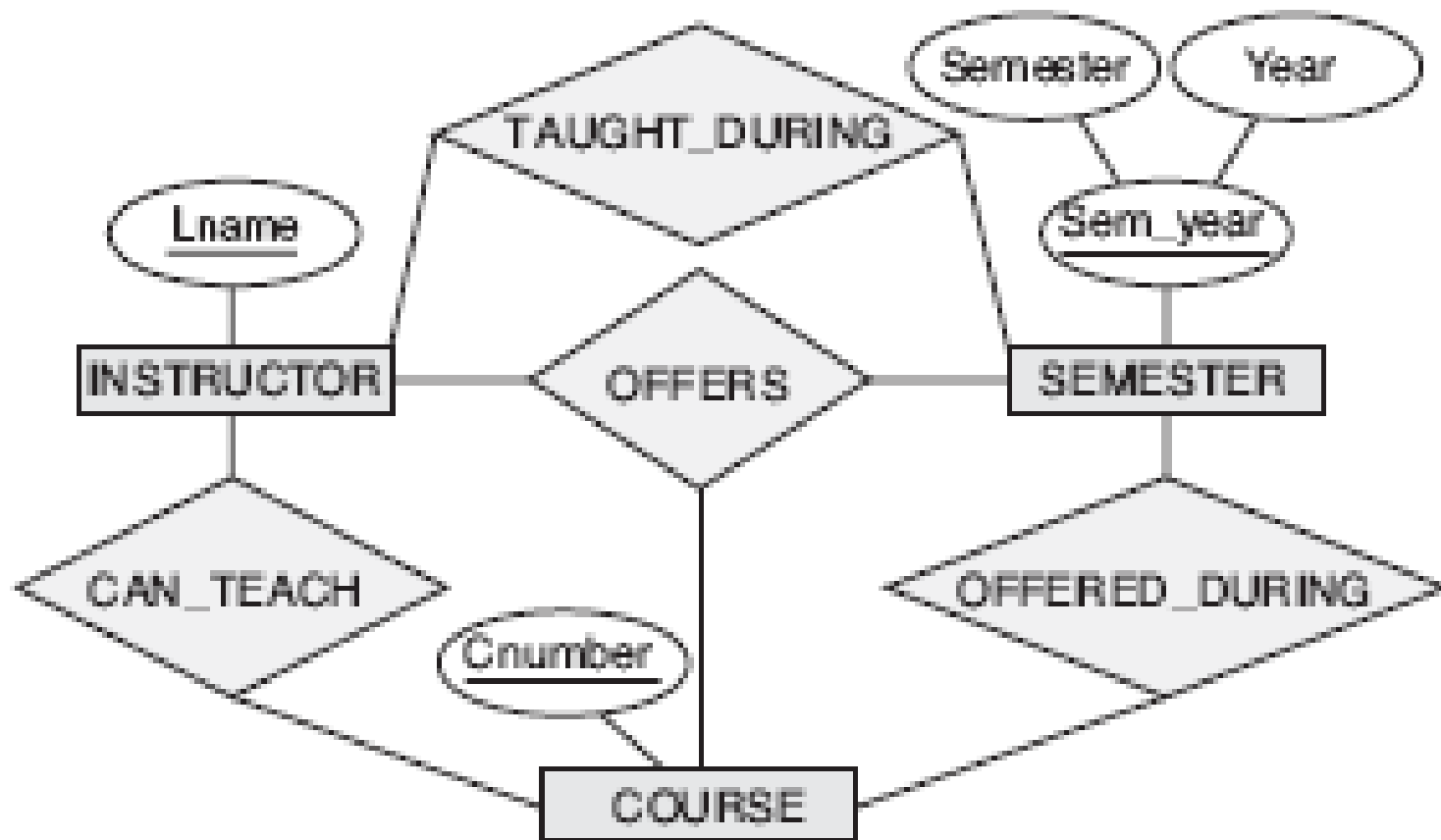
(c)



## E-R Diagram with a Ternary Relationship







# Attributes

- An entity is represented by a set of attributes, that is descriptive properties possessed by all members of an entity set.

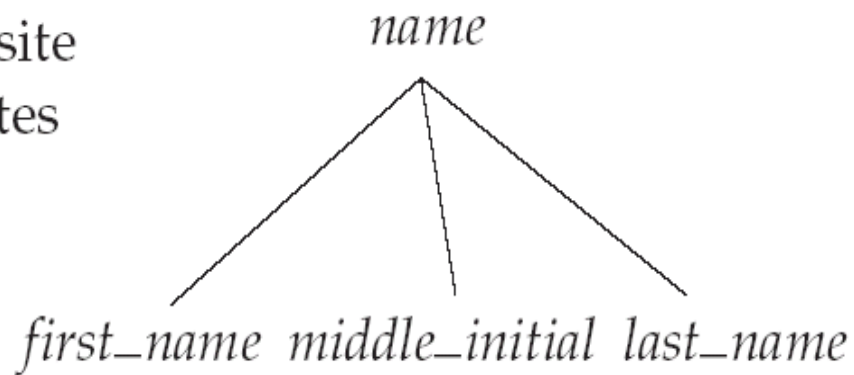
## Example:

*customer = (customer\_id, customer\_name, customer\_street, customer\_city )*  
*loan = (loan\_number, amount )*

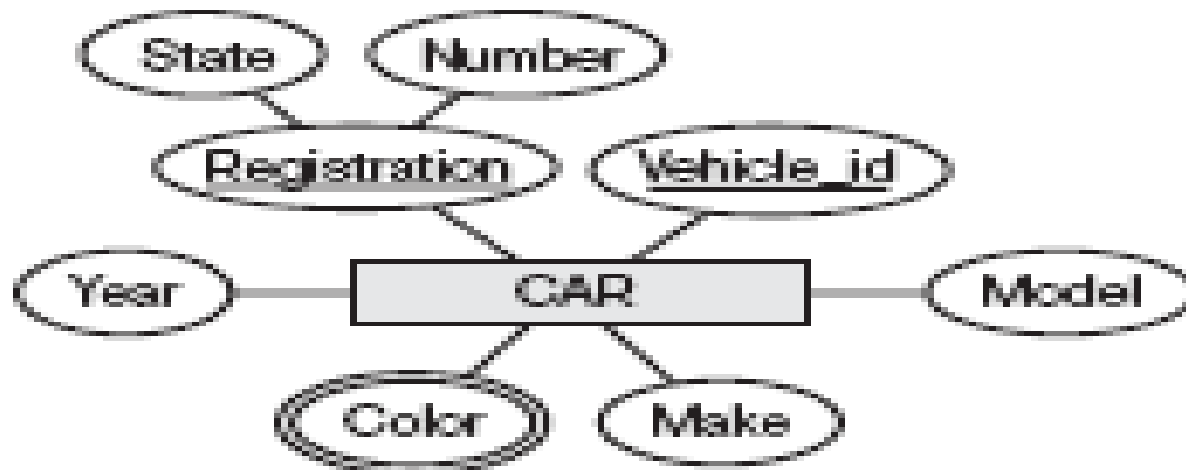
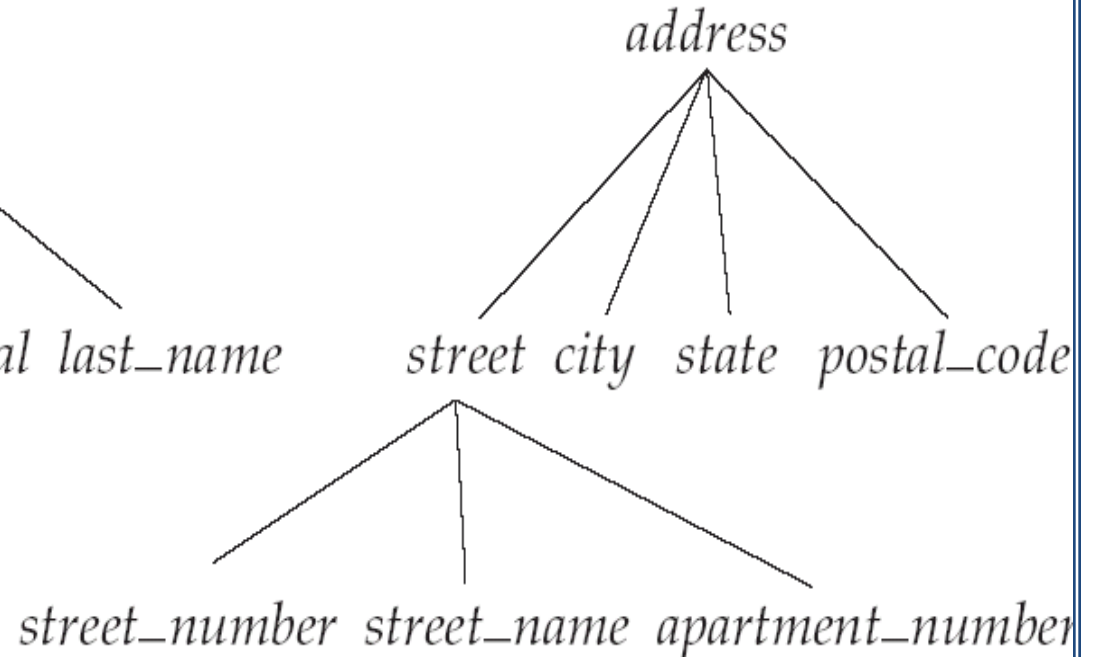
- **Domain** – the set of permitted values for each attribute
- **Attribute types:**
  - *Simple* and *composite* attributes.
  - *Single-valued* and *multi-valued* attributes
    - Example: multivalued attribute: *phone\_numbers*
  - *Derived* attributes
    - Can be computed from other attributes
    - Example: age, given date\_of\_birth

## Composite Attributes

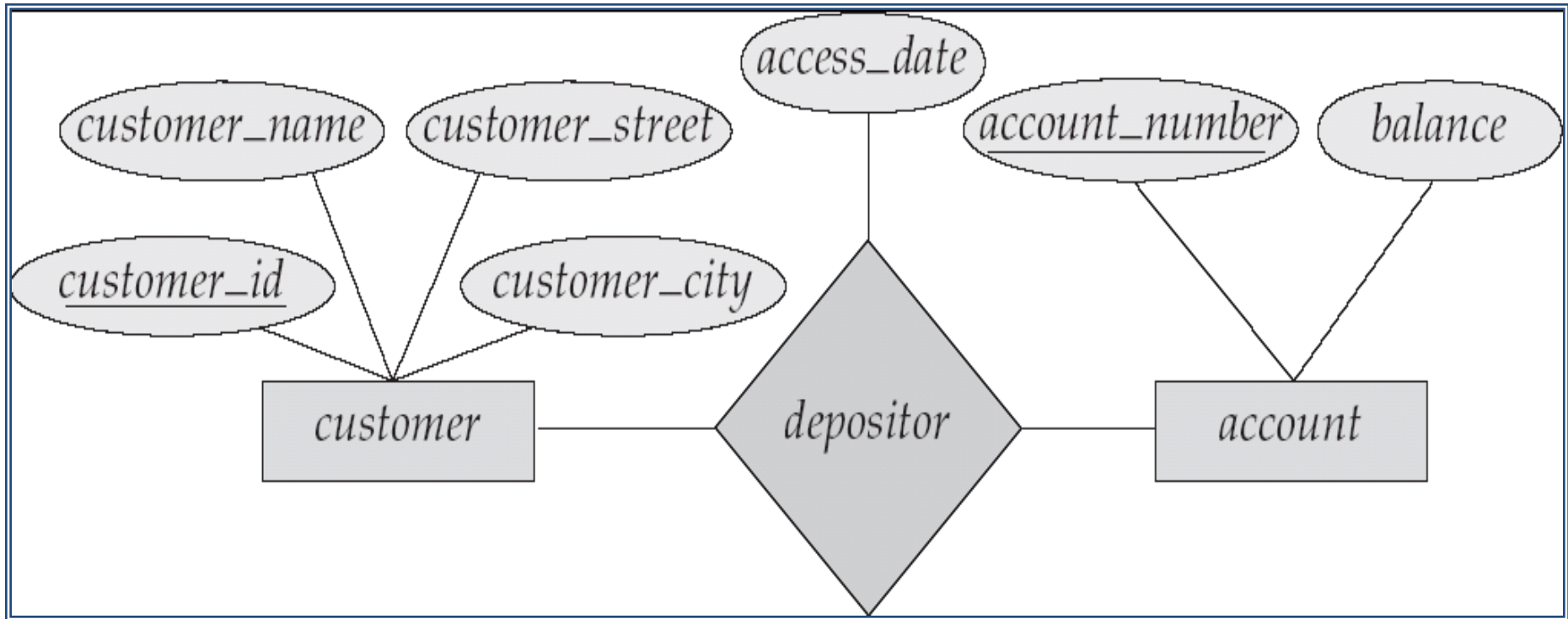
Composite  
Attributes



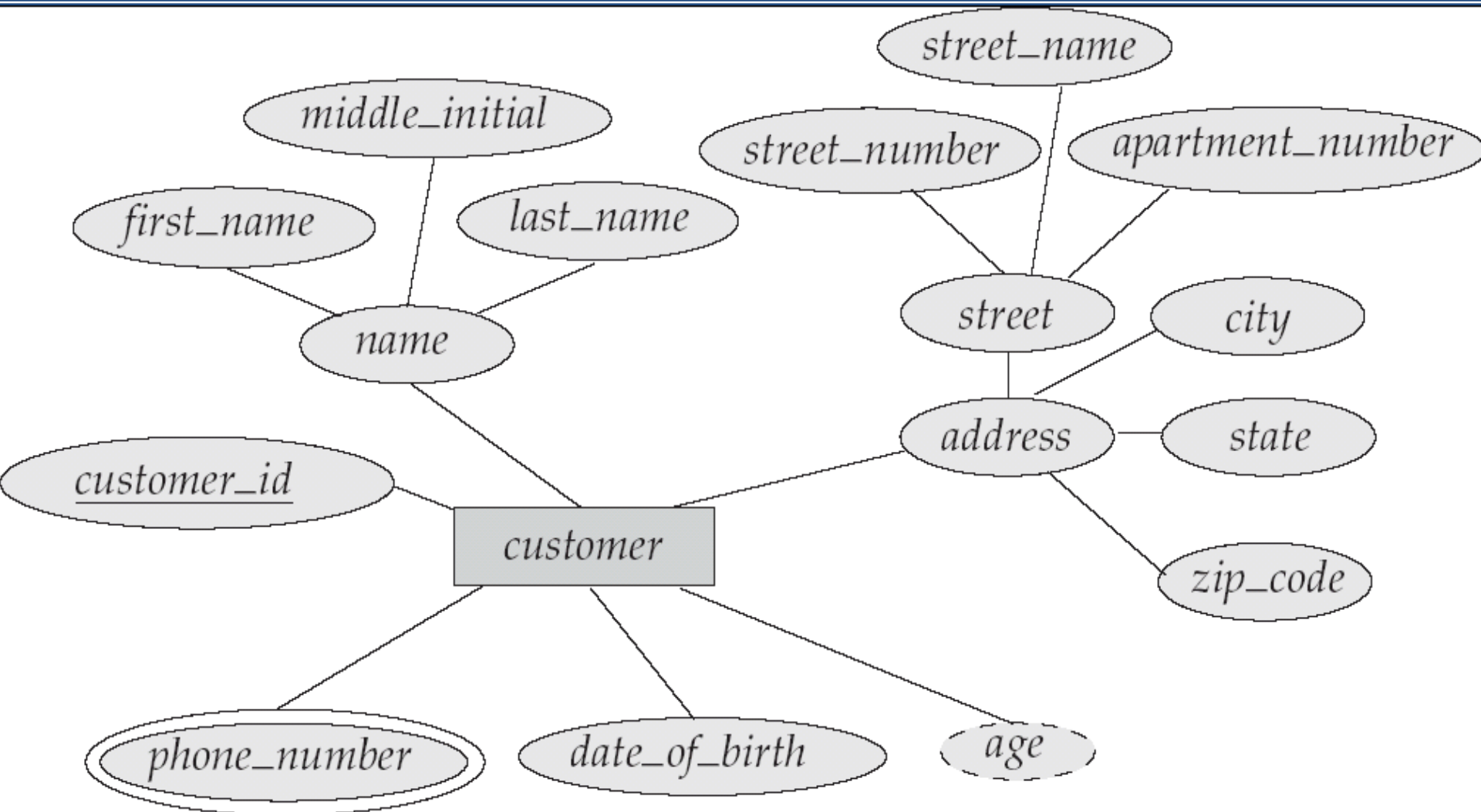
Component  
Attributes



## Relationship Sets with Attributes



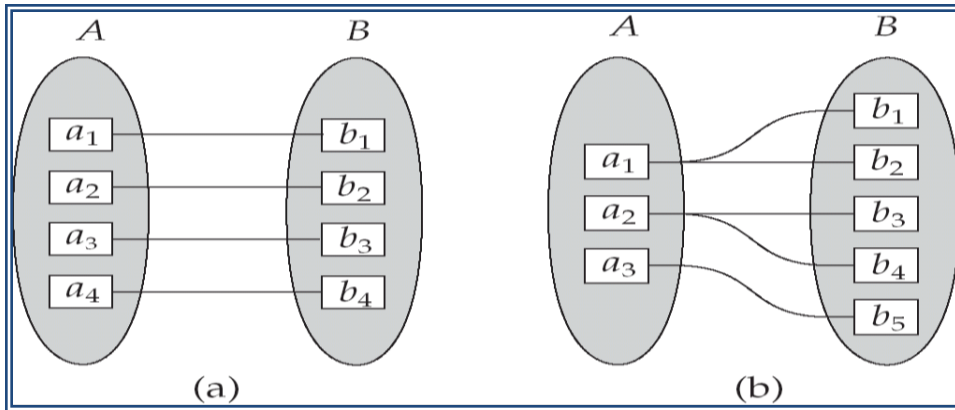
## E-R Diagram With Composite, Multivalued, and Derived Attributes



# Mapping Cardinality Constraints

- **Express the number of entities to which another entity can be associated via a relationship set.**
- **Most useful in describing binary relationship sets.**
- **For a binary relationship set the mapping cardinality must be one of the following types:**
  - **One to one**
  - **One to many**
  - **Many to one**
  - **Many to many**

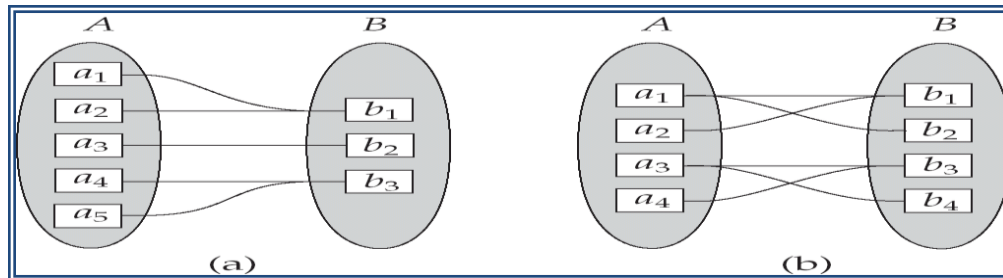
## Mapping Cardinalities...contd.



One to one

One to many

Note: Some elements in  $A$  and  $B$  may not be mapped to any elements in the other set

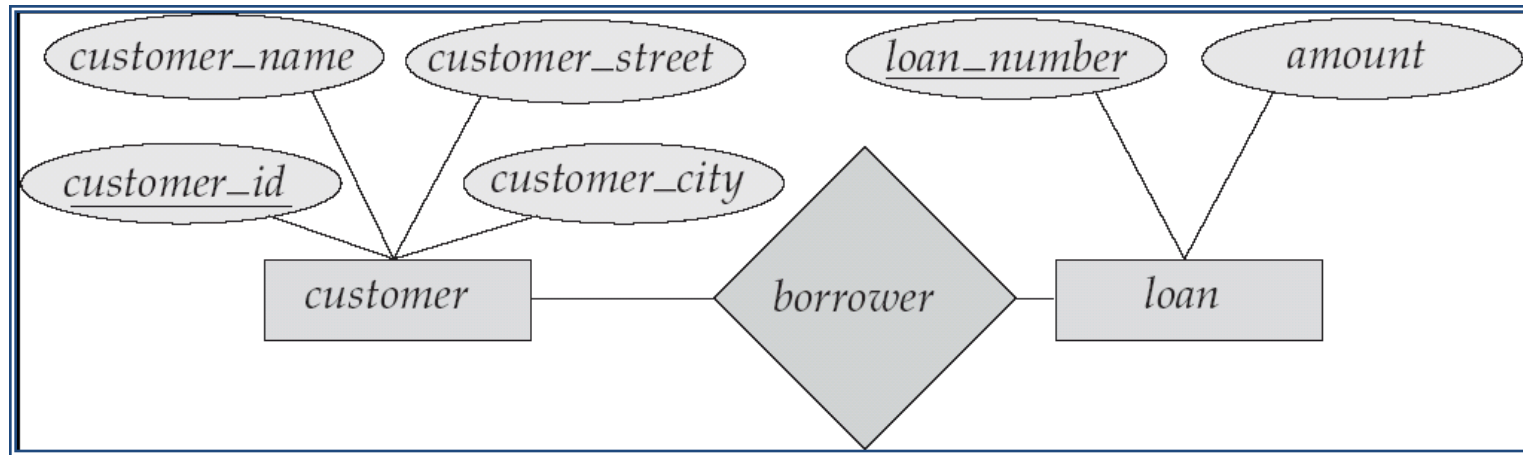


Many to one

Many to many

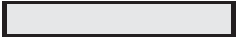
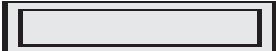
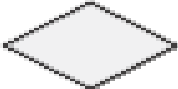
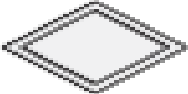







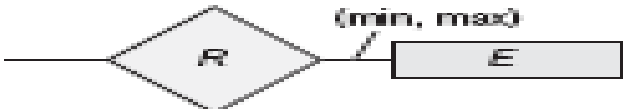
Note: Some elements in  $A$  and  $B$  may not be mapped to any elements in the other set

# E-R Diagrams



- Rectangles represent entity sets.
- Diamonds represent relationship sets.
- Lines link attributes to entity sets and entity sets to relationship sets.
- Ellipses represent attributes
  - Double ellipses represent multivalued attributes.
  - Dashed ellipses denote derived attributes.
- Underline indicates primary key attributes



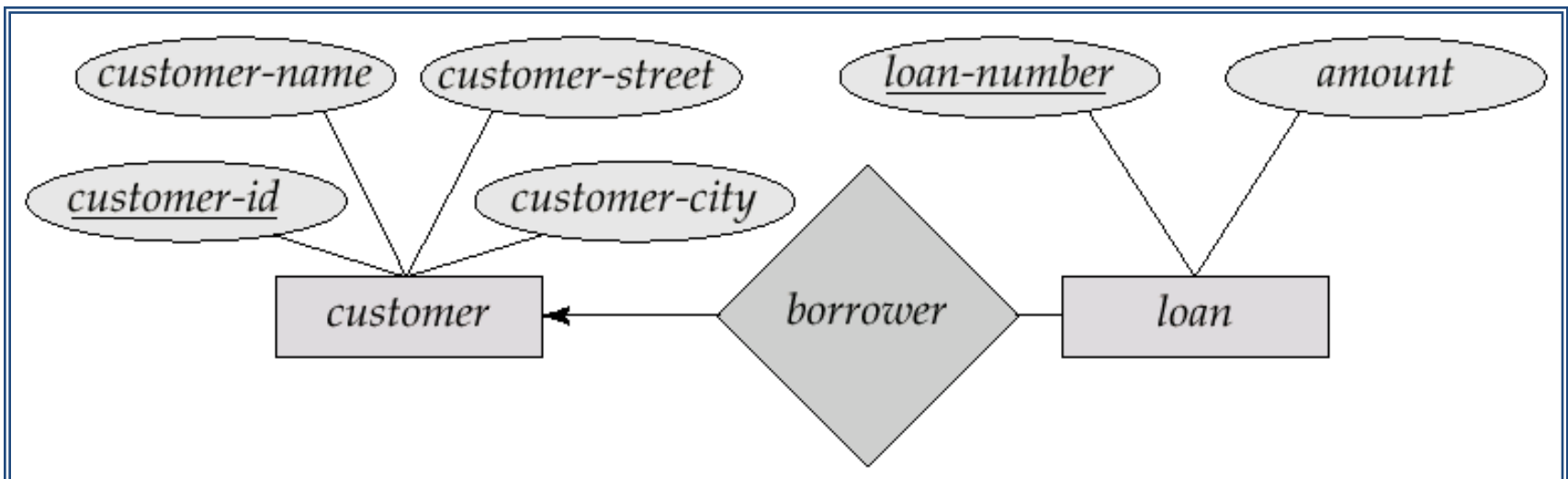
| Symbol   | Meaning   |
|--|---|
|     | Entity  |
|     | Weak Entity   |
|     | Relationship  |
|     | Identifying Relationship  |
|     | Attribute   |
|     | Key Attribute   |
|     | Multivalued Attribute   |
|    | Composite Attribute   |
|     | Derived Attribute   |
|   | Total Participation of $E_2$ in $R$                             |
|  | Cardinality Ratio 1 : N for $E_1:E_2$ in $R$                    |
|  | Structural Constraint (min, max) on Participation of $E$ in $R$ |

# Cardinality Constraints

- We express cardinality constraints by drawing either a directed line (  $\rightarrow$  ), signifying “one,” or an undirected line (—), signifying “many,” between the relationship set and the entity set.
- One-to-one relationship:
  - A customer is associated with at most one loan via the relationship *borrower*
  - A loan is associated with at most one customer via *borrower*

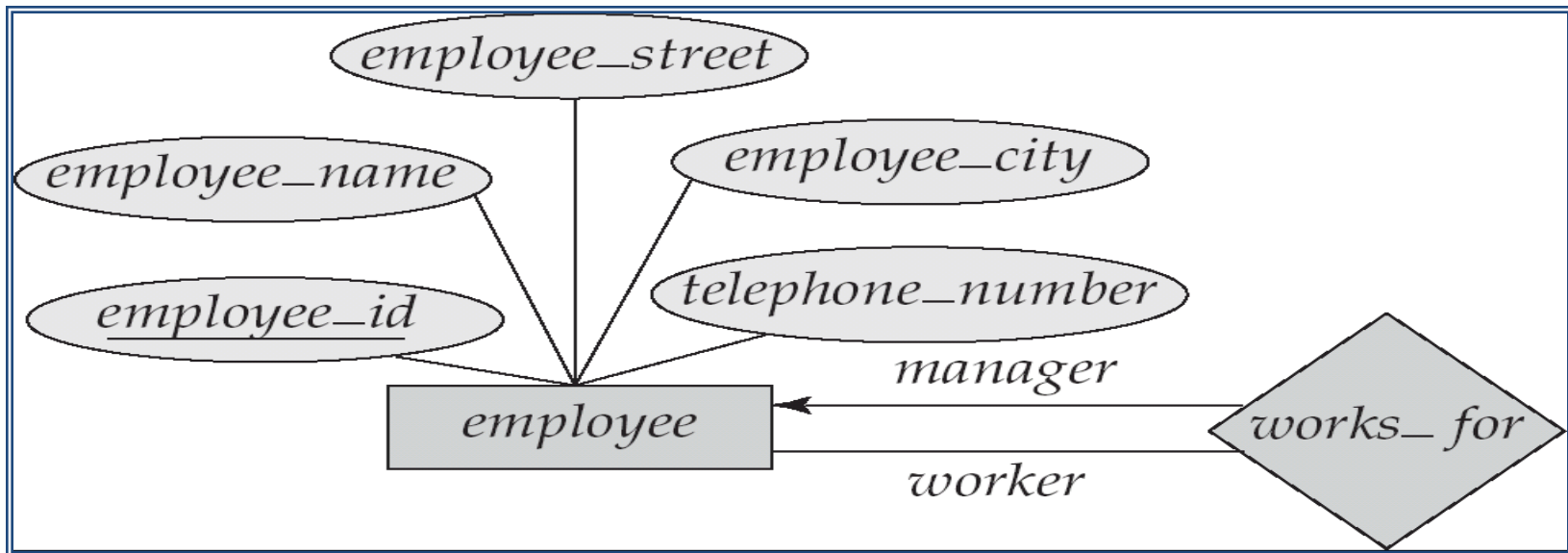
## One-To-Many Relationship

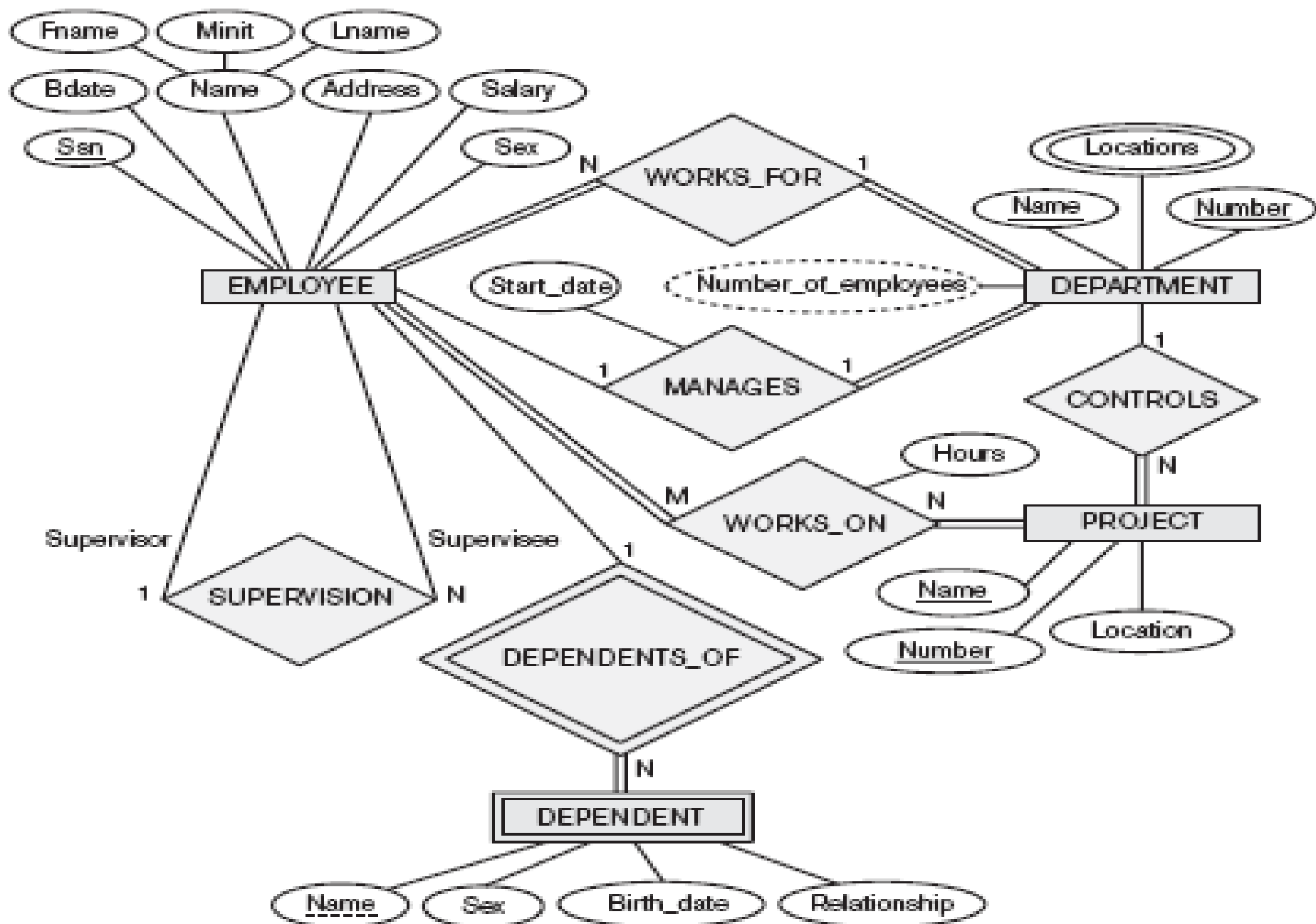
- In the one-to-many relationship a loan is associated with at most one customer via *borrower*, a customer is associated with several (including 0) loans via *borrower*



## Roles

- Entity sets of a relationship need not be distinct.
- The labels “manager” and “worker” are called **roles**; they specify how employee entities interact via the works\_for relationship set.
- Roles are indicated in E-R diagrams by labeling the lines that connect diamonds to rectangles.
- Role labels are optional, and are used to clarify semantics of the relationship



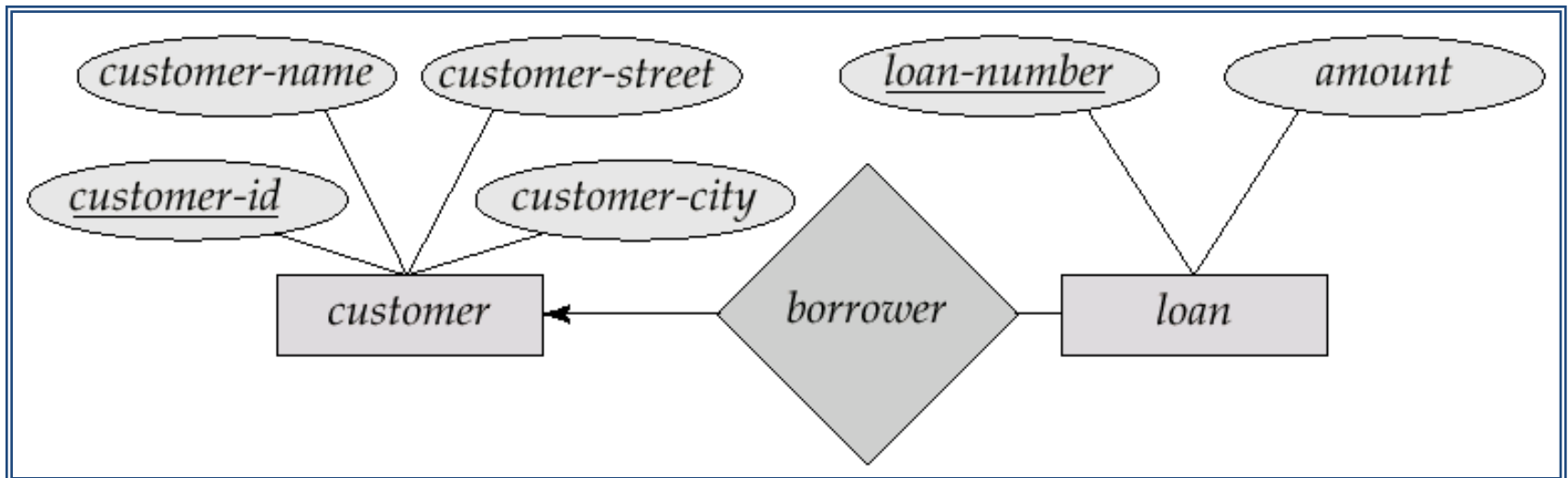


# Cardinality Constraints

- We express cardinality constraints by drawing either a directed line (->), signifying “one,” or an undirected line (—), signifying “many,” between the relationship set and the entity set.
- One-to-one relationship:
  - A customer is associated with at most one loan via the relationship *borrower*
  - A loan is associated with at most one customer via *borrower*

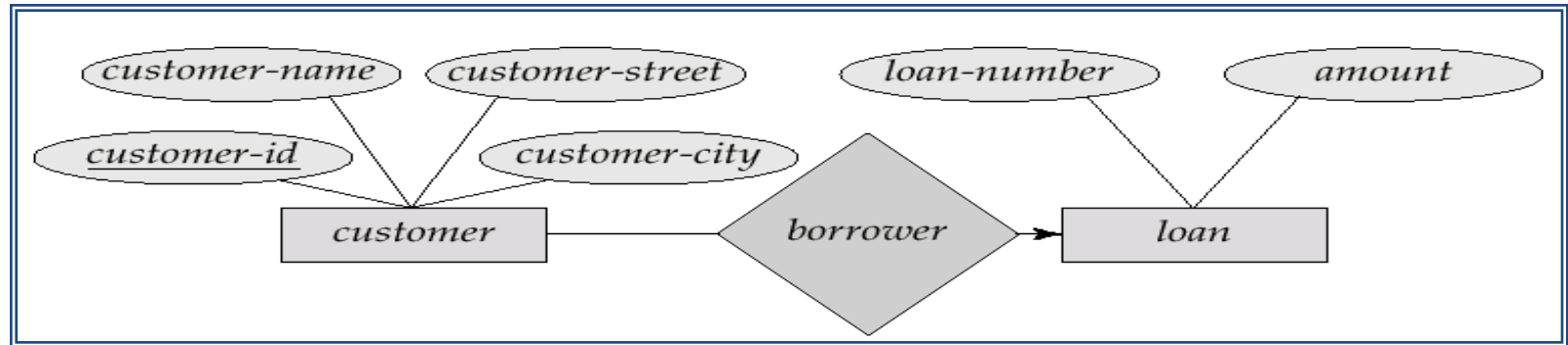
## One-To-Many Relationship

- In the one-to-many relationship a loan is associated with at most one customer via *borrower*, a customer is associated with several (including 0) loans via *borrower*



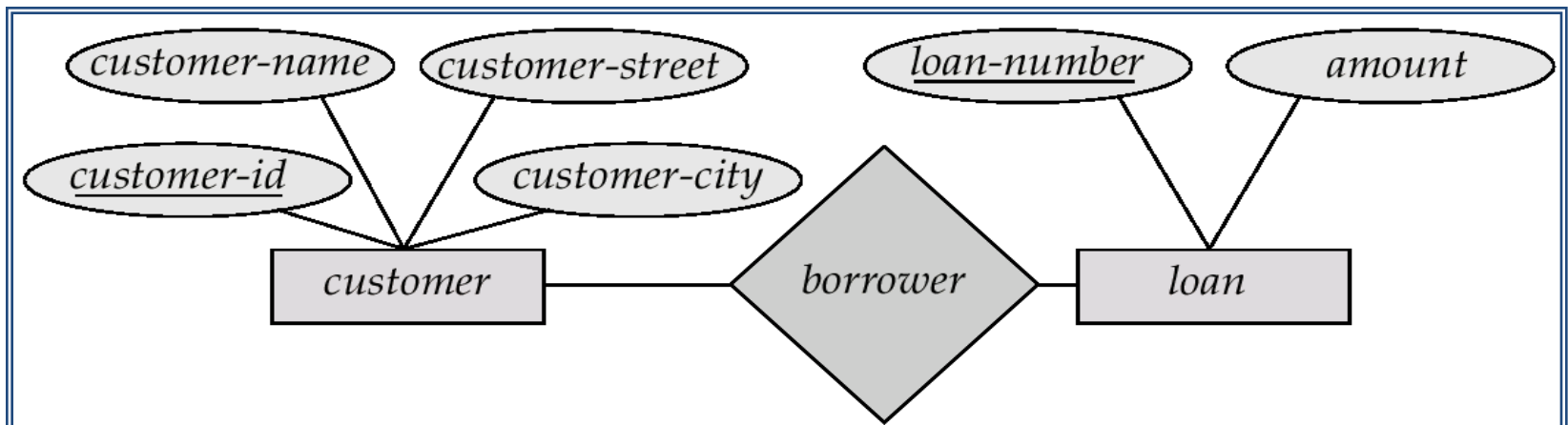
## Many-To-One Relationships

- In a many-to-one relationship a loan is associated with several (including 0) customers via *borrower*, a customer is associated with at most one loan via *borrower*



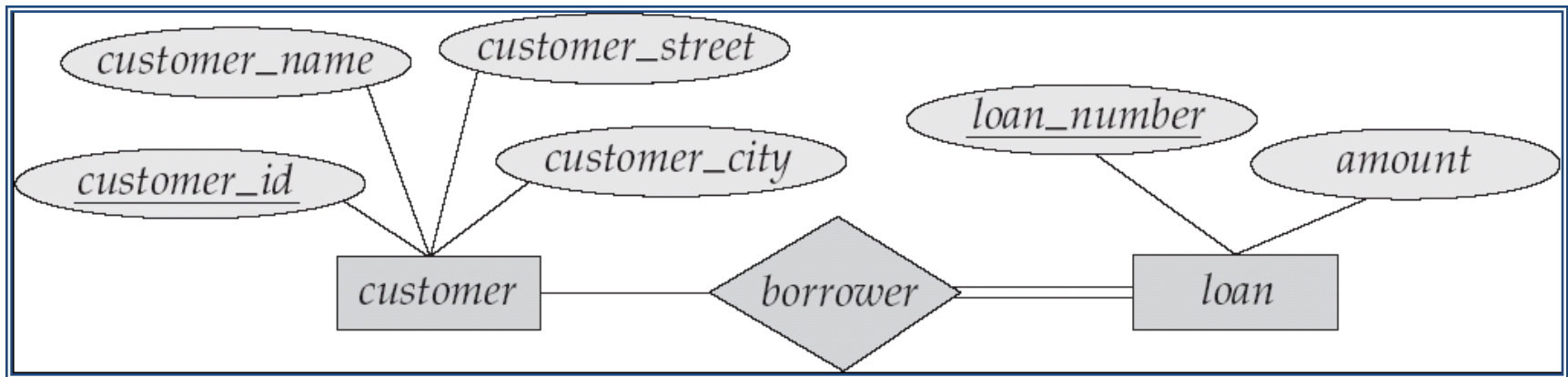
## Many-To-Many Relationship

- A customer is associated with several (possibly 0) loans via *borrower*
- A loan is associated with several (possibly 0) customers via *borrower*



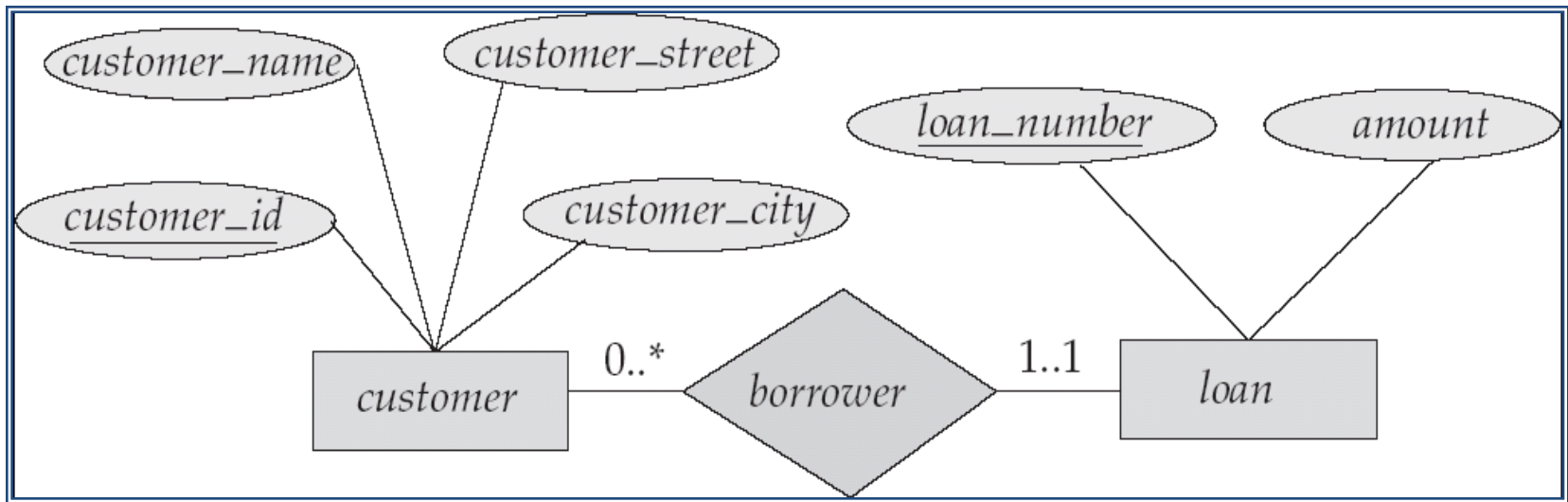
# Participation of an Entity Set in a Relationship Set

- Total participation (indicated by double line): every entity in the entity set participates in at least one relationship in the relationship set
  - E.g. participation of loan in borrower is total
    - ▶ every loan must have a customer associated to it via borrower
- Partial participation: some entities may not participate in any relationship in the relationship set
  - Example: participation of customer in borrower is partial



# Alternative Notation for Cardinality Limits

- Cardinality limits can also express participation constraints



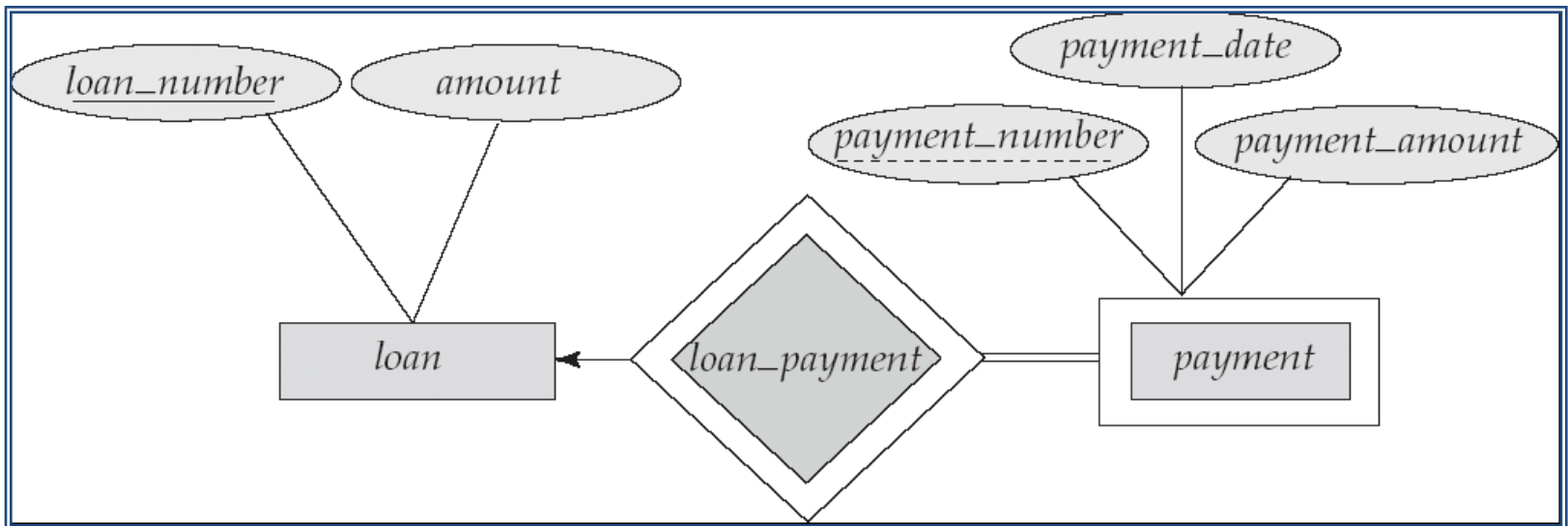


## Weak Entity Sets

- An entity set that does not have a primary key is referred to as a **weak entity set**.
- The existence of a weak entity set depends on the existence of a **identifying entity set**
  - It must relate to the identifying entity set via a total, one-to-many relationship set from the identifying to the weak entity set
  - Identifying relationship depicted using a double diamond
- The **discriminator** (or partial key) of a weak entity set is the set of attributes that distinguishes among all the entities of a weak entity set.
- The primary key of a weak entity set is formed by the primary key of the strong entity set on which the weak entity set is existence dependent, plus the weak entity set's discriminator.

## Weak Entity Sets ...contd.

- We depict a weak entity set by double rectangles.
- We underline the discriminator of a weak entity set with a dashed line.
- `payment_number` – discriminator of the *payment* entity set
- Primary key for *payment* – (`loan_number`, `payment_number`)



# Reduction to Relation Schemas

- Primary keys allow entity sets and relationship sets to be expressed uniformly as *relation schemas* that represent the contents of the database.
- A database which conforms to an E-R diagram can be represented by a collection of schemas.
- For each entity set and relationship set there is a unique schema that is assigned the name of the corresponding entity set or relationship set.
- Each schema has a number of columns (generally corresponding to attributes), which have unique names.

## Representing Entity Sets as Schemas

- A **strong entity** set reduces to a schema with the same attributes.
- A **weak entity** set becomes a table that includes a column for the primary key of the identifying strong entity set  
payment =  
( loan\_number, payment\_number, payment\_date, payment\_amount )

## Representing Relationship Sets as Schemas

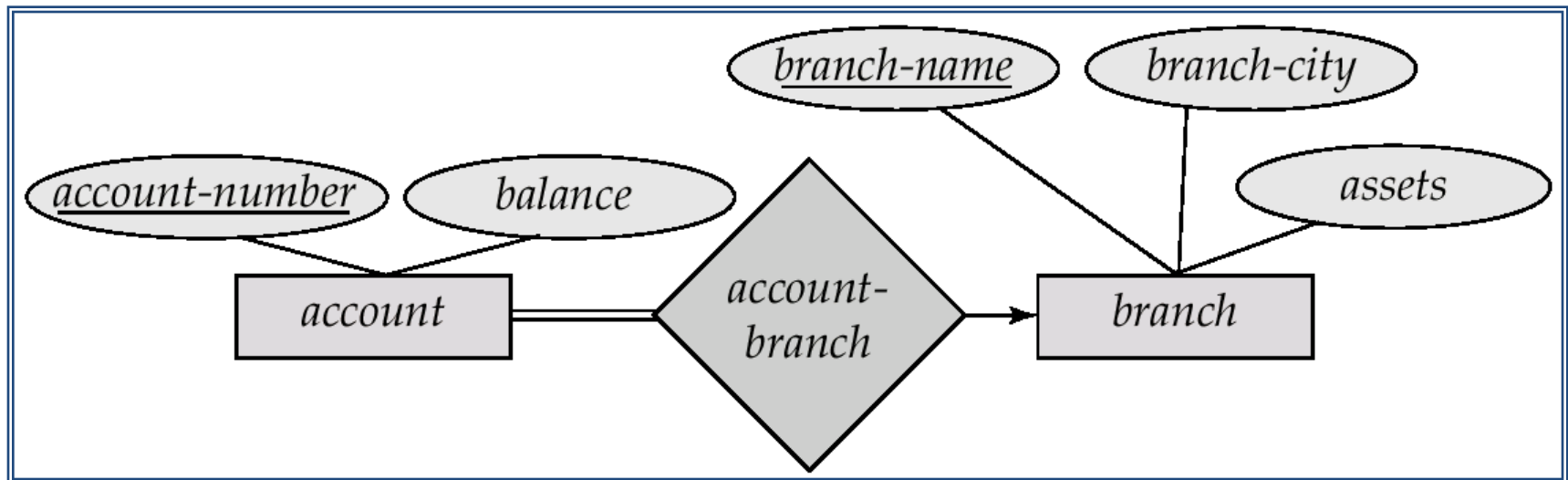
- A **many-to-many relationship** set is represented as a schema with attributes for the primary keys of the two participating entity sets, and any descriptive attributes of the relationship set.
- Example: schema for relationship set borrower  
*borrower = (customer\_id, loan\_number)*

## Redundancy of Schemas...contd.

- For one-to-one relationship sets, either side can be chosen to act as the “many” side.
  - That is, extra attribute can be added to either of the tables corresponding to the two entity sets.
- If participation is *partial* on the “many” side, replacing a schema by an extra attribute in the schema corresponding to the “many” side could result in null values
- The schema corresponding to a relationship set linking a weak entity set to its identifying strong entity set is redundant.
  - Example: The payment schema already contains the attributes that would appear in the loan\_payment schema (i.e., loan\_number and payment\_number).

# Redundancy of Schemas

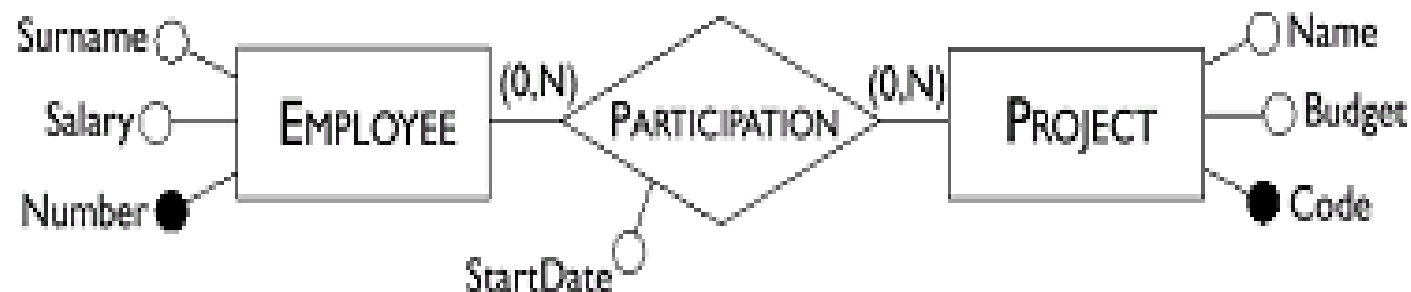
- **Many-to-one** and **one-to-many** relationship sets that are total on the many-side can be represented by adding an extra attribute to the “many” side, containing the primary key of the “one” side
- **Example:** Instead of creating a schema for relationship set `account_branch`, add an attribute `branch_name` to the schema arising from entity set `account`



# Composite and Multivalued Attributes

- **Composite attributes are flattened out by creating a separate attribute for each component attribute**
  - Example: given entity set *customer* with composite attribute name with component attributes *first\_name* and *last\_name* the schema corresponding to the entity set has two attributee *name.first\_name* and *name.last\_name*
- **A multivalued attribute *M* of an entity *E* is represented by a separate schema *EM***
  - Schema *EM* has attributes corresponding to the primary key of *E* and an attribute corresponding to multivalued attribute *M*
  - Example: Multivalued attribute *dependent\_names* of *employee* is represented by a schema:  
$$employee\_dependent\_names = ( \underline{employee\_id}, dname )$$
  - Each value of the multivalued attribute maps to a separate tuple of the relation on schema *EM*  
  
For example, an employee entity with primary key 123-45-6789 and dependents Jack and Jane maps to two tuples:  
  
(123-45-6789 , Jack) and (123-45-6789 , Jane)

# Many-to-Many Relationships



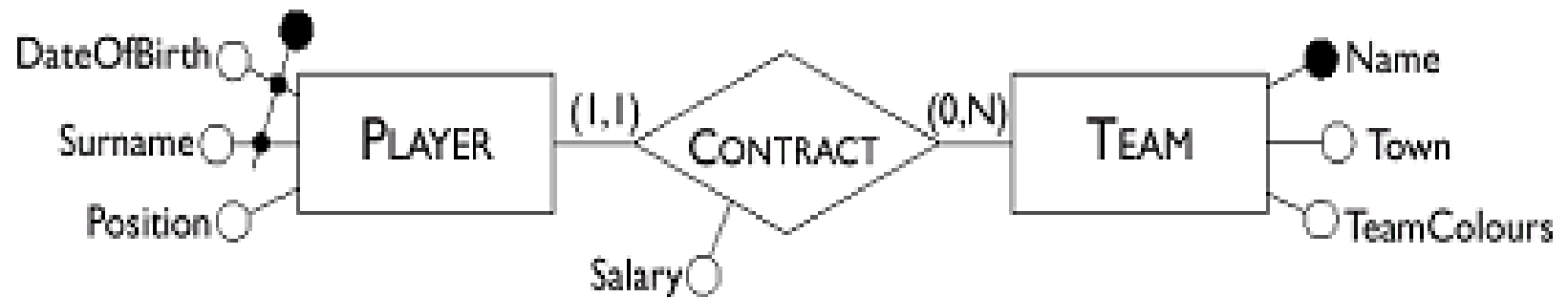
Employee(Number, Surname, Salary)

Project(Code, Name, Budget)

Participation(Number, Code, StartDate)



# One-to-Many Relationships



Player(Surname, DateOfBirth, Position)

Team(Name, Town, TeamColours)

Contract(PlayerSurname, PlayerDateOfBirth, Team, Salary)

*OR*

Player(Surname, DateOfBirth, Position, TeamName, Salary)

Team(Name, Town, TeamColours)

# One-to-One Relationships



Head(Number, Name, Salary, Department, StartDate)

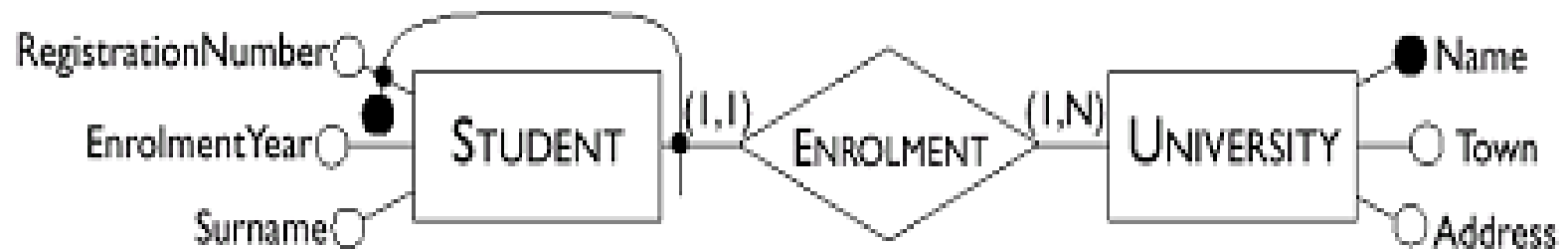
Department(Name, Telephone, Branch)

*Or*

Head(Number, Name, Salary, StartDate)

Department(Name, Telephone, HeadNumber, Branch)

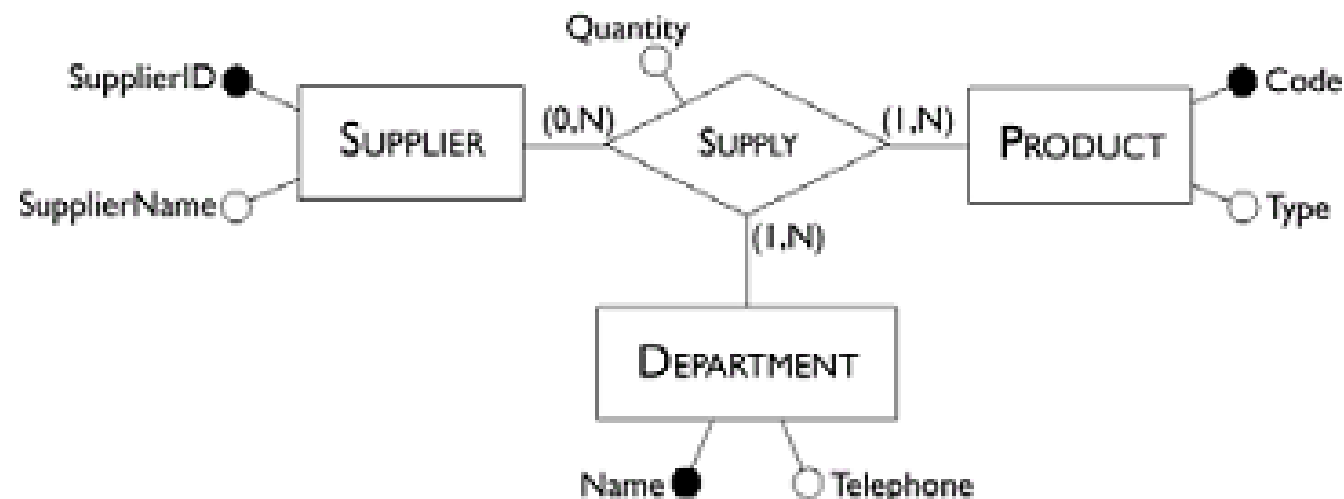
# Weak Entities



Student(RegistrationNumber, University, Surname, EnrolmentYear)

University(Name, Town, Address)

# Ternary Relationships



Supplier(SupplierID, SupplierName)

Product(Code, Type)

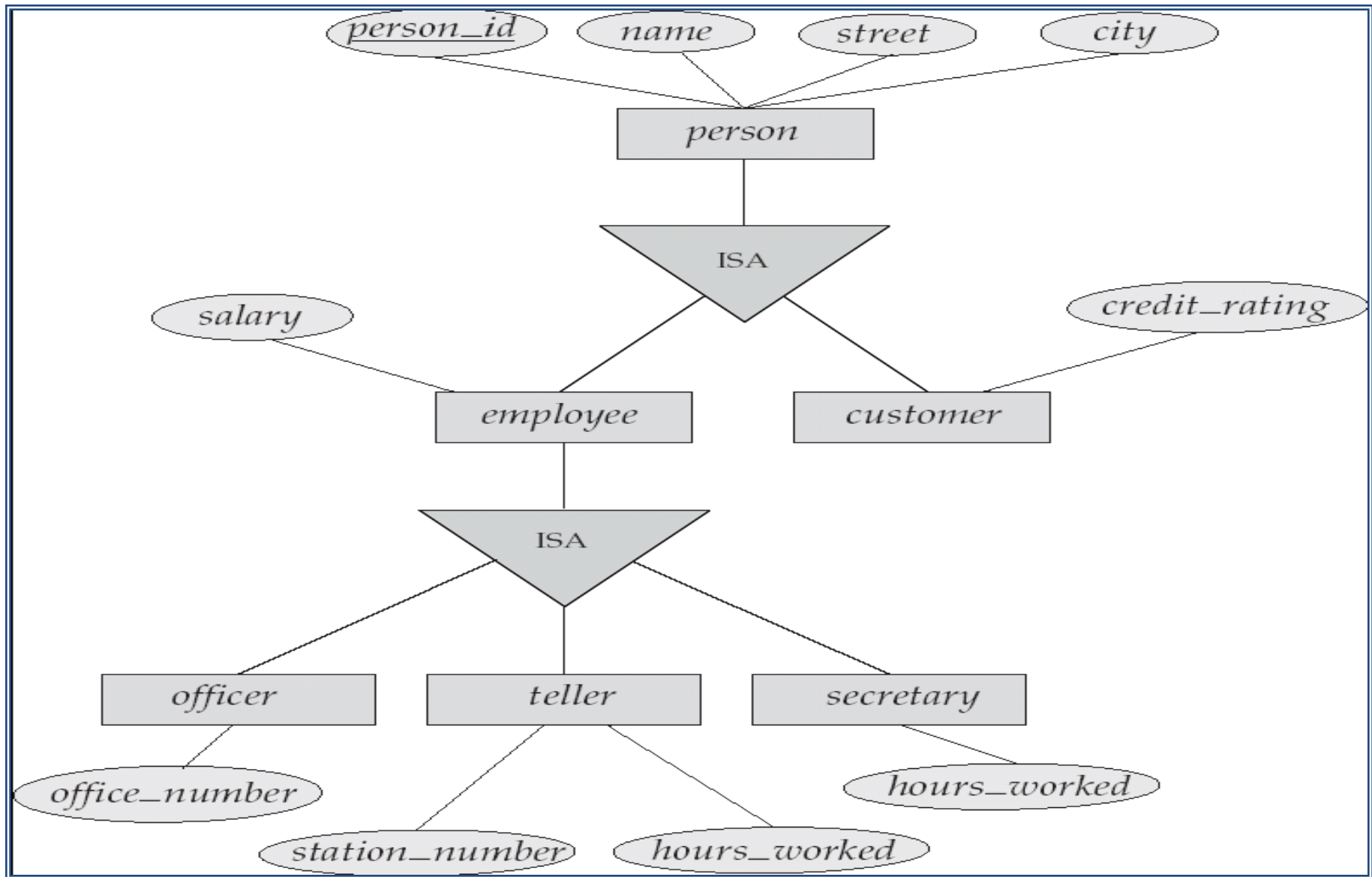
Department(Name, Telephone)

Supply(Supplier, Product, Department, Quantity)

## Extended E-R Features: Specialization

- Top-down design process; we designate subgroupings within an entity set that are distinctive from other entities in the set.
- These subgroupings become lower-level entity sets that have attributes or participate in relationships that do not apply to the higher-level entity set.
- Depicted by a *triangle* component labeled ISA (E.g. *customer “is a” person*).
- **Attribute inheritance** – a lower-level entity set inherits all the attributes and relationship participation of the higher-level entity set to which it is linked.

# Specialization



## Extended ER Features: Generalization

- A **bottom-up design process** – combine a number of entity sets that share the same features into a higher-level entity set.
- Specialization and generalization are simple inversions of each other; they are represented in an E-R diagram in the same way.
- The terms specialization and generalization are used interchangeably.

### Specialization and Generalization

- Can have multiple specializations of an entity set based on different features.
- E.g. **permanent\_employee** vs. **temporary\_employee**, in addition to **officer** vs. **secretary** vs. **teller**
- Each particular employee would be
  - a member of one of **permanent\_employee** or **temporary\_employee**,
  - and also a member of one of **officer**, **secretary**, or **teller**
- The ISA relationship also referred to as **superclass - subclass** relationship

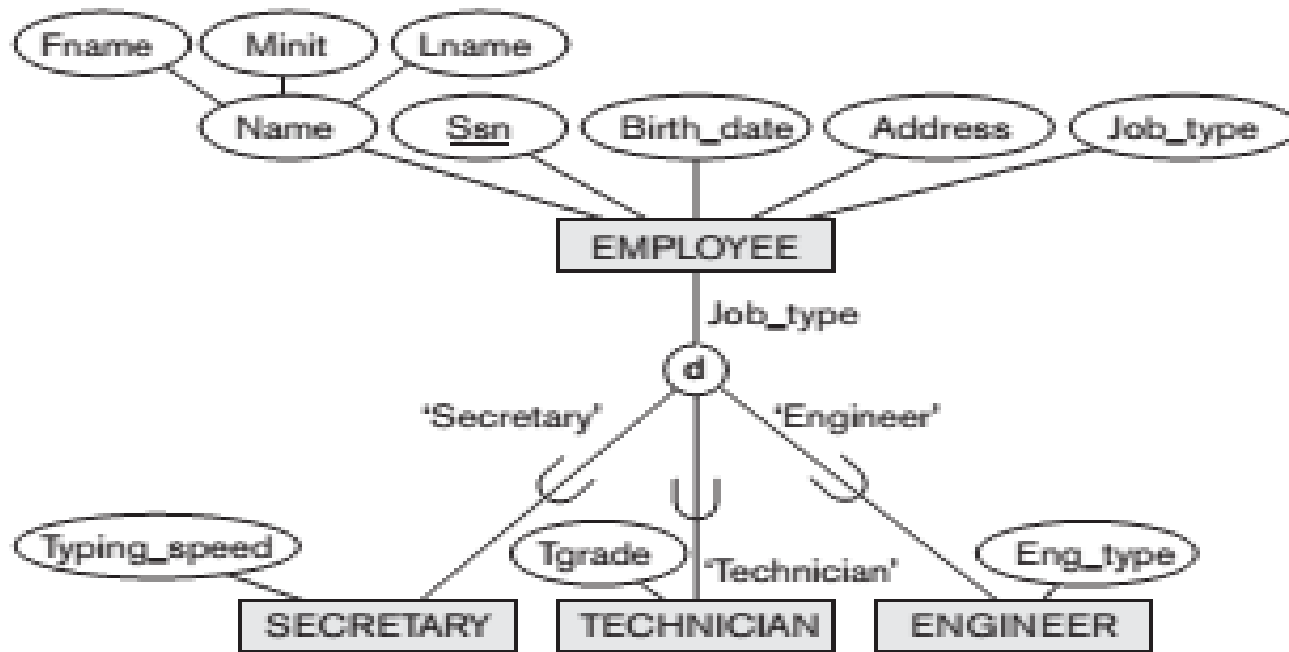
# Design Constraints on a Specialization/Generalization

- **Constraint on which entities can be members of a given lower-level entity set.**
  - **Condition-defined**

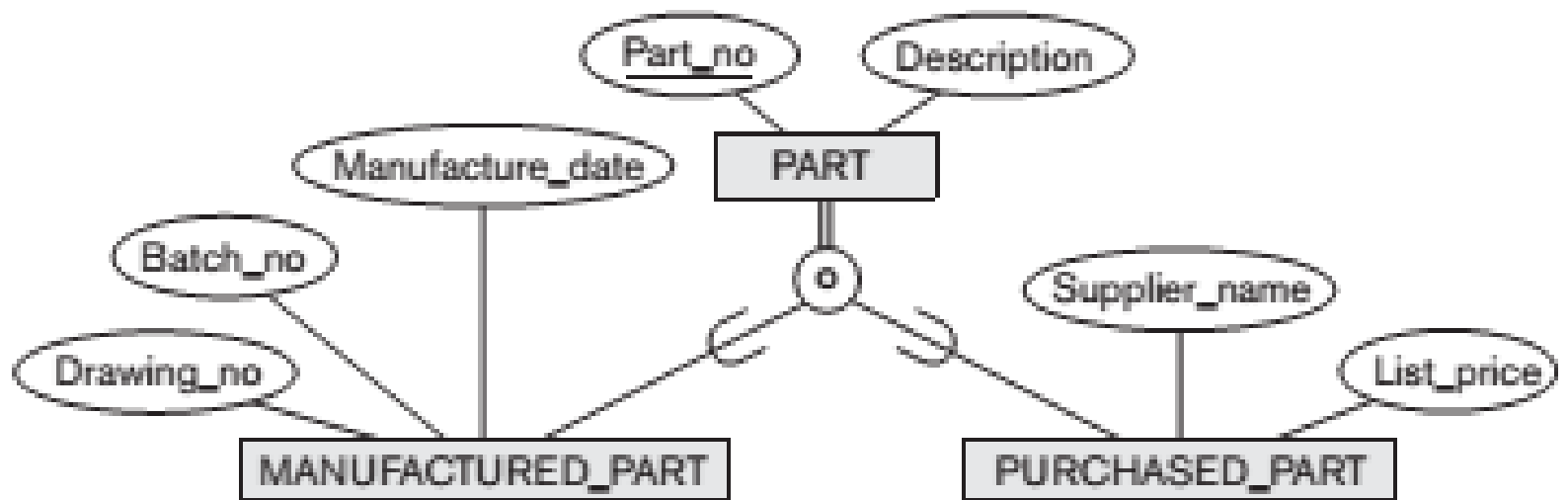
Example: all customers over 65 years are members of *senior-citizen* entity set; *senior-citizen* ISA *person*.
  - **User-defined**
- **Constraint on whether or not entities may belong to more than one lower-level entity set within a single generalization.**
  - **Disjoint**
    - An entity can belong to only one lower-level entity set
    - Noted in E-R diagram by writing *disjoint* next to the ISA triangle
  - **Overlapping**
    - An entity can belong to more than one lower-level entity set
- **Completeness constraint** -- specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within a generalization.
  - **Total** : an entity must belong to one of the lower-level entity sets
  - **Partial**: an entity need not belong to one of the lower-level entity sets



**Attribute Defined or Condition Defined Subclasses:** In some specializations we can determine exactly the entities that will become members of each subclass by placing a condition on the value of some attribute of the superclass. Such subclasses are called predicate-defined (or condition-defined) subclasses.



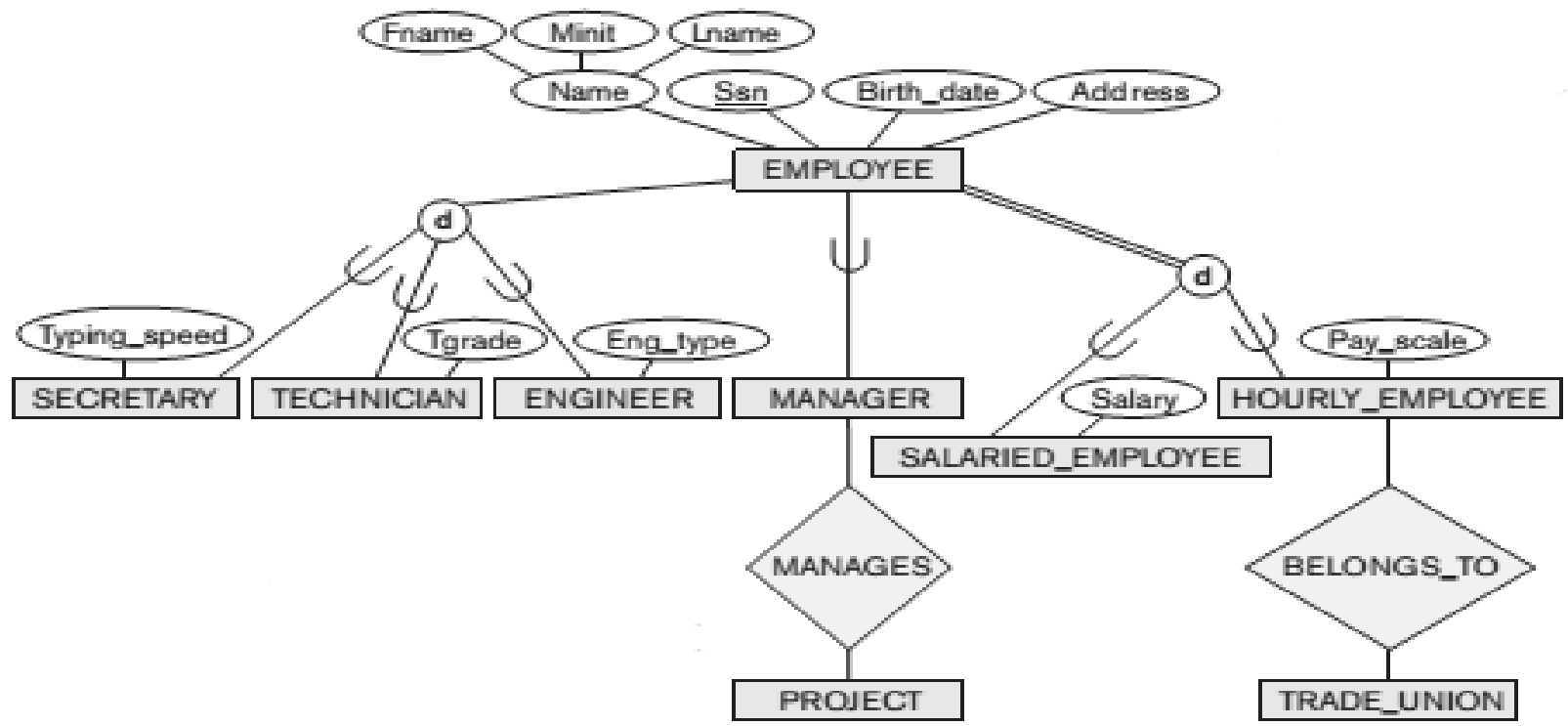
**User Defined Subclass:** When we do not have a condition for determining membership in a subclass, the subclass is called user-defined. Membership in such a subclass is determined by the database users when they apply the operation to add an entity to the subclass.

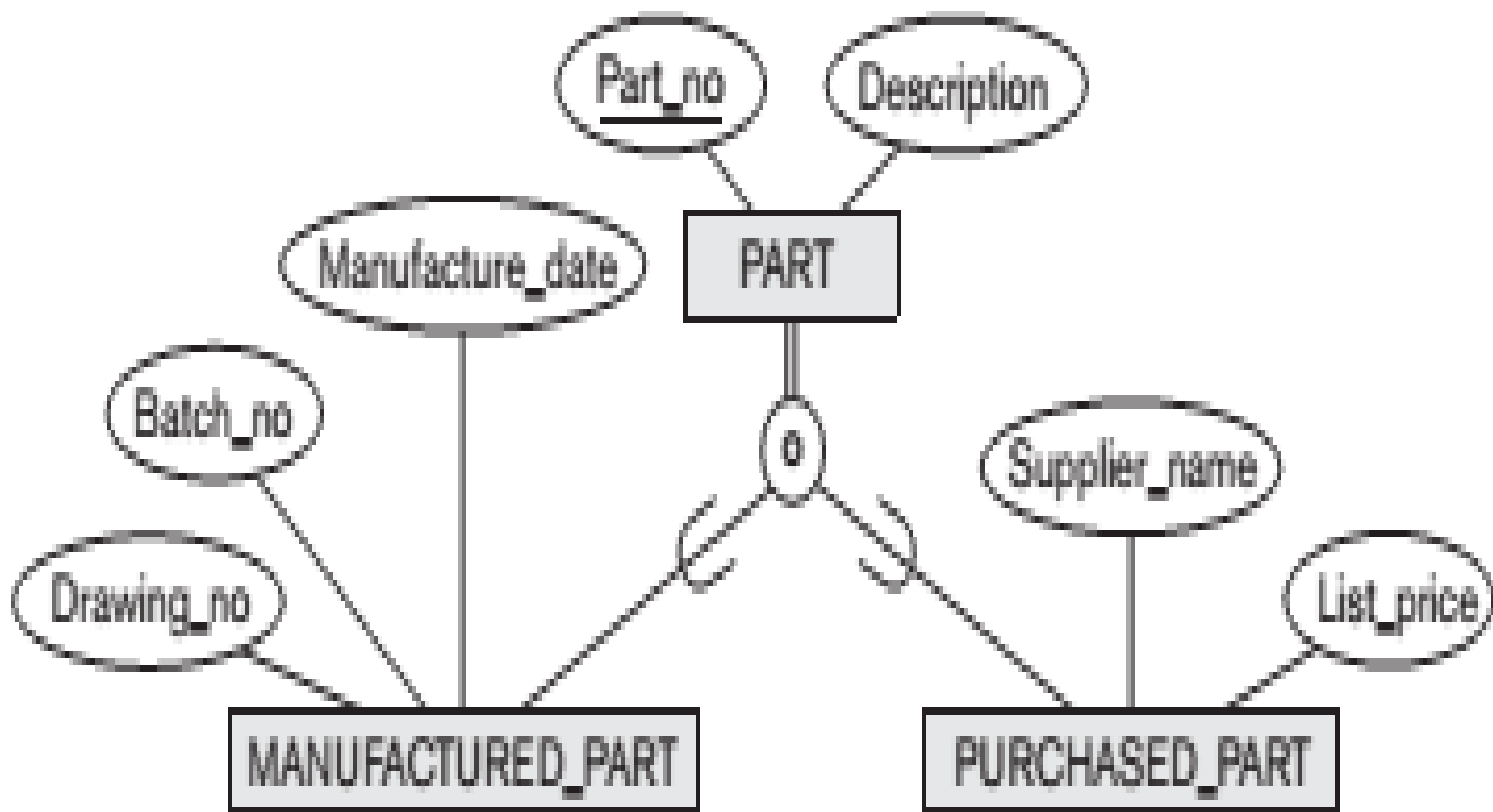


**A total specialization constraint** specifies that every entity in the superclass must be a member of at least one subclass in the specialization. A double line is used to display a partial specialization

E.g.: every EMPLOYEE must be either an HOURLY\_EMPLOYEE or a SALARIED\_EMPLOYEE

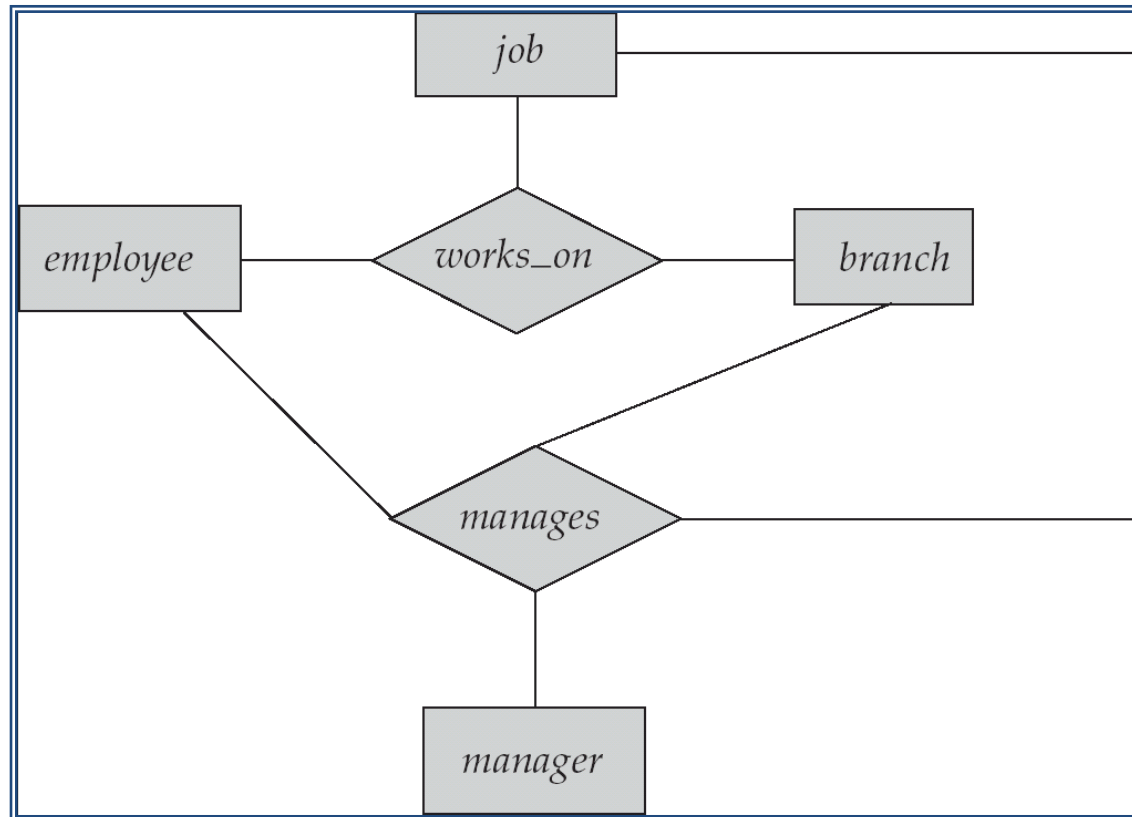
**Partial Participation** : Which allows an entity not to belong to any of the subclasses. A single line is used to display a partial specialization





# Aggregation

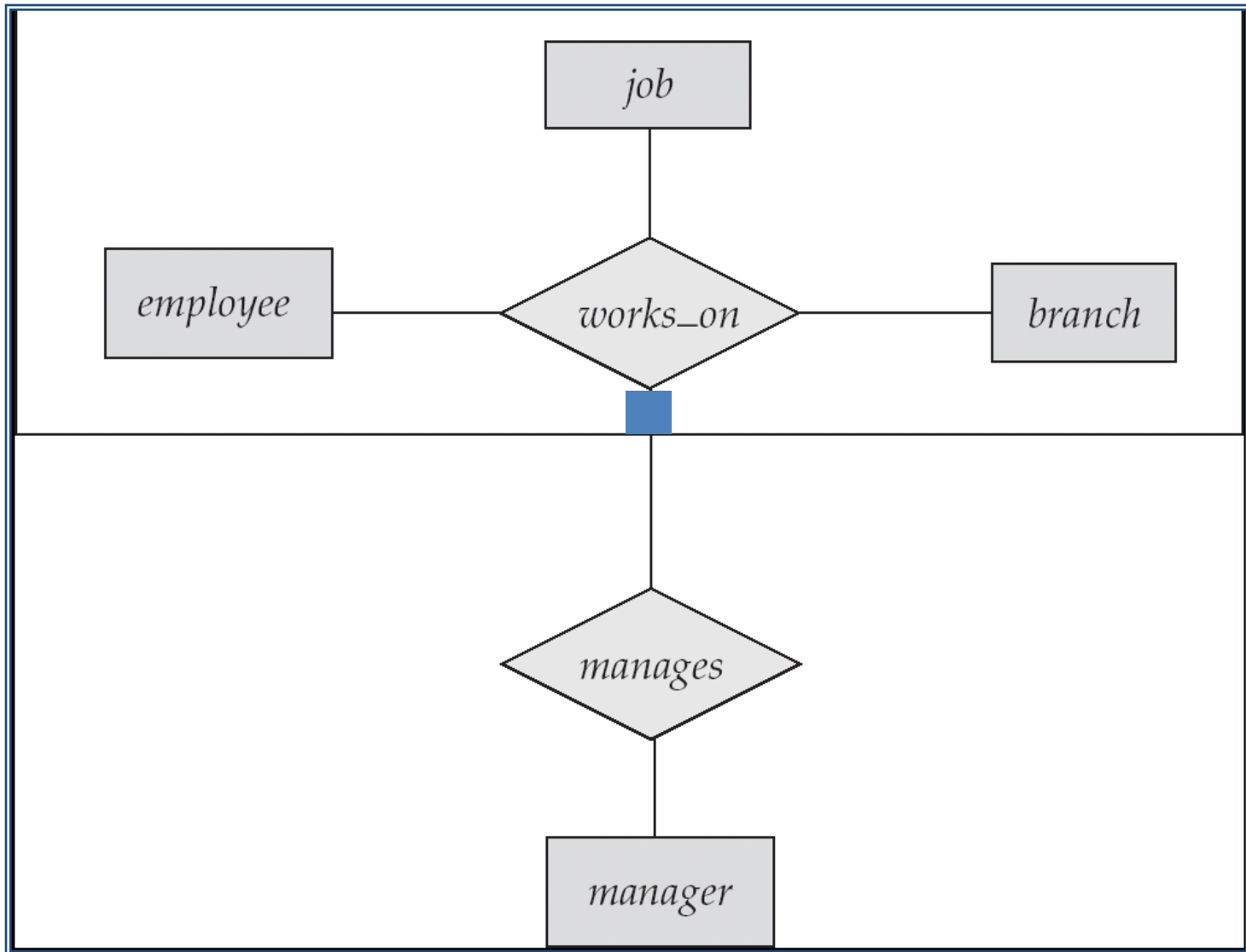
- Consider the ternary relationship *works\_on*, which we saw earlier
- Suppose we want to record managers for tasks performed by an employee at a branch



## Aggregation . . . contd.

- Relationship sets *works\_on* and *manages* represent overlapping information
  - Every *manages* relationship corresponds to a *works\_on* relationship
  - However, some *works\_on* relationships may not correspond to any *manages* relationships
    - So we can't discard the *works\_on* relationship
- Eliminate this redundancy via *aggregation*
  - Treat relationship as an abstract entity
  - Allows relationships between relationships
  - Abstraction of relationship into new entity
- Without introducing redundancy, the following diagram represents:
  - An employee works on a particular job at a particular branch
  - An employee, branch, job combination may have an associated manager

## E-R Diagram With Aggregation



# Representing Specialization via Schemas

- Method 1:
  - Form a schema for the higher-level entity
  - Form a schema for each lower-level entity set, include primary key of higher-level entity set and local attributes

| schema          | attributes                 |
|-----------------|----------------------------|
| <i>person</i>   | <i>name, street, city</i>  |
| <i>customer</i> | <i>name, credit_rating</i> |
| <i>employee</i> | <i>name, salary</i>        |

- Drawback: getting information about, an *employee* requires accessing two relations, the one corresponding to the low-level schema and the one corresponding to the high-level schema



# Representing Specialization as Schemas (Cont.)

- **Method 2:**
  - **Form a schema for each entity set with all local and inherited attributes**

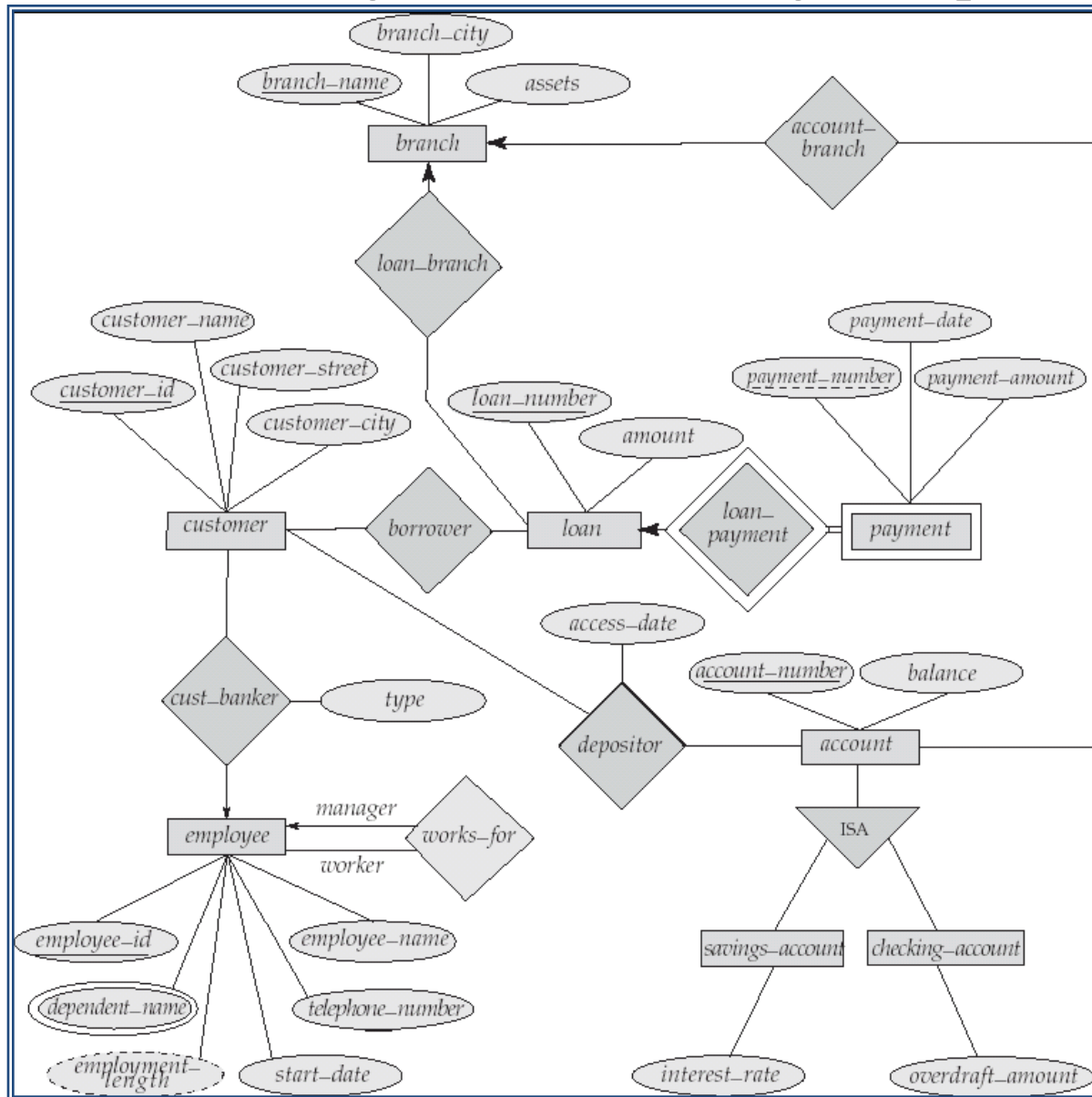
| <b>schema</b>   | <b>attributes</b>                        |
|-----------------|--|
| <i>person</i>   | <i>name, street, city</i>                |
| <i>customer</i> | <i>name, street, city, credit_rating</i> |
| <i>employee</i> | <i>name, street, city, salary</i>        |

- **If specialization is total, the schema for the generalized entity set (*person*) not required to store information**
  - **Can be defined as a “view” relation containing union of specialization relations**
  - **But explicit schema may still be needed for foreign key constraints**
- **Drawback: *street* and *city* may be stored redundantly for people who are both customers and employees**

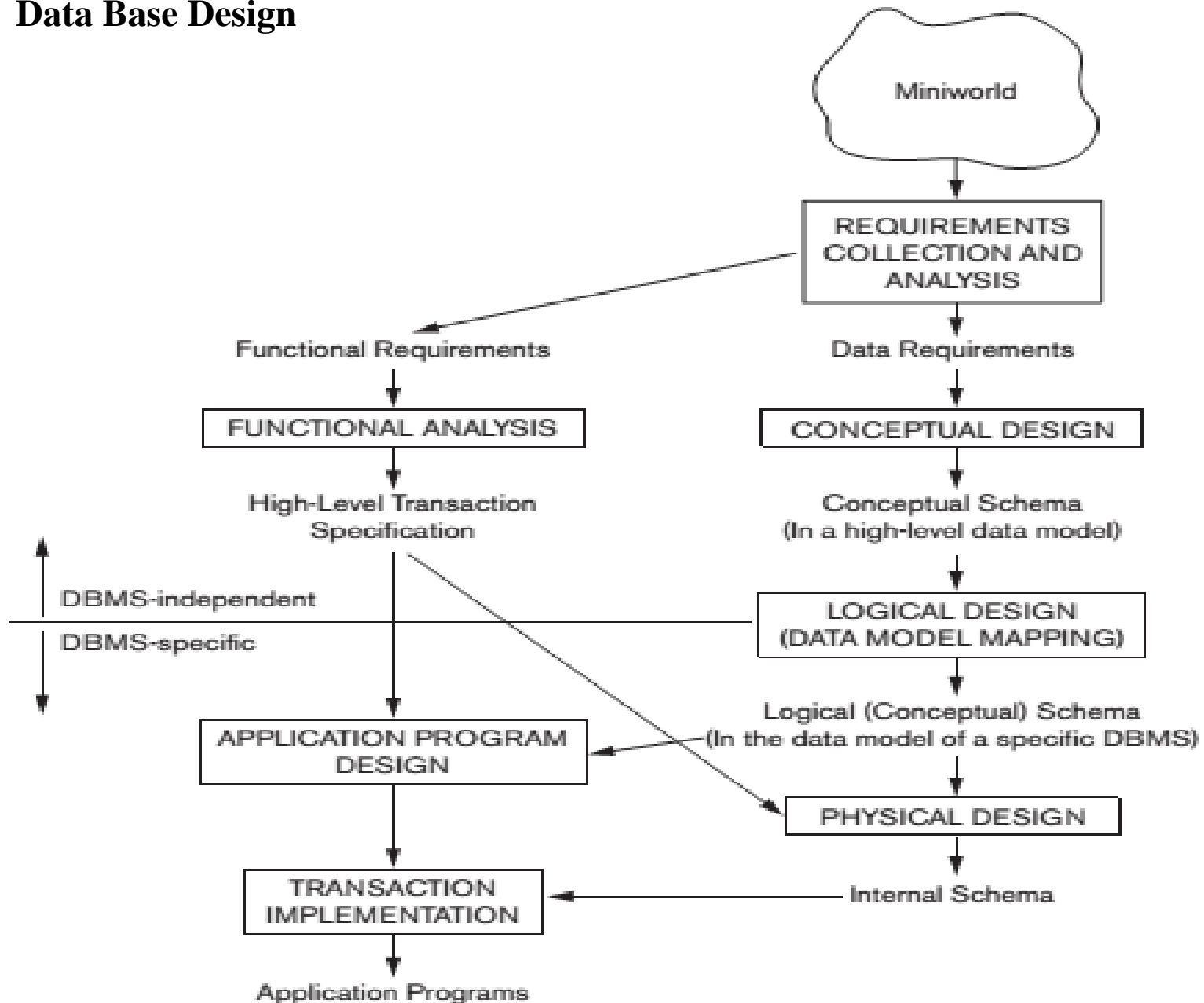
# E-R Design Decisions

- The use of an attribute or entity set to represent an object.
- Whether a real-world concept is best expressed by an entity set or a relationship set.
- The use of a ternary relationship versus a pair of binary relationships.
- The use of a strong or weak entity set.
- The use of specialization/generalization – contributes to modularity in the design.
- The use of aggregation – can treat the aggregate entity set as a single unit without concern for the details of its internal structure.

# E-R Diagram for a Banking Enterprise



# Data Base Design



# Relational Database Design

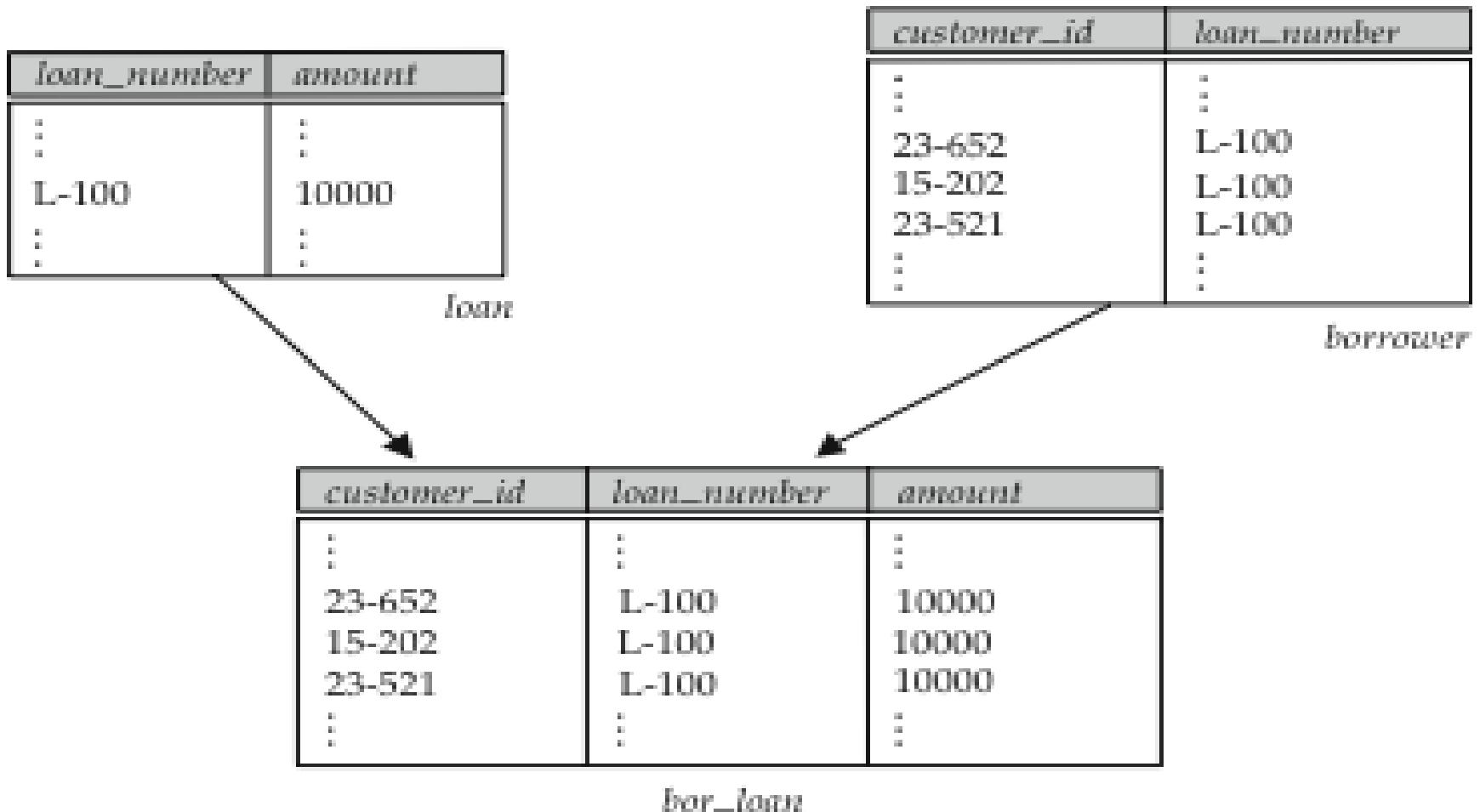
- **Features of Good Relational Design**
- **Atomic Domains and First Normal Form**
- **Decomposition Using Functional Dependencies**
- **Functional Dependency Theory**
- **Algorithms for Functional Dependencies**
- **Decomposition Using Multivalued Dependencies**
- **More Normal Form**
- **Database-Design Process**
- **Modeling Temporal Data**

# The Banking Schema

- branch = (branch\_name, branch\_city, assets)
- customer = (customer\_id, customer\_name, customer\_street, customer\_city)
- loan = (loan\_number, amount)
- account = (account\_number, balance)
- employee = (employee\_id, employee\_name, telephone\_number, start\_date)
- dependent\_name = (employee\_id, dname)
- account\_branch = (account\_number, branch\_name)
- loan\_branch = (loan\_number, branch\_name)
- borrower = (customer\_id, loan\_number)
- depositor = (customer\_id, account\_number)
- cust\_banker = (customer\_id, employee\_id, type)
- works\_for = (worker\_employee\_id, manager\_employee\_id)
- payment = (loan\_number, payment\_number, payment\_date, payment\_amount)
- savings\_account = (account\_number, interest\_rate)
- checking\_account = (account\_number, overdraft\_amount)

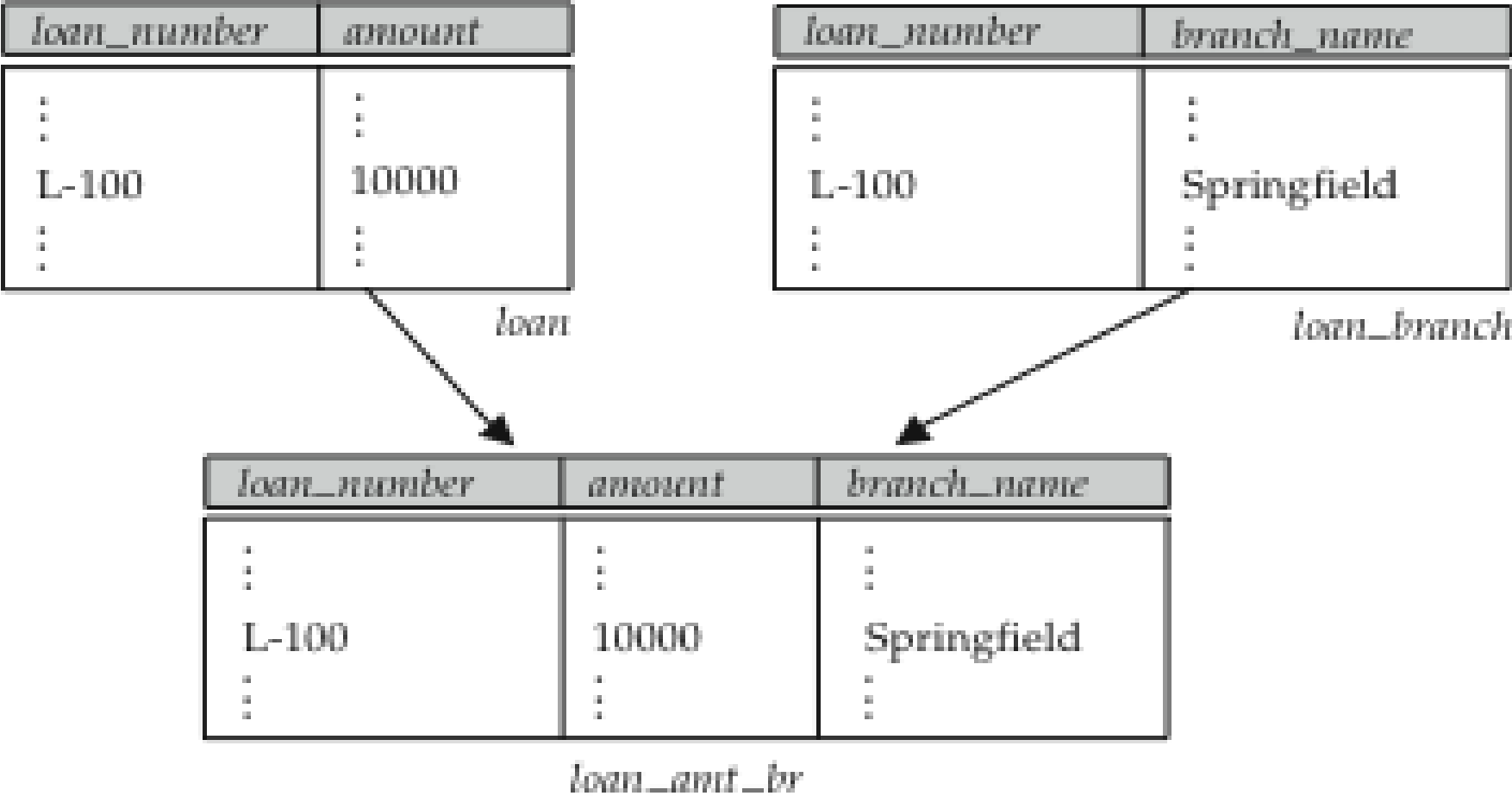
# Design Alternatives: Combine Schemas

- Suppose we combine borrower and loan to get  
**bor\_loan = (customer\_id, loan\_number, amount )**
- Result is possible repetition of information (L-100 in example below)



# Design Alternatives: A Combined Schema Without Repetition

- Consider combining `loan_branch` and `loan`  
`loan_amt_br = (loan_number, amount, branch_name)`
- No repetition but we have to create tuple with null value for amount

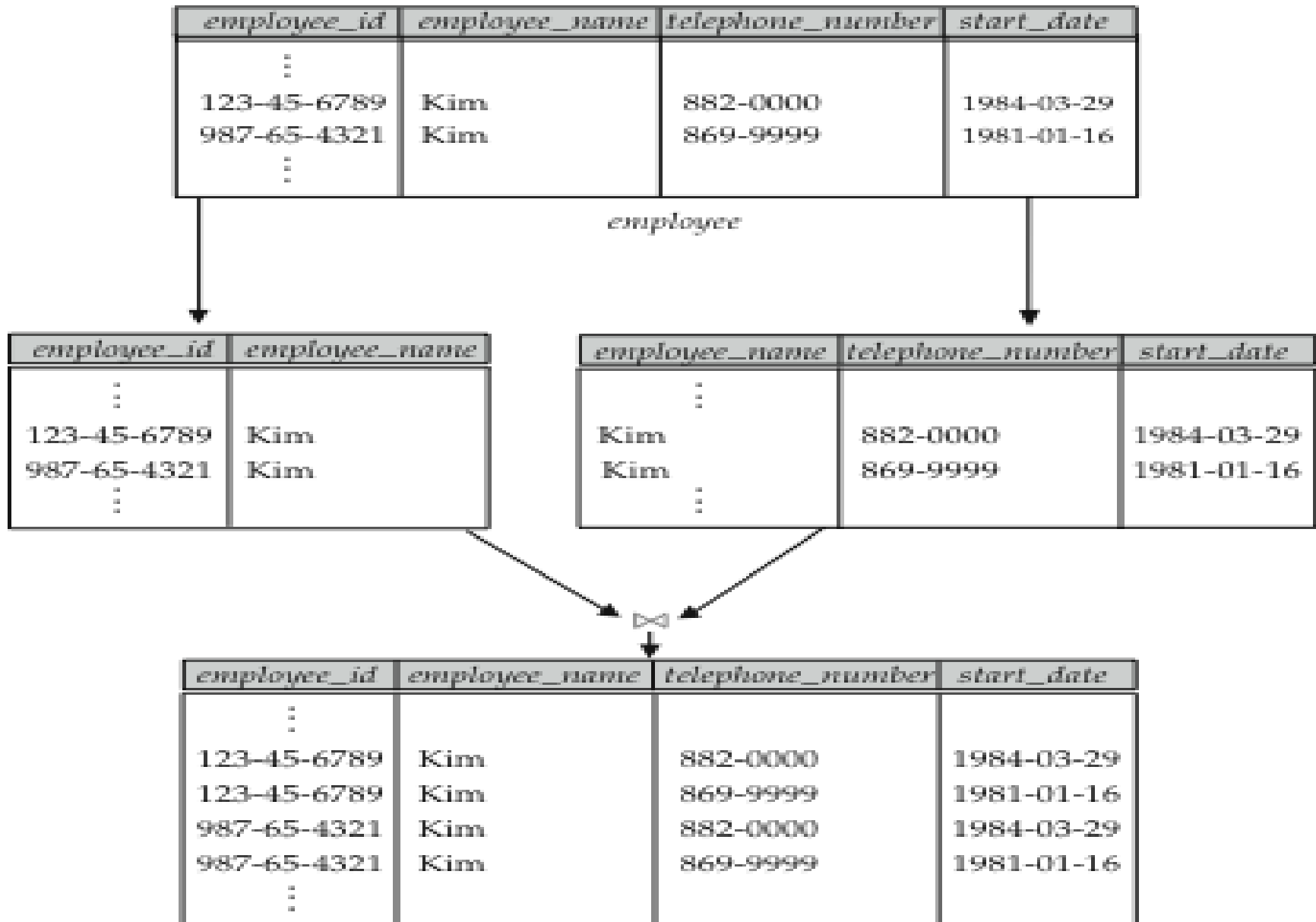




## Design Alternatives: Smaller Schemas

- Suppose we had started with *bor\_loan*. How would we know to split up (decompose) it into *borrower* and *loan*?
- Write a rule “if there were a schema (*loan\_number*, *amount*), then *loan\_number* would be a candidate key”
- Denote as a functional dependency:  
$$\text{loan\_number} \rightarrow \text{amount}$$
- In *bor\_loan*, because *loan\_number* is not a candidate key, the amount of a loan may have to be repeated. This indicates the need to decompose *bor\_loan*.
- **Not all decompositions are good.** Suppose we decompose *employee* into  
*employee1* = (*employee\_id*, *employee\_name*)  
*employee2* = (*employee\_name*, *telephone\_number*, *start\_date*)
- The next slide shows how we lose information - we cannot reconstruct the original *employee* relation -- and so, this is a lossy decomposition.

# A Lossy Decomposition



# Guideline 1

- Design a relation schema so that it is easy to explain its meaning. Do not combine attributes from multiple entity types and relationship types into a single relation. Intuitively, If a relation schema corresponds to one entity type or one relationship type, it is straightforward to interpret and to explain its meaning . Otherwise, if the relation corresponds to mixture of multiple entities and relationships, Semantic ambiguities will result and the relation cannot be easily explained

# First Normal Form

- **Domain is atomic if its elements are considered to be indivisible units**
  - **Examples of non-atomic domains:**
    - **Set of names, composite attributes**
    - **Identification numbers like CS101 that can be broken up into parts**
- **A relational schema R is in first normal form if the domains of all attributes of R are atomic.**
- **Non-atomic values complicate storage and encourage redundant (repeated) storage of data**
  - **E.g.: Set of accounts stored with each customer, and set of owners stored with each account**
  - **We assume all relations are in first normal form**

## **First Normal Form...contd.**

- **Atomicity is actually a property of how the elements of the domain are used.**
  - **E.g.: Strings would normally be considered indivisible**
  - **Suppose that students are given roll numbers which are strings of the form *CS0012* or *EE1127***
  - **If the first two characters are extracted to find the department, the domain of roll numbers is not atomic.**
  - **Doing so is a bad idea: leads to encoding of information in application program rather than in the database.**

# Goal — Devise a Theory for the Following

- Decide whether a particular relation  $R$  is in “good” form.
- In the case that a relation  $R$  is not in “good” form, decompose it into a set of relations  $\{R_1, R_2, \dots, R_n\}$  such that:
  - Each relation is in good form
  - The decomposition is a lossless-join decomposition
- Our theory is based on:
  - Functional dependencies
  - Multivalued dependencies

# Functional Dependencies

- Constraints on the set of legal relations.
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.
- A functional dependency is a generalization of the notion of a *key*.
- Let  $R$  be a relation schema     $\alpha \subseteq R$  and  $\beta \subseteq R$
- The functional dependency  $\alpha \rightarrow \beta$  holds on  $R$  if and only if for any legal relations  $r(R)$ , whenever any two tuples  $t_1$  and  $t_2$  of  $r$  agree on the attributes  $\alpha$ , they also agree on the attributes  $\beta$ . That is,  
$$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$
- E.g.: Consider  $r(A,B)$  with the following instance of  $r$ .
- On this instance,  $A \rightarrow B$  does NOT hold, but  $B \rightarrow A$  does hold.

|   |   |
|---|---|
| 1 | 4 |
| 1 | 5 |
| 3 | 7 |

**List all functional dependencies holds for the following relations**

| <b>A</b>  | <b>B</b>  | <b>C</b>  | <b>D</b>  |
|-----------|-----------|-----------|-----------|
| <b>a1</b> | <b>b1</b> | <b>c1</b> | <b>d1</b> |
| <b>a1</b> | <b>b2</b> | <b>c1</b> | <b>d2</b> |
| <b>a2</b> | <b>b2</b> | <b>c2</b> | <b>d2</b> |
| <b>a2</b> | <b>b3</b> | <b>c2</b> | <b>d3</b> |
| <b>a3</b> | <b>b3</b> | <b>c2</b> | <b>d4</b> |

| <b>A</b>  | <b>B</b>  | <b>C</b>  |
|-----------|-----------|-----------|
| <b>a1</b> | <b>b1</b> | <b>c1</b> |
| <b>a1</b> | <b>b1</b> | <b>c2</b> |
| <b>a2</b> | <b>b1</b> | <b>c1</b> |
| <b>a2</b> | <b>b1</b> | <b>c3</b> |



| A  | B  | C  | D  |
|----|----|----|----|
| a1 | b1 | c1 | d1 |
| a1 | b2 | c1 | d2 |
| a2 | b2 | c2 | d2 |
| a2 | b3 | c2 | d3 |
| a3 | b3 | c2 | d4 |

In the above relation r the following functional dependency holds:

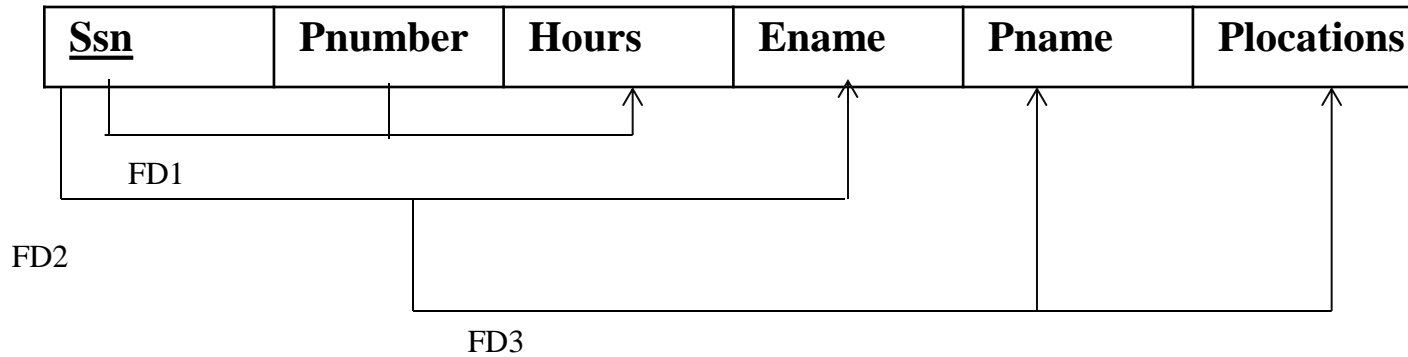
$A \rightarrow C$

| A  | B  | C  |
|----|----|----|
| a1 | b1 | c1 |
| a1 | b1 | c2 |
| a2 | b1 | c1 |
| a2 | b1 | c3 |

Give all functional dependencies satisfied by the above relation r.

$A \rightarrow B$  and  $C \rightarrow B$

## Emp\_Proj



- FD1: {Ssn, Pnumber}  $\rightarrow$  Hours
- FD2: Ssn  $\rightarrow$  Ename
- FD3: Pnumber  $\rightarrow$  {Pname, Plocations}

# Functional Dependencies ...Contd.

- $K$  is a superkey for relation schema  $R$  if and only if  $K \rightarrow R$
- $K$  is a candidate key for  $R$  if and only if  
 $K \rightarrow R$ , and for no  $\alpha \subset K$ ,  $\alpha \rightarrow R$
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys.
- Consider the schema:  $bor\_loan = (\underline{customer\_id}, \underline{loan\_number}, amount)$ .  
**We expect this functional dependency to hold:**  
 $loan\_number \rightarrow amount$   
**but would not expect the following to hold:**  
 $amount \rightarrow customer\_name$

# Use of Functional Dependencies

- We use functional dependencies to:
  - Test relations to see if they are legal under a given set of functional dependencies.
    - If a relation  $r$  is legal under a set  $F$  of functional dependencies, we say that  $r$  satisfies  $F$ .
  - Specify constraints on the set of legal relations
    - We say that  $F$  holds on  $R$  if all legal relations on  $R$  satisfy the set of functional dependencies  $F$ .
- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.
- E.g.: A specific instance of *loan* may, by chance, satisfy
$$amount \rightarrow customer\_name.$$
- A functional dependency is trivial if it is satisfied by all instances of a relation
$$\text{General Form: } \alpha \rightarrow \beta \text{ is trivial if } \beta \subseteq \alpha$$
  - $\{customer\_name, loan\_number\} \rightarrow customer\_name$
  - $customer\_name \rightarrow customer\_name$

# Closure of a Set of Functional Dependencies

- Given a set  $F$  of functional dependencies, there are certain other functional dependencies that are logically implied by  $F$ .
  - E.g.: If  $A \rightarrow B$  and  $B \rightarrow C$ , then we can infer that  $A \rightarrow C$
- The set of **all** functional dependencies logically implied by  $F$  is the *closure* of  $F$ .
- We denote the *closure* of  $F$  by  $F^+$ .
- $F^+$  is a superset of  $F$ .

## Procedure for Computing $F^+$

- To compute the closure of a set of functional dependencies  $F$ :

$$F^+ = F$$

**repeat**

**for each** functional dependency  $f$  in  $F^+$

    apply reflexivity and augmentation rules on  $f$

    add the resulting functional dependencies to  $F^+$

**for each** pair of functional dependencies  $f_1$  and  $f_2$  in  $F^+$

**if**  $f_1$  and  $f_2$  can be combined using transitivity

**then** add the resulting functional dependency to  $F^+$

**until**  $F^+$  does not change any further

# Closure of a Set of Functional Dependencies

- Given a set  $F$  set of functional dependencies, there are certain other functional dependencies that are logically implied by  $F$ .
  - For example: If  $A \rightarrow B$  and  $B \rightarrow C$ , then we can infer that  $A \rightarrow C$
- The set of **all** functional dependencies logically implied by  $F$  is the *closure* of  $F$ .
- We denote the *closure* of  $F$  by  $F^+$ .
- We can find all of  $F^+$  by applying Armstrong's Axioms:
  - IR1: If  $\beta \subseteq \alpha$ , then  $\alpha \rightarrow \beta$  (**reflexivity**)
  - IR2: If  $\alpha \rightarrow \beta$ , then  $\gamma \alpha \rightarrow \gamma \beta$  (**augmentation**)
  - IR3: If  $\alpha \rightarrow \beta$ , and  $\beta \rightarrow \gamma$ , then  $\alpha \rightarrow \gamma$  (**transitivity**)
  - IR4: If  $\{x \rightarrow yz\}$  then  $x \rightarrow y$  (**Decomposition / Projective**)
  - IR5: If  $x \rightarrow y$ ,  $x \rightarrow z$  then  $x \rightarrow yz$  (**Union/Addition**)
  - IR6: If  $x \rightarrow y$ ,  $wy \rightarrow z$  then  $wx \rightarrow z$  (**Pseudotransitive**)

→ These rules are

  - **sound** (generate only functional dependencies that actually hold) and
  - **complete** (generate all functional dependencies that hold).

- **$R = (\underline{Ssn}, Pnumber, Hours, Ename, Pname, Plocation)$**

**$F = \{ Ssn \rightarrow Ename,$   
 **$Pnumber \rightarrow \{Pname, Plocation\},$**   
 **$\{Ssn, Pnumber\} \rightarrow Hours \}$****

**Compute Members of  $F^+$**



■  **$R = (\underline{Ssn}, Pnumber, Hours, Ename, Pname, Plocation)$**

**$F = \{ Ssn \rightarrow Ename,$   
 **$Pnumber \rightarrow \{Pname, Plocation\},$**   
 **$\{Ssn, Pnumber\} \rightarrow Hours\}$****

**Compute Members of  $F^+$**

**Compute Members of  $F^+$**

**$Ssn^+ \rightarrow \{Ssn, Ename\}$**   
 **$\{Pnumber\}^+ = \{pnumber, Pname, Plocation\}$**   
 **$\{ssn, Pnumber\}^+ = \{Ssn, Pnumber, Ename, Pname, Plocation, Hours\}$**

▪ **Given  $R = (A, B, C, G, H, I)$**

**$F = \{ A \rightarrow B$**

**$A \rightarrow C$**

**$CG \rightarrow H$**

**$CG \rightarrow I$**

**$B \rightarrow H\}$**

**Compute  $F^+$ .**

- $R = (A, B, C, G, H, I)$   
 $F = \{$ 
  - $A \rightarrow B$
  - $A \rightarrow C$
  - $CG \rightarrow H$
  - $CG \rightarrow I$
  - $B \rightarrow H\}$
- Members of  $F^+$ 
  - $A \rightarrow H$ 
    - by transitivity from  $A \rightarrow B$  and  $B \rightarrow H$
  - $AG \rightarrow I$ 
    - by augmenting  $A \rightarrow C$  with  $G$ , to get  $AG \rightarrow CG$   
and then transitivity with  $CG \rightarrow I$
  - $CG \rightarrow HI$ 
    - by augmenting  $CG \rightarrow I$  to infer  $CG \rightarrow CGI$ ,  
and augmenting of  $CG \rightarrow H$  to infer  $CGI \rightarrow HI$ ,  
and then transitivity

- Compute the closure of the following set F of functional dependencies for relation schema R= (A, B,C, D, E)

$A \rightarrow BC$

$CD \rightarrow E$

**$B \rightarrow D$**

**$E \rightarrow A$**

■ **First Normal Form**

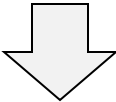
DEPARTMENT

| Dname  | <u>Dnumber</u> | Dmgr_ssn | Dlocations |
|--|----------------|----------|------------|
| <div style="display: flex; justify-content: space-around; align-items: center;"><div style="border-left: 1px solid black; border-right: 1px solid black; width: 30%; height: 40px; margin: 0 auto;"></div><div style="border-left: 1px solid black; border-right: 1px solid black; width: 30%; height: 40px; margin: 0 auto;"></div><div style="border-left: 1px solid black; border-right: 1px solid black; width: 30%; height: 40px; margin: 0 auto;"></div></div> |                |          |            |

- Remove the attribute Dlocations that violates 1NF and place it in a separate relation DEPT\_LOCATIONS along with the primary key Dnumber of DEPARTMENT. The primary key of this relation is the combination of {Dnumber, Dlocations}

| <u>Dnumber</u> | <u>Dlocations</u> |
|----------------|-------------------|
|----------------|-------------------|

- Expand the key so that there will be a separate tuple in the original relation DEPARTMENT relation for each location of a DEPARTMENT.



| <u>Dname</u> | <u>Dnumber</u> | Dmgr_ssn | Dlocations   |
|--------------|----------------|----------|--------------|
| Research     | 10             | 35       | {Lab1, Lab2} |
| Admin        | 1              | 25       | First floor  |

| <u>Dname</u> | <u>Dnumber</u> | Dmgr_ssn | Dlocations  |
|--------------|----------------|----------|-------------|
| Research     | 10             | 35       | Lab1        |
| Research     | 10             | 35       | Lab2        |
| Admin        | 1              | 25       | First floor |

- If a maximum number of values known for the attribute then replace the attribute by atomic attributes.

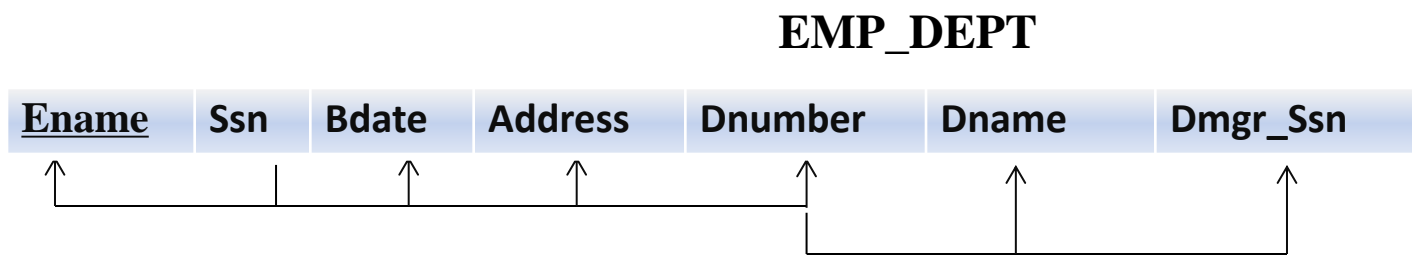
| <u>Dname</u> | <u>Dnumber</u> | Dmgr_ssn | Dlocation1  | Dlocation2 |
|--------------|----------------|----------|-------------|------------|
| Research     | 10             | 35       | Lab1        | lab2       |
| Admin        | 1              | 25       | First floor | Null       |

■ **Insertion Anomalies:**

- To insert a new employee tuple into EMP\_PROJ, we must include either the attribute values for the department that the employee works for or Null.
- It is difficult to insert a new department that has no employees as in the EMP\_DEPT relation. Only option is place Null values for Employee which is not possible .

■ **Deletion Anomalies:** If we delete from an employee that happens to represent last the last employee working for a particular department, the information concerning that department is lost from the database.

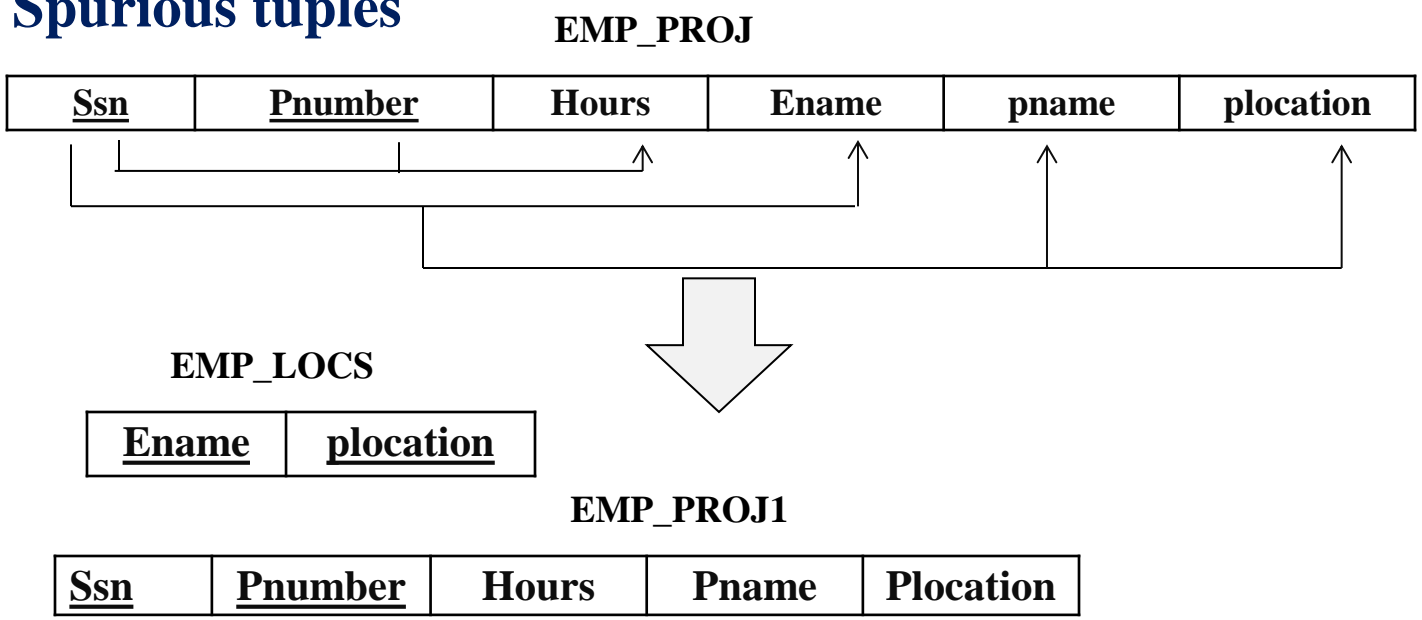
■ **Update Anomalies:** In EMP\_DEPT if we change the value of one of the attributes of a particular department we must update the tuples of all employee who work in that department.



- **Guideline2: Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations. If any anomalies are present note them clearly and make sure that the programs that update the database will operate correctly.**
  
- **Generally real world problems which includes many of the attributes do not apply to all tuples in the relation, we end up with many NULLs in those tuples. This can waste space at storage level and may also lead to problems with understanding the meaning of the attributes.**
  
- **Issues with Null Values in the tuple**
  - **Applying for Aggregate functions**
  - **Select & join operations requires comparison**
  - **Null have multiple interpretations**
    - **The attributes does not apply to this tuple.**
    - **The attribute value for this tuple is unknown**
    - **The value is known but absent that time it was not recorded .**

- **Guideline 3:** As far as possible, avoid placing attributes in a base relation whose values may frequently the NULL. If NULLs are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation.

- **Generation of Spurious tuples**



- Natural Join of EMP\_LOCS and EMP\_PROJ1 will generates spurious tuples



# Canonical Cover

- Sets of functional dependencies may have redundant dependencies that can be inferred from the others
  - E.g.:  $A \rightarrow C$  is redundant in:  $\{A \rightarrow B, B \rightarrow C\}$
  - Parts of a functional dependency may be redundant
    - E.g.: on RHS:  $\{A \rightarrow B, B \rightarrow C, A \rightarrow CD\}$  can be simplified to  $\{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$
    - E.g.: on LHS:  $\{A \rightarrow B, B \rightarrow C, AC \rightarrow D\}$  can be simplified to  $\{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$
- Intuitively, a canonical cover of  $F$  is a “minimal” set of functional dependencies equivalent to  $F$ , having no redundant dependencies or redundant parts of dependencies

## Extraneous Attributes

- An attribute of a functional dependency is extraneous if we remove it without changing the closure of the set of functional dependencies.
- Consider a set  $F$  of functional dependencies and the functional dependency  $\alpha \rightarrow \beta$  in  $F$ .
  - Attribute  $A$  is **extraneous** in  $\alpha$  if  $A \in \alpha$  and  $F$  logically implies  $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$ .
  - Attribute  $A$  is **extraneous** in  $\beta$  if  $A \in \beta$  and the set of functional dependencies  $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$  logically implies  $F$ .
- E.g.: Given  $F = \{A \rightarrow C, AB \rightarrow C\}$ 
  - $B$  is extraneous in  $AB \rightarrow C$  because  $\{A \rightarrow C, AB \rightarrow C\}$  logically implies  $A \rightarrow C$  (I.e. the result of dropping  $B$  from  $AB \rightarrow C$ ).
- Example: Given  $F = \{A \rightarrow C, AB \rightarrow CD\}$ 
  - $C$  is extraneous in  $AB \rightarrow CD$  since  $AB \rightarrow C$  can be inferred even after deleting  $C$

## Testing if an Attribute is Extraneous

- Consider a set  $F$  of functional dependencies and the functional dependency  $\alpha \rightarrow \beta$  in  $F$ .
- To test if attribute  $A \in \alpha$  is extraneous in  $\alpha$ 
  1. compute  $(\{\alpha\} - A)^+$  using the dependencies in  $F$
  2. check that  $(\{\alpha\} - A)^+$  contains  $\beta$ ; if it does,  $A$  is extraneous in  $\alpha$
- To test if attribute  $A \in \beta$  is extraneous in  $\beta$ 
  1. compute  $\alpha^+$  using only the dependencies in  $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ ,
  2. check that  $\alpha^+$  contains  $A$ ; if it does,  $A$  is extraneous in  $\beta$

## Canonical cover

- A canonical cover  $F_c$  for  $F$  is a set of functional dependencies such that  $F$  logically implies all dependencies in  $F_c$  and  $F_c$  logically implies all dependencies in  $F$ . furthermore,  $F_c$  must have the following properties:
  - No functional dependency in  $F_c$  contains extraneous attribute.
  - Each left side of a functional dependency in  $F_c$  is unique. That is, there are no two dependencies  $\alpha_1 \rightarrow \beta_1$   $\alpha_2 \rightarrow \beta_2$  in  $F_c$  such that  $\alpha_1 = \alpha_2$

# Canonical Cover

- A **canonical cover** for  $F$  is a set of dependencies  $F_c$  such that
  - $F$  logically implies all dependencies in  $F_c$ , and
  - $F_c$  logically implies all dependencies in  $F$ , and
  - No functional dependency in  $F_c$  contains an extraneous attribute, and
  - Each left side of functional dependency in  $F_c$  is unique.

- **To compute a canonical cover for  $F$ :**

**repeat**

Use the union rule to replace any dependencies in  $F$

$$\alpha_1 \rightarrow \beta_1 \text{ and } \alpha_1 \rightarrow \beta_2 \text{ with } \alpha_1 \rightarrow \beta_1 \beta_2$$

Find a functional dependency  $\alpha \rightarrow \beta$  with an

extraneous attribute either in  $\alpha$  or in  $\beta$

If an extraneous attribute is found, delete it from  $\alpha \rightarrow \beta$

**until**  $F$  does not change

## Extraneous Attributes

- Consider the following set F of functional dependency on schema (A,B,C) Compute the canonical cover for F.

$$A \rightarrow BC$$

$$B \rightarrow C$$

$$A \rightarrow B$$

$$AB \rightarrow C$$

$$\text{Answer: } A \rightarrow B$$

$$B \rightarrow C$$

- $F = \{AB \rightarrow CD, A \rightarrow E, \& E \rightarrow C\}$  Find canonical cover of F.

## Extraneous Attributes

- Consider the following set F of functional dependency on schema (A,B,C) Compute the canonical cover for F.

$$A \rightarrow BC$$

$$B \rightarrow C$$

$$A \rightarrow B$$

$$AB \rightarrow C$$

**Answer:  $A \rightarrow B$**

**$B \rightarrow C$**

$$\left. \begin{array}{l} A \rightarrow BC \\ A \rightarrow B \end{array} \right\} A \rightarrow BC$$

A is extraneous  $AB \rightarrow C$  from  $F - \{AB \rightarrow C\} \cup \{B \rightarrow C\}$  implies  $B \rightarrow C$

C is extraneous  $A \rightarrow BC$  implies  $A \rightarrow B$  &  $A \rightarrow C$  Implies  $A \rightarrow B$

- $F = \{A \rightarrow BC, B \rightarrow AC, \& C \rightarrow AB\}$  Find canonical cover of F.

# Computation of Super key from FD's

**Given:** Drinkers(name, addr, beersLiked, manf, favBeer)

**Reasonable FD's to assert:**

1. **name**  $\rightarrow$  **addr**
2. **name**  $\rightarrow$  **favBeer**
3. **beersLiked**  $\rightarrow$  **manf**

| name    | addr       | beersLiked | manf   | favBeer   |
|---------|------------|------------|--------|-----------|
| Janeway | Voyager    | Bud        | A.B.   | WickedAle |
| Janeway | Voyager    | WickedAle  | Pete's | WickedAle |
| Spock   | Enterprise | Bud        | A.B.   | Bud       |

Because **name**  $\rightarrow$  **addr**

Because **beersLiked**  $\rightarrow$  **manf**

Because **name**  $\rightarrow$  **favBeer**



**Compute the closure of the following set F of functional dependencies for relation schema  $R = \{A, B, C, D, E\}$ .**

**$A \rightarrow BC$**

**$CD \rightarrow E$**       **List the candidate keys for R.**

**$B \rightarrow D$**

**$E \rightarrow A$**

**Compute the closure of the following set F of functional dependencies for relation schema  $R = \{A, B, C, D, E\}$ .**

**$A \rightarrow BC$   
 $CD \rightarrow E$     **List the candidate keys for R.**  
 $B \rightarrow D$   
 $E \rightarrow A$**

**Given:  $A \rightarrow BC, B \rightarrow D$  so  $A \rightarrow D$  so  $A \rightarrow DC \rightarrow E$  therefore  $A \rightarrow ABCDE$   
 $E \rightarrow A, A \rightarrow ABCDE$ , so  $E \rightarrow ABCDE$   $CD \rightarrow E$ , so  $CD \rightarrow ABCDE$   $B \rightarrow D, BC \rightarrow CD$ , so  $BC \rightarrow ABCDE$**

**Attribute closure:**

**$A \rightarrow ABCDE$**

**$B \rightarrow BD$**

**$C \rightarrow C$**

**$D \rightarrow D$**

**$E \rightarrow ABCDE$**

**$AB \rightarrow ABCDE$**

**$AC \rightarrow ABCDE$**

**$BD \rightarrow BDAD \rightarrow ABCDE$**

**$AE \rightarrow ABCDE$**

**$BC \rightarrow ABCDE$**

**$BE \rightarrow ABCDE$**

**$CD \rightarrow ABCDE$**

**$CE \rightarrow ABCDE$**

**$DE \rightarrow ABCDE$**

**$ABC \rightarrow ABCDE$**

**$ABD \rightarrow ABCDE$**

**$ABE \rightarrow ABCDE$**

**$ACD \rightarrow ABCDE$**

**$ACE \rightarrow ABCDE$**

**$ADE \rightarrow ABCDE$**

**$BCD \rightarrow ABCDE$**

**$BDE \rightarrow ABCDE$**

**$CDE \rightarrow ABCDE$**

**$ABCD \rightarrow ABCDE$**

**$ABCE \rightarrow ABCDE$**

**$ABDE \rightarrow ABCDE$**

**$ACDE \rightarrow ABCDE$**

**$BCDE \rightarrow ABCDE$**

**The candidate keys are A, E, CD, and BC**

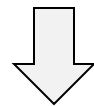
# Lossless-join Decomposition

- For the case of  $R = (R_1, R_2)$ , we require that for all possible relations  $r$  on schema  $R$

$$r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$$

- A decomposition of  $R$  into  $R_1$  and  $R_2$  is lossless join if and only if at least one of the following dependencies is in  $F^+$ :
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$
- If  $R_1 \cap R_2$  forms a superkey of either  $R_1$  or  $R_2$ , the decomposition of  $R$  is lossless decomposition.

**bor\_loan**  $\rightarrow$  (customer\_id, loan\_number, amount)



decomposed into

**borrower**  $\rightarrow$  (customer\_id, loan\_number)

**loan**  $\rightarrow$  (loan\_number, amount)

Here **borrower**  $\cap$  **loan** = **loan\_number**  
thus it is lossless decomposition

Check the following:

- $R = (A, B, C)$   
 $F = \{A \rightarrow B, B \rightarrow C\}$ 
  - Can be decomposed in two different ways
    - $R_1 = (A, B), R_2 = (B, C)$ 
      - Lossless-join decomposition:  
 $R_1 \cap R_2 = \{B\}$  and  $B \rightarrow BC$
      - Dependency preserving
    - $R_1 = (A, B), R_2 = (A, C)$ 
      - Lossless-join decomposition:  
 $R_1 \cap R_2 = \{A\}$  and  $A \rightarrow AB$
      - Not dependency preserving

Let  $F_i$  be the set of dependencies  $F^+$  that include only attributes in  $R_i$ .

- A decomposition is dependency preserving, if
$$(F_1 \cup F_2 \cup \dots \cup F_n)^+ = F^+$$
- If it is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive.

# BCNF Decomposition

- $R = (A, B, C)$   
 $F = \{A \rightarrow B$   
 $B \rightarrow C\}$   
 $\text{Key} = \{A\}$ 
  - $R$  is not in BCNF ( $B \rightarrow C$  but  $B$  is not superkey)
  - Decomposition
    - $R_1 = (B, C)$
    - $R_2 = (A, B)$

## Comparison of BCNF and 3NF

- It is always possible to decompose a relation into a set of relations that are in 3NF such that:
  - the decomposition is lossless
  - the dependencies are preserved
- It is always possible to decompose a relation into a set of relations that are in BCNF such that:
  - the decomposition is lossless
  - it may not be possible to preserve dependencies.

# Comparison of BCNF and 3NF

- It is always possible to decompose a relation into a set of relations that are in 3NF such that:
  - Decomposition is lossless
  - Dependencies are preserved
- It is always possible to decompose a relation into a set of relations that are in BCNF such that:
  - the decomposition is lossless
  - it may not be possible to preserve dependencies.

# Multivalued Dependencies (MVDs)

- Let  $R$  be a relation schema and let  $\alpha \subseteq R$  and  $\beta \subseteq R$ . The multivalued dependency

$$\alpha \twoheadrightarrow \beta$$

holds on  $R$  if in any legal relation  $r(R)$ , for all pairs for tuples  $t_1$  and  $t_2$  in  $r$  such that  $t_1[\alpha] = t_2[\alpha]$ , there exist tuples  $t_3$  and  $t_4$  in  $r$  such that:

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$

$$t_3[\beta] = t_1[\beta]$$

$$t_3[R - \beta] = t_2[R - \beta]$$

$$t_4[\beta] = t_2[\beta]$$

$$t_4[R - \beta] = t_1[R - \beta]$$

E.g.:

|       | $\alpha$        | $\beta$             | $R - \alpha - \beta$ |
|-------|-----------------|---------------------|----------------------|
| $t_1$ | $a_1 \dots a_i$ | $a_{i+1} \dots a_j$ | $a_{j+1} \dots a_n$  |
| $t_2$ | $a_1 \dots a_i$ | $b_{i+1} \dots b_j$ | $b_{j+1} \dots b_n$  |
| $t_3$ | $a_1 \dots a_i$ | $a_{i+1} \dots a_j$ | $b_{j+1} \dots b_n$  |
| $t_4$ | $a_1 \dots a_i$ | $b_{i+1} \dots b_j$ | $a_{j+1} \dots a_n$  |

# Use of Multivalued Dependencies

- We use multivalued dependencies in two ways:
  1. To test relations to determine whether they are legal under a given set of functional and multivalued dependencies
  2. To specify constraints on the set of legal relations. We shall thus concern ourselves only with relations that satisfy a given set of functional and multivalued dependencies
- If a relation  $r$  fails to satisfy a given multivalued dependency, we can construct a relations  $r'$  that does satisfy the multivalued dependency by adding tuples to  $r$ .
- **Fourth Normal Form**
  - A relation schema  $R$  is in 4NF with respect to a set  $D$  of functional and multivalued dependencies if for all multivalued dependencies in  $D^+$  of the form  $\alpha \twoheadrightarrow \beta$ , where  $\alpha \subseteq R$  and  $\beta \subseteq R$ , at least one of the following hold:
    - $\alpha \twoheadrightarrow \beta$  is trivial (i.e.,  $\beta \subseteq \alpha$  or  $\alpha \cup \beta = R$ )
    - $\alpha$  is a superkey for schema  $R$
  - If a relation is in 4NF it is in BCNF



## ■ Non Additive Join Decomposition into 4 NF

Whenever we decompose a relation schema into  $R_1 = (X \cup Y)$  and  $R_2 = (R - Y)$  based on an  $MVD: X \twoheadrightarrow Y$  that holds in R, the decomposition has the non additive join property.

$$R_1 \cap R_2 \twoheadrightarrow (R_1 - R_2)$$

OR

$$R_1 \cap R_2 \twoheadrightarrow (R_2 - R_1)$$

**Algorithm:** Input: A universal relation R and a set of functional & multivalued dependencies F.

1. Set  $D := \{R\}$
2. While there is a relational schema Q in D that is not in 4NF, do  
{ choose a relation schema Q in D that is not in 4NF;  
  find a nontrivial  $MVD: X \twoheadrightarrow Y$  in Q that violates 4NF;  
  replace Q in D by two relation schemas  $(Q - Y)$  and  $(X \cup Y)$ ;  
};

▪ **Decompose the relation schema R in to 4NF with Nonadditive join property**

**R =(A, B, C, G, H, I)**

**F ={ A  $\rightarrow\rightarrow$  B**

**B  $\rightarrow\rightarrow$  HI**

**CG  $\rightarrow\rightarrow$  H }**

- $R = (A, B, C, G, H, I)$   
 $F = \{ A \twoheadrightarrow B$   
 $\quad B \twoheadrightarrow HI$   
 $\quad CG \twoheadrightarrow H \}$
- $R$  is not in 4NF since  $A \twoheadrightarrow B$  and  $A$  is not a superkey for  $R$
- **Decomposition**
  - a)  $R_1 = (A, B)$   $(R_1 \text{ is in 4NF})$
  - b)  $R_2 = (A, C, G, H, I)$   $(R_2 \text{ is not in 4NF})$
  - c)  $R_3 = (C, G, H)$   $(R_3 \text{ is in 4NF})$
  - d)  $R_4 = (A, C, G, I)$   $(R_4 \text{ is not in 4NF})$
- Since  $A \twoheadrightarrow B$  and  $B \twoheadrightarrow HI$ ,  $A \twoheadrightarrow HI$ ,  $A \twoheadrightarrow I$ 
  - e)  $R_5 = (A, I)$   $(R_5 \text{ is in 4NF})$
  - f)  $R_6 = (A, C, G)$   $(R_6 \text{ is in 4NF})$