# COMP 5311

- Instructor: **Dimitris Papadias**
- WWW page: http://www.cse.ust.hk/~dimitris/5311/5311.html

- **Textbook**
  Database System Concepts, A. Silberschatz, H. Korth, and S. Sudarshan.
- **Reference**
  Database Management Systems, Raghu Ramakrishnan and Johannes Gehrke.

- **Grading Policy:** final, presentation/survey
- Exams will be with open books and notes.

# Course Outline

- E/R Model
- Relational Model, Algebra
- SQL
- Functional Dependencies and Relational Database Design
- File Systems
- Indexing
- Query Processing and Implementation of Relational Operators
- Query Optimization
- Transactions

# What is a Database Management System (DBMS)

- Collection of interrelated data + Set of programs to access the data
- DBMS contains information about a particular enterprise
- DBMS provides an environment that is both *convenient* and *efficient* to use.
- Database Applications:
  - Banking: all transactions
  - Airlines: reservations, schedules
  - Universities: registration, grades
  - Sales: customers, products, purchases
  - Manufacturing: production, inventory, orders, supply chain
  - Human resources: employee records, salaries, tax deductions
- Databases touch all aspects of our lives

# DBMS vs File Systems

- In the early days, database applications were built on top of file systems
- Drawbacks of using file systems to store data:
  - Data redundancy and inconsistency
    - Multiple file formats, duplication of information in different files
  - Difficulty in accessing data
    - Need to write a new program to carry out each new task
  - Integrity problems
    - Integrity constraints (e.g. account balance > 0) become part of program code
    - Hard to add new constraints or change existing ones

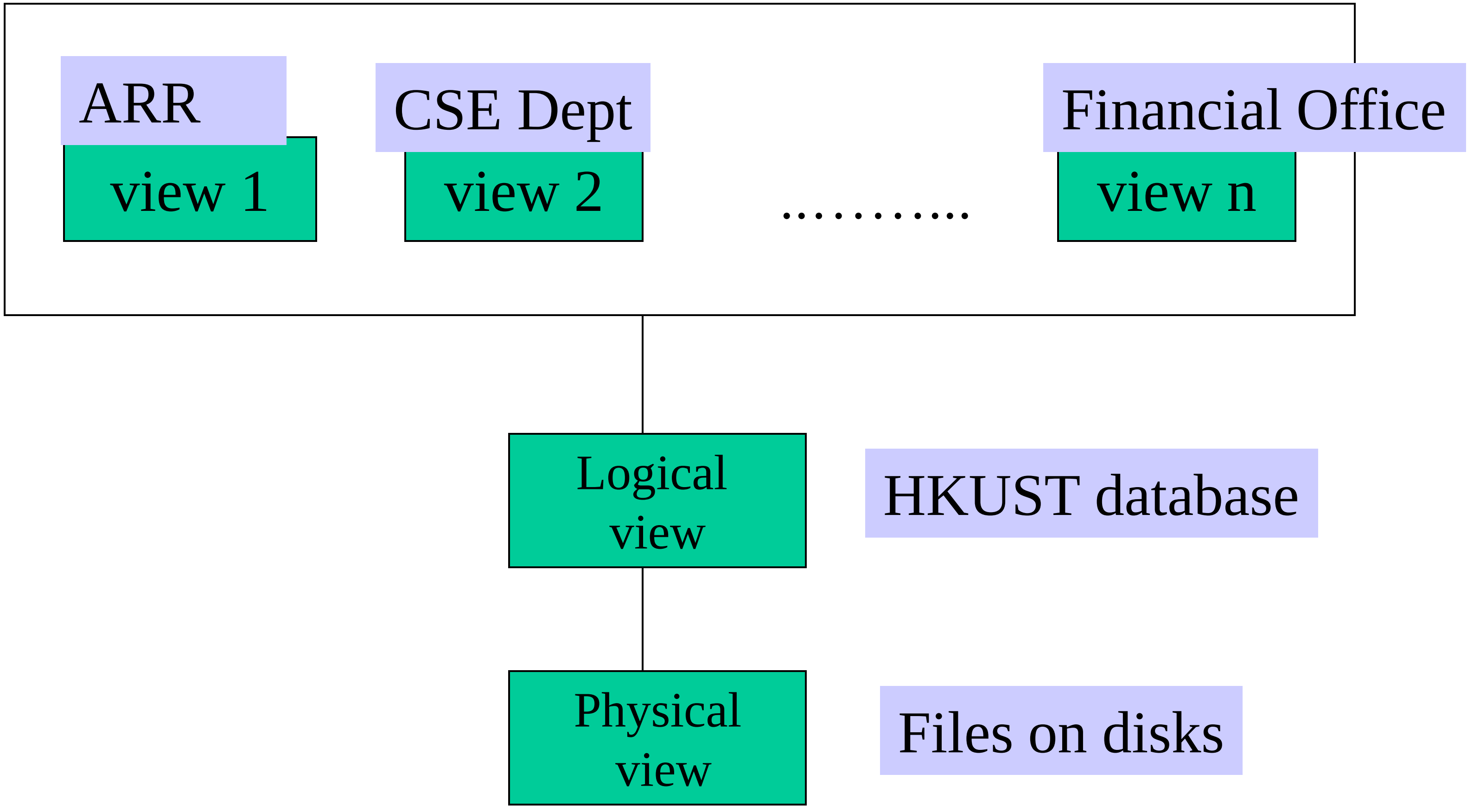
# DBMS vs File Systems (cont)

- Drawbacks of using file systems (cont.)
  - Atomicity of updates
    - Failures may leave database in an inconsistent state with partial updates carried out
    - E.g. transfer of funds from one account to another should either complete or not happen at all
  - Concurrent access by multiple users
    - Concurrent accesses needed for performance
    - Uncontrolled concurrent accesses can lead to inconsistencies
      - E.g. two people reading a balance and updating it at the same time
  - Security problems
- DBMS offer automated solutions to all the above problems; they solve problems caused by different people writing different applications independently.

# Data Independence

- One big problem in application development is the *separation* of applications from data
- Do I have change my program when I …
  - replace my hard drive?
  - store the data in a b-tree instead of a hash file?
  - partition the data into two physical files (or merge two physical files into one)?
  - store salary as floating point number instead of integer?
  - develop other applications that use the same set of data?
  - add more data fields to support other applications?
- Solution: introduce levels of *abstraction*.

# Three Levels of Abstraction



# Three Levels of Abstraction (cont.)

- Physical level: describe how a record is stored on disks.
  - e.g., "Divide the customer records into 3 partitions and store them on disks 1, 2 and 3."
- Logical level: describes data stored in database, and the relationships among the data. Similar to defining a record type in Pascal or C:

```
Type customer = record
          name: string;
          street: string;
          city: integer; end;
```

- View level: Define a subset of the database for a particular application. Views can also hide information (e.g. salary) for security purposes.

# Data Independence

- Ability to modify a schema definition in one level without affecting a schema definition in the next higher level.

- The interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.

- Two levels of data independence:

  - Physical data independence (users are shielded from changes in the physical structure of the data)

  - Logical data independence (users are shielded from changes in the logical structure of the data)

# Data Models

- A collection of tools for describing:
  - data
  - data relationships
  - data semantics
  - data constraints

# Basic Concepts of ER

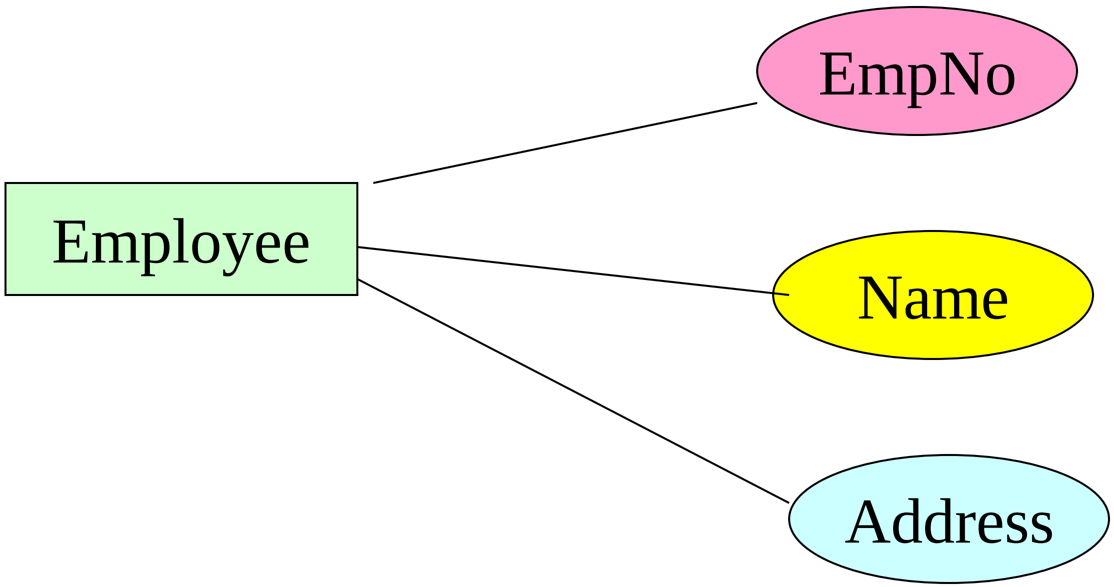
- ER is a model for the logical level
  - it describes the structure of the database at a high abstraction level
- A database can be modeled as
  - a collection of entities
  - relationship among entities
- An entity is an object that exists independently and is distinguishable from other objects.
  - an employee, a company, a car, a student, a class etc.
  - color, age, etc. are not entities

# Attributes

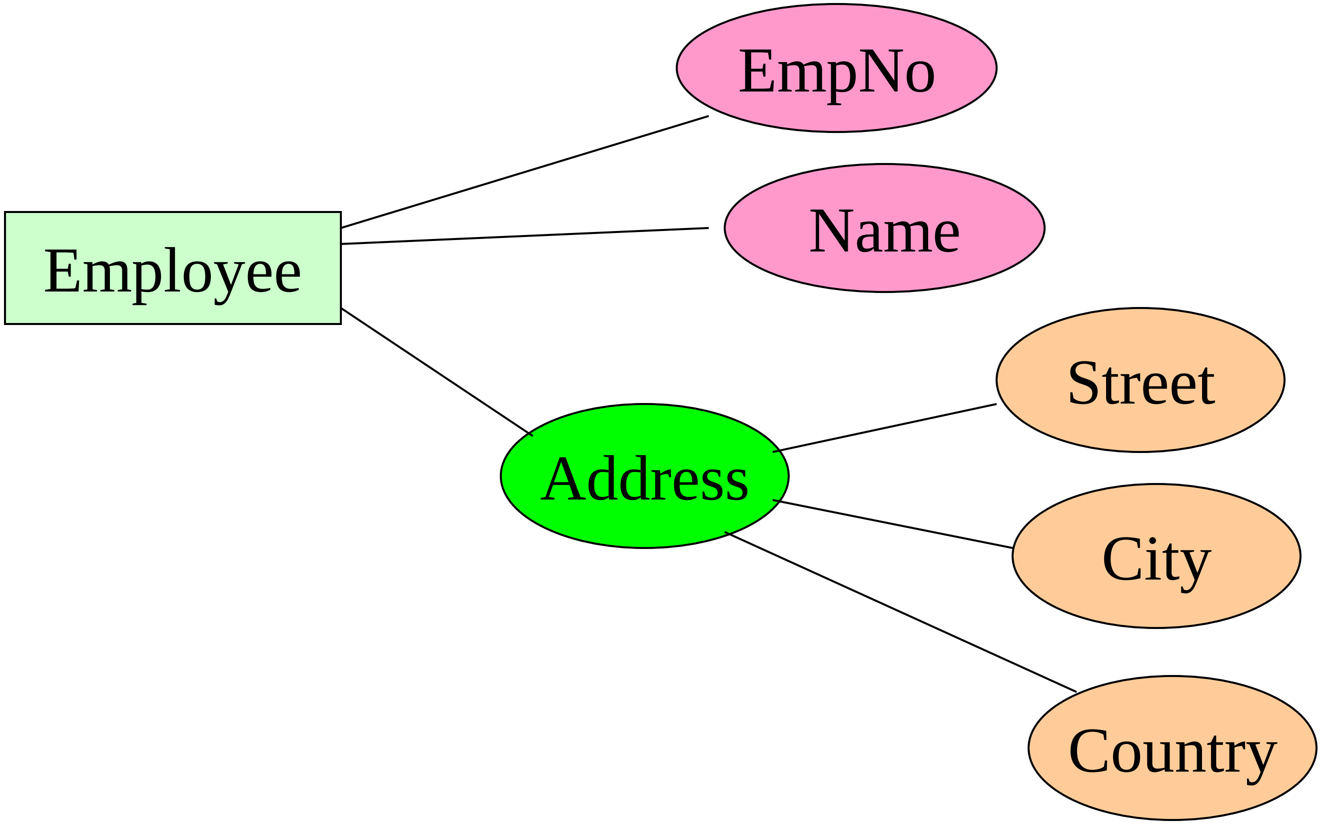
- Properties of an entity or a relationship
  - name, address, weight, height are properties of a Person entity.
- date of marriage is a property of the relationship Marriage.

# Types of Attributes

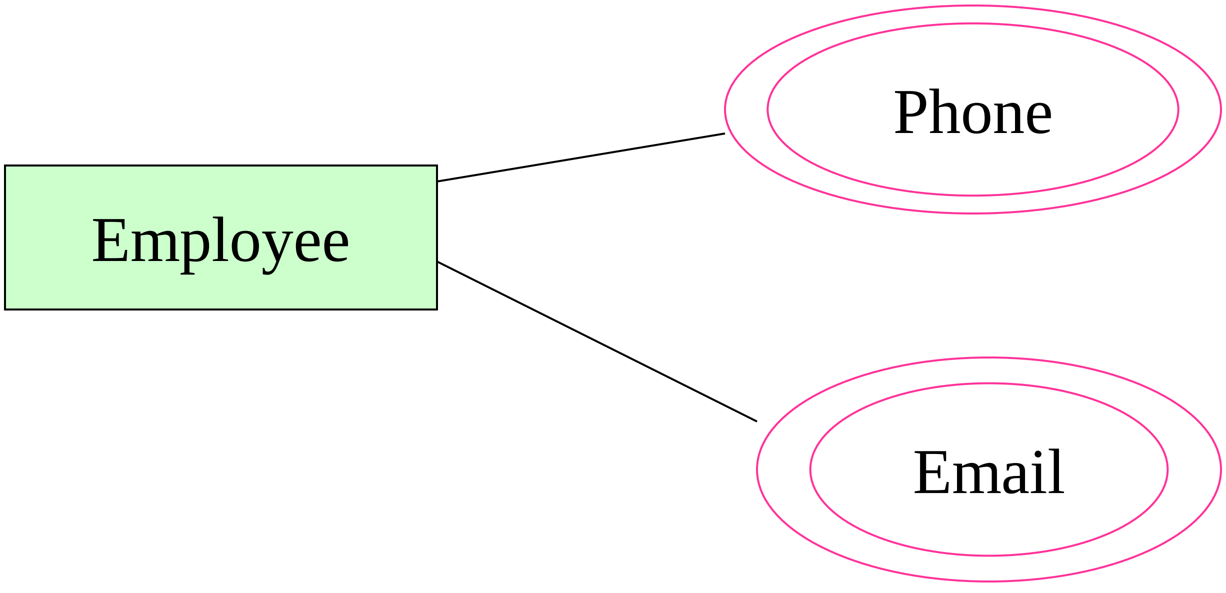
- Simple attribute: contains a single value.



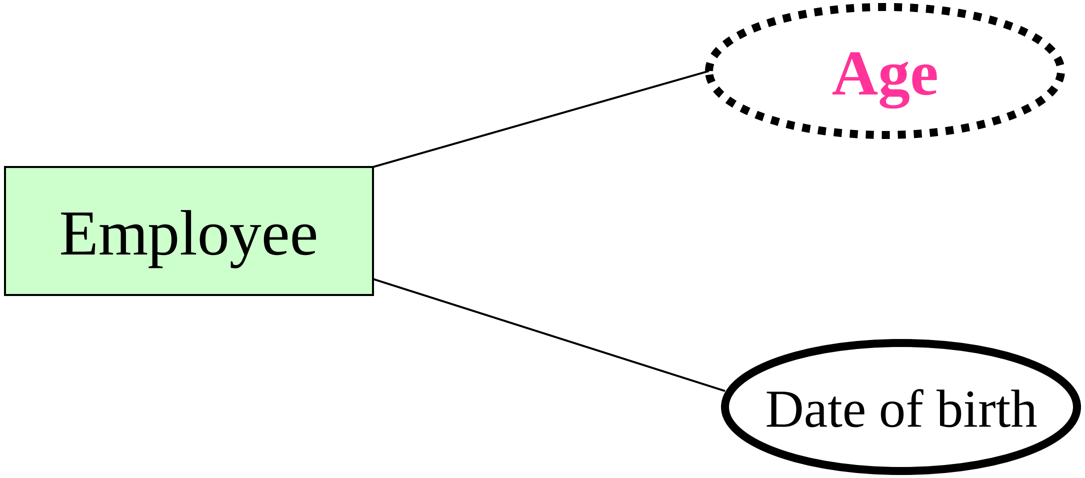
- Composite attribute: consists of several components (e.g., address)



- Multivalued attribute: contains more than one value

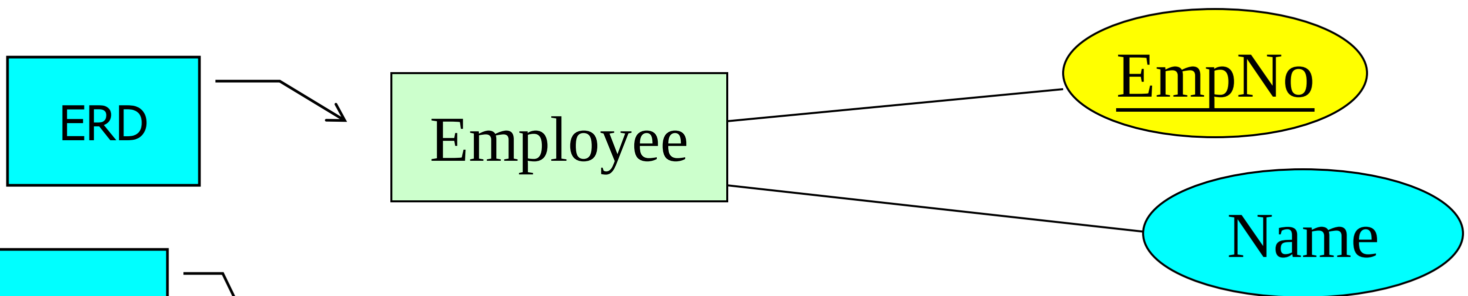


- Derived attribute: computed from other attributes (e.g., age can be computed from the date of birth and the current date)



# Key Attributes

- A set of attributes that can uniquely identify an entity



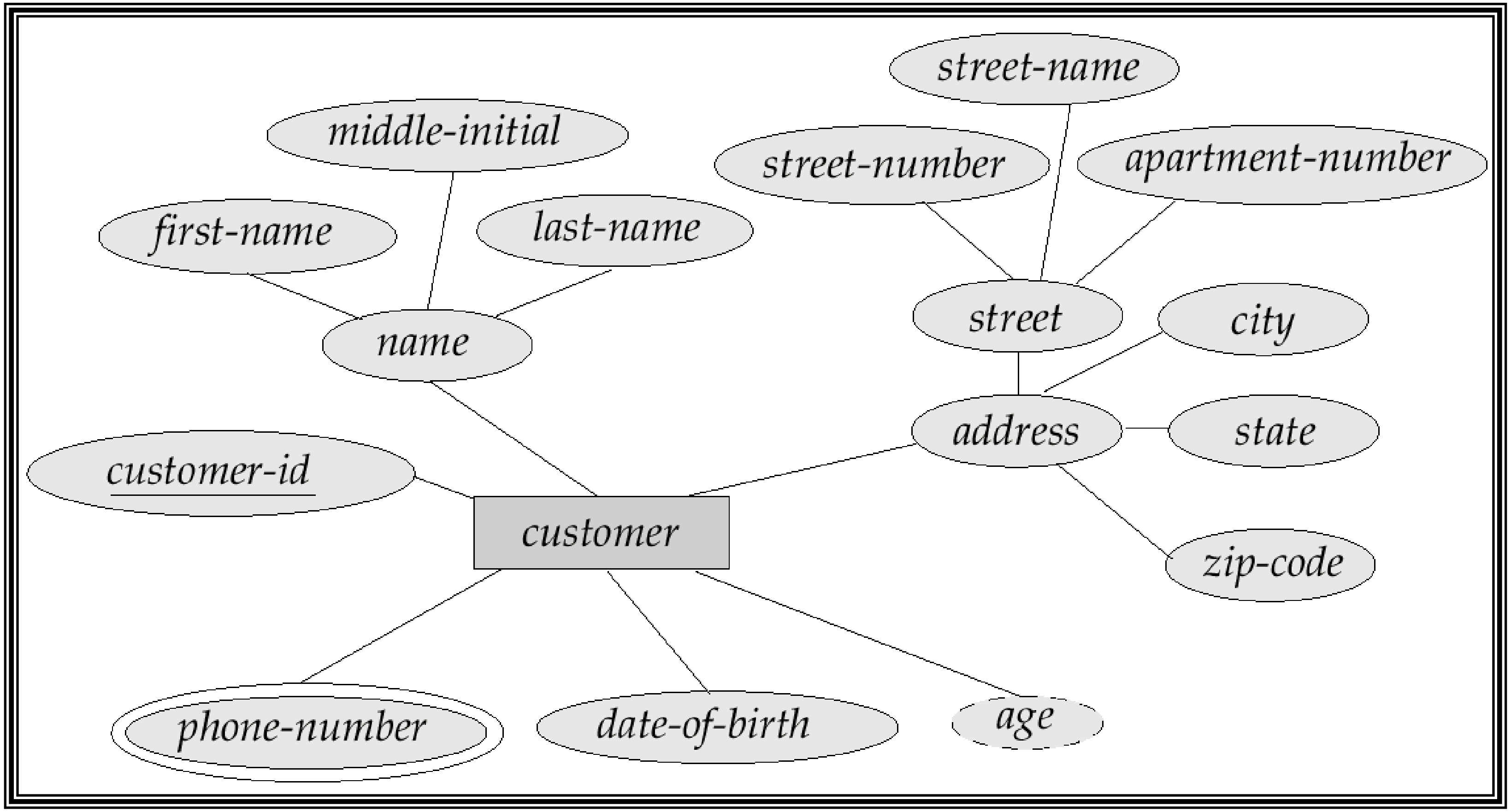

tabular

| EmpNo | Name | . . . |
|---|---|---|
| 123456 | John Wong | . . . |
| 456789 | Mary Cheung | . . . |
| 146777 | John Wong | . . . |

# Key Attributes

- An entity may have more than one key
  - A minimal set of attributes that uniquely identifies an entity is called a candidate key.
    - Question: which are possible candidate keys for HKUST students?
  - Only one candidate key is selected to be the primary key.
    - Question: which is the primary key for HKUST students?
  - Sometimes artificial keys maybe created
    - Example: assume that we want to store information about the current offering of COMP 5311. We can select a unique number (e.g., 1235) to serve as the key.
    - Question: which are possible alternatives for this example, without introducing additional attributes?
    - Composite Key: contains two or more attributes

# Example Entity (Customer)



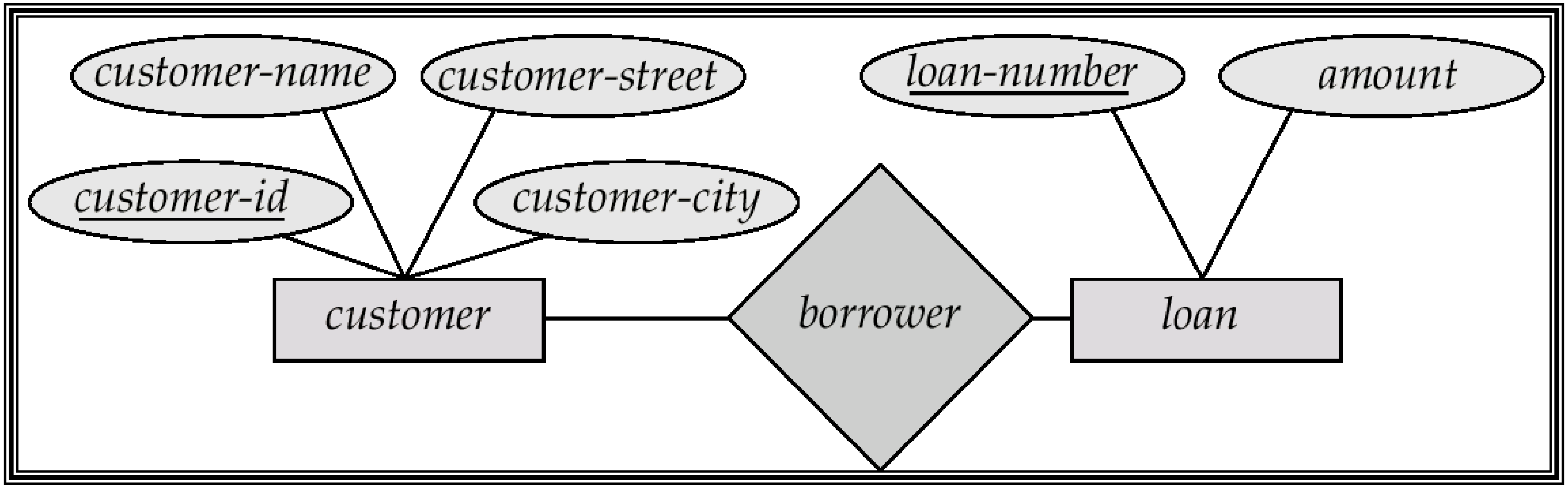
# Relationship

- A relationship is an association among several entities

- The degree refers to the number of entity sets that participate in a relationship set.
- Relationship sets that involve two entity sets are *binary* (or degree two).
- Relationships among more than two entity sets are rare.  (More on this later.)

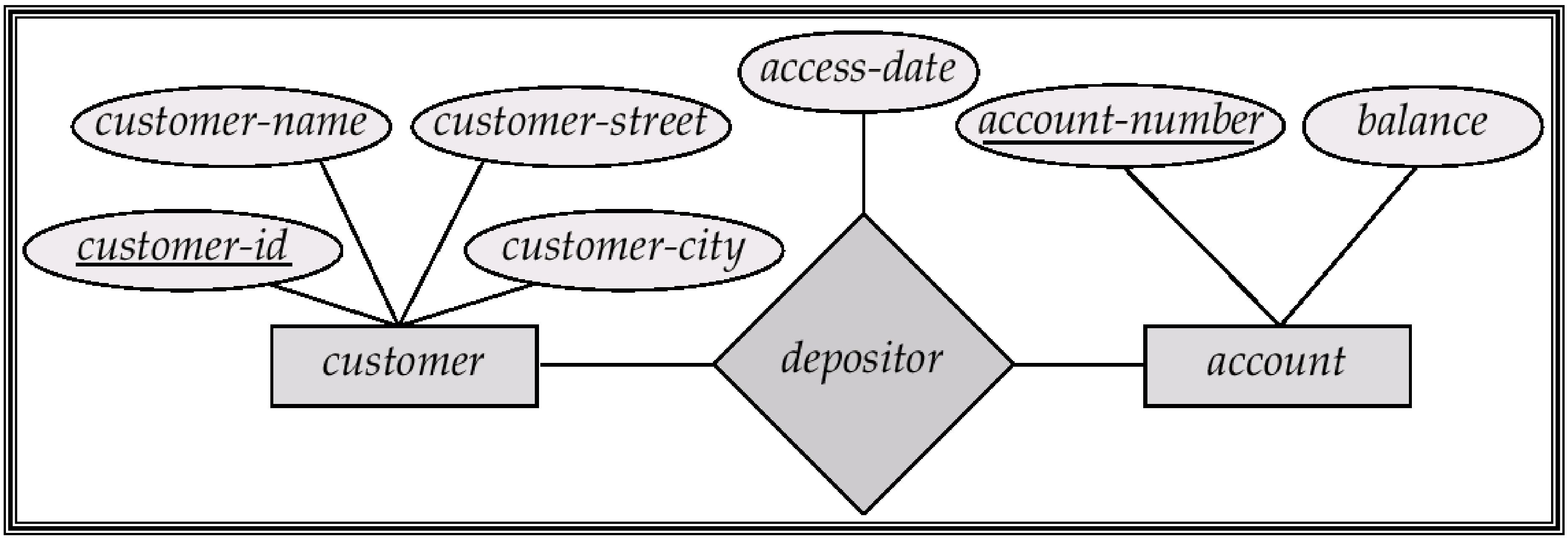
# Example of (Binary) Relationship

- Borrower is a relationship between Customers and Loans (it means that a customer can be associated with one or more loans and vice versa).



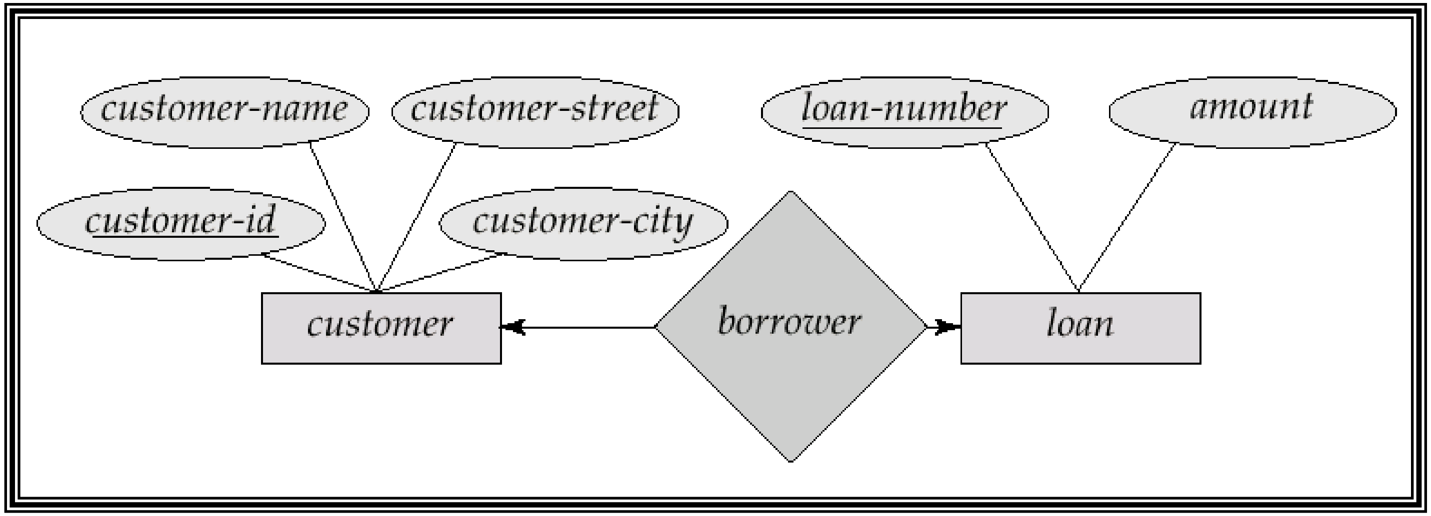
# Relationship Sets with Attributes

- Depositor is a relationship between Customers and Accounts
- Access-date is an attribute of Depositor



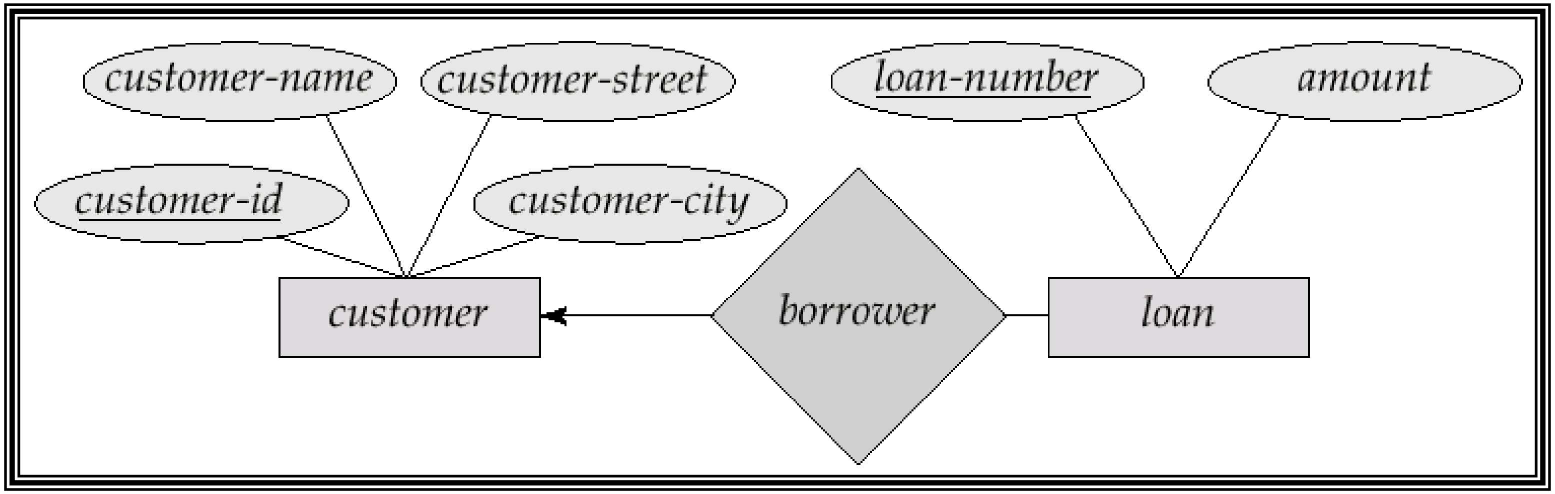
# Cardinality Constraints

- 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.
- E.g.: 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*



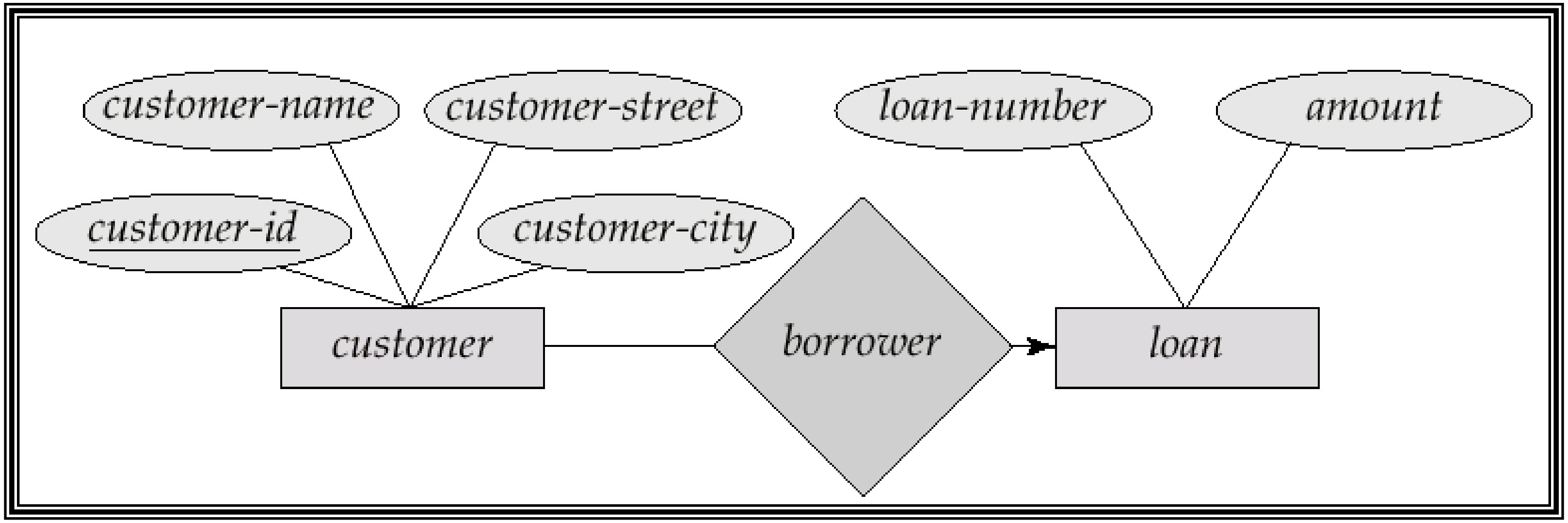
# One-To-Many Relationship

- 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*

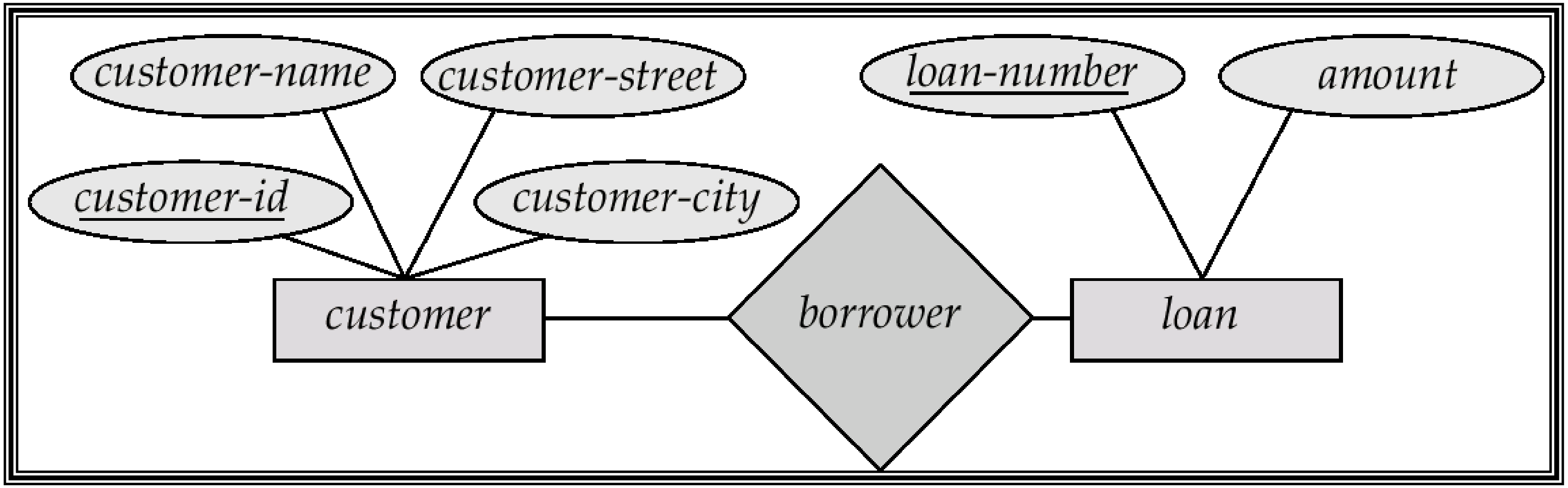


# Many-To-One Relationships

- 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



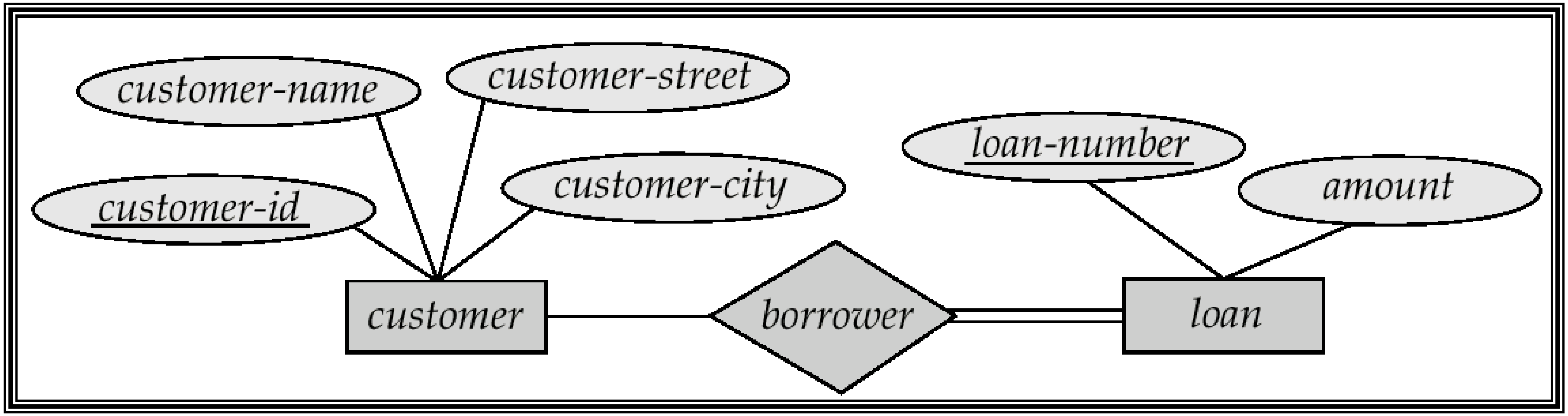

- A customer is associated with several (possibly 0) loans via borrower
- A loan is associated with several (possibly 0) customers via borrower

# 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

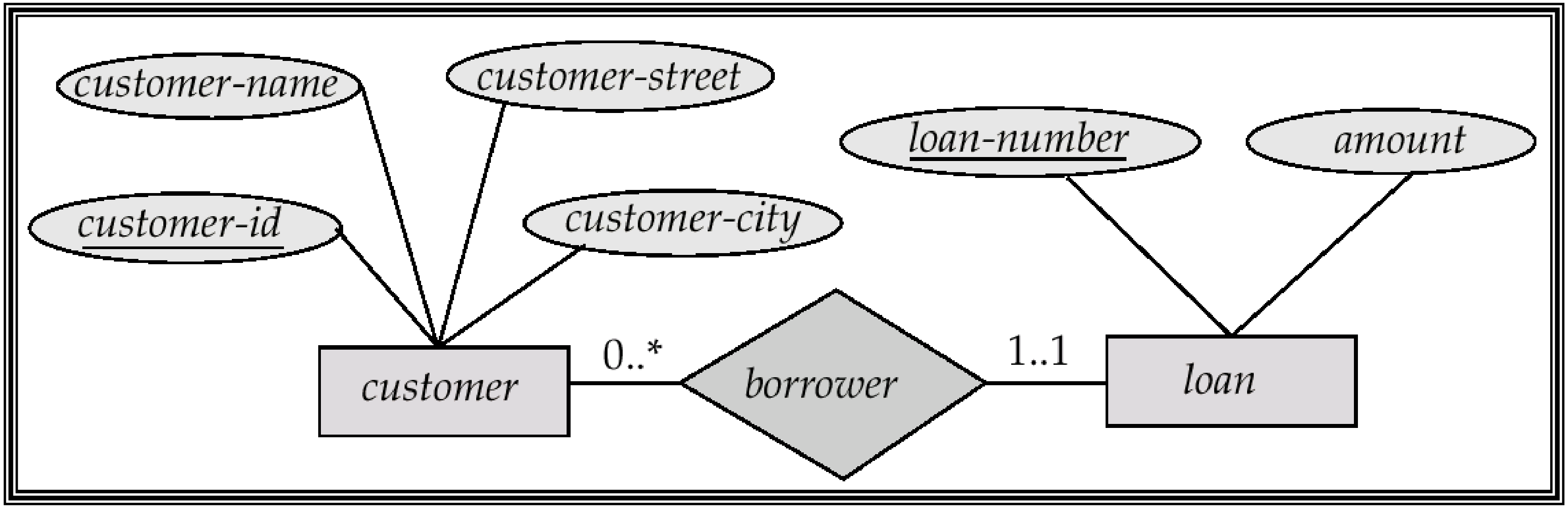
    every loan must have at least a customer associated to it via borrower
- Partial participation: some entities may not participate in any relationship in the relationship set
  - E.g. participation of *customer* in *borrower* is partial

    some customers may not have any loans



# Alternative Notation for Cardinality Limits

Cardinality limits can also express participation constraints

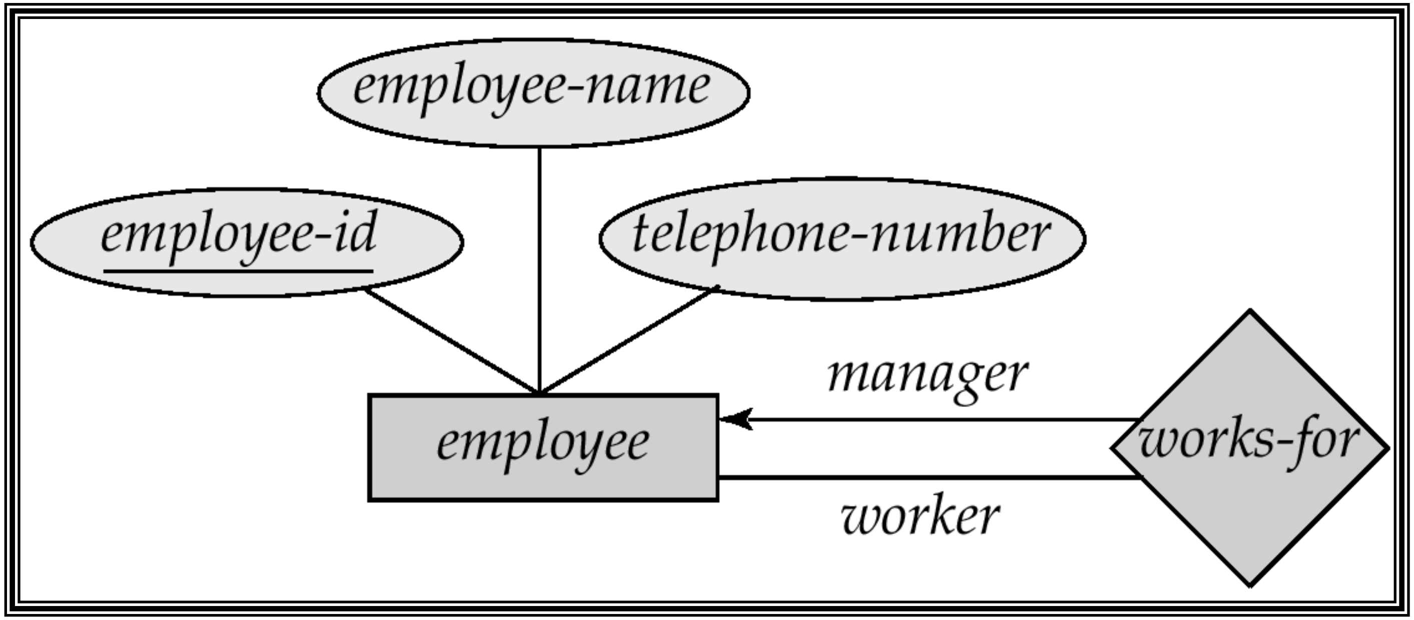


# Cardinality Limits

- The edge between *loan* and *borrower* has a cardinality constraint of 1..1,
  - meaning the minimum and the maximum cardinality are both 1.
  - each loan must have exactly one associated customer.
- The limit 0..* on the edge from *customer* to *borrower* indicates that a customer can have zero or more loans.
- Thus, the relationship *borrower* is
  - one to many from *customer* to *loan*,
  - the participation of *loan* in *borrower* is total.

# 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

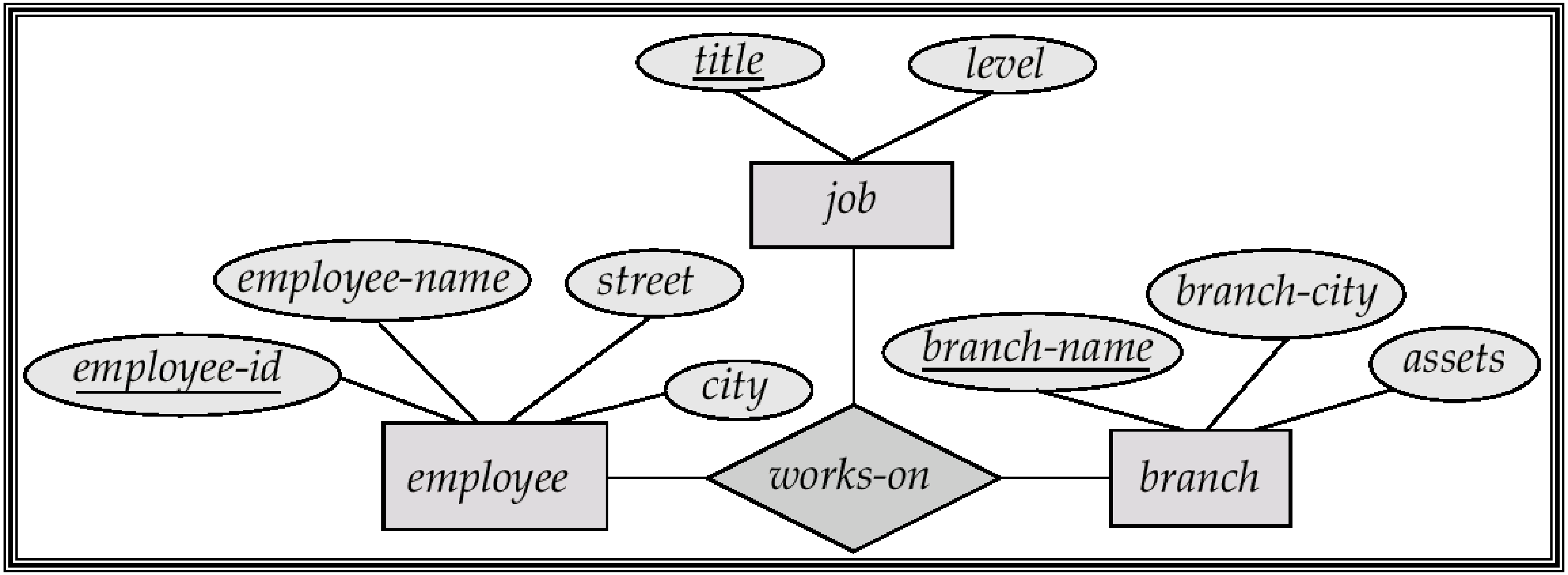


# Keys for Relationship Sets

- The combination of primary keys of the participating entity sets forms a super key of a relationship set.
  - (customer-id, account-number) is the super key of depositor
  - This means that a pair of entities can have at most one relationship in a particular relationship set.
    - E.g. if we wish to track all access-dates to each account by each customer, we cannot assume a relationship for each access. Solution: use a multivalued attribute for access dates.
- Must consider the mapping cardinality of the relationship set when deciding the candidate keys

# E-R Diagram with a Ternary Relationship

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*



# Binary Vs. Non-Binary Relationships

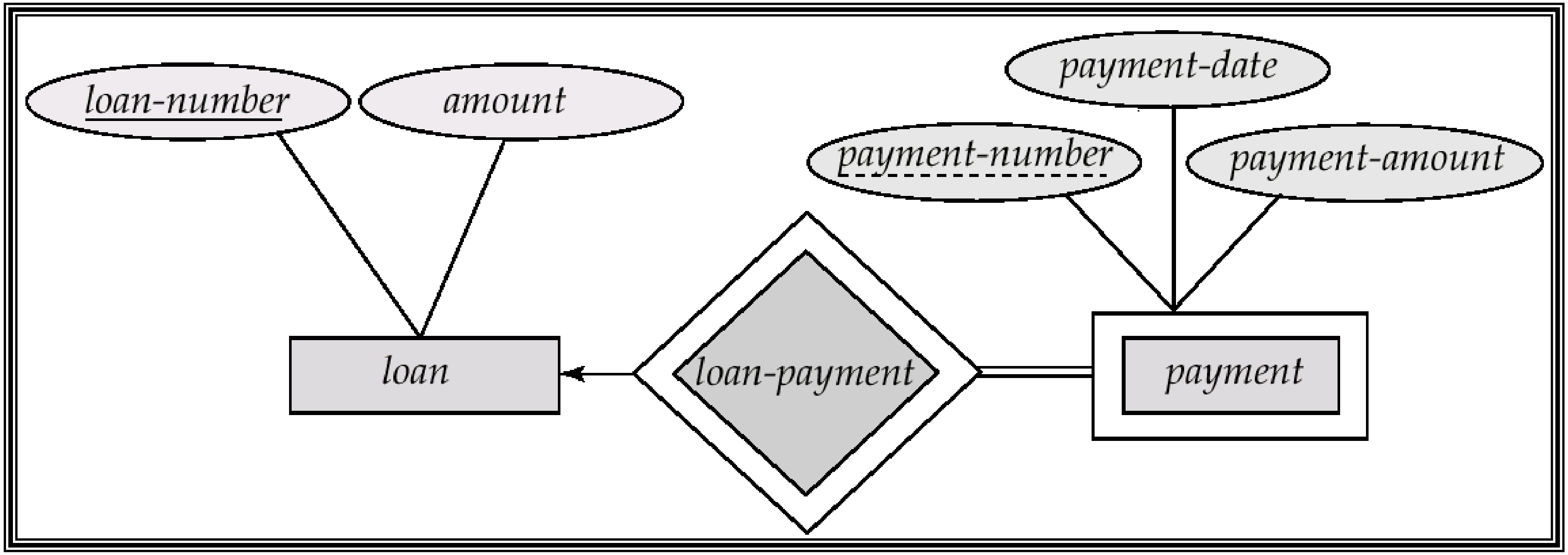
- Some relationships that appear to be non-binary may be better represented using binary relationships
  - E.g.  A ternary relationship parents, relating a child to his/her father and mother, is best replaced by two binary relationships, father and mother
    - Using two binary relationships allows partial information (e.g. only mother being known)
  - But there are some relationships that are naturally non-binary
    - E.g. *works-on*

# Weak Entity Sets

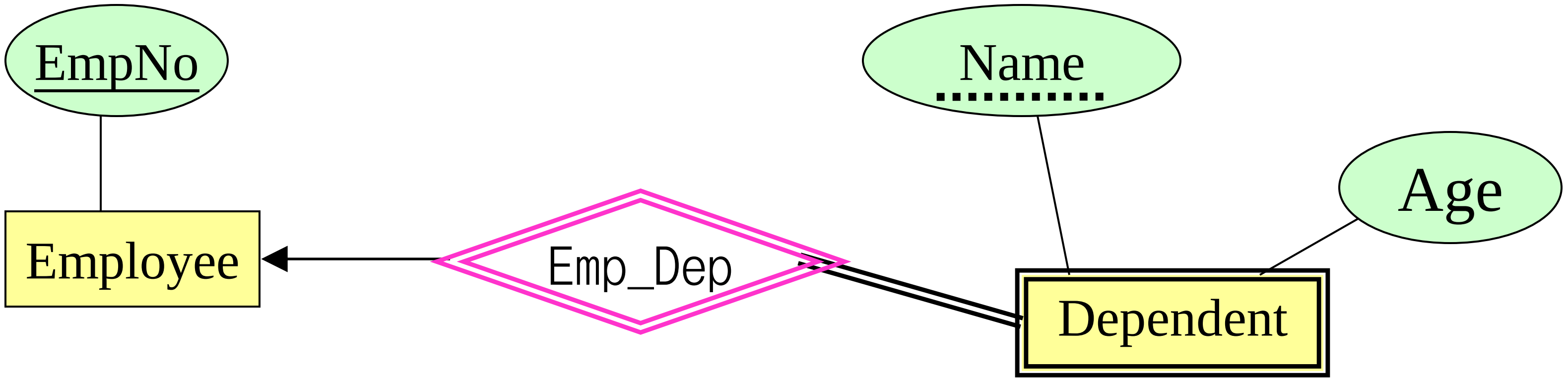
- An entity set that does not have a primary key is referred to as a *weak entity set*.
- The existence of a weak entity set depends on the existence of a *identifying entity set*
  - it must relate to the identifying entity set via a total, one-to-many relationship set from the identifying to the weak 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 (Cont.)

- 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*)

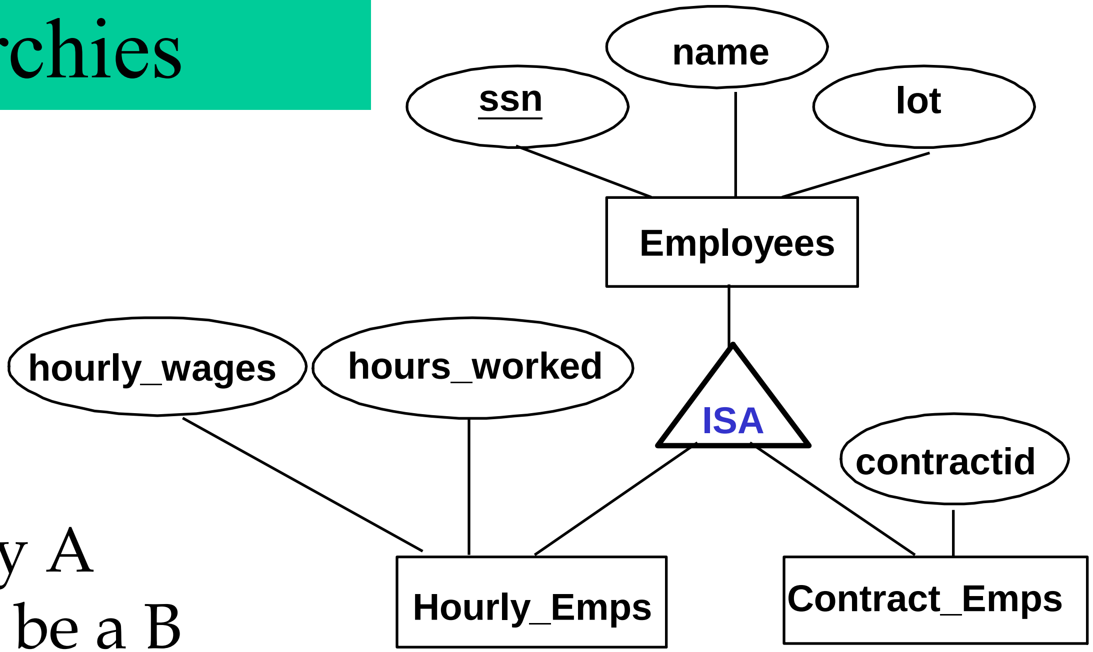


- **Another example of weak entity type**




- A child may not be old enough to have a HKID number
- Even if he/she has a HKID number, the company may not be interested in keeping it in the database.

# ISA (`is a') Hierarchies

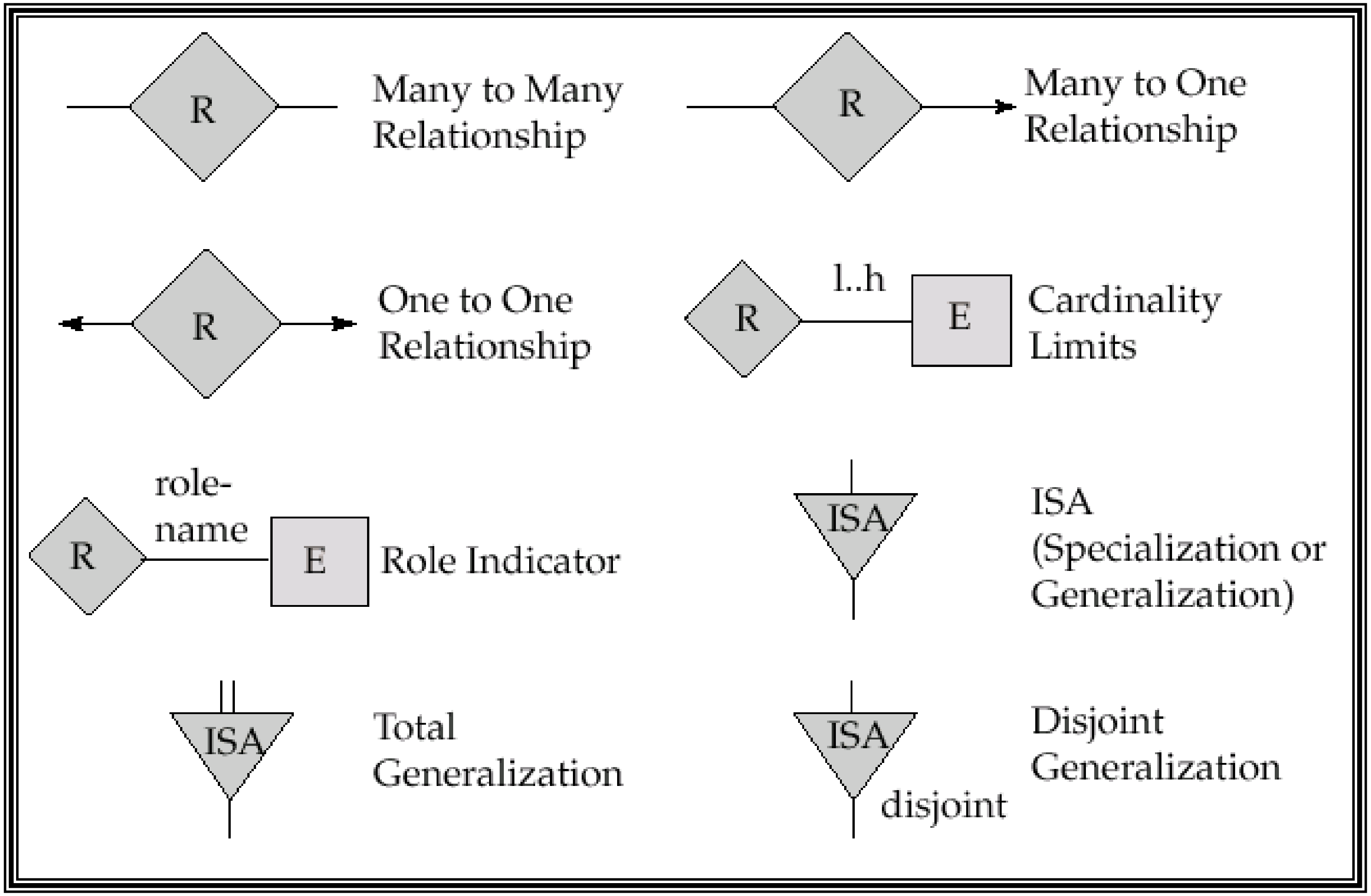



As in C++, or other PLs, attributes are inherited.

If we declare A **ISA** B, every A entity is also considered to be a B entity.

- Overlap constraints: Can Joe be an Hourly_Emps as well as a Contract_Emps entity? (*Allowed/disallowed*)
- Covering constraints: Does every Employees entity also have to be an Hourly_Emps or a Contract_Emps entity? *(Yes/no)*
- Reasons for using ISA:
  - To add descriptive attributes specific to a subclass.
  - To identify entities that participate in a relationship.

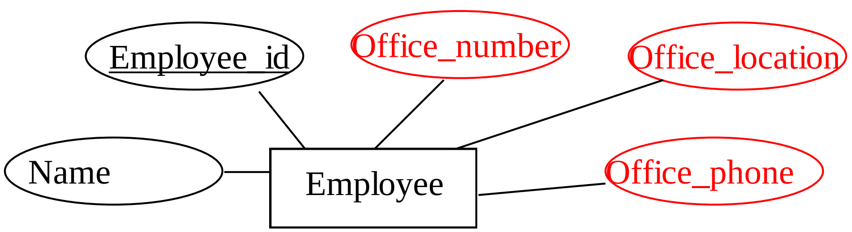
# Summary of Symbols (Cont.)



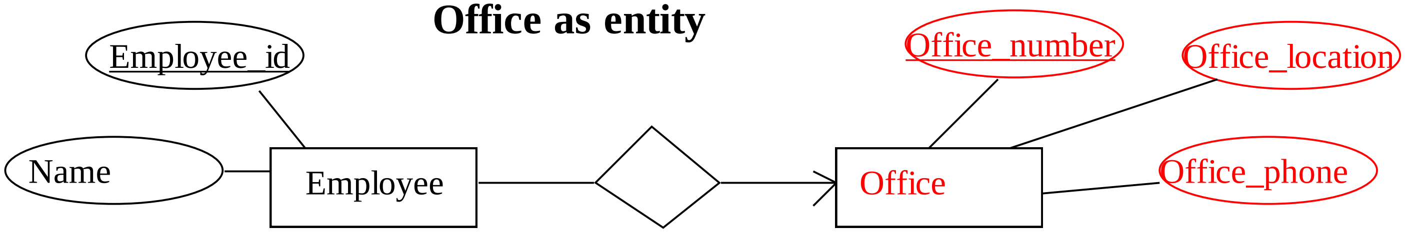
# ER Design Decisions - Attribute vs Entity

- For each employee we want to store the office number, location of the office (e.g., Building A, floor 6), and telephone.
- Several employees share the same office

**Office as attribute**

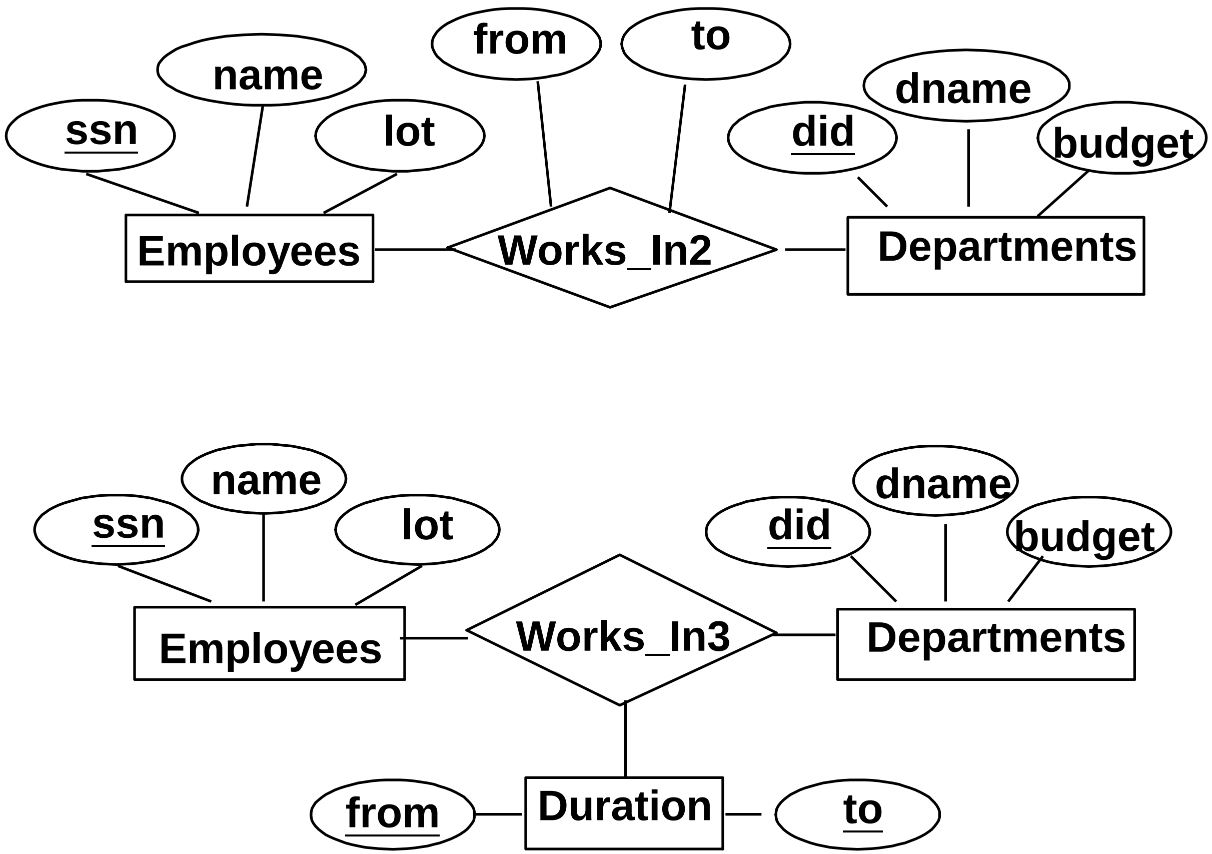



**Office as entity**



# ER Design Decisions - Entity vs Relationship

- You want to record the period that an employ works for some department.



# ER Design Decisions - Strong vs. Weak Entity

- Example: What if in the accounts example
  - an account must be associated with exactly one branch
  - two different branches are allowed to have accounts with the same number.