Topic 3

ER Model and Deriving Relational Schema

From

Chapters 3 and 4 of Fundamentals of Database Systems, Authors: Elmasri and Navathe

Publisher: Addison Wesley -Pearson

By: Abdolreza Abhari

CPS510

Ryerson University

Topics in this Section

- Design of the conceptual schema
- Entity-relationship (ER) model
  - Entities
  - Relationships
  - Attributes
- ER diagrams
- Deriving relational schema from ER model

Design of the Conceptual Schema

- *Stage one:* Choice of model: User requirements and real world concepts should go into the design model. If using E-R model, E-R diagram and relational schemas are the results of this stage.

Design of the Conceptual Schema

- *Stage one:* Choice of model: User requirements and real world concepts should go into the design model. If using E-R model, E-R diagram and relational schemas are the results of this stage.
- *Stage two:* Normalization: Conceptual schema are then used in normalization. This further leads to adjusted diagrams and a normalized relational model. (will be discussed later)

Design of the Conceptual Schema

- *Stage one:* Choice of model: User requirements and real world concepts should go into the design model. If using E-R model, E-R diagram and relational schemas are the results of this stage.
- *Stage two:* Normalization: Conceptual schema are then used in normalization. This further leads to adjusted diagrams and a normalized relational model. (will be discussed later)
- *Stage three:* Optimization: (will be discussed later). The outcome of this stage is the data dictionary and the database description.

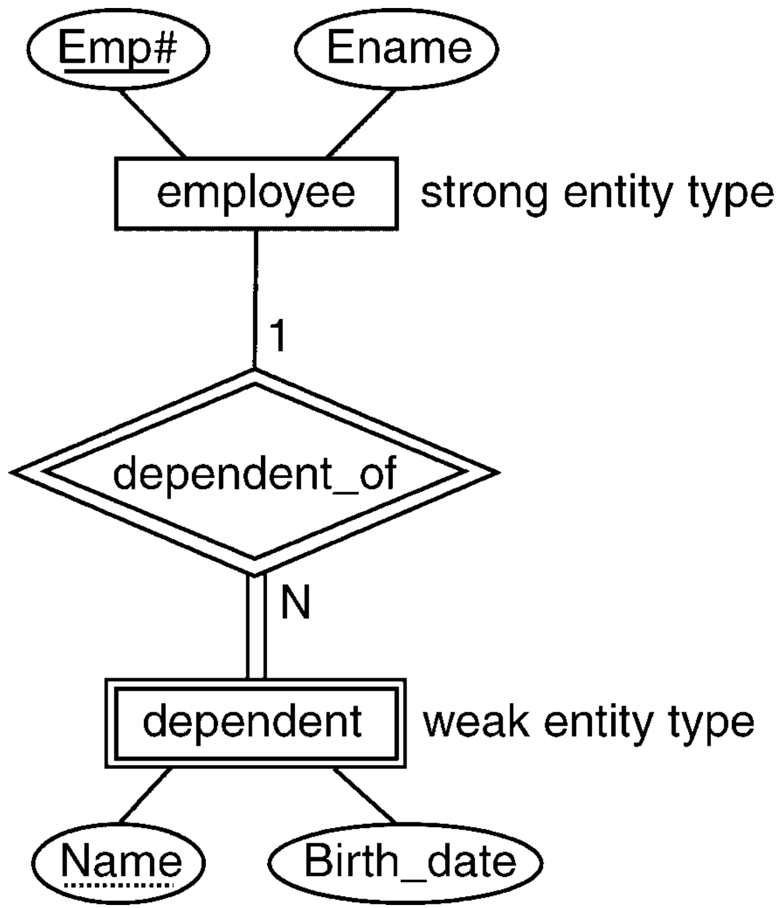
Entity-Relationship Model

- Entity-relationship (ER) model
  - ∗ Introduced by Peter Chen in 1976
- ER model consists of
  - » Entities
  - » Relationships
  - » Attributes
- No single, generally accepted notation
  - ∗ Note that
    - » You may find variants of the notation used here
    - » You may also find symbols different from the ones we use

Entities

- Entity is a "thing in the real world with an independent existence"
  - ∗ Two types of entities
    - – Strong/regular entity (simply called *entity*)
    - – Weak entity
  - ∗ Strong entity types are called *owner* or *dominant* entity types
    - – Exist on their own
  - ∗ Weak entity types are called *dependent* or *subordinate* entity types
    - – Existence of weak entity depends on the existence of another strong entity

Entities (cont'd)



Weak Entity Example

- Every dependent must be associated with an employee
- If an employee is deleted from the database, dependent must also be deleted
- We can delete a dependent without affecting employee
- Weak entity type is indicated by double outlined box

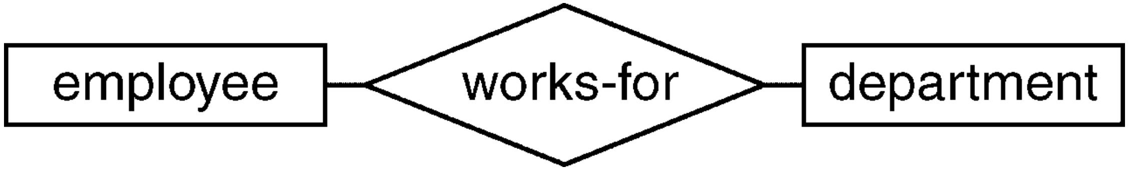
Relationships

A relationship associates entities with one another

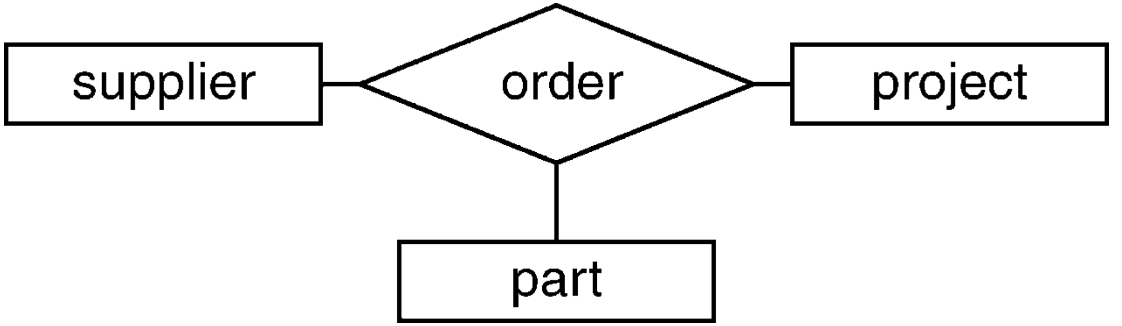
Degree of a relationship type is the number of participating entity types

Examples

- Binary relationship



- Ternary relationship



Relationships (cont'd)

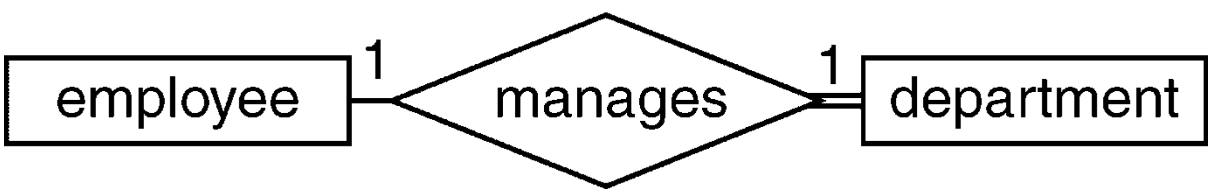
- Mapping constraints
  - * one-to-one (1:1)
  - * one-to-many (1:N)
  - * many-to-many (M:N)

Examples

one-to-one

» A manager can manage only one department

» Each department can have only one manager

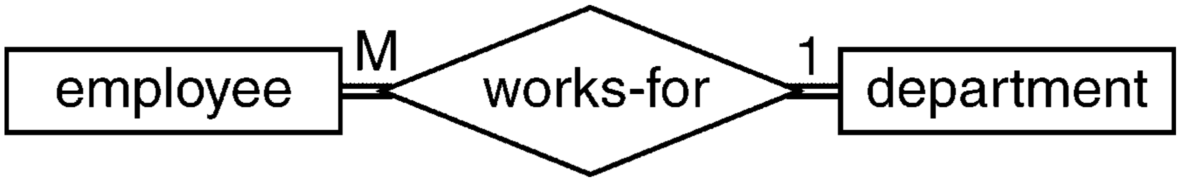


Relationships (cont'd)

one-to-many

» A department can have several employees

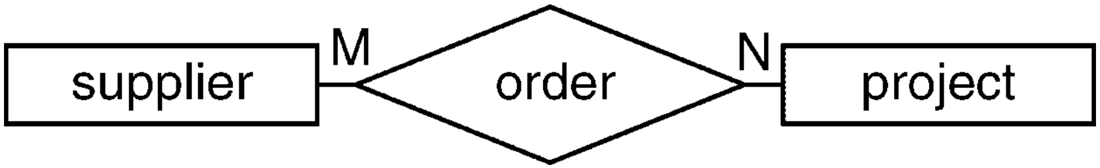
» Each employee may work in only one department



many-to-many

•A supplier can supply parts to several projects

•A project can receive parts from several suppliers

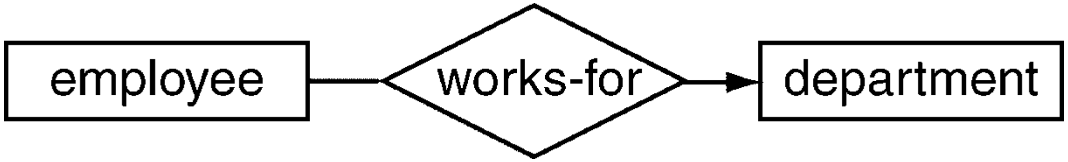


Relationships (cont'd)

- Alternative notation for mapping constraints
  - Uses directed line to represent one-to-many or one-to-one mapping

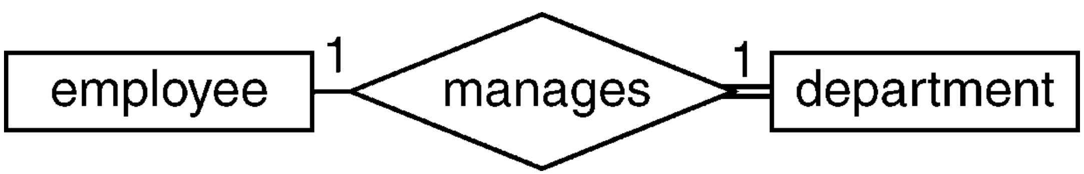
•one-to-one example

•one-to-many example



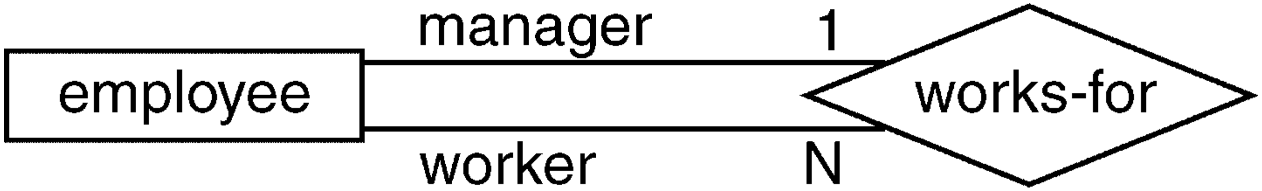
Relationships (cont'd)

- Participation constraints
  - Two types
    - Total participation (indicated by double lines)
      - Every department must be managed by a manager
      - **`department`** participation in **`manages`** relationship is total
    - Partial participation (indicated by a single line)
      - Not every employee manages a **`department`**
      - **`employee`** participation in **`manages`** relationship is partial



Relationships (cont'd)

- Recursive relationships
  - ∗ Each entity in a relationship plays a *role*
  - ∗ In a recursive relationship
    - » An entity participates more than once in different roles
- Example
  - ∗ 1:N recursive relationship on **`employee`** entity
    - » Each manager manages several workers
    - » Each worker may have only one manager



Attributes/Properties

- Describe properties of entities and relationships
  - » Both entities and relationships can have attributes
- Types of attributes
  - » Simple or composite
  - » Single-valued or multi-valued
  - » Stored/based or derived
  - * An attribute can be a
    - – Key or non-key
  - * An attribute can have a
    - – Null value

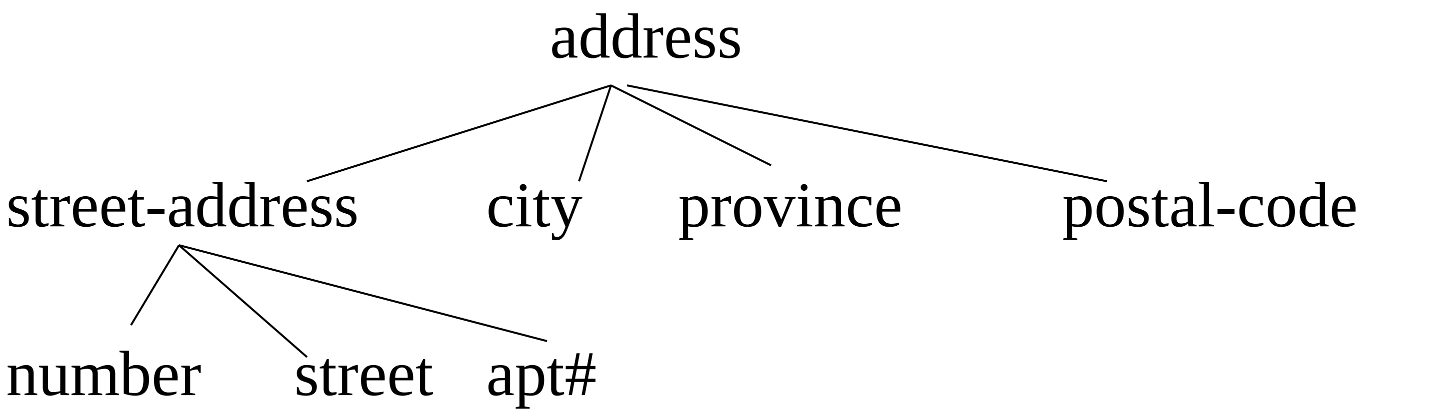
  in some circumstances

Attributes (cont'd)

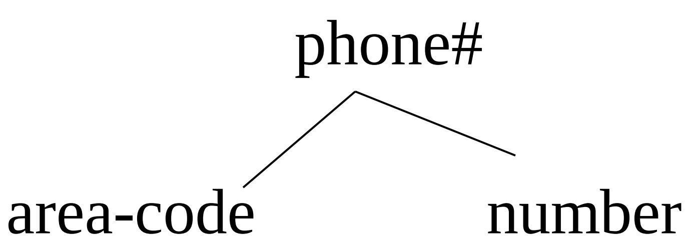
- Simple versus composite attributes
  - ∗ Simple attributes
    - – Not divisible
      - ✈called *atomic* attributes
    - – Examples
      - ✈**`part#, weight`**
  - ∗ Composite attributes
    - – Consists of several simple attributes
    - – Useful if the user refers it
      - ✈sometimes as a unit
      - ✈other times as individual components

Attributes (cont'd)

Example: Need a list of all suppliers located on Yonge street



Example: Require a list of all customers in 416 area code



Attributes (cont'd)

- Single- versus multi-valued attributes
  - ∗ Single-valued
    - » Examples: **`SIN`**, **`part-weight`**
  - ∗ Multi-valued
    - » Examples: **`college-degrees`**, **`skills`**

- Stored versus derived attributes
  - ∗ Stored attribute
    - » Example: **`date-of-birth`**
  - ∗ Derived attribute
    - » Example: **`age`**

Attributes (cont'd)

- Key or non-key attributes
  - Key attribute
    - An attribute that is unique
      - distinct for each individual entity instance
      - Examples: **`emp#`**, **`SIN`**, **`student#`**
      - Can used to identify an entity
  - Key attributes are shown underlined in the ER diagram
    - A key attribute may not be a single attribute
      - All attributes that form the key are shown underlined
      - We show only one key attribute
        - ✈Different notation is used in the text (not recommended)

Attributes (cont'd)

Keys and Identifiers

- Each entity in an entity type needs to be identified uniquely
  - Sometimes artificial attributes are created to facilitate
    - E.g. **`student#`**, **`employee#`**
  - One or more attributes can be used as an entity identifier
    - For **`marks`** entity type, **`student#`** and **`course#`** are required to find the **`grade`**

Attributes (cont'd)

* Candidate key
  * » Minimal subset of attributes that uniquely identifies an entity
    * – Example: **`employee#`**

      `SIN`
* Primary key
  * » The candidate key chosen by the designer to access each entity
    * – Example: **`employee#`**
  * » Can be defined for strong entities
  * » Weak entities may not have primary keys associated with them
  * » Note:
    * – Strong and weak only from a particular application point of view
      * ✈ Not inherent in the physical world

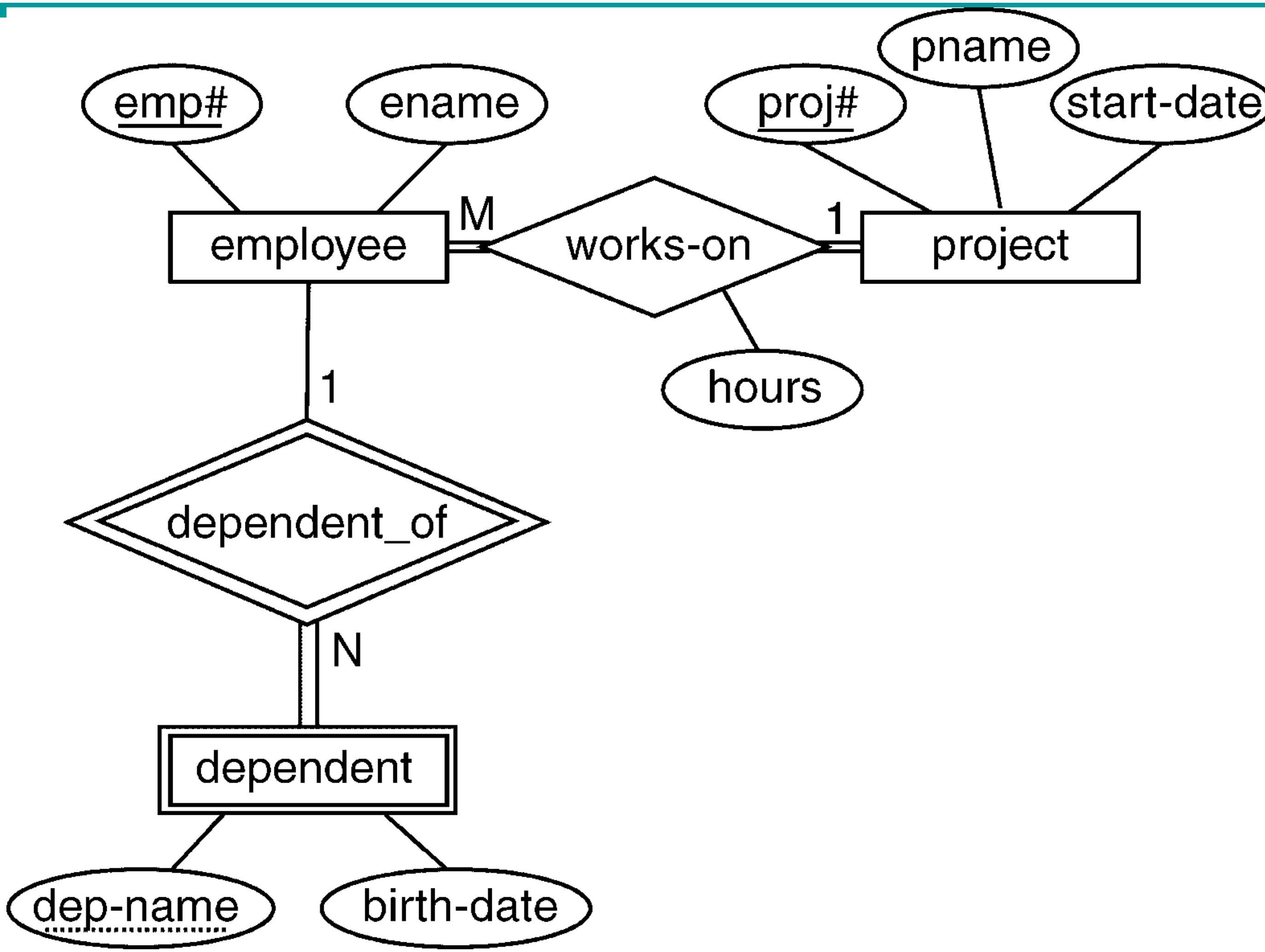
Attributes (cont'd)

- Primary key for weak entity types
  - The entity **`dependent`** cannot be identified uniquely
    - Several people may have the same name
    - We need to identify different dependents of a particular employee
  - Primary key of a weak entity type is formed by the primary key of the associated strong entity plus the weak entity discriminator
  - Example
    - **`Emp#, dep-name`** may serve as a primary key for the weak entity type **`dependent`**

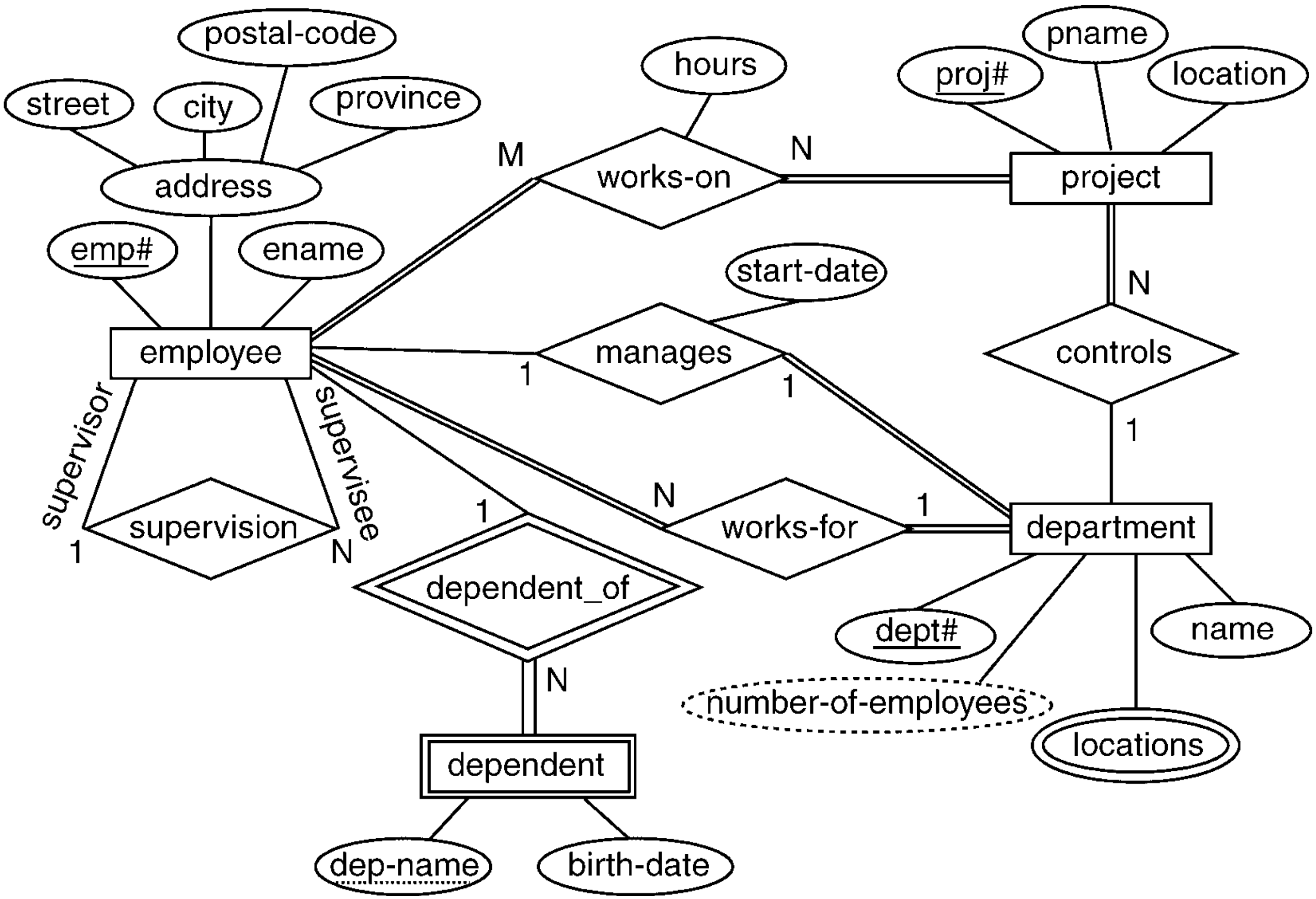
Attributes (cont'd)

- Null values
  - A special attribute value NULL is created to represent various things
    - » Not applicable
      - – A single-family home may not have **`apt#`** attribute
    - » Unknown
      - – missing information
        - ✈Not known at this time
        - ✈Examples: **`citizenship`**, **`grade`**
      - – not known
        - ✈We don't know if the attribute value exists
        - ✈Example: **`email-address`**

ER Diagram Example - 1



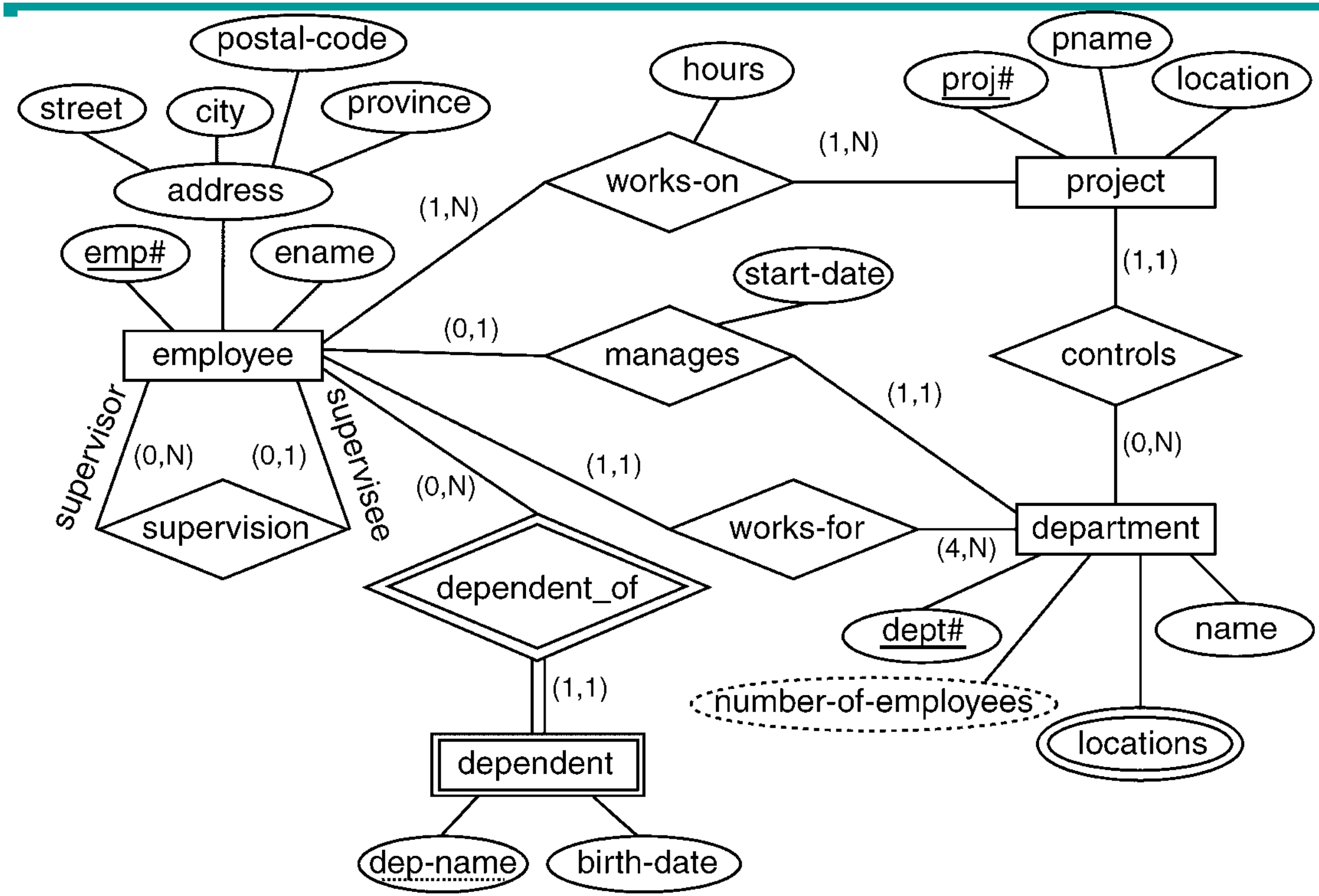
ER Diagram Example - 2



Alternative Notation for Structural Constraints

- Associate a pair of integer numbers
  - (**`min`**, **`max`**) where $0 \leq$ **`min`** $\leq$ **`max`**
  - Each entity must participate in at least **`min`** at most **`max`** relationship instance *at all times*
- More flexible mapping constraints than the three types described before
  - Can easily be applied to relationships of any degree
- Participation constraints can also be specified
  - **`min`** = 0 implies partial participation
  - **`min`** > 0 implies total participation

ER Diagram Example with (min, max)

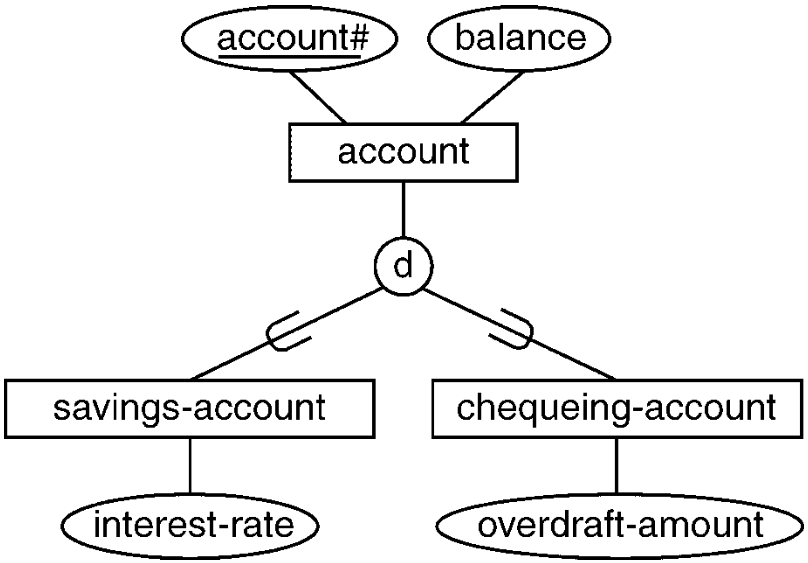


Subclass and Superclass

- Subclass(Subtypes) and superclass(Supertypes)
  - Subclass allows sub-groupings of entities
  - **`student`** entity type can have **`part-time`** and **`full-time`** student subclasses
  - **`student`** is said to be *superclass*
  - Attribute inheritance
    - » Member of a subclass inherits all the attribute of its superclass
    - » Each subclass can have its own attributes
      - – in addition to the inherited attributes

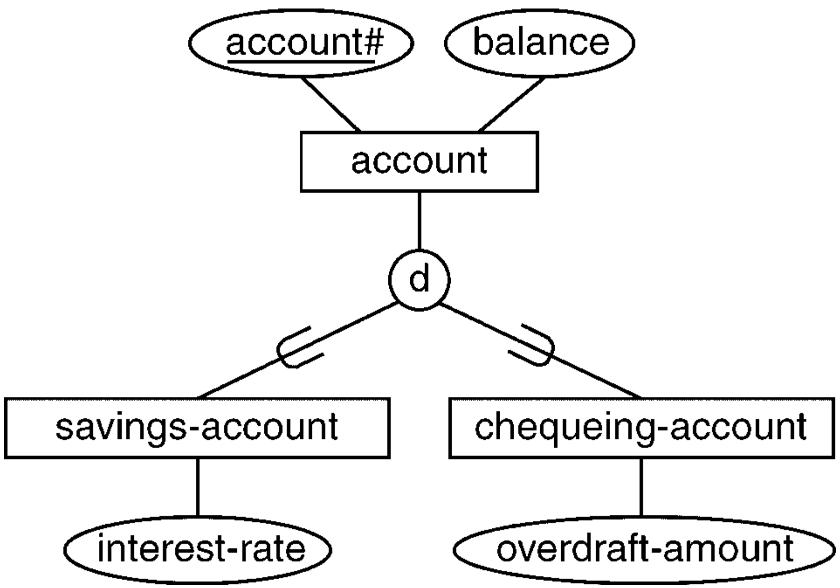
Specialization

- Specialization
  - Process of defining a set of subclasses of an entity type
    - Usually based on some distinguishing characteristic of the entity type
    - Multiple specializations can be defined on a single entity type
  - Example
    - **account** can be specialized into **savings-account** and **chequeing-account**

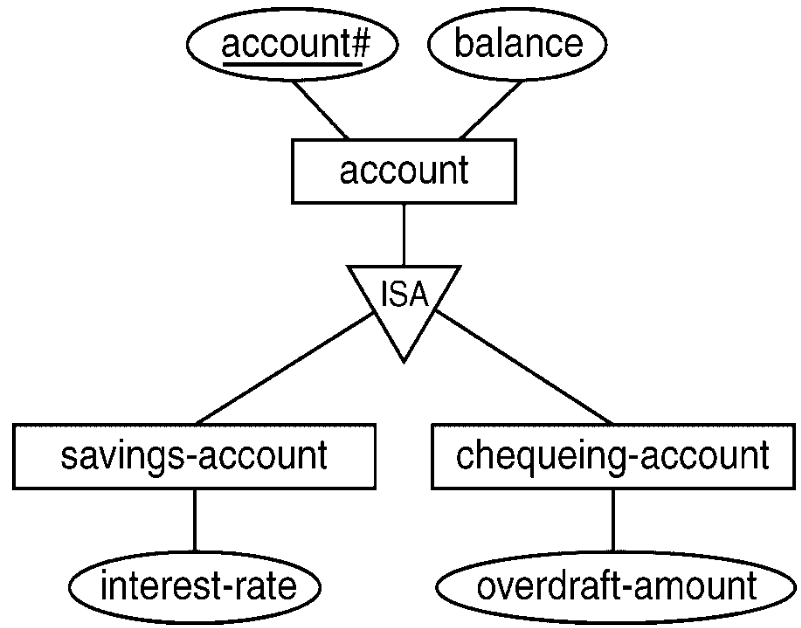


Specialization (cont'd)

Our notation



ISA notation



Specialization (cont'd)

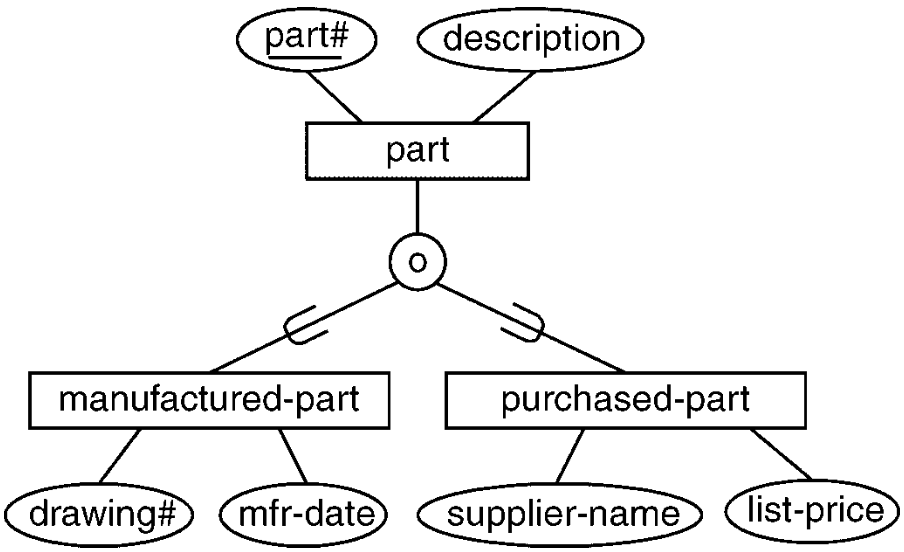
- Two constraints
  - » Disjointness constraint
  - » Completeness constraint
- Disjointness constraint
  - ∗ Disjoint
    - » An entity can be a member of at most one of the subclasses of the specialization
    - » We use "d" in ER diagrams to represent disjoint constraint
  - ∗ Overlapping
    - » The same entity can be a member of more than one subclass of the specialization
    - » We use "o" in ER diagrams to represent overlapping constraint

Specialization (cont'd)

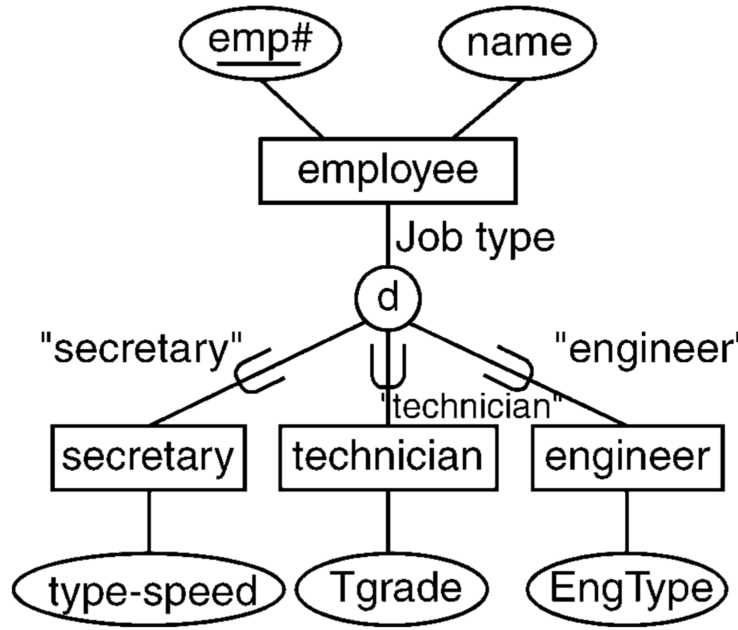
- Completeness constraint
  - ∗ Total
    - » Every entity in the superclass must be a member of some subclass in the specialization
  - ∗ Partial
    - » An entity may not belong to any of the subclasses in the specialization
- This leads to four types of specialization
    - » disjoint, total
    - » disjoint, partial
    - » overlapping, total
    - » overlapping, partial

Specialization (cont'd)

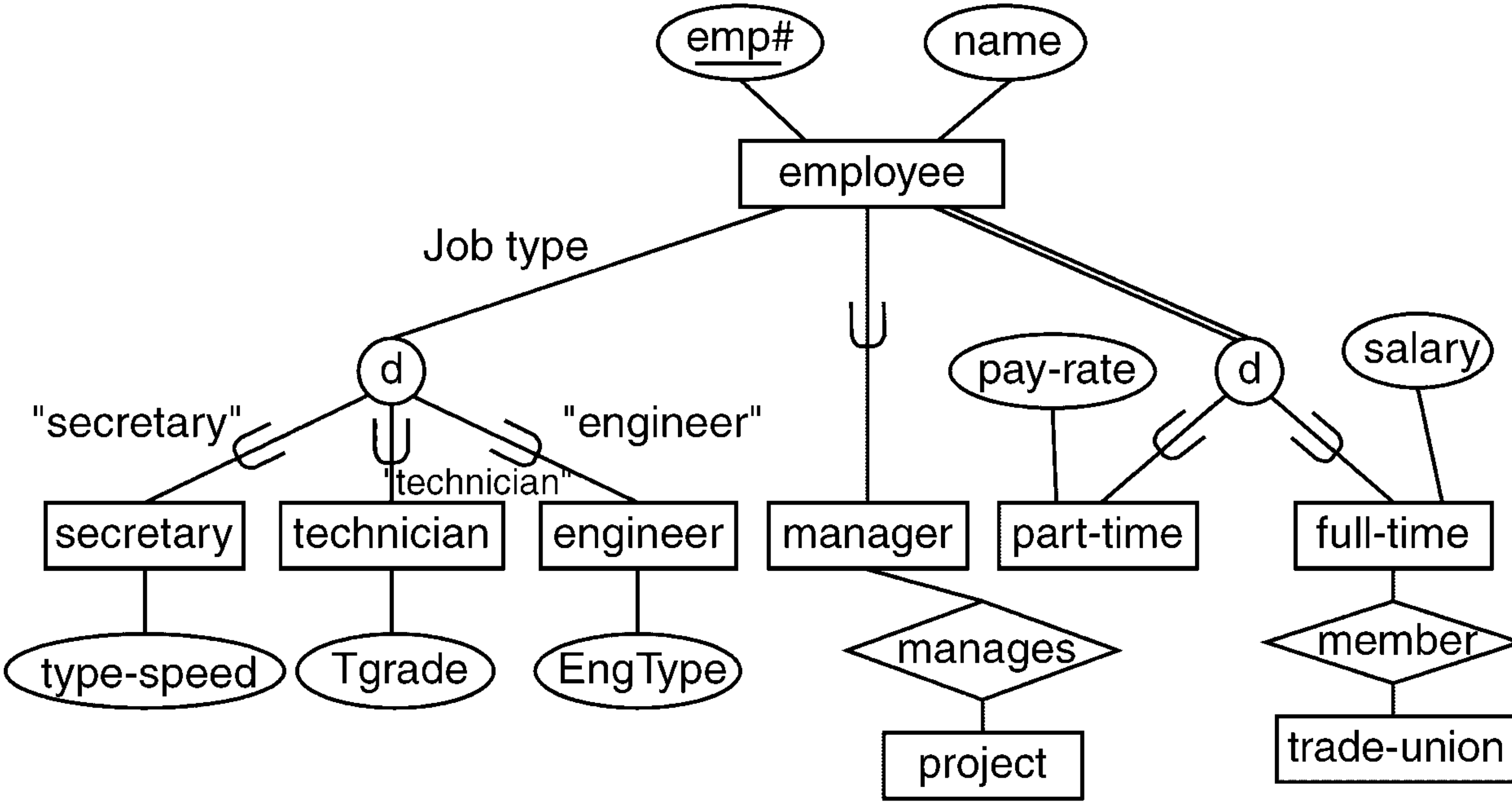
Specialization with overlapping subclasses



Attribute-defined specialization



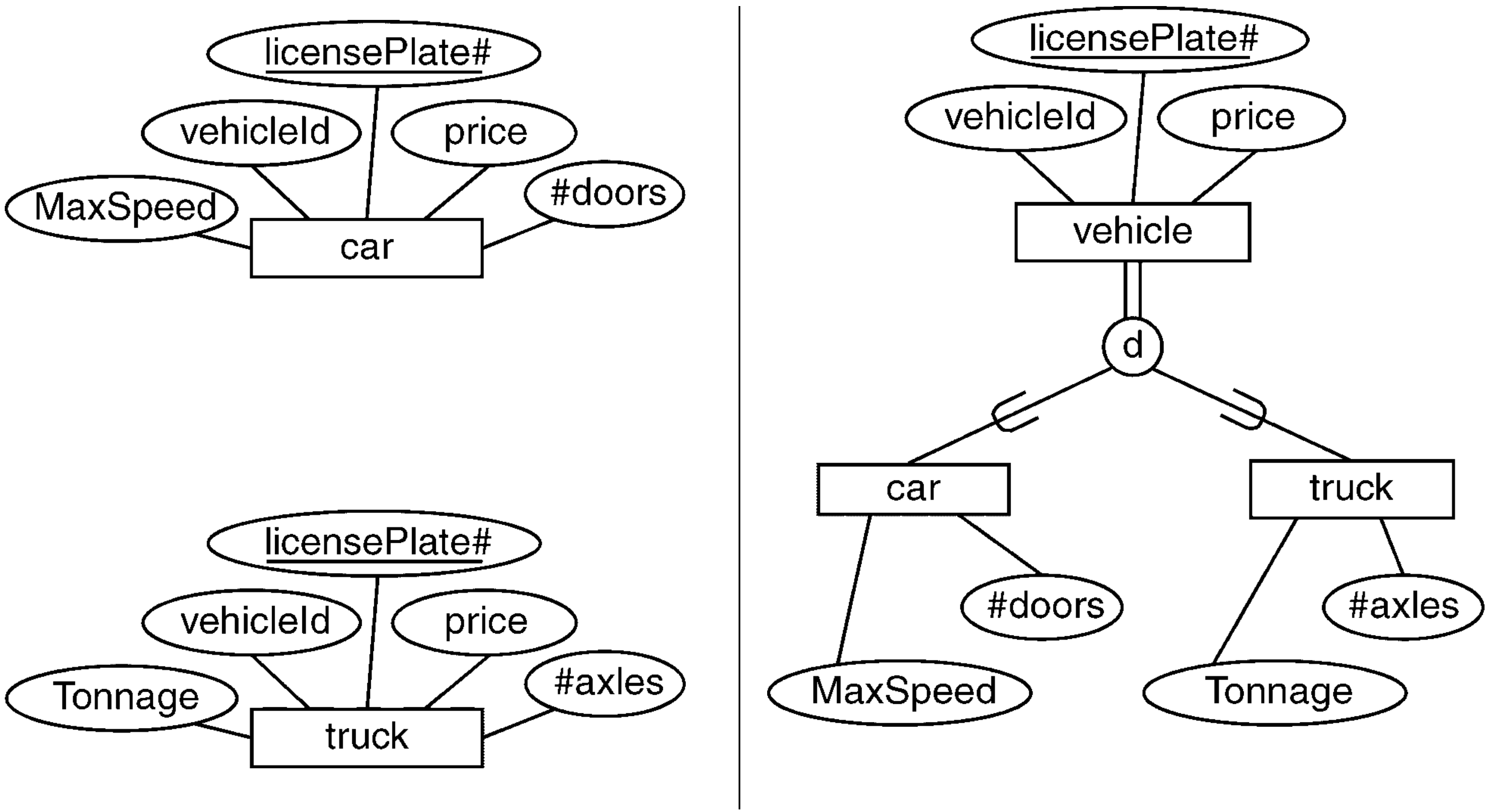
Specialization (cont'd)



Generalization

- Generalization
  - ∗ Result of taking the union of two or more lower-level entity types to produce a higher-level entity type
    - – The original entity types are special subclasses and the new higher-level entity type is the superclass
    - – Functionally the inverse of the specialization process
  - ∗ We don’t use any special notation for generalization
  - ∗ The original entities that are used in generalization are special subclasses.
  - ∗ In other words in generalization every higher-level entity must also be a lower-level entity but specialization does not have this constraint.

Generalization



Deriving Relational Schema

- Fairly straightforward to derive relational schema from the ER diagrams

Strong Entity

- ∗ An entity type E with attributes $A_1, A_2, \ldots, A_K$ is represented as a k-degree relation

$$E(A_1, A_2, \ldots, A_K)$$

  - » Each tuple of the relation represents one entity in the entity type
  - » Include only simple components of a composite attribute

Deriving Relational Schema (cont'd)

Relationship

- A relationship R among entity types $E_1, E_2, \ldots, E_K$
  - Let $P_1, P_2, \ldots, P_K$ be the primary keys of the entity sets $E_1, E_2, \ldots, E_K$ respectively
- Relationship R has attributes $A_1, A_2, \ldots, A_R$
- The relationship R is represented as a (k+r)-degree relation

$$R(P_1, P_2, \ldots, P_K, A_1, A_2, \ldots, A_R)$$

Deriving Relational Schema (cont'd)

Weak entity

- A weak entity type W has attributes $A_1, A_2, \ldots, A_W$
- Depends on strong entity type S with primary key P_S
- The weak entity is represented as

$$W(P_S, A_1, A_2, \ldots, A_W)$$

Multi-valued attribute

- A multi-valued attribute A_M of entity type E (or relationship type R) with primary key A_K is represented by

$$M(A_K, A_M)$$

- A_K and A_M together form the primary key to M

Deriving Relational Schema (cont'd)

Example

* The project-employee ER diagram (Example 1) is converted to the following five relations:

  EMPLOYEE (<u>emp#</u>, ename)

  PROJECT (<u>proj#</u>, pname, start-date)

  WORKS-ON (<u>emp#, proj#</u>, hours)

  DEPENDENT (<u>emp#, dep-name</u>, birth-date)

  DEPENDENT-OF (<u>emp#, dep-name</u>)

* Primary key shown underlined
* The last relation is redundant

Deriving Relational Schema (cont'd)

Example (cont'd)

* Problems in representing the weak entity type
  - » Using dep-name as the key means if two dependents of the same employee have the same name we have duplicated keys.
  - » Multiple occurrences of a dependent may be avoided by giving the dependent its own unique identifier
  - » The modified schema is

EMPLOYEE (<u>emp#</u>, ename)

PROJECT (<u>proj#</u>, pname, start-date)

WORKS-ON (<u>emp#, proj#</u>, hours)

DEPENDENT (<u>dep-id</u>, dep-name, birth-date)

DEPENDENT-OF (<u>emp#, dep-id</u>)

Deriving Relational Schema (cont'd)

For 1:1 and 1:M Relations

- We can avoid a separate relation by adding attributes to the associated entity
  - » Reduces redundancy
- Example revisited
  - » The revised schema is

EMPLOYEE (<u>emp#</u>, proj#, ename, hours)

PROJECT (<u>proj#</u>, pname, start-date)

~~**WORKS-ON (<u>emp#, proj#</u>, hours)**~~

DEPENDENT (<u>dep-id</u>, dep-name, birth-date)

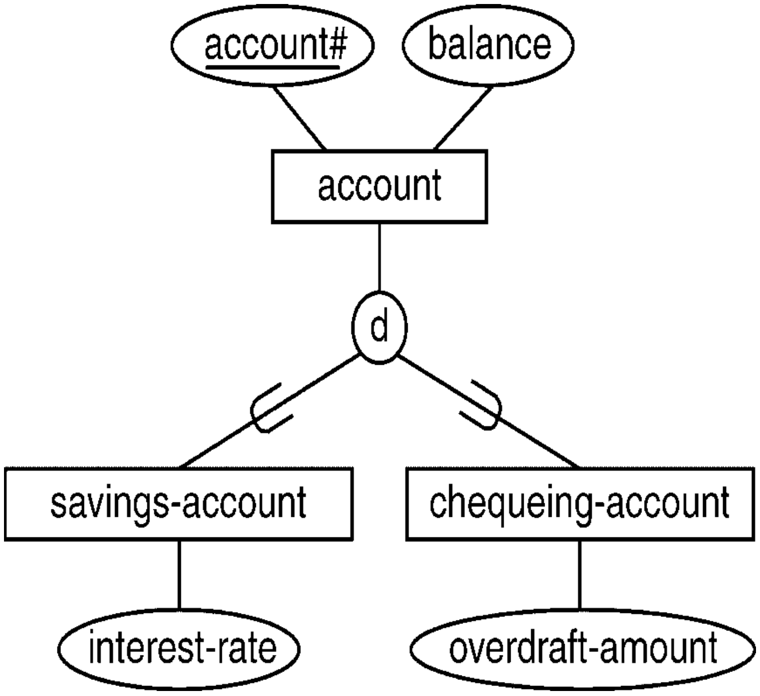
DEPENDENT-OF (<u>emp#, dep-id</u>)

Deriving Relational Schema (cont'd)

- Two methods for deriving relational schema from an ER diagram with specialization/generalization
  - Method 1
    - Create a table for the higher-level entity
    - For each lower-level entity, create a table which includes a column for each of its attributes plus for primary key of the higher-level entity
  - Method 2
    - Do not create a table for the higher-level entity
    - For each lower-level entity, create a table which includes a column for each of its attributes plus a column for each attribute of the higher-level entity

Deriving Relational Schema (cont'd)

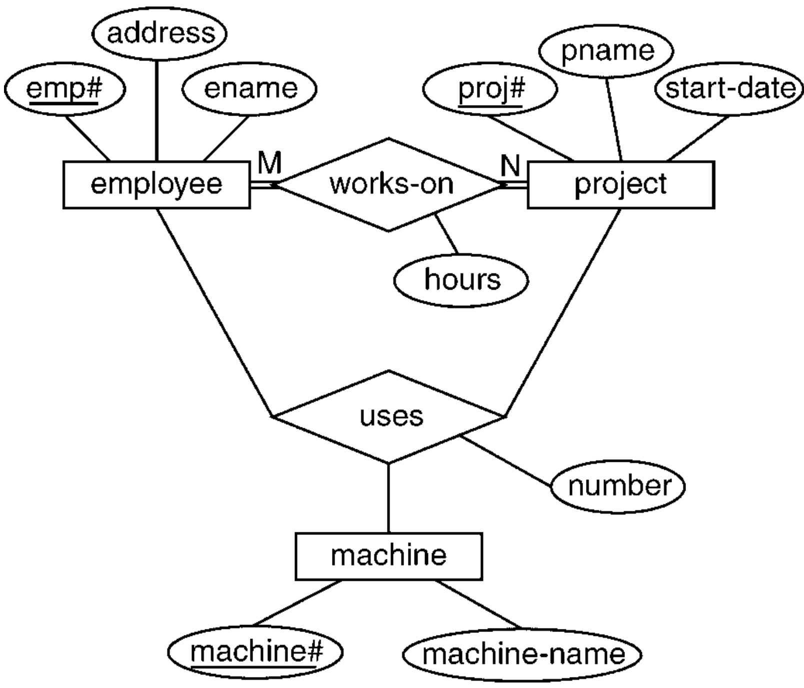
- Method 1
  - **account (<u>account#</u>, balance)**
  - **savings-account (<u>account#</u>,**
  - **interest-rate)**
  - **chequeing-account (<u>account#</u>,**
  - **overdraft-amount)**



- Method 2
  - **savings-account (<u>account#</u>,**
  - **balance, interest-rate)**
  - **chequeing-account (<u>account#</u>,**
  - **balance, overdraft-amount)**

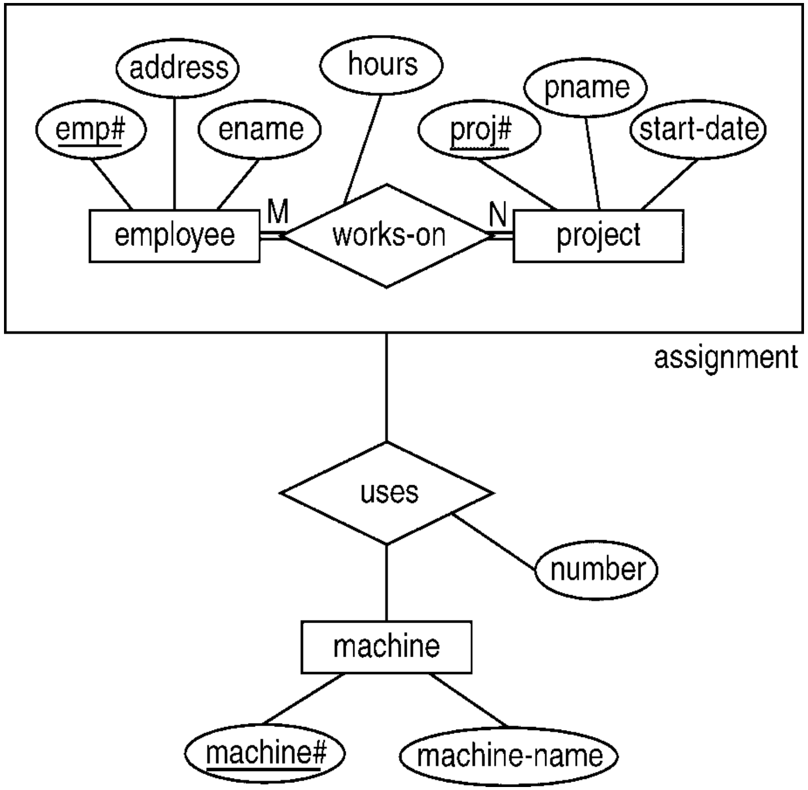
Aggregation

- Motivation
  - A limitation of the ER model
    - Not possible to express relationship among relationships
  - We may have to use two or more relationships
    - **works-on** and **uses** relationships are independent
    - But it is complicated because we just wanted to show that when employee works on a project, he/she uses a machine…



Aggregation (cont'd)

- Aggregation is an abstraction through which relationships are treated as higher-level entities
- Example
  - ∗ We create a new higher-level entity called **assignment**
  - ∗ Now we can establish relationships by treating this new entity as a regular entity



Aggregation (cont'd)

- Deriving relational schema
  - ∗ Transform the higher-level entity
    - » Use the procedure described before
  - ∗ Transform the aggregate relationship
    - » Entity types participating in the higher-level entity H: E_1, E_2, …, E_{K-1}
    - » Let P_1, P_2, …, P_K be the primary keys of E_1, E_2, …, E_K respectively
    - » Attributes of relationship R between entity types H and E_K: A_1, A_2, …, A_R
    - » The relationship is represented by

$$R(P_1, P_2, \ldots, P_{K-1}, P_K, A_1, A_2, \ldots, A_R)$$

Aggregation (cont'd)

Example

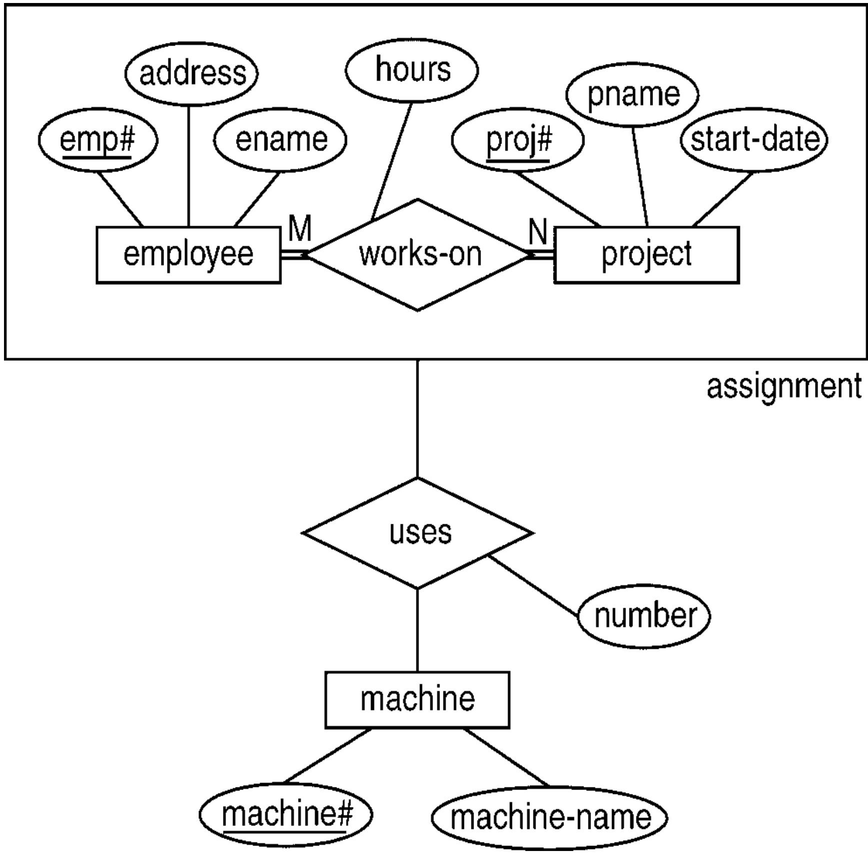
EMPLOYEE(<u>emp#</u>, ename, adddress)

PROJECT(<u>proj#</u>, pname, start-date)

WORKS-ON(<u>emp#,proj#</u>, hours)

MACHINE(<u>machine#</u>, machine-name)

USES(<u>emp#,proj#, machine#</u>, number)



E/R Diagram and Data Dictionary

- As mentioned before, data dictionary is the database designer’s database
- The results of E/R diagram can be used to identify the kinds of objects the dictionary needs to support
- For example a weak or strong entity, total or partial participation in a relationship and a supertype or subtype entity and etc., all can be explained in a data dictionary.

Project: University Database

Consider the following requirements for a university database

- The university keeps track of each student's name, address, student number, social insurance number, and the courses they have registered.
- In addition, for undergraduate and graduate students the degree program (BA, BCS, MSc, PhD) they are in is also maintained. (For other students such as special students, exchange students etc. this information is not needed.)

University Database

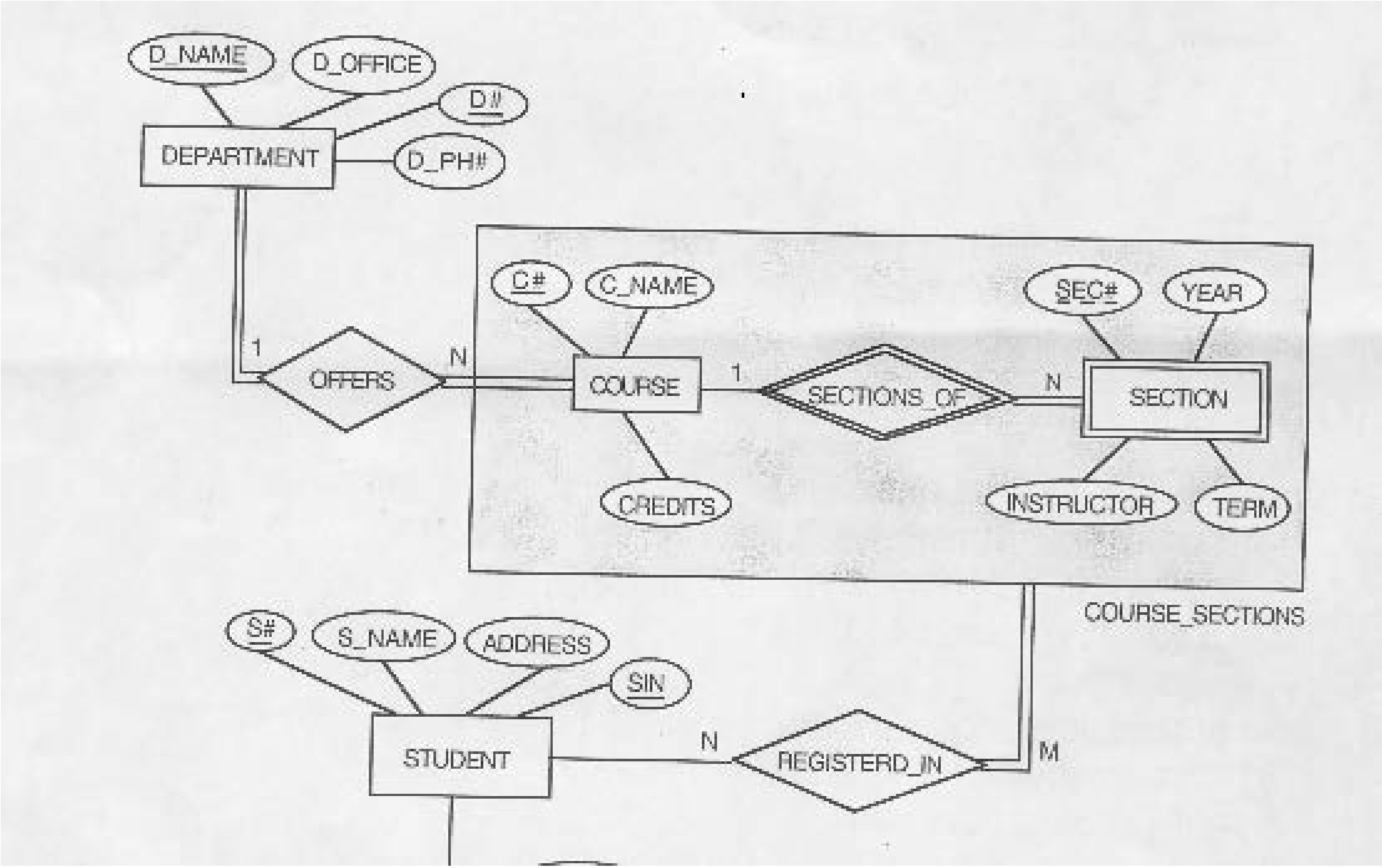
- For graduate students, a list of degrees held (degree, university, and the year degree was awarded) and their office in the department and phone number are included in the database.
- All graduate students are financially supported either by a teaching assistantship (TA) or by a research assistantship (RA). For the TAs we would like to keep the number of hours per week they are working and for the RAs the research project they are associated with (just research project name).

University Database

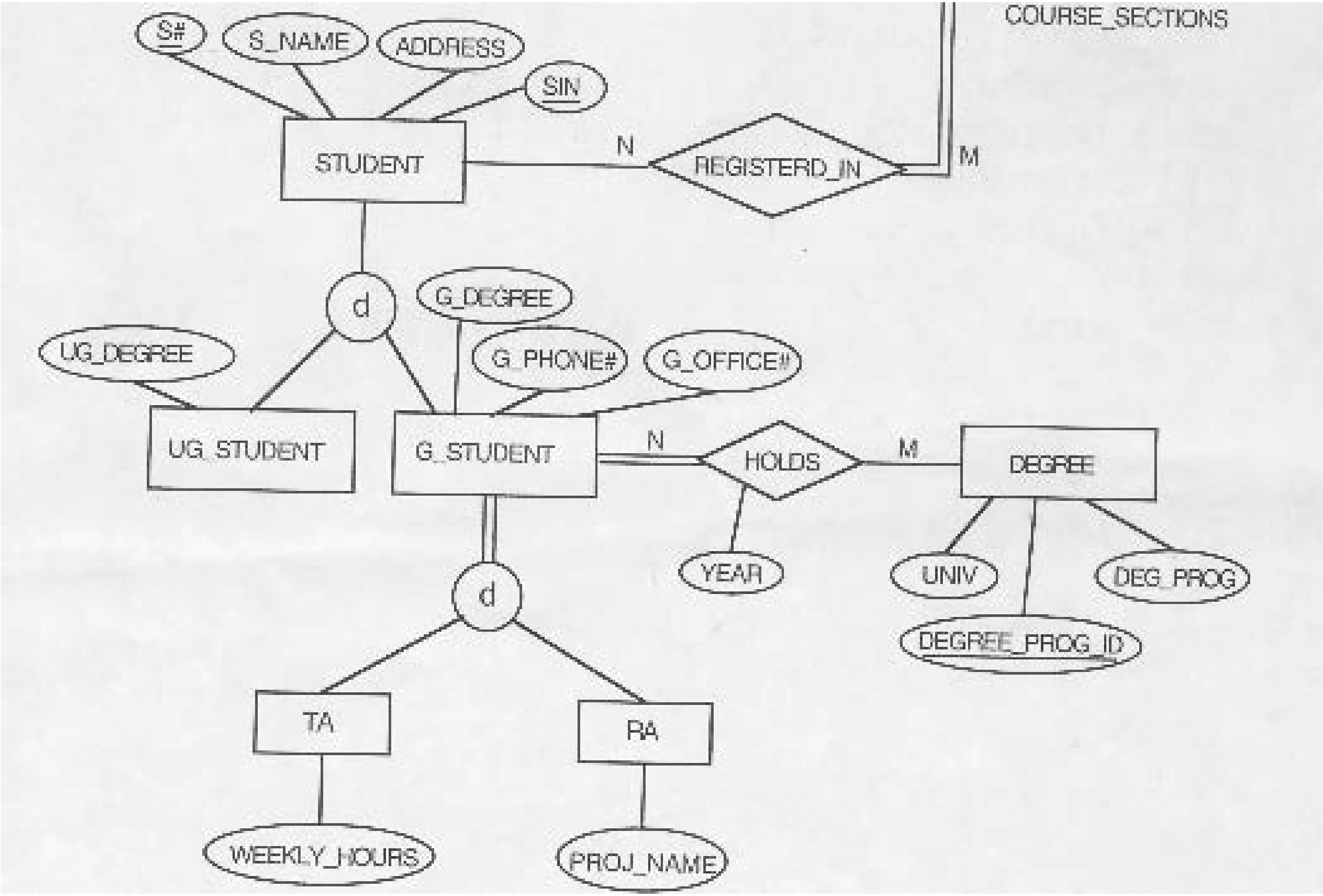
- Each department is represented. The data about departments are its name, department code, office number, and office phone. Both name and code have unique values for each department.
- Each course has a course number, course name, number of credits and the offering department. The value of course number is unique for each course.

University Database

- Each section has an instructor, term, year, course, and section number. The section number distinguishes different sections of the same course that are taught during the same year; its values are 1, 2, 3,..., up to the number of sections taught during each year.
- The ER diagram is shown in the next two slides
- Specify a preliminary relational database schema for ER diagram



CONTINUES IN THE NEXT SLIDE


COURSE_SECTIONS
S#
S_NAME
ADDRESS
SIN
STUDENT
N
REGISTERD_IN
M
d
G_DEGREE
UG_DEGREE
G_PHONE#
G_OFFICE#
UG_STUDENT
G_STUDENT
N
HOLDS
M
DEGREE
YEAR
UNIV
DEG_PROG
DEGREE_PROG_ID
d
TA
RA
WEEKLY_HOURS
PROJ_NAME

Bank Database