

Not related to class material..

Mental Health Awareness

- ✧ Towards exam time, **stress** and **anxiety** levels increase **significantly** among UofT students

- ✧ Take care of your mental health and well being!
 1. Eat healthy + Exercise
 2. Surround yourself with good people
 3. Set realistic goals
 4. Value yourself
 5. Break up the routine occasionally
 6. Get help when you need it

(Uoft Health & Wellness Centre:
<http://www.studentlife.utoronto.ca/hwc>)



The Entity/Relationship (E/R) Model & DB Design

Introduction to databases

CSC343, Fall 2015

Based on slides by Manos Papagelis

Thanks to Ryan Johnson, John Mylopoulos, Arnold Rosenbloom
and Renee Miller for material in these slides

[illegible]

Overview

- Using the **Entity/Relationship** (ER) Model to model the real world
- From there, designing a **database schema**
 - Restructuring of an E/R model
 - Translating an E/R model into a logical model (DB Schema)

THE ENTITY/RELATIONSHIP (E/R) MODEL

Conceptualizing the real-world

- DB design begins with a **boss** or **client** who wants a database.



- E.g. My first task a Software Engineer was to “[design a database to automate student admission processes](#)”



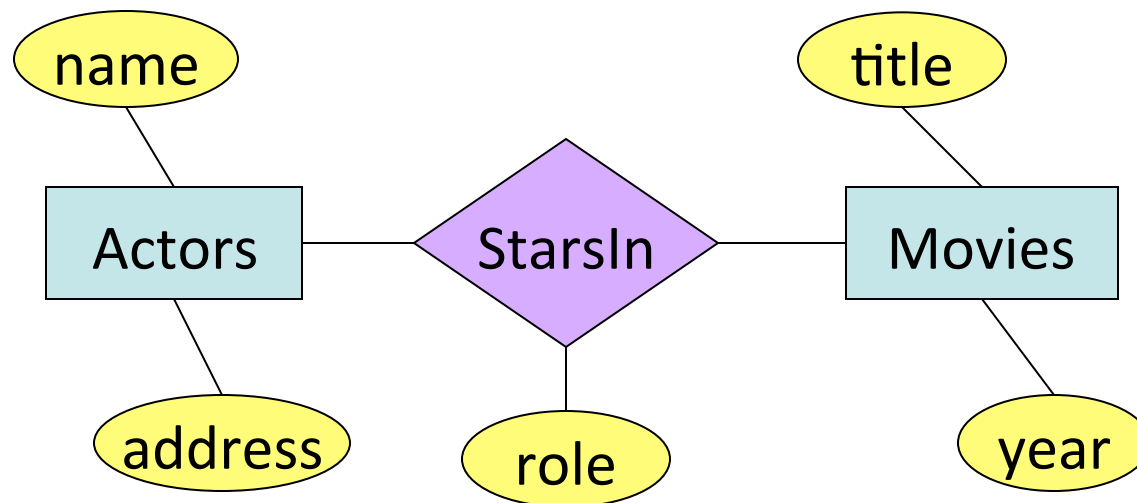
Conceptualizing the real-world

- We must **map** the *entities* and *relationships* of the **world** into the concepts of a **database**.
This is called **modeling**.
- Sketching the key components is an efficient way to develop a design.
 - Sketch out (and debug) schema designs
 - Express as many constraints as possible
 - Convert to relational DB once the client is happy



Entity/Relationship Model

- Visual data model (diagram-based)
 - Quickly “chart out” a database design
 - Easier to “see” big picture
 - Comparable to class diagrams in UML
- Basic concept: *entities* and their *relationships*, along with the *attributes* describing them



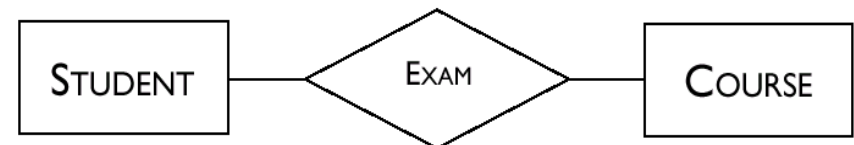
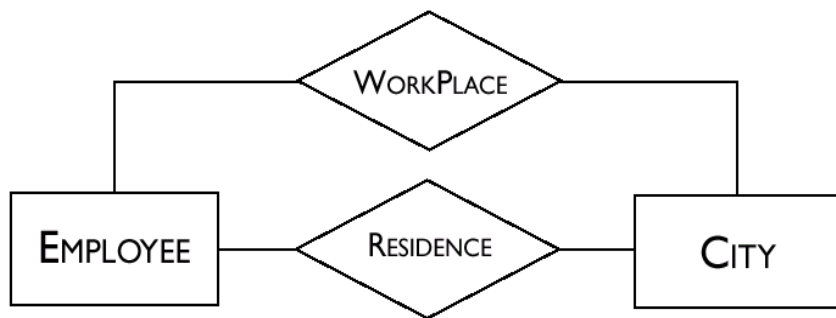
Entity Sets

- An **entity set** represents a category of objects that have properties in common and an autonomous existence (e.g., City, Department, Employee, Sale)
- An **entity** is an instance of an entity set (e.g., Stockholm is a City; Peterson is an Employee)

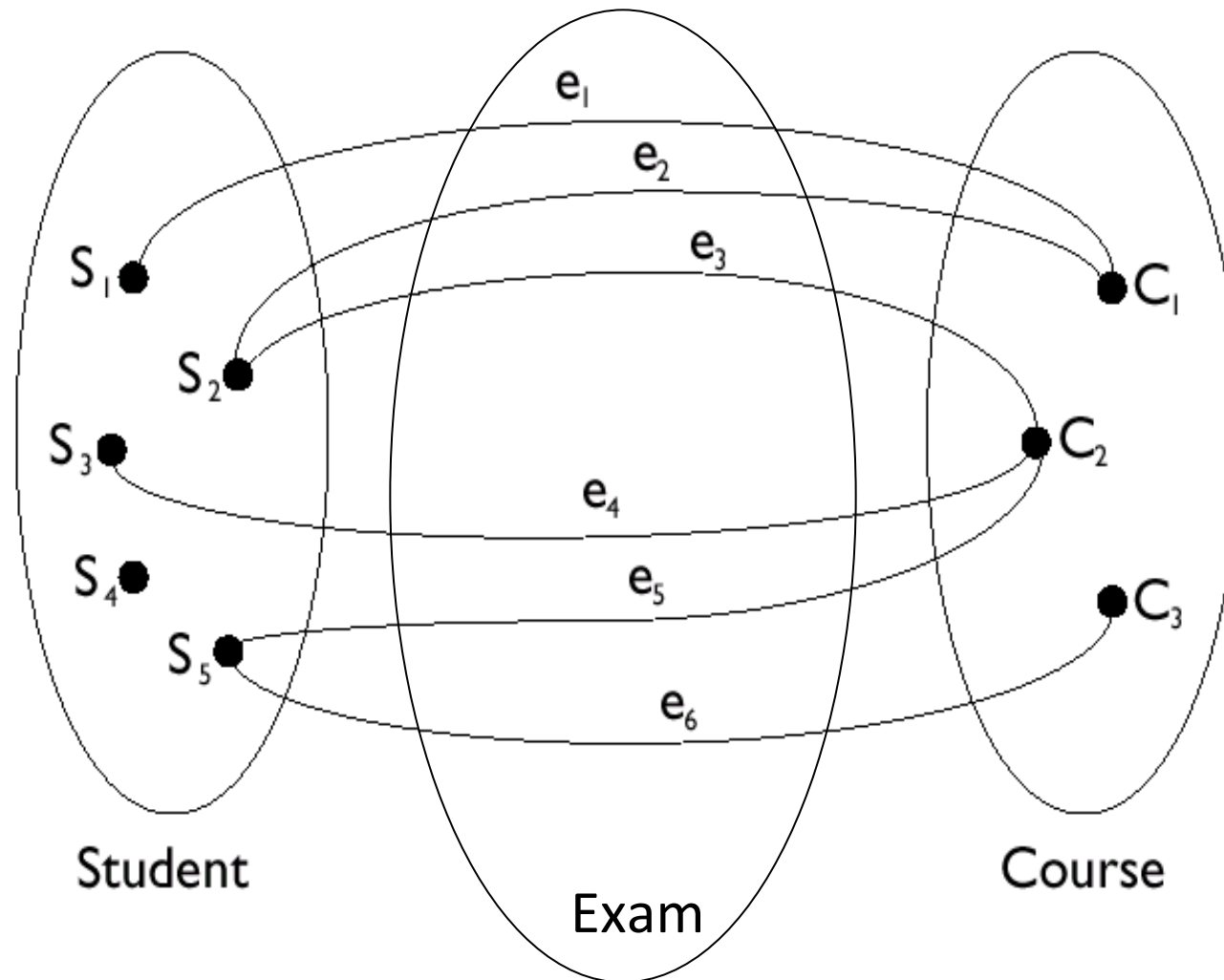


Relationship Sets

- A **relationship set** is an association between 2+ entity sets (e.g., **Residence** is a relationship set between entity sets **City** and **Employee**)
- A **relationship** is an instance of a **n-ary** relationship set (e.g., the pair <Johanssen, Stockholm> is an instance of relationship **Residence**)



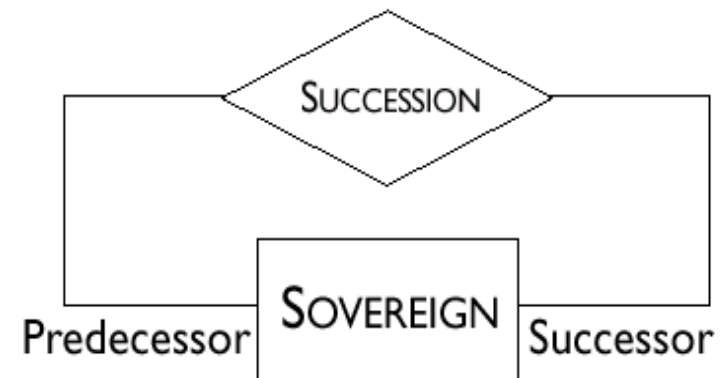
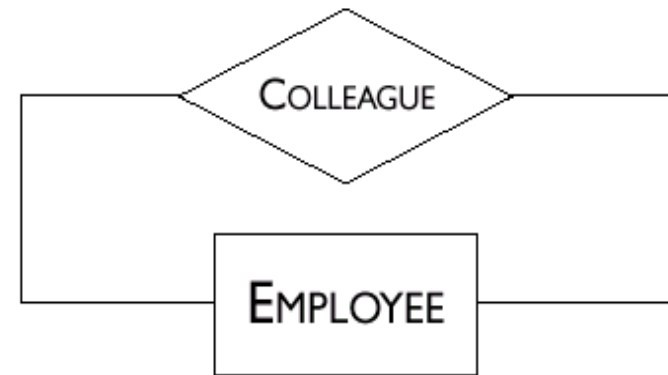
Example of Instances for Exam



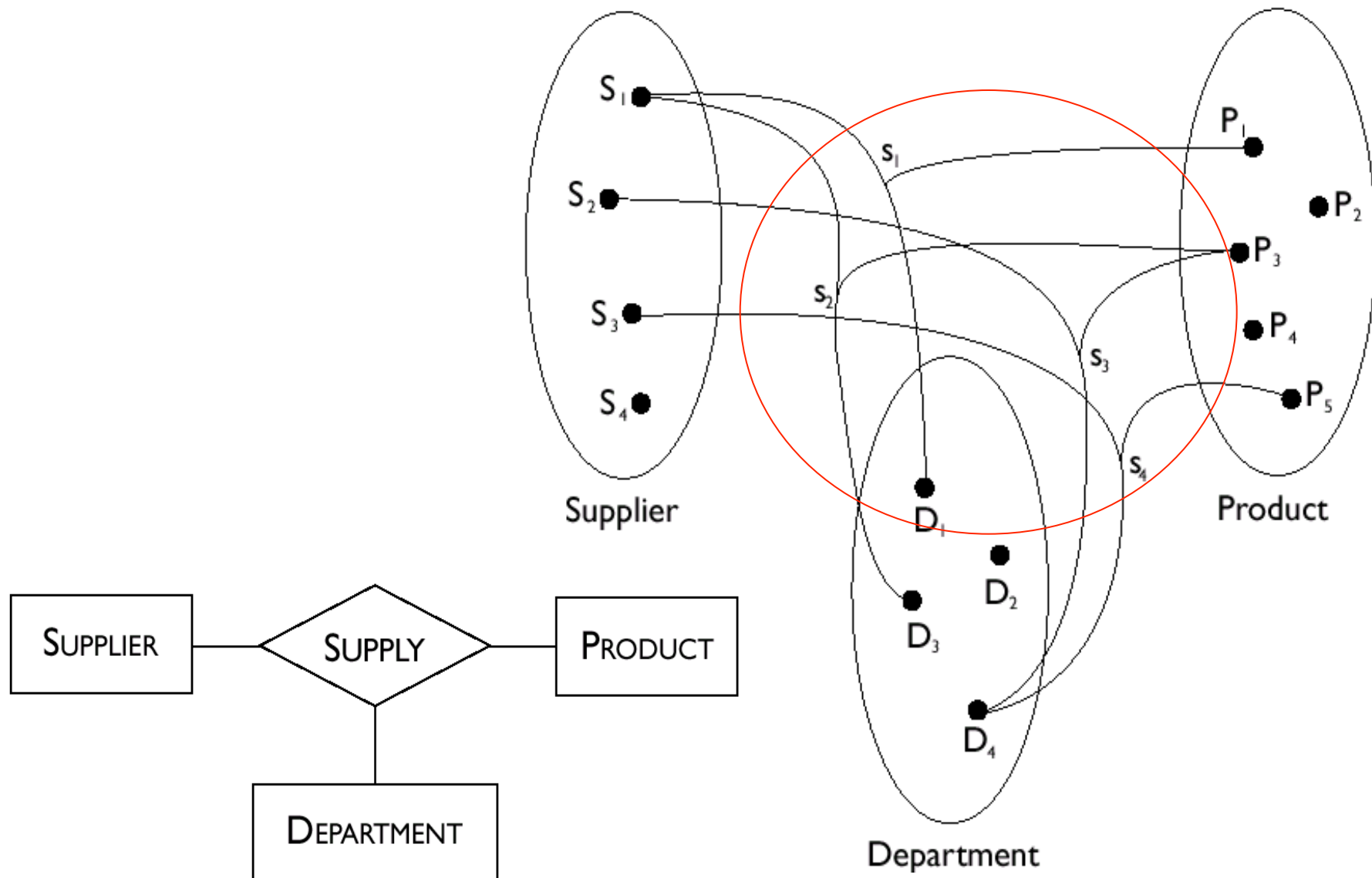
A student can't take more than one exam for a particular course

Recursive Relationships

- Recursive relationships relate an entity to itself
- Note in the second example that the relationship is not symmetric
 - In this case, it is necessary to indicate the two **roles** that the entity plays in the relationship

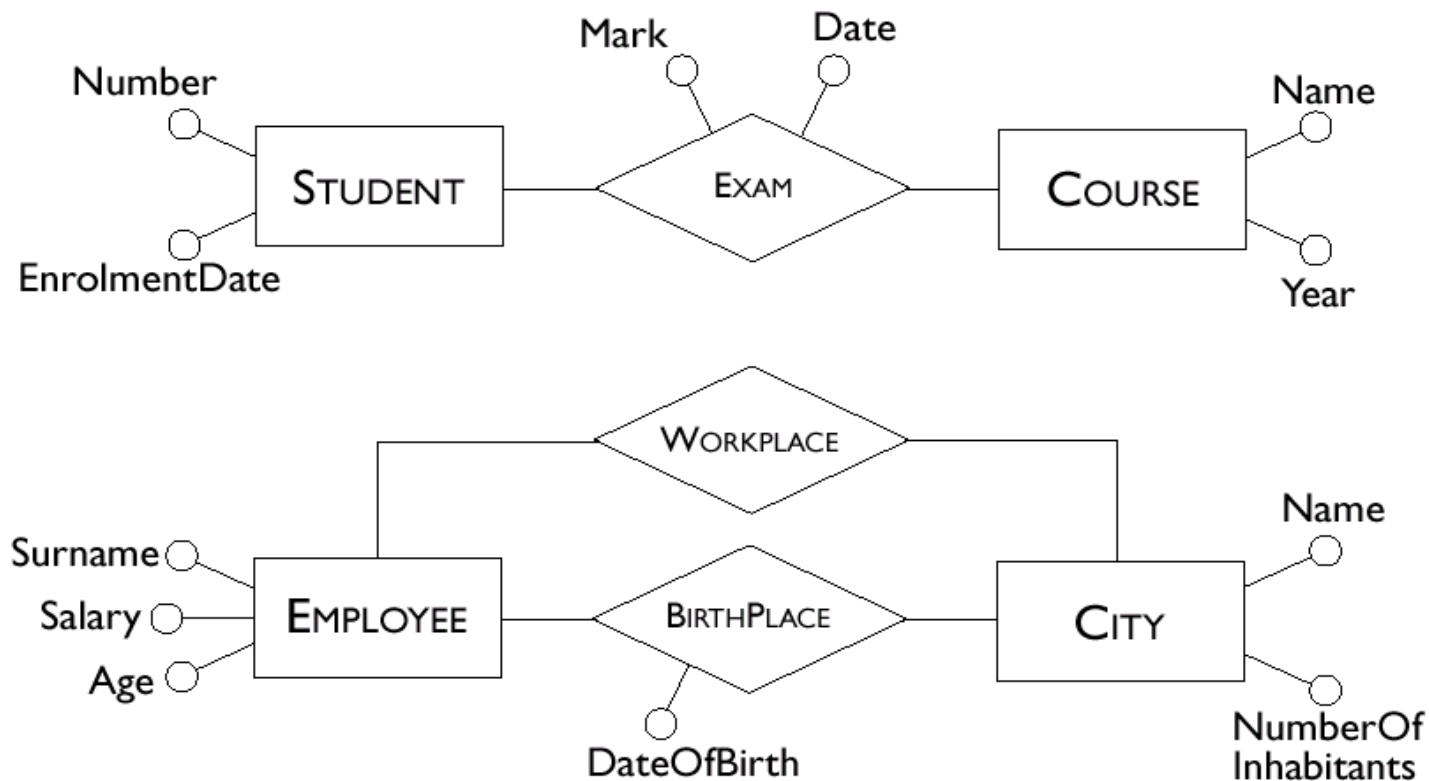


Ternary Relationships



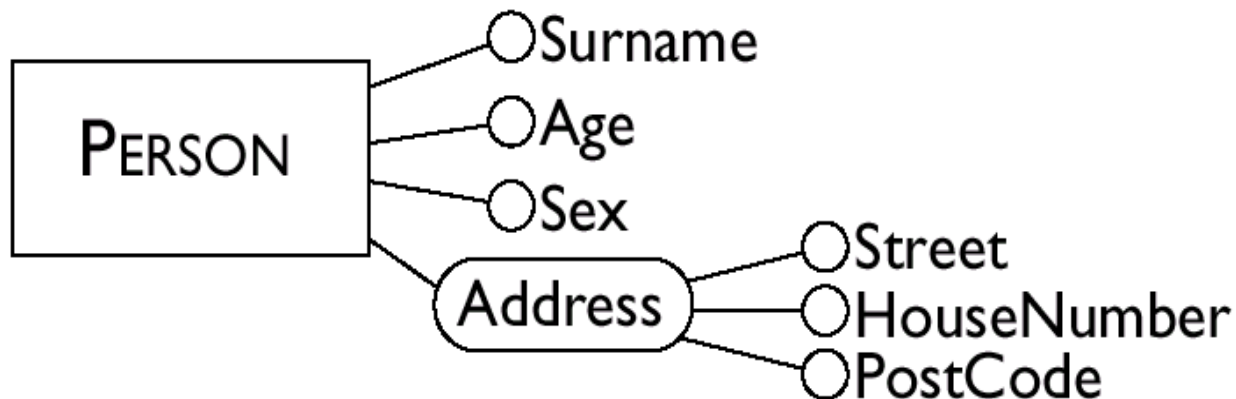
Attributes

- Describe elementary properties of entities or relationships (e.g., Surname, Salary and Age are attributes of Employee)
- May be single-valued, or multi-valued

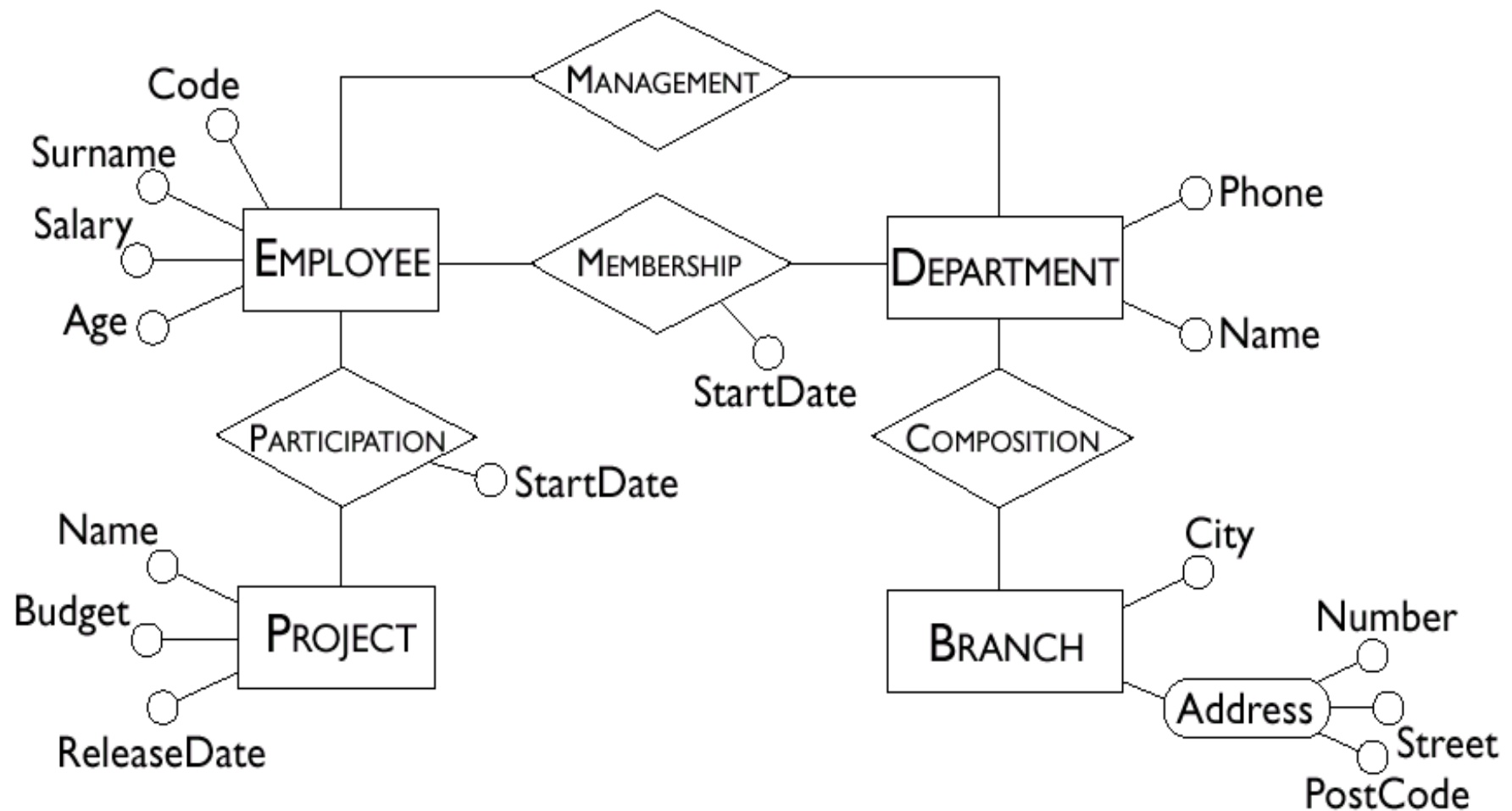


Composite Attributes

- **composite attributes** are grouped attributes of the same entity or relationship that have closely connected meaning or uses



Example Schema with Attributes



Cardinalities

- Each entity set participates in a relationship set with a minimum (**min**) and a maximum (**max**) cardinality
- Cardinalities **constrain** how entity instances participate in relationship instances
- Graphical representation in E/R Diagrams:
pairs of (**min, max**) values for each entity set

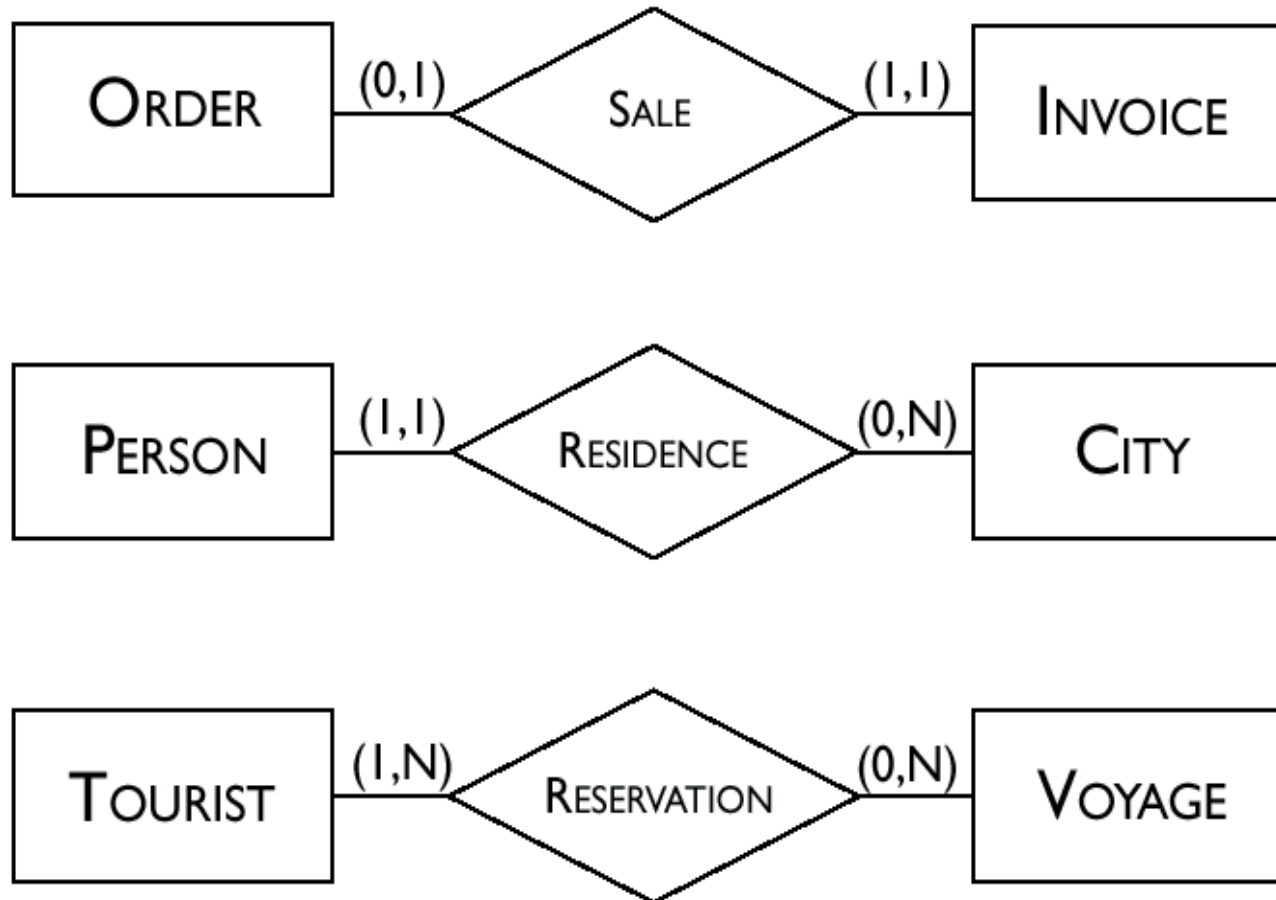


Notice that: an entity might not participate in any relationship

Cardinalities (cont.)

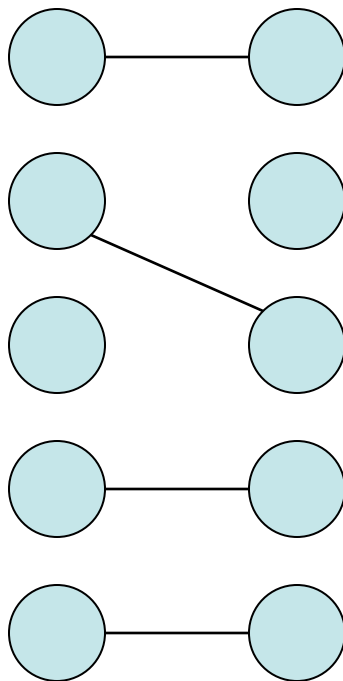
- In principle, cardinalities are pairs of non-negative integers (n, N) such that $n \leq N$, where N means "any number"
- minimum cardinality n :
 - If 0, entity participation in a relationship is optional
 - If 1, entity participation in a relationship is mandatory
- maximum cardinality N :
 - If 1, each instance of the entity is associated at most with a single instance of the relationship
 - If N , then each instance of the entity is associated with many instances of the relationship

Cardinality Examples

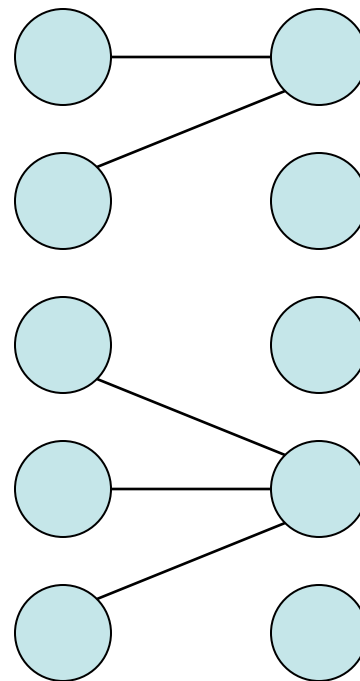
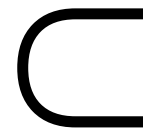


Multiplicity of relationships

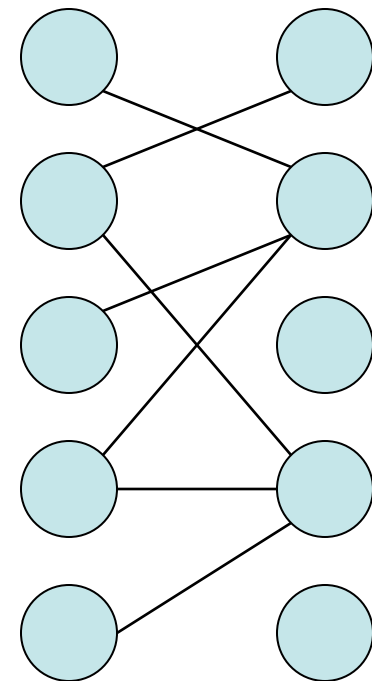
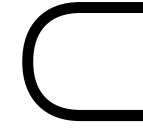
If entities $E1$ and $E2$ participate in relationship R with cardinalities $(n1, N1)$ and $(n2, N2)$ then the multiplicity of R is $N1\text{-to-}N2$ (which is the same as saying $N2\text{-to-}N1$)



1-to-1



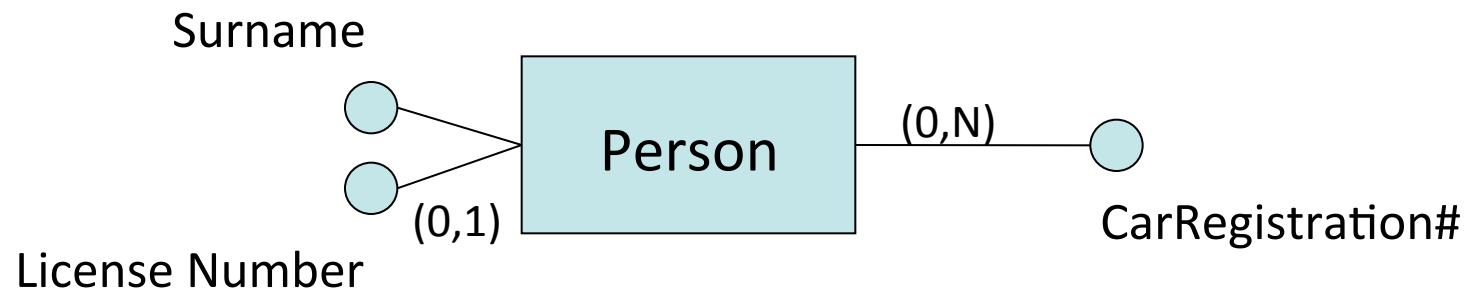
N-to-1 OR 1-to-N



N-to-N

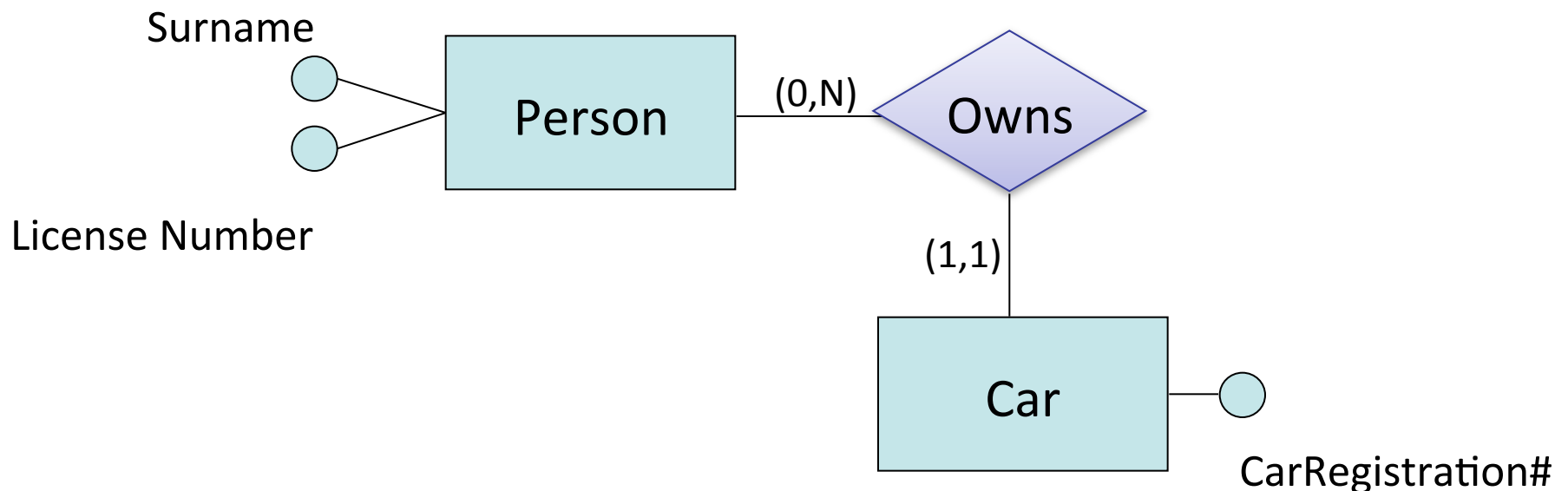
Cardinalities of Attributes

- Describe min/max number of values an attribute can have
- When the cardinality of an attribute is (1, 1) it can be omitted (**single-valued attributes**)
- The value of an attribute, may also be **null**, or have **several** values (**multi-valued attributes**)



Cardinalities of Attributes (cont.)

- **Multi-valued** attributes often represent situations that can be modeled with **additional entities**. E.g., the ER schema of the previous slide can be revised into:

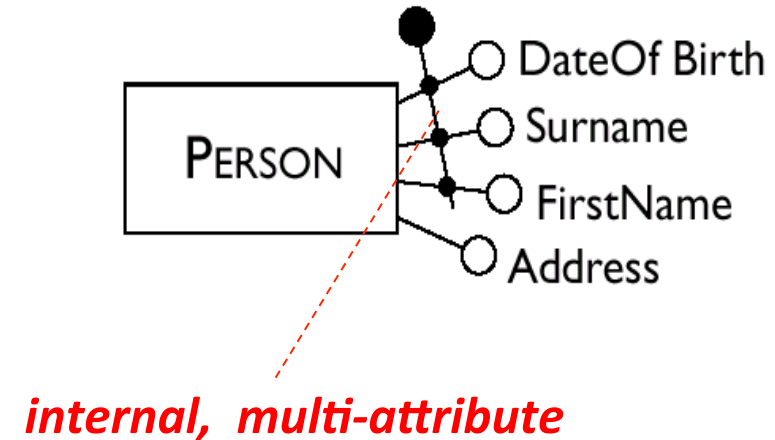
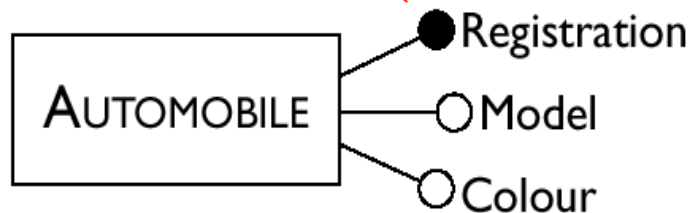


Keys in E/R

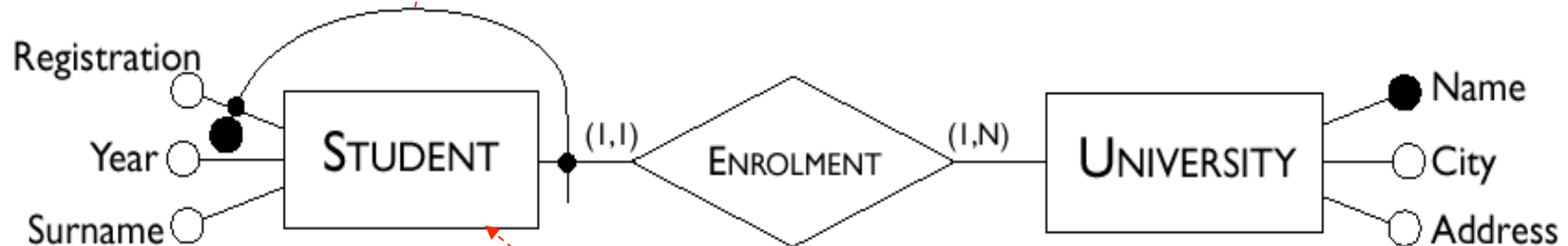
- **Keys** consist of minimal sets of attributes which uniquely identify instances of an entity set
 - socialInsurance# may be a key for Person
 - firstName, middleName, lastName, address may be a key for Person
- In most cases, a key is formed by one or more **attributes** of the entity itself (*internal keys*)
- Sometimes, an entity doesn't have a key among its attributes. This is called a *weak entity*.
Solution: the keys of related entities brought in to help with identification (becoming *foreign keys*).
- A key for a **relationship** consists of the keys of the *entities* it relates

Examples of Keys in E/R

internal, single-attribute



foreign, multi-attribute

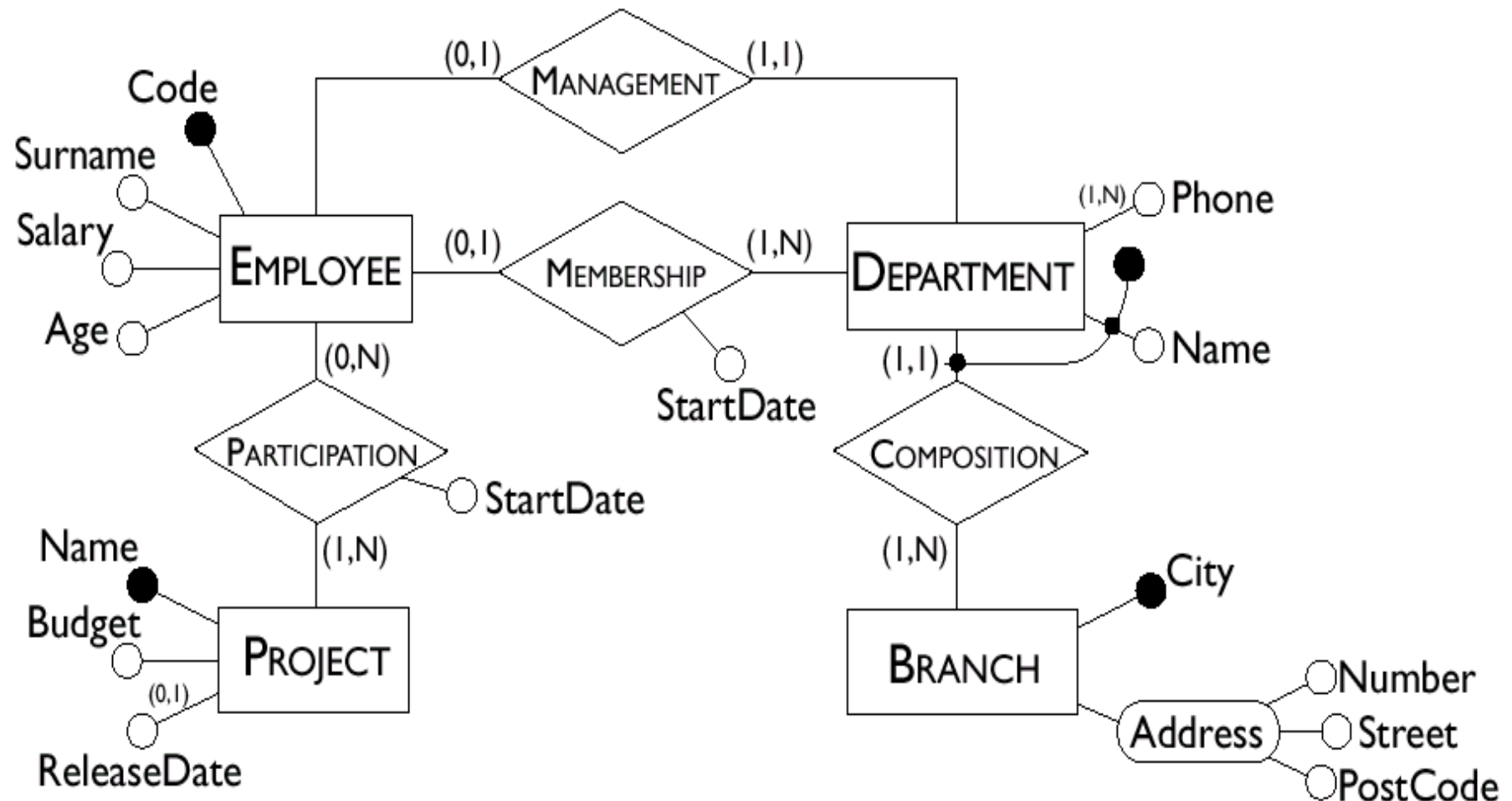


Weak entity

General Observations about Keys

- A **key** may consist of one or more attributes, provided that each of the **attributes** has **(1,1)** cardinality
- A **foreign key** can involve one or more **entities**, provided that each of them is member of a relationship to which the entity to be identified participates in the **relationship** with cardinality equal to **(1,1)**
- A **foreign key** may involve an **entity** that has itself a foreign key, as long as *cycles* are not generated
- Each entity set must have **at least one** (internal or foreign) key

Schema with Keys



Challenge: modeling the “real world”

- Life is arbitrarily complex
 - Directors who are also actors? Actors who play multiple roles in one movie? Animal actors?
- **Design choices**: Should a concept be modeled as an entity, an attribute, or a relationship?
- **Limitations of the ER Model**: A lot of data semantics can be captured but some cannot
- Key to successful model: **parsimony**
 - As complex as necessary, but no more
 - Choose to represent only “relevant” things



EXAMPLE

From real world to E/R Model

We wish to create a database for a company that runs training courses. For this, we must store data about trainees and instructors. For each course participant (about 5,000 in all), identified by a code, we want to store her social security number, surname, age, sex, place of birth, employer's name, address and telephone number, previous employers (and periods employed), the courses attended (there are about 200 courses) and the final assessment for each course. We need also to represent the seminars that each participant is attending at present and, for each day, the places and times the classes are held.

Each course has a code and a title and any course can be given any number of times. Each time a particular course is given, we will call it an 'edition' of the course. For each edition, we represent the start date, the end date, and the number of participants. If a trainee is self-employed, we need to know her area of expertise, and, if appropriate, her title. For somebody who works for a company, we store the level and position held. For each instructor (about 300), we will show the surname, age, place of birth, the edition of the course taught, those taught in the past and the courses that the tutor is qualified to teach. All the instructors' telephone numbers are also stored. An instructor can be permanently employed by the training company or freelance.

Identify the **entities**

We wish to create a database for a company that runs training courses. For this, we must store data about the *trainees* and the *instructors*. For each *course participant* (about 5,000), identified by a code, we want to store her social security number, surname, age, sex, place of birth, employer's name, address and telephone number, previous employers (and periods employed), the courses attended (there are about 200 courses) and the final assessment for each course. We need also to represent the *seminars* that each participant is attending at present and, for each day, the places and times the classes are held.

Each *course* has a code and a title and any course can be given any number of times. Each time a particular course is given, we will call it an 'edition' of the course. For each edition, we represent the start date, the end date, and the number of participants. If a trainee is self-employed, we need to know her area of expertise, and, if appropriate, her title. For somebody who works for a company, we store the level and position held. For each *instructor* (about 300), we will show the surname, age, place of birth, the edition of the course taught, those taught in the past and the courses that the *tutor* is qualified to teach. All the instructors' telephone numbers are also stored. An instructor can be permanently employed by the training company or freelance.

Glossary

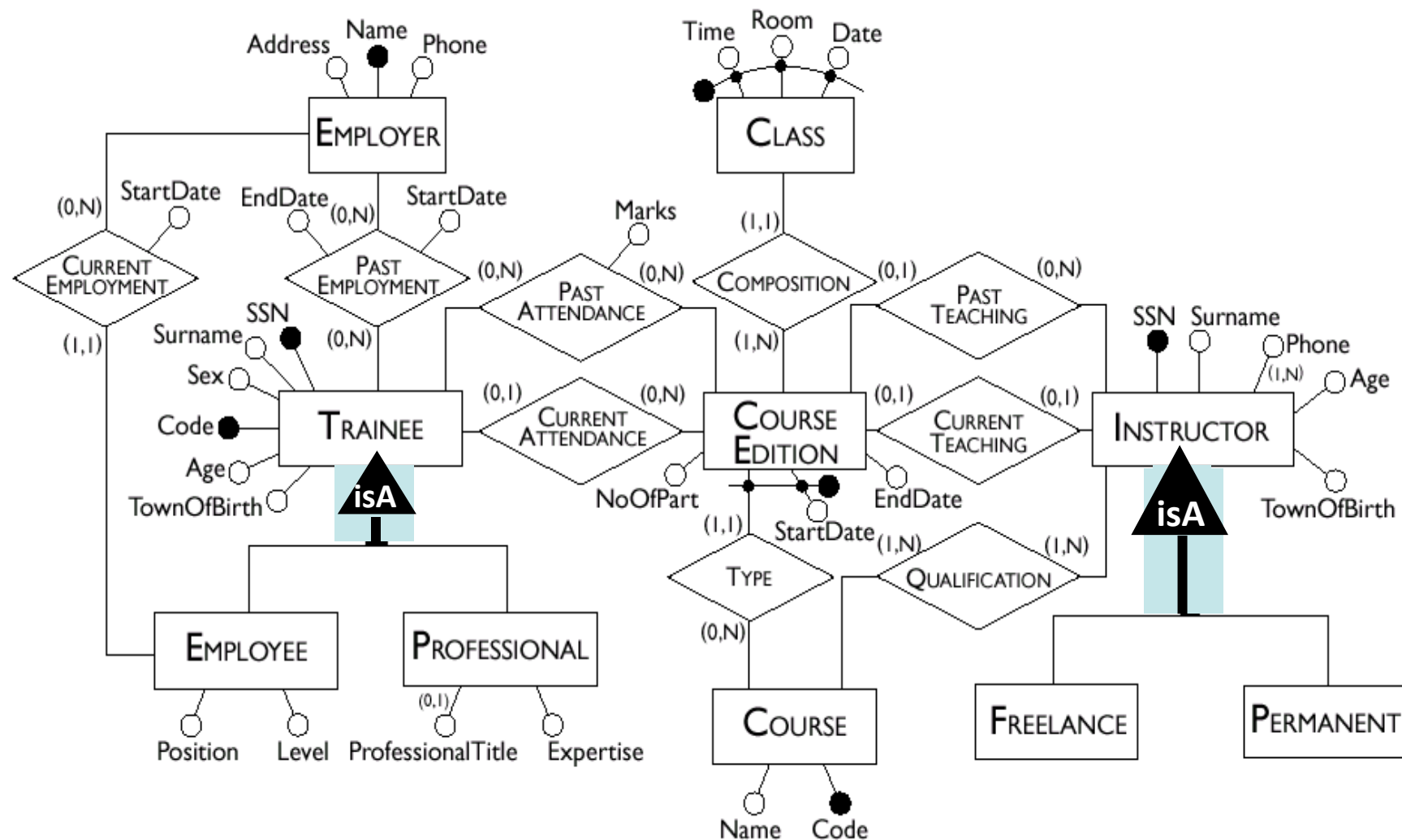
Term	Description	Synonym	Links
Trainee	Participant in a course. Can be an employee or self-employed.	Participant	Course, Company
Instructor	Course tutor. Can be freelance.	Tutor	Course
Course	Course offered. Can have various editions.	Seminar	Instructor, Trainee
Company	Company by which a trainee is employed or has been employed.		Trainee

More Annotations: Attributes

We wish to create a database for a company that runs training courses. For this, we must store data about *trainees* and *instructors*. For each *course participant* (about 5,000), identified by a code, we want to store her *social security number, surname, age, sex, place of birth, employer's name, address and telephone number, previous employers (and periods employed)*, courses attended (there are about 200 courses) and the final assessment for each course. We need also to represent *seminars* that each participant is attending at present and, *for each day, the places and times the classes are held*.

Each *course* has a code and a title and any course can be given any number of times. Each time a particular course is given, we will call it an 'edition' of the course. *For each edition, we represent the start date, the end date, and the number of participants*. If a trainee is self-employed, we need to know her area of expertise, and, if appropriate, her title. For somebody who works for a company, we store the level and position held. For each *instructor* (about 300), we will show *the surname, age, place of birth, the edition of the course taught*, those taught in the past and the courses that the *tutor* is qualified to teach. All the instructors' telephone numbers are also stored. An instructor can be permanently employed by the training company or freelance.

... the E/R model result



FROM E/R MODEL TO DATABASE SCHEMA

(Relational) Database Design

- Given a **conceptual schema** (ER, but could also be UML), generate a **logical (relational) schema**
- It is helpful to divide the design into two steps:
 - *Restructuring of the ER schema*, based on criteria for the *optimization* of the schema
 - *Translation into the logical model*, based on the features of the logical model (in our case, the relational model)

1. RESTRUCTURING AN E/R MODEL

Restructuring Overview

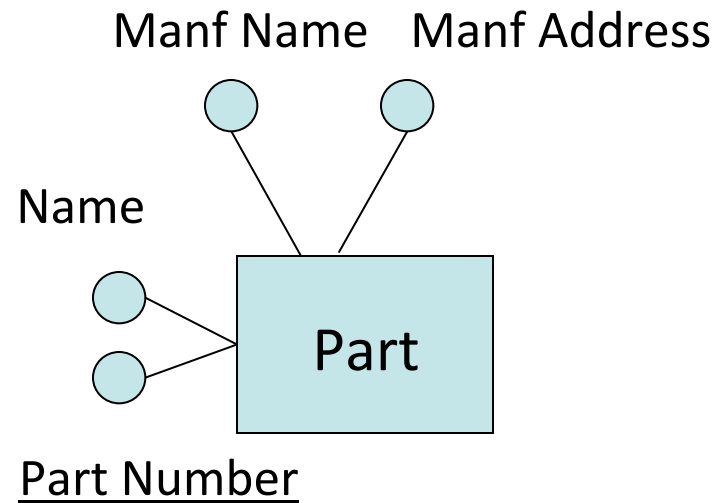
Input: E/R Schema

Output: Restructured E/R Schema

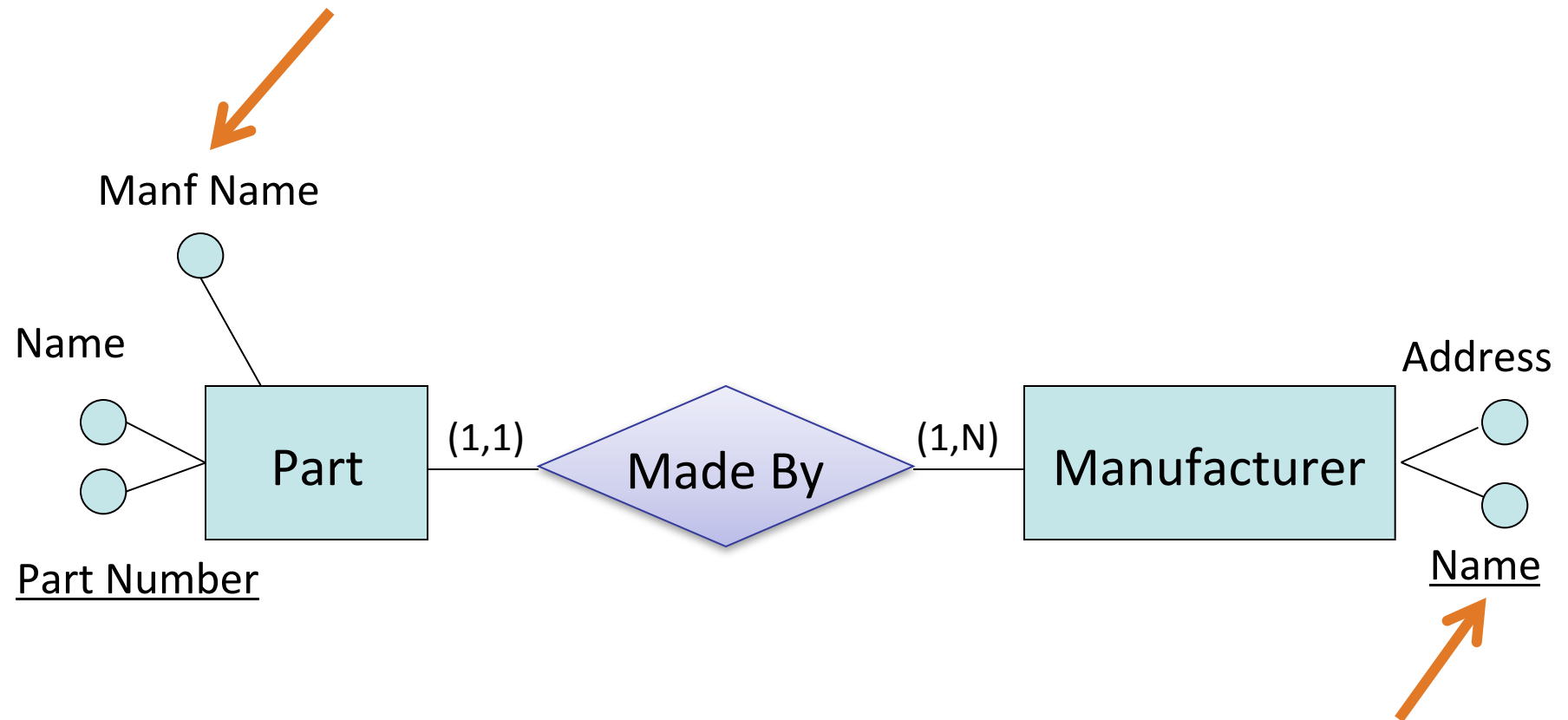
Restructuring includes:

- Analysis of **redundancies**
- Choosing **entity** set vs **attribute**
- Limiting the use of *weak* entity sets
- Selection of **keys**
- Creating entity sets to replace **attributes** with cardinality *greater than one*

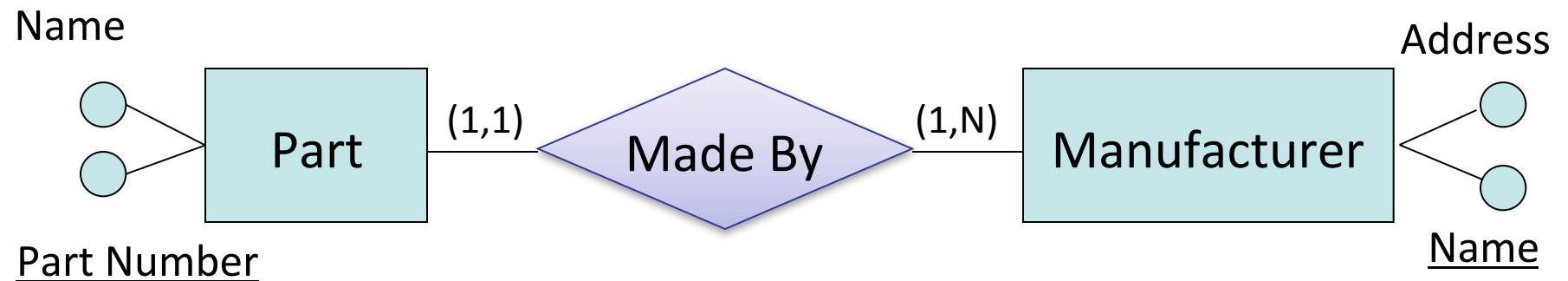
Example: redundancy



Example: redundancy



Example: no redundancy

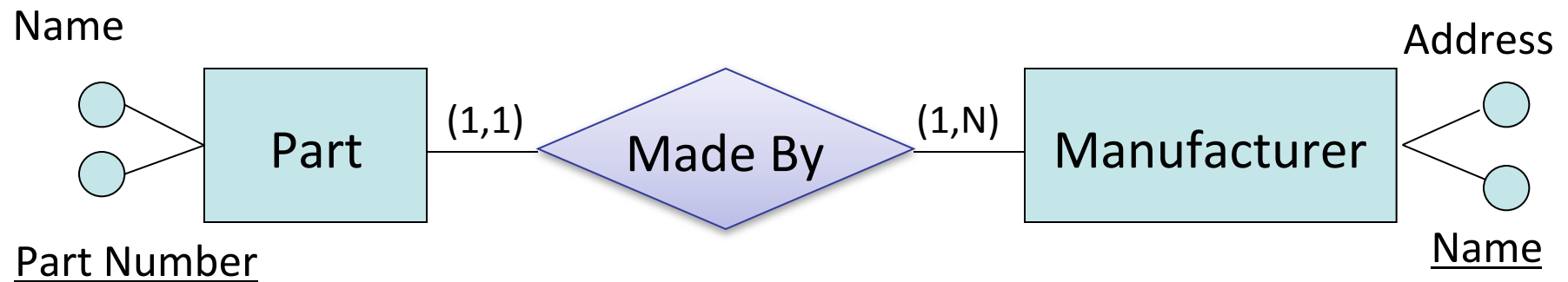


Entity Sets Versus Attributes

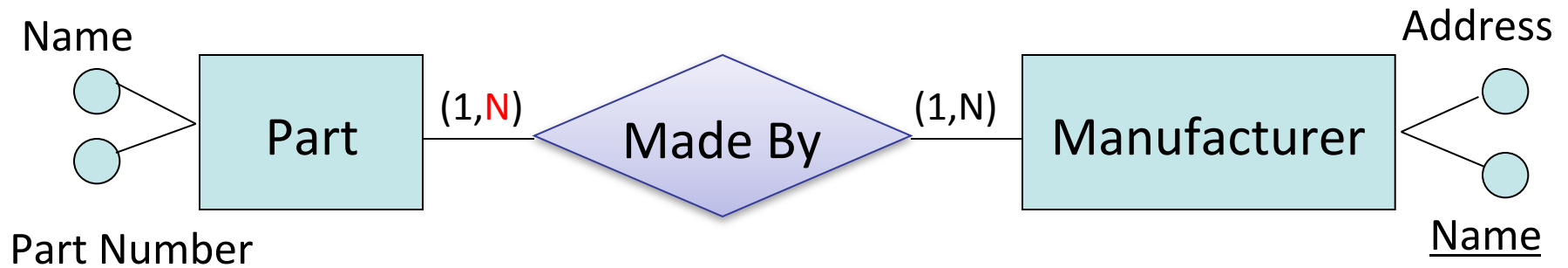
- An entity set should satisfy at least one of the following conditions:
 - a) It is more than the name of something; it has at least one **nonkey attribute**.
 - or
 - b) It is the “**many**” in a many-one or many-many relationship.
- Rules of thumb
 - A “thing” in its own right => Entity Set
 - A “detail” about some other “thing” => Attribute
 - A “detail” correlated among many “things” => Entity Set

Really this is just about avoiding redundancy

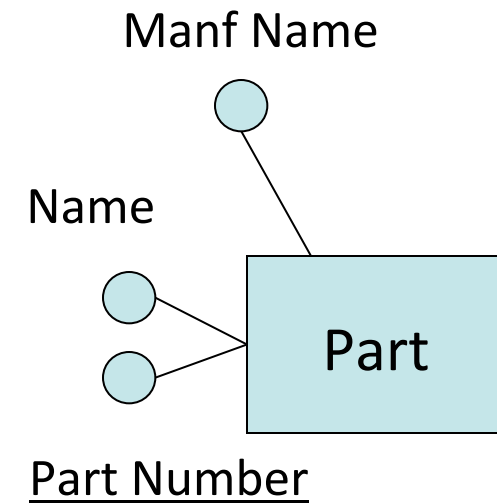
E.S. vs. attributes: examples



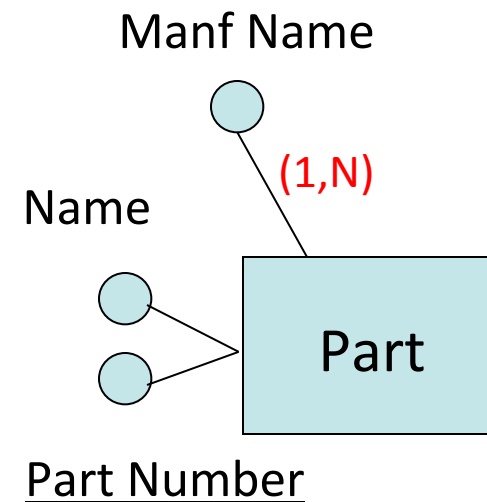
Domain fact change: **A part can have more than one manufacturer ...**



E.S. vs. attributes: examples



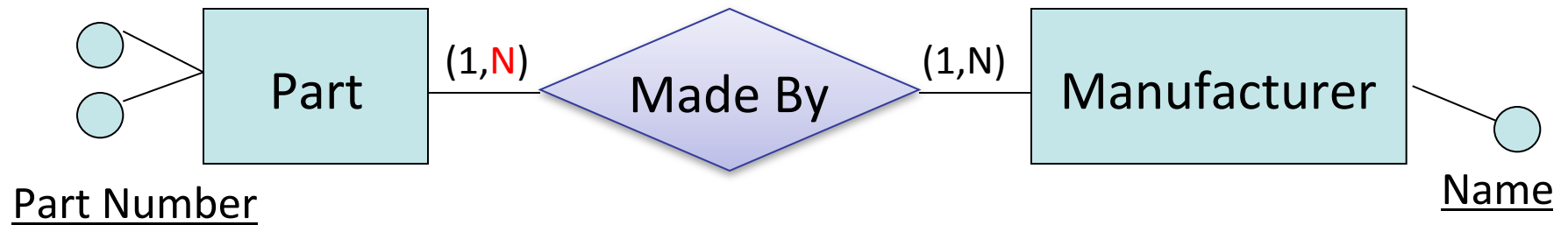
Domain fact change: **A part can have more than one manufacturer ...**



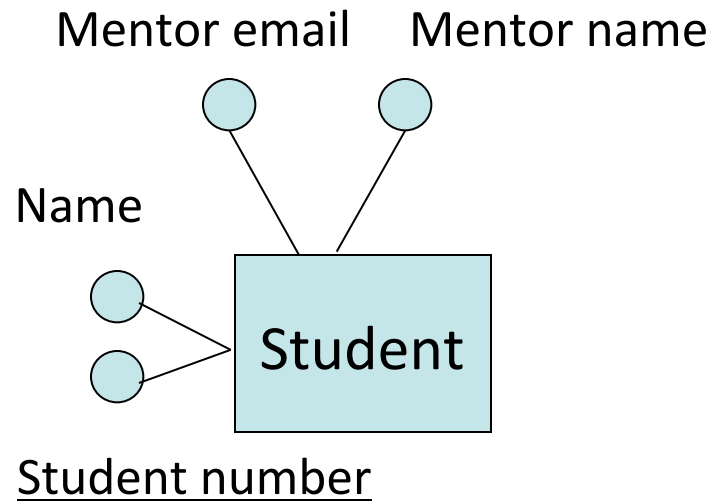
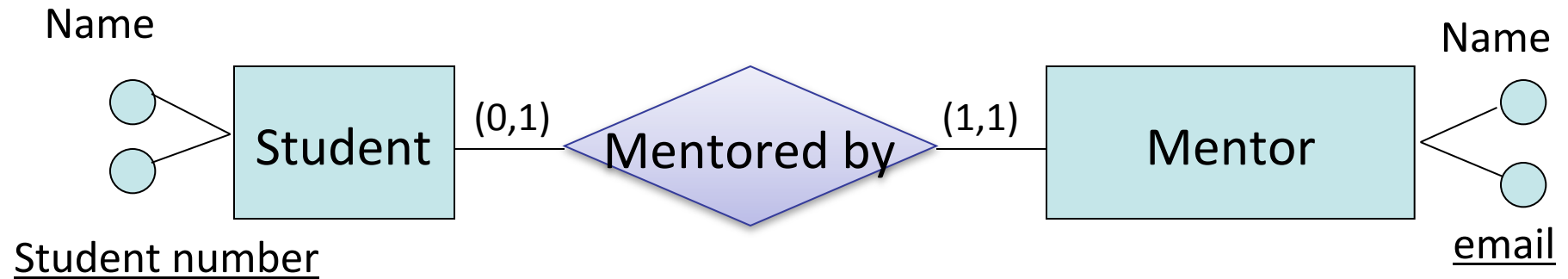
E.S. vs. attributes: examples

1. A part can have more than one manufacturer
2. No *address* attribute

Name

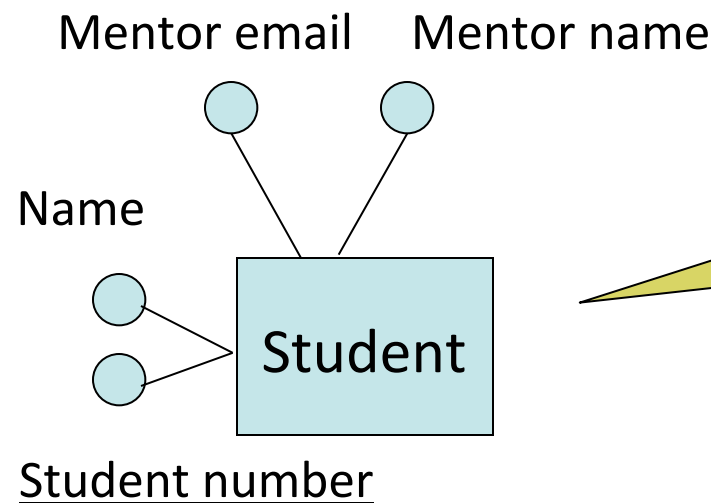
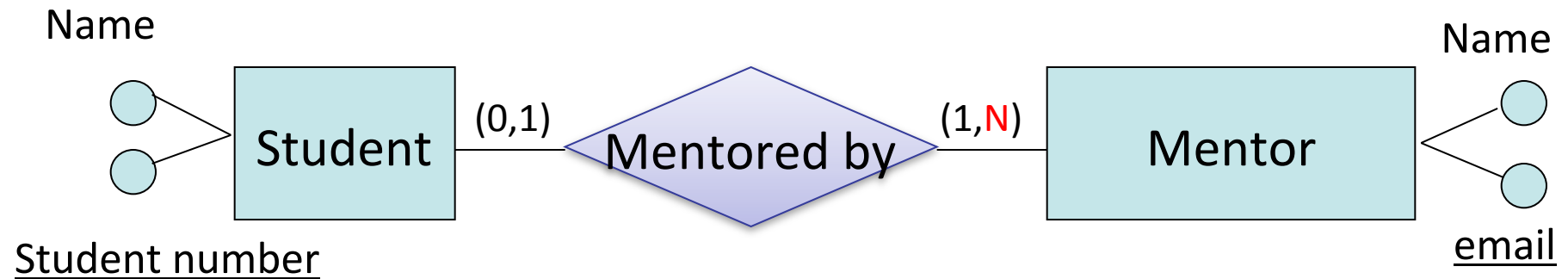


E.S. vs. attributes: examples



E.S. vs. attributes: examples

Domain fact change: **A mentor can have more than one mentee ...**



What if students have **NO** mentors?

we lose mentor information entirely..

When to use **weak** entity sets?

- The usual reason is that there is no global authority capable of creating **unique ID's**
- **Example:** global student repository
 - it is unlikely that there could be an agreement to assign unique student numbers across all students in the world

Don't Overuse Weak Entity Sets

- Beginning database designers often doubt that anything could be a key by itself
 - They make all entity sets weak, supported by all other entity sets to which they are linked
- It is usually better to create unique IDs
 - Social insurance number, automobile VIN, etc.
 - Useful for many reasons (next slide)

Selecting a Primary Key

- Every relation must have a **primary key**
- The criteria for this decision are as follows:
 - Attributes with **null** values **cannot** form primary keys
 - **One/few** attributes is preferable to many attributes
 - **Internal** keys preferable to external ones
(weak entities depend for their existence on other entities)

Keeping keys simple



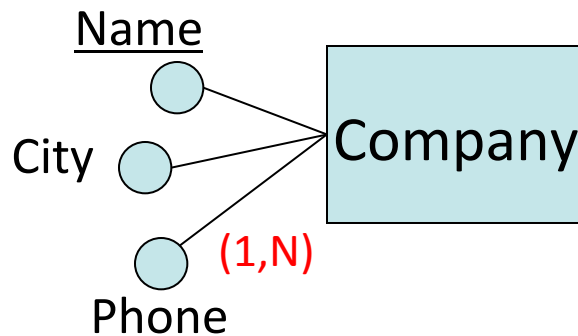
Multi-attribute and/or string keys...

- ... waste space (are redundant)
 - e.g. Movies(title, year, ...): 2 string attributes, ~16 bytes
 - Number of movies ever made $\ll 2^{32}$ (4 bytes)
 - => Integer **movieID** key saves 75% space and a lot of typing
- ... break encapsulation
 - e.g. Patient(firstName, lastName, phone, ...)
 - Security/privacy hole
 - => Integer **patientID** prevents information leaks
- ... are brittle (nasty interaction of above two points)
 - Name or phone number change?
 - Patient with no phone? Two movies with same title and year?
 - => **Internal ID** always exists, immutable, unique

Also: computers are really good at integers...

Attributes with cardinality > 1

- The relational model doesn't allow **multi-valued attributes**. We must convert these to **entity sets**.



2. TRANSLATING AN E/R MODEL INTO A DB SCHEMA

Translation into a Logical Schema

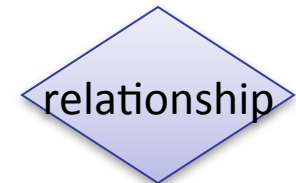
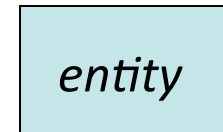
Input: E/R Schema

Output: Relational Schema

- Starting from an E/R schema, an equivalent relational schema is constructed
 - “**equivalent**”: a schema capable of representing the same information
- A good translation should also:
 - not allow **redundancy**
 - not invite unnecessary **null** values

The general idea

- Each **entity set** becomes a *relation*.
Its **attributes** are
 - the attributes of the entity set.
- Each **relationship** becomes a *relation*.
It's **attributes** are
 - the **keys** of the **entity sets** that it connects, plus
 - the **attributes** of the **relationship** itself.



Many-to-Many Binary Relationships

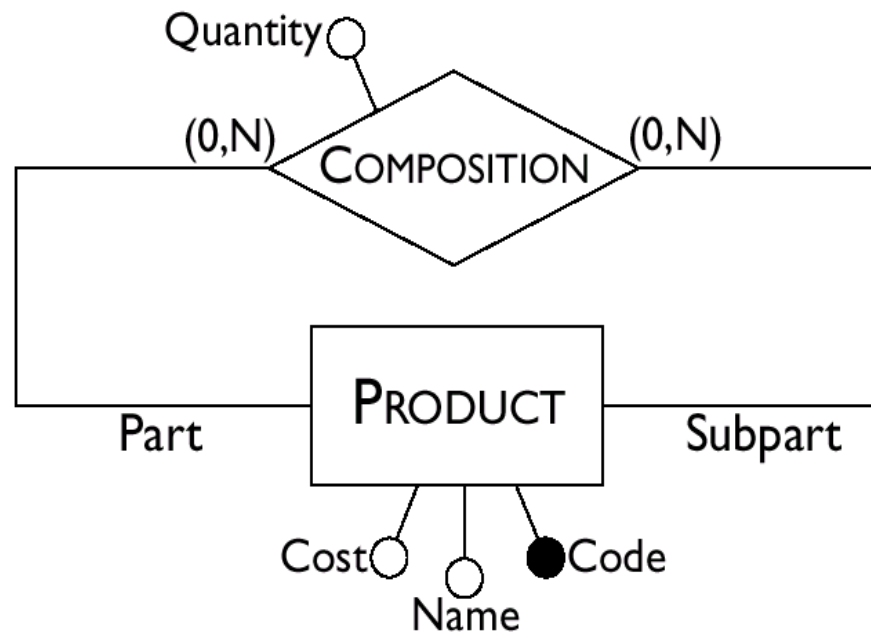


Employee(Number, Surname, Salary)

Project(Code, Name, Budget)

Participation(Number, Code, StartDate)

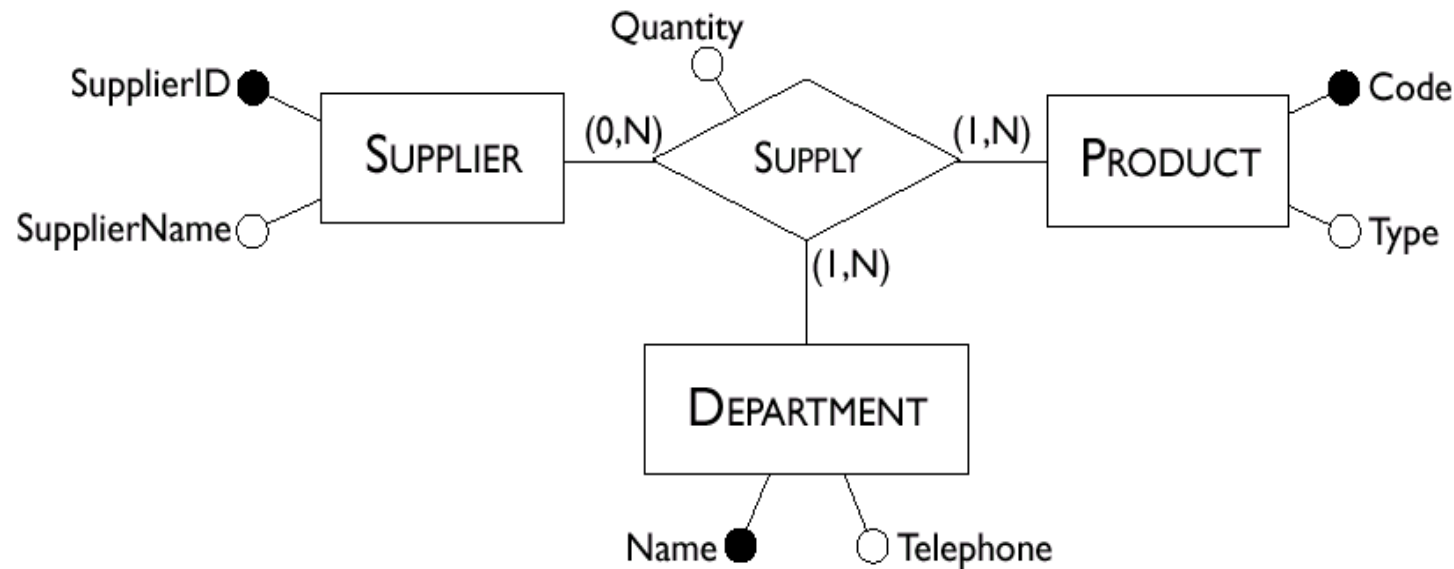
Many-to-Many Recursive Relationships



Product(Code, Name, Cost)

Composition(Part, SubPart, Quantity)

Ternary Many-to-Many Relationships



Supplier(SupplierID, SupplierName)

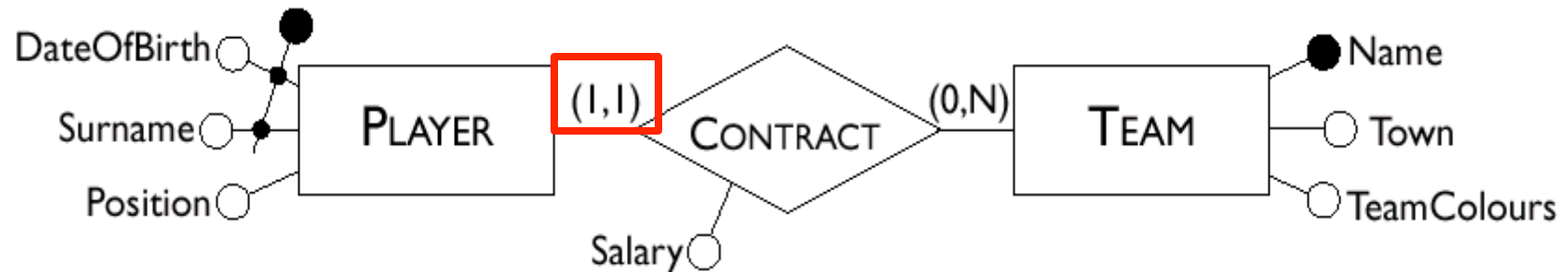
Product(Code, Type)

Department(Name, Telephone)

Supply(Supplier, Product, Department, Quantity)



One-to-Many Relationships with mandatory participation for one



Player(Surname, DateOfBirth, Position)

Team(Name, Town, TeamColours)

~~Contract(PlayerSurname, PlayerDateOfBirth, Team, Salary)~~

Contract(PlayerSurname, PlayerDateOfBirth, Team, Salary)

OR

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

Team(Name, Town, TeamColours)

(*"Messi"*, *"1987"*, *"Barcelona"*, 1234)
(*"Messi"*, *"1987"*, *"Real Madrid"*, 2334)





One-to-One Relationships

with **mandatory** participation for both



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

Department(Name, Telephone, Branch)

Or

Head(Number, Name, Salary, StartDate)

Department(Name, Telephone, HeadNumber, Branch)



One-to-One Relationships with **optional** participation for one



Employee(Number, Name, Salary)

Department(Name, Telephone, Branch, HeadNumber, *StartDate*)

Or, if both entities are optional..?

Employee(Number, Name, Salary)

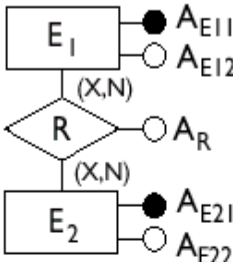
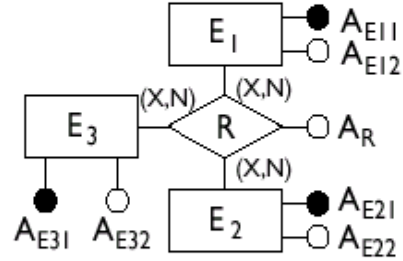
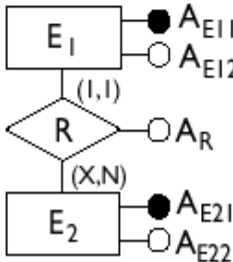
Department(Name, Telephone, Branch)

Management(HeadNumber, DeptName, StartDate)

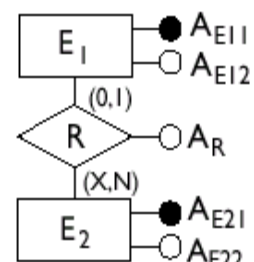
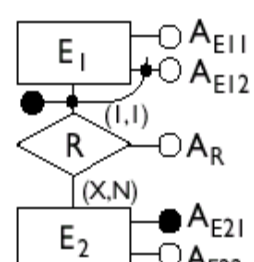
Summary of Types of Relationship

- many-to-many (binary or ternary)
- one-to-many
 - mandatory: (1,1) on the “one” side
 - optional: (0,1) on the “one” side
- one-to-one
 - both mandatory: (1,1) on both sides
 - one mandatory, one optional:
(1,1) on one side and (0,1) on other side
 - both optional: (0,1) on both sides

Summary of Transformation Rules

Type	Initial schema	Possible translation
Binary many-to-many relationship		$E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $R(\underline{A_{E11}}, \underline{A_{E21}}, A_R)$
Ternary many-to-many relationship		$E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $E_3(\underline{A_{E31}}, A_{E32})$ $R(\underline{A_{E11}}, \underline{A_{E21}}, \underline{A_{E31}}, A_R)$
One-to-many relationship with mandatory participation		$E_1(\underline{A_{E11}}, A_{E12}, A_{E21}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$

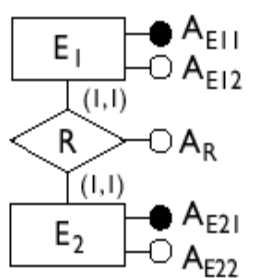
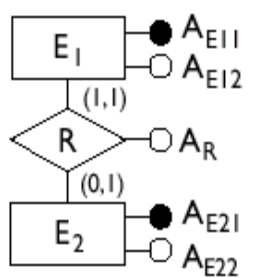
...More Rules...

Type	Initial schema	Possible translation
One-to-many relationship with optional participation		$E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $R(\underline{A_{E11}}, \underline{A_{E21}}, A_R)$ <p>Alternatively:</p> $E_1(\underline{A_{E11}}, A_{E12}, A_{E21}^*, A_R^*)$ $E_2(\underline{A_{E21}}, A_{E22})$
Relationship with external identifiers		$E_1(\underline{A_{E12}}, \underline{A_{E21}}, A_{E11}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$

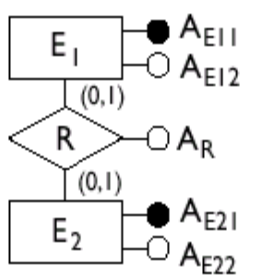
A_{E21} shouldn't be key (typo)

* Indicates that nulls are allowed

...Even More Rules...

Type	Initial schema	Possible translation
One-to-one relationship with mandatory participation for both entities		$E_1(\underline{A_{E11}}, A_{E12}, \underline{A_{E21}}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$ <p>Alternatively:</p> $E_2(\underline{A_{E21}}, A_{E22}, \underline{A_{E11}}, A_R)$ $E_1(\underline{A_{E11}}, A_{E12})$
One-to-one relationship with optional participation for one entity		$E_1(\underline{A_{E11}}, A_{E12}, \underline{A_{E21}}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$

...and the Last One...

Type	Initial schema	Possible translation
One-to-one relationship with optional participation for both entities		$\underline{E_1(A_{E11}, A_{E12})}$ $\underline{E_2(A_{E21}, A_{E22}, A_{E11}^*, A_R^*)}$ <p>Alternatively:</p> $\underline{E_1(A_{E11}, A_{E12}, A_{E21}^*, A_R^*)}$ $\underline{E_2(A_{E21}, A_{E22})}$ <p>Alternatively:</p> $\underline{E_1(A_{E11}, A_{E12})}$ $\underline{E_2(A_{E21}, A_{E22})}$ $\underline{R(A_{E11}, A_{E21}, A_R)}$

Will the schema be “good”?

- If we use this process, will the schema we get be a good one?
- The process should ensure that there is **no redundancy**.
- But only with respect to what the **E/R diagram** represents.
- Crucial thing we are missing: **functional dependencies**.
(We only have keys, not other FDs.)
- So we still need **FD theory**.

Redundancy can be *desirable*

- We've talked a lot about avoiding redundancy.
- It's a **disadvantage**: because of larger storage requirements, (but, usually at negligible cost) and the necessity to carry out additional operations in order to keep the derived data consistent
- But it's also an **advantage**: a reduction in the number of accesses necessary to obtain derived information
- The decision to maintain or eliminate a redundancy is made by comparing the **cost of operations** that involve the redundant information and **the storage needed**, in the case of presence or absence of redundancy.

Performance analysis is required to decide about redundancy