FROM E/R MODEL TO DATABASE SCHEMA

Steps

- *Restructure* the ER schema to improve it, based on criteria
- *Translate* the schema into the relational model
- *Add missing constraints* to the schema

1. RESTRUCTURING AN E/R MODEL

Restructuring Overview

Input: E/R Schema

Output: Restructured E/R Schema

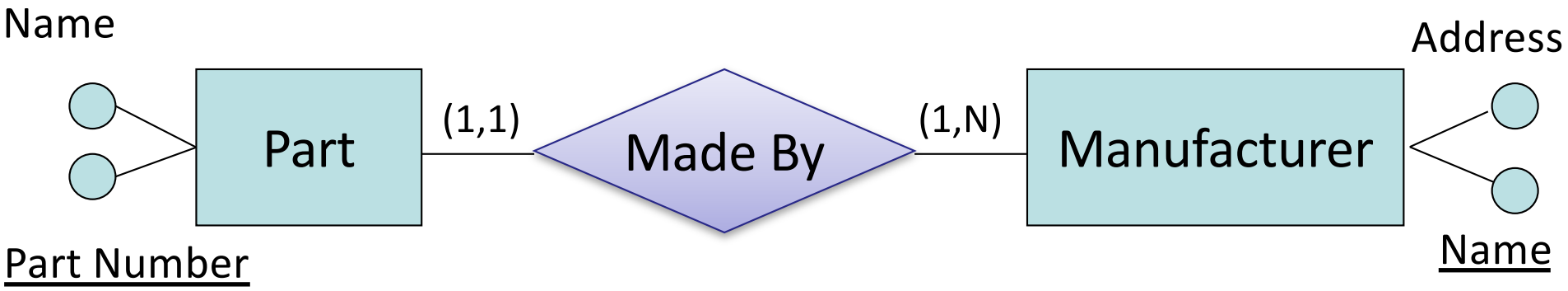
It includes (not necessarily in this order):

a. Analysis of redundancies
b. Choosing entity set vs attribute
c. Limiting the use of weak entity sets
d. Settling on keys
e. Creating entity sets to replace attributes with cardinality greater than one

1a. Analysis of redundancies

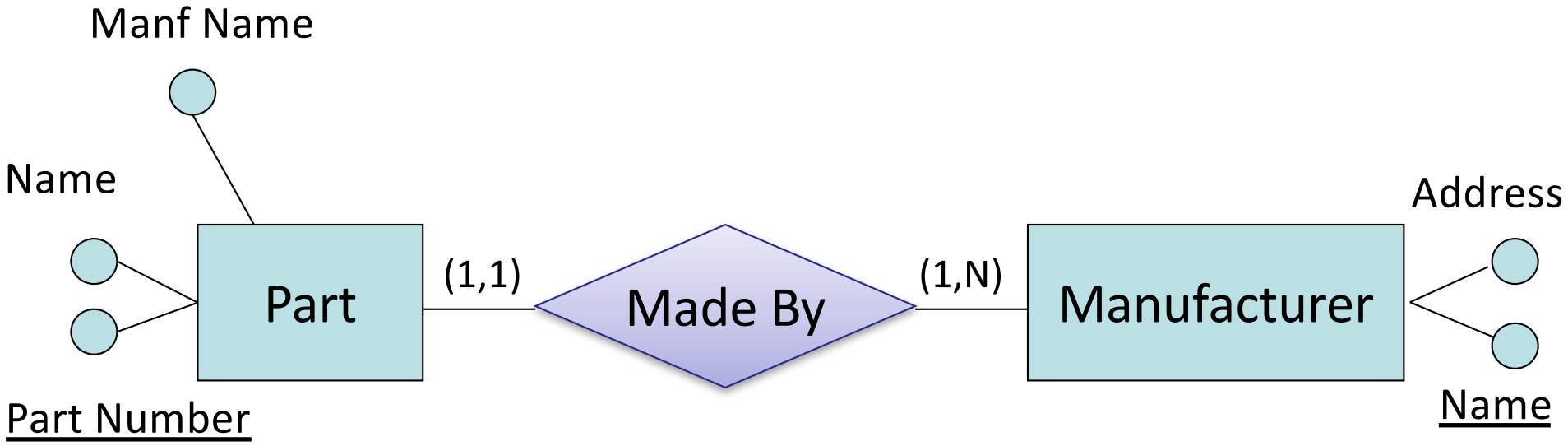
Example: no redundancy

It is not redundant to have Name twice.



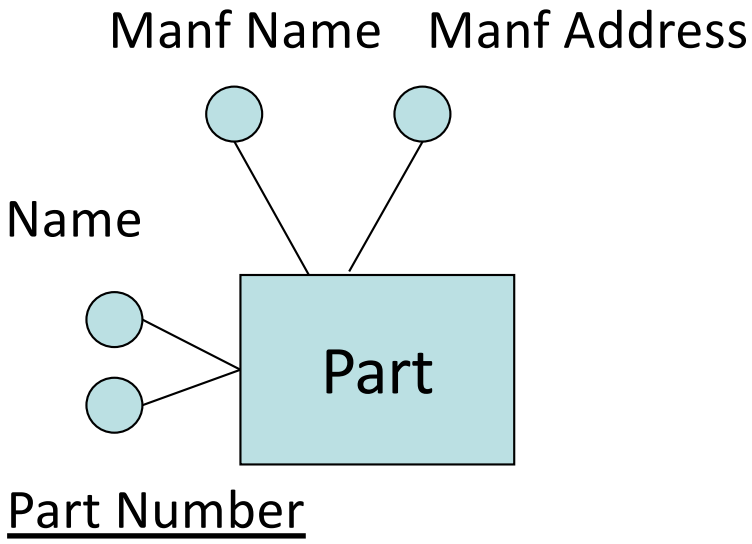
Example: redundancy

What is redundant here?



Example: redundancy

What is redundant here?



1b. Entity sets vs attributes

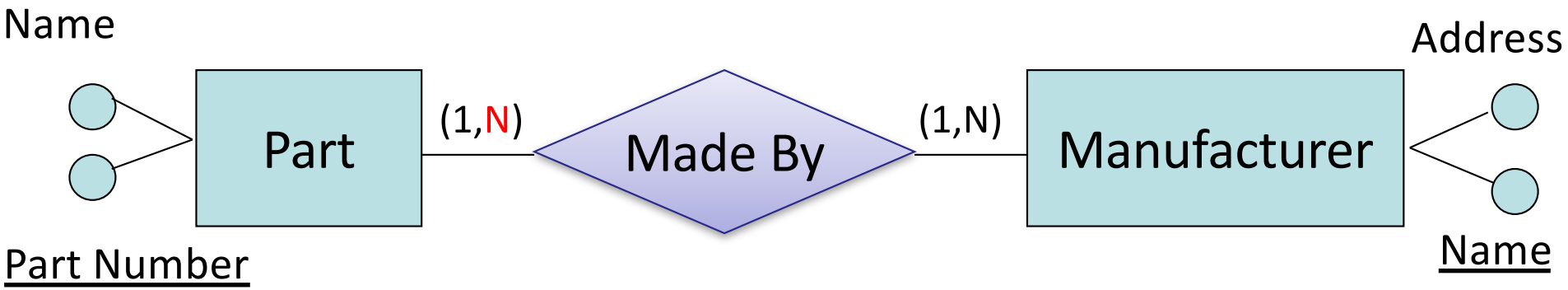
Overall, Prefer Attributes

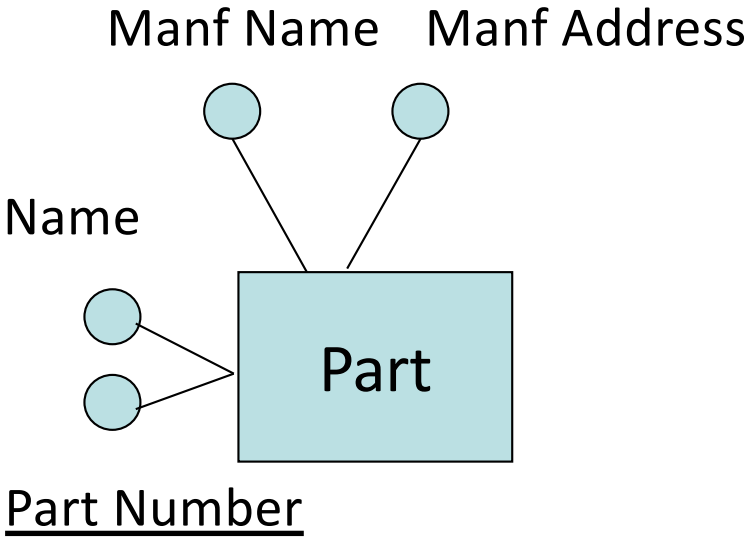
- An entity set should satisfy at least one of the following conditions:
  - It is more than the name of something; it has at least one non-key attribute, or
  - It is the “many” in a many-one or many-many relationship.
- Rules of thumb
  - A “thing” in its own right => Entity Set
  - A “detail” about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

This is just about avoiding redundancy

E.S. vs. attributes: examples

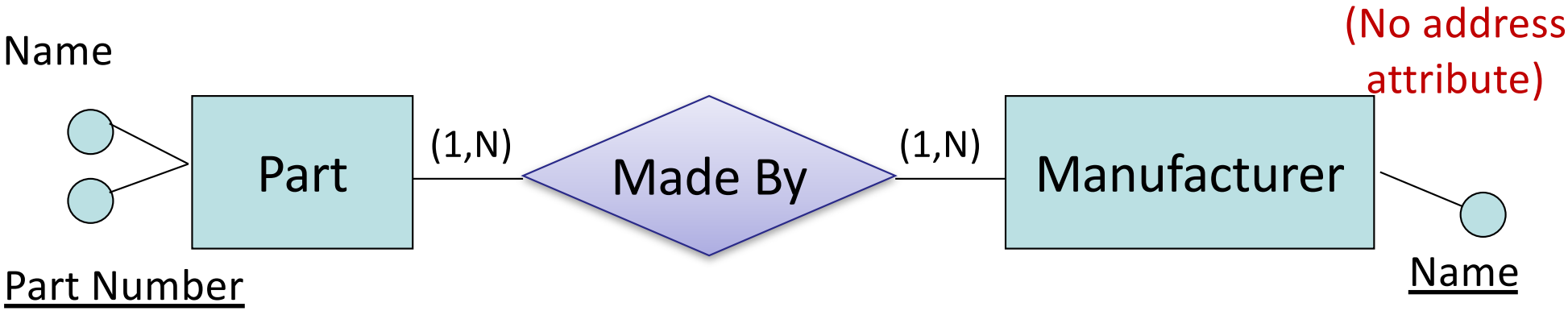
Domain fact change: A part can have more than one manufacturer …

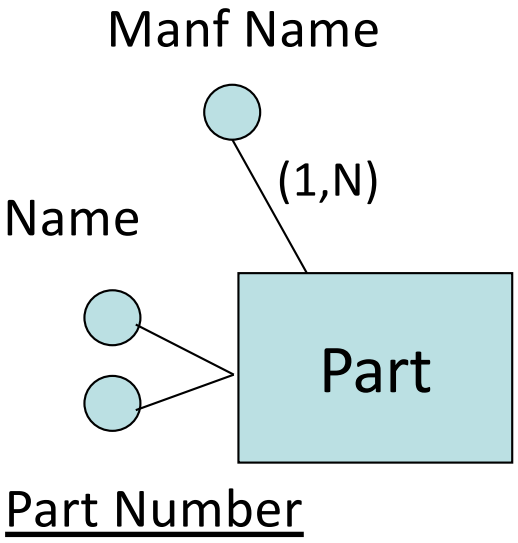




E.S. vs. attributes: examples

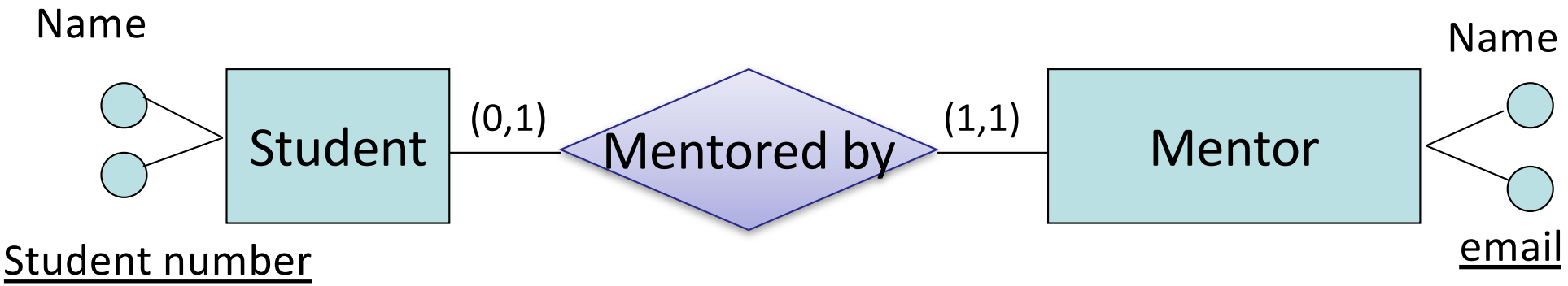
Domain fact change: Not representing Manufacturer address …

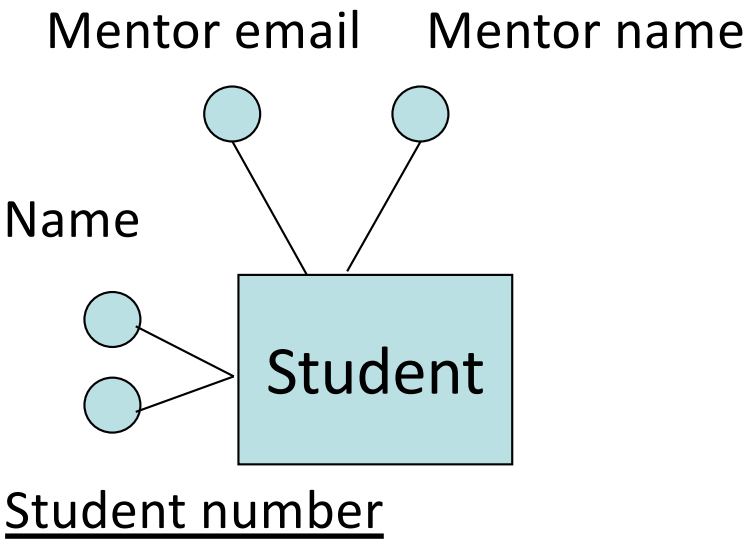




E.S. vs. attributes: examples

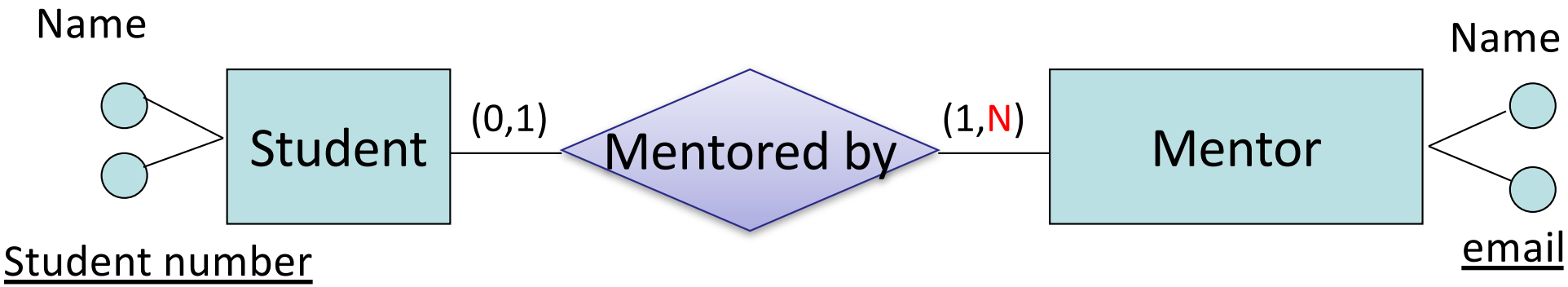
New domain

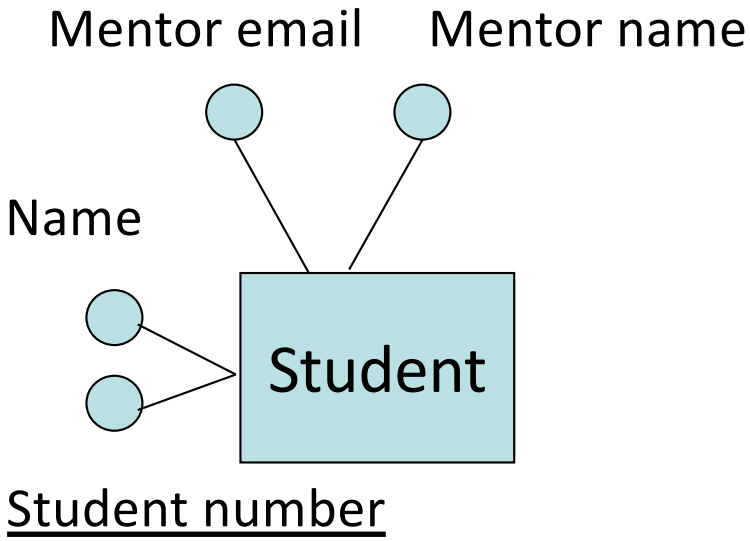




E.S. vs. attributes: examples

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





1c. Limiting weak entity sets

When to use weak entity sets?

- The usual reason is that there is no global authority capable of creating unique ID's
- Example: it is unlikely that there could be an agreement to assign unique student numbers across all students in the world
- It may seem the only choice is a weak entity set.
- It is usually better to create unique IDs
  - Social insurance number, automobile VIN, etc.

1d. Settling on keys

Settling on keys

- Make sure that every entity set has a key
- Keep in mind:
  - Attributes with null values cannot be part of primary keys
  - Internal keys are preferable to external ones
  - A key that is used by many operations to access instances of an entity is preferable to others
  - A key with one/few attributes is preferrable
  - An integer key is preferrable

Avoid multi-attribute and string keys

- They are wasteful
  - e.g. Movies(<u>title</u>, <u>year</u>, …): 2 attributes in key, requires ~16 bytes
  - Number of movies ever made (<< 2^{32}) can be distinguished with 4 bytes

  => Having an integer movieID key saves 75% space and a lot of typing
- They break encapsulation
  - e.g. Patient(<u>firstName</u>, <u>lastName</u>, <u>phone</u>, …)
  - Security/privacy hole

  => Integer patientID prevents information leaks
- They are brittle (nasty interaction of above two points)
  - Name or phone number change? Parent and child with same name?
  - Patient with no phone? Two movies with same title and year?

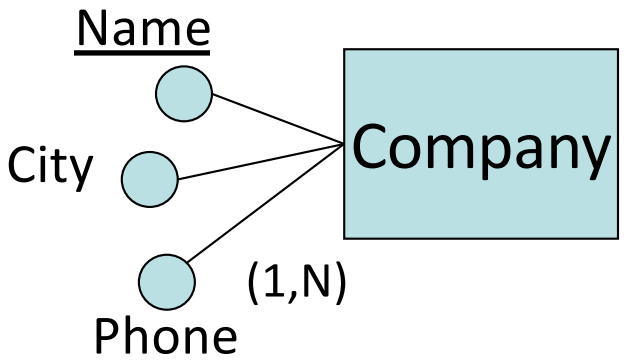
  => Internal ID always exist, are immutable, unique

Also: computers are really good at integers!

1e. Creating entity sets to replace attributes with cardinality greater than one

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

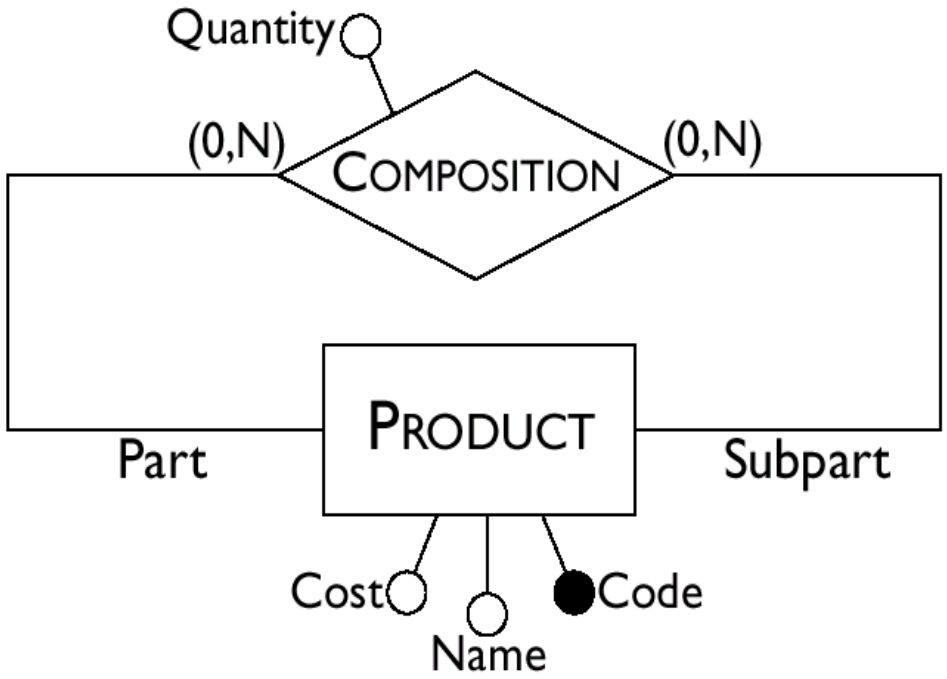
- Starting from an E/R schema, an equivalent relational schema is constructed
  - "equivalent": a schema capable of representing the same information
- A good translation should also:
  - not allow redundancy
  - not invite unnecessary null values

The general idea

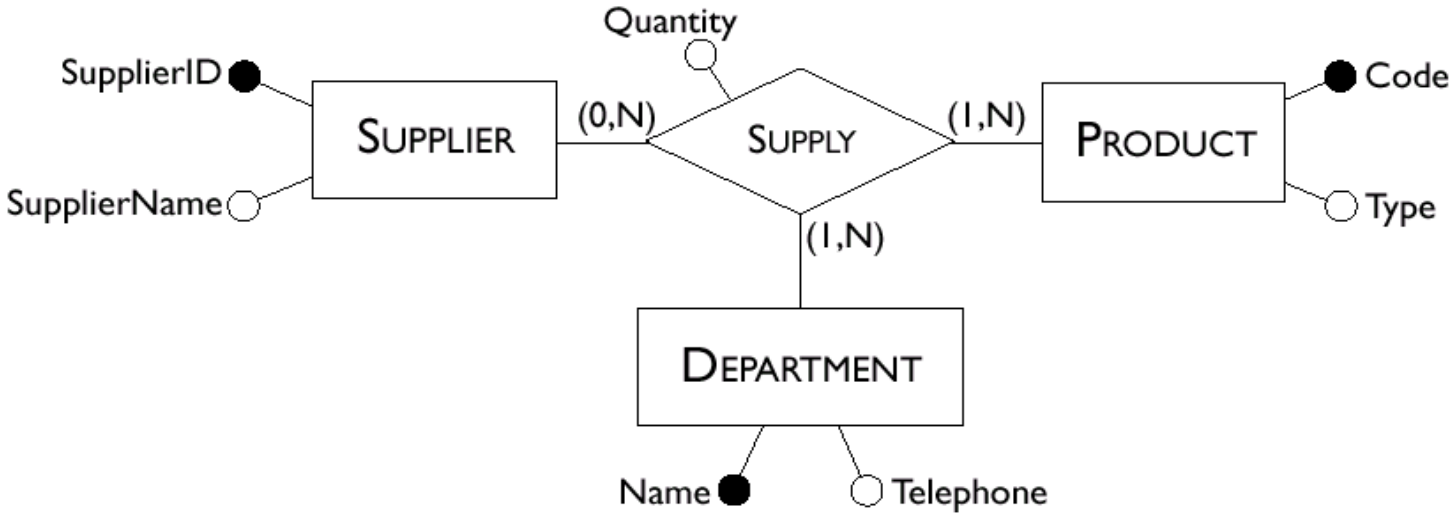
- Each entity set becomes a relation. Its attributes are
  - the attributes of the entity set.
- Each relationship becomes a relation. Its attributes are
  - the keys of the entity sets that it connects, plus
  - the attributes of the relationship itself.
- We'll see opportunities to simplify.

Many-to-Many Binary Relationships

Many-to-Many Recursive Relationships



Many-to-Many Ternary Relationships

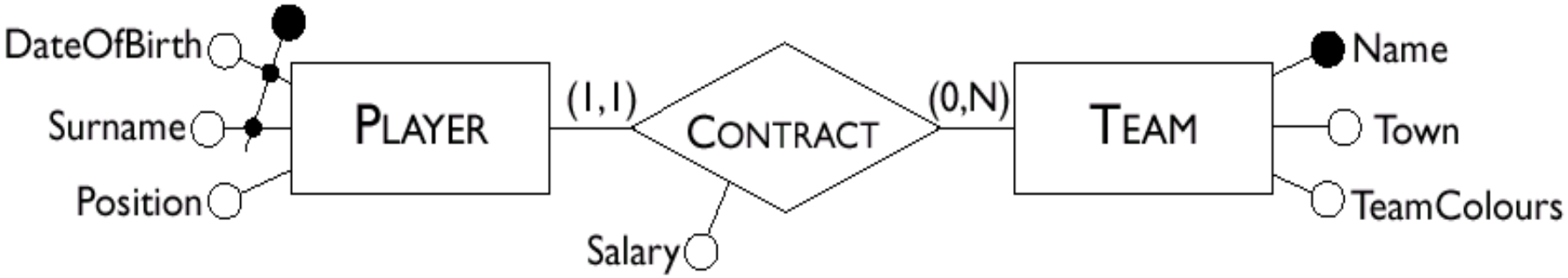


Simplifications

- These straight translations are acceptable.
- But we can often simplify.

One-to-Many Relationships

with mandatory participation on the “one” side



One-to-One Relationships

with mandatory participation for both

One-to-One Relationships

with mandatory participation for both

The standard translation has 3 relations (what is key of management)?

One-to-One Relationships

with optional participation for one

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

3. FINAL CONSIDERATIONS

During or after the translation, we should be sure to:

- Express the foreign-key constraints
- Consider better names for the referring attributes
- Express the ”max 1” constraints
- Express the “min 1” constraints

Names for Referring Attributes

Example:

Employee(<u>Number</u>, Surname, Salary)

Project(<u>Code</u>, Name, Budget)

Participation(<u>Number, Code</u>, StartDate)

Participation[Number] ⊆ Employee[Number]

Participation[Code] ⊆ Project[Code]

What about “max 1” constraints?

Our simplified v1 (with Management info moved over to Head):

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

Department(Name, Telephone, Branch)

Head[Department] ⊆ Department[Name]

Did we enforce that every head has at most one department?

What about “max 1” constraints?

Our simplified v1 (with Management info moved over to Head):

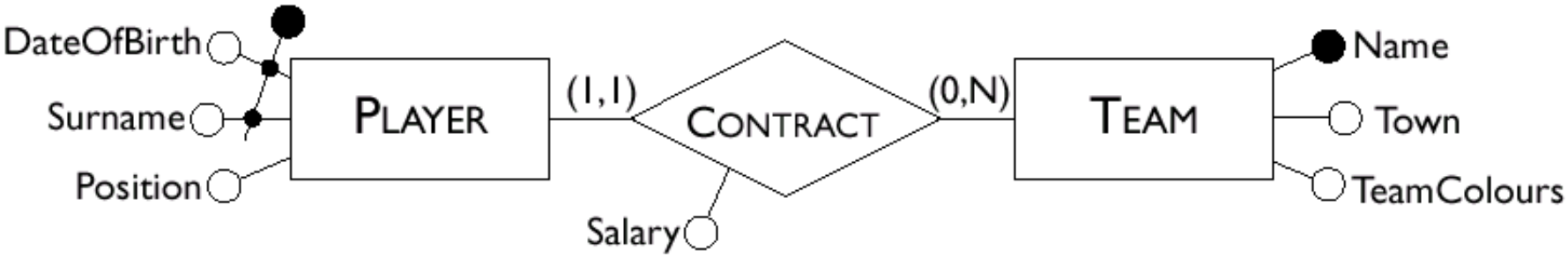
Head(Number, Name, Salary, Department, StartDate)

Department(Name, Telephone, Branch)

Head[Department] ⊆ Department[Name]

Did we enforce that every department has at most one head?

What about “min 1” constraints?

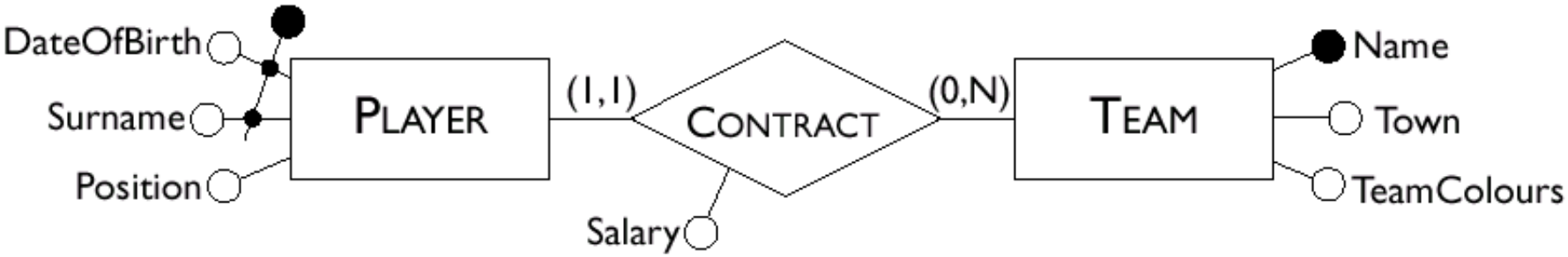


Our simplified translation:

Player(<u>Surname, DOB</u>, Position, TeamName, Salary)
Team(<u>Name</u>, Town, TeamColours)
Player[TeamName] ⊆Team[Name]

Did we enforce that every player must have a team?

What about “min 1” constraints?



Our standard (non simplified) translation:

Player(Surname, DOB, Position)
Team(Name, Town, TeamColours)
Contract(PlayerSurname, PlayerDOB, Team, Salary)
Contract[PlayerSurname, PlayerDOB]⊆Player[Surname, DOB]
Contract[Team] ⊆Team[Name]

Did we enforce that every player must have a team?

What about “min 1” constraints?

Our simplified translation:

Player(<u>Surname, DOB</u>, Position, TeamName, Salary)
Team(<u>Name</u>, Town, TeamColours)
Player[TeamName] ⊆Team[Name]

Does this enforce that every team must have a player?

What about “min 1” constraints?

In our translation from E/R to a relational schema, we expressed the constraints using relational notation.

But in SQL, only some of them can be written as foreign keys.

- They must refer to attributes that are primary key or unique.

Will the schema be “good”?

- If we use this process, will the schema we get be a good one?
- The process should ensure that (a) the data can be represented, (b) there is no redundancy, and (c) most constraints are enforced.
- But only with respect to what’s in the E/R diagram.
- Crucial thing we are missing: functional dependencies. (We only have keys, not other FDs.)
- So a good design process includes normalization at the end.

Redundancy can be *desirable*

- Disadvantages of redundancy:
  - More storage (but usually at negligible cost)
  - Additional operations to keep the data consistent
- Advantages of redundancy:
  - Speed: Fewer accesses necessary to obtain information
- So a good design process includes considering whether to allow some redundancy.
- How to decide to maintain or eliminate a redundancy? Examine:
  - the speedup in operations made possible by the redundant information
  - the relative frequency of those operations
  - the storage needed for the redundant informations