CSCC43 - Fall 2023

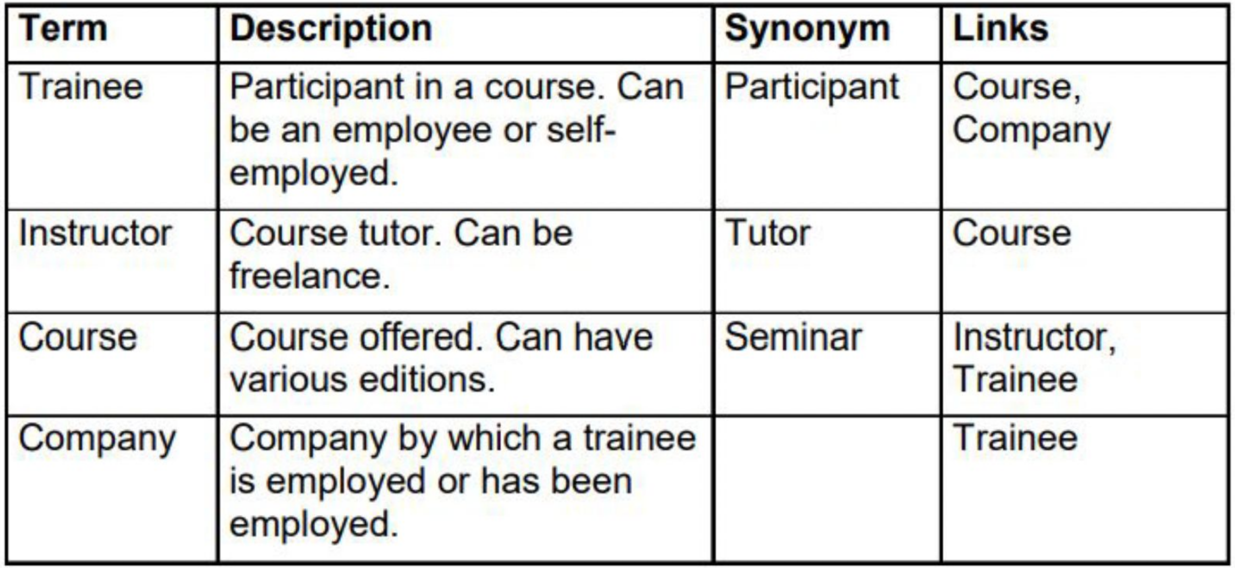
Translating an ER into DB Schema

From Real world to ER

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.

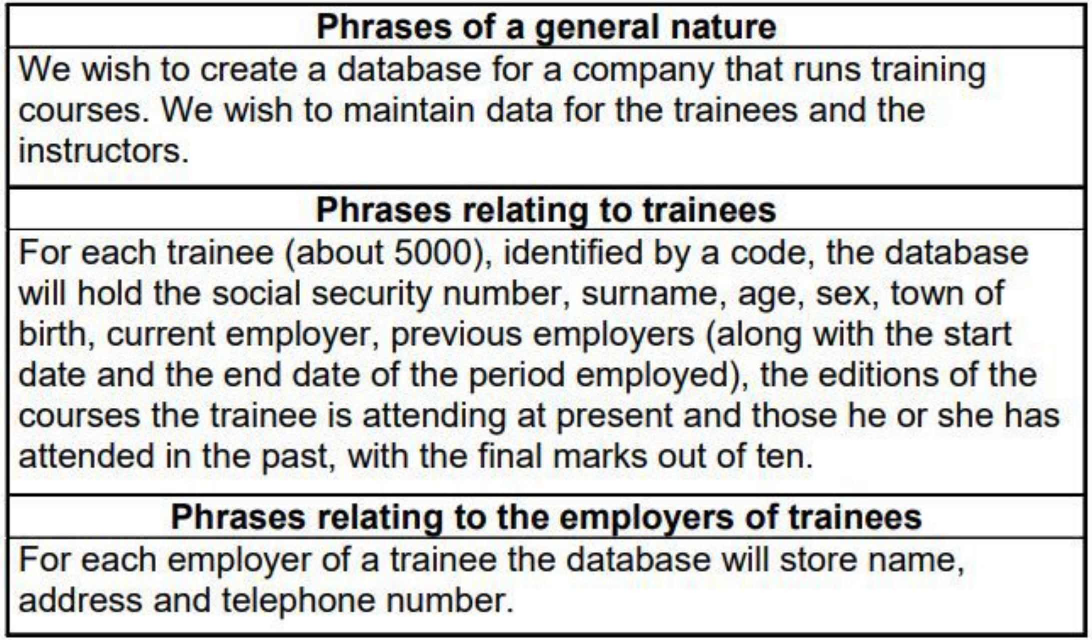
Each course has a code and a title and any course can be given any number of times. Each time a particular course is given, we will call it an 'edition' of the course. For each edition, we represent the start date, the end date, and the number of participants. If a trainee is self-employed, we need to 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

Glossary Example



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

Structuring the requirement



Phrases of a general nature
We wish to create a database for a company that runs training courses. We wish to maintain data for the trainees and the instructors.
Phrases relating to trainees
For each trainee (about 5000), identified by a code, the database will hold the social security number, surname, age, sex, town of birth, current employer, previous employers (along with the start date and the end date of the period employed), the editions of the courses the trainee is attending at present and those he or she has attended in the past, with the final marks out of ten.
Phrases relating to the employers of trainees
For each employer of a trainee the database will store name, address and telephone number.

Restructuring an E/R Model

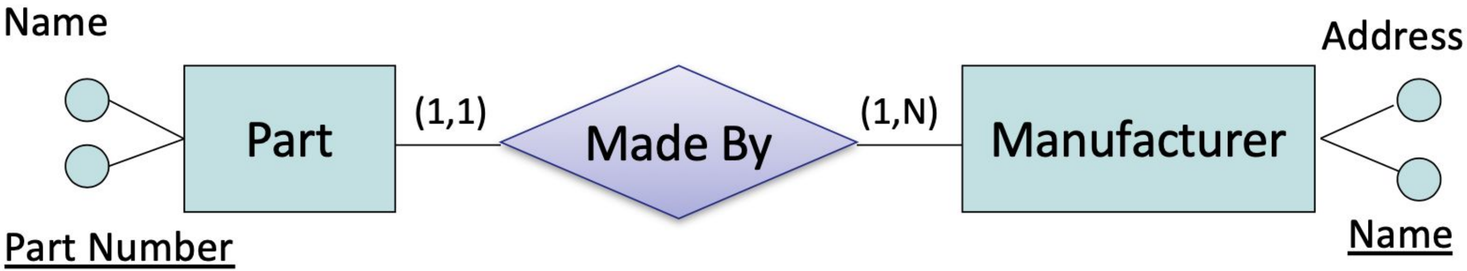
Input: E/R Schema

Output: Restructured E/R Schema

It includes (not necessarily in this order):

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

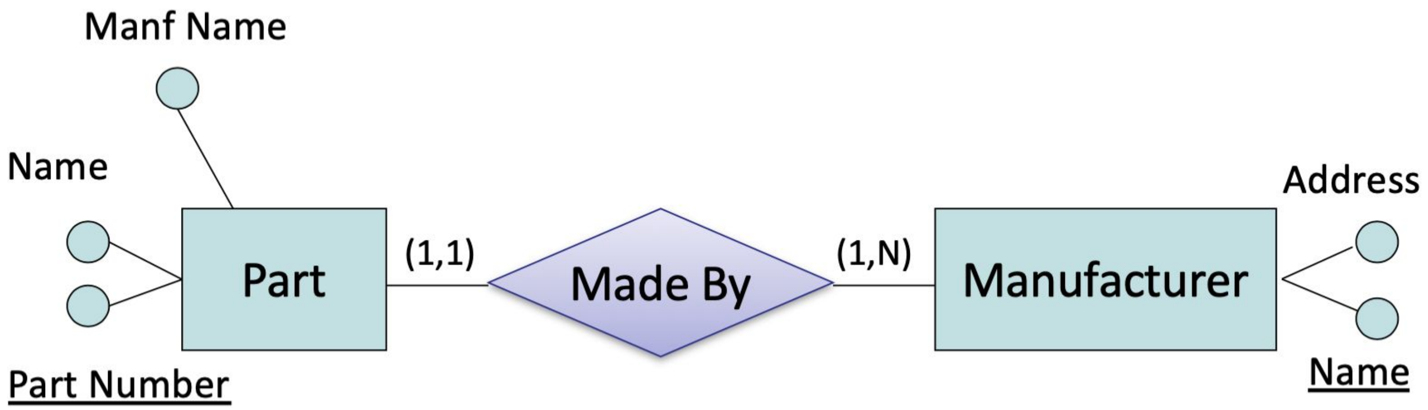
Analysis of Redundancies



Is having Name twice redundant ?

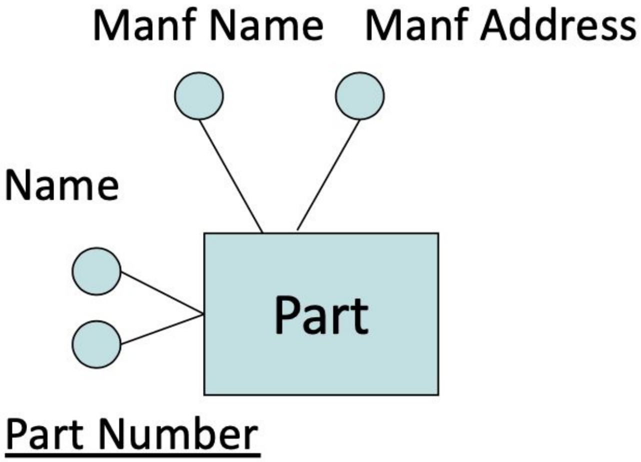
Example - Redundancy

What’s redundant here



Example Redundancy

What’s redundant here



Entity Set vs 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.

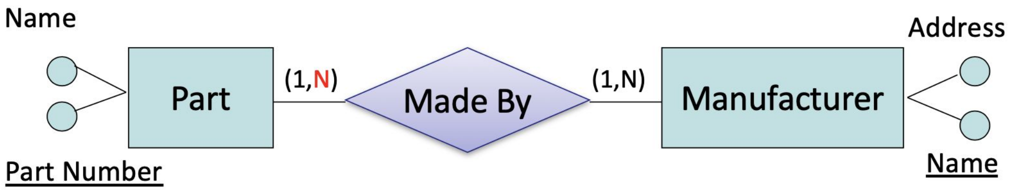
Rules of thumb

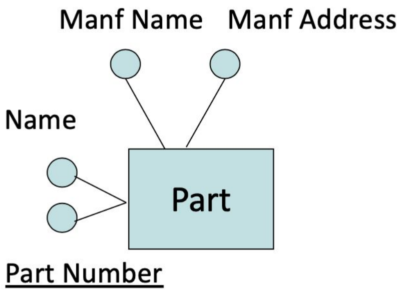
- A “thing”in its own right => Entity Set
- A “detail” about some other “thing” => Attribute

This is just about avoiding redundancies

Entity Set vs Attribute examples

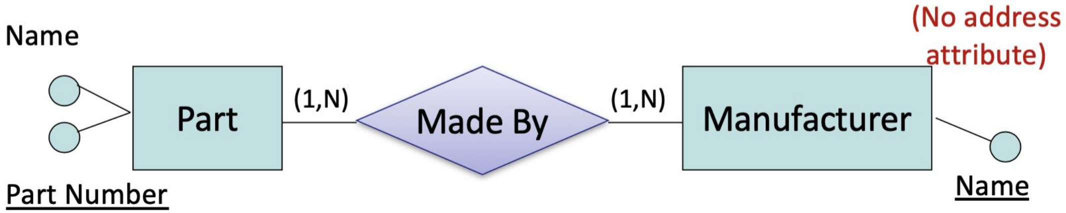
Domain Fact Change: A part can have more than one manufacture

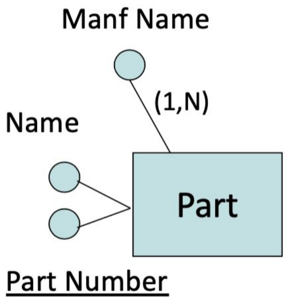




Contd.

Domain Fact Change: Not representing Manufacturer address





Limiting Weak Entity

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.

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 preferable
- An Integer key is preferable

Avoid Multi Attribute and String Keys

They are wasteful

- E.g. Movies(title,year, ...): 2 attributes in key, requires ~16 bytes–
- Number of movies ever made (<< 2) can be distinguished with 4 bytes
- Having an integer movieID key saves 75% space and a lot of typing

They break encapsulation

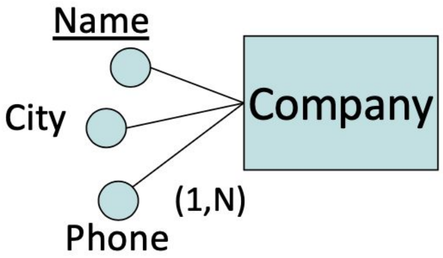
- e.g.Patient(firstName,lastName,phone, ...)
- Security/privacy hole=> Integer patientID prevents information leaks

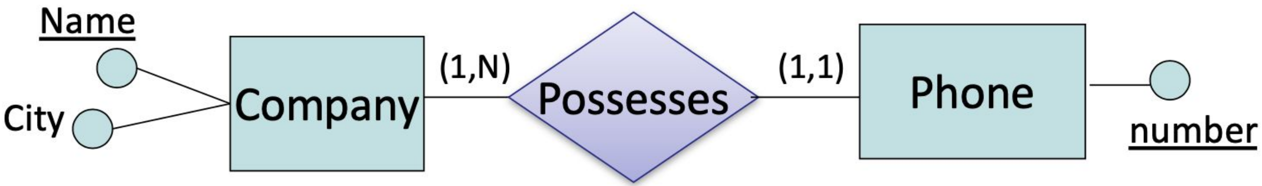
They are brittle (nasty interaction of above two points)

- 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

Attributes with Cardinality > 1

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





Design Decisions

- Consider address of a trainee.
  - Is it an entity or relationship?

- Consider address for a telephone company database, which has to keep track of how many and what type of phones are available in any one household, who lives there (there may be several phone bills going to the same address) etc.

- How do we represent employment of a trainee by a particular employer?
- How do we represent a course edition?

Translating into 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.

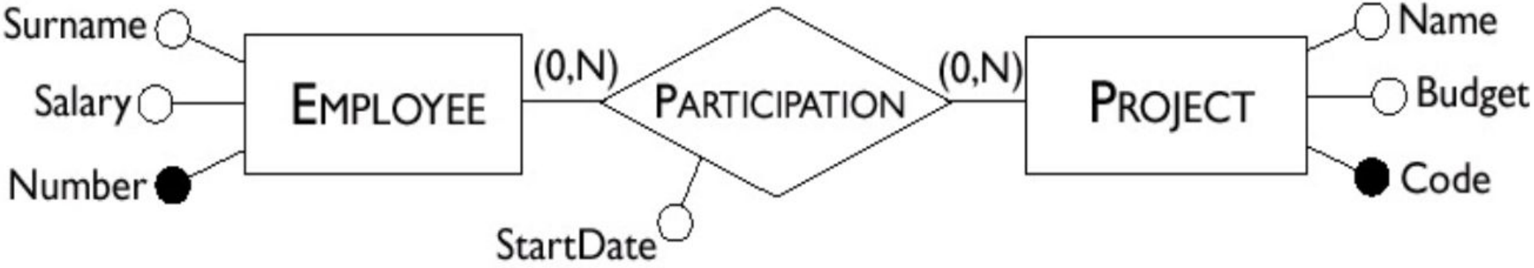
Each relationship becomes a relation.

It’s attributes are

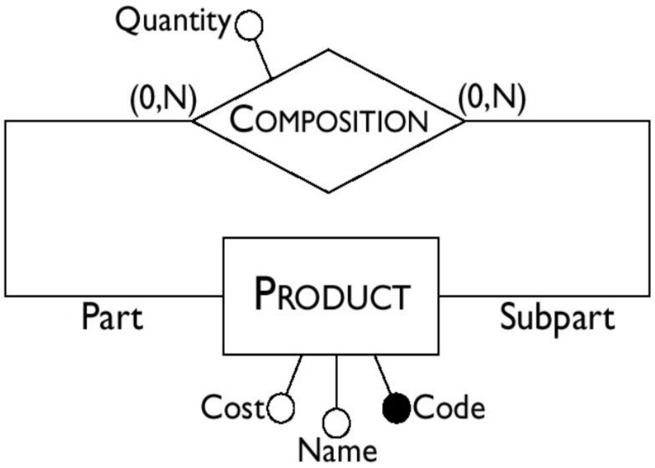
- the keys of the entity sets that it connects, plus
- the attributes of the relationship itself.

We’ll see opportunities to simplify.

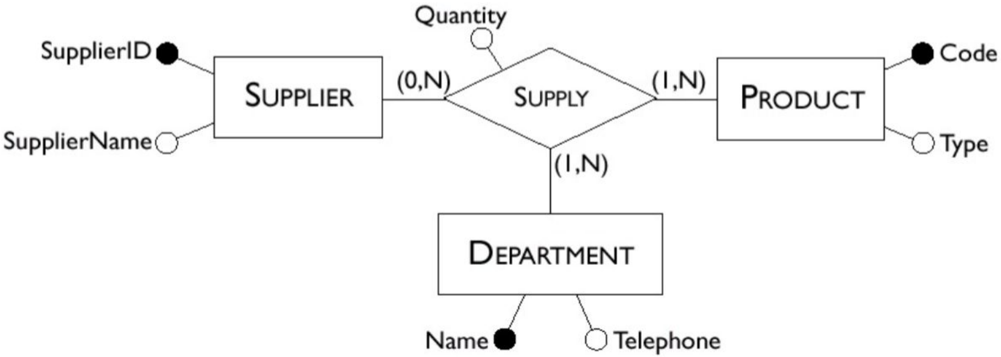
Many to Many Binary Relationships



Many to Many Recursive Relationships



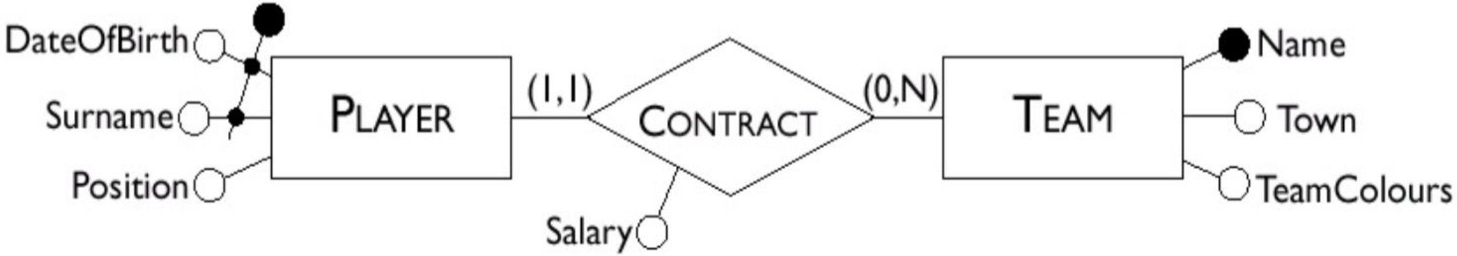
Many to Many Ternary Relationships



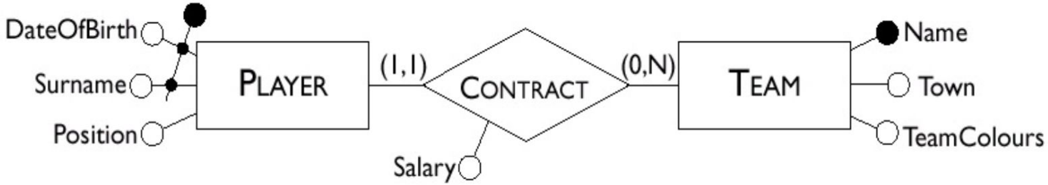
Simplification

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

One to Many Relationships - Mand participation on one side



One to Many Relationships - Mand participation on one side

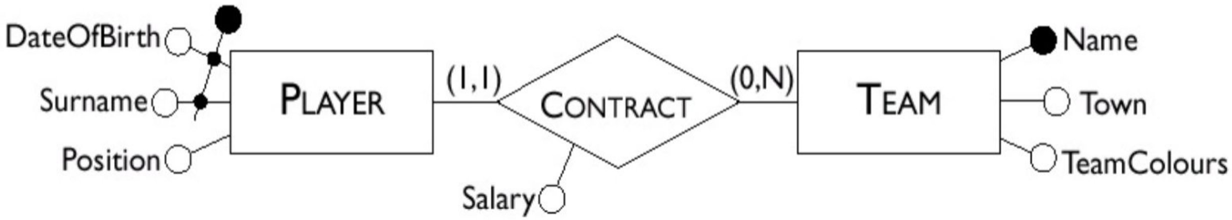


Standard translation:

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

Contract[PlayerSurname, PlayerDOB]⊆Player[Surname, DOB]
Contract[Team] ⊆Team[Name]

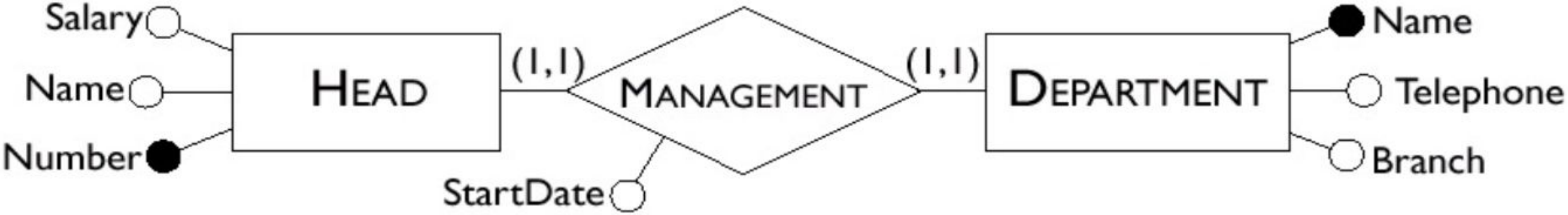
Contd.



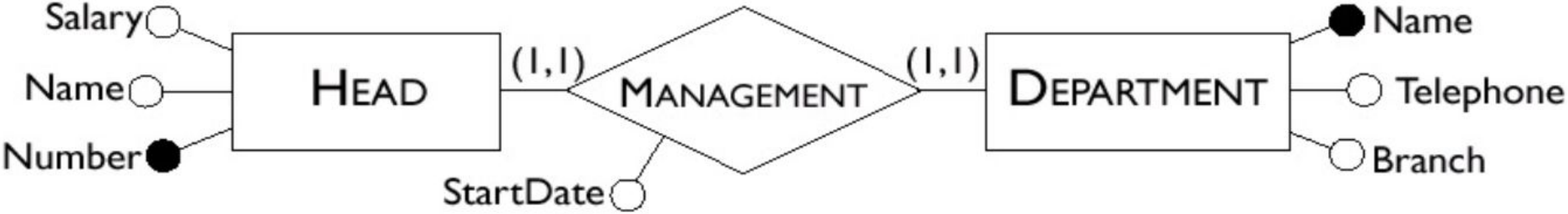
Simpler translation:

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

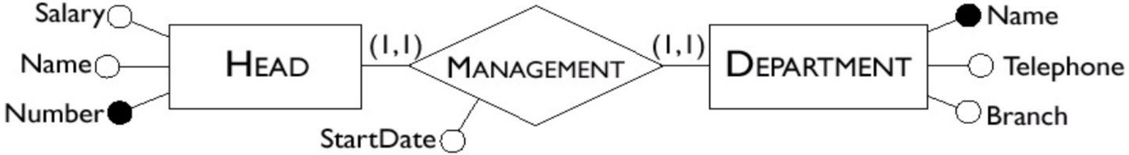
One to One Relationships - Mand participation for both



One to One Relationships - Mand participation for both



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



Simpler v1 (with Management info moved over to Head):

Head(<u>Number</u>, Name, Salary, Department, StartDate)

Department(<u>Name</u>, Telephone, Branch)

Head[Department] ⊆ Department[Name]

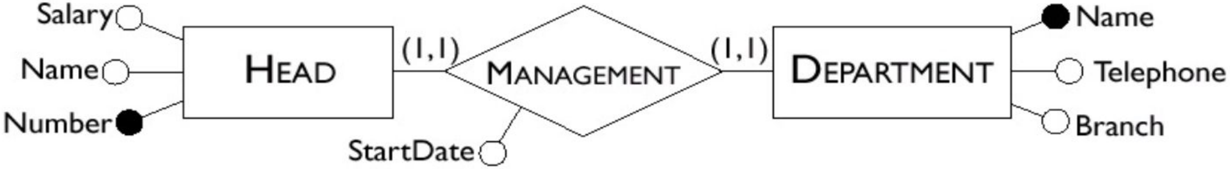
Simpler v2 (with Management info moved over to Department):

Head(<u>Number</u>, Name, Salary)

Department(<u>Name</u>, Telephone, Branch, HeadNumber, StartDate)

Department[HeadNumber] ⊆ Head[Number]

Contd.



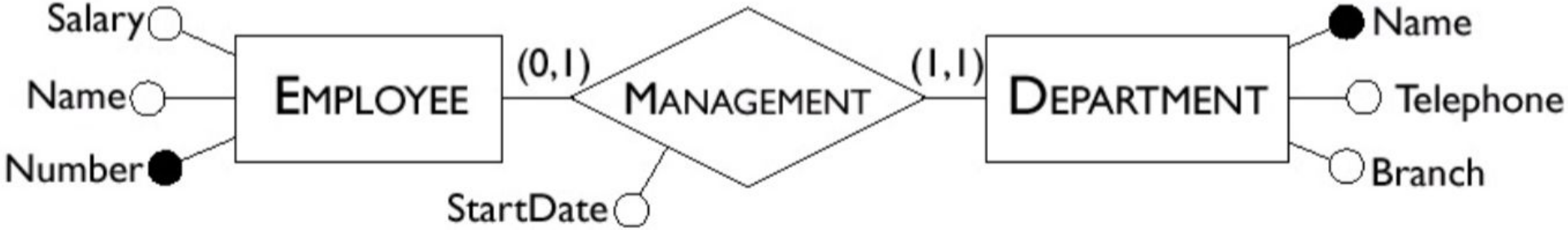
Simpler v3 (with Management info split between Head and Department):

Head(<u>Number</u>, Name, Salary, StartDate)

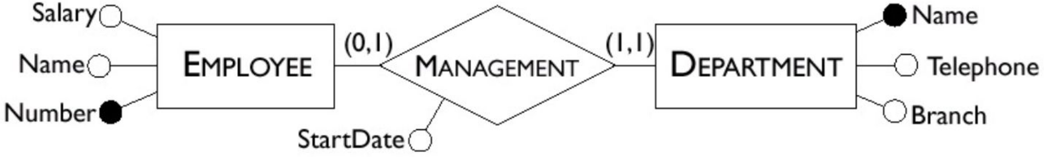
Department(<u>Name</u>, Telephone, HeadNumber, Branch)

Department[HeadNumber] ⊆ Head[Number]

One to One Relationships with optional participation for one



Contd.



Standard:

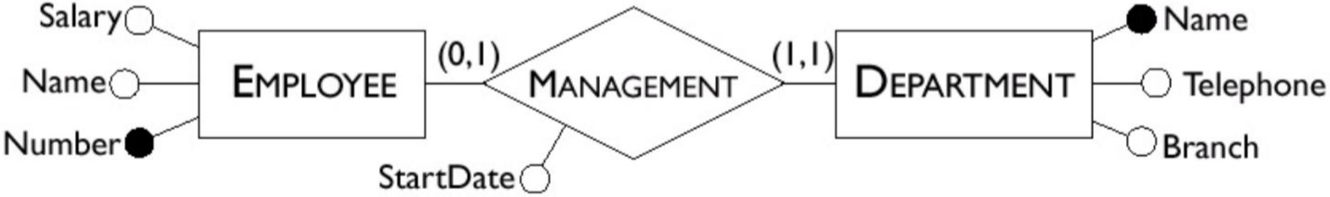
Employee(Number, Name, Salary)
Department(Name, Telephone, Branch)

Management(Head, Department, StartDate)

Management[Head] ⊆ Employee[Number]

Management[Department] ⊆ Department[Name]

Contd.



Simpler:

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

Department(<u>Name</u>, Telephone, Branch, Head, StartDate)

Department[Head] ⊆ Employee[Number]

Summary of Types of Relationship

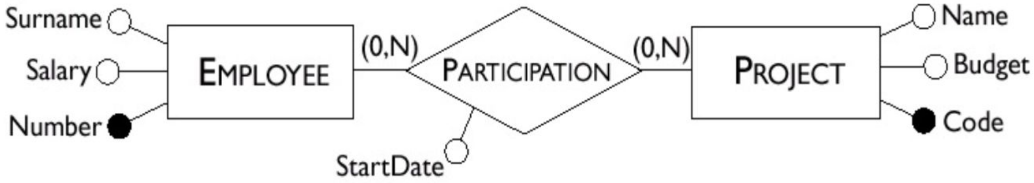
- Many to many (binary or ternary)
- one-to-many
  - mandatory: (1,1) on the “one” side
  - optional: (0,1) on the “one” side
- one-to-one
  - both mandatory: (1,1) on both sides
  - one mandatory, one optional:
    - (1,1) on one side and (0,1) on other side
  - both optional: (0,1) on both sides

Final Consideration

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

Name of Referring Attributes



Employee(Number, Surname, Salary)

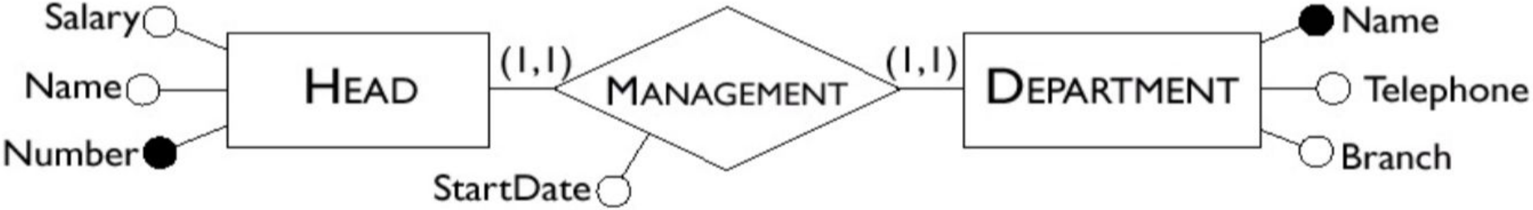
Project(Code, Name, Budget)

Participation(Number, Code, StartDate)

Participation[Number] ⊆ Employee[Number]

Participation[Code] ⊆ Project[Code]

What about max 1 constraint



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

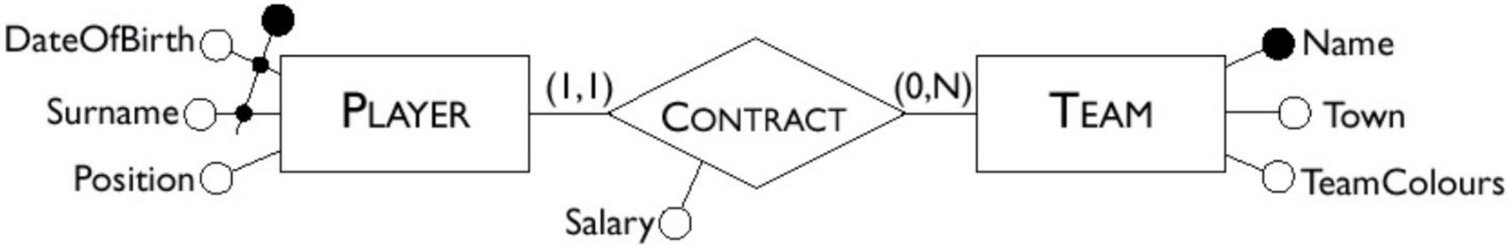
Head(<u>Number</u>, Name, Salary, Department, StartDate)

Department(<u>Name</u>, Telephone, Branch)

Head[Department] ⊆ Department[Name]

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

What about Min 1 constraint

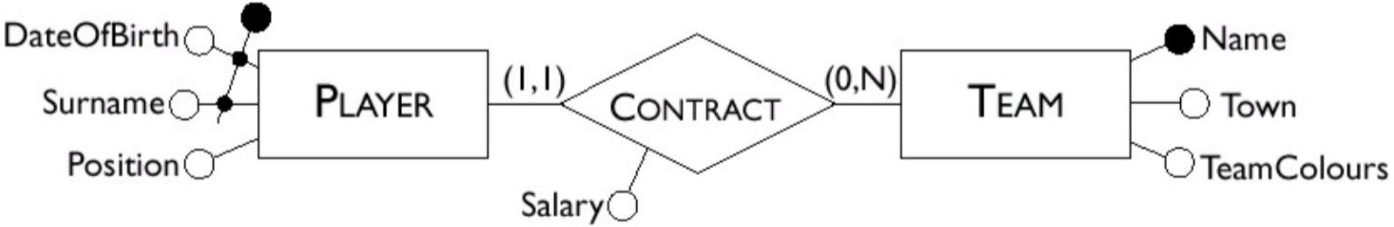


Our simplified translation:

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?

Contd.

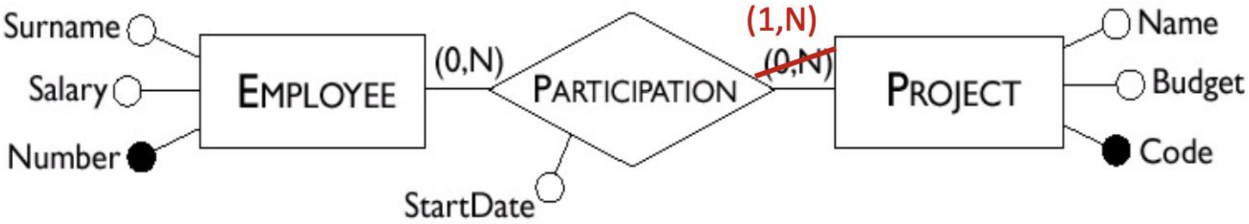


Our standard (non simplified) translation:

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

Does this enforce that every team must have a player?

Contd.

- In our translation from E/R to a relational schema, We expressedthe 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 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

- Advantage 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

Thank You