Approaches to ER schema design - Evolutionary (agile?) modelling

Reading: Elmasri & Navathe, Fundamentals of Database Systems, Chapter 3

LAST LECTURE

- know, apply (and be able to draw) the **THREE** main different ER notions
- know, apply (and be able to draw) the **THREE** types of attribute an entity type can have
- know, apply (and be able to draw) the **TWO** types of constraint that can be put on relationships

Designing an ER schema

- Need to identify basic components:
  - entity types, relationship types, attributes
  - and for each of these components:
    - *key* attributes (unique for each entity)
    - cardinality and participation constraints of relationships
    - different entity types

Strategies to ER design

- **top-down**: start with schema containing high-level abstractions and apply successive top-down refinements
- **bottom-up**: start with a schema containing basic abstractions then proceed by combining and adding to these
- **inside-out**: start from a central set of concepts, that are most evident and spread outwards, by considering new concepts in the vicinity of existing ones

Evolutionary data modelling

- Evolutionary data modelling is an approach that proceeds in an incremental manner
  - an initial ***slim*** model is created that satisfies some initial requirements (need to decide which)
  - the model is then refined in a set of iterations, adding details (need to decide which at each iteration)
- At each iteration, a database can be built with a set of functionalities, queries, interface etc.
  - (we will ignore this and only discuss data modelling)

Is Evolutionary = Agile?

- Agile data modelling is evolutionary data modelling done **in a collaborative manner**
    - **Agile** is a set of principles, not a specific technique (see the agile manifesto at https://agilemanifesto.org/)
- you can decide whether you want to apply evolutionary modelling in a highly collaborative setting or in a traditional development setting
    - *(we will have a go at evolutionary, agile (scrum based) ER modelling in Assignment 2, though this approach will not be mandatory - more in Tutorial 3)*

but <u>where</u> do we start?

- From the requirements, as usual
- need to find a way to make sense of them, in a systematic and efficient manner

User ‘stories’

- primary tool for Agile/Scrum/Extreme Programming strategies
- a user story is a very high level and very concise statement of a requirement
- much <u>much</u> smaller than a “use case”: it’s literally just one sentence!

Examples of stories

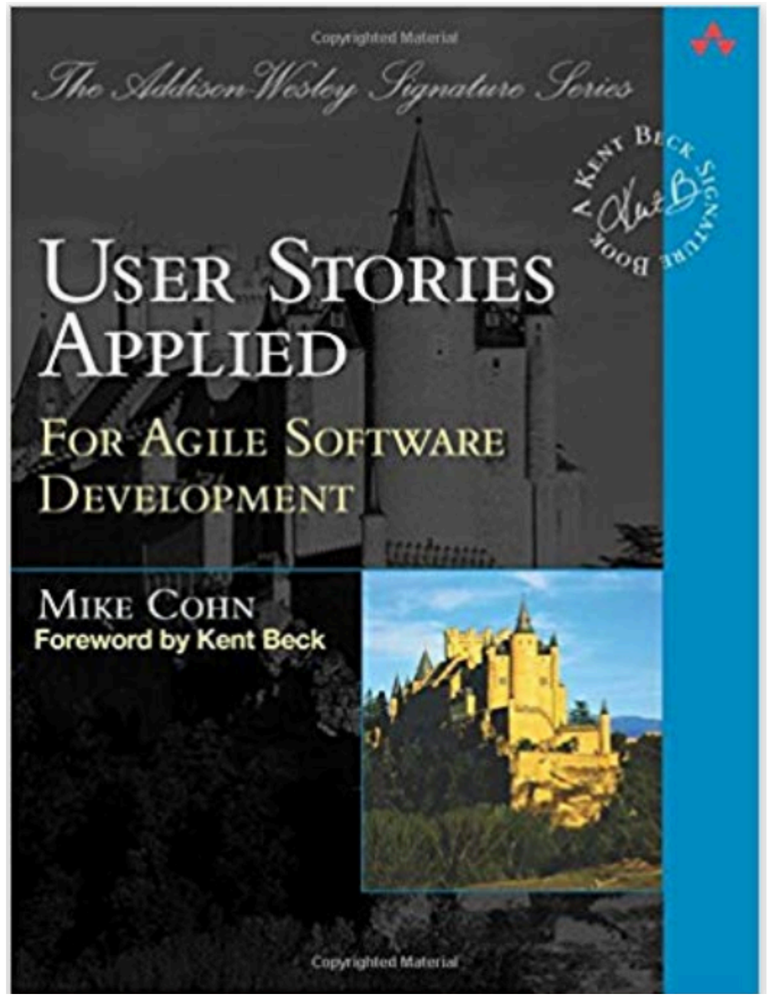
1. Students can enrol in a module online
2. Students can only enrol in a module if it is included in their programme
3. Students can see their marks online
4. Lecturers can input their feedback on the Virtual Learning Environment
5. Timetables can be downloaded and printed

Each of these is ONE SINGLE STORY

Creating a story set

- can collect them informally (just sentences extracted from the requirement analysis) or use a method/format/template
- Important thing is to collect them systematically:
  - number them
  - order them
  - prioritise them

User story template



- *As a (role) I want (something) so that (benefit).*

- Or more complex (remember the Tutorial?):
  - *As a… [which type of user has this need?]*
  - *I need/want/expect to… [what does the user want to do?]*
  - *So that… [why does the user want to do this?]*
  - *When… [what triggers the user's need?]*
  - *Because… [is the user constrained by any circumstances?]*

Evolutionary strategy

- Collect, order and prioritise your user stories
- Decide how many iterations you want to make
- Decide which new stories you want to include in the design at each iteration
- Proceed to create an Entity Relationship model that represents those user stories

General criteria for design

1. if a concept has significant properties and/or describes classes of objects with an autonomous existence, it is appropriate to represent it as an **entity**

    - for example, an instructor can be an entity, as it possesses various properties (name etc) and its existence does not depend from other concepts

General criteria for design

2. if a concept has a simple structure, and has no relevant properties associated with it, it is convenient to represent it as an **attribute** of another concept to which it refers

    - for example: a town may well be an entity in general, but for this application it can more appropriately be modelled as an attribute

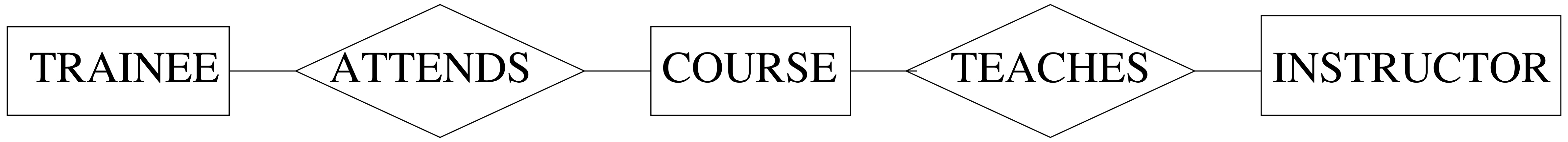
General criteria for design

3. if the requirements contain a concept that provides a logical link between two or more entities, this concept can be represented by a **relationship**
    - for example, the concept of attending a course.

We wish to create a system for a company that runs training courses. For each course participant, identified by a code, we want to store the national insurance number, surname, age, sex, place of birth, employer's name, address and telephone number, previous employers (and period employed), the course attended and the final assessment of 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 course is given, we call it an "edition" of the course. For each edition, we represent the start and end dates and the number of participants. If a trainee is a self employed professional, we need to know his or her area of expertise, and, if appropriate, his or her title. For somebody who works for a company we store the level and position held. For each instructor we will show surname, age, place of birth, the edition the course is taught, those taught in the past and the courses the tutor is qualified to teach. All the instructor's telephone numbers are also stored. An instructor can be permanently employed or freelance.

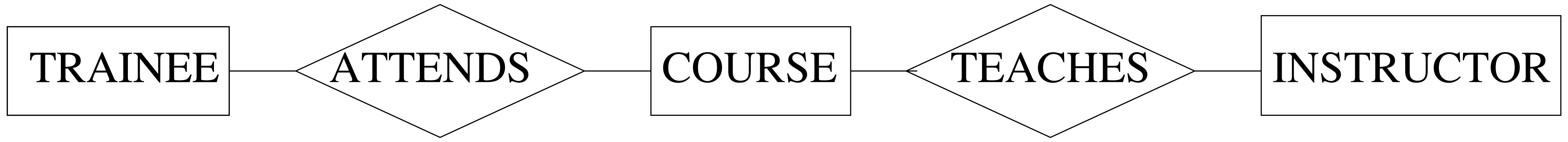
1st iteration: two stories

1. Trainees attend Courses
2. Instructors teach Courses

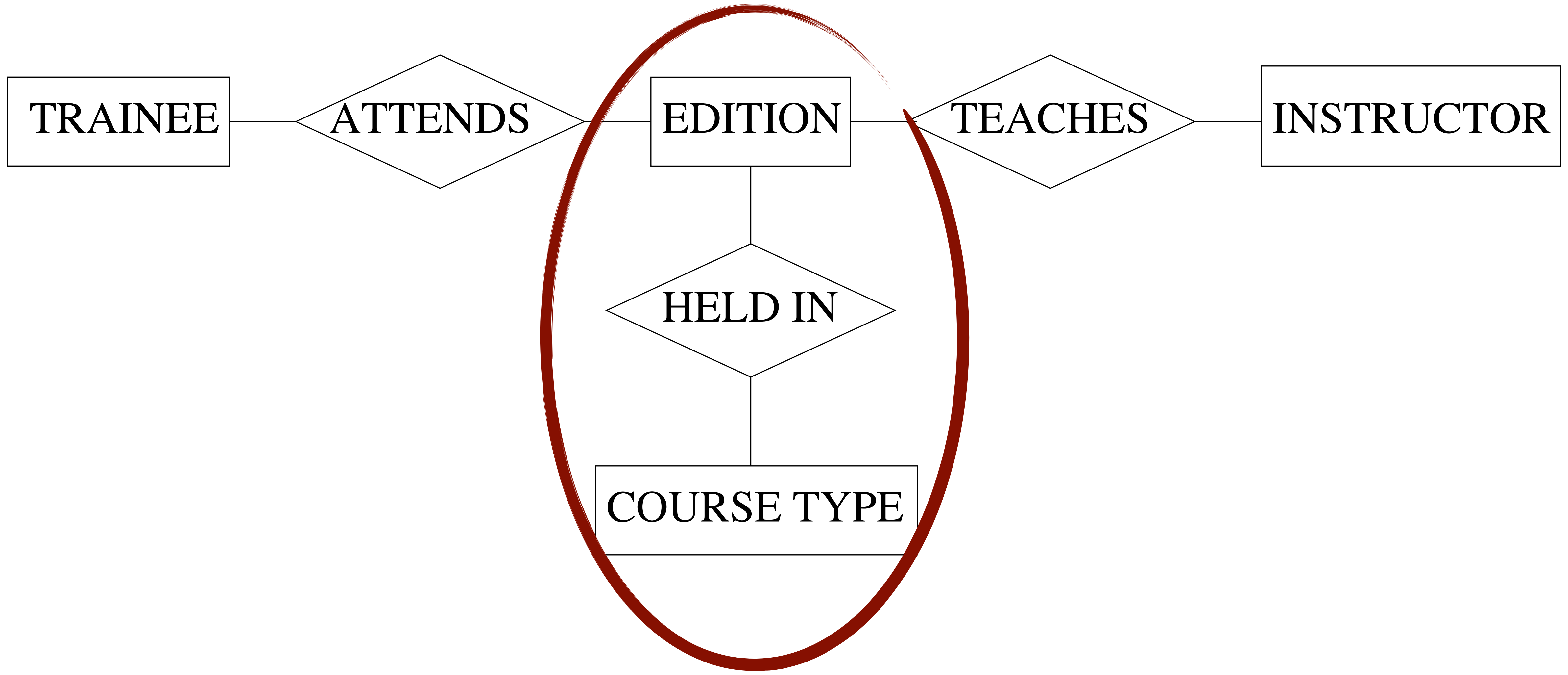


2nd iteration: three stories

3. Courses are held in “editions”
4. Trainees can be self employed professionals or work for a company
5. We distinguish between current and past editions



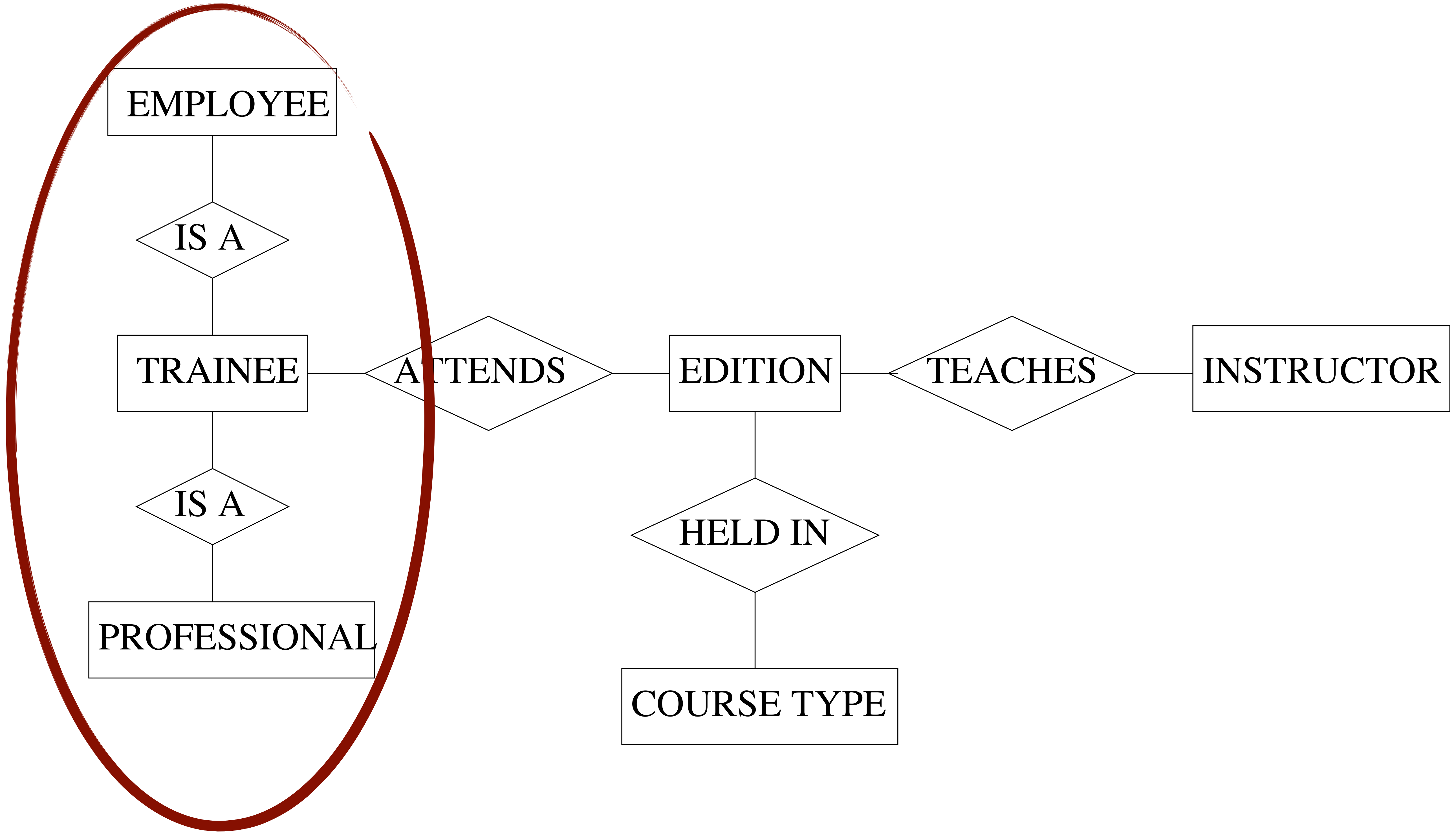
- *Story 3: "Courses can be held in Editions"*
  - *from 1 entity to 2 entities+relationship*
    - *identify cases in which an entity describes two different concepts logically linked to each other:*



- *Story 4: Trainees can be self employed professionals or work for a company*

- *from 1 entity to 1 entity+N entities+N relationships*

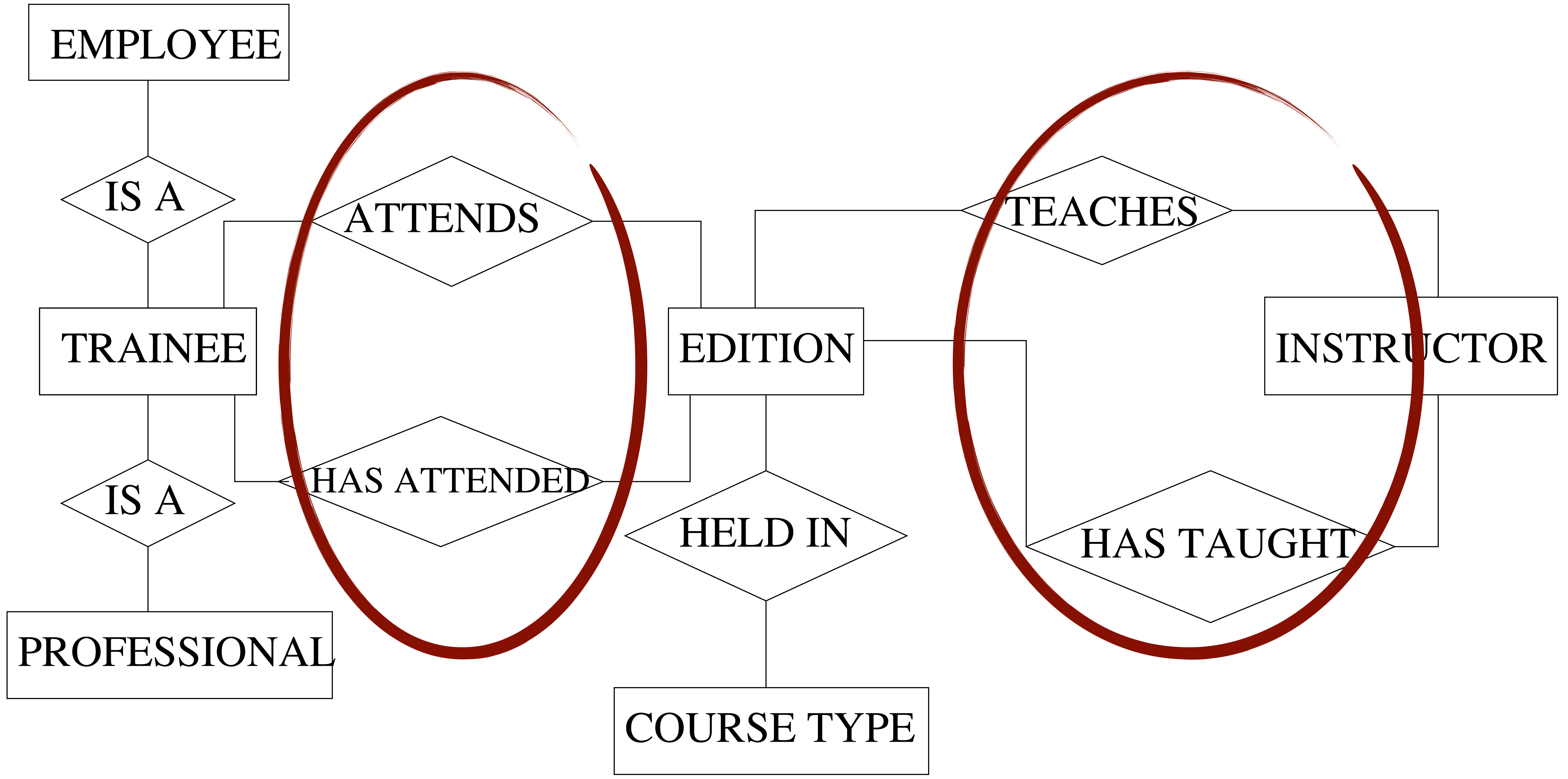
    - *identify cases in which an entity is made up of distinct sub-entities:*



- *Story 5: we distinguish between current and past edition of a course*

- *from 1 relationship to multiple relationships*

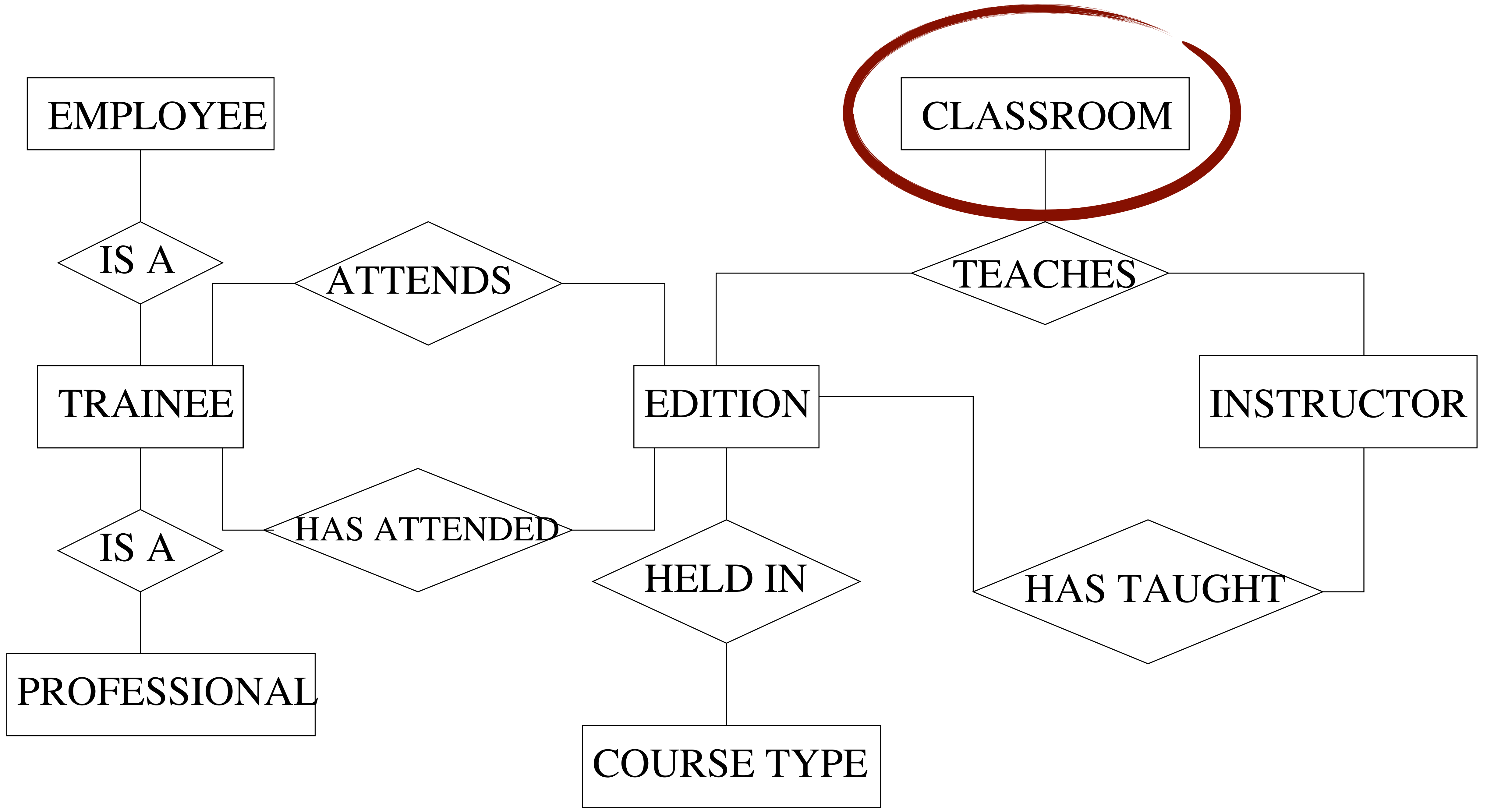
  - *identify cases in which a relationship describes two or more different concepts linking the same entities:*



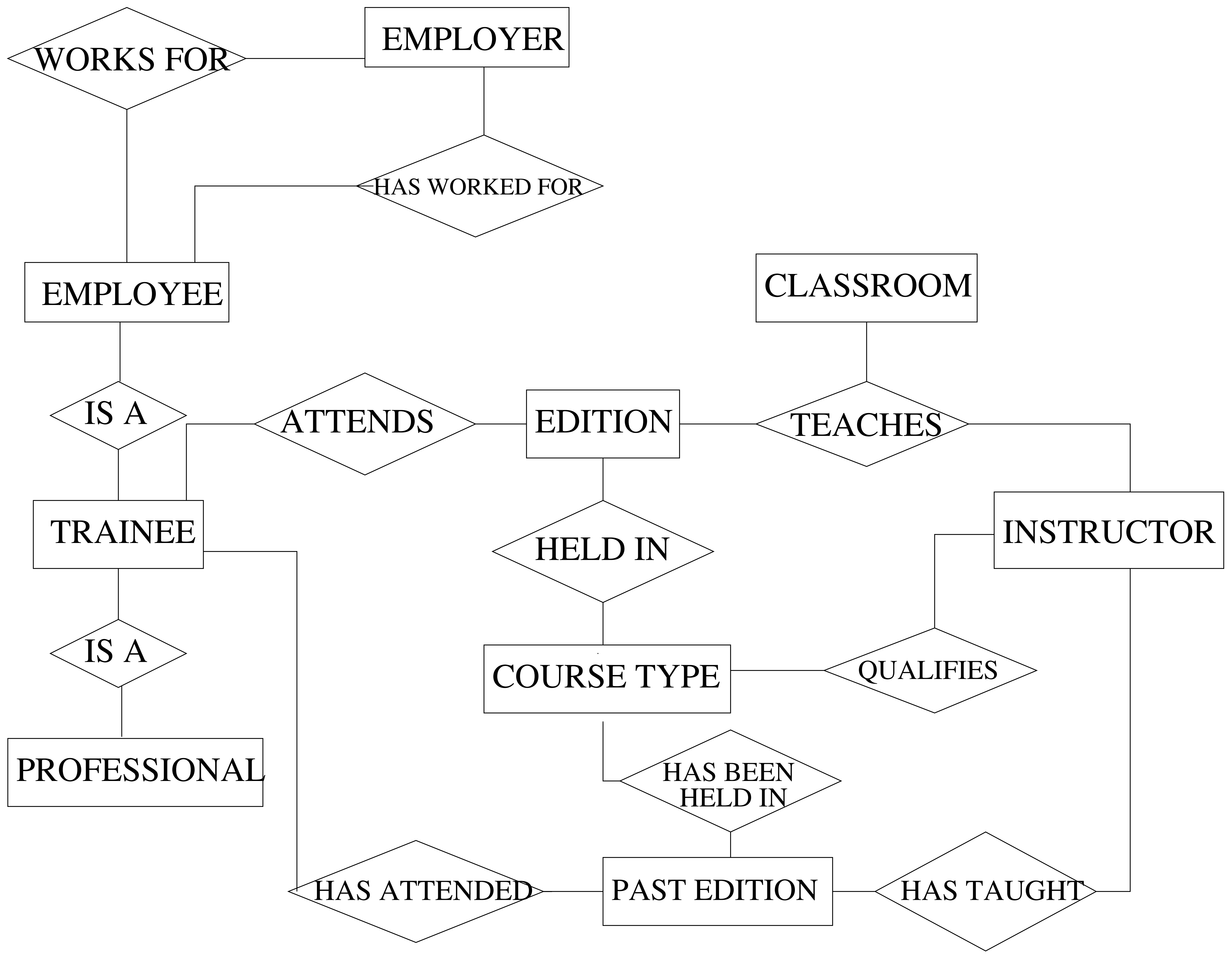
3rd iteration: four stories

6. Courses are held in classrooms
7. Instructors only teach Courses for which they are qualified
8. We archive past editions of courses keeping summary data
9. We maintain data of trainees' employers

- *Story 6: Courses are held in classrooms*
  - *from 1 relationship to 1 entity + relationships*
    - *identify cases in which a relationship describes a concept having an autonomous existence:*



- *Story 7, 8 and 9: spot the differences!*



dodgy entities

- Not very comfortable with the “Edition” entity
- also with the “Trainee” and “Professional” entities
- they intuitively seem to have some different quality to them
    - Edition is just the “installation” of one course, it’s not a different entity…
- we want to have a way to “mark” this difference

A special type of entity: *weak entity type*

- these are entity types which cannot be identified in isolation
- instances are identified because they "belong" to specific entities from another entity type, known as *identifying owner*
  - for instance, the content of a lecture theatre (white boards, desks, etc.) cannot typically be identified directly (unless we label every single item on campus)
  - the lecture theatre is their identifying owner, so we can talk about "the front desk in the Ashton Lecture Theatre"

Weak ➜ has an owner

- the relationship type that relates the weak entity to its owner is the weak entity's *identifying relationship*
  - in the example above, the "is in" relationship
- weak entity types might have a partial key, to distinguish one weak entity from other weak entities related to the same owner
  - for example "desk 1 (or 2, 3 etc.) in the Ashton LT"

Weak entity vs total participation

- a weak entity cannot exist in isolation, must have an owner

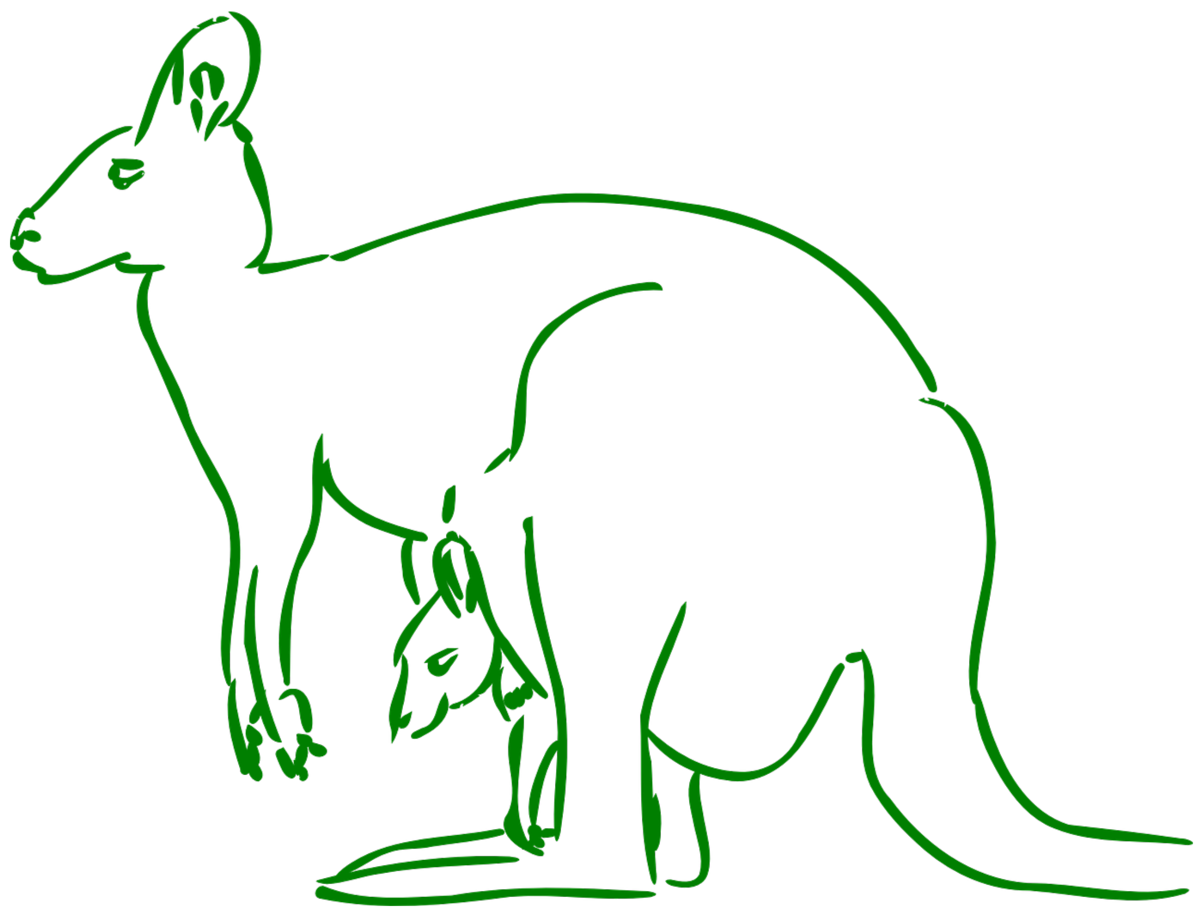
- so, it's often confused with an entity in a "total participation" relationship

strong entity
total participation

  - "a lecturer *must* work for a department", but lecturer is not a weak entity (they have a "staff no.", they can be identified)

weak entity

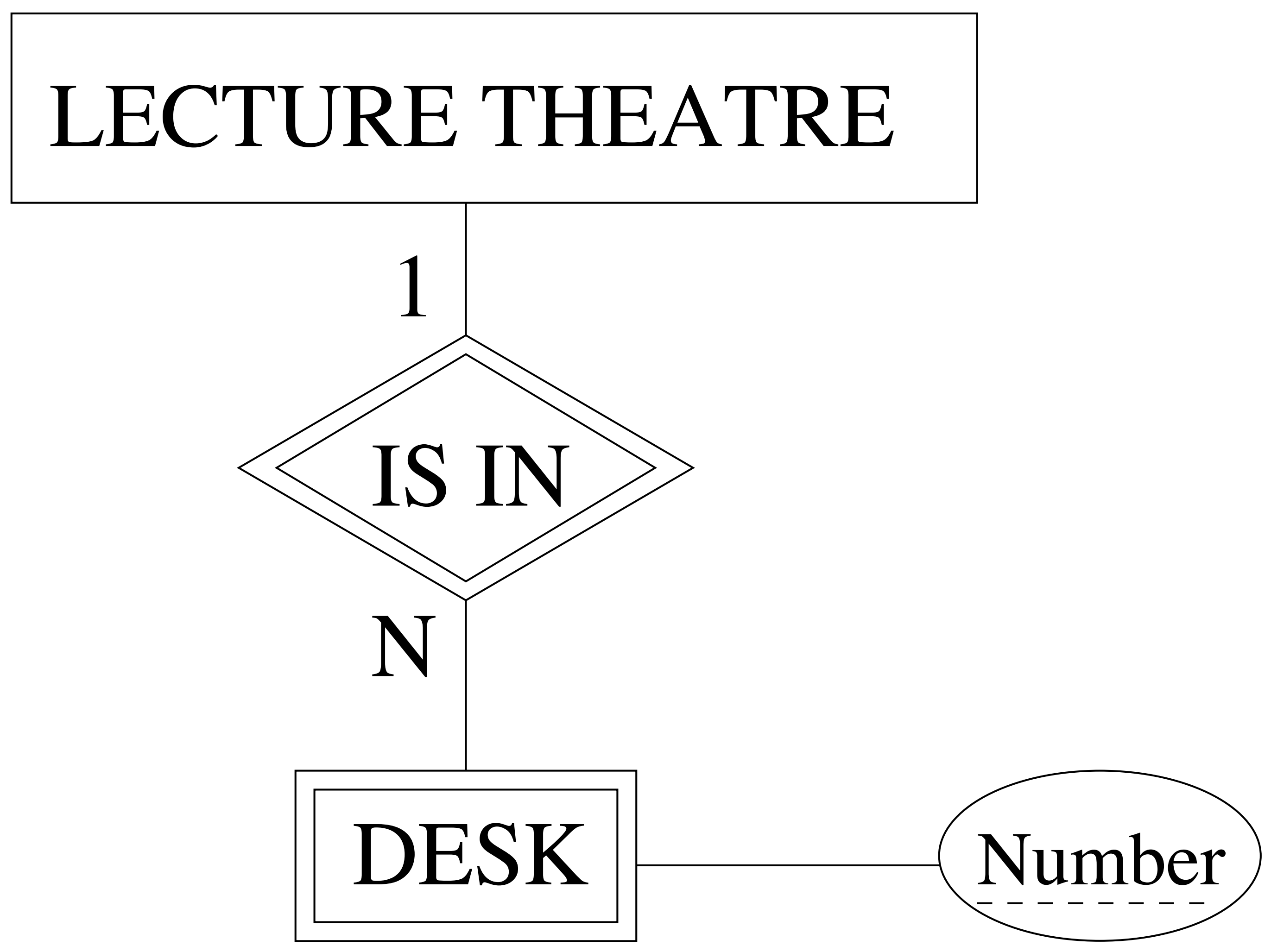
  - "a desk *must* belong to a lecture theatre", and is weak as we don't have a direct ID for it



Useful metaphor: the entity owner "carries" other weak entities

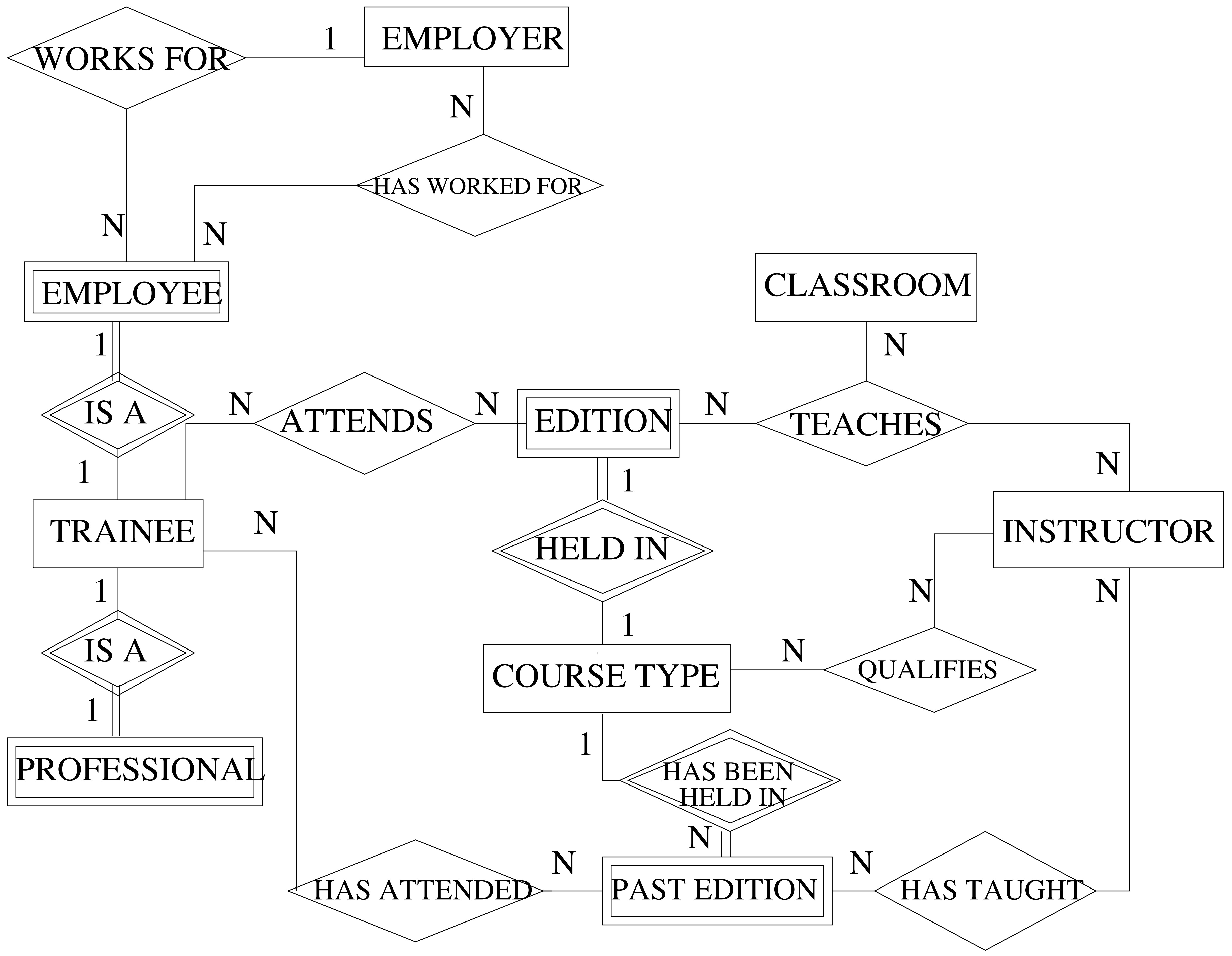
ER notation: weak entity

- A weak entity type is represented as a double box
- and the identifying relation as a double diamond
- a partial key has a dotted underline

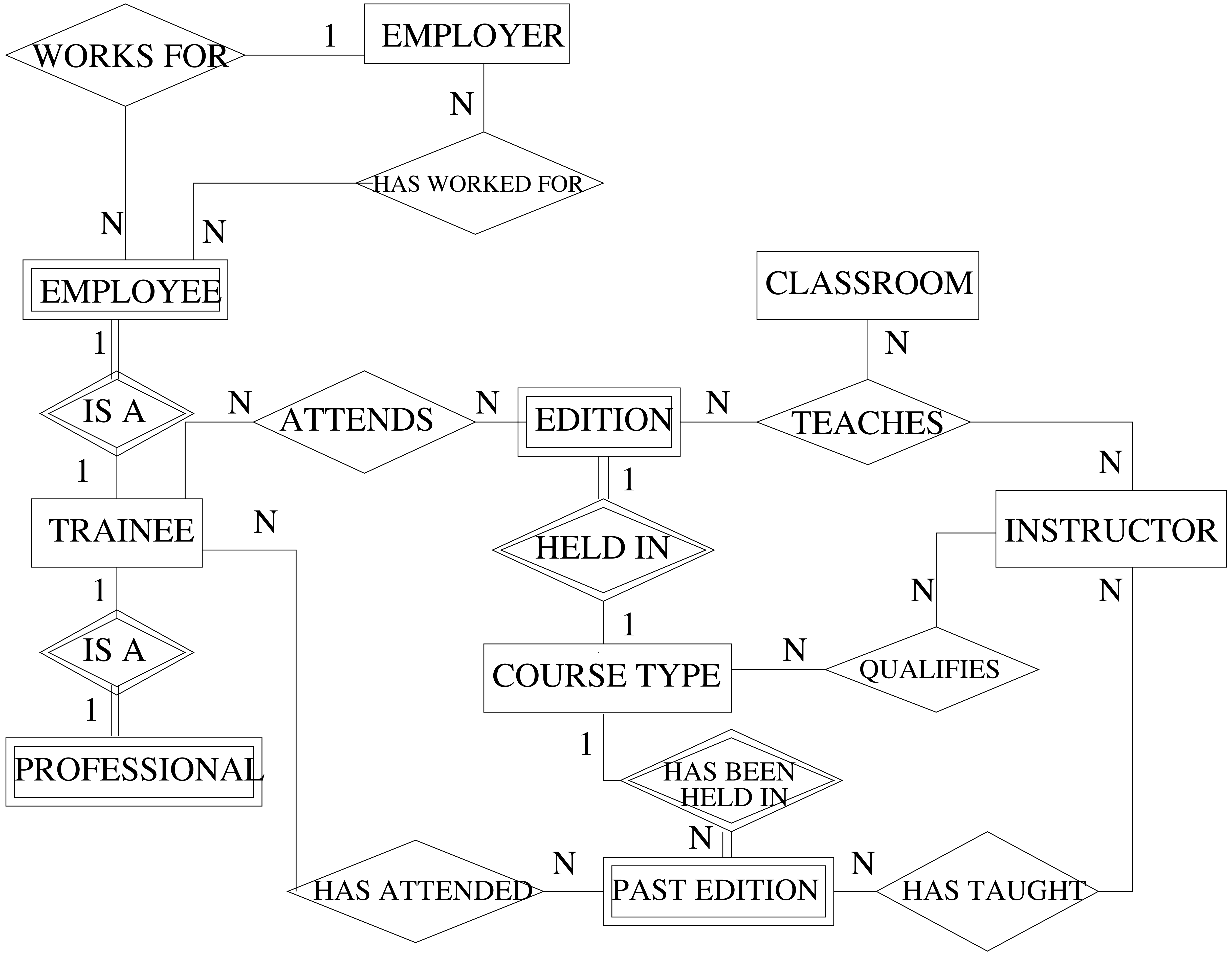


with this in mind…

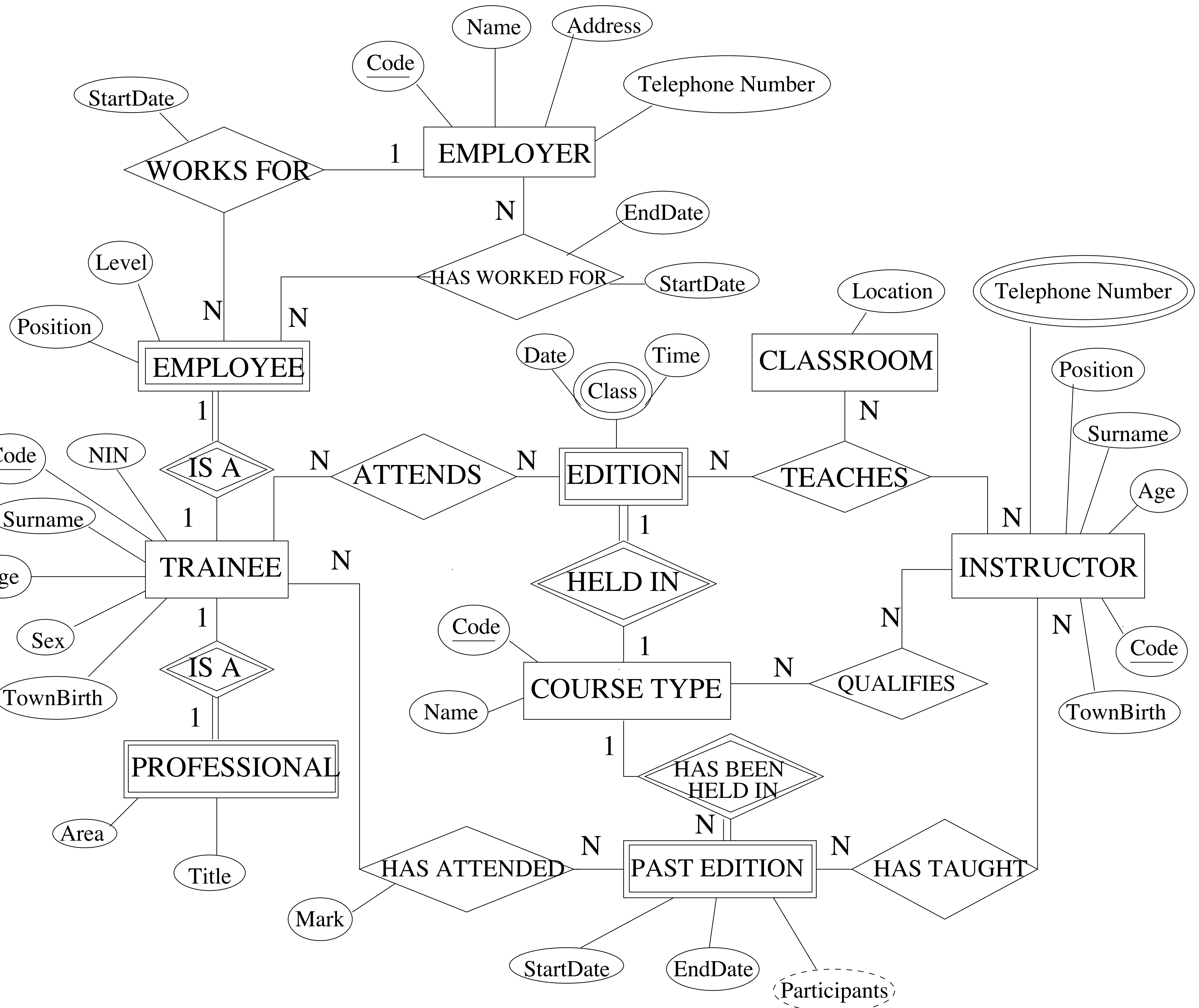
- *establish cardinalities and participation constraints for relationship, and **strong/weak** entity types*



- *and now, finish off with the attributes (you could add attributes at each iteration - or you could add just the main attributes, then refine, …)*



- *Final ER schema*



Example from the textbook: the Company

- A company is organised in Departments which can be in several Locations
- A Department controls a number of Projects
- The company has Employees
- An employee is assigned to **one** Department, but may work on **many** Projects
- An employee may have **one** supervisor (also an employee of the company)
- A Department is managed by **one** employee
- For each employee, the company keeps track of their dependents (eg. spouse, children, etc) for insurance purposes

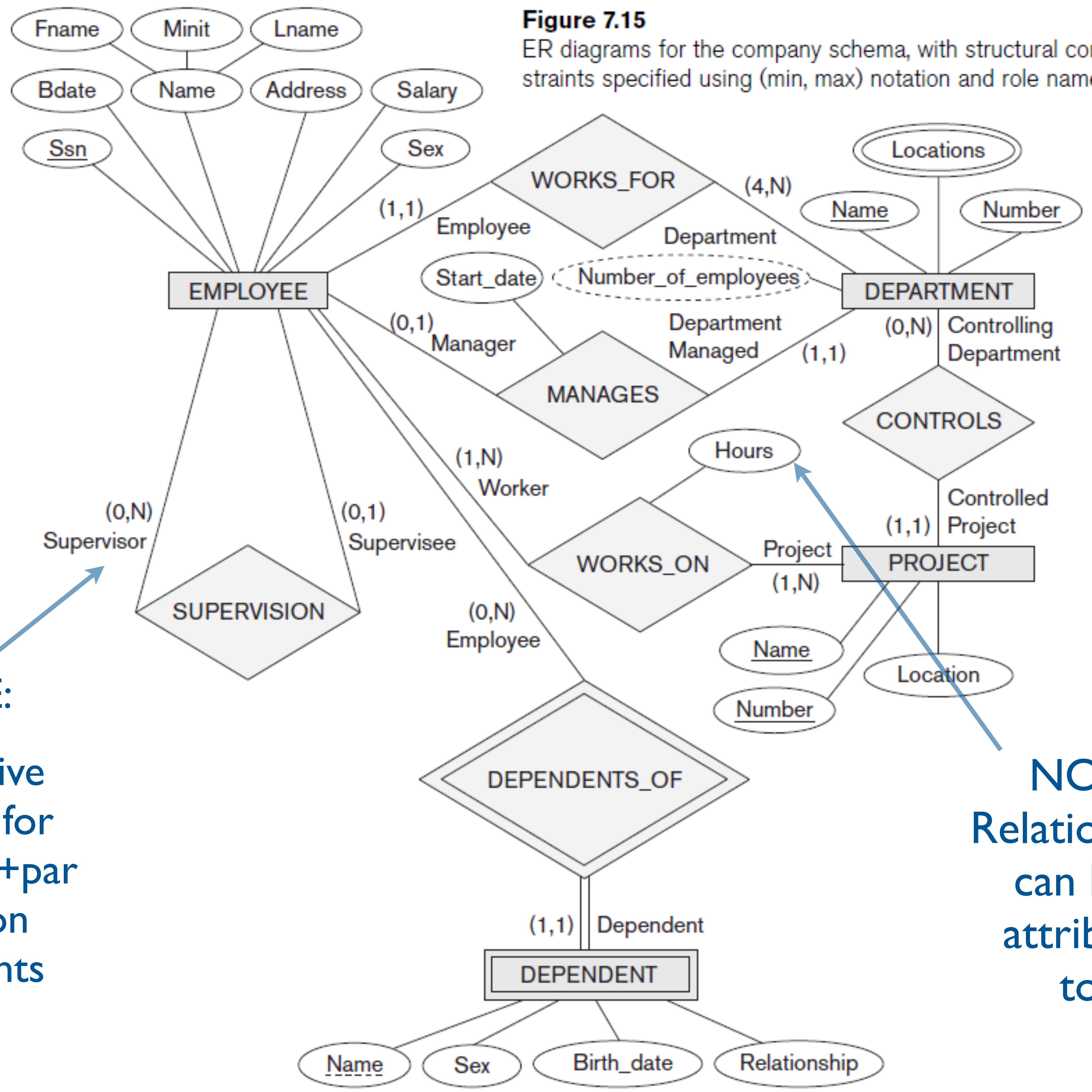


Figure 7.15
ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names.

NOTE:

Alternative notation for cardinality+participation constraints

NOTE:
Relationships can have attributes too