Databases and Info Systems

Database Design

Dr. Seán Russell
sean.russell@ucd.ie,

School of Computer Science,
University College Dublin

March 18, 2020

Table of Contents

1 Designing an Information System
- Information System Life Cycle
- Information Systems and Databases

2 Database Design

3 Conceptual Design

4 Enhanced Entity Relationship Model (EER)

5 Creating an E-R Model

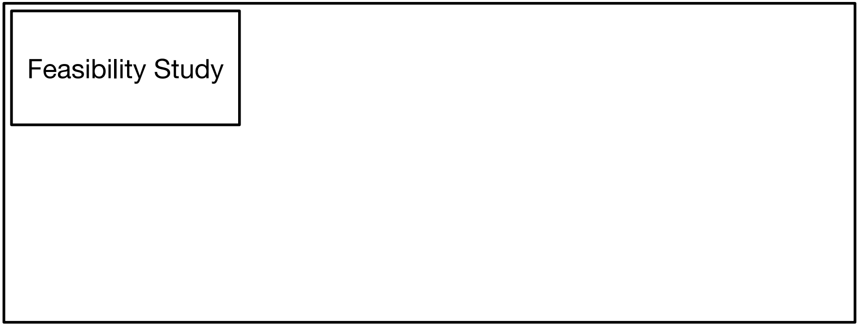
Designing a Database

- Database design in just one of the many activities in the development of an **Information System** (IS) within an organisation
- It should therefore be presented within the wider context of the **information system life cycle**
- An information system is a component of an organisation that **manages** (gets, processes, stores, communicates) the **information** of interest
    - Usually, the information system operates in support of other components of the organisation

Components of Computer based IS

- Database (DB)
- Database Software
- Application Software
- Computer Hardware
- Personnel using and developing the system

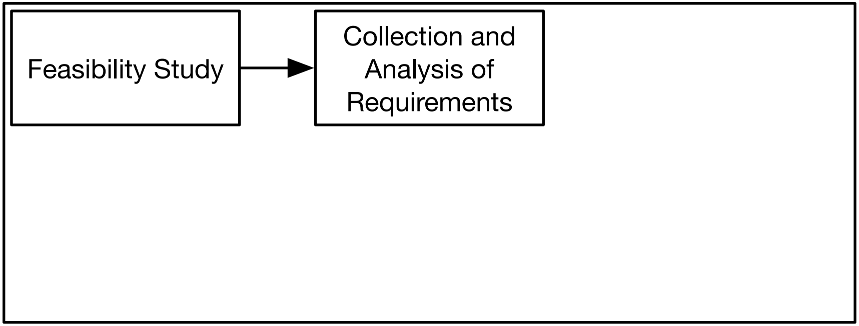
Note: The DB is a **fundamental** component



Feasibility Study

The feasibility study is carried out to;

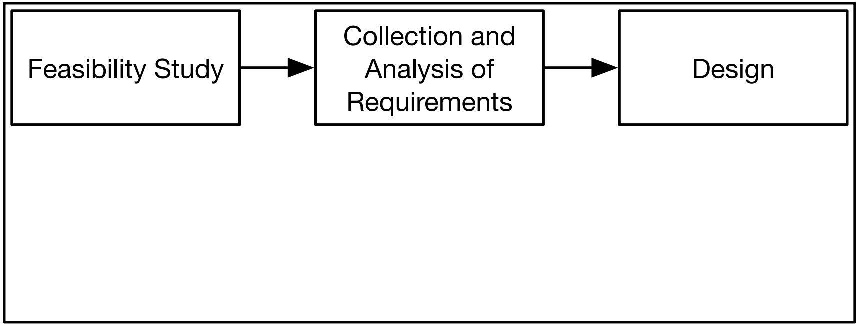
- Define the costs of the various possible solutions
- Establish the priorities for the creation of the various components of the system



Collection and Analysis of Requirements

This is where we define and study the properties and functionality of the information system

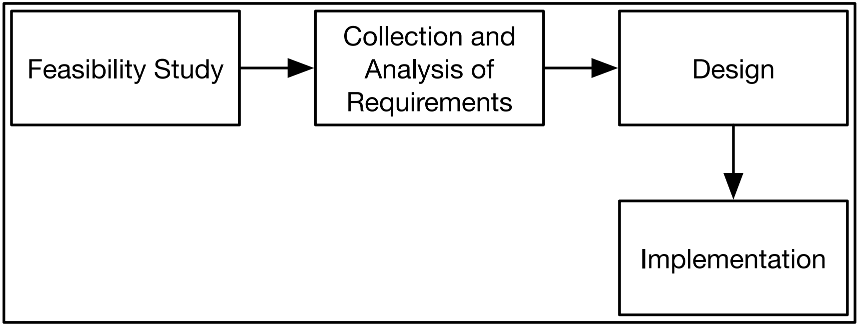
- We work out what the system should be able to do



Design

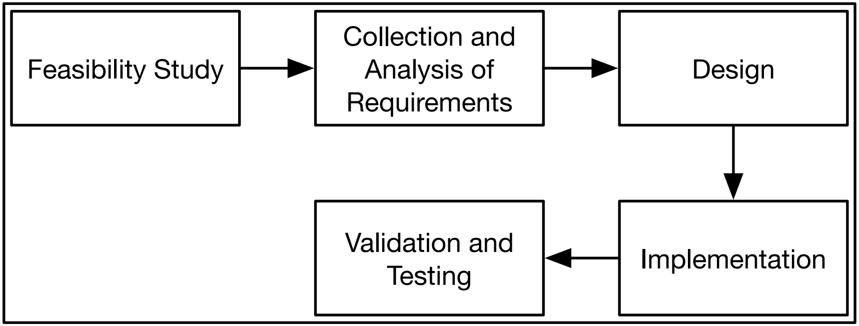
This is generally two tasks;

1. Designing the database
2. Designing the applications that will use the database



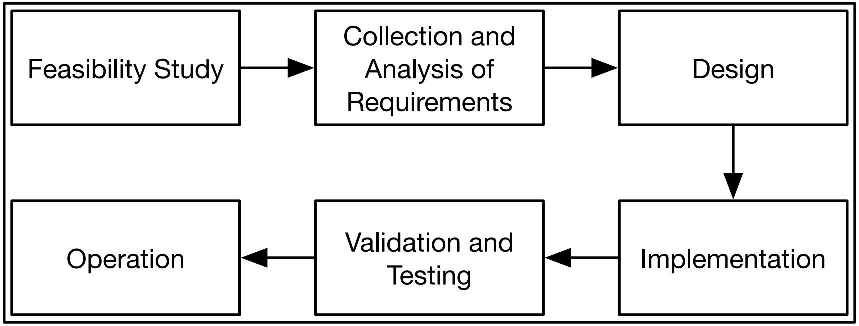
Implementation

The information system is implemented (Database and Applications)



Validation and Testing

This is where we check that the system is functioning correctly and working well.



Operation

Finally, in the operation phase, the information system becomes live and performs the tasks that it was designed for.

- The database is only one of the parts of an information system
  - This also includes programs, user interfaces and other service programs
- However, because the data is a very important part of this system, we will specifically study database design
- So we will only be looking at the parts of an information system that is closely related to databases
  - We will focus on **data design** and the related activities of **requirements collection and analysis**

Table of Contents

1 Designing an Information System

2 Database Design

3 Conceptual Design

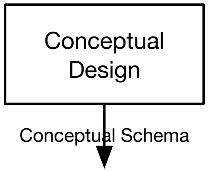
4 Enhanced Entity Relationship Model (EER)

5 Creating an E-R Model

A Database Design Methodology

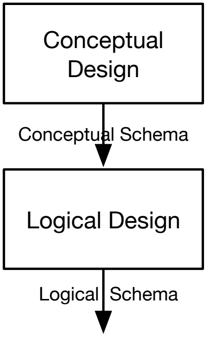
- The most common databse design methodology is based on a simple engineering principle
- Separate the decisions relating to **what** to represent in the database from the decisions relating to **how**
- This methodology is divided into three phases to be followed consecutively

Conceptual Design



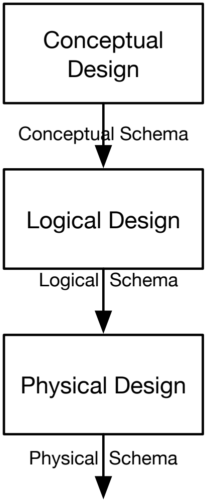
- The purpose of this is to represent the informal requirements of an application.
- Focus on the concepts that should be modelled, NOT the way they will be stored in a database (this will come later)
- Output is a conceptual schema, usually using and **Entity-Relationship Model**.

Logical Design



- This consists of the translation of the conceptual schema into the logical schema of the database that refers to a logical data model.
- This is the definition of the tables and attributes that will be created to store the data in our database.

Physical Design



- In this phase, the logical schema is completed with the details of the physical implementation (file organisation and indexes) on a given DBMS.
- The product is called the physical schema and refers to the physical data model.

Table of Contents

1 Designing an Information System

2 Database Design

3 Conceptual Design
- The Entity Relationship Model
- Relationships
- Entity Types
- Example

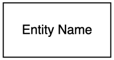
4 Enhanced Entity Relationship Model (EER)

5 Creating an E-R Model

E-R Model

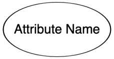
- The Entity Relationship (E-R) model is a **conceptual data model**
- It is capable of describing the data requirements of an application in a way that is easy to understand and is independent of the criteria for the management and organisation of data on a database system
- An E-R model is normally shown as a graphical representation

Entities

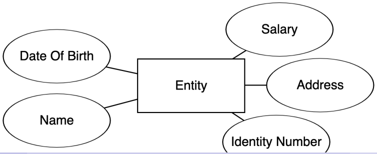
- Entity: 

- An "entity" is some thing that we want to store data about.
- It can be something that has;
    - **physical existence** (e.g. a person, building, car, etc.) or
    - **conceptual existence** (e.g. a university module, a job, etc.)
- In the E-R model we define types of entities.
- An "occurrence" of an entity is similar to an object that is an instance of a class.

Attributes

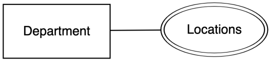
- Attribute: 

- Entities are described by their "attributes", which are the properties that describe an entity.
- Each attribute has a "domain" (a set of allowed values)
- For example, an "employee" entity might be described by the following attributes: identity number, name, date of birth, salary, address, etc.

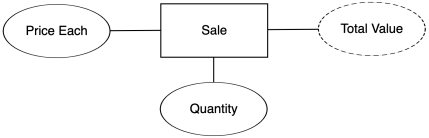


Attributes

- A double oval indicates that an entity can have multiple values for the same attribute

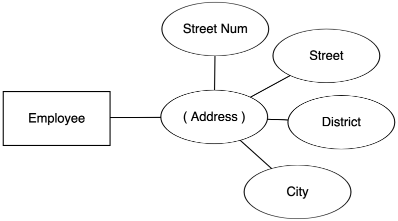


- A dashed oval indicates that an attribute is a derived attribute (i.e. it can be calculated from the entity's other attributes).



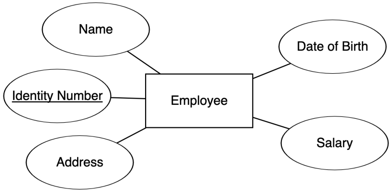
Composite Attributes

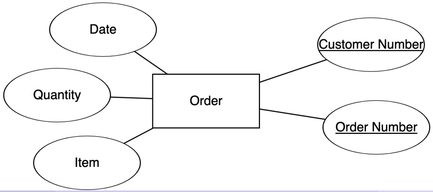
- Sometimes, an attribute can be a **composite attribute**, made up of a number of component attributes



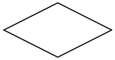
Keys

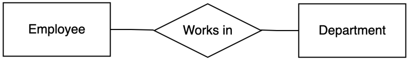
- If an attribute is the entity's primary key (or is part of a composite primary key), then its name is underlined



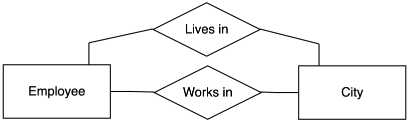


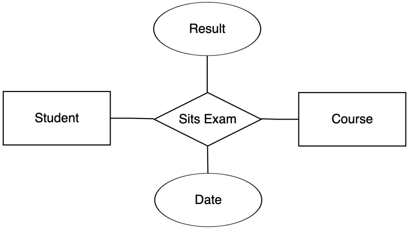
Relationships

- Relationship: 
- A relationship is a meaningful association between two or more entity types. Some examples:
  - An employee WORKS IN a department.
  - A student is REGISTERED FOR a university module.
  - A car is OWNED BY a person.
  - A salesperson SELLS a product to a customer.

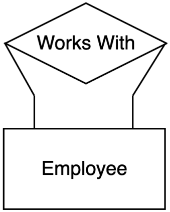


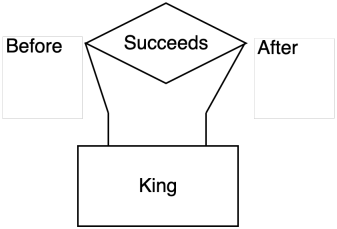
Relationship Examples





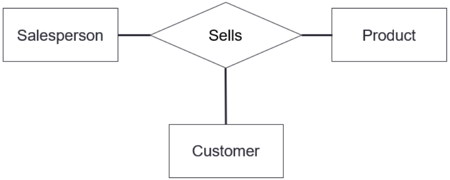
Recursive Relationships





Ternary Relationships

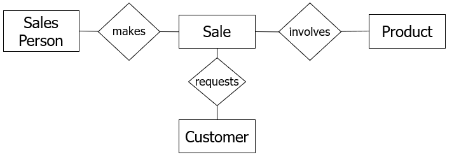
- Sometimes we will be modelling relationships between 3 entities
  - These are known as ternary relationships



- These should not be left in the conceptual model
- They should be replaced with a number of binary relationships before the model is completed

Ternary Relationships

- Typically, we replace the relationship with an entity
- This entity will then have binary relationships with the other types in the original relationship



Optionality of Relationships

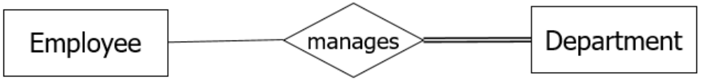
- A relationship can be optional or mandatory.
- If a relationship is mandatory
  - An entity occurrence at one end of the relationship must be related to another entity occurrence on the other end.
  - Also known as **total participation**
- If a relationship is optional
  - An entity occurrence on one end of the relationship may exist without being related to another entity occurrence on the other end.
  - Also known as **partial participation**

Examples of Optionality

- Optionality can be different to the two entities involved in a relationship
- Consider the example of students and courses
    - A student must be registered for a course: this is mandatory
    - A course can exist before any students have enrolled: this is optional

Drawing Optional Relationships

- Mandatory participation in a relationship is shown with a double line.
- Here, every department must have an employee that manages it: mandatory.
- However, not every employee is required to manage a department: optional

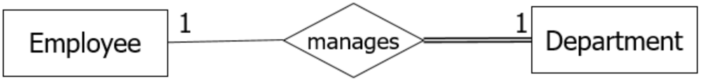


Cardinality of Relationships

- The cardinality of a relationship refers to the number of entity occurrences that are involved in the relationship.
- There are three main types of cardinality, in each case, we talk about the maximum number of other entities that an entity can be related to
  - 1-to-1 (1:1)
  - 1-to-Many (1:N)
  - Many-to-Many (M:N)

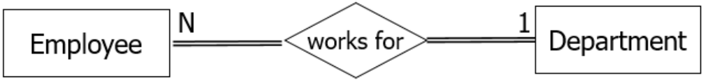
1-to-1

- Each employee can manage one department at most.
- Each department can be managed by one employee at most



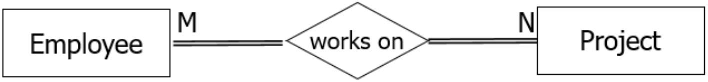
1-to-Many

- Each employee works for one department
- Each department can have many employees working for it



Many-to-Many

- Each employee can work on many projects.
- Each project can have many employees working on it.



Strong and Weak Entities

- Most entity types are strong entity types, they have their own primary key and can exist by themselves.
- However, some entity types are weak entity types, which cannot exist without a relationship to another entity type

Example

- Our company stores data about employee's dependants (spouse and children) for insurance purposes.
- We store the following information about each:
  - Name
  - Sex
  - Relationship to employee
  - Date of birth

Key

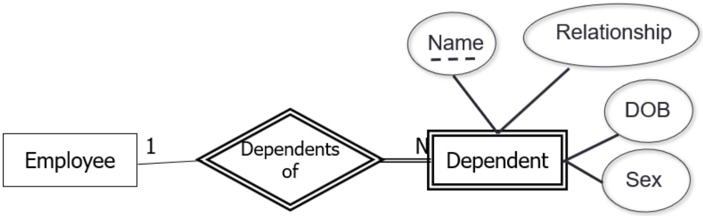
- In this schema, the Dependent entity type has no appropriate key
- Two employees may have a dependent with the same name, sex, date of birth and relationship, even though they are two different people
- We only see them as distinct entities when we know which employee they are associated with (their owner entity)

Partial Key

- For weak entities, we define a partial key. That is, something that can be used as a key if we know which owner entity it is related to.
- If we assume that an employee will not have two dependants with the same name, we can use "name" as a partial key

Weak Entities

- An attribute that is part of a partial key has its named underlined with a dashed line.
- The weak entity's box has a double line.
- The relationship that identifies its owner entity also has a double line.



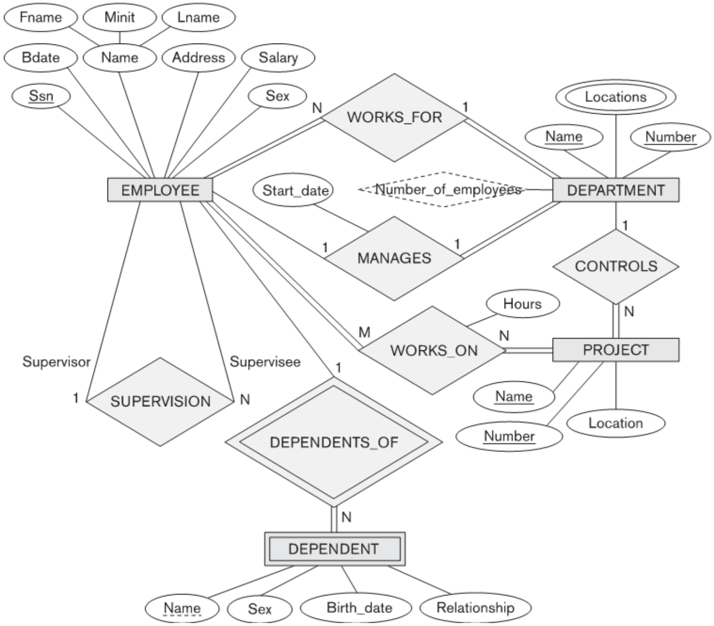

Fname
Minit
Lname
Bdate
Name
Address
Salary
Ssn
Sex
N
WORKS_FOR
1
Locations
Name
Number
EMPLOYEE
Start_date
Number_of_employees
DEPARTMENT
1
1
MANAGES
1
CONTROLS
Hours
N
M
WORKS_ON
N
PROJECT
Supervisor
Supervisee
1
1
SUPERVISION
N
Name
Location
Number
DEPENDENTS_OF
N
DEPENDENT
Name
Sex
Birth_date
Relationship

Table of Contents

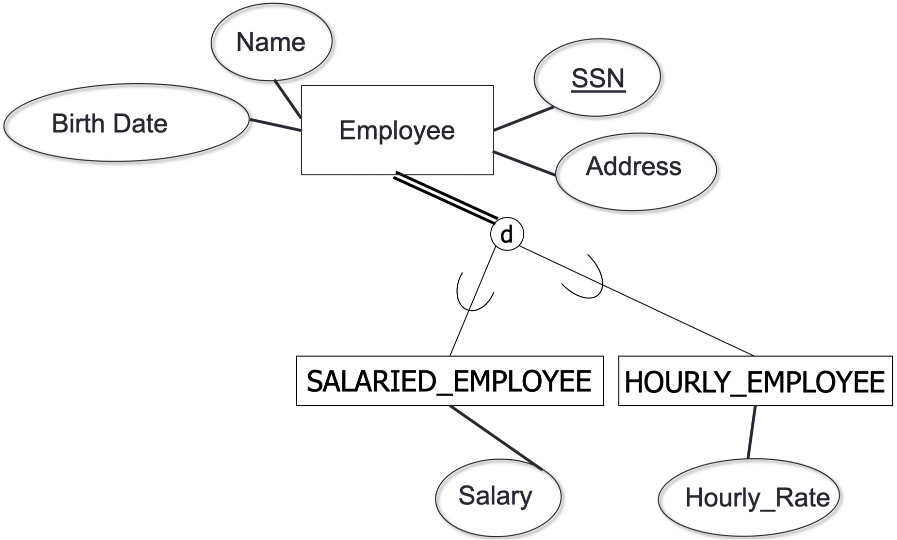
1. Designing an Information System
2. Database Design
3. Conceptual Design
4. Enhanced Entity Relationship Model (EER)
   - Supertypes and Subtypes
5. Creating an E-R Model

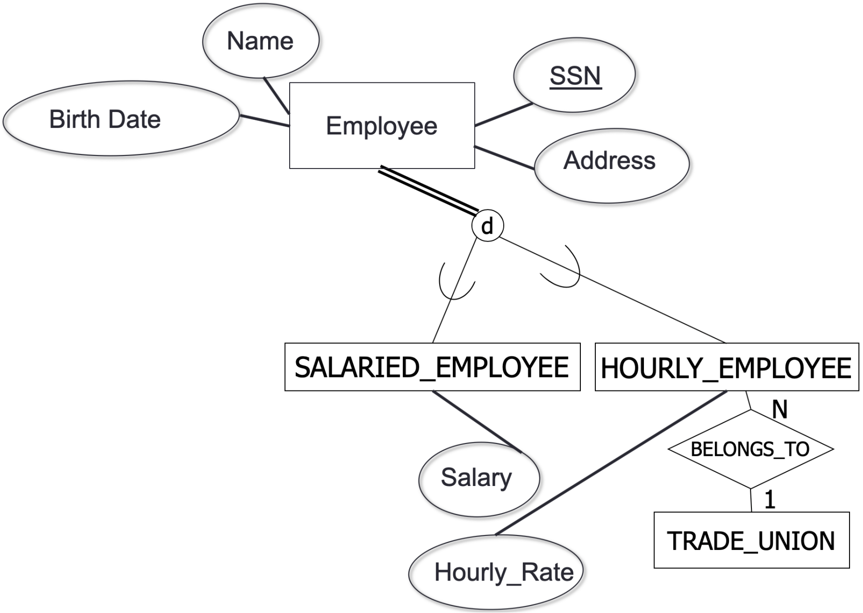
- In terms of data modeling, much progress has been made since the original proposal of the Entity Relationship (E-R) model.
- Particularly in the areas of knowledge representation and object modeling (e.g. for Object Oriented Programming).
- This has led to the development of the Enhanced Entity Relationship Model (EER).
- In particular, this adds the ability to define supertypes and subtypes

- Some entity types have subsets or subtypes that should be represented separately
- This is most commonly seen where
  - A subgroup/subtype has its own specific attributes that are not common to all instances of the entity type
  - A subgroup/subtype has its own specific relationships that are not common to all instances of the entity type

Example

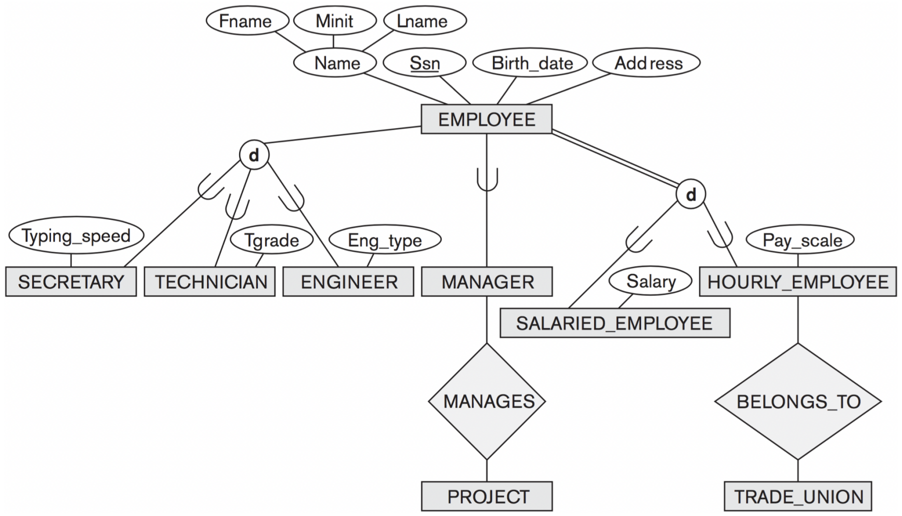
- In a company, we store data about an EMPLOYEE entity type
- Employees can be paid per hour, or using an annual salary
    - A salaried employee has a "salary" attribute that hourly employees do not have.
    - An hourly employee has a "hourly rate" attribute that salary employees do not have.
- This suggests that the "employee" entity type should have two subtypes: `SALARIED_EMPLOYEE` and `HOURLY_EMPLOYEE`


Name
SSN
Birth Date
Employee
Address
d
SALARIED_EMPLOYEE
HOURLY_EMPLOYEE
Salary
Hourly_Rate


Name
Birth Date
Employee
SSN
Address
d
SALARIED_EMPLOYEE
HOURLY_EMPLOYEE
N
BELONGS_TO
1
Salary
Hourly_Rate
TRADE_UNION

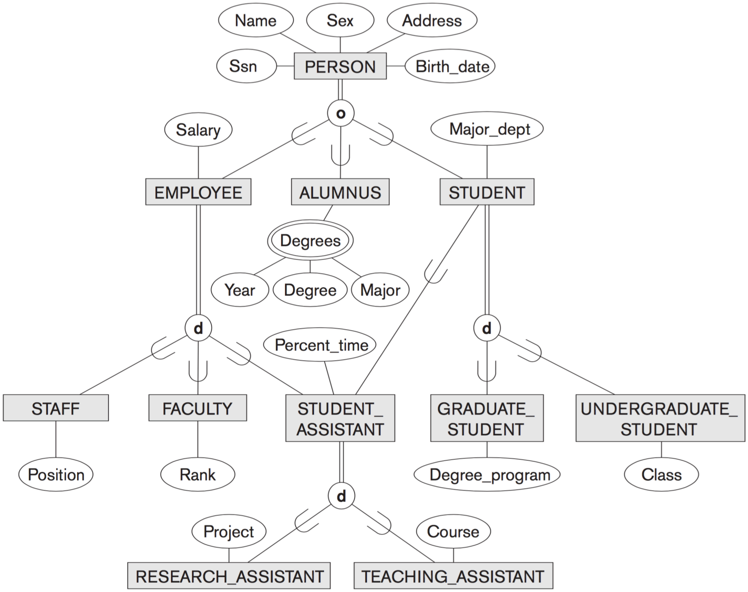
Inheritance

- Every occurrence of a subtype is also an occurrence of the supertype.
- May seem strange: one particular person is represented both by an `EMPLOYEE` entity and a `SALARIED_EMPLOYEE` entity.
- Every property of the supertype (attributes, keys, relationships, etc.) is also a property of the subtype.
- This is known as inheritance (similar to OOP).


Fname
Minit
Lname
Name
Ssn
Birth_date
Address
EMPLOYEE
d
d
Typing_speed
Tgrade
Eng_type
Pay_scale
Salary
SECRETARY
TECHNICIAN
ENGINEER
MANAGER
HOURLY_EMPLOYEE
SALARIED_EMPLOYEE
MANAGES
BELONGS_TO
PROJECT
TRADE_UNION

Disjoint and Overlapping Subtypes

- The subtypes we have seen so far have all been disjoint: meaning that a particular entity (e.g. an employee) can only belong to one of a group of subclasses.
- An alternative is to have overlapping subtypes, where an entity could be a member of multiple related entity types.
- For example, a student who does work in a university may be a member of a PERSON supertype, but be both a STUDENT and an EMPLOYEE


Name
Sex
Address
Ssn
PERSON
Birth_date
o
Salary
Major_dept
EMPLOYEE
ALUMNUS
STUDENT
Degrees
Year
Degree
Major
d
Percent_time
d
STAFF
FACULTY
STUDENT_ ASSISTANT
GRADUATE_ STUDENT
UNDERGRADUATE_ STUDENT
Position
Rank
Degree_program
Class
d
Project
Course
RESEARCH_ASSISTANT
TEACHING_ASSISTANT

Specialisation and Generalisation

- Specialisation and Generalisation are processes that can be used to identify supertypes and subtypes.
- For specialisation, we begin with a supertype, and find any attributes/relationships that are particular to certain members. Groups of members with common attributes/relationships become subtypes.
- Generalisation is the opposite, where we begin with specific types, and identify attributes and relationships they have in common, so as to create a supertype

Table of Contents

1 Designing an Information System

2 Database Design

3 Conceptual Design

4 Enhanced Entity Relationship Model (EER)

5 Creating an E-R Model
- Example Requirements for a Database
- Glossary of Terms
- General criteria for data representation
- Trainee Section

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 5000), identified by a code, we want to store the social security number, surname, age, place of birth, employer's name, address and telephone number, previous employers (and period employed), the courses attended (there are about 200 courses) 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 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 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 (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 can be freelance

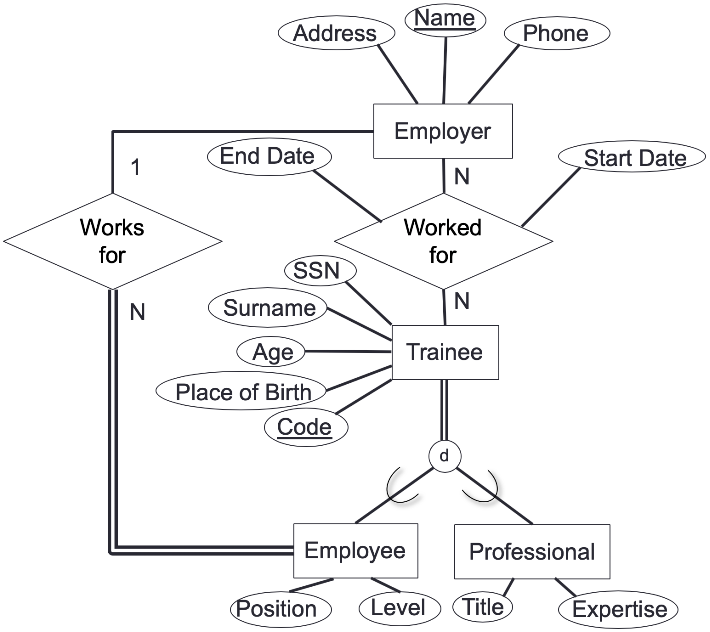
It can be useful to turn this natural language description into a glossary of terms, which helps to describe the concepts we want to store data about, and their relationships to one another

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

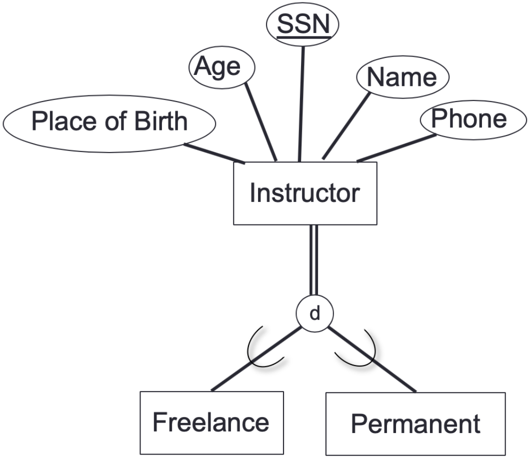
- If a concept has significant properties and/or describes classes of objects with an autonomous existence, it is appropriate to represent it by an entity.
- If a concept has a simple structure, and has no relevant properties associated with it, it can be represented by an attribute of another concept to which it refers.
- If the requirements contain a concept that provides a logical link between two (or more) entities, this can be represented by a relationship.
- If one or more concepts are particular cases of another concept, it is appropriate to represent them by means of a supertype/subtype relationship.

- Because E-R diagrams can get quite large, we do not need to squeeze everything into one diagram.
- We can divide our work into three sections:
  - Trainee section.
  - Course section.
  - Instructor section.
- After completing these, we can draw a diagram to connect these together.
- We will finish with 4 diagrams in total

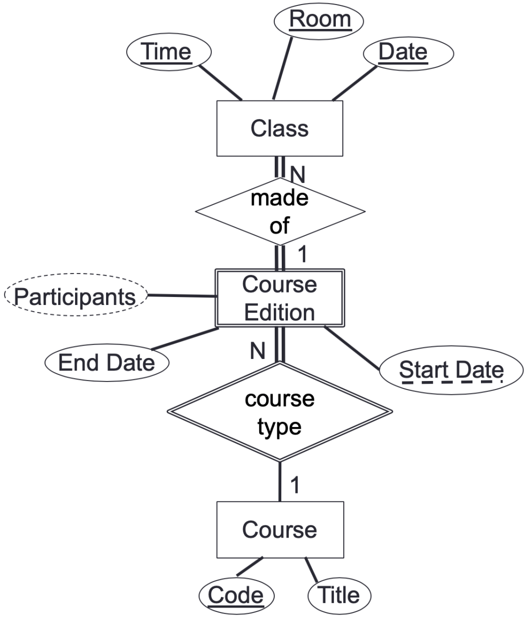
For each course participant (about 5000), identified by a code, we want to store the **social security number**, **surname**, **age**, **place of birth**, **employer's name**, **address** and **telephone number**, **previous employers** (and **period employed**), the courses attended (there are about 200 courses) and the final assessment of each course.
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**.


Address
Name
Phone
Employer
End Date
Start Date
1
N
Works for
Worked for
N
SSN
Surname
Age
Place of Birth
Code
Trainee
N
d
Employee
Professional
Position
Level
Title
Expertise

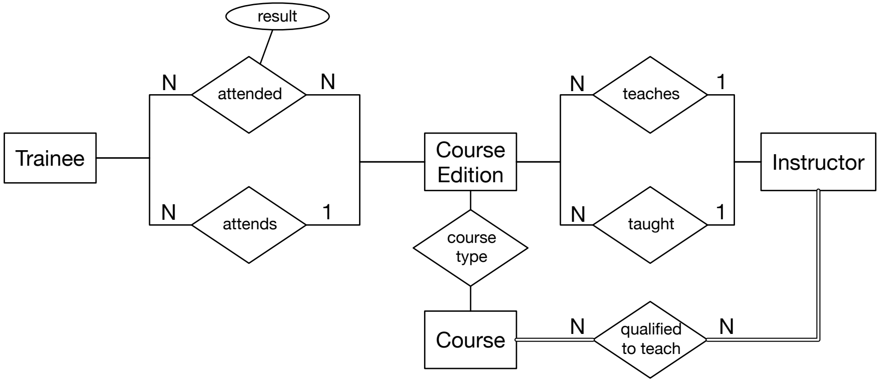
For each instructor (about 300), we will show the **name**, **age**, **place of birth**, the **edition** of the course taught, those **taught in the past** and the courses that the tutor is **qualified to teach**. All the instructors' **telephone numbers** are also stored. An instructor can be **permanently employed** by the training company or can be **freelance**


SSN
Age
Name
Place of Birth
Phone
Instructor
d
Freelance
Permanent

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**.


Room
Time
Date
Class
N
made of
1
Participants
Course Edition
End Date
N
Start Date
course type
1
Course
Code
Title

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 5000), identified by a code, we want to store the . . . the **courses attended** (there are about 200 courses) 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. For each **instructor** (about 300), we will show . . . the **edition of the course taught**, those t**aught in the past** and the courses that the tutor is **qualified to teach**.


result
N
attended
N
Trainee
N
attends
1
Course Edition
course type
Course
N
teaches
1
N
taught
1
Instructor
N
qualified to teach
N