# Data Modelling

A data model is a language or a set of concepts that can be used to describe the structure of the database and may describe data at any of three levels – e.g. ER / Relations / Files

Reminder – easily confused terms

- A **schema** (plural **schemata**) is the description of a database in terms of the constructs provided by the data model.
- A **data model** is the language with which schemata are described.
  - Note - the language could be graphical or textual.
- **Meta-data** is the information contained in the schemata - literally data about data.
- The **catalogue** or **data dictionary** is the set of files which hold the meta-data.



## Semantic Data Models

These were introduced from the early 1970's onwards to describe the database in quasi-real world terms - i.e. as objects and the relationships between them

They are usually textual descriptions, but often come with graphical interfaces

Increasingly, they appear as front-ends to DBMS - as **schema editors**

- e.g. Relationship View in Microsoft Access

They vary in:

- the kinds of **construct** they provide; -
  - some provide constraints and even the ability to describe object behaviour as well as structure
- the particular **flavour** of those kinds -
  - different kinds of entity or relationship may be allowed

and

- the degree of **detail** which can be described



## The Entity Relationship Model

The Entity Relationship (E-R) model describes all data as belonging to one of three categories:

- A set of **entity types**
  - such as: *Student*, *Book*, *Course*
- A set of **relationship types** between the entity types, such as:
  - Student *borrows* Book
  - Student *attends* Course
- For each entity type and relationship type, a set of **attributes**
  - such as: *name*, *address*, *lengthOfLoan*



## An Example

**Key Slide**

The following data requirements are identified for a company :

- The company has a set of departments
- Each department has a name, number, manager and possibly several locations
- The manager is an employee and started managing the department on a given date
- A department controls several projects, each with a name, number and location
- Each employee has a name, address, salary, supervisor, department, sex, date of birth and national insurance number
- An employee may work on many projects, not all in their own department, and works for a (potentially different) number of hours on each of these projects
- Each employee has a set of dependants, each with a name, date-of-birth, sex and familial relationship to the employee

## Entities and Properties

An **entity** is an object in the real world that we wish to store data about.

- For example: Glasgow University or Richard Cooper or the IT course or the DBMS module

An entity is likely to have **properties** which describe it. Some examples:

- Glasgow University - name, address, principal, etc.
- Richard Cooper - name, address, staff number, national insurance number, etc.
- the IT course - the title, the director, the set of modules, etc.
- the DBMS module - the lecturer, the meeting room, the times of lecture, etc.



**Key Slide**

## Entity Types

**Person**

Entities are grouped together into **entity types** or **entity sets**

- Thus there will be entity types University, Person, Course and Module

The entities in each type all have the same properties:

- i.e. they have the same attributes and take part in the same relationships

Each Entity Type consists of:

- an **extension** - the set of entities which are entities of this type
- an **intension** - which describes:
  - the **name** of the entity type
  - the **name** and **meaning** of each attribute
  - any **constraints** which should hold for all the instances of this type

The Entity type will also usually have a **key**:

- one (or possibly more) of the attributes which are **unique** for all instances
  - for example a student's matriculation number



## Initial Conceptual Design I

For the Company example, the following entity types are identifiable:

- *Department*
- *Project*
- *Employee*
- *Dependent*

**Note**

Company itself is **not** an entity type - it is the whole database

Identify the properties of each entity type:

- *Department* has
  - *name, number, manager (with start date), locations , projects, employees*
- *Project* has:
  - *name, number, location, controllingDepartment, employees (with hours worked on)*
- *Employee* has:
  - *NINumber, name, address, salary, department, supervisor, supervisees, sex, date of Birth, projects worked on (with hours)*



## Initial Conceptual Design II

*Dependent* has

- *name, sex, date of Birth, relationship, related employee*

**Some points to note**

- Some properties (e.g. locations) must hold more than one value
- Some properties (name, address) may be best stored as a number of component values
- How do we represent *Employees* working on *Projects* and the hours worked?
  - Create composite property *WorksOn* made up of *Project* and *Hours*
- Some of the properties are from one entity type to another, others are not
  - We separate these out, calling properties from one entity type to another **relationships**, the others are **attributes**
- *Dependent* doesn't seem to have a key

Key Slide

## Attributes


There are two kinds of property

- ones which consist of one or more atomic values - these are **attributes**
- ones whose value is another entity - these are **relationships**

Some attributes are **unique** - these are **candidate keys** which could be used to identify the entities and one will be chosen to be the *primary key*.

The attributes of the company database are:

- *Department - name, number, {locations}*
- *Employee - National Insurance Number, **name**, **address**, salary, sex, birthdate,*
- *Project - name, number, location*
- *Dependent - name, sex, DofB, relationship*

**Notes -** *locations* is multi-valued - curly brackets show this
*name* and *address* have component values - bold font shows this

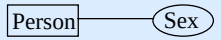


## Different Kinds of Attribute

Key Slide

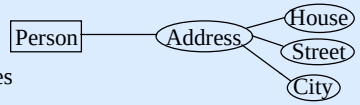
Attributes can be:

- **simple** (**atomic**) - e.g. age, sex
  - these consist of indivisible values




or

- **composite** - e.g. address, date-of-birth
  - these are made up of component values

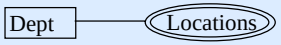



---

- **single-valued** - e.g. age and title
  - these contain just one value

or

- **multi-valued** - e.g. locations - a set of addresses
  - these may contain more than one value




---

- **base valued** - e.g. project number
  - these are explicitly stored in the database.

or

- **derived** - e.g. age
  - these can be calculated by a small piece of program (e.g age from birthdate)




## More on Attributes

If the value of an attribute is unknown, a **null** value must be stored

Attributes are values related to an entity and do not have an independent existence

- Thus if the Glasgow University entity is deleted from the database so is its name, address, etc.

The choice of which attributes are associated with each entity is a modelling decision

The choice of whether to store information as an attribute or a related entity is also a modelling choice

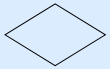
- For example, for an Estate Agent database you might model addresses as entities in their own right

Each simple attribute is associated with a **value set** or **domain**, e.g. integer, which describes all possible values that an attribute may take



Key Slide

## Relationships

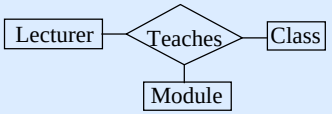


A relationship set or relationship type is a set of associations between the members of two or more entity sets

For example:

- **binary** relationships such as:
  - *Lecturer **works-for** a Department*


- **ternary** relationships such as:
  - *Lecturer **teaches** a Module to a Class*




The number of participating entity types is called the **degree** of the relationship

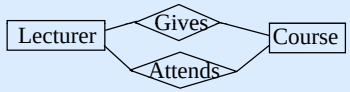
If attributes are used to model a relationship we must choose the direction

- For example, *works-for*. Is this?
  a set of *Lecturers* which is an attribute of *Department*
  or a single *Department* attribute of a *Lecturer*
  A relationship captures both of these ideas
  This is a big difference from using a Java to program the same data

## More on Relationships

Note that there may be more than one relationship between two entity types. For example

- a *Lecturer gives a Course*

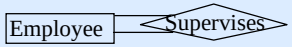



and

- a *Lecturer attends a Course*

Note also that an entity type might be in relationship with itself. For example:

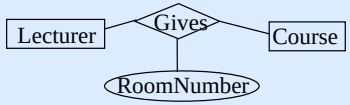
- an *Employee supervises an Employee*
  - This is an example of a **recursive** relationship




Relationships may themselves have attributes:

For instance:

- *Lecturer gives Course*
  - may have the attribute *roomNumber*






Key Slide

## Cardinality Constraints on Relationship Types

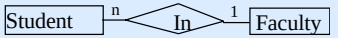
The **cardinality ratio** specifies the number of entity instances that can participate from each side of the relationship. For binary relationships, this can be one of:

**one to one** (1-1)


exactly one instance of each entity type can take part in the relationship

e.g. for each University *Department*, there is one *Head* and that person only heads one *Department*

**one to many** (1-N)




several entities of one type are associated with a single entity of another

e.g. each *Student* is associated with one *Faculty*, but each *Faculty* has many *Students*

**many to many** (M-N)


several entities of one type are associated with several entities of the other.

e.g. each *Student* takes several *Modules*, many *Students* take each module

**Note** - specifying multiple membership does not mean that there **has to be** many entities taking part- just that the database must **cope with** many entities.



## Participation Constraints on Relationship Types

These specify whether the existence of an entity depends on its participation in the relationship.

Key Slide

There are two options:


- **Total Participation**
  - Every member of the type must participate in the relationship
    - e.g. All Lecturers work-for a Department


- **Partial Participation**
  - Not all members need to participate
    - e.g. Some Lecturers head a class

Note: Cardinality and Participation Constraints are jointly called **Structural Constraints**



## Refining the Conceptual Design I

1. Transform properties that represent relationships into **relationship types**
   - ***Manages*** - a 1-1 relationship type between Employee (P) and Department (T)
   - ***Works-for*** - a 1-N relationship between Department (T) and Employee (T)
   - ***Controls*** - a 1-N relationship between Department (P) and Project (P)
   - ***Supervises*** - a recursive 1-N relationship between Employee(P) & Employee(P)
   - ***Works-on*** - an M-N relationship between Employee (T) and Project (T)
   - **Depends-on** - a 1-N relationship between Dependent (T) and Employee (P)
     - P(artial) and T(otal) describe the participation constraints.

**Notes**

- Some of the decisions are clear-cut - not every employee is a manager
- Others are less clear cut – there may be employees who don't work on projects

2. Identify those attributes which are now **attributes of relationship types**
   - *managerStartDate* - the date an *Employee* starts managing a *Department* is an attribute of *Manages*
   - *hours* - the number of hours an *Employee* works on a *Project* is an attribute of *Works-On*

## Weak Entity Types

**Key Slide**

Some entity types do not have keys in their own right, but are dependent upon some other entity type(s) to guarantee their uniqueness

- These are called **Weak Entity Types** which can be contrasted with **Strong Entity Types** -those which do have keys

Dependants only have attributes *name, sex, birthdate, relationship*

- No combination of these can be guaranteed unique
  - For instance, two employees may have a 66 year old aunt called Jane Smith

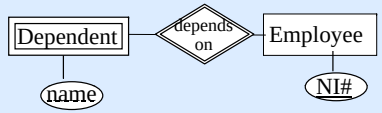
We might however assume that the dependants of each individual employee can be uniquely determined by name, say

- Taken together, the *Dependent name* and the *Employee NInumber* will be unique
- In this case, Dependent name is called a **partial key**
- and Employee is called the **identifying owner**
- and the relationship between Dependent and Employee is called an **identifying relationship**



## Further Refining the Conceptual Design

3. Make *Dependent* a weak entity type
   - and make *depends-on* an identifying relationship.




**Note**

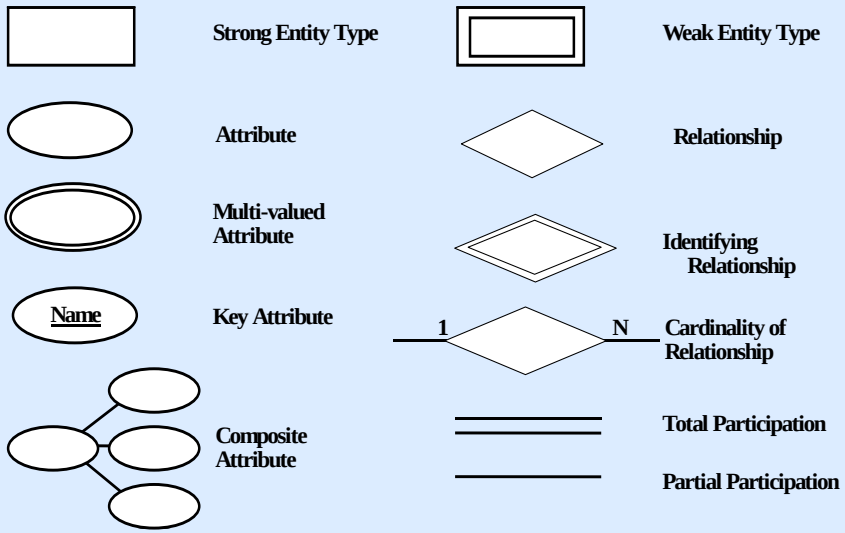
- We now expect that when we remove an Employee from the database we will also removes the Dependents of that employee
  - viz: weak entities do not have an independent existence

We can now summarise the conceptual model using an E-R diagram.

An E-R diagram is a fully connected graph, with different shapes for the main constructs (see next slide for symbols.)



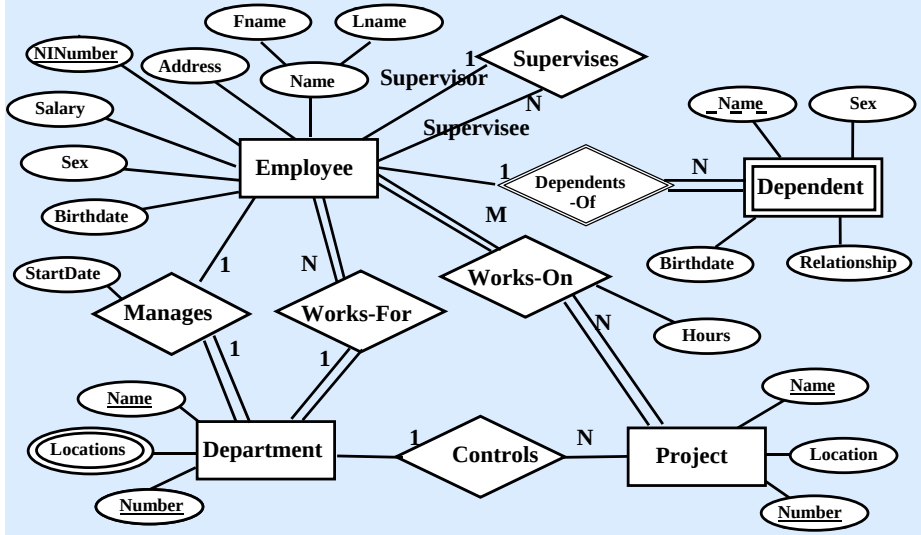
## E-R Diagram Notation



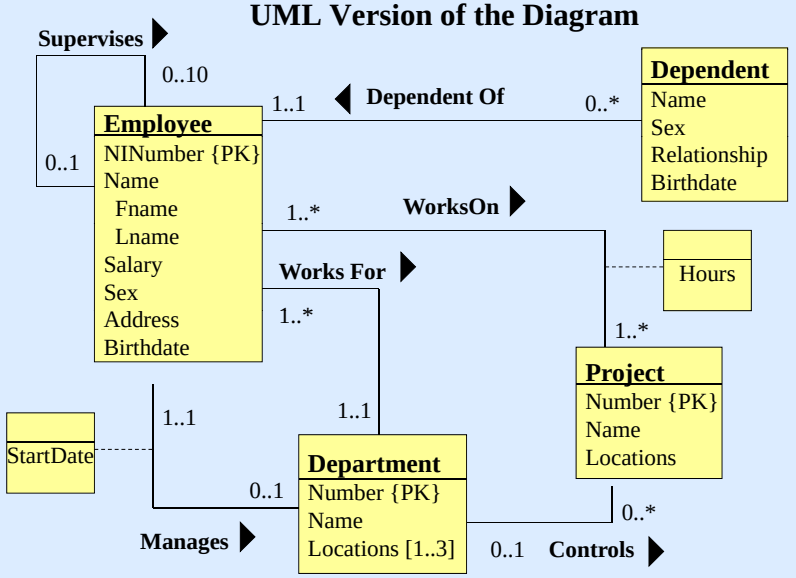



## ER Diagram for the Company Database

**Key Slide**



## UML Version of the Diagram






## Traps in ER Modelling

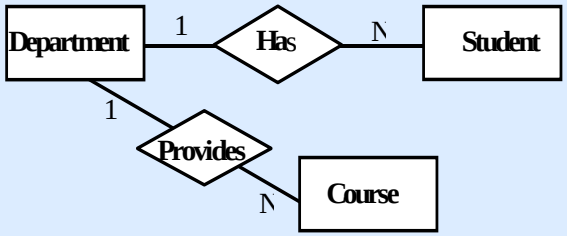
1. Do not **duplicate relationships with attributes**:
   - For instance, there is no *Department* attribute for *Manager* – this is covered by the *Manages* relationship
     - even though this will result in a column in the table
     - **Remember attributes become fields, but not all fields come from attributes**
2. Do not **duplicate relationships** which already exist implicitly
   - For instance, there is no relationship between *Dependent* and *Department* to mean the department of the dependent's employee, since this already there through *DependentsOf* and *WorksFor*
3. However, avoid disconnection through **fan traps** and **chasm traps**



## Fan Traps

These obscure the relationship between two entities

- In the following, each student is attached to exactly one department as is each course, but you cannot tell which courses each student attends.




There needs to be a further relationship between *Student* and *Course* to add this extra information.

Fan traps occur when two supposedly related entity types are connected only through N-1 relationships with some third entity type

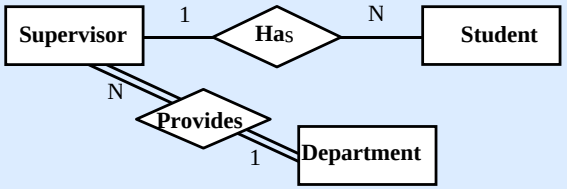
- The solution is to add a new relationship or to change which entity type is at the centre



## Chasm Traps

These imply relationships between entity types which do not hold

- In the following, it has been decided to model the student information using the intuition that a student's department can be inferred through a supervisor




However, not all students have supervisors (partial participation), so what is the department of such students?

- The solution is to put in a direct relationship between student and department