



Topic 05: Conceptual database design

ICT285 Databases
Dr Danny Toohey

About this topic

This is the first of three topics that deal specifically with database design as a process. Database design can be divided into three basic stages: conceptual, logical and physical.

The **conceptual design** phase (this topic) constructs a model of the enterprise's data needs independent of any database model. This ensures firstly that design can progress without the need for compromise due to limitations in the data model, and secondly that the data model most appropriate to the particular enterprise's requirements can be selected when the requirements are more fully known.

Topic Learning Outcomes

After completing this topic you should be able to:

- Explain where conceptual data modelling fits in the database development life cycle and SDLC
- Describe the features of the Entity-Relationship model: entities, attributes, relationships, cardinality, weak entities
- Describe the features of the Extended Entity-Relationship model: supertypes and subtypes
- Interpret different types of Entity-Relationship diagram notation
- Draw an Entity-Relationship Diagram from a description
- Use Entity-Relationship modelling as a means of eliciting and checking user requirements for a system
- Check your Entity-Relationship Diagram for correctness
- Avoid typical 'traps' of poor ER modelling practice

Resources for this topic

READING

- Text, Chapter 5 “Data Modelling with the Entity-Relationship Model”
- Chen, P.P.S., 1976, The entity-relationship model-toward a unified view of data, ACM Transactions on Database Systems, 1(1), pp. 3-36. On My Unit Readings
(a classic paper, but not examinable)

Kroenke, D.M., and Auer, D.J., 2016, Database Processing: Fundamentals, Design and Implementation, 14th Edition, Pearson, Boston.

Lab 05

In Lab 05 you will practice creating several ERDs that demonstrate some of the typical patterns found in data modelling. We will use Microsoft Visio for this. Visio is a drawing tool that can be used to create many different types of diagrams, including ERDs. It is a useful tool for ER modelling as it not only draws the diagram, but allows you to create properties and constraints that apply to it. It therefore helps move us from conceptual design (the ERD) towards logical design (a set of normalised tables).

Topic Outline

1. Where conceptual modelling fits into database design
2. Entity-relationship modelling
 - Entities and attributes
 - Relationships
 - Weak entities and ID-dependent entities
 - Generalisation/specialisation
 - Different notations
3. How to construct an ERD
4. How to check your ERD for correctness (separate document)

Topic 05: Part 01

Where conceptual modelling fits in database design

Database Design

Process of creating a design for a database that will support the enterprise's mission statement and mission objectives for the required database system

Three phases of database design:

- Conceptual database design
- Logical database design
- Physical database design

Conceptual Database Design (this topic)

- Process of constructing a model of the data used in an enterprise, independent of *all* physical considerations
- Data model is built using the information in users' requirements specification.
- Conceptual data model is source of information for logical design phase.
- Conceptual data models can be drawn using Entity-Relationship diagrams, UML, or other modelling techniques

Logical and Physical Database Design (looking ahead to topics 6,7)

Logical design

- Process of constructing a model of the data used in an enterprise based on a specific data model (e.g. relational), but independent of a particular DBMS and other physical considerations.
- Conceptual data model is refined and mapped on to a logical data model.

Physical design

- Process of producing a description of the database implementation in secondary storage.
- Describes base relations, file organizations, and indexes used to achieve efficient access to data. Also describes any associated integrity constraints and security measures.
- Tailored to a specific DBMS system.

Data Modelling

- Data modelling is used to represent what data is of interest to an organisation - the main 'things' of interest, their characteristics, and how they are related to one another
- Together with process models, which represent the activities that go on in an organisation, data models represent the **essential** (implementation-independent) view of what a system does
- Although the data model is only a part of the overall system specification, it has a big impact on the overall usefulness and flexibility of the system
- Data models can be drawn at various levels of detail:
 - High level or **enterprise** model, showing all the data of interest
 - Models for particular applications, which will form the basis for the database design

Why bother with data models?

A data model ensures we understand:

- each user's perspective of the data
- nature of the data itself, independent of its physical representations
- use of data across user views

Main purposes of data modeling include:

- to assist in *understanding* the meaning (semantics) of the data
- to facilitate *communication* about the information requirements

Topic 05: Part 02

Entity Relationship Modelling

Entity-Relationship Modelling

One of the most commonly used techniques for conceptual data modelling

- Entity-Relationship Diagrams (ERDs) are the diagrams produced

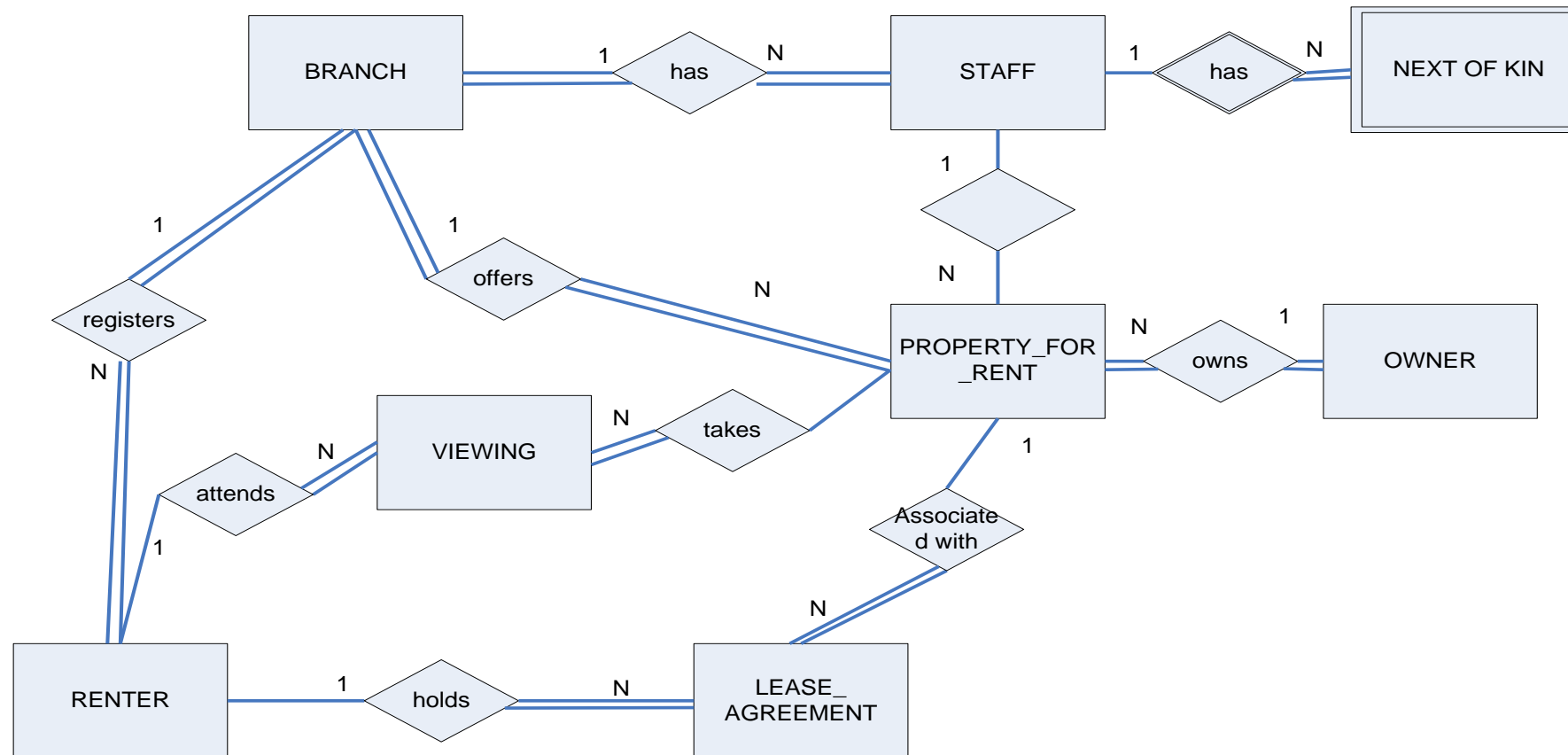
First developed by Chen in the early 1970s

- ‘Grew up with’ the relational model, so a very close match to it and shares many concepts
- Many variations in diagramming techniques and terminology – but the modelling process itself is essentially the same
- The readings illustrate both the original ‘Chen’ notation and the common ‘crow’s feet’ notation

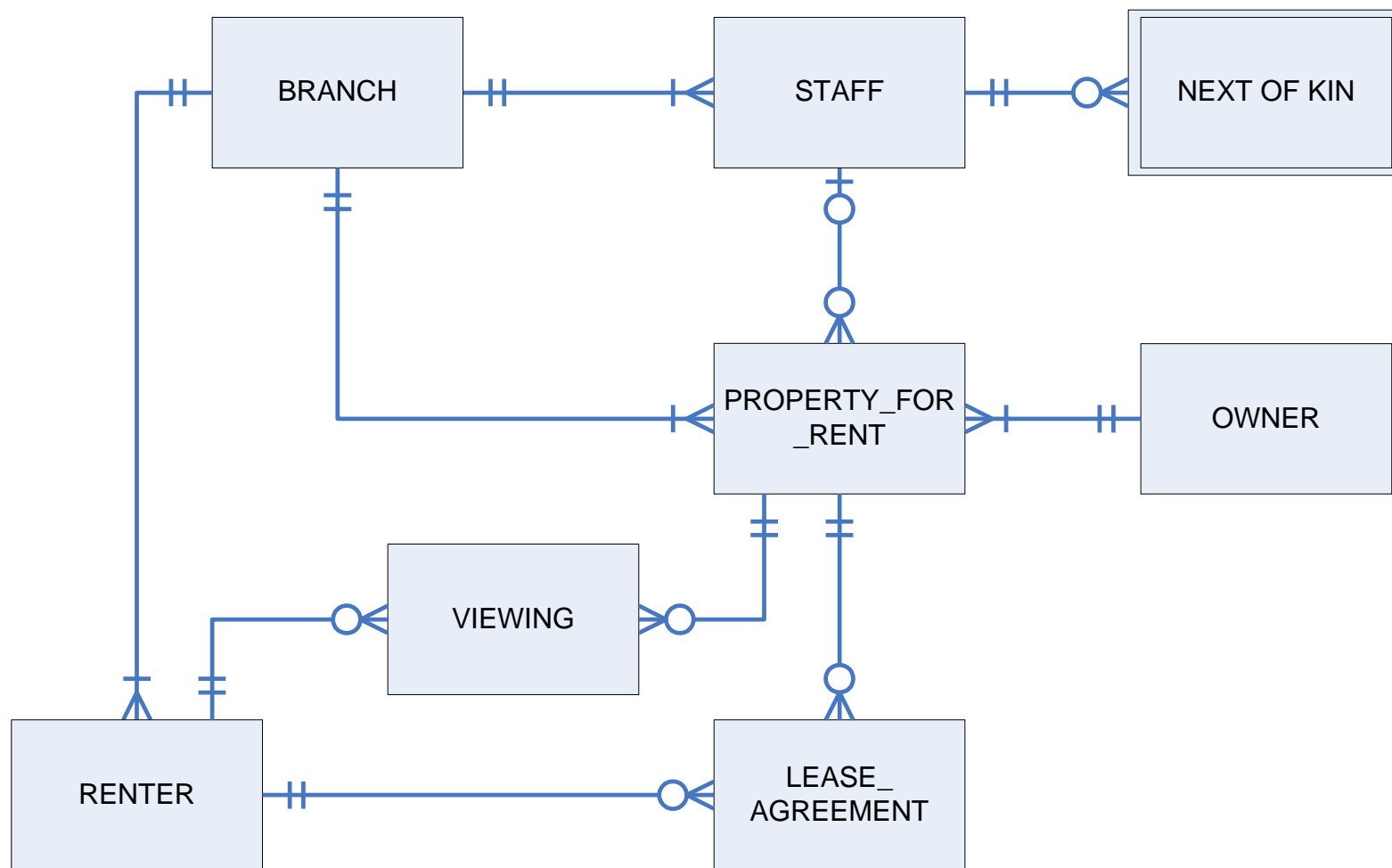
Other types/versions of ER models

- Extended E-R model
 - ...extensions to the Chen model for subtyping
- Information Engineering (IE)
 - by James Martin (1990); it uses “crow’s foot” notation
- IDEF1X
 - a national standard developed by the National Institute of Standards and Technology (see online appendix E)
- Unified Modeling Language (UML)
 - by the Object Management Group
 - developed to support object-oriented methodologies (see online appendix D)

ER Diagram: Chen notation (without attributes)



ER Diagram: Crow's Feet



What E-R Modelling is...

- A concise way of representing the data requirements of the problem at hand
- A way of finding out what you know and don't know about that problem
- A means of discussing the problem domain with the client, during requirements determination
- A means of producing a diagram that can be implemented more or less directly, especially as a relational database
- Straightforward to learn, with a fairly small set of rules and syntax
- One of the most useful skills that you can have!

And what E-R Modelling is NOT

- .. It is NOT something you do after you've built the database, because you need to produce documentation
- .. It is NOT something that you do because you have been told to create an ERD, but which you don't look at again for the rest of the SDLC
- .. It is NOT something in which the occasional inaccuracy or inconsistency doesn't matter, because it's "just" a diagram

Entity-Relationship Modelling – entities and attributes

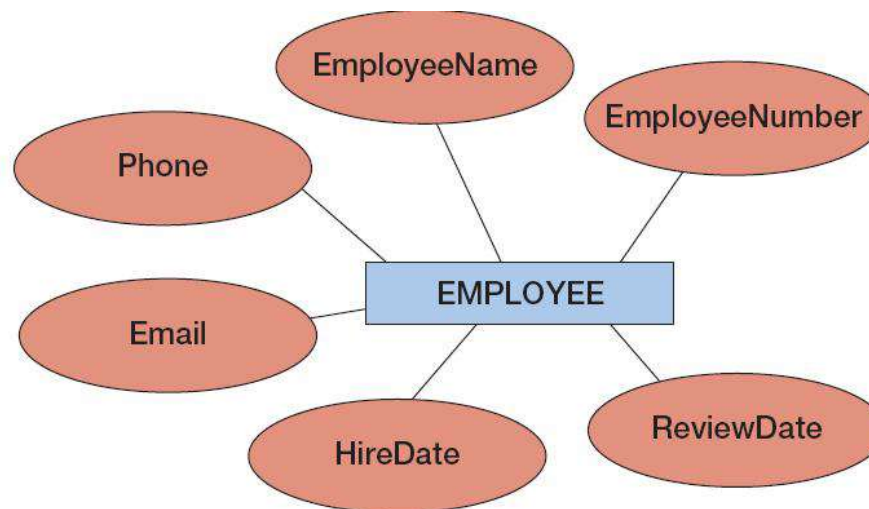


Entities and attributes

- An entity is a something of *relevance* to the system being modelled – something you are interested in storing information about
- It has *existence* – either physical or abstract
- It has properties – characteristics that describe it, which we call *attributes*
- It has many individual *instances* that make it up
- Each instance can be *identified* uniquely

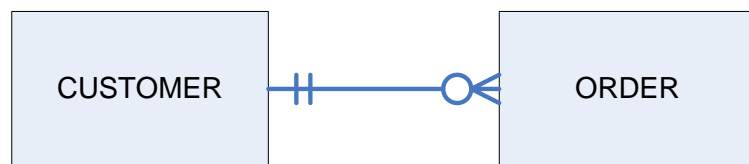
Entities and attributes - example

- EMPLOYEE could be an entity in a data model for a Human-Resources system
- EMPLOYEE would represent the set of all individual employees in the system
- Each EMPLOYEE could be described by ATTRIBUTES such as name, address, phone, etc



Drawing entities and attributes

- Entities are drawn with a rectangle and named with a simple, **singular noun**.
 - e.g. EMPLOYEE, **NOT EMPLOYEES**
 - The name is always singular as it reduces confusion when determining relationship cardinality (later)
- Attributes are shown on the diagram in some notations, not in others (but if not shown, they need to be documented elsewhere)



EMPLOYEE	
PK	<u>EmployeeID</u>
	Title
	FirstName
	LastName
	Email
	StreetAddress
	City
	MobilePhone

Identifier attributes

- There is always one attribute, or a combination of attributes, whose value is unique and which we can use to identify the individual entity instance
- This is exactly the same concept as you have met in creating and normalising relations
- In our example EmployeeID is unique and is called the *identifier* of the entity EMPLOYEE
- Strictly, entities have identifiers, tables have primary keys – though we often refer to the PK of an entity
- Identifiers can be COMPOSITE; i.e., made up of more than one attribute

Entities, tables, relationships and foreign keys

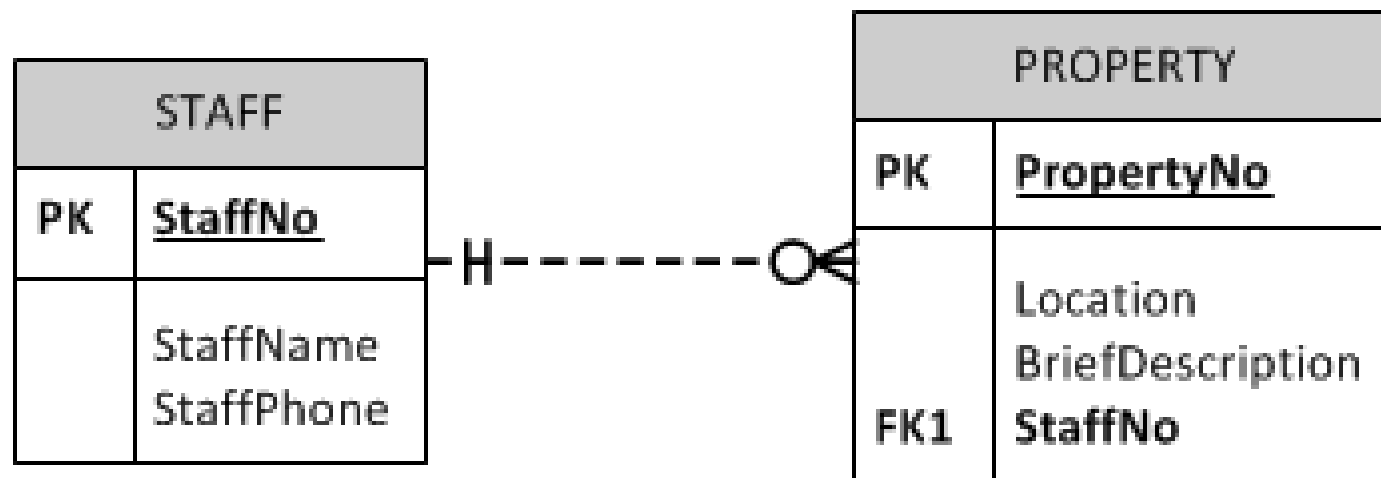
- The principal difference between an entity and a table (relation) is that you can express a relationship between entities without using foreign keys – the line shows the relationship
- This makes it easier to work with entities in the early design process where the very existence of entities and the relationships between them is uncertain
- However, the tables you create from the entities later will definitely have foreign keys to represent the relationships in the ERD

Primary keys and foreign keys in ERDs

- There are different “schools of thought” on whether FKs should be included in an ERD, or if they should be left until the logical design stage
- If the ERD is going to be turned into a relational database (which it usually is) then including PKs and FKs is a good way of checking your conceptual model before creating the schema
- In this unit you WILL include PKs and FKs in the fully-attributed ERDs you create for assignments
- Some software (e.g. Visio 2010, though not 2016) will insert the FKs automatically where there is a corresponding PK in a related entity

Entities and attributes

This notation is the one used in Visio 2010, which shows the attributes, primary keys and foreign keys within the entity symbol - not all notations would do this



Entity types and entity instances

- Formally, what we draw in an ERD is the *entity type* (or entity set)
EMPLOYEE
 - ‘Entities’ are, properly speaking, the individual instances
Fred, Ann
- However, the term ‘entity’ is so widely used for entity type that we shall continue to use it that way here, and when we need to refer to the individual instances of the entity we will call them *entity instances*

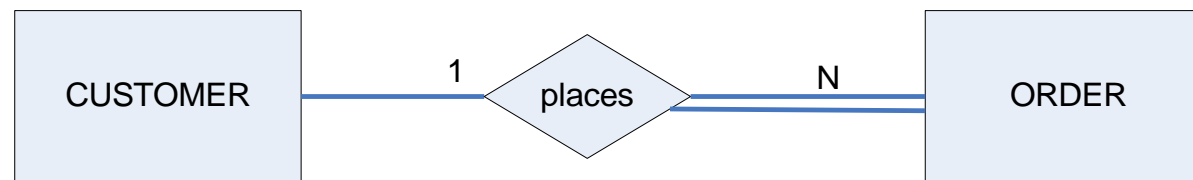
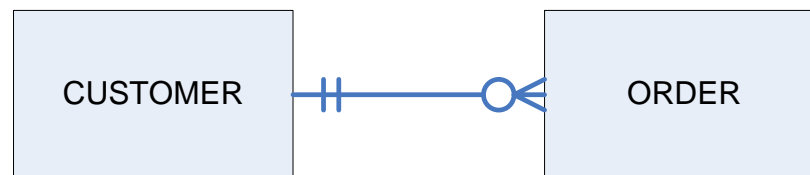
Entity-Relationship Modelling –relationships

Relationships

- Entities can be associated with one another in relationships
- A relationship can involve one, two (most usually), or more entities
- The way in which entities are associated with each other is what represents the *meaning* of the data stored, how it is related to other data:
 - A CUSTOMER places an ORDER
 - A STUDENT enrolls in a UNIT OFFERING
 - An EMPLOYEE works in a DEPARTMENT

Drawing relationships

- A relationship between two entities is shown as a line linking the entities
- Again, there are different conventions for how relationships are drawn



Implementing relationships in database tables: looking ahead

- Looking ahead: the relationships we show on the ERD are eventually **implemented** as links between tables (primary key – foreign key), so that related data can be accessed
- So it's vital that the relationships are defined correctly for the particular system, otherwise you won't be able to use queries get at the data you need

View Ridge Gallery ERD

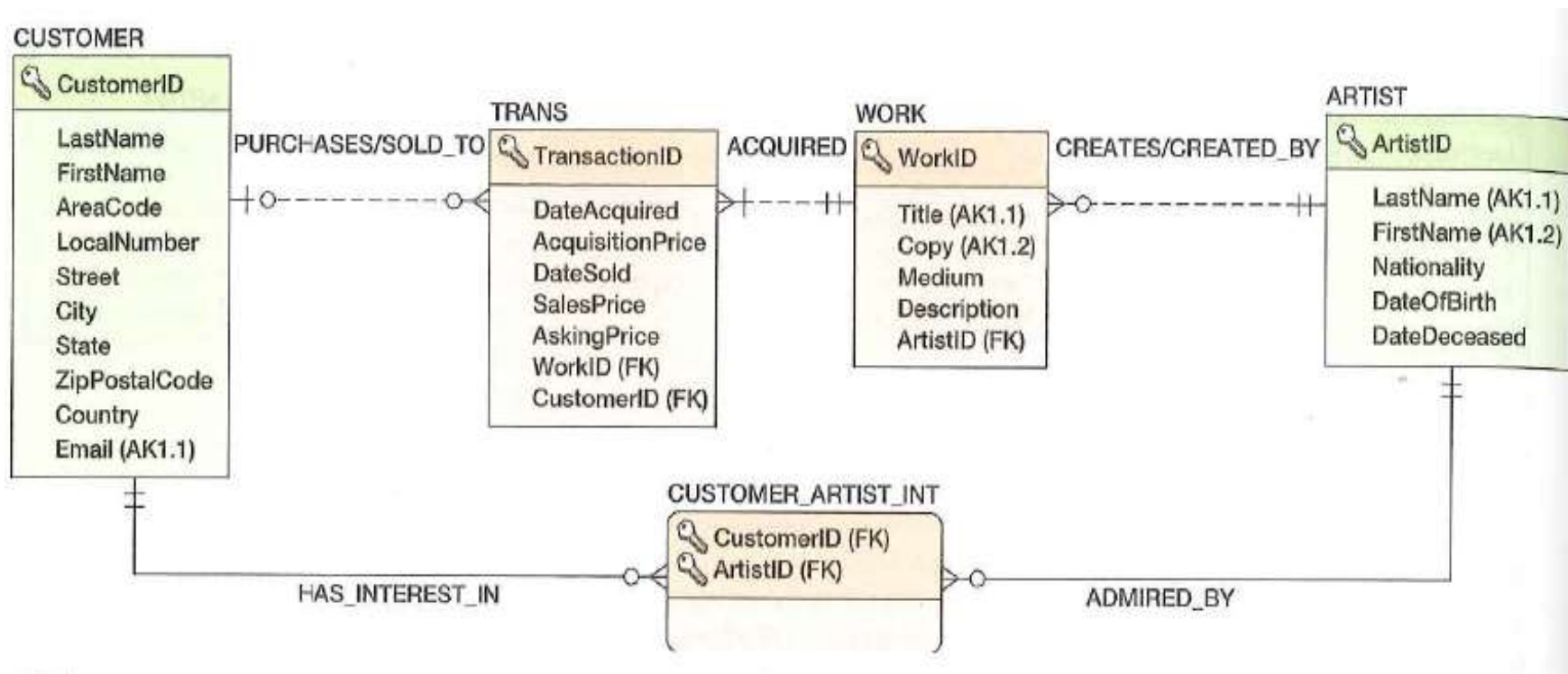
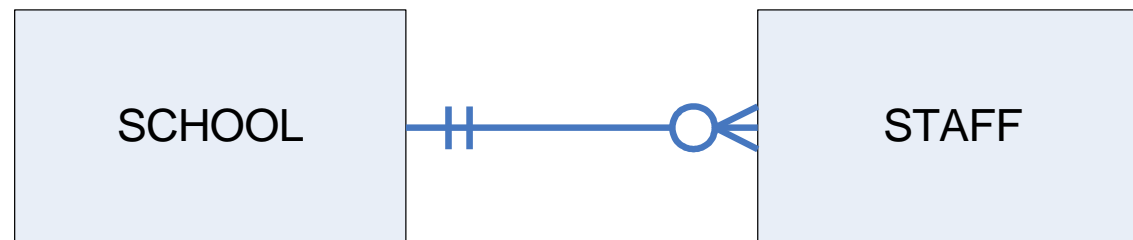


Image from Kroenke, D.M., and Auer, D.J., 2016, Database Processing: Fundamentals, Design and Implementation, 14th Edition, Pearson, Boston.

How many?

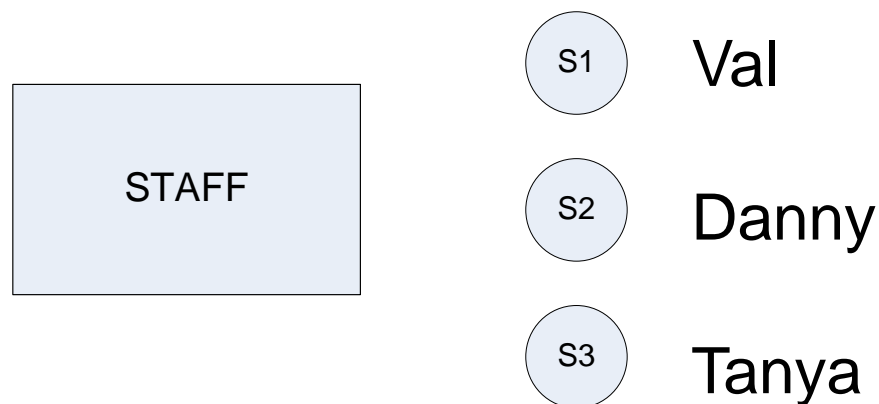
More information about the way entities are related can be gained from considering **how many** of one entity instance are associated with how many of the other

- e.g. At Murdoch, each SCHOOL can have many STAFF members in it. However, each STAFF member belongs to only one SCHOOL
- Thus, the relationship between SCHOOL and STAFF is **one-to-many**:

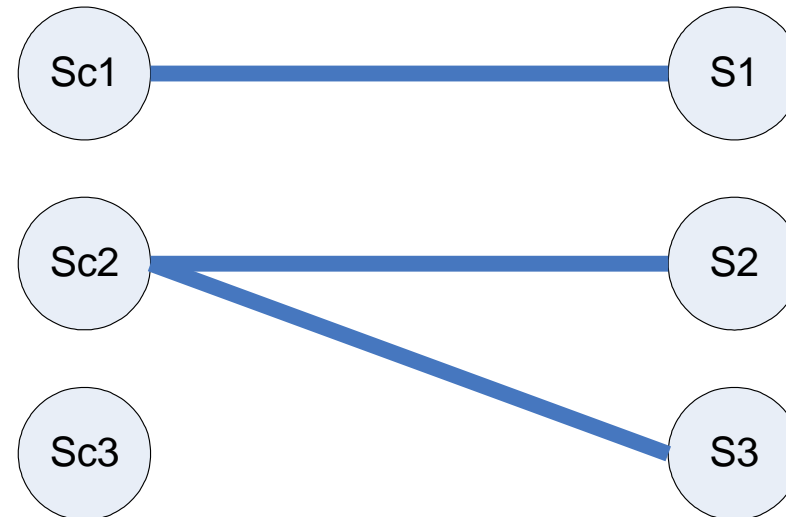
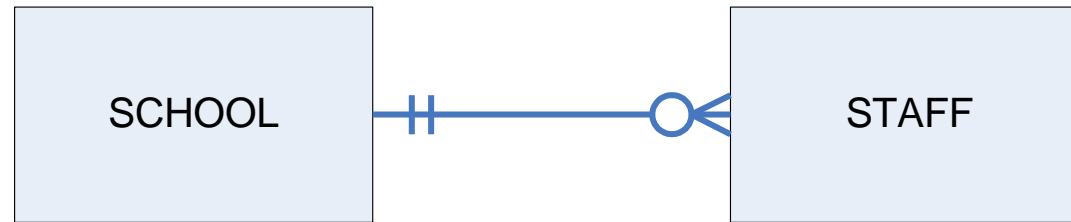


Occurrence diagrams

- ER diagrams show only the collective class of entities (STAFF), not the individual occurrences of the entities (Danny, Val, Tanya)
- However, you can also draw occurrence diagrams (also called *semantic net diagrams* or instance diagrams) that show individual entity instances.
- Semantic net diagrams can be very useful in clarifying exactly what is meant by the cardinality of a relationship – how many instances of one entity are related to an instance of another



Example – Occurrence diagram

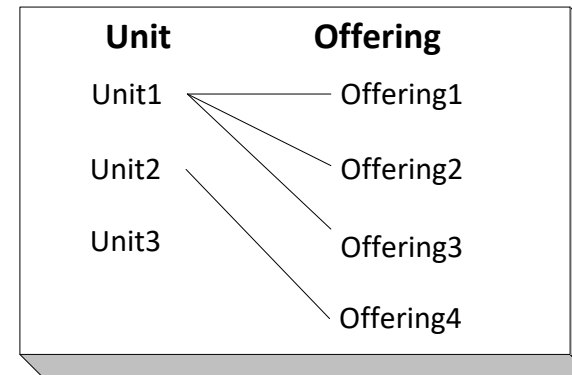


Maximum Cardinality

- The **maximum cardinality** of a relationship describes the **maximum** number of occurrences of one entity that are associated with a single occurrence of the other
 - This is also known in some texts simply as cardinality

Maximum Cardinality

- The occurrence diagram shows that **one** instance of a **unit** can be related to **zero, one or many** instances of an **offering**.
- Thus the **maximum cardinality** of this relationship is **many**
- We can also see that **one offering** can be associated with **one and only one** unit.
- Thus the **maximum cardinality** of this relationship is **one**



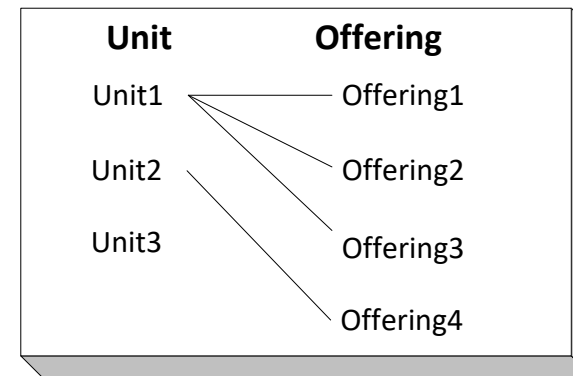
Minimum Cardinality (optionality; participation)

The **minimum cardinality** of a relationship describes the **minimum** number of occurrences of one entity that are associated with a single occurrence of the other

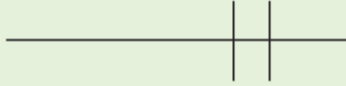

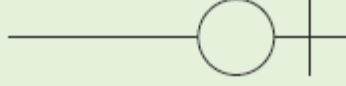
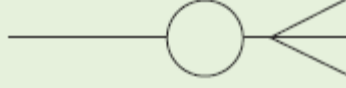
- This is also known in some texts as optionality, ordinality, or participation
- Minimum cardinality of 0 means an **optional** relationship
- Minimum cardinality of 1 means a **mandatory** relationship

Minimum Cardinality

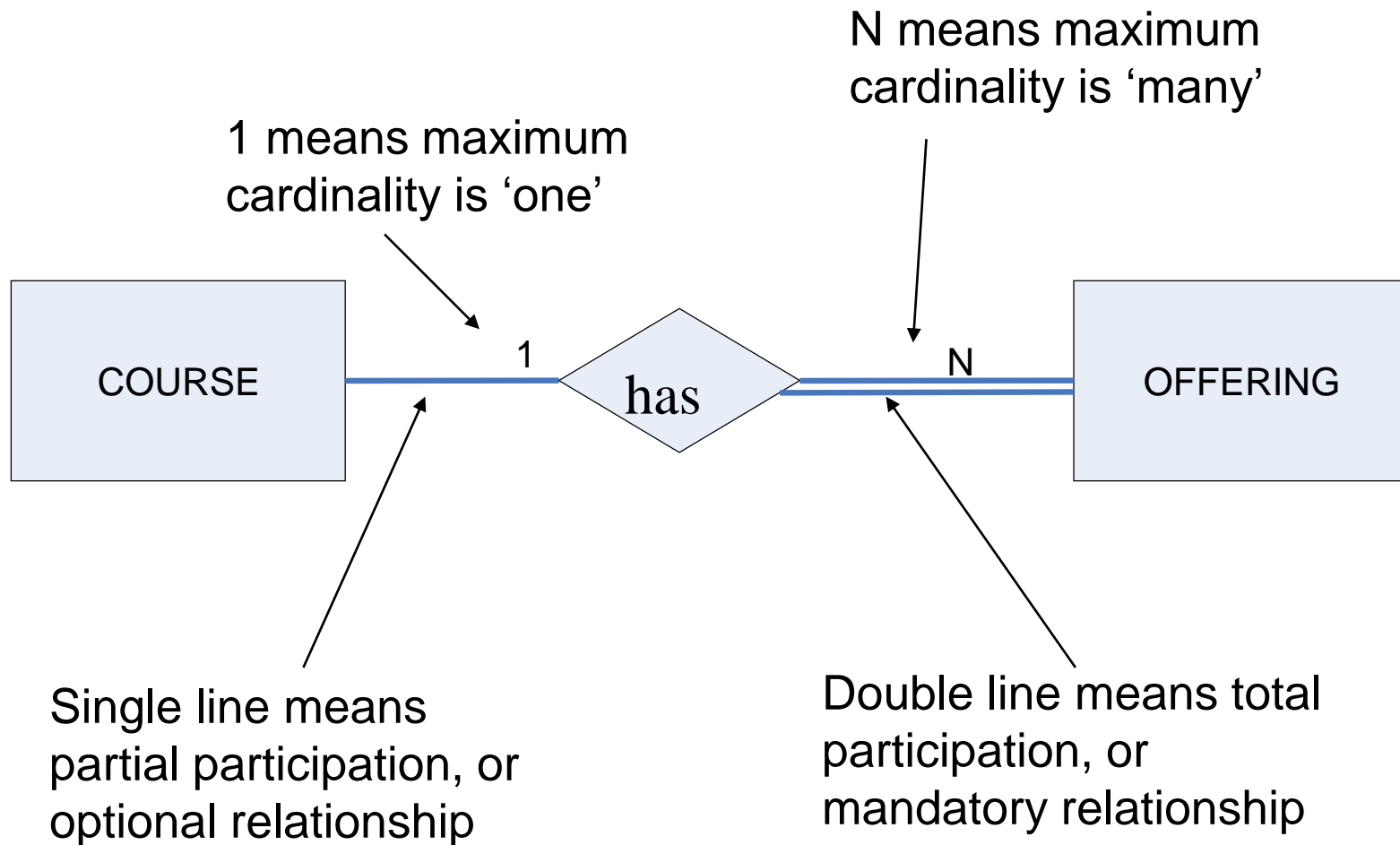
- This occurrence diagram shows that **one** instance of a **UNIT** can be related to **zero, one** or **many** instances of an **OFFERING**.
- Thus the **minimum cardinality** of this relationship is **zero (i.e. it is OPTIONAL)**
- We can also see that **one OFFERING** can be associated with **one and only one UNIT**.
- Thus the **minimum cardinality** of this relationship is **one (i.e. it is MANDATORY)**



Representing Cardinality: Crow's Feet

Symbol	Meaning
	Mandatory – One
	Mandatory – Many
	Optional – One
	Optional – Many

Representing Cardinality: Chen



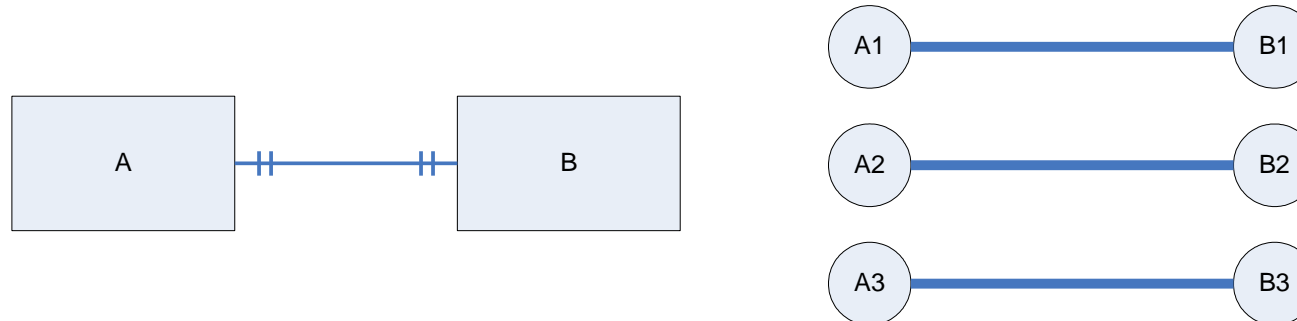
Types of Cardinality

The basic types of cardinality are:

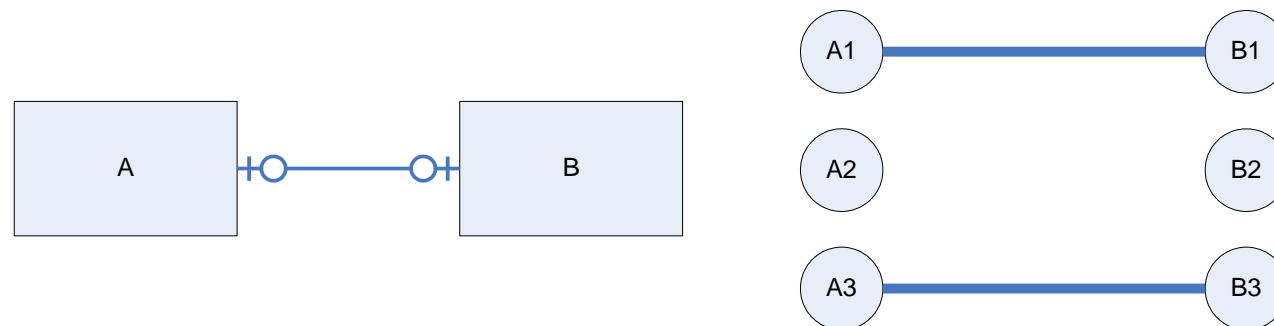
- One-to-one (1:1)
- One-to-many (1:N)
- Many-to-many (M:N)

1:1 relationship

Mandatory relationship on both sides:

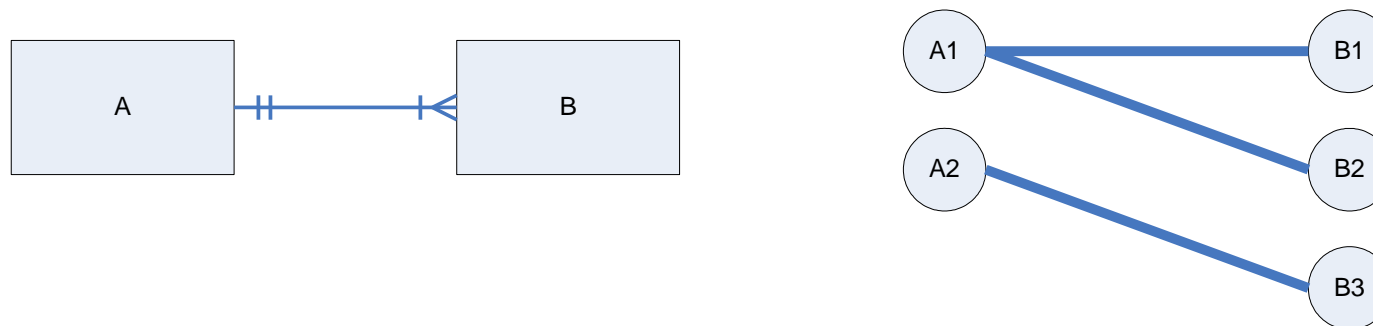


Optional relationship on both sides:

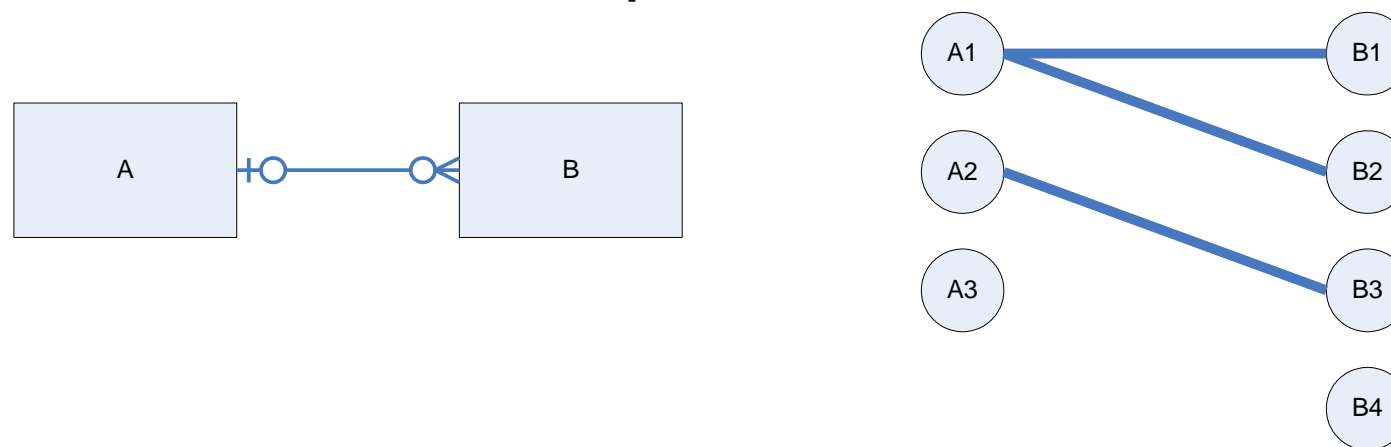


1:N relationship

Mandatory relationship on both sides:

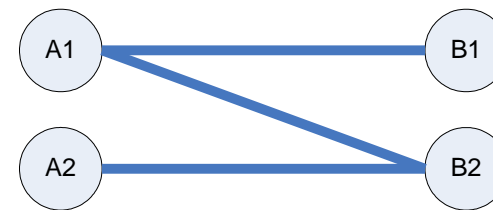


Optional relationship on both sides:

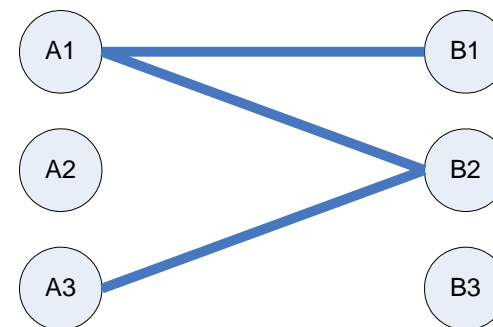
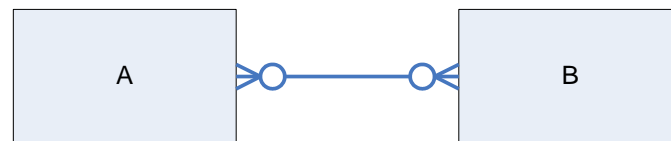


M:N relationship

Mandatory relationship on both sides:



Optional relationship on both sides:



Reading an ERD



Based on the ERD above, we can say that:

- An Office has zero or one Faculty working in it
- A member of Faculty works in one and only one Office
- A member of Faculty team teaches zero or many Offerings
- An Offering is taught by zero or many Faculty

Example

Draw:

- An occurrence diagram
- An entity-relationship diagram

for the following description:

“A car may be driven by many drivers during the week, or by none. A driver may drive many cars, and always drives at least one”

Example

Draw an ERD for the following description. Explain any assumptions you need to make where there is insufficient information.

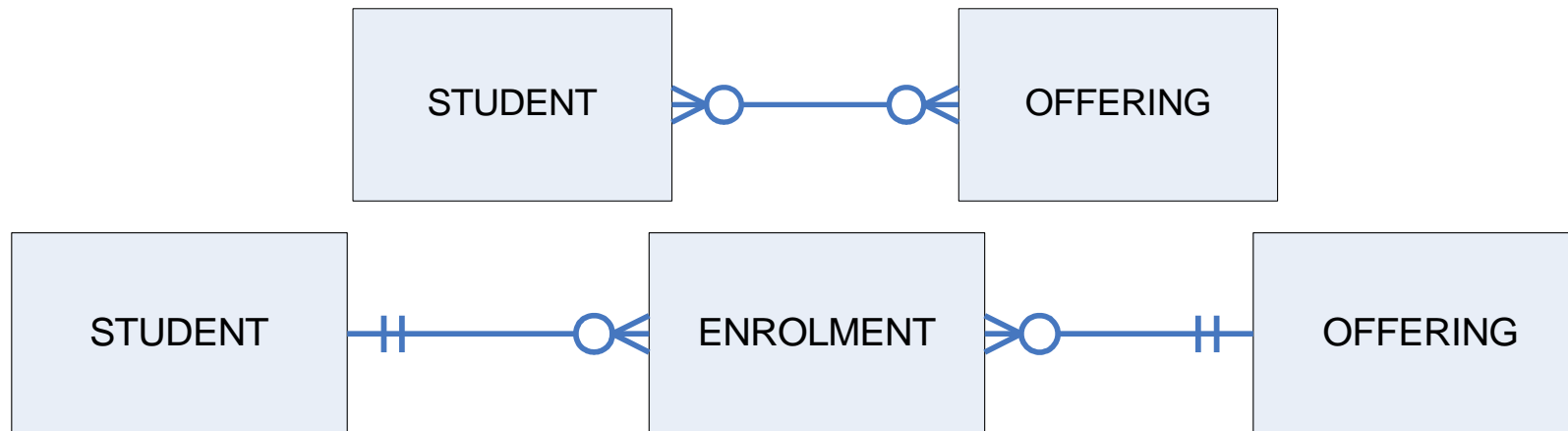
“A cricket team has several players, but a player plays on only one team. A team has one manager”

Many-to-many relationships

- M:N relationships are sometimes called non-specific relationships
- They can always be resolved into *pairs of 1:N* relationships, and this should be done during modelling, before the ERD is mapped to relational tables
 - because we can't implement a M:N relationship directly as tables
- Resolving the relationship gives more precision to the model, and may reveal more attributes/ relationships than were previously apparent

Resolving an M:N relationship using an associative entity

Create another entity to represent the combination of the participating entities:

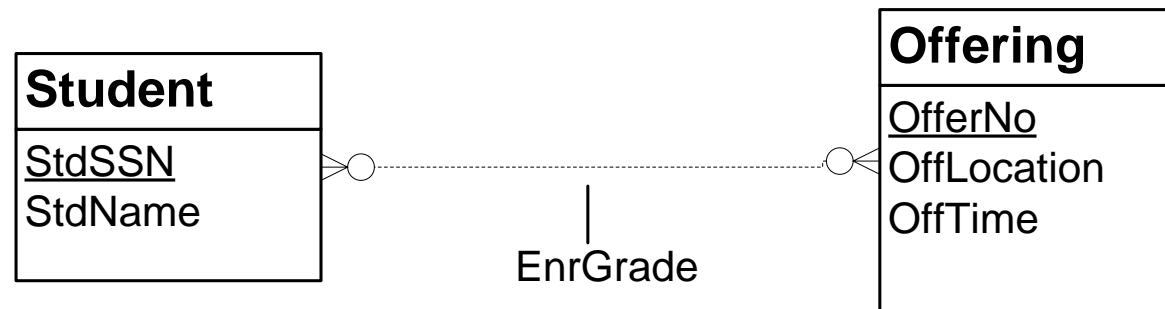


- This is known as an **associative** entity (also called an **association** or **intersection** entity)
- The identifier of the associative entity is the combination of the identifiers from the related entities

M:N Relationships with attributes

In a M:N relationship there may be attributes associated with the **combination** of both entities

- e.g. a grade is associated with a student enrolled in an offering of a unit

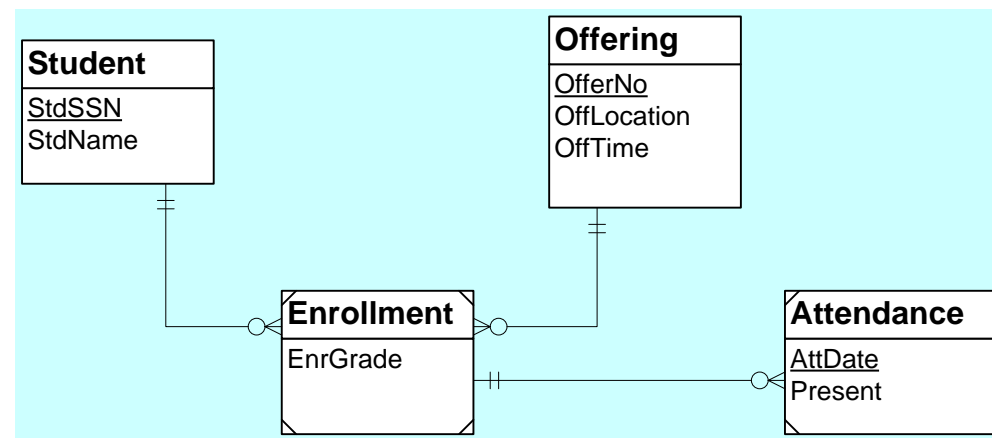


The diagram is not normally left like this, though

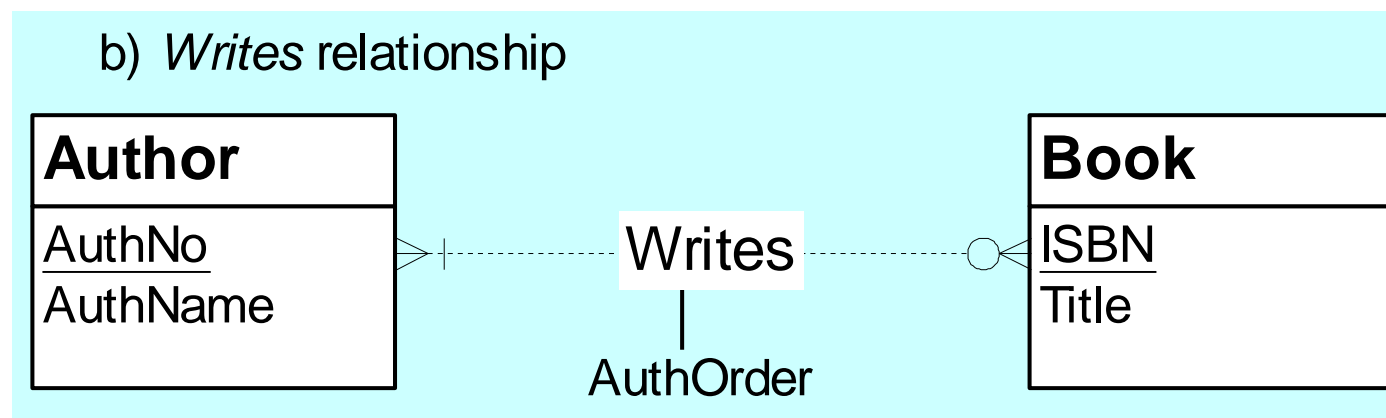
Improved precision

Replacing the M:N with two relationships and an associative entity can improve precision of the model:

- The association may have attributes (e.g. Grade for the Student:Offering relationship)
- The association may need to be related to other entities



Example...



How would this be drawn using an associative (intersection) entity?

Other types of relationships

Other types of relationships you may meet include:

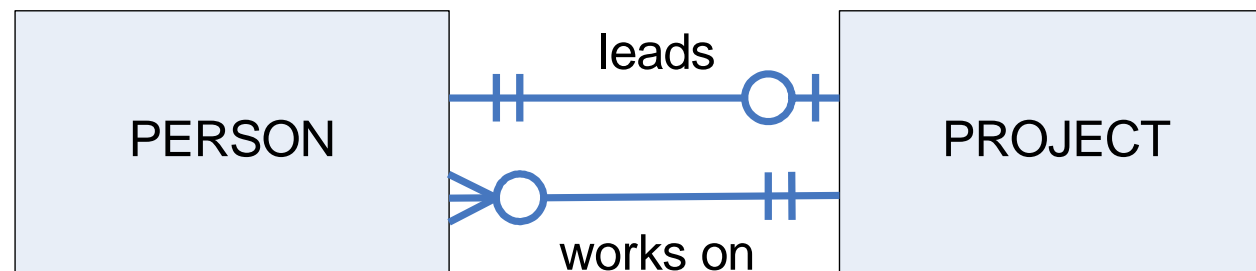
- More than one relationship between the same entities
- Recursive relationships
- Multi-way relationships

More than one relationship between the same entities

This may be necessary to represent the complete meaning of the system:

- PERSON *works on* PROJECT
PERSON *leads* PROJECT
- STUDENT *borrow*s LIBRARY BOOK
- STUDENT *recall*s LIBRARY BOOK

Model with separate relationships, as they may have different maximum and minimum cardinalities



Recursive Relationships

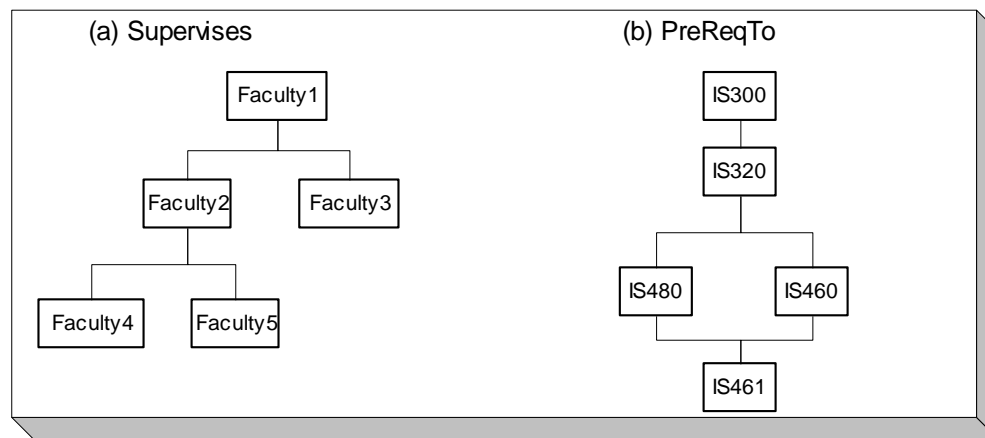
- A recursive relationship occurs when an entity has a relationship with itself
 - Also known as **unary** or **self-referencing** relationships

Examples:

- A member of faculty may supervise other members of faculty and, in turn be supervised by other members of faculty
- A unit may be a prerequisite for other units

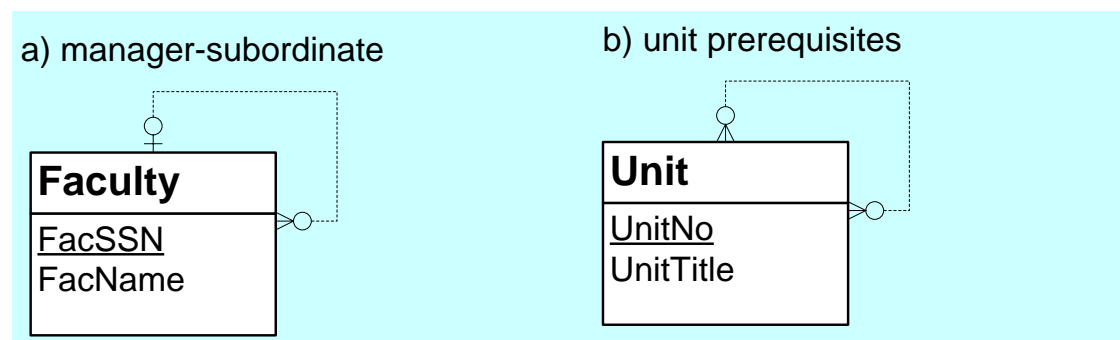
Example of a recursive relationship

(drawn schematically, not an ERD):



Drawing recursive relationships

The ERD shows a relationship line drawn from one side of the entity to the other, using the usual cardinality and optionality symbols



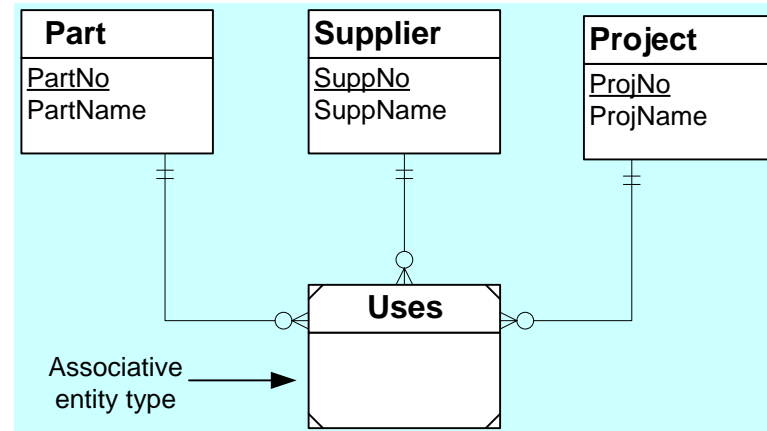
Multi Way (M-Way) Relationships

Relationship between more than two entities

- Binary – two entities (the usual)
- *Ternary* – three entities
- *Quarternary* – four entities
- *N-ary* – many entities

Generally represented as a series of **binary** relationships with an **associative** entity to represent the relationship

M-Way Relationships

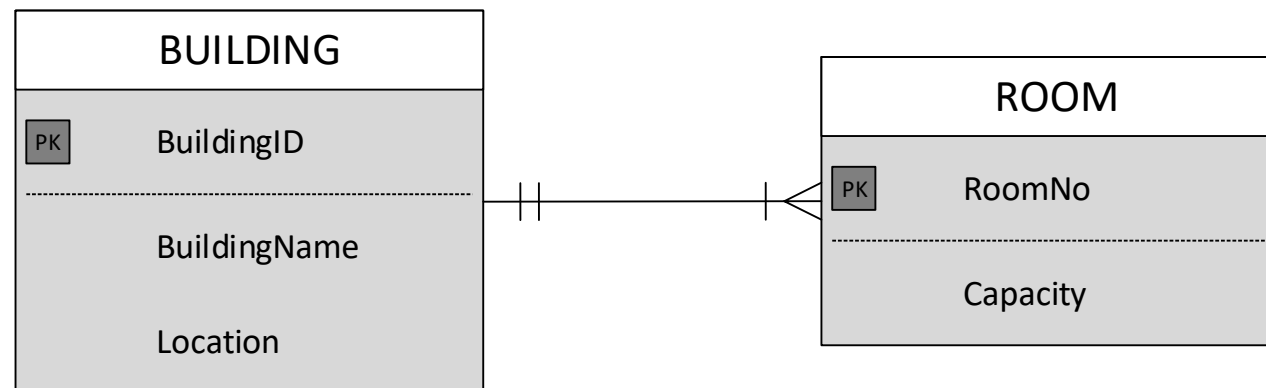


Entity-Relationship Modelling –Weak entities

Weak entities

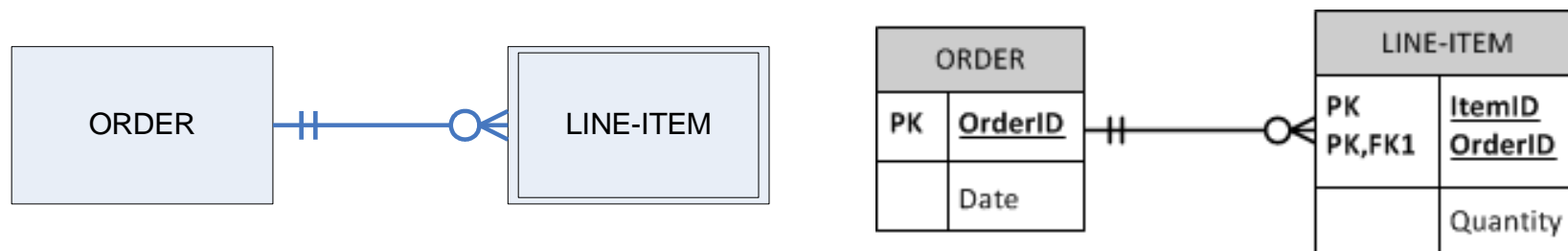
- Weak entities are those whose instances cannot exist without an instance of another entity:
 - a weak entity is *existence-dependent* on the owner entity

In the example below, a room cannot have an existence separate from the building it is in



Weak entities in ERDs

- Usually have a different symbol for the entity
- Always have a mandatory relationship with their owner entity
- Usually (not always) have a composite primary key made up in part of the primary key of the owner entity – if it does it is known as an **ID-dependent entity**



Examples of weak entities

- Typically used to model collections of things within the owner entity
 - An Order and its Line-Items – a line item doesn't make sense except in the context of an order
 - Phone calls within an itemised telephone bill
 - Fault reporting on an item of machinery
 - Legs within an airline route
- Modelling history - attribute variation over time
 - Person and their weights on different dates
- Also used where the entity has no meaning within the database without its owner entity
 - Staff and their Next-of-Kin

Entity-Relationship Modelling – Generalisation/Specialisation and Subtypes

Generalisation/specialisation

- Additional concepts were soon incorporated into the original ER model and called the Extended Entity-Relationship (EER) model.
- The most useful additional data modelling concept of the Extended ER (EER) model is called generalisation/specialisation (subtyping)
 - Based on the idea of classification and inheritance
- If you did ICT284, it's very similar to subclasses in UML
- Be aware that there are lots of different diagramming conventions for generalisation/ specialisation!

Classification

The process of 'grouping' objects based on some similarities:

- Biological taxonomies, such as Plants/Animals/Fungi
- Administrative classifications such as Students/Faculty/Administrative Staff

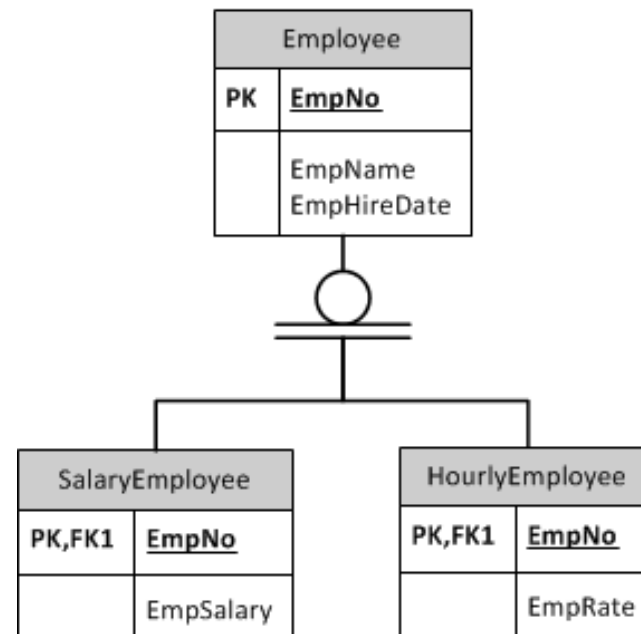
Classification is what allows us to model inheritance/generalisation for our database

- Can model entities that share common characteristics, but which also have differences

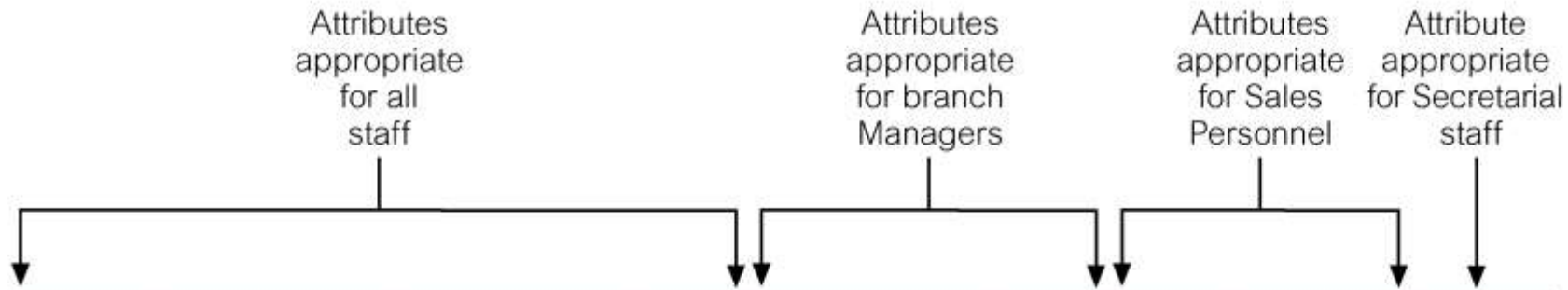
Subtypes and supertypes

An entity in a subtype represents same 'real world' object as in supertype, and may possess subtype specific attributes, as well as those associated with the supertype

- Subtypes inherit attributes of supertypes (direct and indirect)
- e.g. the SalaryEmployee subtype inherits the EmpName and EmpHireDate attributes



AllStaff relation holding details of all staff



staffNo	name	position	salary	mgrStartDate	bonus	sales Area	car Allowance	typing Speed
SL21	John White	Manager	30000	01/02/95	2000			
SG37	Ann Beech	Assistant	12000					
SG66	Mary Martinez	Sales Manager	27000			SA1A	5000	
SA9	Mary Howe	Assistant	9000					
SL89	Stuart Stern	Secretary	8500					100
SL31	Robert Chin	Snr Sales Asst	17000			SA2B	3700	
SG5	Susan Brand	Manager	24000	01/06/91	2350			

Constraints on Specialisation / Generalisation

Participation constraint

- Determines whether every member in supertype must participate as a member of a subtype.
- May be mandatory or optional:
 - **mandatory**: every instance of a supertype must also be an instance in the subtypes
 - **optional**: there may be instances in the supertype that are not in the subtype

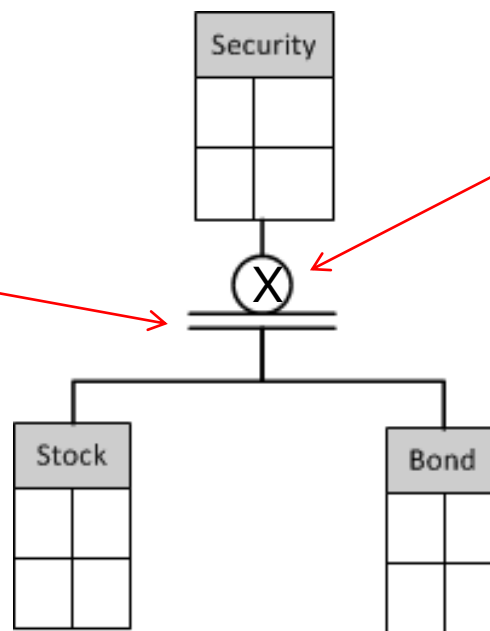
Exclusive or inclusive constraint

- Describes the relationship between members of the subtypes and indicates whether member of a supertype can be a member of one, or more than one, subtype.
- May be **exclusive** (no entity instance in common – *or*) or **inclusive** (overlapping – *and*) (also called **disjoint** and nondisjoint)

Example

- The **Exclusive** constraint below means that if a security is a stock it cannot also be a bond
- The **mandatory participation** constraint means that every security *must* be either a stock or a bond

in Visio 2010
mandatory participation is shown with a double line



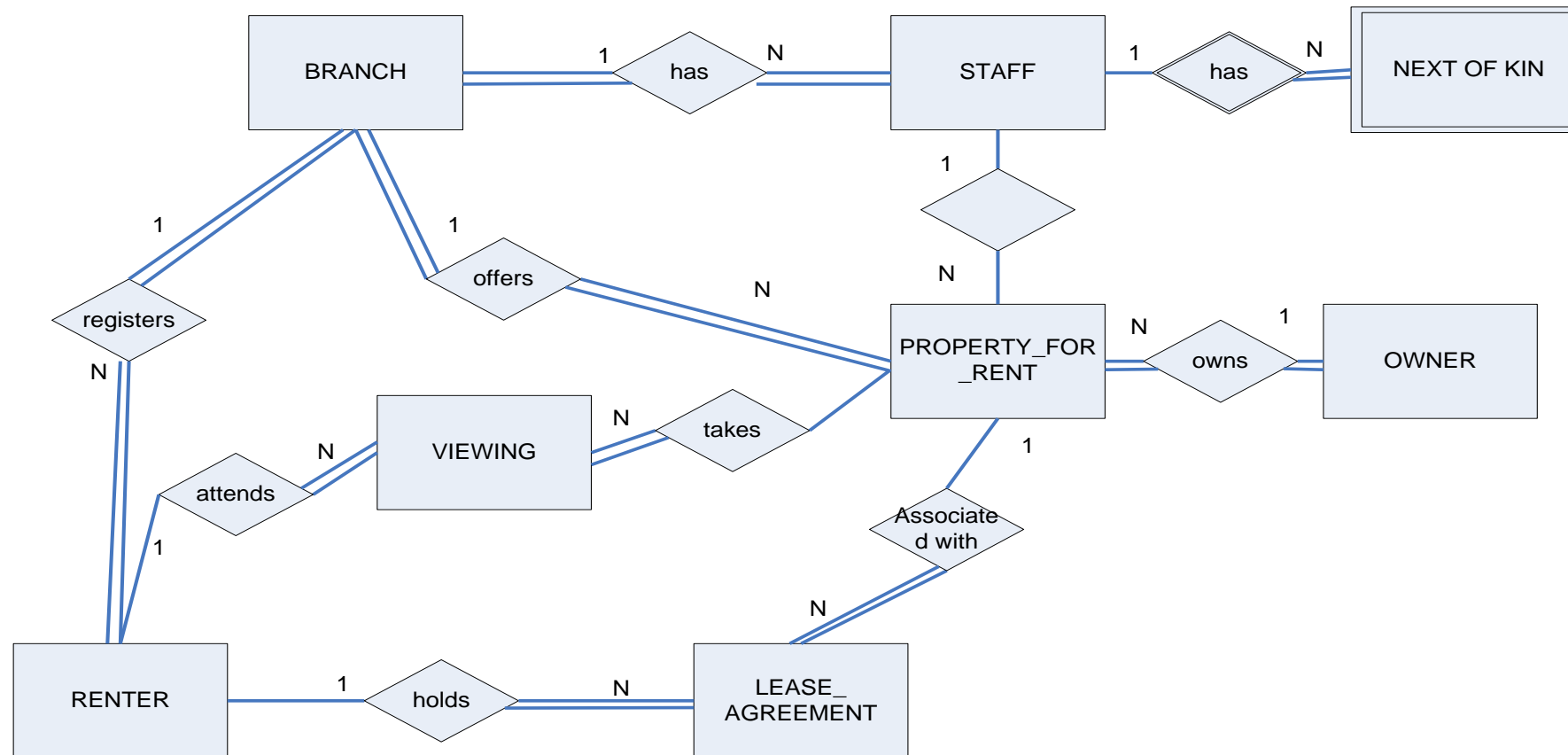
The *exclusive* constraint can be shown with an X or d in the circle



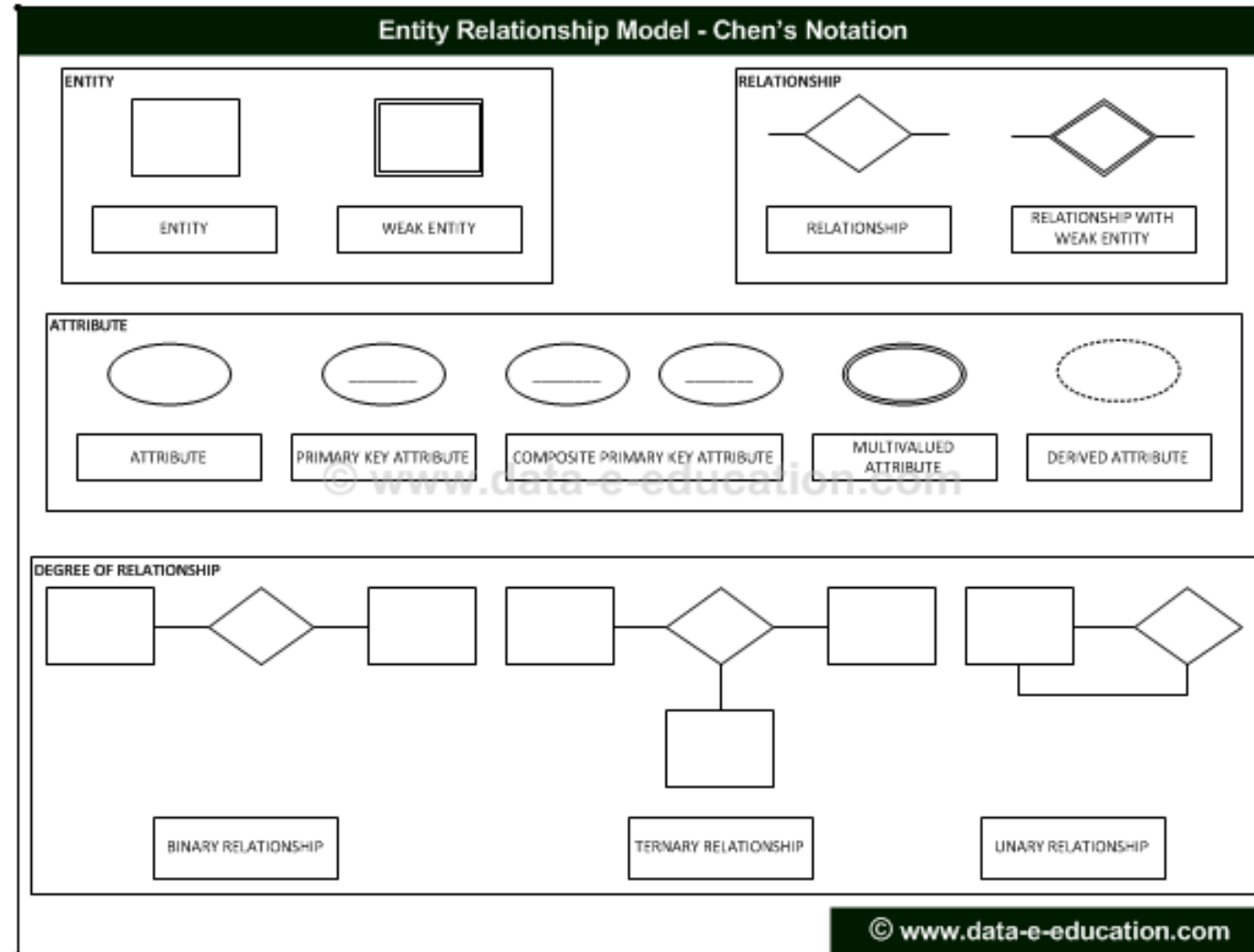
Different types of Entity-Relationship diagram notation: review



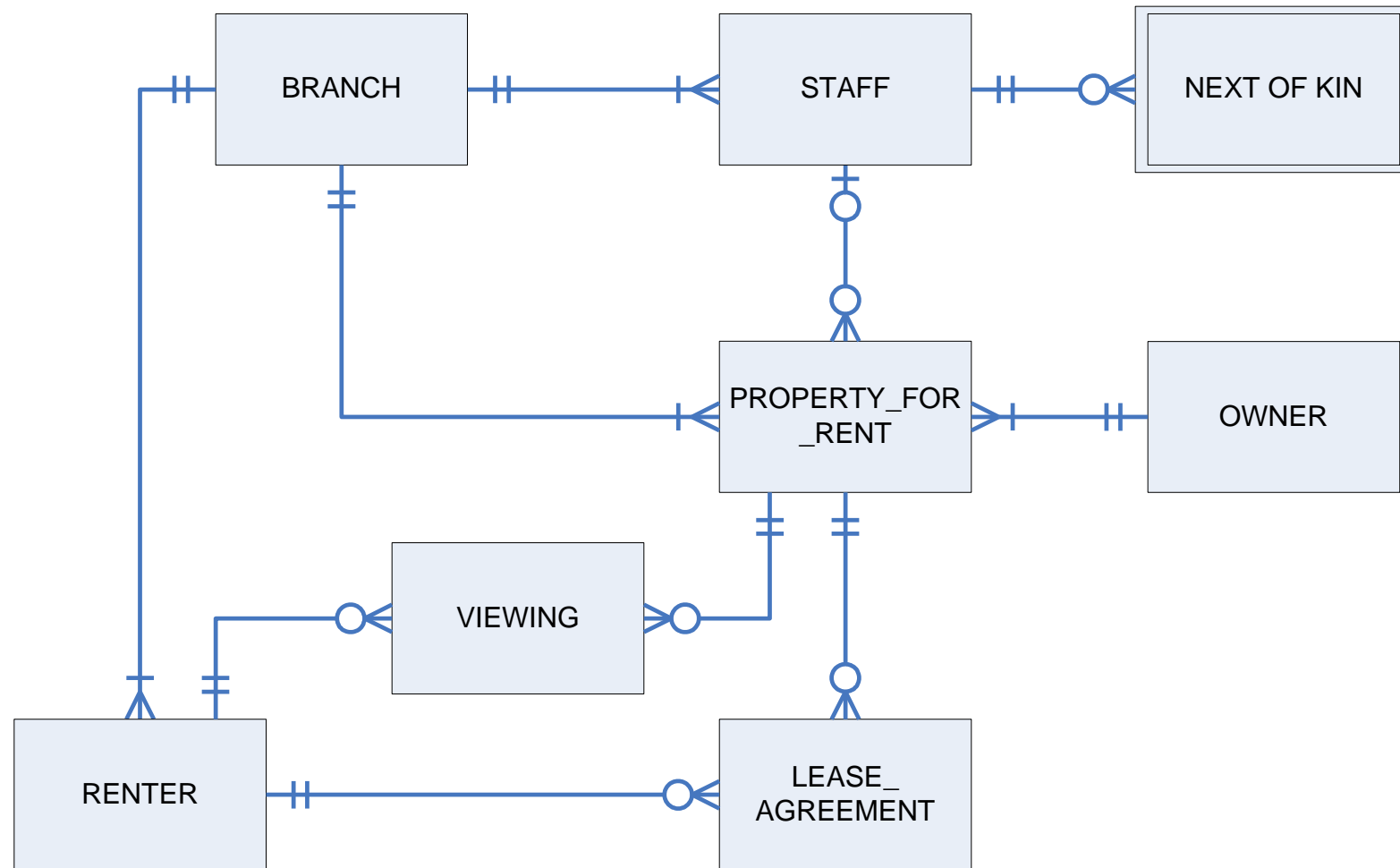
ER Diagram: Chen notation (without attributes)



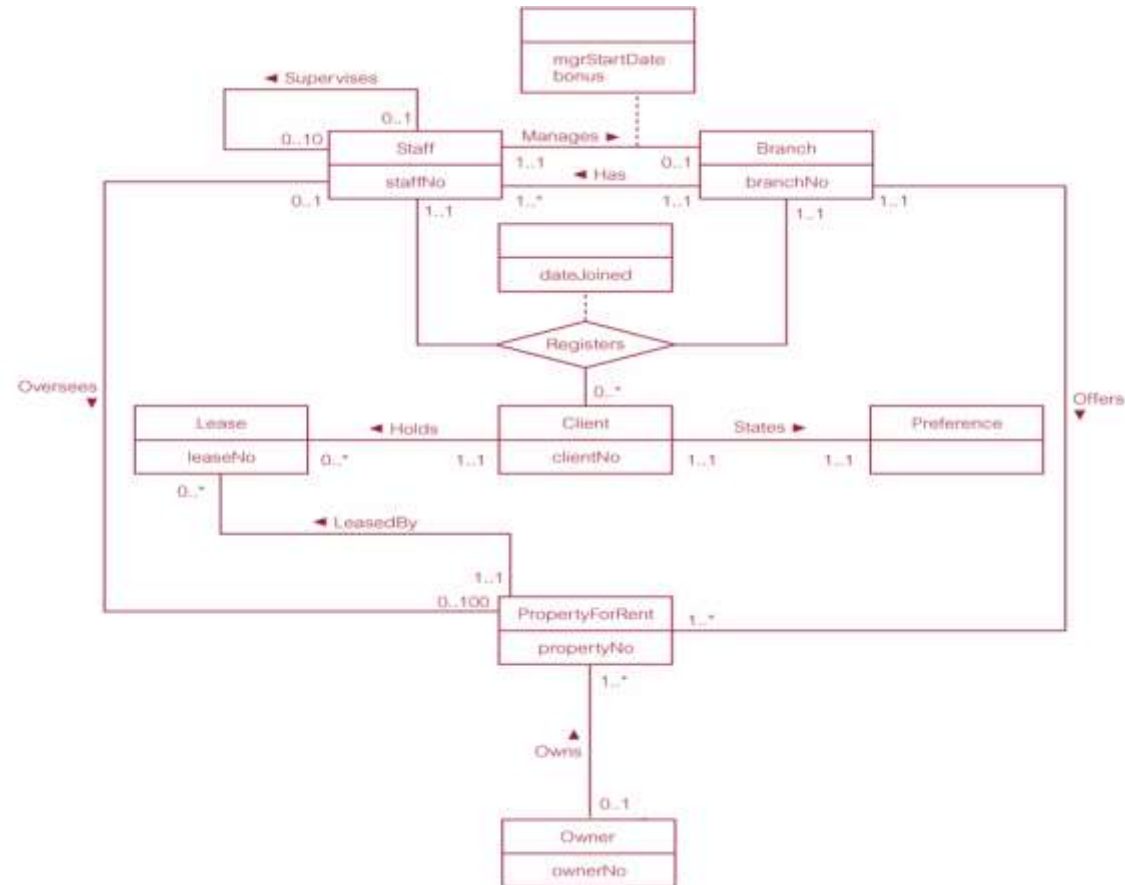
Chen Notation



ER Diagram: Crow's Feet



UML class diagram (not an ERD but similar features)



More in ICT284!

So which should you use??

All notations have their good and bad points

- Chen allows very precise modelling of attributes, but is very bulky to draw
- Crow's feet are concise and simple, but you need to document the attributes separately as you go
- Fully-attributed ERDs can be difficult to read, but provide a very clear mapping to the logical design and implementation

When drawing your own ERDs, use any one that you are comfortable with, so long as:

- You don't make up your own!
- You use the same convention all the way through the ERD
- You include a legend with the diagram to make it quite clear what the symbols represent

In an assignment or the exam, use the one that's asked for 😊

You should be able to READ any of the notations



Topic 05: Part 03
How to construct an ERD

How to construct an ERD

- **Discover entities**
 - Entities must be about only one thing
 - Methods such as 'noun technique' (ICT284)
- **Construct context (high level) data model**
 - Show entities and relationship cardinality
- **Identify the keys of each entity**
 - Show primary key
- **Create fully attributed data model**
 - Add attributes.
 - Attributes must relate only to that entity
- **Improve data model using normalization**

This is generally a useful approach to follow, although many notations don't support the inclusion of attributes within the ERD itself – however, they should always be documented

How to construct an ERD: an iterative process!

- Identify entities, attributes, relationships
 - In a written or verbal description, nouns are often entities, verbs give clues as to relationships
 - Other sources are forms, reports, existing databases
- Sketch the ERD
 - Normalise the ERD
 - Remove multivalued attributes, resolve M:N relationships
- Check for 'syntax' errors
- Check that the diagram is complete
- Do further checks for correctness, including verifying assumptions with client

Using Entity-Relationship modelling in Requirements Analysis

- The ultimate aim of drawing an ERD is to create a representation or model of the system.
- However, it's also a useful way of exploring ideas as well as documenting them - finding out what you know and don't know about the system
- Whenever you draft a diagram, note on it:
 - What you are sure about
 - What you are unsure about
 - What questions you need to ask to clarify the parts you aren't sure about

Exercise: Rent-a-Van

Draw an ERD to represent the Rent-a-Van data requirements:

“Rent-a-Van retails minivans for a number of manufacturers. Each manufacturer offers several models of its minivan (e.g., SE, LE, GT). Each model comes with a standard set of equipment (e.g., the Acme SE comes with wheels, seats, and an engine)

Minivans can have a variety of additional equipment or accessories (radio, air conditioning, automatic transmission, airbag, etc.), but not all accessories are available for all minivans (e.g., not all manufacturers offer a driver’s side airbag).

Some sets of accessories are sold as packages (e.g., the luxury package might include stereo, six speakers, cocktail bar, and twin overhead fluffy dice).”

Patterns in forms, reports and E-R models

Patterns

- Because a lot of business processes and transactions do much the same thing, you will find similarities between different organisational data models, although the particular names of the entities may differ
 - e.g. representing a *customer order for particular items* is usually likely to have the form:
Customer 1–N Order 1– N OrderItem N-1 Item
- Part of becoming an expert data modeller is about recognising typical **patterns** and knowing how to work with them
- Patterns can often be recognised from an organisation's forms and reports
- The textbook illustrates several of these: read through the examples yourself to get an idea of some typical patterns (*not examinable – just useful!*)

Data model patterns described in Kroenke

- Strong Entity Patterns (1:1, 1:N, M:N)
- ID-Dependent Relationships
 - The Association Pattern
 - The Multivalued Attribute Pattern
 - The Archetype/Instance Pattern
- Mixed Identifying and NonIdentifying Patterns
 - The Line-Item Pattern
 - The For-Use-By Pattern
 - Recursive Patterns
- Patterns based on reports

Topic 05: Part 04

How to check your ERD for correctness

How do you know if you've got it right?

Even though it's a diagram, you can check an ERD for errors in the same sort of way as you do programs

- Syntax errors
 - Errors in the 'grammar' of the technique – **using the notation incorrectly**
- Logic errors
 - Errors that mean the system isn't represented correctly – **business rules are not represented**
- Run-time errors
 - Errors that mean the system **won't perform correctly in practice** and over time

Errors in the diagramming notation: Check -

- Entities named correctly?
 - Must have singular names
 - Must have unique names
 - Must represent types of things, not individual instances
- Identifiers (primary keys) indicated?
(if you are using a notation that shows attributes)
- Relationships have complete cardinality/optionality indicated at both ends?
- Relationships are only between entities – not between other relationships?

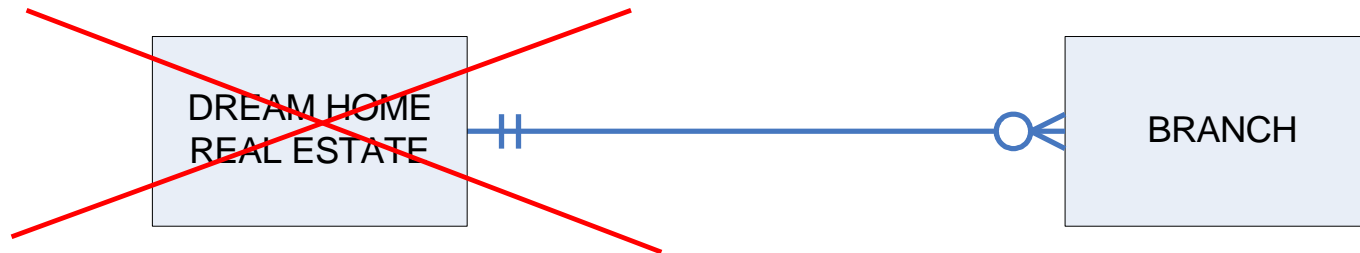
Errors that mean the system isn't represented correctly according to the business rules: Check -

- Are your entities really entities?
- Are your attributes really attributes?
- Are your relationships are really relationships?
- Are the correct entities related?
- Is the cardinality of relationships correct?
- Try to get into the habit of recognising typical patterns of entities and relationships
 - These include both traps to avoid, and typical business usage patterns – this is largely a matter of experience
- *we'll discuss all of these next*

Are your entities really entities?

1. Modelling the system as an entity

- Recognised by the entity being named for the system, and problems in working out what relationships it has
- Often arises because of a literal reading of a description: "Dream Home Real Estate has several branches..."
- It is **wrong** because an entity must have the potential for many individual instances, and 'the system' is only one instance



Ask yourself: are there any *other* real estate agencies to be modelled in this system? If so, keep it as an entity (called AGENCY), if not, remove it

Are your entities really entities?

2. Modelling reports as entities

- Clue: an entity called 'Weekly Report' or similar
- Again, follows from a literal reading of the problem description
 - This is **wrong** because reports are an *output* of the database - the result of querying on the database contents (often from many individual tables)



To fix: Remove the 'report' entity, and check that all the information you require for the report can be found from the other entities in the data model

Are your entities really entities?

3. Modelling attributes as entities

- If you find you can't define any attributes of an entity, consider whether it is really an entity in itself, or just an attribute of the related entity
- Make it an attribute if:
 - It only takes one value for the entity instance, and that value is likely to be specific to the particular instance
- Consider keeping it as a separate entity if:
 - It can be multivalued for the entity (since multivalued attributes not permitted in RM)
 - The values it can take would be reusable by other instances of the related entity

Example

Student has date of birth

- Would not model as an entity as there is only ever one value for the student



The fact that many students could share a DOB is not likely to be relevant - date of birth is not of interest in its own right in this system

Another example...

Student has Address

- Would possibly model StudentAddress as a weak entity if multiple addresses were to be stored,
 - e.g. semester address, mailing address, permanent address



Another example

Car has Colour – entity or attribute?

- Potentially a separate entity COLOUR if multiple colours possible for that model, and using a standard colour set that could apply to many models
- In this example, Colour could be of interest in its own right (and you would probably find extra attributes, specific to car manufacture, such as code or batch number)



Are your entities really entities?

4. Entities that aren't related to any other entities

- This is usually a sign that the entity doesn't belong in this system
 - Check for the other errors first – maybe you have represented the system itself, or a report
 - Occasionally you would want a standalone entity to provide a set of lookup values, but this situation is very unlikely in any data model you will be asked to draw in ICT285!

Are your entities really entities?

5. Entities that are the same thing, but in a different state

A 'Prospect' eventually becomes a 'Student' – should this be modelled as:

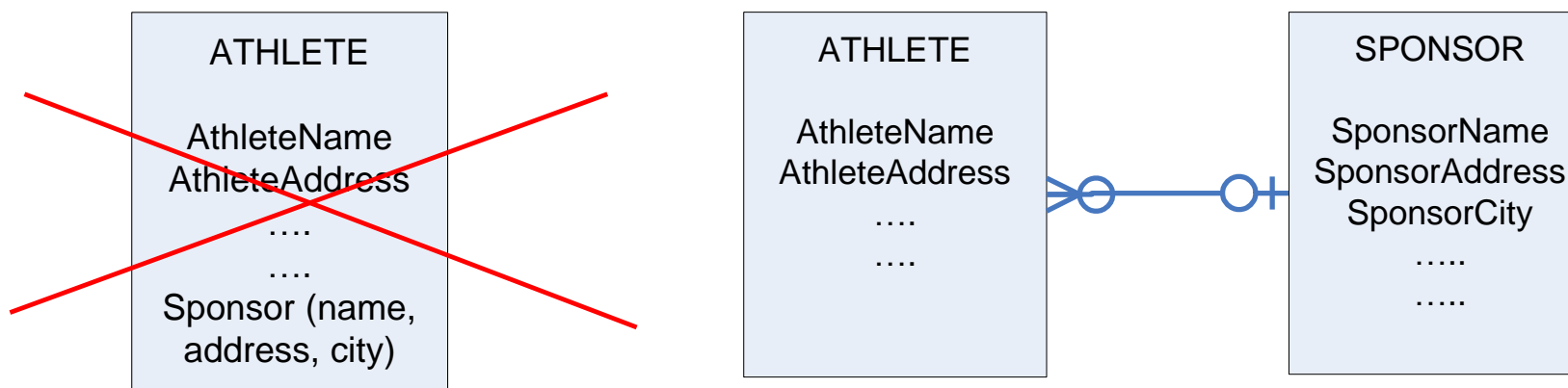
- Two related entities?
- An entity Student, with attribute 'State' which can take the value Prospective?
- A generalisation (supertype) entity Student, with subtypes Enrolled and Prospective?

•Ask – are there different attributes? Are there different relationships?

Are your attributes really attributes?

1. Attributes that should be entities

- A complex or multivalued attribute often becomes a related entity instead when the entity is fully normalised



(Primary keys and foreign keys not shown)

Are your attributes really attributes?

2. Mistaking values for attributes

- This can easily be done when using printed forms as the basis for the data model – can be tempting to use them too literally:

Payment method:

Paypal	Credit	Cash
<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

- Separate attributes PayPal, Credit, Cash, each with Yes/No values??
 - If another payment method is accepted, need to redo database design – so not a good solution
- Attribute PaymentMethod, with possible values PayPal, Credit, Cash?
 - Extendible to another payment method simply by adding a new value to the list of valid ones for that field – *more flexible solution*

Are your relationships really relationships?

Check that you haven't modelled *process* instead of relationship

- In the data model, it doesn't necessarily matter *how* something comes about, just *what* the result is
- e.g. Secretary updates Property table – should this indicate a relationship between Secretary and Property entities??
 - If we need to keep track of who updates what, for auditing purposes, then YES
 - Otherwise NO

Are the correct entities related?

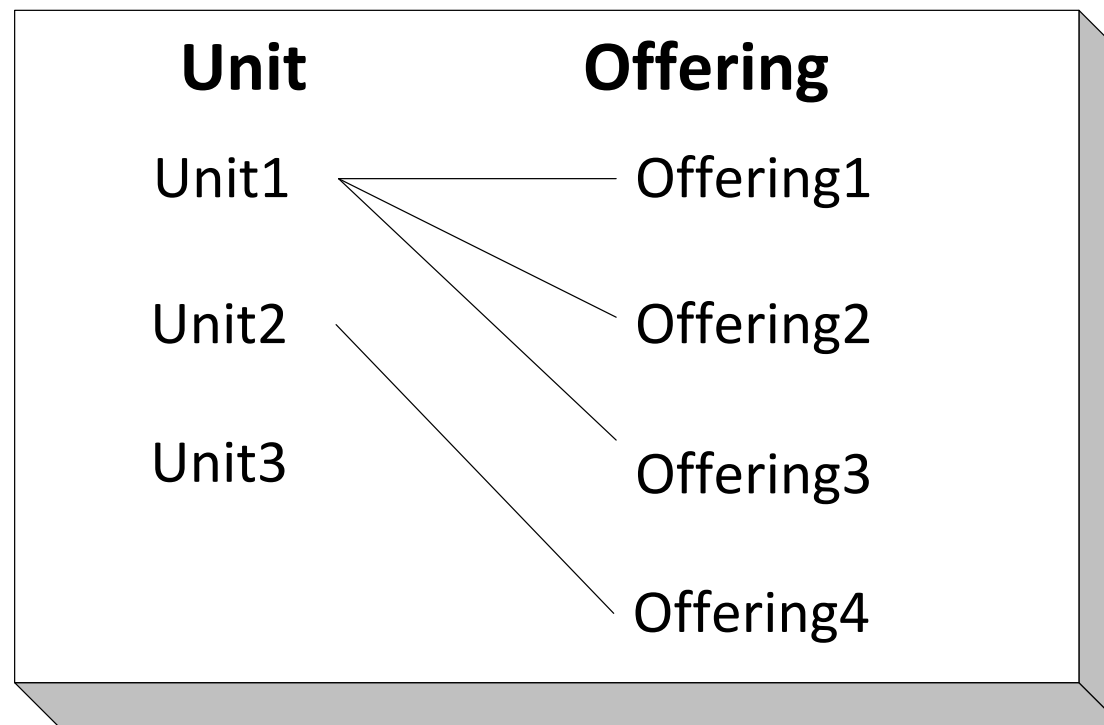
- Getting the correct entities, but relating them incorrectly, is the cause of some typical 'traps' which we shall consider shortly
- Generally, these traps arise from representing indirect relationships rather than the most direct

Is the cardinality of relationships correct?

Check using occurrence diagrams to be sure you understand what the cardinality represents, and make sure this matches the business rules of the system

- Remember how to read cardinality: it refers to instances of the entity, not the set
 - i.e., 'how many instances of entity B are related to ONE instance of entity A?'
- Make sure the cardinality is correct for the entire lifetime of the database

Checking an ERD using Occurrence Diagrams



(What is the corresponding ERD?)

Get into the habit of recognising typical patterns of entities and relationships

These include:

- Traps to avoid (next section)
- Typical ER constructions:
 - Expanding a M:N relationship to two 1:N
 - Use of a weak entity to represent history
- Typical business usage patterns

Proficiency is largely a matter of experience and practice!

Errors that mean the system won't function correctly

1. Forgetting about the timescale of the database

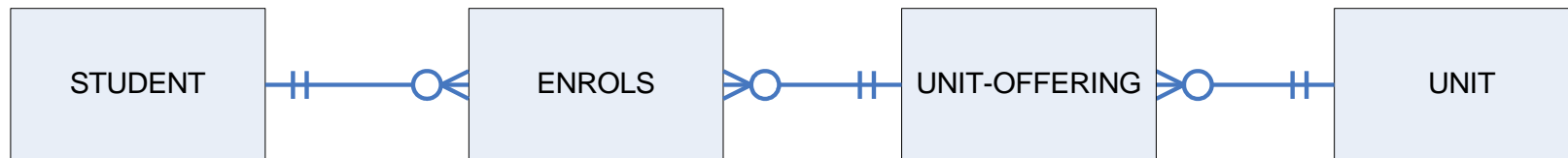
A typical error is to forget about *time* so that you end up with an ERD which will work for a single occasion, but not repeated use

- Ask yourself: does this ERD model the database for the length of time it will last?
- Usually the solution is to add another entity to represent multiple occasions of something (see example on next slide)

Example: thinking of the time dimension



- OK for this semester... but what happens if you fail ICT285 and need to come back next semester?



- This is better, can now have multiple enrolments in a 'unit-offering'
- Still need to ask: would this distinguish different years adequately for the system?

Errors that mean the system won't function correctly

2. Being inflexible

Not really an error, but more like poor practice – you should always design a system to be flexible:

- Prepare for multiple occurrences of things that are only 'supposed' to happen once
- Be able to handle a larger set of possible values than the ones given
- Allow for the possibility of different behaviour by using subtypes
- Keep concepts as distinct as possible through normalisation
 - it's easy to combine entities using queries, but impossible to split up something that is a whole

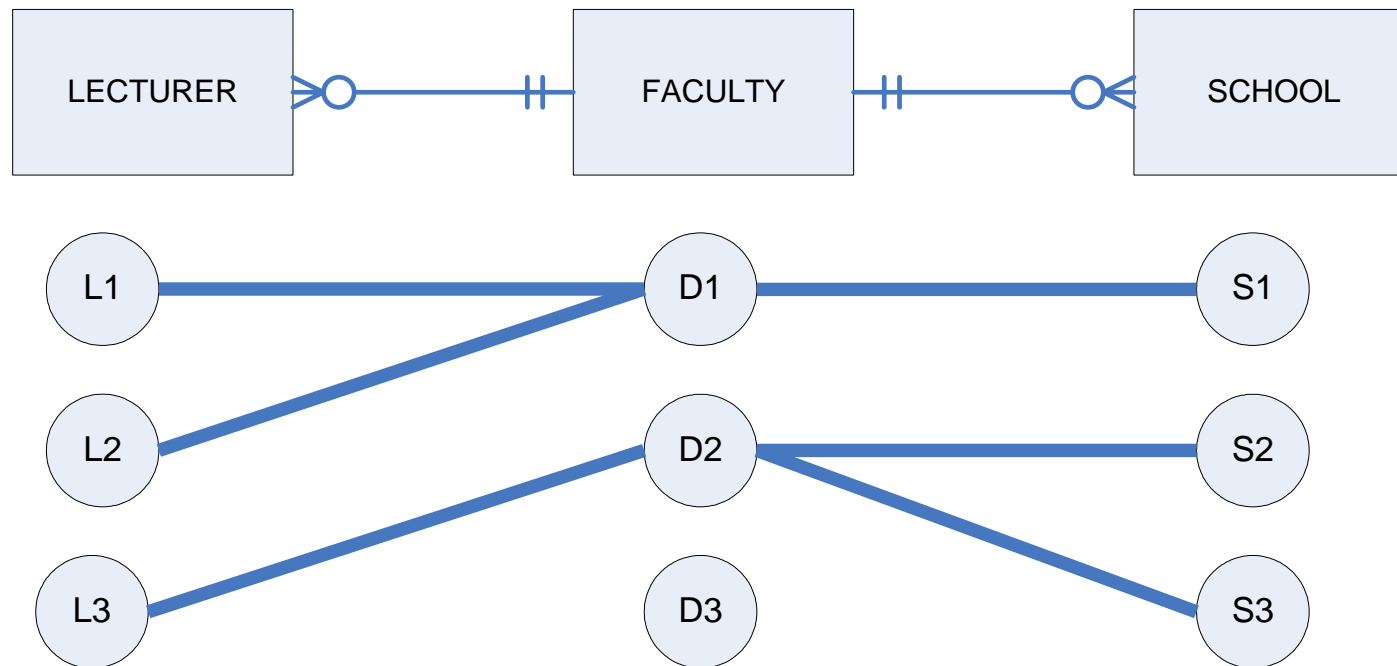
Classic 'Traps' – errors to avoid

Fan Trap

Chasm Trap

Incomplete Circular Relationships

Fan Trap



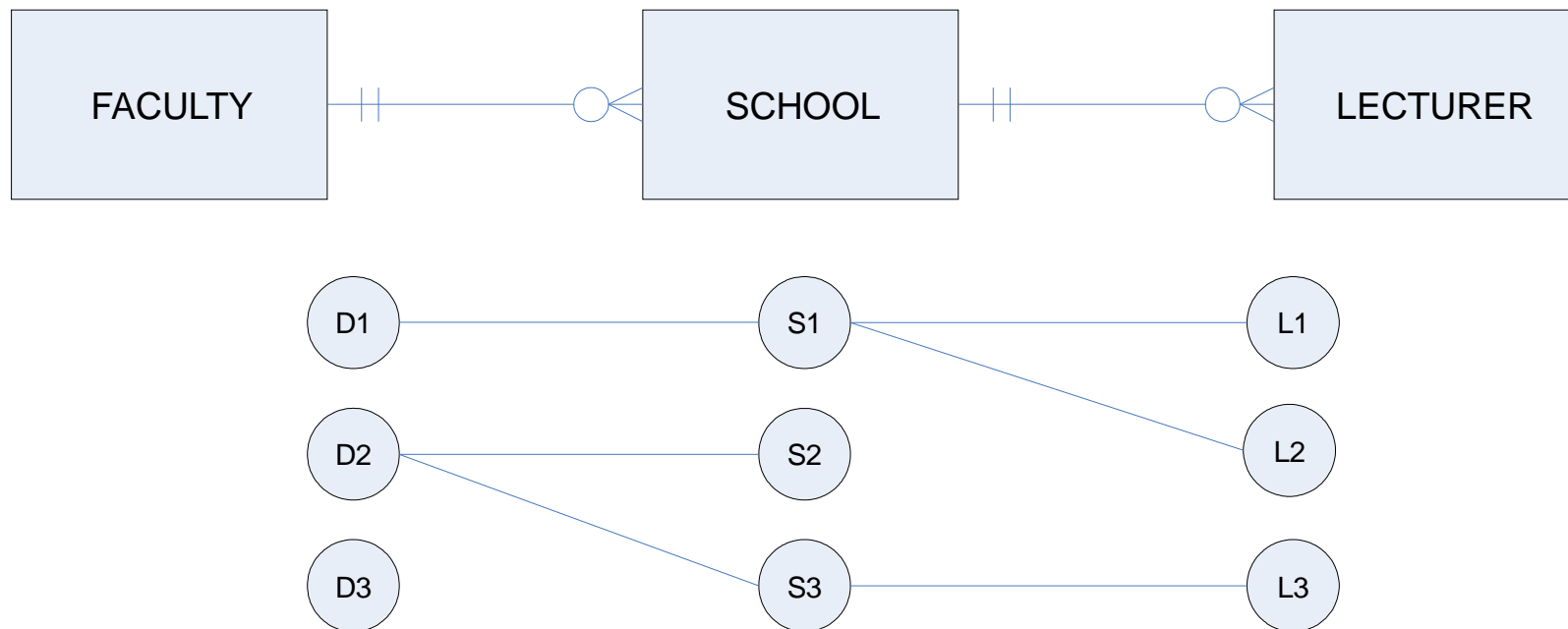
Which School does Lecturer 3 work in??

Fan Trap explanation

There is potential for a Fan Trap whenever there is a **many-to-one** then **one-to-many** relationship

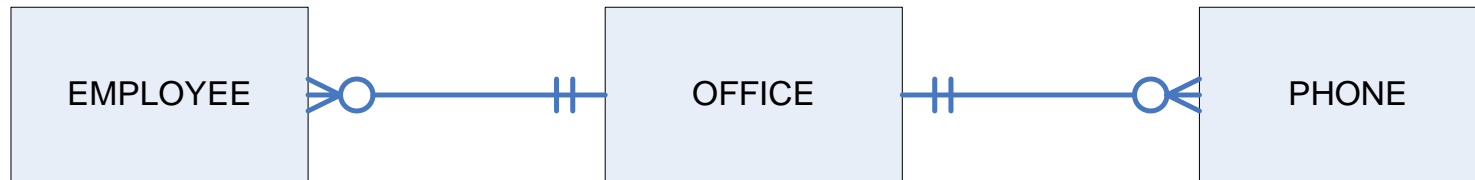
- Example - At Murdoch, each is in a particular Faculty, and a Faculty has many Schools. A Faculty also has many lecturers
 - We can't find out from the ERD on the previous slide what School which lecturer works in - yet we know (from our knowledge of Murdoch) that lecturers are associated with particular Schools
- Problem arose because we drew a *derived* relationship (Faculty has Lecturer) instead of the *direct* one, School has Lecturer
 - **Solution: Redraw using the direct relationship**

Fan Trap: Solution



Now we can see that Lecturer 3 works in School 3

Fan Trap – another example



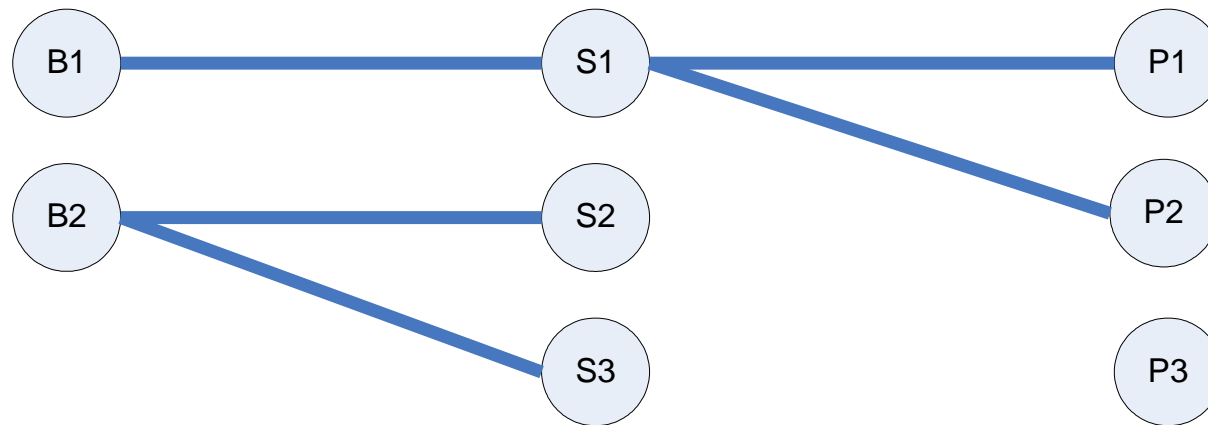
- Is this a fan trap??
 - Suppose we have an open plan office, with no connection between phone and employee - any employee can use any phone in the office
 - Then there would be no problem with this representation

Checking for the Fan Trap

Not every N:1 1:N pattern is a fan trap – but many are, so look out for it

- The fan trap also applies (potentially) if there is a **many-to-many** relationship followed by a one-to-many or a many to many
- (because the M:N relationship can be resolved into two 1:N relationships with a new associative or 'intersection' entity)

Chasm Trap



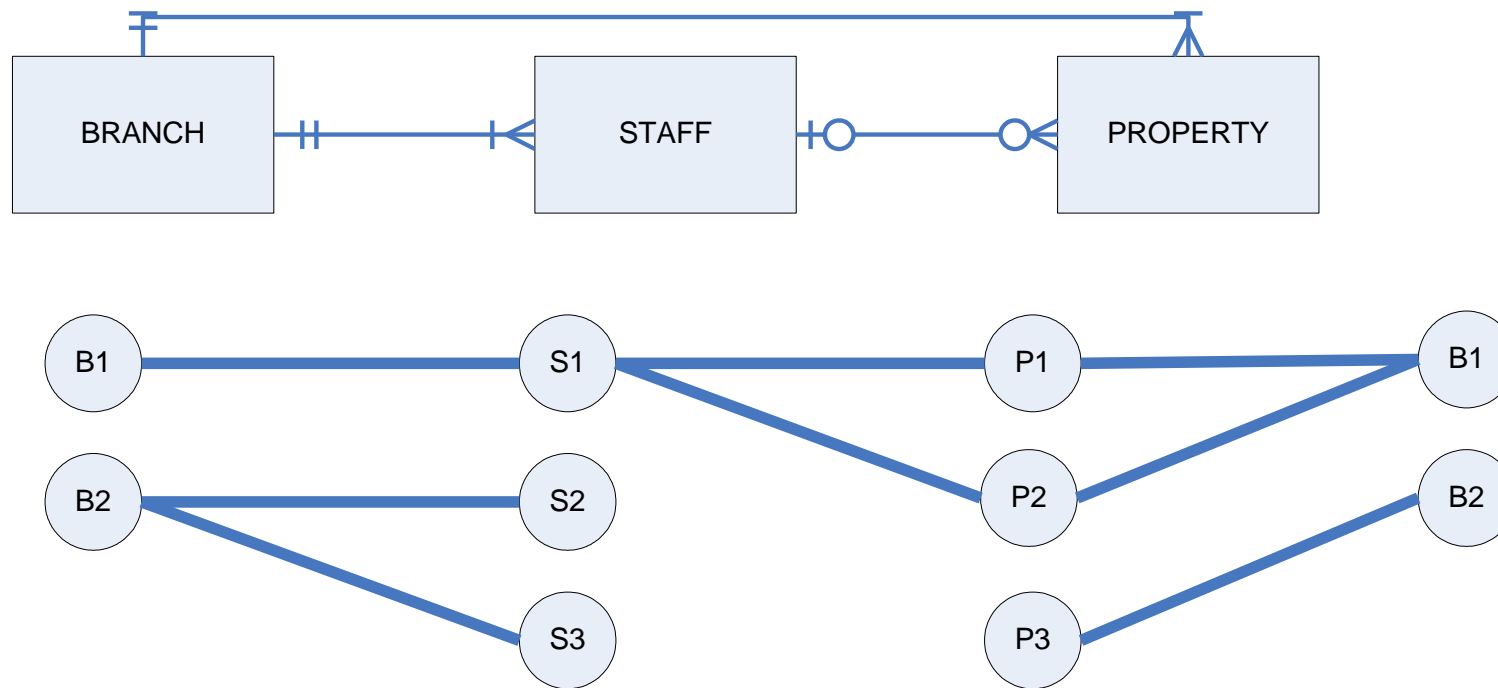
Which branch manages Property P3?

Chasm Trap - explanation

The Chasm Trap arises when there are *optional* relationships in a pathway between related entities, so that for some entity instances it is impossible to get at the relationship

- In the example, not all properties at a branch are assigned to staff – so it is impossible to tell which branch these properties are registered at
- **Solution: add in the direct relationship**

Chasm Trap



Now we can see that Property 3 is managed by Branch 2

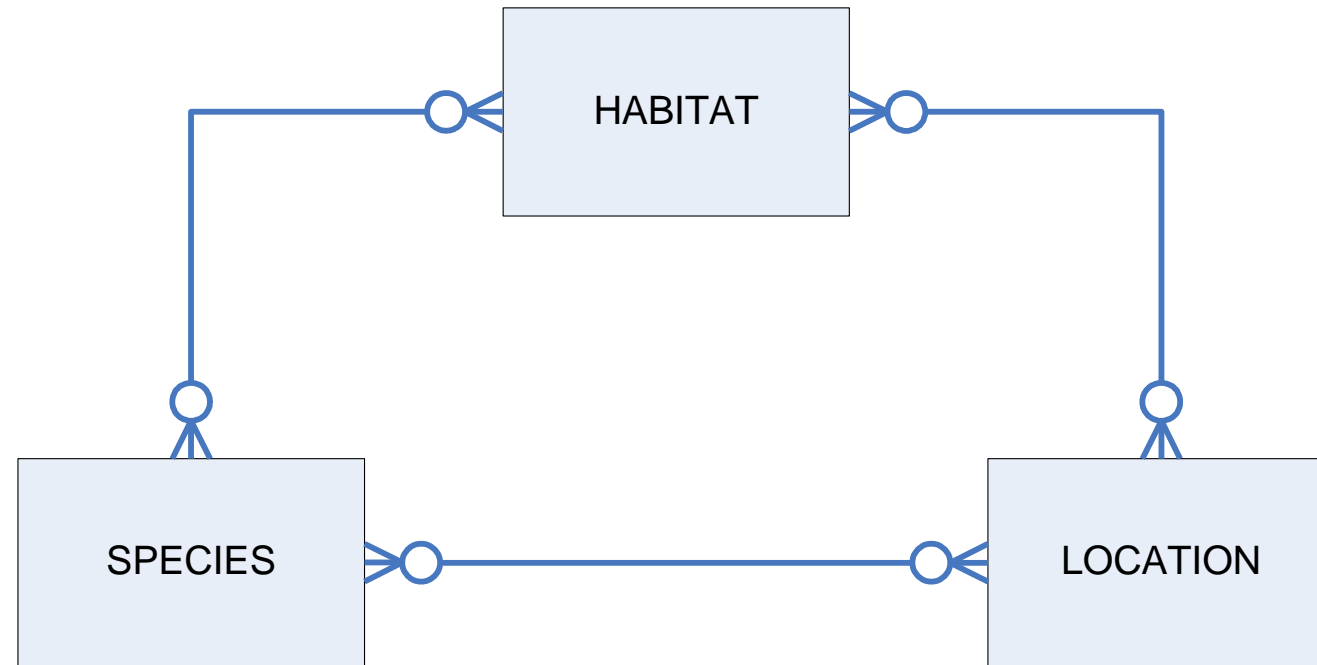
(always check that both sets of relationships are actually needed)

Checking for the Chasm Trap

Check any pathway of related entities that includes optional relationships for potential chasm traps

- Again, the presence of optional relationships may not indicate a problem, but it is always worth checking
- Solutions may include adding another relationship, or restructuring the existing ones

Incomplete circular relationships



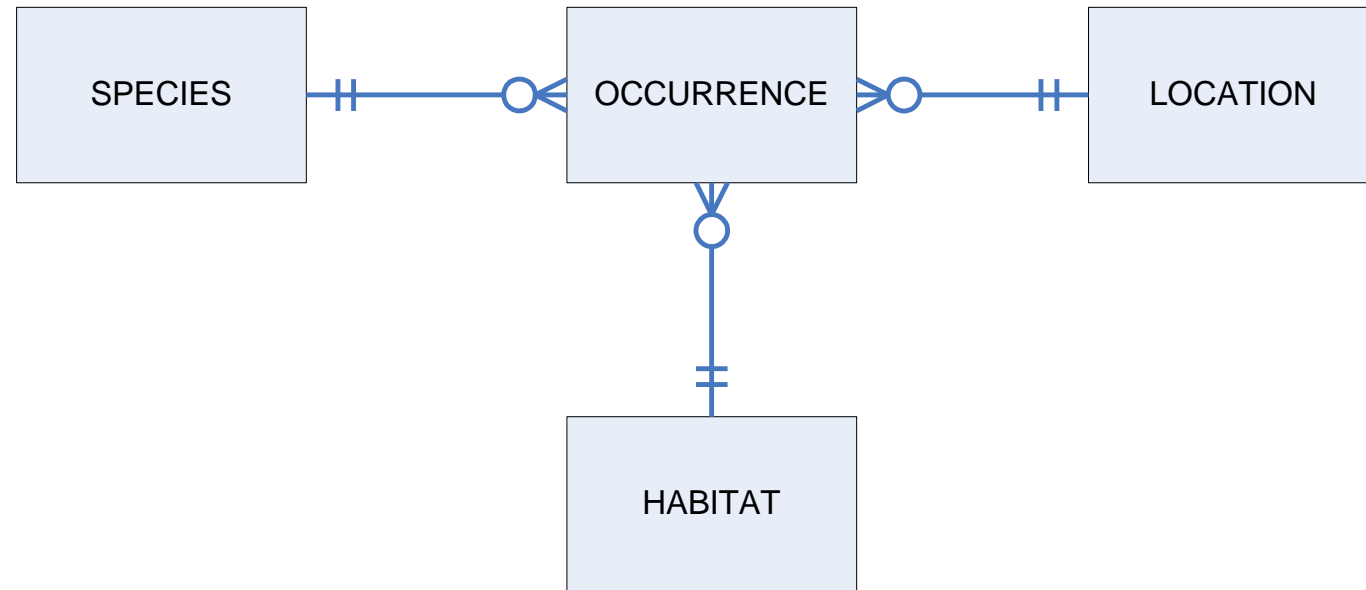
We have many-to-many relationships everywhere... but can we find out if a particular species is found in a particular habitat and location?

Incomplete circular relationships – explanation

Incomplete circular relationships can arise when you need to represent a combination of more than two entities

- Often a clue is a combination of several many-to-many relationships, which don't solve the problem even when expanded
- **Solution is to create another entity, to deal with the combination directly**

Incomplete circular relationships: Solution



Create a central entity **Occurrence** to represent the *combination* of Species, Habitat and Location



Topic 05: Part 05
Conclusion

A final word on entity relationship diagrams....

ERDs are a valuable modelling tool for database design:

- Specify the meaning of a system
- Explain the designer's understanding of the system to the client
- Basis for logical database design (next topic)

Try to develop a consistent way of drawing ERDs

- Make use of legends wherever required (including assignments and exams!) to assist others in understanding your diagrams
- Learn to use a drawing tool (such as Visio) well
- Learn from your mistakes and learn from your successes – this is the way you will build up expertise

Learning outcomes revisited

After completing this topic you should be able to:

- Explain where conceptual data modelling fits in the database development life cycle and SDLC
- Describe the features of the Entity-Relationship model: entities, attributes, relationships, cardinality, weak entities
- Describe the features of the Extended Entity-Relationship model: supertypes and subtypes
- Interpret different types of Entity-Relationship diagram notation
- Draw an Entity-Relationship Diagram from a description
- Use Entity-Relationship modelling as a means of eliciting and checking user requirements for a system
- Check your Entity-Relationship Diagram for correctness
- Avoid typical 'traps' of poor ER modelling practice

What's next?

In the next topic, we continue our coverage of the database design process by looking at logical design, and how to convert a conceptual model in an ERD to a logical design in a set of normalised tables.