Database development

Introduction

When you are building a house, being able to put one brick on top of another in a sensible way is an important skill. However there are many others that are needed if you are to complete the whole project. From the early design stages, through planning application, to the coordination of subcontractors and final inspections, there are many aspects of the task that do not rely just on detailed practical skills.

The same is true of information systems and applications. There is always an end user whose needs and preferences have to be taken into account. There is always some organisational framework in which the project takes place. This will have its own rules, budgets and reporting requirements. Typically, the application will involve many people working as a team, and finally there are acceptance tests to be conducted so that everyone can get paid.

It is therefore not enough just to be able to set up a database structure that works. You also need to be able to describe it clearly and precisely to different types of people (managers, developers, end users), and you need to be able to follow a structured process that allows all of the stakeholders to express their requirements and to give feedback as the project proceeds.

This module is mainly about the technology. However, even the most specialised technical expert needs to consider the context in which their work takes place. The skills needed can be summarised as *communication* and *coordination*.

Communication

Constructing a clear ER diagram is a major factor in successful communication at the technical level. It is simple and captures most of the vital information that others on the development team need. However, an ER diagram is not a good medium for communicating with end users. Sometimes for large databases, and ER diagram takes up too much space and even amongst technical team members, you need something more succinct.

Communicating across the technical/non-technical boundary is one of the most widely-recognised problems in systems development. Learning a range of techniques to use in different circumstances will help you avoid problems.

Coordination

The stereotype is a reclusive genius who shuts himself away (yes, the stereotype is male) and brings out a fully functional application some time later to awestruck acclaim. In reality, this never happens. However, because the communication part is difficult, skilled technical people typically do not enjoy talking to users about their requirements or to managers about the need to report on their progress. Left to their own devices, these people are likely to avoid those unpleasant activities for the much more rewarding one of writing software. Although their work may be of the highest standard and their inventiveness beyond question, it is still possible they will deliver something that the end users cannot operate easily, and a lot of money is wasted in rework.

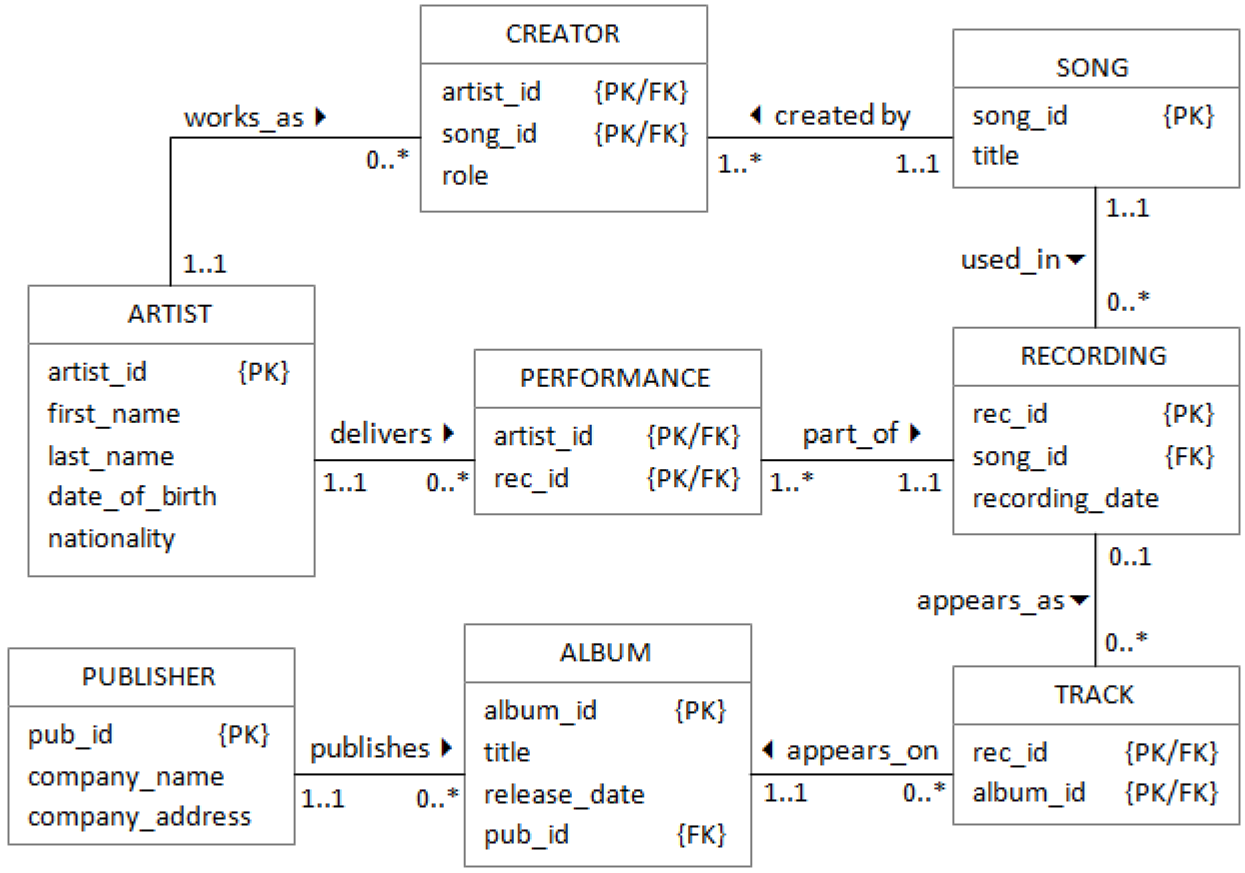
The purposes of a controlled development process are:

- to ensure that all relevant opinions are taken into account at appropriate times
- to provide a framework for making project decisions on things like timescales and project team sizes
- to identify and correct problems as early as possible

Describing schemas

A realistic schema for a large database system can include hundreds of tables. Once it reaches that size, getting all of the tables onto a single ER diagram is no longer feasible. One alternative is to break the diagram down into smaller parts, but even then the problem is not solved completely. A better solution is to have a way of expressing the same information in a much more condensed form.

Although there are no recognised standards for expressing schema contents in text-only form, there are some common conventions which it is important to know.



A schema is basically a list of relations (tables) and their attributes (columns). We could therefore represent the simple schema in the diagram above like this:

artist(artist_id, first_name, last_name, data_of_birth, nationality)
creator(*artist_id*, *song_id*, role)
song(song_id, title)
performance(*artist_id*, *rec_id*)
recording(rec_id, *song_id*, recording_date)
track(*rec_id*, *album_id*)
album(album_id, title, release_date, *pub_id*)
publisher(pub_id, company_name, company_address)

In this list, a relation's attributes appear inside the brackets following the relation name. The primary key is shown underlined, and a foreign key is shown in italic.

You might think that the information about relationships is greatly reduced in the text version. In fact, you can reconstruct the ER diagram almost entirely by using the foreign keys. For example, the CREATION relation has two foreign keys which are also primary keys. This immediately suggests that the table is a link table which resolves a many-to-many relationship. One of those foreign keys has the same name as the primary key in the ARTIST table which shows where the first relationship is. The second links to the SONG relation. The location of the foreign key identifies the "many" end of each of the relationships and confirms what we thought at the beginning about the nature of the CREATOR relation.

We could go through the whole schema like this and reconstruct most of the information in the diagram. The only thing that would be lost in this case would be the relationship labels. Actually, there are some other types of information that could be lost in this format. One-to-one relationships would be difficult to identify, for example, as would relationships where the foreign key has a different name from its corresponding primary key. These can only be reconstructed by reference to the *semantics* of the names - that is to say, their meaning. This illustrates how it is important to choose meaningful names for your relations and attributes.

Despite these small drawbacks, this way of describing a schema is very compact and has sufficient information for reasonably experienced database users.

Business rules

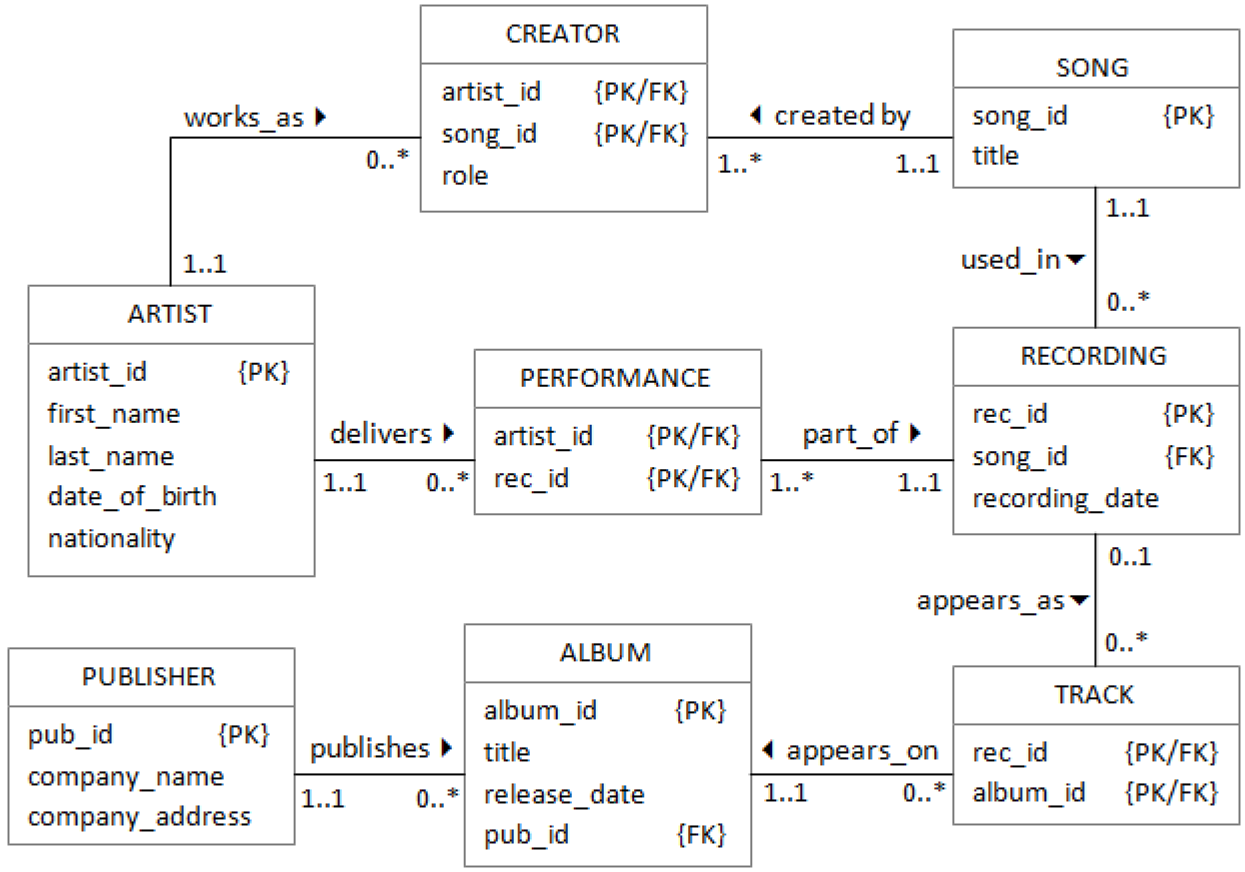
There are several identifiable points during the development of a database structure where it is important to talk directly to the end users. One of those times it right at the beginning when you are finding out exactly what the application will need to do, and therefore what data it will have to store.

At that time, you may already be picturing the data structure as an ER diagram; however, the end users will have great difficulty in understanding that representation. An alternative that is much more user-friendly is to talk about *business rules*. These are simple statements that capture either structural requirements or constraints on the data.

Structural rules

These rules describe the cardinality of relationships among entities in the system. Using the example of the music database again, the structural business rules would be as follows.

- One artist may create many songs, and one song must be created by one or more artists
- One artist may perform on many recordings, and one recording must have one or more performers
- One album is made up of many track recordings, and one recording may be used on many albums
- One publisher publishes many albums, and one album is published by one publisher



Notice that the business rules specify the relationship from both ends. It is important to elicit this information from the users at the beginning of the design process. Once you have identified a relationship between two entities during a user interview, it is always a good idea to ask the questions about cardinality explicitly. For example, if the user tells you that an artist may create many songs, that suggests a 1:* relationship. When you ask the question *"How many artists create one song?"*, the user will probably tell you that the answer is one or many. This shows that the relationship is actually *:*.

The business rules shown above do not mention the link tables that are shown in the ER diagram. The reason for this is that the entities CREATOR and PERFORMANCE will not have much meaning for the users you are talking to. The entity TRACK will be confusing because the user will probably want to use that term for the entity labelled RECORDING. Part of your job as designer is to include the appropriate amount of detail in your communications. Remember that the users are operating at the external level in the ANSI-SPARC model. Their understanding of the structures will be different from the one you are most interested in at the conceptual level. Expressing the structure as a set of business rules helps communication across the external/conceptual boundary.

Data rules

Users can also give you the information you need to define constraints on the data. For most of the tables in the music database, there are no problems, but the *role* column in the CREATOR table needs some clarification. The need for this column would arise out of asking the user how an artist can contribute to the creation of a song. The answer might be that the artist can be the lyricist or the composer, or both. Thus the only permissible values for the role column are *lyricist* and *composer*. There is no standard form for these rules, but you could write it like this:

- An artist can contribute to the creation of a song as lyricist or composer

Database lifecycle

Designing a database is usually done as part of a larger development project. A database on its own is not much good - you also need the user interface at the very least. From this observation, you might conclude that the database can simply be treated as another system component, and that a standard development methodology can be applied. There are some specific aspects of database design that bring this into question, however:

- **No other system components use the same conceptual framework, and therefore the database needs specific attention.**
  The identification of entities and attributes, and the definition of their relationships is not required in any other part of the system. It requires specific skills and techniques, and should not be artificially dependent on other parts of the process.
- **The database is a logically fundamental component on which everything else depends**
  The user interface and other system components rely on the database structure being correct. If that structure needs correction at an advanced stage of the development, many other components will need to be checked and possibly corrected as well. The database needs to be prioritised in the development in contrast to other components.
- **It is difficult to test a partially constructed database**
  Several current approaches to software development are based on the principle of iterative delivery of prototypes. Because the database presents a model of the business environment with static relationships between entities, opportunities to test parts in isolation are limited.

Connolly and Begg (p. 264) offer a graphical summary of the development process to show where the database development activity is concentrated (see next page).

In a real development project, each of the activities represented by boxes in the diagram might have its own documentation and sign-off procedures. In a one-person project many of these are not required. In this module, we are primarily concerned with the activities in the lighter box labelled *Database design*. Before looking in more detail at the three phases of database design, however, there are some useful observations to make about this breakdown of the development process.

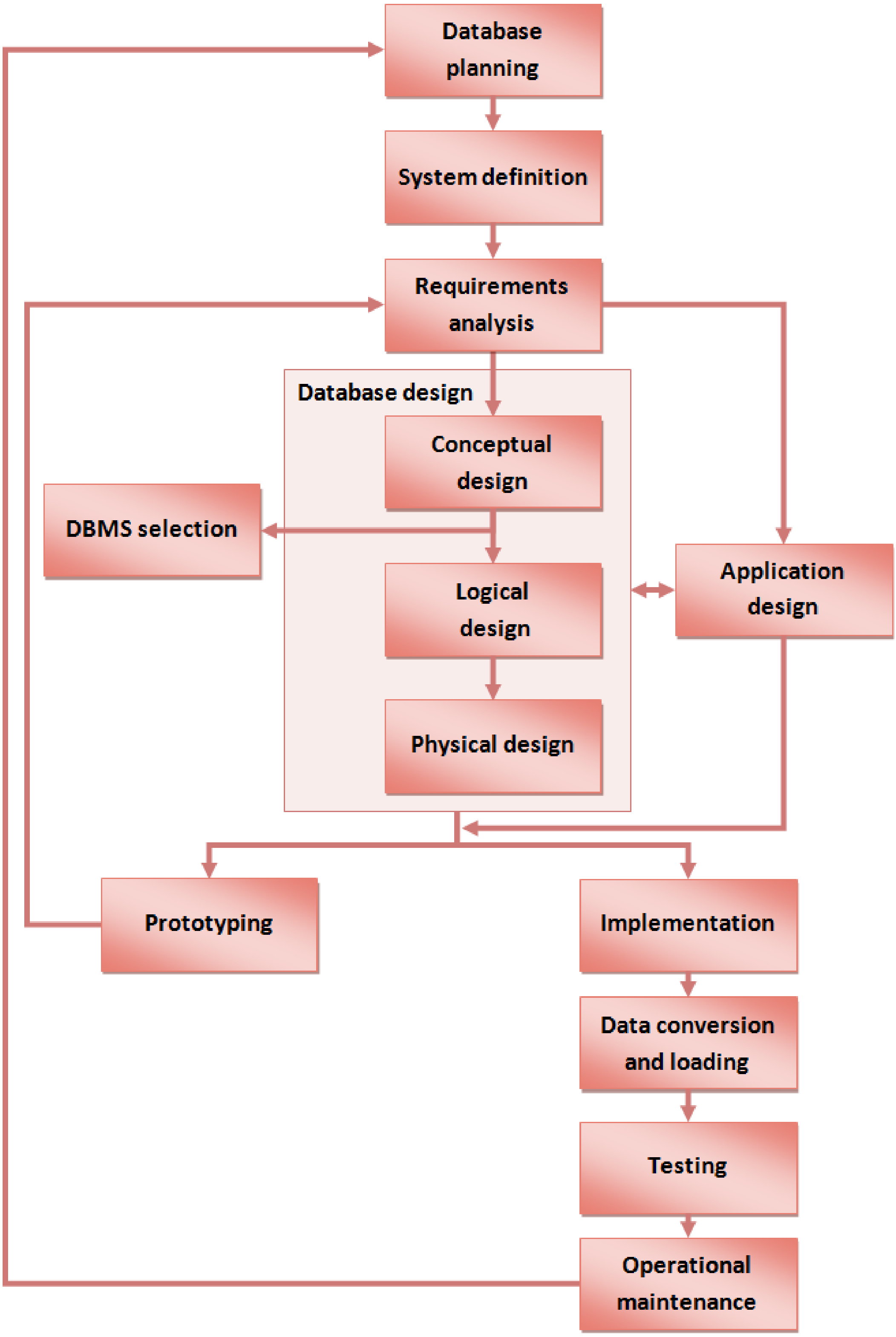
Those activities that appear above the *Database design* box are primarily concerned with developing an understanding of the project requirements. It is during the *Requirements analysis* activity that user interviews will be conducted to identify entities, attributes relationships and constraints.

The four activities in the lower right-hand corner are those that take place once the development is complete. *Implementation* in this context refers to the task of creating a live version of the development system. *Data conversion and loading* is where the data required for the operation of the new system is retrieved from its current location and any modifications are done before storing in the new database. *Testing* in this context means ensuring that the live system works as expected, that is to say, in comparison with the development system. It does not refer to the usual testing that is done throughout the development process.

Application design and *Prototyping* refer to the other development activities required to produce the user interface and other components.

A specific activity for DBMS selection appears on the diagram because there may be special requirements that arise out of the *Database design* process.

All of these activities are described in greater detail in chapter 10 of Connolly and Begg. If you are up to date on the rest of the work for the module, this would be a good time to read through that chapter. Because you are working individually on small artificial projects, many of these activities will seem to have little point just now; however, a good appreciation of how the work of designing the database fits in with the rest of a development project is vital in a team situation. Remember that in third year, you may be involved in a group project: these additional activities will have more relevance at that time.


Database planning
System definition
Requirements analysis
Database design
Conceptual design
DBMS selection
Logical design
Application design
Physical design
Prototyping
Implementation
Data conversion and loading
Testing
Operational maintenance

Conceptual design

The word *conceptual* implies an abstraction away from the messy complications of reality. It suggests a pleasing and common-sense fit with the way something is generally understood. That is the case here, and differentiates the process of conceptual design from logical design (discussed later).

Recall that the ANSI-SPARC model has three levels, external, conceptual and internal. The process of conceptual design is concerned with establishing the data model at the middle level which is also known as the *community view*. The reason for this additional name is that this level must capture all the user requirements in terms of data and structure in a single, well-structured way. How do you check that the community view is accurate? You check with the community, of course. This means that your conceptual model may include features such as *:* relationships that are not possible to implement directly with relational technology *if that is how the users understand the data*. The conceptual model, therefore is all about communication: it is your synthesis of the information you have collected from users, and it must correspond with their understanding of the subject area.

As well as *:* relationships, we have also mentioned one or two other examples of structures that require further work before they can be implemented with relational technology. One of them is the generalisation/specialisation relationship that may exist between entities. This relationship might be explicit in a conceptual model, but might be implemented in different ways by a set of database tables.

Connolly and Begg suggest the following process for conceptual design:

1. Identify entity types
2. Identify relationship types
3. Identify and associate attributes with entity or relationship types
4. Determine attribute domains
5. Determine candidate, primary and alternate key attributes
6. Consider using enhanced modelling concepts
7. Check model for redundancy
8. Validate conceptual model against user transactions
9. Review conceptual data model with user

All of these steps should now be familiar. If not check back to the relevant section in these notes, or read section 16.3 of Connolly and Begg.

Logical design

Logical design is essentially the process of making the conceptual design work once it has been agreed by the users. Whereas the conceptual design needs to reflect the users' understanding of the world, the logical design needs to provide the detail on how the model can be represented using relational technology.

The starting point for the logical design should be the ER diagram from the conceptual design stage. The outputs from the logical design stage are a new ER diagram that reflects the actual relational implementation, and a data dictionary.

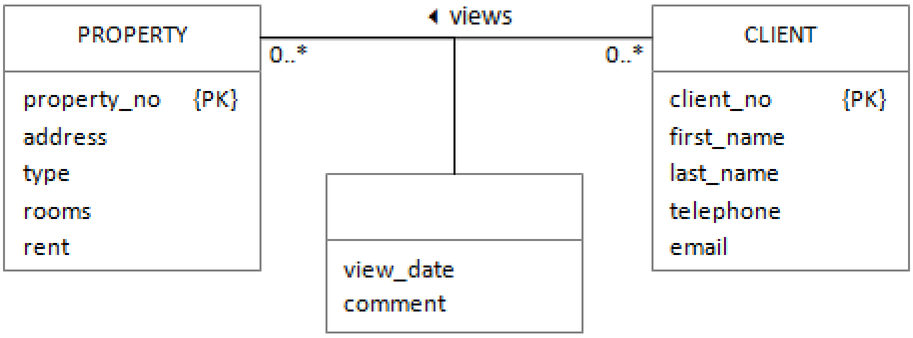
ER diagram

Confusion can arise because there are two ER diagrams involved. Remember that the first reflects the community view of the data and must be comprehensible to the users. The second represents the technical view of the model which takes into account the requirements of the relational platform. This level of detail is usually too difficult for users to understand. Possible causes of confusion for users might be:

- The inclusion of foreign keys
- The existence of link entities introduced to resolve *:* relationships
- The replacement of generalisation/specialisation relationships with a practical equivalent

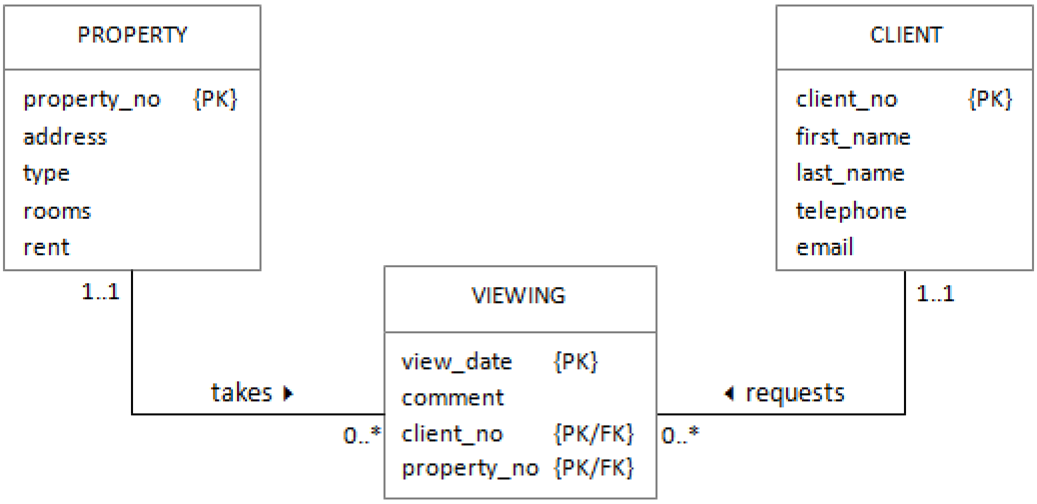
The following example is taken from the worked example used all the way through Connolly and Begg. If you are finding it difficult to follow any of the processes discussed here, please go through this example in the book. Each stage of refining the data model is discussed in detail with examples.

When designing a database for a property agency, it is established that PROPERTY and CLIENT are two important entity types. A CLIENT can view a PROPERTY, and when they do, they can leave a comment. The conceptual model might be represented as shown below:



Notice that there is a *:* relationship between the two entities. Notice, too, that the relationship itself has some attributes associated with it. From what you know of relational structure, you should realise that this is not possible: attributes are only found in tables and relationships are represented by pairs of keys. In the conceptual model, though, this is perfectly valid. The purpose of the conceptual model is to understand the world as the users see it.

During the logical design stage, the challenge is to take the agreed conceptual model and map it onto a set of tables that will give an equivalent description of the subject area. You know a third table is required to resolve the *:* relationship, and the 'floating' attributes can be included. The logical model would therefore look like this:



Connolly and Begg suggest the following process for logical design:

1. Derive relations for the logical model
2. Validate relations using normalisation (this is covered in week 6)
3. Validate relations against user transactions
4. Check integrity constraints
5. Review logical model with the user

Data dictionary

The data dictionary is a detailed catalogue of the data requirements for every table and every column that will be used in the database. There is no standard format, but a table like the one below is perfectly adequate:

Table name	Column name	Contents	Type	Format	Domain	Mandatory	Key	Reference
PROPERTY	property_no	Unique id	Integer	09999	1-99999	Y	PK	
	address	Postal address	Text	Xxxxx		Y		
	type	Type code	Text	X	F, H	Y		
	rooms	Number of rooms	Integer	09	1-16	Y		
	rent	Monthly rent	Money	£0999	100-2000	Y		
CLIENT	client_no	Unique id	Integer	09999	1-99999	Y	PK	
	first_name	First name	Text	Xxxxx				
	last_name	Last name	Text	Xxxxx		Y		
	telephone	Telephone number	Text		[0-9],+,(,), space	Y		
	email	email address	Text					
VIEWING	view_date	Date of viewing	Date	dd/mm/yyyy		Y	PK	
	comment	Notes	Text					
	client_no	Client reference	Integer	09999	1-99999	Y	PK/ FK	CLIENT
	property_no	Property reference	Integer	09999	1-99999	Y	PK/ FK	PROPERTY

The data dictionary duplicates some of the information on the ER diagram, but this is useful for cross-checking. Any corrections should be applied to both documents.

The data dictionary also provides additional information which is used in the next design stage. The term data dictionary is also used in another sense which is briefly discussed in week 9.

Physical design

The conceptual model is supposed to be independent of any physical considerations. The logical model is supposed to take account only of the fact that we are using a relational database. The physical design process, however, fine-tunes the model to suit a particular database platform which might include:

- Selecting datatype definitions for each column
- Designing constraints on the data
- Designing user views (covered in week 8)
- Designing indexes to improve query times (covered in week 8)

In addition, the physical design process investigates such practical details as:

- How the database files should be spread across one or more disks
- How much storage space is required
- How the data is likely to grow over time
- Designing security mechanisms (covered in week 11)
- Platform-specific performance tuning

Many of these activities fall to the database administrator (DBA), and are discussed in week 9. Much of the detail, however, is beyond the scope of this module.