Relational Database Systems (Higher)

Chapter 1 briefly summarises the material covered at Intermediate 2 level. Candidates without prior attainment of this unit at Intermediate 2 may also find it helpful to have access to the Section 3 materials.

Chapter 1: Database fundamentals

Definitions

A database is a collection of related information about a set of persons or objects.

A database management system (DBMS) is a software package which is used to create, manipulate and present data from electronic databases. Example include Microsoft Access, Filemaker Pro, MySQL, Oracle, Ingres, Postgres, SQL Server and many others.

Traditional databases

Problems associated with manual paper-based databases:

• the storage of paper records was very bulky, often requiring several large filing cabinets
• data was often duplicated in several records
• keeping records up-to-date was difficult and time consuming, and often resulted in data inconsistency, where duplicated values were updated in one record but not in others
• many people were employed to maintain the records, which was costly
• searching for records was time consuming
• producing reports, such as sorted lists or data collated from several sources, was extremely time consuming, if not impossible.

Computerised databases were developed in order to address these problems. In the following pages, the term ‘database’ is used to mean a computerised database.
Benefits of using databases

Computerised databases have several advantages over manual databases.

- Searching and sorting operations can be performed much more quickly and easily.
- Information is more easily available to users, due to improved methods of data retrieval.
- Security and integrity of data is improved, due to the methods of data storage and updating.
- No data redundancy (data duplication). In a relational database, data is never stored more often than is necessary.
- Data consistency. This means that when data is updated, this is only done once, so removing the possibility of data inconsistency.
- Data independence. In a DBMS, the software that processes the data is separate from the data itself. This means that several programs can be developed to perform operations on the data without affecting the data itself.

Case study: DVD rentals

Here is a review of the DVD rentals example that was covered in Section 3.

Ali’s Mini Market is a local store with a DVD rental section. To rent a DVD, customers must register as members.

Ali records the details of members on a member list, part of which is shown in Figure 1.1.

**Figure 1.1: DVD rentals: member list**

<table>
<thead>
<tr>
<th>Member number</th>
<th>Title</th>
<th>Forename</th>
<th>Surname</th>
<th>Telephone no.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1012</td>
<td>Miss</td>
<td>Isobel</td>
<td>Ringer</td>
<td>293847</td>
</tr>
<tr>
<td>1034</td>
<td>Mr</td>
<td>John</td>
<td>Silver</td>
<td>142536</td>
</tr>
<tr>
<td>1056</td>
<td>Mr</td>
<td>Fred</td>
<td>Flintstone</td>
<td>817263</td>
</tr>
<tr>
<td>1097</td>
<td>Mrs</td>
<td>Annette</td>
<td>Kirton</td>
<td>384756</td>
</tr>
</tbody>
</table>

Ali has a DVD rentals list that is used to record details of rentals for each DVD. Part of the DVD rentals list is shown in Figure 1.2.
**Figure 1.2: DVD rentals list**

<table>
<thead>
<tr>
<th>DVD code</th>
<th>Title</th>
<th>Cost</th>
<th>Date Out</th>
<th>Date Due</th>
<th>Member number</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>002</td>
<td>Finding Nemo</td>
<td>£2.50</td>
<td>03/09/04</td>
<td>04/09/04</td>
<td>1034</td>
<td>John Silver</td>
</tr>
<tr>
<td>003</td>
<td>American Pie</td>
<td>£2.50</td>
<td>27/08/04</td>
<td>28/08/04</td>
<td>1056</td>
<td>Fred Flintstone</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01/09/04</td>
<td>02/09/04</td>
<td></td>
<td>Isobel Ringer</td>
</tr>
<tr>
<td>008</td>
<td>The Pianist</td>
<td>£2.50</td>
<td>04/09/04</td>
<td>06/09/04</td>
<td>1097</td>
<td>Annette Kirton</td>
</tr>
<tr>
<td>011</td>
<td>Notting Hill</td>
<td>£2.50</td>
<td>27/08/04</td>
<td>28/08/04</td>
<td>1012</td>
<td>I Ringer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>04/09/04</td>
<td>06/09/04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>014</td>
<td>Prime Suspect</td>
<td>£2.00</td>
<td>27/08/04</td>
<td>28/08/04</td>
<td>1086</td>
<td>Annette Kirton</td>
</tr>
<tr>
<td>015</td>
<td>Shrek</td>
<td>£1.50</td>
<td>10/09/04</td>
<td>11/09/04</td>
<td>1034</td>
<td>Joan Silver</td>
</tr>
</tbody>
</table>

**Entities and relationships**

In the DVD rentals system, MEMBER and DVD RENTAL are called **entities**. An **entity** represents a person or object. Each entity has a set of **attributes** that describe examples or **instances** of that entity. The attributes of the DVD RENTAL entity are Code, Title, Cost, Date Out, Date Due and Member Number, and the attributes of the MEMBER entity are Member Number, Name and Telephone Number.

In a manual system, attributes may contain several values. For example, in Ali’s DVD rentals list in Figure 1.2, the attributes Date Out, Date Due, Member Number and Name have more than one entry for DVDs 003 and 011. These are called **multi-valued attributes**.

In a computerised database, **single-valued attributes** are used in preference to multi-valued attributes.

The MEMBER **entity** is the whole table.

**Figure 1.3: Entities, attributes and instances**

The MEMBER **entity** is the whole table

Each column stores one **attribute**, e.g. Member Name

<table>
<thead>
<tr>
<th>MEMBER</th>
<th>Member name</th>
<th>Telephone number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1012</td>
<td>Isobel Ringer</td>
<td>293847</td>
</tr>
<tr>
<td>1034</td>
<td>John Silver</td>
<td>142536</td>
</tr>
<tr>
<td>1056</td>
<td>Fred Flintstone</td>
<td>817263</td>
</tr>
<tr>
<td>1097</td>
<td>Annette Kirton</td>
<td>384756</td>
</tr>
</tbody>
</table>
The entity and its attributes are written as:
MEMBER(Member Number, Title, Forename, Surname, Telephone Number)

DVD RENTAL(DVD Code, Title, Cost, Date Out, Date Due, Member Number, Name, Telephone Number)

Note that the entity name is usually written in CAPITALS and in the singular rather than the plural form, i.e. MEMBER not MEMBERS. The attributes are listed in the brackets. Sometimes it is easier to write the attributes one underneath the other.

When a DVD is rented, a link is made between a DVD RENTAL and a MEMBER. This is called a relationship. Each member can rent out many DVDs, but each DVD can only be rented by one member at a time, so we say there is a one-to-many relationship between MEMBER and DVD RENTAL.

This is sometimes written as... MEMBER 1:M DVD RENTAL

1:M stands for one-to-many.

Figure 1.4 shows a diagram of the relationship between MEMBER and DVD RENTAL. This is called an entity-relationship diagram. The line joining the entities is called a crow’s foot, and the ‘toes’ are at the ‘many’ end of the relationship.

*Figure 1.4: Entity-relationship diagram showing a one-to-many relationship*
There are other types of relationship between entities. For example, every vehicle has a registration number, and each registration number corresponds to a single vehicle, so there is a one-to-one relationship between VEHICLE and REGISTRATION NUMBER. In a school, every pupil has many teachers, and each teacher teaches many pupils, so there is a many-to-many relationship between PUPIL and TEACHER.

These relationships can be written as

VEHICLE 1:1 REGISTRATION NUMBER

PUPIL M:N TEACHER

M:N stands for many-to-many (the number of teachers and pupils may be different, which is why N is used instead of M)

*Figure 1.5: Entity-relationship diagrams showing one-to-one and many-to-many relationships*

**Data modelling** is the process of defining the entities, attributes and relationships that are required. The result is called a *data model*.

**Exercise 1**

1. Explain what is meant by *data inconsistency* in a database. Give an example to illustrate your answer.
2. Explain why flat file databases often result in data inconsistencies.
Chapter 2: The relational model

The main idea behind a relational database is that data is grouped into entities that are related, in order to minimise data duplication and achieve data integrity.

In order to create a relational database, it is necessary first to produce a data model to represent the data being stored. The data model will indicate which entities are required, which attributes belong to each entity, and what relationships exist between the entities.

A significant feature of the relational model is that, in order to implement the data model, any many-to-many relationships between entities are removed and replaced with one-to-many relationships.

Entity occurrence

One method of identifying the relationships between entities is called entity occurrence modelling. It can be a useful technique if you have some sample data to work with.

*Figure 2.1: Entity occurrence diagram for the DVD rentals system shown in Figures 1.1 and 1.2.*
The lines between the entities show how the instances of each entity are linked. For example, they show that member 1034 has rented DVDs 002 and 015, and that DVD 003 has been rented by members 1012 and 1056.

By looking at the DVD RENTAL entity, we can see that some DVDs (e.g. 003 and 011) are related to more than one member, because each DVD can be rented by many members.

By looking at the MEMBER entity, we can see that many members have rented more than one DVD. Each member can rent many DVDs.

Taken together, this means that there is a many-to-many relationship between Member and DVD.

The relational model does not permit many-to-many relationships. Instead, a process called normalisation breaks down the many-to-many relationship into two separate one-to-many relationships.

A word of caution

Entity occurrence modelling is only as effective as the data that is available and can be misleading. For example, if the data had only included members who had rented one single DVD, then the entity-occurrence diagram would have suggested a one-to-many relationship, even though it is possible for members to rent more than one.

In this situation, ‘dummy’ data can be made up to complete the picture, e.g. adding a link between a member and another DVD to show that it is possible for a member to rent more than one DVD.

From data model to database

Once the entities, attributes and relationships in the data model have been defined, a database can be created.

Each entity in the data model becomes a table in the database. Each attribute of an entity becomes a field in a table in the database.
More about keys

In order to establish the relationships between the tables in the database, each entry, or record, in a table must be able to be uniquely identified by a key. A key is a field, or set of fields, the values of which uniquely identify a record. In any table, there may be more than one field, or set of fields, which can uniquely identify each record—these are called candidate keys. The candidate key that is chosen to be used is called the primary key.

In our DVD rentals example, Member Number is a candidate key for the MEMBER entity. The key is identified by underlining it, as shown:

MEMBER(Member Number, Name, Telephone Number)

In the DVD RENTALS entity, the Member Number attribute is called a foreign key. A foreign key is a field that is not a primary key in its own table, but is a primary key in another table.

In this example, Member Number is a foreign key in DVD RENTAL, because it is the primary key in MEMBER.

A key consisting of only one field is called an atomic key. In this example, Member Number is an atomic key.

A key that is made up of a set of fields is called a compound key.

For example, the combination of Name and Telephone Number is a compound key that is also a candidate key. This compound key is identified by underlining both attributes:

MEMBER(Member Number, Name, Telephone Number)

Sometimes it is necessary to ‘make up’ a key to uniquely identify each record. This is called a surrogate key. A surrogate key is usually a number, or similar code, that has no meaning or purpose other than to uniquely identify each record. Surrogate keys are used when there is no candidate key available, or when the only candidate key is a large compound key.

In the above example, Member Number would be a surrogate key. This type of surrogate key is commonly used, and its purpose is to easily and uniquely identify each member without having to refer to the member’s name or telephone number.
Choosing a key

What makes a good key? In general, the key should be as short and simple as possible.

• An atomic key is a better choice than a compound key.
• A numeric field is a better choice of key than a text field. This is because they can be stored more compactly in the database than text fields.

This can be summarised as the KISS rule:

KISS = Keep It Short and Simple

A key must also have the following properties:

• it must have a value—it can never be blank (or ‘null’).
• it should not change over time.

This last property can be a problem if a key is also a meaningful identifier. For example, consider a key, Employee Code, to identify a company's employees according to the office they work in, as shown below.

<table>
<thead>
<tr>
<th>Employee Code</th>
<th>Name</th>
<th>Office</th>
</tr>
</thead>
<tbody>
<tr>
<td>G024</td>
<td>J Smith</td>
<td>Glasgow</td>
</tr>
<tr>
<td>D009</td>
<td>K Rossi</td>
<td>Dundee</td>
</tr>
<tr>
<td>E017</td>
<td>S Swami</td>
<td>Edinburgh</td>
</tr>
<tr>
<td>A006</td>
<td>R Patel</td>
<td>Aberdeen</td>
</tr>
</tbody>
</table>

Meaningful identifiers like this can be very useful. The Employee Code shown above can allow someone to instantly identify which office an employee works in. This would not have been possible if the Employee Code had not included a letter indicating the office.

However, suppose that K Rossi moves from the Dundee office to Glasgow. The key will need to change, causing all sorts of problems. A meaningful identifier should only be used as a key if it can be guaranteed not to change. For example, a person’s driving licence number is made up of their initials, surname, and date of birth to make a unique identifier. A person called J R Hartley, born on 25 April 1963, might have a driving licence number:

HARTL 604253 JR4RQ
This is a meaningful identifier, because we can work out someone’s name (at least the first five letters of their surname and initials) and date of birth from the identifier. The identifier is not likely to change, which makes this a suitable key. (If someone does change their name, usually as a result of getting married, a new driving licence must be issued).

**Update anomalies**

In a flat file database, the data is stored in a single table. Like a manual card index file, a flat file contains one card or record for each entry in the file.

*Figure 2.2: A flat file database for the DVD rentals system*

<table>
<thead>
<tr>
<th>DVD code</th>
<th>Title</th>
<th>Cost</th>
<th>Date out</th>
<th>Date in</th>
<th>Member number</th>
<th>Name</th>
<th>Telephone number</th>
</tr>
</thead>
<tbody>
<tr>
<td>002</td>
<td>Finding Nemo</td>
<td>£2.50</td>
<td>03/09/04</td>
<td>04/09/04</td>
<td>1034</td>
<td>John Silver</td>
<td>142536</td>
</tr>
<tr>
<td>003</td>
<td>American Pie</td>
<td>£2.50</td>
<td>27/08/04</td>
<td>28/08/04</td>
<td>1056</td>
<td>Fred Flintstone</td>
<td>817263</td>
</tr>
<tr>
<td>003</td>
<td>American Pie</td>
<td>£2.50</td>
<td>01/09/04</td>
<td>02/09/04</td>
<td>1012</td>
<td>Isobel Ringer</td>
<td>293847</td>
</tr>
<tr>
<td>008</td>
<td>The Pianist</td>
<td>£2.50</td>
<td>04/09/04</td>
<td>06/09/04</td>
<td>1097</td>
<td>Annette Kirton</td>
<td>384756</td>
</tr>
<tr>
<td>011</td>
<td>Notting Hill</td>
<td>£2.50</td>
<td>27/08/04</td>
<td>28/08/04</td>
<td>1012</td>
<td>Isobel Ringer</td>
<td>293847</td>
</tr>
<tr>
<td>011</td>
<td>Notting Hill</td>
<td>£2.50</td>
<td>04/09/04</td>
<td>06/09/04</td>
<td>1056</td>
<td>Fred Flintstone</td>
<td>817263</td>
</tr>
<tr>
<td>014</td>
<td>Prime Suspect</td>
<td>£2.00</td>
<td>27/08/04</td>
<td>28/08/04</td>
<td>1097</td>
<td>Annette Kirton</td>
<td>384756</td>
</tr>
<tr>
<td>015</td>
<td>Shrek</td>
<td>£1.50</td>
<td>10/09/04</td>
<td>11/09/04</td>
<td>1034</td>
<td>Joan Silver</td>
<td>142536</td>
</tr>
</tbody>
</table>

There are a number of problems with flat file databases.

- Data is very likely to be duplicated. For example, because a new record would be created for each DVD rented, this means that the member details would have to be copied onto each new record.

- The duplication of data leads to the possibility of **data inconsistency**. This happens when the same data, stored on two separate records, differs. Usually this is the result of human error in copying the data. It then becomes difficult or impossible to know which record contains the correct version of the data.
The problem of data inconsistency is only one of the results of the design of the table above.

- There is no way of storing the details of a member who hasn’t rented any DVDs. If the DVD Code is the key of the table, then a DVD Code must be provided. This is called an **insertion anomaly**.

- If a member’s details have to be amended, for example a new telephone number, then the update must be performed several times, in each record where that member’s details are recorded. This can lead to inconsistencies and errors. This is called a **modification anomaly**.

- If a DVD is removed from the database, perhaps to make room for more recent movies, then it may also result in removing the only recorded details for a member. This is called a **deletion anomaly**.

These three types of problems are known collectively as **update anomalies**.

The solution to the problems of update anomalies found in flat file databases is to use a **relational database**. A relational database stores data in more than one table. The idea is to ensure that data is only entered and stored once, so removing the possibility of data duplication and inconsistency.

A process called **normalisation** is used to work out what tables are required and which data items should be stored in each table.
**Exercise 2**

The following table shows some data about classes undertaken by students at Hogwarts School of Witchcraft and Wizardry.

<table>
<thead>
<tr>
<th>Student</th>
<th>Student code</th>
<th>Course</th>
<th>Course code</th>
<th>Teacher</th>
</tr>
</thead>
<tbody>
<tr>
<td>V Crabbe</td>
<td>S602</td>
<td>Defence Against the Dark Arts</td>
<td>GL</td>
<td>G Lockhart</td>
</tr>
<tr>
<td>H Granger</td>
<td>G470</td>
<td>Care of Magical Creatures</td>
<td>RH</td>
<td>R Hagrid</td>
</tr>
<tr>
<td>D Malfoy</td>
<td>S616</td>
<td>Defence Against the Dark Arts</td>
<td>GL</td>
<td>G Lockhart</td>
</tr>
<tr>
<td>P Patil</td>
<td>G475</td>
<td>Charms</td>
<td>FF</td>
<td>F Flitwick</td>
</tr>
<tr>
<td>H Potter</td>
<td>G476</td>
<td>Potions</td>
<td>SS</td>
<td>S Snape</td>
</tr>
<tr>
<td>H Potter</td>
<td>G476</td>
<td>Divination</td>
<td>ST</td>
<td>S Trelawney</td>
</tr>
<tr>
<td>R Weasley</td>
<td>G478</td>
<td>Transfiguration</td>
<td>MM</td>
<td>M McGonagall</td>
</tr>
</tbody>
</table>

The Student Code and Course Code are unique to each student and course, respectively. The Student Code is based on the house into which the student has been placed (e.g. G472 indicates Gryffindor house), and the Course Code consists of the teacher’s initials (e.g. SS indicates Severus Snape).

1. Using the data provided to illustrate your answer, explain why a database based on this table would exhibit:

   (a) an insertion anomaly
   (b) a deletion anomaly
   (c) an update anomaly.

2. Identify a suitable candidate key for the table shown.

3. (a) Using the sample data provided, draw an entity occurrence diagram to illustrate the relationship between Student and Course. Use Student Code and Course Code to identify the instances of each entity.
   (b) State the relationship between Student and Course.

4. (a) Explain why the meaningful identifier Student Code is an appropriate key for identifying a Student.
   (b) Explain why the meaningful identifier Course Code is **not** an appropriate key for identifying a Course.
Chapter 3: Normalisation

We have seen how the use of multiple tables in a database can reduce data duplication and inconsistencies.

The question of which tables are required is not always obvious. A process called normalisation is used to accomplish this. To illustrate the process, consider Figure 3.1.

Figure 3.1: Mail-order invoice

<table>
<thead>
<tr>
<th>Bits‘n’PCs Limited</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order Number: 654321</td>
</tr>
<tr>
<td>Customer Number: 234567</td>
</tr>
<tr>
<td>Customer Name: Mrs Joanna Bloggs</td>
</tr>
<tr>
<td>Address: 12 High Street, Auchenshoogle, Inverness-shire</td>
</tr>
<tr>
<td>Post Code: IV99 2QW</td>
</tr>
<tr>
<td>Telephone No.: 01999 123456</td>
</tr>
</tbody>
</table>

You ordered the following items:

<table>
<thead>
<tr>
<th>Item Code</th>
<th>Description</th>
<th>Unit Cost</th>
<th>Quantity</th>
<th>Sub-Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1170</td>
<td>Medium widgets (blue)</td>
<td>£2.25</td>
<td>3</td>
<td>£6.75</td>
</tr>
<tr>
<td>5499</td>
<td>Large deluxe widget (red)</td>
<td>£9.99</td>
<td>1</td>
<td>£9.99</td>
</tr>
<tr>
<td>7937</td>
<td>Small economy widget (green)</td>
<td>£0.59</td>
<td>5</td>
<td>£2.95</td>
</tr>
</tbody>
</table>

TOTAL: £19.69
Un-normalised form

The first stage in the process of normalisation is to represent the data items required in un-normalised form (UNF). This simply involves writing down all the attributes that must be stored in the database, and identifying a key (shown underlined). Un-normalised form is also sometimes known as zero normal form (0NF).

*Figure 3.2: The data model for Fig 3.1 in UNF*

ORDER (Order Number
Order Date
Customer Number
Customer Name
Address
Post Code
Telephone Number
Item Code
Description
Unit Cost
Quantity)

Note that the attributes Sub-Total and Total are not included in the data model. This is because they are *derived* attributes, i.e. their values are calculated from the values of other attributes, and so they don’t need to be stored in the database.

If we were to represent the data in the order form in a table with the fields listed in Figure 3.2, it would look something like this:
It is clear that, because the order is for several items, there is more than one value in each of the fields Item Code, Description, Unit Cost and Quantity for this Order Number. These are called multi-valued attributes or repeating data items.

**Figure 3.4: The data model in UNF with repeating data items identified**

ORDER  
(Order Number  
Order Date  
Customer Number  
Customer Name  
Address  
Post Code  
Telephone Number  
Repeating items  
Item Code  
Description  
Unit Cost  
Quantity)
First normal form

The next stage of normalisation is first normal form (1NF). To produce a data model in 1NF, we must remove all multi-valued attributes, so we remove the repeating data items to form a new entity, as shown in Figure 3.5.

Figure 3.5: The data model in 1NF

<table>
<thead>
<tr>
<th>ORDER</th>
<th>(Order Number, Order Date, Customer Number, Customer Name, Address, Post Code, Telephone Number)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDER_ITEM</td>
<td>(*Order Number, Item Code, Description, Unit Cost, Quantity)</td>
</tr>
</tbody>
</table>

Note that the key field Order Number has been copied from the ORDER entity to the new ORDER_ITEM entity. If this were not done, then there would be no link between the items ordered and the order itself.

However, Order Number by itself is not a correct key for the ORDER_ITEM entity, as there will be many items for each order. The **compound key** (Order Number, Item Code) forms the primary key for this entity.

Because Order Number is the primary key for the ORDER entity, it is shown as a **foreign key** (marked with an asterisk) in the ORDER_ITEM entity. Order Number is both part of the compound primary key and also a foreign key.

A data model is in 1NF if it has no **multi-valued attributes**.
Figure 3.6: Data in the normalised ORDER and ORDER_ITEM tables in 1NF

ORDER

<table>
<thead>
<tr>
<th>Order Number</th>
<th>Order Date</th>
<th>Customer Number</th>
<th>Customer Name</th>
<th>Address</th>
<th>Post Code</th>
<th>Telephone Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>654321</td>
<td>1/9/04</td>
<td>234567</td>
<td>Mrs Joanna Bloggs</td>
<td>12 High Street Auchenshoogle Inverness-shire</td>
<td>IV99 123456</td>
<td></td>
</tr>
</tbody>
</table>

ORDER_ITEM

<table>
<thead>
<tr>
<th>Order Number</th>
<th>Item Code</th>
<th>Description</th>
<th>Unit Cost</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>654321</td>
<td>1170</td>
<td>Medium widgets (blue)</td>
<td>£2.25</td>
<td>3</td>
</tr>
<tr>
<td>654321</td>
<td>5499</td>
<td>Large deluxe widget (red)</td>
<td>£9.99</td>
<td>1</td>
</tr>
<tr>
<td>654321</td>
<td>7937</td>
<td>Small economy widget (green)</td>
<td>£0.59</td>
<td>5</td>
</tr>
</tbody>
</table>

In Figure 3.6, you can see that the repeating data items from the table in Figure 3.3 have been removed. Notice that Order Number is duplicated in each entry in the ORDER_ITEM table – this is a necessary duplication, as it links each item to the order in which it was placed.

Now suppose there are several orders for large red deluxe widgets. The ORDER_ITEM table might look like Figure 3.7.

Figure 3.7: Sample data in the ORDER_ITEM table in 1NF

<table>
<thead>
<tr>
<th>Order Number</th>
<th>Item Code</th>
<th>Description</th>
<th>Unit Cost</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>654321</td>
<td>5499</td>
<td>Large deluxe widget (red)</td>
<td>£9.99</td>
<td>1</td>
</tr>
<tr>
<td>975310</td>
<td>5499</td>
<td>Large deluxe widget (red)</td>
<td>£9.99</td>
<td>2</td>
</tr>
<tr>
<td>864208</td>
<td>5499</td>
<td>Large deluxe widget (red)</td>
<td>£9.99</td>
<td>3</td>
</tr>
</tbody>
</table>

Clearly the Item Code, Description and Unit Cost values are duplicated in each entry. If the price of item 5499 were to change, this would have to be updated three times, which is both inefficient and could result in data inconsistency. This is a modification anomaly.
There are some other problems:

- it is not possible to enter details for an item which has not yet been ordered (because an order number is required as part of the compound key). This is an **insertion anomaly**.
- if an order is deleted (perhaps because it is cancelled), this could remove the only record containing an item’s details. This is a **deletion anomaly**.

To resolve these problems requires the next stage of normalisation.

**Second normal form**

To produce a data model in second normal form (2NF), examine any entity with a compound key (in this case ORDER_ITEM) to see if any attributes are dependent on just one part of the compound key. These are called **partial dependencies**.

In the ORDER_ITEM entity, we can see that:

- Order Number is part of the key
- Item Code is part of the key
- Description is always the same for any given Item Code, no matter what the Order Number is, so this is dependent on the Item Code
- Unit Cost is also always the same for any given Item Code, no matter what the Order Number is, so this is dependent on the Item Code
- Quantity refers to the Item Code, but will change for different orders, so it is dependent on both Order Number and Item Code.

To produce 2NF, any attributes which are dependent on a part of the compound key are removed to form a new entity. So in this case, Description and Unit Cost are removed to form a new entity. Figures 3.8 and 3.9 show the data model and sample data in 2NF.
Figure 3.8: The data model in 2NF

ORDER (Order Number
Order Date
Customer Number
Customer Name
Address
Post Code
Telephone Number)

ORDER_ITEM (*Order Number
*Item Code
Quantity)

ITEM (Item Code
Description
Unit Cost)

Note that the key field Item Code has been copied from the ORDER_ITEM entity to the new ITEM entity. If this were not done, then there would be no link between the items ordered and the description and unit cost of those items.

The attribute on which Description and Unit Cost were dependent (i.e. Item Code) becomes the primary key of the new entity. Because Item Code is now a primary key for the ITEM entity, it is shown as a foreign key in the ORDER_ITEM entity.

A data model is in 2NF if it is in 1NF and there are no partial dependencies.
Understanding dependencies

An attribute A is dependent on another attribute B if there is only one possible value for A given any value for B.

For example, in the mail order system, consider the relationship between Order Number and Order Date. Order Date is dependent on Order Number, because for any given Order Number, there is only one possible date (i.e. the date the order was placed).

However, the reverse is not true. Order Number is not dependent on Order Date, because for any given Order Date, there may be more than Order Number (because many orders may have been placed on that date).

In Figure 3.9, you can see that the duplicated data from the table in Figure 3.7 has been removed. The only data that is duplicated in each entry in the ORDER_ITEM table is the Item Code – but this time it is necessary duplication, as it tells us which item was ordered on each order.

By normalising the data in this way, we have resolved the insertion, deletion and modification anomalies that existed before.

• We can add an item to the ITEM table without it having to be on an order.
• We can delete an order in the ORDER table without deleting details of the items on the order.
• We can update item details once in the ITEM table without affecting the orders for that item in the ORDER_ITEM table.
Now suppose that a customer places several orders. The ORDER table might look like Figure 3.10.

**Figure 3.10: Sample data in the ORDER table in 2NF**

<table>
<thead>
<tr>
<th>Order Number</th>
<th>Order Date</th>
<th>Customer Number</th>
<th>Customer Name</th>
<th>Address</th>
<th>Post Code</th>
<th>Telephone Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>654321</td>
<td>1/9/04</td>
<td>234567</td>
<td>Mrs Joanna Bloggs</td>
<td>12 High Street Auchenshoogle</td>
<td>IV99 2QW</td>
<td>01999 123456</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Inverness-shire</td>
<td></td>
<td></td>
</tr>
<tr>
<td>135790</td>
<td>15/9/04</td>
<td>234567</td>
<td>Mrs Joanna Bloggs</td>
<td>12 High Street Auchenshoogle</td>
<td>IV99 2QW</td>
<td>01999 123456</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Inverness-shire</td>
<td></td>
<td></td>
</tr>
<tr>
<td>246801</td>
<td>28/9/04</td>
<td>234567</td>
<td>Mrs Joanna Bloggs</td>
<td>12 High Street Auchenshoogle</td>
<td>IV99 2QW</td>
<td>01999 123456</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Inverness-shire</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Clearly, the customer details are duplicated in each entry. If a customer’s address or telephone number were to change, this would have to be updated three times, which is both inefficient and presents the possibility of data inconsistency. This is a **modification anomaly**.

As before, there are some other problems.

- It is not possible to enter details for a customer unless they have placed an order (because Order Number is required as a key). This is an **insertion anomaly**.
- If an order is deleted, this could remove the only record containing a customer’s details. This is a **deletion anomaly**.

To resolve these problems requires the final stage of normalisation.
Third normal form

To produce a data model that is in third normal form (3NF), we must examine all the entities produced so far to see if there are any non-key attributes which are dependent on any other non-key attributes. These are called non-key dependencies (or an indirect or transitive dependency).

In the ORDER table, we can see that Customer Name, Address, Post Code and Telephone Number are all dependent on Customer Number. These attributes can be removed to form a new entity CUSTOMER, as shown in Figure 3.11.

Figure 3.11: The data model in 3NF

ORDER
(Order Number
Order Date
*Customer Number)

CUSTOMER
(Customer Number
Customer Name
Address
Post Code
Telephone Number)

ORDER_ITEM
(*Order Number
*Item Code
Quantity)

ITEM
(Item Code
Description
Unit Cost)

Note that the key field Customer Number has been copied from the ORDER entity to the new CUSTOMER entity. If this were not done, then there would be no link between the order and the customer.

The attribute on which the customer details were dependent (i.e. Customer Number) becomes the primary key of the new entity. Because Customer Number is now a primary key for the CUSTOMER entity, it is shown as a foreign key in the ORDER entity.

A data model is in 3NF if it is in 2NF and there are no non-key dependencies.
In Figure 3.12, you can see that the duplicated data from the table in Figure 3.10 has been removed. The only data that is duplicated in each entry in the ORDER table is the Customer Number—but as before it is necessary duplication, as it tells us which customer placed each order.

*Figure 3.12: Data in the normalised ORDER and CUSTOMER tables in 3NF*

<table>
<thead>
<tr>
<th>ORDER</th>
<th>CUSTOMER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order Number</td>
<td>Order Date</td>
</tr>
<tr>
<td>654321</td>
<td>1/9/04</td>
</tr>
<tr>
<td>135790</td>
<td>15/9/04</td>
</tr>
<tr>
<td>246801</td>
<td>28/9/04</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CUSTOMER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer Number</td>
</tr>
<tr>
<td>234567</td>
</tr>
</tbody>
</table>

By normalising the data in this way, we have resolved the insertion, deletion and modification anomalies that existed before.

- We can add a customer to the CUSTOMER table without the customer having to place an order.
- We can delete an order in the ORDER table without deleting details of the customer who placed the order.
- We can update a customer's details once in the CUSTOMER table without affecting the orders placed by that customer in the ORDER table.
Here is a summary of the normalisation process, from un-normalised form (UNF) to third normal form (3NF). Note that, at each new stage, only the entities which have been further normalised have been shown. The arrows indicate where entities are unchanged from one stage to the next.

This is one way of representing the normalisation process, which avoids writing down all the entities at every stage!

<table>
<thead>
<tr>
<th>UNF</th>
<th>1NF</th>
<th>2NF</th>
<th>3NF</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDER (Order Number, Customer Number, Order Date, Customer Name, Address, Post Code, Telephone Number)</td>
<td>ORDER (Order Number, Order Date, Customer Number, Customer Name, Address, Post Code, Telephone Number)</td>
<td>ORDER (Order Number, Order Date, Customer Number, Customer Name, Address, Post Code, Telephone Number)</td>
<td>ORDER (Order Number, Order Date, Customer Number, Customer Name, Address, Post Code, Telephone Number)</td>
</tr>
<tr>
<td>ORDER_ITEM (*Order Number, Item Code, Description, Unit Cost, Quantity)</td>
<td>ORDER_ITEM (*Order Number, Item Code, Description, Unit Cost, Quantity)</td>
<td>ITEM (Item Code, Description, Unit Cost, Quantity)</td>
<td>ITEM (Item Code, Description, Unit Cost, Quantity)</td>
</tr>
</tbody>
</table>

**Memory Aid**

In each entity in third normal form, each attribute is dependent on:

- the key (in 1NF by removing multi-valued attributes)
- the whole key (in 2NF by removing partial dependencies)
- and nothing but the key (in 3NF by removing non-key dependencies)
Entity-relationship diagram

*Figure 3.13: An entity-relationship diagram for the data model in 3NF*

![E-R Diagram](image)

In the E-R diagram in Figure 3.13, the relationships shown are all one-to-many (1:M). Note that entity occurrence modelling would indicate that there is a many-to-many relationship between the ORDER and ITEM entities, because each order can include many items, and each item can be included on many orders.

A relational database cannot be implemented with many-to-many relationships. One of the results of normalisation is that many-to-many relationships are removed, and each is broken down into one-to-many relationships.

In Figure 3.13, the many-to-many relationship between ORDER and ITEM has been replaced by creating a new entity ORDER_ITEM, which was introduced at 1NF. This process is shown in Figure 3.14.

ORDER_ITEM is known as a ‘weak’ entity, because it has been created artificially to allow the many-to-many relationship to be removed. ORDER and ITEM are known as ‘strong’ entities, because they correspond to ‘real’ objects.
Source documents

Most databases are either set up from scratch or from existing manual systems. When setting up a database from scratch, normalisation helps to establish which tables are required.

Existing manual systems are usually paper-based, and often show evidence of data duplication, with all the possibility of data inconsistency that results. To produce a normalised data model, the data items required must be extracted from the existing paper documents. These are called source documents. Care is required, when producing a normalised data model from a paper-based system, in examining the source documents that are available.

To illustrate the normalisation process, let us reconsider the DVD rentals example.

Figures 3.15 and 3.16 show extracts from the member list and DVD rentals list.

Figure 3.15: Extract from the DVD member list

<table>
<thead>
<tr>
<th>Member number</th>
<th>Title</th>
<th>Forename</th>
<th>Surname</th>
<th>Telephone no.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1012</td>
<td>Miss</td>
<td>Isobel</td>
<td>Ringer</td>
<td>293847</td>
</tr>
<tr>
<td>1034</td>
<td>Mr</td>
<td>John</td>
<td>Silver</td>
<td>142536</td>
</tr>
<tr>
<td>1056</td>
<td>Mr</td>
<td>Fred</td>
<td>Flintstone</td>
<td>817263</td>
</tr>
<tr>
<td>1097</td>
<td>Mrs</td>
<td>Annette</td>
<td>Kirton</td>
<td>384756</td>
</tr>
</tbody>
</table>
To normalise the existing system, we must first identify the data items represented on the member list and DVD rentals list.

**Un-normalised form**

The first stage in the process of normalisation is to represent the data items required in UNF. This simply involves writing down all the attributes that must be stored in the database, and identifying a key.

In Figures 3.15 and 3.16, the attributes are split across two source documents. They are combined together into UNF, as shown in Figure 3.17.
Note

- There are two attributes called Title, but these refer to different data items. The first (from the member list) is a person’s title (i.e. Mr/Mrs/Miss/Ms), but the second is the title of the DVD (e.g. Finding Nemo). We need to distinguish between these by giving them separate names (e.g. Title; DVD Title).
- The attributes Member Number (in the member list) and Member (in the DVD rentals list) actually refer to the same data items, so should be listed only once. In this case, we will call the attribute Member Number.
- In the source documents, ‘Number’ was abbreviated to ‘No.’ in the Telephone No. attribute. We should ensure that we are consistent, so this should be written as Telephone Number, to be consistent with Member Number.

This gives us the following un-normalised form.

*Figure 3.18: Data model in UNF after resolving inconsistencies*

```
DVD_RENTAL
    (Member Number
     Title
     Forename
     Surname
     Telephone Number
     DVD Code
     DVD Title
     Cost
     Date Hired
     Date Due)
```

The normalisation process can then continue from this point.
Exercise 3
For each of the following examples, represent the data model in UNF, and normalise the data model, showing first, second and third normal form. Draw an entity-relationship diagram to represent the data model in 3NF.

1. A school administration system: each record contains information about a pupil and the subjects studied.

<table>
<thead>
<tr>
<th>Pupil Code</th>
<th>Name</th>
<th>Tutor Group</th>
<th>Year</th>
<th>Year Head</th>
<th>Set</th>
<th>Subject</th>
<th>Teacher</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>29865</td>
<td>Greg Dike</td>
<td>5PW</td>
<td>5</td>
<td>Mr Niven</td>
<td>5.2.2</td>
<td>English</td>
<td>Mr Dunn</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.1.3</td>
<td>Maths</td>
<td>Ms Napier</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.5.4</td>
<td>Physics</td>
<td>Mrs Newton</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.4.1</td>
<td>Info Systems</td>
<td>Mr Codd</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.3.1</td>
<td>Art &amp; Design</td>
<td>Miss Emin</td>
<td>39</td>
</tr>
<tr>
<td>73645</td>
<td>Susan Smith</td>
<td>5CV</td>
<td>5</td>
<td>Mr Niven</td>
<td>5.2.1</td>
<td>English</td>
<td>Mrs Bacon</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.1.4</td>
<td>Maths</td>
<td>Mr Fermat</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.3.3</td>
<td>Geography</td>
<td>Ms Cook</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.4.1</td>
<td>Info Systems</td>
<td>Mr Codd</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.5.2</td>
<td>Art &amp; Design</td>
<td>Mr Angelo</td>
<td>38</td>
</tr>
<tr>
<td>42315</td>
<td>Teresa Jewel</td>
<td>5PW</td>
<td>5</td>
<td>Mr Niven</td>
<td>5.2.2</td>
<td>English</td>
<td>Mr Dunn</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.1.4</td>
<td>Maths</td>
<td>Mr Fermat</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.5.4</td>
<td>Physics</td>
<td>Mrs Newton</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.3.2</td>
<td>French</td>
<td>Miss Kew</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.5.2</td>
<td>Art &amp; Design</td>
<td>Mr Angelo</td>
<td>38</td>
</tr>
</tbody>
</table>
2. A school library system: each record contains data about a pupil and the books they have borrowed.

<table>
<thead>
<tr>
<th>Pupil Code</th>
<th>Name</th>
<th>Address</th>
<th>Telephone Number</th>
<th>Tutor Group</th>
<th>Tutor</th>
<th>Book Number</th>
<th>Title</th>
<th>Return Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>29865</td>
<td>Greg Dike</td>
<td>31 Woodend Avenue</td>
<td>937482</td>
<td>5PW</td>
<td>P Wilkinson</td>
<td>8273</td>
<td>Rapid French</td>
<td>13/9/04</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2653</td>
<td>Physics is Hard</td>
<td>21/9/04</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4283</td>
<td>Higher Maths</td>
<td>15/9/04</td>
</tr>
<tr>
<td>73645</td>
<td>Susan Smith</td>
<td>17a Hill Street</td>
<td>472849</td>
<td>5CV</td>
<td>C Vallance</td>
<td>9574</td>
<td>Hamlet</td>
<td>28/9/04</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2834</td>
<td>Cataclysmic Chemistry</td>
<td>15/9/04</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4283</td>
<td>Higher Maths</td>
<td>22/9/04</td>
</tr>
<tr>
<td>42315</td>
<td>Teresa Jewel</td>
<td>2 Hawthorn Gardens</td>
<td>587293</td>
<td>5PW</td>
<td>P Wilkinson</td>
<td>3901</td>
<td>The Great Masters</td>
<td>9/10/04</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9574</td>
<td>Hamlet</td>
<td>14/9/04</td>
</tr>
</tbody>
</table>

3. A video rental store: each record contains data about a member and the films they have rented on DVD.

<table>
<thead>
<tr>
<th>Member Number</th>
<th>Name</th>
<th>Telephone Number</th>
<th>DVD Code</th>
<th>Film Code</th>
<th>Title</th>
<th>Cost</th>
<th>Date Hired</th>
<th>Date Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Greg Dike</td>
<td>192837</td>
<td>185</td>
<td>1423</td>
<td>The English Patient</td>
<td>£2.50</td>
<td>14/04/04</td>
<td>15/04/04</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>235</td>
<td>1524</td>
<td>Finding Nemo</td>
<td>£2.00</td>
<td>27/08/04</td>
<td>29/08/04</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>114</td>
<td>1426</td>
<td>Shrek</td>
<td>£2.00</td>
<td>05/09/04</td>
<td></td>
</tr>
<tr>
<td>1524</td>
<td>Susan Smith</td>
<td>283746</td>
<td>157</td>
<td>1287</td>
<td>An American in Paris</td>
<td>£2.50</td>
<td>27/08/04</td>
<td>28/08/04</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>206</td>
<td>1423</td>
<td>The English Patient</td>
<td>£1.50</td>
<td>03/09/04</td>
<td>04/09/04</td>
</tr>
<tr>
<td>1764</td>
<td>Teresa Jewel</td>
<td>827376</td>
<td>313</td>
<td>1524</td>
<td>Finding Nemo</td>
<td>£2.00</td>
<td>27/08/04</td>
<td>29/08/04</td>
</tr>
</tbody>
</table>
4. Hogwarts School of Witchcraft and Wizardry: each record contains data about a student and the classes they take.

<table>
<thead>
<tr>
<th>Student Code</th>
<th>Student</th>
<th>House</th>
<th>Head of House</th>
<th>Class</th>
<th>Class Code</th>
<th>Teacher</th>
</tr>
</thead>
<tbody>
<tr>
<td>G468</td>
<td>H Granger</td>
<td>Gryffindor</td>
<td>Prof McGonagall</td>
<td>Care of Magical Creatures</td>
<td>RH</td>
<td>R Hagrid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Charms</td>
<td>FF</td>
<td>F Flitwick</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Transfiguration</td>
<td>MM</td>
<td>M McGonagall</td>
</tr>
<tr>
<td>S616</td>
<td>D Malfoy</td>
<td>Slytherin</td>
<td>Prof Snape</td>
<td>Defence Against the Dark Arts</td>
<td>GL</td>
<td>G Lockhart</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Divination</td>
<td>ST</td>
<td>S Trelawney</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Potions</td>
<td>SS</td>
<td>S Snape</td>
</tr>
<tr>
<td>G476</td>
<td>H Potter</td>
<td>Gryffindor</td>
<td>Prof McGonagall</td>
<td>Care of Magical Creatures</td>
<td>RH</td>
<td>R Hagrid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Defence Against the Dark Arts</td>
<td>GL</td>
<td>G Lockhart</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Potions</td>
<td>SS</td>
<td>S Snape</td>
</tr>
</tbody>
</table>

5. A movie database: each record contains data about a film and its stars.

<table>
<thead>
<tr>
<th>Film Code</th>
<th>Title</th>
<th>Year Released</th>
<th>Director</th>
<th>Director ID</th>
<th>Actor</th>
<th>Actor ID</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>90PW1</td>
<td>Pretty Woman</td>
<td>1990</td>
<td>Gary Marshall</td>
<td>GM07</td>
<td>Julia Roberts</td>
<td>JR07</td>
<td>Vivian Ward</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Richard Gere</td>
<td>RG12</td>
<td>Edward Lewis</td>
</tr>
<tr>
<td>99NH1</td>
<td>Notting Hill</td>
<td>1999</td>
<td>Roger Mitchell</td>
<td>RM11</td>
<td>Julia Roberts</td>
<td>JR07</td>
<td>Anna Scott</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Hugh Grant</td>
<td>HG25</td>
<td>William Thacker</td>
</tr>
<tr>
<td>94FW1</td>
<td>Four Weddings and a Funeral</td>
<td>1994</td>
<td>Mike Newell</td>
<td>MN13</td>
<td>Hugh Grant</td>
<td>HG25</td>
<td>Charles</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Andie MacDowell</td>
<td>AM32</td>
<td>Carrie</td>
</tr>
<tr>
<td>90GC1</td>
<td>Green Card</td>
<td>1990</td>
<td>Peter Weir</td>
<td>PW03</td>
<td>Gérard Depardieu</td>
<td>GD19</td>
<td>Georges Fauré</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Andie MacDowell</td>
<td>AM32</td>
<td>Broné Mitchell</td>
</tr>
<tr>
<td>05HP1</td>
<td>Harry Potter and the Goblet of Fire</td>
<td>2005</td>
<td>Mike Newell</td>
<td>MN13</td>
<td>Daniel Radcliffe</td>
<td>DR38</td>
<td>Harry Potter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Alan Rickman</td>
<td>AR26</td>
<td>Severus Snape</td>
</tr>
</tbody>
</table>
6. Kinlochforest and District School Sports are held each year in nine primary schools. Each school has its own sports day, and its own local dignitary invited to present the certificates. Shown below is an example of a certificate issued to an entrant, alongside a list of some of the schools and sports day details.

**Sports Certificate**

**School Sports Day details**

(a) Represent the data in these source documents in UNF.

(b) You may assume that there are no two competitors with the same name. Select a suitable key, and normalise the data model to 3NF. Draw an E-R diagram to show the relationships between the entities in 3NF.
7. Normalise the data in Question 2 to 3NF, starting with Book Number as the key. Think carefully about the repeating data items at UNF. (Hint: for each book number, which attributes can have more than one value?)

8. Normalise the data in the mail order invoice (see page 61) to 3NF, starting with Customer Number as the key. Again, think carefully about the repeating data items at UNF.

What goes wrong this time? See Appendix A for a detailed explanation and solution.
Chapter 4: Database design

Once a data model has been completed, it is necessary to design the database. This involves deciding on the properties of each field.

Properties of fields

For each field in the database, you must consider the following.

- **The name**
  
  You should take care to choose sensible field names and make sure that your naming is consistent in each table. For example, if you choose to abbreviate Member Number to Member No. (rather than Member Num. or Member #), you should also abbreviate Telephone Number to Telephone No.

- **The data type**
  
  This may be one of the following.

  - **Text, string or alphanumeric**
    
    e.g. Smith, EH991AB, £13+VAT.

  - **Numeric**
    
    Either integer (whole numbers) or real (floating point), e.g. 13, 3.14.

  - **Currency**
    
    A special type of numeric field for monetary values, e.g. 12.00, 2.50, 0.15.

  - **Date or time**
    
    Dates may be in dd/mm/yyyy format or ‘long date’ format. Times may be in hh:mm:ss format or ‘long time’ format e.g. 01/01/1990, 1 January 1990, 13:30:00, 1:30 p.m.

  - **Boolean**
    
    Yes or no. Named after George Boole, mathematician and logician, who created binary logic, e.g. AND, OR, NOT, YES/NO.

  - **Link**
    
    A reference to a file located outside the database.

  - **Object**
    
    Data such as a picture or sound file.
• The key
  Whether the field is a primary key (PK), part of a compound key, or a foreign key (FK).

• Validation
  Whether the field must have a value, or can be left blank (called a presence check).
  Whether the value of the field is limited to certain values (called a restricted choice check), e.g. title = Mr/Mrs/Miss/Ms.
  Numeric fields may be subject to a range check, e.g. the year group for a secondary school pupil must be between 1 and 6.

• Default value
  It is also useful to consider whether a value should have a default value. For example, a transaction date might have a default value of today’s date (generated automatically by the computer). Some DBMSs provide an ‘autonumber’ field where the value is generated automatically by the computer: this is usually used to provide unique values for surrogate keys.

• Formatting
  Some fields may require input to follow a certain pattern (called an ‘input mask’), e.g. a postcode must contain letters and numbers in a particular configuration.
  Some fields may need to be displayed in a particular way, e.g. a date may be displayed as 6/9/04 or as September 6, 2004.

The structure of tables and the properties of fields are usually represented in the form of a data dictionary.
Storing names, addresses and telephone numbers

When storing information about people in databases, the title, forename and surname are usually stored as separate fields.

Title: Mrs
Forename: Joanna
Surname: Bloggs

This allows the data to be sorted more flexibly, e.g. in alphabetical order of surname. It also allows the data to be used in different combinations (e.g. a letter may begin ‘Dear Mrs Bloggs’).

Addresses are usually stored with three address lines (perhaps including a postal town and region) plus a line for postal code.

Address 1 12 High Street
Address 2 Auchenshoogle
Address 3 Inverness-shire
Post Code IV99 2QW

Telephone numbers are always stored as text fields rather than number fields. This is because the first digit is 0, which is not normally stored in a numeric field, and also allows applications to dial a customer’s telephone number directly from the database. Currently, all domestic UK telephone numbers (including mobile phone numbers) consist of 11 digits.

The same applies to some other ‘numbers’, such as ISBNs (International Standard Book Numbers) and Vehicle Registration Numbers.

More about data types

Figure 4.1 shows the range of data types available for use in a table definition in Microsoft Access, while Figure 4.2 shows how data would be entered. Note how a field with a Boolean data type (e.g. Vegetarian) is shown as a check box. A field with an Object data type (e.g. Photo) is shown as text, the value of which is determined by the type of object file inserted (see Figure 4.3).
Figure 4.1: A table definition in Microsoft Access showing the use of data types: default value (autonumber), text, date/time, real numeric, Boolean, object, link.

![Figure 4.1](image1)

Figure 4.2: A record in Microsoft Access showing data entry for the data types specified in Figure 4.1

![Figure 4.2](image2)

Figure 4.3: Inserting an object data type in Microsoft Access

![Figure 4.3](image3)
Figure 4.4: A table definition in Filemaker Pro showing the use of data types: default value (autonumber), text, date/time, real numeric, Boolean, object, link (called a 'container' in Filemaker Pro).

Figure 4.5: An autonumber field in Filemaker Pro is achieved by creating a number field and setting the auto-enter options as shown.
Figure 4.6: Inserting an object into a field with a container data type in Filemaker Pro

Data dictionary

A data dictionary is simply a table that lists the fields of each table in a data model, together with the properties of each field. The data dictionary is important because this information can then be used to create a database for the data model using any chosen RDBMS.

On the following page, there is a sample data dictionary for the DVD rentals example.
<table>
<thead>
<tr>
<th>Entity</th>
<th>Attribute</th>
<th>Key</th>
<th>Data Type</th>
<th>Data Type</th>
<th>Validation</th>
<th>Format</th>
<th>Unique</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>DVD</td>
<td>DVD Code</td>
<td>PK</td>
<td>Integer</td>
<td>Integer</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>Film Code</td>
<td>FK</td>
<td>Integer</td>
<td>Integer</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MEMBER</td>
<td>Member Number</td>
<td>PK</td>
<td>Integer</td>
<td>Integer</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>Title</td>
<td>PK</td>
<td>Text (15)</td>
<td>Text (20)</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Forename</td>
<td>PK</td>
<td>Text (20)</td>
<td>Text (20)</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Surname</td>
<td>PK</td>
<td>Text (20)</td>
<td>Text (20)</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Address 1</td>
<td>PK</td>
<td>Text (20)</td>
<td>Text (20)</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Address 2</td>
<td>PK</td>
<td>Text (20)</td>
<td>Text (20)</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Address 3</td>
<td>PK</td>
<td>Text (20)</td>
<td>Text (20)</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Post Code</td>
<td>PK</td>
<td>Text (8)</td>
<td>Text (8)</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>FILM</td>
<td>Film Code</td>
<td>PK</td>
<td>Integer</td>
<td>Integer</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>Title</td>
<td>PK</td>
<td>Text (30)</td>
<td>Text (20)</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>LOAN</td>
<td>Member Number</td>
<td>PK</td>
<td>Integer</td>
<td>Integer</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Loan Code</td>
<td>PK</td>
<td>Integer</td>
<td>Integer</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Date Hired</td>
<td>PK</td>
<td>Date</td>
<td>Date</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>Date Due</td>
<td>PK</td>
<td>Date</td>
<td>Date</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

**Relational Database Systems (Higher)**

© Learning and Teaching Scotland
Notes:

• Primary keys and foreign keys are indicated by PK and FK.
• Foreign keys must have the same data type as the corresponding primary key.
• A compound key (e.g. for the LOAN entity) is shown by PK against each of the fields in the key.
• The fields in a compound key are not individually unique. There may be more than one loan record for Member Number 1034; more than one loan record for DVD Code 011; and more than one loan record for 06/09/04. It is the combination that is unique—so there can only be one record for Member Number 1034, with DVD Code 011 on 06/09/04.
• The length of text fields is shown by the number of characters in brackets.
• In the postcode and telephone number formats, A = alphabetic character (required); ? = alphabetic character (optional); 0 = digit (required); 9 = digit (optional). Some DBMSs may not support formatting of text fields, or may use different symbols.
• Some DBMSs may not support different date formats. The format dd/mm/yy indicates a 'short date' format, e.g. 06/09/04. There are other valid date formats.
• Some DBMSs allow data to be stored and displayed independently. For example, all internal UK telephone numbers consist of 11 digits, generally a 4 or 5 digit area code followed by a 6 or 7 digit number, e.g. (01234) 567890 or (0131) 987 6543. It is not always necessary to store brackets or spaces, as these are only required to make the number easier to read. (For 'direct dial' uses, where a computer is used to dial the number, only the digits are required.) So, it may be possible to store only the digits, yet specify a display and entry format that includes brackets and/or spaces.
• All postcodes consist of a maximum seven characters. Although the most common postcodes are of the form IV99 9ZZ, there are exceptions. See Appendix B for more details. As with telephone numbers, it may be possible to store only the characters of the postcode yet specify a format that allows for a space to be inserted.

Exercise 4

1. Give two reasons why the format of a telephone number makes a numeric data type unsuitable for its storage.

2. Create a data dictionary for the 3NF solution to Question 3 in Exercise 3.
Chapter 5: Advanced implementation in Microsoft Access

Having analysed a data model to third normal form, and designed a data dictionary, you are now ready to implement your database.

You can think of a database solution as being like a tower. The tables form the foundations, which is why the data dictionary is so important in helping to get the design of the tables right.

The next level of the tower is queries, which are used to perform searching, sorting and calculations.

Next, user views of the data in the database are used for input and output of data to and from the tables, e.g. to display the results of queries.

Navigation features are added to the user interface to connect the forms, reports and layouts together so that the user can easily perform various tasks, using the database.

Tables and referential integrity

In a relational database, tables are linked through the use of foreign keys. A foreign key is an attribute that is not a key in its own table, but is a primary key in another table. A foreign key links a record in one table with a related record in another table.

For the database to work properly, it is essential to guarantee that a foreign key always refers to a record that exists in the other table. This is called referential integrity.

For example, an entry in the LOAN table can only refer to an existing record in the MEMBER table, because a loan can only be made by a member.
A further aspect of referential integrity is that if a record is deleted from one table, then it may be necessary to delete the references to that record in other tables. So, if a member leaves the club, then all loans made by that member can also be deleted.

In Microsoft Access, referential integrity is established by setting the relationships between the tables in the Relationships window, as shown in Figures 5.1, 5.2 and 5.3.

*Figure 5.1: Relationships before setting referential integrity*

![Figure 5.1: Relationships before setting referential integrity](image1)

*Figure 5.2: Setting referential integrity in Microsoft Access*

![Figure 5.2: Setting referential integrity in Microsoft Access](image2)

*Figure 5.3: Results of setting referential integrity in Microsoft Access*

![Figure 5.3: Results of setting referential integrity in Microsoft Access](image3)
Achieving referential integrity

As well as checking for referential integrity, many RDBMSs provide a way of ensuring that the user cannot mistakenly enter an invalid value for a foreign key.

In Microsoft Access, **lookups** are used to ensure referential integrity, as shown in Figures 5.4 and 5.5. In the LOAN table, the user can only enter a value for Member Number which already exists in the MEMBER table.

*Figure 5.4: Using the lookup wizard in Microsoft Access*

![Figure 5.4: Using the lookup wizard in Microsoft Access](image1)

*Figure 5.5: Selecting the value for a foreign key by using a lookup list in Microsoft Access*

![Figure 5.5: Selecting the value for a foreign key by using a lookup list in Microsoft Access](image2)
More about validation

This section shows how to implement a number of standard validation checks using Microsoft Access.

**Presence check**
*Figure 5.6: Achieving a presence check by setting the ‘Required’ property to ‘Yes’. Key fields also have the ‘Indexed’ property set to ‘No Duplicates’*

![Image of Presence Check](image1)

**Range check**
*Figure 5.7: Achieving a range check by setting the ‘Validation Rule’ property*

![Image of Range Check](image2)
Restricted choice check
Figure 5.8: Achieving a restricted choice check using a lookup

Default values
Figure 5.9: A default value setting for a currency field
Figure 5.10: A default value setting of the current date for a date field

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order Date</td>
<td>Date/Time</td>
</tr>
</tbody>
</table>

![Default Value Setting](image1)

Formatting

Formatting settings affect how data is displayed. The most common formatting options affect numeric, date and time fields. The following screenshots show how formatting options are set for these data types in Microsoft Access.

Figure 5.11: Date and time formats in Microsoft Access

![Date and Time Formats](image2)

Figure 5.12: Numeric formats in Microsoft Access

![Numeric Formats](image3)
Formatting options can also be applied to some other data types, such as graphics, as the following screenshot shows.

*Figure 5.13: Formatting options for displaying a graphic in an object frame*

![Input Mask Wizard](image)

**Input masks**

An input mask is similar to a format setting, but affects the way that data is input to a field, rather than how a field value is displayed. Figure 5.14 shows the input mask wizard in Microsoft Access, which can be used to specify the format of data entered into a field.

*Figure 5.14: The input mask wizard in Microsoft Access*
In the postcode input mask >LL09 0LL the symbols have the following meaning.

0  Digit (0 through 9, entry required; plus [+] and minus [–] signs not allowed).
9  Digit or space (entry not required; plus and minus signs not allowed).
L  Letter (A through Z, entry required).
>  Causes all characters that follow to be converted to uppercase.

There are other characters that can be used in an input mask, as shown below.

**Description**

#  Digit or space (entry not required; blank positions converted to spaces, plus and minus signs allowed).
A  Letter or digit (entry required).
a  Letter or digit (entry optional).
&  Any character or a space (entry required).
C  Any character or a space (entry optional).
,,:;/-  Decimal placeholder and thousands, date, and time separators.
<  Causes all characters that follow to be converted to lowercase.

**Queries**

The term *query* is used to describe any process that performs searching or sorting operations in a relational database. In Access, queries are also used to perform calculations.

**Searching**

Searching is the process of selecting records from a table or combination of tables. To perform the query, three items must be identified.

1. Which fields will be used to identify the records required?
2. What are the criteria for identifying the records required?
3. Which fields will be displayed?
**Search 1: Single condition**
For example, to identify the names and telephone numbers of club members who have rented the DVD Shrek:

1. the fields required are Forename, Surname and Telephone Number (from the MEMBER table) and Film Title (from the FILM table)
2. the criteria are Film Title = Shrek
3. the fields to be displayed are Forename, Surname and Telephone Number.

Figure 5.15 shows how this query would be set up.

*Figure 5.15: Using a single search condition. Note that all four entities are required to link MEMBER with FILM.*

A **complex search** involves more than one search condition (and usually more than one field).
**Search 2: Multiple conditions (AND)**
Consider a request to list those members who have rented the DVD Shrek since 16 August 2004.

1. The fields required are Forename, Surname and Telephone Number (from the MEMBER table), Film Title (from the FILM table), and Date Hired (from the LOAN table).
2. The criteria are Film Title = Shrek AND Date Hired > 16/08/2004.
3. The fields to be displayed are Forename, Surname and Telephone Number.

Figure 5.16 shows how this query would be set up.

*Figure 5.16: Using multiple search conditions involving two fields*
**Search 3: Multiple conditions (OR)**

Next, consider a request to list those members who have rented Shrek or Finding Nemo.

1. The fields required are Forename, Surname and Telephone Number (from the MEMBER table) and Film Title (from the FILM table).
2. The criteria are Film Title = Shrek OR Film Title = Finding Nemo.
3. The fields to be displayed are Forename, Surname and Telephone Number.

Figure 5.17 shows how this query would be set up.

*Figure 5.17: Using multiple search conditions involving one field.*

Note that it is possible that a member may have rented both Shrek *and* Finding Nemo. In this case, the member’s name and telephone number would be listed twice in the search results. These duplicate results can be eliminated. In Microsoft Access, the ‘Group By’ option is used to achieve this, as shown in Figure 5.18.
Search 4: Combining conditions

Finally, consider a request for a list of members who have rented either Shrek or Finding Nemo since 16 August 2004.

1. The fields required are as for Search 2: Forename, Surname and Telephone Number (from the MEMBER table), Film Title (from the FILM table), and Date Hired (from the LOAN table).

2. The criteria are more complicated this time:

   Film Title = Shrek AND Date Hired > 16/08/2004
   OR
   Film Title = Finding Nemo AND Date Hired > 16/08/2004.

   Note that the Date Hired condition must be repeated for each film.

3. The fields to be displayed are Forename, Surname and Telephone Number.
Figure 5.19 shows how this query would be set up. Great care must be taken when setting up queries involving combinations of search conditions, to ensure that the correct records are returned.

**Figure 5.19: Using multiple search conditions involving multiple fields**

Sorting

Usually, the results of a selection query are presented in some kind of order. A sorting operation is performed to achieve this. To perform a sort, two items must be identified.

1. Which field (or fields) will be used to decide the order of records?
2. For each field selected, will the order of sorting be *ascending* or *descending*?

For example:

- to produce a list of people with the tallest first, the records would be sorted in *descending* order of *height*
- to produce a list of people with youngest first, the records would be sorted in *ascending* order of *age*.

A very common way of ordering records relating to people is in alphabetical order. To achieve alphabetical ordering requires the records to be sorted in *ascending* order of *surname*.
A complex sort involves more than one sort condition, involving two or more fields.

For example, as we have seen in Chapter 4, a person’s name is usually stored in three fields: Title (e.g. Mr/Mrs/Miss/Ms), Forename and Surname. To achieve ‘telephone book order’, the name is sorted in ascending order of surname, then ascending order of forename. In this case, the Surname field is the primary sort key, and the Forename field is the secondary sort key. Figure 5.20 shows how such a complex sort is set up in a query.

Figure 5.20: A complex sort. Note that the fields used (Surname and Forename) must be selected in the correct order, with the primary sort key to the left of the secondary sort key.
Calculations

One of the main advantages that computers have over humans is the ability to perform calculations very quickly, as many times as necessary, without errors.

In a relational database, calculations are performed using expressions or formulas. Here is an expression to calculate a student’s total mark from three tests:

$$= [\text{Test 1}] + [\text{Test 2}] + [\text{Test 3}]$$

Some calculations are performed for each record, using other values in the record. This type of calculation is a **horizontal** calculation.

Other calculations are **vertical** calculations, which are based on the value of a single field taken from a set of records. The average mark for Test 1 would be an example, where the expression required would be similar to $$=\text{Average}([\text{Test 1}])$$.

<table>
<thead>
<tr>
<th>Name</th>
<th>Test 1</th>
<th>Test 2</th>
<th>Test 3</th>
<th>Total Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>J Bloggs</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>27</td>
</tr>
<tr>
<td>J Public</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>21</td>
</tr>
</tbody>
</table>

Horizontal calculations are often known as **calculated fields**, and vertical calculations are known as **summary fields**.

All calculated values are examples of derived data, which are not normally stored in a relational database. In Access, the derived data is calculated when required (using an expression in a query or a text box in a form or report).
Most RDBMSs provide a range of functions that can be used in expressions. The table below lists some of the main categories of these functions, with some examples of common functions for each category.

<table>
<thead>
<tr>
<th>Category</th>
<th>Example of functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate</td>
<td>Sum, Avg, Max, Min, Count</td>
</tr>
<tr>
<td>Mathematical</td>
<td>Sin, Cos, Tan, Int, Round</td>
</tr>
<tr>
<td>Text</td>
<td>Left, Right, Mid, Len, UCase, LCase, InStr, Replace</td>
</tr>
<tr>
<td>Logical</td>
<td>IIf, IsNull, IsNumeric, IsError</td>
</tr>
<tr>
<td>Conversion</td>
<td>Str, Val, Date$, Text-to-Date</td>
</tr>
<tr>
<td>Date</td>
<td>Date, Day, Month, Year, Weekday, WeekdayName, DateDiff</td>
</tr>
<tr>
<td>Time</td>
<td>Time, Hour, Minute, Second</td>
</tr>
</tbody>
</table>

**Working with dates**

In many databases where dates are recorded (e.g. date of birth, date hired, date returned, etc.) it is necessary to be able to perform calculations involving dates.

For example:

- to search for all DVDs rented in the last week, you would use the expression
  
  `Date() - [Date Hired] <= 7`

- to calculate a person's age in years from their date of birth, you would use the expression
  
  `DateDiff("yyyy",[Date of Birth],Date())`
Parameterised queries

Many database management systems allow queries to be saved. This allows the queries to be used in the future without having to set them up each time they are required.

However, very often a query is required to be used many times, but with varying criteria. For example, consider a query set-up to list all DVD rentals for July. To list the DVD rentals for August would either require a new query to be set-up, or the existing query to be changed.

An alternative that is available in some RDBMSs is to use a parameterised query. This is a query where the user enters a value to be searched for. Figure 5.21 shows the design and implementation of a parameterised query.

Figure 5.21: A parameterised query in Microsoft Access. The first screenshot shows the query design, with the text prompt replacing the criteria; the second screenshot shows the data entry box.
User views
One of the advantages that a DBMS has over a spreadsheet is its ability to present information in a variety of different ways without changing the underlying structure or content of the data stored. These are called user views, and are created using forms and reports.

A form or report is usually based on a query, which selects the required fields from the appropriate tables, sorting the results if necessary, and performing any horizontal calculations.

Report structure
A database report is made up of a number of sections, as shown in Figure 5.22.

*Figure 5.22: Structure of a database report*

<table>
<thead>
<tr>
<th>Report Header</th>
<th>Report Footer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text/data/data to appear at the top of the report</td>
<td>Text/data to appear at the end of the report</td>
</tr>
<tr>
<td><strong>Page Header</strong></td>
<td><strong>Main Detail Footer</strong></td>
</tr>
<tr>
<td>Text/data to appear at the top of each page of the report</td>
<td>Text/data to appear below each entry in the main detail section</td>
</tr>
<tr>
<td><strong>Main Detail Header</strong></td>
<td><strong>Main Detail Section</strong></td>
</tr>
<tr>
<td>Text/data to appear above each entry in the main detail section</td>
<td>Data from selected records in a table or query</td>
</tr>
<tr>
<td><strong>Main Detail Footer</strong></td>
<td><strong>Page Footer</strong></td>
</tr>
<tr>
<td><strong>Report Footer</strong></td>
<td></td>
</tr>
<tr>
<td>Text/data to appear at the foot of each page of the report</td>
<td>Text/data to appear at the top of the report</td>
</tr>
</tbody>
</table>

**DVD Rental Statistics**

**Details for J Bloggs**

<table>
<thead>
<tr>
<th>Memb No</th>
<th>Address</th>
<th>Tel No</th>
</tr>
</thead>
<tbody>
<tr>
<td>142312</td>
<td>Main Street</td>
<td>123456</td>
</tr>
</tbody>
</table>

Total DVDs rented to date: 26

**End of page 1**

Total DVDs rented by all members: 3,218

© Learning and Teaching Scotland
Summary information

One of the aspects that distinguishes a relational database from a spreadsheet is the ability to summarise information.

The five most common summary calculations that are performed are as follow:

- **Sum** to add values to give a total, e.g. total cost of DVD rentals last month.
- **Average** to find an average value, e.g. average cost of hire per DVD.
- **Count** to count the number of records found, e.g. number of DVDs rented per member.
- **Maximum** to find the highest value, e.g. highest number of rentals per DVD (to find the most popular DVD).
- **Minimum** to find the lowest value, e.g. lowest number of rentals per member.

Summary information is produced by creating a **summary field**. A summary field is a calculated field with a formula to perform the calculation, and is placed in a **summary section** of a report.

For example, to calculate the total cost of DVD rentals for a member, a summary field would be created containing the formula =Sum([Cost]) in a summary section, as shown in Figure 5.23. Used in this way, the summary field will calculate the total cost of rentals for each member.

*Figure 5.23: A report with summary field in summary section (Member Number Footer)*
However, the advantage of a database is that the report can be used with the results of any query, so that we could use it to find the total cost of rentals for members who have rented DVDs in the last week only.

In addition, by placing the same summary field in a different summary section, we can get the database to calculate a different result.

For example, Figure 5.24 shows summary fields in the Member Number Footer, Page Footer and Report Footer sections of the report. The fields in Member Number Footer will show the total cost and number of rentals for each member, Page Footer will show the total cost and number of rentals displayed on that page, while Report Footer will show the total cost and number of all rentals displayed in the report.

*Figure 5.24: A report with summary fields in the Page Footer and Report Footer sections.*

Sub-forms and sub-reports

Because relational databases are designed to represent data that is related, very often a single table of results is not adequate for presenting the data. Instead, what is required is a sub-table. In Access, sub-forms and sub-reports are used to achieve this.

Figure 5.25 shows how the Form Wizard recognises the relationships between tables, and Figure 5.26 shows the form with sub-form that is created as a result.
Figure 5.25: The Form Wizard automatically recognises relationships between tables

Figure 5.26: A form with sub-form created by the Form Wizard
Navigation

Relational databases form the basis of many complex applications. For example, e-commerce websites such as Amazon are based on relational databases, which store information about customers, products, and orders. However, what makes these websites so successful is their ease of use.

Figure 5.27: Navigation bar and search box from Amazon website

In Figure 5.27, when you click on the DVD tab on the navigation bar, the browser loads up the DVD web page, which performs a search for information on DVDs from the relational database on the Amazon web server.

If you enter a word or phrase into the keyword search box, a web page will be displayed showing the results of a search on the relational database, with the results sorted in order. Figure 5.28 shows the results of a search for ‘ring’.
When you add items to your shopping basket, the current total of the items in the basket is calculated and displayed on the web page (Figure 5.29).

**Figure 5.29: Shopping basket display from Amazon, showing sub-total.**
All these searching, sorting and calculating operations are performed by the relational database without the user being aware of using it.

We can achieve something similar with desktop database packages such as Microsoft Access, using navigation features available with the software. For example, Access provides a number of buttons that allow the user to find records, add new records, delete records, navigate between records, open forms, print reports, etc. Figures 5.30 and 5.31 show how a customised user interface to a database form can improve its user-friendliness.

*Figure 5.30: A standard form with sub-form based on a query*
**Figure 5.31:** A customised form with sub-form. Note the use of customised buttons

![Customised Form with Sub-Form](image)

### Macros and scripting

In Figure 5.31, the buttons that provide navigation and other user interface features use **scripting** to carry out their tasks (see Figure 5.32).

**Figure 5.32:** The Command Button Wizard shows the range of pre-defined operations that can be carried out

![Command Button Wizard](image)
Figure 5.33 shows the **script** that is processed when the ‘Find Member’ button is pressed. In Access, scripts are written using Visual Basic for Applications (VBA). With some knowledge of VBA, the database designer can edit or create their own scripts to perform other actions as required.

**Figure 5.33: VBA script for the ‘Find Member’ button**

An alternative to scripting is to use **macros**. A **macro** is a sequence of one or more actions that each perform a particular operation within the DBMS. Macros are particularly useful for automating common tasks. Figure 5.34 shows how a simple macro would be created to open a form and navigate to the last record selected from a query’s results.

**Figure 5.34: Creating a macro in Microsoft Access**
SQL and Microsoft Access

RDMSs are based on a data definition language called Structure Query Language, or SQL (often pronounced ‘sequel’). In fact, some RDMSs take their name from SQL – e.g. SQL Server and MySQL. SQL is an example of a data definition language (DDL) – that is, it can be used to create tables and define fields. SQL is also a data manipulation language (DML) which can be used to perform searching and sorting, calculations and summary functions.

Microsoft Access provides a graphical user interface (GUI) for using SQL, so that the user doesn’t have to be concerned with entering commands and getting the syntax of the instructions correct.

However, if you wish to, Access allows you to see the SQL statements created when you create a query (see Figures 5.35, 5.36, and 5.37). Some queries in Access can only be performed using SQL, e.g. union and pass-through queries.

**Figure 5.35: Query design window in Microsoft Access**

**Figure 5.36: Switching to SQL View in Microsoft Access**
Figure 5.37: SQL View of a query in Microsoft Access

Exercise 5

1. Figure 5.28 shows some search results for the keyword ‘ring’ from the Amazon website.

   The results of the search in each category indicate the total number of results found, e.g. ‘See all 89 results in DVD…’

   Explain how the total number of results found can be calculated by the relational database.

2. Figure 5.29 shows a customer’s shopping basket.

   Explain how the sub-total of £47.55 can be calculated by the relational database.
Chapter 6: Advanced implementation in FileMaker Pro

Having analysed a data model to third normal form, and designed a data dictionary, you are now ready to implement your database.

You can think of a database solution as being like a tower. The tables form the foundations, which is why the data dictionary is so important in helping to get the design of the tables right.

The next level of the tower is queries, which are used to perform searching, sorting and calculations.

Next, user views of the data in the database are used for input and output of data to and from the tables, e.g. to display the results of queries.

Navigation features are added to the user interface to connect the forms, reports and layouts together so that the user can easily perform various tasks, using the database.

Tables and referential integrity

In a relational database, tables are linked through the use of foreign keys. A foreign key is an attribute that is not a key in its own table, but is a primary key in another table. A foreign key links a record in one table with a related record in another table.

For the database to work properly, it is essential to guarantee that a foreign key always refers to a record that exists in the other table. This is called referential integrity.

For example, an entry in the LOAN table can only refer to an existing record in the MEMBER table, because a loan can only be made by a member.
A further aspect of referential integrity is that if a record is deleted from one table, then it may be necessary to delete the references to that record in other tables. So, if a member leaves the club, then all loans made by that member can also be deleted.

In Filemaker Pro, referential integrity is established by defining relationships between the files, as shown in Figures 6.1 and 6.2.

**Figure 6.1: Setting referential integrity in Filemaker Pro**

![Figure 6.1: Setting referential integrity in Filemaker Pro](image1)

**Figure 6.2: Results of setting referential integrity in Filemaker Pro**

![Figure 6.2: Results of setting referential integrity in Filemaker Pro](image2)
Achieving referential integrity

As well as checking for referential integrity, many RDBMSs provide a way of ensuring that the user cannot mistakenly enter an invalid value for a foreign key.

In Filemaker Pro, value lists are used to ensure referential integrity, as shown in Figures 6.3, 6.4 and 6.5. In the LOAN table, the user can only enter a value for Member Number which already exists in the MEMBER table.

Figure 6.3: Selecting the value for a foreign key by defining a value list in Filemaker Pro

Figure 6.4: Specifying the values for the foreign key, Member No. in Filemaker Pro
More about validation

This section shows how to implement a number of standard validation checks using Filemaker Pro.

**Presence check**

*Figure 6.6: Achieving a presence check in Filemaker Pro, by checking the ‘Not empty’ box). Key values can also be established by checking the ‘Unique’ box.*

[Diagram showing Filemaker Pro settings for validation]
**Range check**

Figure 6.7: Achieving a range check in Filemaker Pro, by setting the ‘In range’ property

**Restricted choice check**

Figure 6.8: Achieving a restricted choice check using a value list in Filemaker Pro
Figure 6.9: Achieving a Boolean field. A text field is created with validation set to a value list of two options

Default values
Figure 6.10: A default value setting for a number field in Filemaker Pro
Figure 6.11: A default value setting of the current date for a date field in Filemaker Pro

Formatting

Formatting settings affect how data is displayed. The most common formatting options affect numeric, date and time fields. The following screenshots show how formatting options are set for these data types in Filemaker Pro.

Figure 6.12: Date formats in Filemaker Pro
Figure 6.13: Time formats in Filemaker Pro

![Time Format for "Start Time" dialog]

Figure 6.14: Numeric formats in Filemaker Pro

![Number Format for "Cost" dialog]
Formatting options can also be applied to some other data types, such as graphics, as the following screenshots show.

*Figure 6.15: Graphic formatting in Filemaker Pro*

**Queries**

The term *query* is used to describe any process that performs searching or sorting operations in a relational database. In Filemaker Pro, ‘Find’ mode is used to perform these, and scripts can be written to carry these out.

**Searching**

Searching is the process of selecting records from a table or combination of tables. To perform the query, three items must be identified.

1. Which fields will be used to identify the records required?
2. Which tables are required to produce the data required?
3. What are the criteria for identifying the records required?
**Search 1: Single condition**

For example, to identify the names and telephone numbers of club members with Member Number over 1000:

1. the fields required are Forename, Surname and Telephone Number
2. the MEMBER table is required, as it contains the fields listed
3. the criteria are Member Number > 1000.

*Figure 6.16: Search 1 performed using a ‘Find’ request*

A **complex search** involves more than one search condition (involving one or more fields).
Search 2: Multiple conditions (AND)
To identify the names and telephone numbers of club members with Surname beginning with M and Member Number over 1000:

1. as before, the fields required are Forename, Surname and Telephone Number
2. as before, the MEMBER table is required.
3. the criteria are Surname = M* AND Member Number > 1000.

Figure 6.17: Search 2 – involving multiple conditions (AND)

Search 3: Multiple conditions (OR)
To identify the names and telephone numbers of club members with Surname beginning with M or N:

1. as before, the fields required are Forename, Surname and Telephone Number
2. as before, the MEMBER table is required
3. the criteria are Surname = M* OR Surname = N*.

Figure 6.18 shows the Find request to perform the search. This time a new request is used to input the second condition.

Figure 6.18: Search 3 – involving multiple conditions (OR)
Search 4: More than one table
To identify the names of club members who rented a DVD in August 2005, together with the rental dates:

1. the fields required are Forename, Surname and Date Hired
2. the tables required are MEMBER and LOAN
3. the criteria are Date Hired = */8/2005.

In order to extract related data from two different tables, you must first determine how the tables are linked. In this example, the tables are linked directly, as Member Number is a foreign key in the LOAN table. In general, the search should be performed on the ‘many’ side of the relationship (i.e. in the LOAN table).

Firstly, a relationship must be set up in the LOAN entity to link LOAN to MEMBER (see Figure 6.19).

Figure 6.19: Setting up a relationship between tables

Secondly, calculated fields must be set up in the LOAN table, the values of which are Forename and Surname (using the ‘Loan to Member’ relationship), as shown in Figure 6.20. (Make sure that the Storage Options are set so that the result of the calculated field is not stored.)
Figure 6.20: Defining a calculated field required to perform a search involving more than one table

Figure 6.21: Find request to perform a search involving more than one table.
Search 5: More than two tables
To identify the names of members who have rented Shrek:

1. the fields required are Forename and Surname (from the MEMBER table) and Film Title (from the FILM table)
2. the MEMBER and FILM tables are required. However, these are not directly linked by a foreign key
3. the criteria are Film Title = Shrek.

In this example, all four tables are required in order to link records from the MEMBER and FILM tables. As before, the LOAN entity is the main link entity. Therefore, a layout is created in the LOAN table.

Firstly, relationships must be set-up to link LOAN to MEMBER, and LOAN to DVD (if not already created). In addition, because the search involves Film Title, a relationship must be established to directly link LOAN to FILM.

Secondly, calculated fields must be added to the LOAN table for the Film Code (using the ‘Loan to DVD’ relationship), Film Title (using the ‘Loan to Film’ relationship) and Forename and Surname (using the ‘Loan to Member’ relationship), as shown in Figure 6.22.

Figure 6.22: Calculated fields required to perform a search involving more than two tables
Finally, consider a request for a list of members who have rented either Shrek or Finding Nemo since the 16 August 2005.

1. The fields required are: Forename and Surname (from the MEMBER table), Film Title (from the FILM table), and Date Hired (from the LOAN table).
2. As before, all four tables are required.
3. The criteria are more complicated this time:
   Film Title = Shrek AND Date Hired > 16/08/2005
   OR
   Film Title = Finding Nemo AND Date Hired > 16/08/2005.

Note that the Date Hired condition must be repeated for each film.

As in Search 5, relationships are set up: ‘Loan to Member’, ‘Loan to DVD’ and ‘Loan to Film’. Similarly, calculated fields are created for Forename, Surname and Film Title.
Sorting

Usually, the results of a selection query are presented in some kind of order. A sorting operation is performed to achieve this. To perform a sort, two items must be identified.

1. Which field (or fields) will be used to decide the order of records?
2. For each field selected, will the order of sorting be **ascending** or **descending**?

For example:

- to produce a list of people with the tallest first, the records would be sorted in **descending** order of **height**
- to produce a list of people with youngest first, the records would be sorted in **ascending** order of **age**.

A very common way of ordering records relating to people is in alphabetical order. To achieve alphabetical ordering requires the records to be sorted in **ascending** order of **surname**.

A **complex sort** involves more than one sort condition, involving two or more fields.

For example, as we have seen in Chapter 4, a person’s name is usually stored in three fields: **Title** (e.g. Mr/Mrs/Miss/Ms), **Forename** and **Surname**. To achieve ‘telephone book order’, the name is sorted in **ascending** order of **surname**, then **ascending** order of **forename**. In this case, the **Surname** field is the **primary sort key**, and the **Forename** field is the **secondary sort key**. Figure 6.25 shows how such a complex sort is set up.
Figure 6.25: A complex sort. Note that the fields used (Surname and Forename) must be selected in the correct order, with the primary sort key listed before the secondary sort key.

Calculations

One of the main advantages that computers have over humans is the ability to perform calculations very quickly, as many times as necessary, without errors.

In a relational database, calculations are performed using expressions or formulas. Here is an expression to calculate a student’s total mark from three tests:

\[ \text{Total Mark} = [\text{Test 1}] + [\text{Test 2}] + [\text{Test 3}] \]

Some calculations are performed for each record, using other values in the record. This type of calculation is a horizontal calculation.

Other calculations are vertical calculations, which are based on the value of a single field taken from a set of records. The average mark for Test 1 would be an example, where the expression required would be similar to \( \text{Average}([\text{Test 1}]) \).
Horizontal calculations are often known as **calculated fields**, and vertical calculations are known as **summary fields**.

All calculated values are examples of derived data, which are not normally stored in a relational database. In Filemaker Pro, horizontal calculations are performed using **calculated fields**. However, unlike other fields, the results of a calculated field are not stored within the database (although this setting can be overridden).

Most RDBMSs provide a range of **functions** that can be used in expressions. The table below lists some of the main categories of these functions, with some examples of common functions for each category.

<table>
<thead>
<tr>
<th>Category</th>
<th>Example of functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate</td>
<td>Sum, Average, Max, Min, Count</td>
</tr>
<tr>
<td>Mathematical</td>
<td>Sin, Cos, Tan, Int, Round</td>
</tr>
<tr>
<td>Text</td>
<td>Left, Right, Middle, Length, Upper, Lower, Position, Replace</td>
</tr>
<tr>
<td>Logical</td>
<td>If, IsEmpty, IsValid, FieldType</td>
</tr>
<tr>
<td>Conversion</td>
<td>NumToText, TextToNum, DateToText, TextToDate</td>
</tr>
<tr>
<td>Date</td>
<td>Today, Day, Month, Year, DayOfWeek, DayName</td>
</tr>
<tr>
<td>Time</td>
<td>Time, Hour, Minute, Seconds</td>
</tr>
</tbody>
</table>
Working with dates

In many databases where dates are recorded (e.g. date of birth, date hired, date returned, etc.) it is necessary to be able to perform calculations involving dates.

For example:

• to search for all DVDs rented in the last week, you would use the expression:

  \[ \text{Today} - \text{Date Hired} \leq 7 \]

• to calculate a person's age in years from their date of birth is not nearly as simple, as there is no pre-defined function to calculate date differences. Here it is...

  \[
  \text{Case}(\text{Month(Today)} > \text{Month(Date of Birth)}, \text{Year(Today)} - \text{Year(Date of Birth)}, \text{Month(Today)} < \text{Month(Date of Birth)}, \text{Year(Today)} - \text{Year(Date of Birth)} - 1, \text{Day(Today)} \geq \text{Day(Date of Birth)}, \text{Year(Today)} - \text{Year(Date of Birth)}, \text{Year(Today)} - \text{Year(Date of Birth)} - 1)
  \]

Parameterised queries

Many database management systems allow queries to be saved. This allows the queries to be used in the future without having to set them up each time they are required.

However, very often a query is required to be used many times, but with varying criteria. For example, consider a query set up to list all DVD rentals for July. To list the DVD rentals for August would either require a new query to be set up, or the existing query to be changed.

An alternative that is available in some RDBMSs is to use a parameterised query. This is a query where the user enters a value to be searched for. Figure 6.26 shows the design of a parameterised query.
Figure 6.26: A parameterised query is created using a script in which the step 'Enter Find Mode' is set to 'Pause' and with 'Restore find requests' unchecked. This will allow the user to enter the search criteria.

**User views**

One of the advantages that a DBMS has over a spreadsheet is its ability to present information in a variety of different ways without changing the underlying structure or content of the data stored. These are called **user views** and are created using layouts in Filemaker Pro. A layout may be based on a script, which performs the required searching or sorting operations.
Report structure

A database report is made up of a number of sections, as shown in Figure 6.27.

*Figure 6.27: Structure of a database report*

<table>
<thead>
<tr>
<th><strong>Report Header</strong></th>
<th>Text/data/data to appear at the top of the report</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Page Header</strong></td>
<td>Text/data to appear at the top of each page of the report</td>
</tr>
<tr>
<td><strong>Main Detail Header</strong></td>
<td>Text/data to appear above each entry in the main detail section</td>
</tr>
<tr>
<td><strong>Main Detail Section</strong></td>
<td>Data from selected records in a table or query</td>
</tr>
<tr>
<td><strong>Main Detail Footer</strong></td>
<td>Text/data to appear below each entry in the main detail section</td>
</tr>
<tr>
<td><strong>Page Footer</strong></td>
<td>Text/data to appear at the foot of each page of the report</td>
</tr>
<tr>
<td><strong>Report Footer</strong></td>
<td>Text/data to appear at the end of the report</td>
</tr>
</tbody>
</table>

**DVD Rental Statistics**

<table>
<thead>
<tr>
<th>Details for J Bloggs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Memb No</strong></td>
</tr>
<tr>
<td>142312</td>
</tr>
</tbody>
</table>

Total DVDs rented to date: 26

*End of page 1*

Total DVDs rented by all members: 3,218
Summary information

One of the aspects that distinguishes a relational database from a spreadsheet is the ability to summarise information.

The five most common summary calculations that are performed are as follow:

- **Sum** to add values to give a total, e.g. total cost of DVD rentals last month.
- **Average** to find an average value, e.g. average cost of hire per DVD.
- **Count** to count the number of records found, e.g. number of DVDs rented per member.
- **Maximum** to find the highest value, e.g. highest number of rentals per DVD (to find the most popular DVD).
- **Minimum** to find the lowest value, e.g. lowest number of rentals per member.

Summary information is produced by creating a **summary field**. A summary field is a calculated field with a formula to perform the calculation, and is placed in a **summary part** of a layout.

For example, to calculate the total cost of DVD rentals for a member, a summary field would be defined in the LOAN table to calculate the Total Cost, as shown in Figure 6.28.

Figure 6.28: **Defining a summary field to calculate the total cost of DVD rentals in the LOAN table.**
A summary part would be placed in a layout in the MEMBER table, as shown in Figure 6.29.

*Figure 6.29: Defining a summary part in the MEMBER table*

Finally, the summary field would be placed in the summary section of a layout, as shown in Figure 6.30.
Figure 6.30: A layout in the MEMBER table with summary field in summary section. This layout uses a portal (see Sub-forms below) based on a relationship ‘Member to Loan’, which lists details of the loans for each member.

Used in this way, the summary field will calculate and display the total cost of rentals for each member. However, the advantage of a DBMS is that the report can be used with the results of any query, so that we could use it to find the total cost of rentals for members who have rented DVDs in the last week only.

In addition, by placing the same summary field in a different summary section, we can get the database to calculate a different result.

**Sub-forms and sub-reports**

Because relational databases are designed to represent data that is related, very often a single table of results is not adequate for presenting the data. Instead, what is required is a **sub-table**. In Filemaker Pro, **portals** are used to create sub-forms and sub-reports.
Navigation

Relational databases form the basis of many complex applications. For example, e-commerce websites such as Amazon are based on relational databases, which store information about customers, products, and orders. However, what makes these websites so successful is their ease of use.

Figure 6.32: Navigation bar and search box from Amazon website
In Figure 6.32, when you click on the DVD tab on the navigation bar, the browser loads up the DVD web page, which performs a search for information on DVDs from the relational database on the Amazon web server.

If you enter a word or phrase into the keyword search box, a web page will be displayed showing the results of a search on the relational database, with the results sorted in order. Figure 6.33 shows the results of a search for ‘ring’.

*Figure 6.33: Search results for ‘ring’ showing the first three records from each category sorted alphabetically (ascending order) by title*

When you add items to your shopping basket, the current total of the items in the basket is calculated and displayed on the web page (Figure 6.34).
All these searching, sorting and calculating operations are performed by the relational database without the user being aware of using it.

We can achieve something similar with desktop database packages such as Microsoft Access and Filemaker Pro, using navigation features available with the software. For example, Microsoft Access provides a number of buttons that allow the user to find records, add new records, delete records, navigate between records, open forms, print reports, etc. Figures 6.35 and 6.36 show how a customised user interface to a database form can improve its user-friendliness.
Figure 6.35: A form with sub-form in Filemaker Pro using a layout with a portal

Figure 6.36: A customised form with sub-form in Filemaker Pro. Note the use of customised buttons
Macros and scripting

In Figure 6.36, the buttons that provide navigation and other user interface features use scripting to carry out their tasks. Figure 6.37 shows a script definition.

Figure 6.37: Defining a script in Filemaker Pro using ScriptMaker

An alternative to scripting is to use macros. A macro is a sequence of one or more actions that each perform a particular operation within the DBMS. Macros are particularly useful for automating common tasks. In Filemaker Pro, macros are implemented using scripts.
Introducing SQL

Relational database management systems (RDBMSs) are based on a data definition language called Structured Query Language or SQL (often pronounced ‘sequel’). In fact, some RDBMSs take their name from SQL – for instance, SQL Server and MySQL.

SQL is an example of a **data definition language** (DDL). That is, it can be used to create tables and define fields.

Figure 6.39 shows statements in SQL to create a database called DVD Rentals, and define a table called MEMBER with fields Member Number, Name, Address, Post Code and Telephone Number.

*Figure 6.39: SQL statements to create a database and define a table*

```
CREATE DATABASE 'DVD Rentals'
CREATE TABLE Member('Member Number' INTEGER, Name VARCHAR(30), Address VARCHAR(50), 'Post Code' VARCHAR(7), 'Telephone Number' VARCHAR(11))
```

SQL is also a **data manipulation language** (DML). That means it can be used to add and delete records, search for records, sort records into order and perform calculations and summaries. In fact, everything you expect to be able to do using a database!

*Figure 6.40: An SQL statement to insert a new record into the MEMBER table*

```
INSERT INTO Member
VALUES ('233', 'Jo Soap', '1 Getting Close', 'IS99 2QZ', '123456')
```
Figure 6.41: An SQL statement to select all fields from the MEMBER table for records for Member Numbers 1–250 and list these in descending order of Member Number

```sql
SELECT * FROM Member
WHERE 'Member Number' BETWEEN 1 and 250
ORDER BY 'Member Number' DESC
```

Figure 6.42: An SQL statement to delete records from the MEMBER table for Members Numbers over 500.

```sql
DELETE FROM Member
WHERE 'Member Number' > 500
```

Exercise 6

1. Figure 6.33 shows some search results for the keyword ‘ring’ from the Amazon website.

   The results of the search in each category indicate the total number of results found, e.g. ‘See all 89 results in DVD…’

   Explain how the total number of results found can be calculated by the relational database.

2. Figure 6.34 shows a customer’s shopping basket.

   Explain how the sub-total of £47.55 can be calculated by the relational database.
Chapter 7: Advanced implementation in MySQL/PHP

Having analysed a data model to third normal form, and designed a data dictionary, you are now ready to implement your database.

You can think of a database solution as being like a tower. The tables form the foundations, which is why the data dictionary is so important in helping to get the design of the tables right.

The next level of the tower is queries, which are used to perform searching, sorting and calculations.

Next, user views of the data in the database are used for input and output of data to and from the tables, e.g. to display the results of queries.

Navigation features are added to the user interface to connect the forms, reports and layouts together so that the user can easily perform various tasks, using the database.

Introducing SQL

Relational database management systems (RDBMSs) are based on a data definition language called Structured Query Language or SQL (often pronounced ‘sequel’). In fact, MySQL takes its name from SQL. SQL is an example of a data definition language (DDL). That is, it can be used to create tables and define fields.

Figure 7.1 shows statements in SQL to create a database called DVD Rentals, and define a table called MEMBER with fields Member Number, Name, Address, Post Code and Telephone Number.
SQL is also a **data manipulation language (DML)**. That means it can be used to add and delete records, search for records, sort records into order and perform calculations and summaries. In fact, everything you expect to be able to do using a database!

MySQL is an RDMS that is widely used for creating ‘dynamic’ websites. A dynamic website is one where the contents change depending on the needs of the user. Search engines and e-commerce sites are examples of dynamic websites. A special language called PHP is often used with...
MySQL to link the database to a web interface. PHP is an example of an interface generation language.

You can download MySQL free from www.mysql.com

The following screenshots are taken from the phpmyadmin web interface for MySQL. You can download phpmyadmin free from www.phpmyadmin.net

You can experiment with MySQL and phpmyadmin at www.phpmyadmin.net/phpMyAdmin

phpMyAdmin is a web interface for MySQL that allows you to perform many database functions without the need to type in SQL script, as shown in Figures 7.5 to 7.9.

*Figure 7.5: Creating a database table in MySQL (using the phpmyadmin web interface)*

*Figure 7.6: Viewing a table that has been created in MySQL*
Figure 7.7: Inserting a record in MySQL

![Inserting a record in MySQL](image)

Figure 7.8: Selecting records in MySQL

![Selecting records in MySQL](image)
Tables and referential integrity

In a relational database, tables are linked through the use of foreign keys. A foreign key is an attribute that is not a key in its own table, but is a primary key in another table. A foreign key links a record in one table with a related record in another table.

For the database to work properly, it is essential to guarantee that a foreign key always refers to a record that exists in the other table. This is called referential integrity.

For example, an entry in the LOAN table can only refer to an existing record in the MEMBER table, because a loan can only be made by a member.

A further aspect of referential integrity is that if a record is deleted from one table, then it may be necessary to delete the references to that record in other tables. So, if a member leaves the club, then all loans made by that member can also be deleted.

Achieving referential integrity

As well as checking for referential integrity, many RDBMSs provide a way of ensuring that the user cannot mistakenly enter an invalid value for a foreign key.

MySQL has no internal facility to maintain referential integrity (with the default MyISAM table type) – instead, the database’s interface must be designed in such a way that a check is made on any value entered to ensure it is valid.
Usually, this is done by using a list on a web page to display the available choices for a field value, from which the user may make a selection.

**More about validation**

This section shows how to implement a number of standard validation checks using MySQL.

**Presence check**

*Figure 7.10: Achieving a presence check by setting the ‘Null’ property to ‘not null’. Key fields also have the ‘Primary’ property set*

**Range check**

Like SQL itself, MySQL has no built-in method of performing a range check. Instead, the interface language (e.g. PHP) is used to check a value being entered before the value is stored.

**Restricted choice check**

*Figure 7.11: Achieving a restricted choice check using an enumerated field type, with a list of comma-separated quoted values in the Length/Values option*

*Figure 7.12: Achieving a Boolean field. An enumerated field is created with values ‘Yes’ and ‘No’*

**Default values**

*Figure 7.13: A default value setting for a floating point field*
Formatting

Formatting settings affect how data is displayed. The most common formatting options affect numeric, date and time fields. With MySQL, all the formatting is performed by an interface generation language, such as PHP.

Queries

The term query is used to describe any process that performs searching or sorting operations in a relational database. In MySQL, queries are also used to perform calculations. All searching, sorting and calculation queries are performed using an SQL expression string.

Searching

Searching is the process of selecting records from a table or combination of tables. To perform the query, three items must be identified.

1. Which fields will be used to identify the records required?
2. Which tables are required to produce the data required?
3. What are the criteria for identifying the records required?

**Search 1: Single condition**

For example, to identify the names and telephone numbers of club members with Member Number over 1000:

1. the fields required are Forename, Surname and Telephone Number
2. the MEMBER table is required, as it contains the fields listed
3. the criteria are Member Number > 1000.

The SQL expression string required:

```
SELECT 'Forename', 'Surname', 'Telephone Number'
FROM Member
WHERE 'Member Number' > 1000
```
The asterisk (*) symbol can be used in an expression string to extract all fields from a table:

```
SELECT *
FROM Member
WHERE 'Member Number' > 1000
```

A complex search involves more than one search condition (involving one or more fields).

**Search 2: Multiple conditions (AND)**

To identify the names and telephone numbers of club members with Surname beginning with M and Member Number over 1000:

1. as before, the fields required are Forename, Surname and Telephone Number
2. as before, the MEMBER table is required
3. the criteria are Surname like “M*” AND Member Number > 1000.

The SQL expression string required is:

```
SELECT 'Forename', 'Surname', 'Telephone Number'
FROM Member
WHERE 'Surname' LIKE "M*" AND 'Member Number' > 1000
```
**Search 3: Multiple conditions (OR)**

To identify the names and telephone numbers of club members with Surname beginning with M or N:

1. as before, the fields required are Forename, Surname and Telephone Number
2. as before, the MEMBER table is required
3. the criteria are Surname like “M*” OR Surname like “N*”.

The SQL expression string required is:

```sql
SELECT 'Forename', 'Surname', 'Telephone Number'
FROM Member
WHERE 'Surname' LIKE 'M*' OR 'Surname' LIKE 'N*'
```

**Search 4: More than one table**

To identify the names of club members who rented a DVD in August 2005, together with the rental dates:

1. the fields required are Forename, Surname and Date Hired
2. the tables required are MEMBER and LOAN
3. the criteria are Date Hired LIKE “/*/8/2005”.

This time the data required is contained within two tables. In order to extract the related records from each table, a **join** operation is performed in SQL. This will combine those records from each table, which have a corresponding record in the other table, into a single set of related records (called an **answer set**). The keyword used to perform this is **INNER JOIN**.

There are four types of **JOIN** operation available in SQL. An **INNER JOIN** is the most commonly used, and selects only those records from each table which have a related record in the other table (e.g. members who have loans). A **LEFT INNER JOIN** selects all records from Table 1, and those records from Table 2 which have related records in Table 1 (e.g. all members, even those who have no loans). A **RIGHT INNER JOIN** selects all records from Table 2, and those records from Table 1 which have related records in Table 2. An **OUTER JOIN** selects all records from both tables (e.g. all members and all loans).
The SQL expression string required for Search 4 is:

```sql
SELECT 'Forename', 'Surname', 'Date Hired'
FROM (Member INNER JOIN Loan)
ON 'Member.Member Number'='Loan.Member Number'
WHERE 'Date Hired' LIKE "*/8/2005"
```

**Search 5: More than two tables**

To identify the names of members who have rented Shrek:

1. the fields required are Forename and Surname (from the MEMBER table) and Film Title (from the FILM table)
2. this time all four tables are required: MEMBER, LOAN, DVD and FILM. As shown by the 3NF E-R diagram, Members and Films are only linked via the LOAN and DVD entities. A nested INNER JOIN statement must be performed to combine these together into a single answer set
3. the criteria are Film Title = “Shrek”.

The SQL expression string required is:

```sql
SELECT 'Forename', 'Surname', 'Title'
FROM (((Member INNER JOIN Loan
ON Member.Member Number=Loan.Member Number)
    INNER JOIN DVD ON Loan.DVD Code = DVD.DVD Code)
    INNER JOIN Film ON DVD.Film Code = Film.Film Code)
WHERE 'Title' = "Shrek"
```
Search 6: Dealing with duplicates
Consider a request to list those members who have rented Shrek or Finding Nemo.

1. The fields required are Forename and Surname (from the MEMBER table) and Film Title (from the FILM table).
2. As before, all four tables are required.
3. The criteria are Film Title = “Shrek” OR Film Title = “Finding Nemo”.

The SQL expression string required is:

```sql
SELECT 'Forename', 'Surname', 'Title'
FROM (((Member INNER JOIN Loan
ON Member.Member Number=Loan.Member Number)
    INNER JOIN DVD ON Loan.DVD Code = DVD.DVD Code)
    INNER JOIN Film ON DVD.Film Code = Film.Film Code)
WHERE 'Title' = "Shrek" OR 'Title' = "Finding Nemo"
```

Note that it is possible that a member may have rented both Shrek and Finding Nemo. In this case, the member’s name and telephone number would be listed twice in the answer set. These duplicate results can be eliminated. The ‘Group By’ option is used to achieve this, as follows:

```sql
SELECT 'Forename', 'Surname', 'Title'
FROM (((Member INNER JOIN Loan
ON Member.Member Number=Loan.Member Number)
    INNER JOIN DVD ON Loan.DVD Code = DVD.DVD Code)
    INNER JOIN Film ON DVD.Film Code = Film.Film Code)
WHERE 'Title' = "Shrek" OR 'Title' = "Finding Nemo"
GROUP BY *
```
Search 7: Combining multiple conditions

Finally, consider a request for a list of members who have rented either Shrek or Finding Nemo since the 16 August 2005.

1. The fields required are: Forename and Surname (from the MEMBER table), Film Title (from the FILM table), and Date Hired (from the LOAN table).
2. As before, all four tables are required.
3. The criteria are more complicated this time:
   Film Title = “Shrek” AND Date Hired > 16/08/2005
   OR
   Film Title = “Finding Nemo” AND Date Hired > 16/08/2005.

Note that the Date Hired condition must be repeated for each film. The SQL expression string required is:

```sql
SELECT 'Forename', 'Surname', 'Title'
FROM (((Member INNER JOIN Loan
             ON Member.Member Number=Loan.Member Number)
          INNER JOIN DVD ON Loan.DVD Code = DVD.DVD Code)
       INNER JOIN Film ON DVD.Film Code = Film.Film Code)
WHERE ('Title' = "Shrek" AND 'Date Hired' > 16/08/2005)
OR ('Title' = "Finding Nemo" AND 'Date Hired' > 16/08/2005)
```

Note that the brackets are required to ensure that the search conditions are correctly applied. Great care is required when combining search conditions in this way to ensure that the correct records are returned in the answer set.

Sorting

Usually, the results of a selection query are presented in some kind of order. A sorting operation is performed to achieve this. To perform a sort, two items must be identified.

1. Which field (or fields) will be used to decide the order of records?
2. For each field selected, will the order of sorting be ascending or descending?
For example:

- to produce a list of people with the tallest first, the records would be sorted in *descending order of height*
- to produce a list of people with youngest first, the records would be sorted in *ascending order of age*.

A very common way of ordering records relating to people is in alphabetical order. To achieve alphabetical ordering requires the records to be sorted in *ascending order of surname*.

A **complex sort** involves more than one sort condition, involving two or more fields.

For example, as we have seen in Chapter 4, a person’s name is usually stored in three fields: title (e.g. Mr/Mrs/Miss/Ms), Forename and Surname. To achieve ‘telephone book order’, the name is sorted in *ascending order of surname*, then *ascending order of forename*. In this case, the Surname field is the **primary sort key**, and the Forename field is the **secondary sort key**.

The following SQL expression string illustrates a complex sort in MySQL:

```
SELECT ‘Forename’, ‘Surname’, ‘Telephone Number’
FROM Member
WHERE ‘Member Number’ > 1000
ORDER BY ‘Surname’, ‘Forename’
```

The sort order is ascending by default. The **DESC** keyword is used to sort in descending order:

```
SELECT ‘Forename’, ‘Surname’, ‘Telephone Number’
FROM Member
WHERE ‘Member Number’ > 1000
ORDER BY ‘Member Number’ DESC
```
Calculations

One of the main advantages that computers have over humans is the ability to perform calculations very quickly, as many times as necessary, without errors.

In MySQL, calculations are performed using SQL expression strings. Here is an expression string to calculate a student’s total mark from three tests:

```sql
SELECT 'Forename', 'Surname', 'Total' AS 'Test1' + 'Test2' + 'Test3'
FROM Student
```

Some calculations are performed for each record, using other values in the record. This type of calculation is a horizontal calculation.

Other calculations are vertical calculations, which are based on the value of a single field taken from a set of records. The average mark for Test 1 would be an example.

<table>
<thead>
<tr>
<th>Name</th>
<th>Test 1</th>
<th>Test 2</th>
<th>Test 3</th>
<th>Total Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>record 1</td>
<td>J Bloggs</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>record 2</td>
<td>J Public</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Horizontal calculations are often known as calculated fields, and vertical calculations are known as summary fields.

In MySQL, the GROUP BY statement, which was introduced in Search 6, is also used to produce summary information for a set of records. For example:
SELECT 'StudentID', AVG('Test')
FROM Student
GROUP BY 'StudentID'

Produces the average of all Test results for each student

SELECT 'StudentID', COUNT(*)
FROM Student, Course
WHERE Student.CourseID = Course.CourseID
GROUP BY 'StudentID'

Produces the number of rows returned. The GROUP BY clause removes duplicate entries.

All calculated values are examples of derived data, which are not normally stored in a relational database. In MySQL, the derived data is not stored but calculated when required using the expression string.

MySQL provides a number of functions that can be used in SQL expression strings. PHP also provides a number of functions for using with data from a MySQL database. The table below lists some of the main categories of these functions, with some examples of common functions for each category.

<table>
<thead>
<tr>
<th>Category</th>
<th>Example of functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate</td>
<td>MySQL: SUM, AVG, MAX, MIN, COUNT</td>
</tr>
</tbody>
</table>
| Mathematical   | MySQL: SIN, COS, TAN, TRUNCATE, ROUND
PHP: Sin, Cos, Tan, Floor, Round |
| Text           | MySQL: LEFT, RIGHT, SUBSTRING, LENGTH, LOWER, UPPER, LOCATE, REPLACE
PHP: SubStr, StrLen, StrToUpper, StrToLower, StrPos, SubStr_Replace |
| Logical        | MySQL: IF, IFNULL
PHP: If, While, For, ForEach, Switch                     |
| Conversion     | MySQL: FORMAT, CONVERT, DATE_FORMAT, STR_TO_DATE
PHP: StrToTime,                                        |
| Date           | MySQL: CURDATE, DATE_FORMAT, DAYOFMONTH, DAYOFWEEK, DAYNAME, MONTH, MONTHNAME, DATEDIFF, DATE_SUB
PHP: GetDate, Date                                      |
| Time           | MySQL: CURTIME, HOUR, MINUTE, SECOND, TIMEDIFF
PHP: getTimeOfDay, StrFTime                             |
Working with dates

In many databases where dates are recorded (e.g. date of birth, date hired, date returned, etc.) it is necessary to be able to perform calculations involving dates.

For example:

• to search for all DVDs rented in the last week, you would use the expression:

```
SELECT * FROM Loan
WHERE DATEDIFF(CURDATE(), ‘Date Hired’) < 7
```

• to calculate a person’s age in years from their date of birth, you would use the expression

```
SELECT (YEAR(CURDATE()) – YEAR(’Date Of Birth’)
– (RIGHT(CURDATE(), 5) < RIGHT(’Date Of Birth’, 5)) AS Age
```

Parameterised queries

Many database management systems allow queries to be saved. This allows the queries to be used in the future without having to set them up each time they are required.

However, very often a query is required to be used many times, but with varying criteria. For example, consider a query set up to list all DVD rentals for July. To list the DVD rentals for August would either require a new query to be set up, or the existing query to be changed.

An alternative that is available in some RDBMSs is to use a parameterised query.
In MySQL, a variable can be used within an SQL expression string, as shown on lines 10 and 11 below.

```php
<?php
    if ($submitsearch == "SUBMIT") {
        $sql = "SELECT 'MEMBER NUMBER'
               WHERE 'DATE HIRED' = $searchdate";
        if (@mysql_query($sql)) {
            ...
        }
    }

User views

One of the advantages that a DBMS has over a spreadsheet is its ability to present information in a variety of different ways without changing the underlying structure or content of the data stored. These are called user views, and are created using forms and reports.

A form or report is usually based on a query, which selects the required fields from the appropriate tables, sorting the results if necessary, and performing any horizontal calculations. In a MySQL application, all output is generally in the form of a web page.
Report structure

A database report is made up of a number of sections, as shown in Figure 7.14.

*Figure 7.14: Structure of a database report*

<table>
<thead>
<tr>
<th>Report Header</th>
<th>Text/data/data to appear at the top of the report</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page Header</td>
<td>Text/data to appear at the top of each page of the report</td>
</tr>
<tr>
<td>Main Detail Header</td>
<td>Text/data to appear above each entry in the main detail section</td>
</tr>
<tr>
<td>Main Detail Section</td>
<td>Data from selected records in a table or query</td>
</tr>
<tr>
<td>Main Detail Footer</td>
<td>Text/data to appear below each entry in the main detail section</td>
</tr>
<tr>
<td>Page Footer</td>
<td>Text/data to appear at the foot of each page of the report</td>
</tr>
<tr>
<td>Report Footer</td>
<td>Text/data to appear at the end of the report</td>
</tr>
</tbody>
</table>

### DVD Rental Statistics

**Page 1**

**Details for J Bloggs**

<table>
<thead>
<tr>
<th>Memb No</th>
<th>Address</th>
<th>Tel No</th>
</tr>
</thead>
<tbody>
<tr>
<td>142312</td>
<td>Main Street</td>
<td>123456</td>
</tr>
</tbody>
</table>

Total DVDs rented to date: 26

*End of page 1*

Total DVDs rented by all members: **3,218**
Summary information

One of the aspects that distinguishes a relational database from a spreadsheet is the ability to summarise information.

The five most common summary calculations that are performed are as follow:

- **Sum** to add values to give a total, e.g. total cost of DVD rentals last month.
- **Average** to find an average value, e.g. average cost of hire per DVD.
- **Count** to count the number of records found, e.g. number of DVDs rented per member.
- **Maximum** to find the highest value, e.g. highest number of rentals per DVD (to find the most popular DVD).
- **Minimum** to find the lowest value, e.g. lowest number of rentals per member.

Summary information is produced by creating a **summary field**. A summary field is a calculated field with a formula to perform the calculation, and is placed in a **summary section** of a report.

In a MySQL application, the aggregate functions SUM, AVG, COUNT, MAX and MIN can be used in an SQL expression string to produce summary information for a set of extracted records. However, certain summary information, such as page totals, would have to be programmed using PHP (e.g. counting the number of records being displayed).

However, the advantage of a database is that the report can be used with the results of any query, so that we could use it to find the total cost of rentals for members who have rented DVDs in the last week only.

**Sub-forms and sub-reports**

Because relational databases are designed to represent data that is related, very often a single table of results is not adequate for presenting the data. Instead, what is required is a **sub-table**.

In MySQL, two separate queries must be executed: one for the main header information, and a second one for the sub-table information. It is up to you to decide how best to present the information.
Navigation

Relational databases form the basis of many complex applications. For example, e-commerce websites such as Amazon are based on relational databases, which store information about customers, products, and orders. However, what makes these websites so successful is their ease of use.

*Figure 7.15: Navigation bar and search box from Amazon website*

In Figure 7.15, when you click on the DVD tab on the navigation bar, the browser loads up the DVD web page, which performs a search for information on DVDs from the relational database on the Amazon web server.

If you enter a word or phrase into the keyword search box, a web page will be displayed showing the results of a search on the relational database, with the results sorted in order. Figure 7.16 shows the results of a search for ‘ring’.
When you add items to your shopping basket, the current total of the items in the basket is calculated and displayed on the web page (Figure 7.17).
All these searching, sorting and calculating operations are performed by the relational database without the user being aware of using it.

By developing some simple techniques using PHP, the same effect can be easily achieved using a MySQL database. Appendix C contains some HTML/PHP code for a simple database, which you can try out for yourself.
Macros and scripting

In a MySQL application, all navigation and user interface features are implemented using scripting – the PHP code within the web pages. Figure 7.18 shows some PHP scripting to perform an SQL expression and display the results.

Figure 7.18: Scripting in PHP

```php
$result = @mysql_query("SELECT Forename, Surname FROM Member");

while ($row = mysql_fetch_array($result)) {
    $forename = $row["Forename"];  
    $surname = $row["Surname"]; 
    echo("<tr><td>$forename</td><td>$surname</td></tr>";  
}
```

An alternative to scripting in a desktop DBMS is to use macros. A **macro** is a sequence of one or more actions that each perform a particular operation within the DBMS. Macros are particularly useful for automating common tasks. Figure 7.19 shows how a simple macro would be created to open a form and navigate to the last record selected from a query’s results.

Figure 7.19: Creating a macro in Microsoft Access
Exercise 7

1. Figure 7.16 shows some search results for the keyword ‘ring’ from the Amazon website.

The results of the search in each category indicate the total number of results found, e.g. ‘See all 89 results in DVD…’

Explain how the total number of results found can be calculated by the relational database.

2. Figure 7.17 shows a customer’s shopping basket.

Explain how the sub-total of £47.55 can be calculated by the relational database.