

Higher Computing Science Information Systems Design & Development

Structures and links (Databases)



An **information system**, an integrated set of components for collecting, storing, and processing data and for delivering information, knowledge, and digital products.

Business firms and other organisations rely on information systems to carry out and manage their operations, interact with their customers and suppliers, and compete in the marketplace. For instance, corporations use information systems to reach their potential customers with targeted messages over the Web, to process financial accounts, and to manage their human resources. Governments deploy information systems to provide services cost-effectively to citizens. Digital goods, such as electronic books and software, and online services, such as auctions and social networking, are delivered with information systems. Individuals rely on information systems, generally Internet-based, for conducting much of their personal lives: for socialising, study, shopping, banking, and entertainment.

Content

National 5

- database structure: flat file, linked tables, primary keys and foreign keys
- field types (text, number, date, time, graphic, object, calculated, link, Boolean)
- validation (including presence check, restricted choice, field length and range)
- database operations search, sort (on multiple fields)
- good design to avoid data duplication and modification errors (insert, delete, update)

Higher

- design notations (entity relationship diagram + data dictionary)
- relationships (one-to-one, one-to-many, many-to-many)
- database structures: relational
- primary keys, including compound keys
- complex database operations (including queries, forms, reports, calculating)

1.1 Introduction

A **database** is used to store information. Databases can contain thousands of pieces of information, stored in a variety of formats. Databases are used as they can be searched and sorted very efficiently and they can allow a number of people to use the same information simultaneously.

Databases contain tables. Each **table** contains records. One **record** is all the data stored about one person or one object. The records contain fields; a **field** is one single piece of information. All the records in a table must have the same fields. Fields can have many different data types.

Databases can contain more than one table and these tables can be linked using **primary keys** and **foreign keys**.

1.2 Database structures: relational

In the previous section you learned how a simple database is constructed and the concept of a table. If a database only has one table then it can be referred to as a **flat-file database**.

A **relational database** is a database which contains more than one table. The tables are linked together by using primary and foreign keys.

Advantages of a Relational Database:

Relational databases were developed to avoid unnecessary duplication of data in the database. Let's look at a Flat-file database to see what we mean by unnecessary duplication:

Library Loans:

Borrower No	Borrower Name	Borrower Address	Borrower Phone	Book No	Title	Author	Loaned Out	Due Back
1001	Egdar Codd	12 High Street, Scholarville	01234 985443	F1301	Wuthering Heights	Bronte, C	12 Jan	11 Feb
1002	Raymond Boyce	6 Castle View, Scholarville	01234 983458	F1301	Wuthering Heights	Bronte, C	12 Feb	11 Mar
1001	Egdar Codd	12 High Street, Scholarville	01234 985443	F1109	To Kill a Mockingbird	Lee, H	14 Jan	13 Feb

As you can see, all the fields about the lender (Lender No, Name, Address...) have to be inserted for each book the lender loans, and the same is also true of the book details each time it is loaned out. This is unnecessary duplication, and can cause problems:

- It is time consuming for the database operator (this is a simplified example - a real lender may need many fields).
- Mistakes can be made (can you see any in the table above?).
- What if the lender leaves the library? The Data Protection Act would mean the library would have to remove the lender's details from the database - how would this work if they have loaned hundreds of books!
- What if you want to add book details to the library database - ideally you would need someone to lend the book so the record is complete.

Let's take a look at a relational version of this database now:

Lender:

Lender No	Lender Name	Lender Address	Lender Phone
1001	Egdar Codd	12 High Street, Scholarville	01234 985443
1002	Raymond Boyce	6 Castle View, Scholarville	01234 983458

Book:

Book No	Title	Author
F1301	Wuthering Heights	Bronte, E
F1109	To Kill a Mockingbird	Lee, H

Loan:

Lender No	Book No	Loaned Out	Due Back
1001	F1301	12 Jan	11 Feb
1002	F1301	12 Feb	11 Mar
1001	F1109	14 Jan	13 Feb

As you can see now:

- all the information about books are stored only once;
- all the information about lenders are stored only once;
- all the information about loans are stored only once.

Each loan is linked (related) by the **book No** and **lender No** to the record in each table. This solves all of the above problems (well, all bar the mistake problem. However, at least if an operator enters the wrong value for some book or borrower data it only needs to be changed once and all loan records will be correct).

1.3 Primary keys

A **primary key** is a field in a table that contains a piece of data that is unique for every record.

This could be a phone number, email address (e.g. CustomerService@Amazon.com), national insurance number (e.g. HMRC records) or even a student's Scottish Candidate Number (e.g. SQA Record of Attainment). A unique field is identified and set as the primary key for each table. A table can only have one primary key and this field should never be left blank. A primary key is used to ensure that every record is different from every other record. This is to avoid records being overwritten, or the wrong record being selected. Primary keys are used when linking tables together.

A **compound key** combines more than one field in order to make a unique key.

In some databases one field is not enough data to uniquely identify a record. In a pack of cards, there are four different suits, hearts, diamonds, spades and clubs. For example, in a pack one single primary key would not be able to identify each different card based on the suit, or on the card value. See Figure 1.1

Figure 1.1: Card example

Suit	Value	Number of times played
Hearts	Ace	5
Diamonds	1	4
Spades	3	3
Clubs	5	8
Clubs	Jack	5

However, if a primary key was made out of the **suit** and the **value** then this would be a unique field and could be used as a primary key.

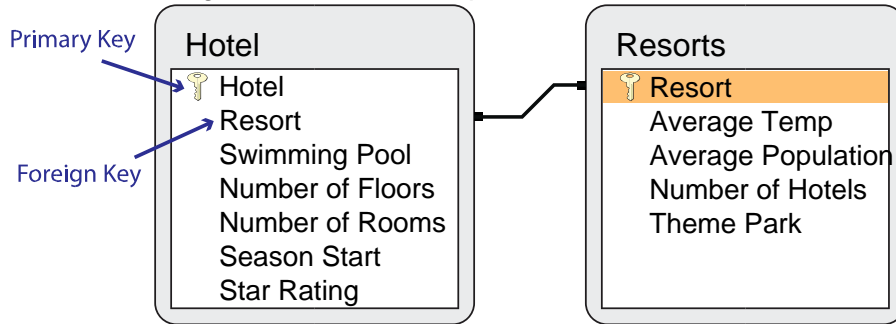
1.4 Relationships

Tables can be linked for different reasons. It may be simpler to have two smaller tables and link them than to have one very large table. Linking tables together may also avoid the unnecessary duplication of information.

In order for tables to be linked together they must each have a primary key.

Once a primary key has been set in one of the tables they can be linked together using the primary key. A primary key can be identified by the key symbol that appears next to it. This sets up a relationship between the two tables.

Figure 1.2: Relationship between two tables



Foreign Keys

A **foreign key** is a field in a table that is the primary key of another table. A foreign key can be used to cross reference information in tables. In Figure 1.2 you will see that Hotel is one primary key and Resort is another primary key. The foreign key is Resort as it also appears in the Hotel table. In a database package like MS access, when you view two tables that have been correctly linked together you will see that + signs appear next to the foreign key. See Figure 1.3.

Figure 1.3: Tables linked together, shown by + sign

	Resort ▼	Average Temp ▼	Average Population ▼	Number of Hotels ▼	Theme Park ▼
+	Amalafi	24	486000	365	<input type="checkbox"/>
+	Capri	22	250000	156	<input type="checkbox"/>
+	Florence	25	521000	450	<input type="checkbox"/>
+	Portofino	23	270000	225	<input checked="" type="checkbox"/>
+	Positano	24	300000	85	<input checked="" type="checkbox"/>
*					<input type="checkbox"/>

If you click on one of the + signs it will expand to show that particular resort information. See Figure 1.4.

Figure 1.4: Table expanded to show linked information

Resort	Average Temp	Average Population	Number of Hotels	Theme Park		
Amalafi	24	486000	365	<input type="checkbox"/>		
Hotel	Swimming Pool	Number of Floors	Number of Rooms	Season Start	Star Rating	Picture
Roma	<input checked="" type="checkbox"/>	3	456	01/05/2013	Five	
Capri	<input checked="" type="checkbox"/>	5	295	01/05/2013	Four	
*	<input type="checkbox"/>					
Capri	22	250000	156	<input type="checkbox"/>		
Florence	25	521000	450	<input type="checkbox"/>		
Portofino	23	270000	225	<input checked="" type="checkbox"/>		
Positano	24	300000	85	<input checked="" type="checkbox"/>		
*				<input type="checkbox"/>		

.....

Consider the following example. A car garage has one table to store the details of customers who bring their cars in, and another to store the details of the type of car and repairs carried out. The Customer table could have a primary key as the customer's phone number and the Car table could have a primary key as the car registration. The Customer table could then have the car registration field included in its table as a foreign key. When an employee accessed a car record they could use the car registration field to bring up details of the customer.

Figure 1.5: Car database linked to Customer database

Registration	Make	Model	Colour
P999 SYU	Toyota	Celica	Black
Phone Number	First Name	Surname	
07545369258	Kevin	Burns	
*			
SV07 NRZ	Nissan	Xtrail	Red
V096 DRZ	Jaguar	XKB	Red
*			

A **one-to-one** relationship is when there are two tables, both having the same number of rows. One row from table 1 relates directly to one row in table 2. This could be relevant if you have two tables in a database, one for customer names and one for customer details. See Figure 1.6.

Figure 1.6: One-to-one relationship

Customer name table	
customer id	Primary key, link to customer id of customer details table
lastname	
firstname	

Customer details table			
customer id	Primary key, link to customer id of customer details table		
height			
weight			
date of birth			
customer id	lastname	firstname	
0001	wilson	jordan	
0002	smith	jessica	
0003	black	kevin	
0004	gardiner	paul	
customer id	height	weight	date of birth
0001	183	75	09/11/1981
0002	156	83	01/12/1967
0003	179	72	21/01/1986
0004	167	93	17/03/1989

You will notice that each row is related to one row in the other table. This design relationship does not bring any benefits to the database. All the fields could be included in one table. However, if some of the fields, such as the height and weight were not regularly used it might make sense to keep the fields in a separate table so that the table is quicker to load. In a small database this would not have a big impact.

In a **one-to-many** relationship each row in-between table 1 can be related to many rows in table 2. This allows information that is regularly used to be saved only once in one table and referenced many times by other tables. It is saved once in one place but can be accessed many times, from different tables. See Figure 1.7.

Figure 1.7: One-to-many relationship

singer id	primary key
firstname	
lastname	

album id	primary key
title	
singer id	foreign key, link to singer table

singer id	firstname	lastname
0001	gary	barlow
0002	robbie	williams
0003	cheryl	cole
0004	dolly	parton

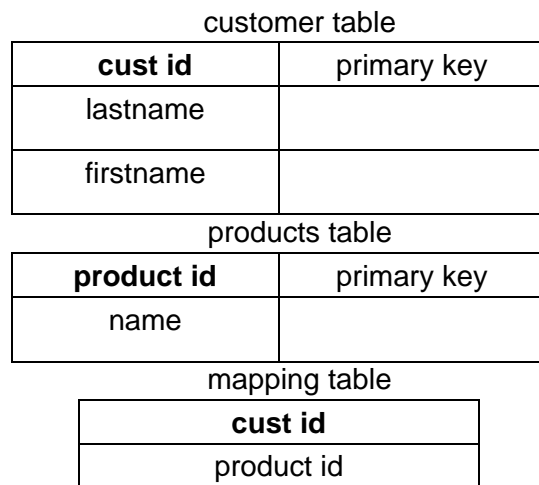
album id	title	singer id
01	Since I saw you last	0001
02	Open road	0001
03	Escapology	0002
04	Better day	0004

.....

Each row in the singer table can be related to many rows in the album table, hence to the name one-to-many. If you turn this relationship around the album table has a many-to-one relationship with the singer table.

Using a one-to-many relationship can reduce the amount of information that needs to be stored as duplicate information does not get stored. It can mean that there are two small tables, which are easier to manage than one large table. It requires a primary key in one table which is linked to a foreign key in the other one. In a **many-to-many** relationship one or more rows in a table can be related to one or more rows in another table. In a many-to-many relationship between tables 1 and 2, each row in table 1 is linked to one or more rows in table 2 and vice versa. A third table is required to implement this type of relationship and is called a mapping table. See Figure 1.8.

Figure 1.8: Many-to-many relationships



If we assume that there are only two customers and two products:

cust id	lastname	firstname
0001	smith	simon
0002	brown	james

product id	name
0001	savings
0002	credit card

cust id	product id
0001	0001
0001	0002
0002	0002

You will notice from the mapping table that Simon Smith has 2 products with the bank, a saving account and a credit card. You can see that both customers own credit cards.

The way that the database has been designed means that customers can have more than one product and products can have more than one customer.

1.5 Data, Entities and Relationships

A data **entity** is something that information will be recorded about in a database. This could be a person, an object, or something abstract like a holiday booking.

Examples of entities about *persons* include:

- a customer
- airline passenger
- a library borrower
- an employee
- a school pupil
- a college student.

Example of *objects* include:

- a product
- a flight
- a book
- a job
- a course

Examples of *abstract entities* include:

- a borrowing,
- a booking,
- a rental.

An **entity set** is a collection of entities of the same type and is represented as a table in a database system.

Each entity has a set of **attributes** which describe examples or **instances** of that entity.

For example, the **attributes** of a school pupil might include the following: forename, surname, address, date of birth, year group, tutor group.

An **instance** of a school pupil might be Joanna Hamilton, whose attributes are: Joanna, Hamilton, 10 Main Road, Kinglass, 01/04/1990, 5, SW.

The **attributes** of a 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.

Attributes can be single-valued or multi-valued. Multi-valued attributes are where there are more than 1 possible entries in a field for a given primary key. As you will see later, the relational database model requires attributes to be single-valued.

Attributes will have constraints on what is permitted. For example, an attribute representing a person's name will contain characters only. This is called the **domain** of an attribute and will define the type and value of data that the attribute can hold.

Examples of data types that attribute may use are:

- **Text** - e.g. a person's name
- **Integer** - e.g. the number of books borrowed from a library
- **Real** - e.g. the conversion rate for foreign currency
- **Object** - a photographic image
- **Boolean** - a True or False value
- **Date** - someone's date-of-birth
- **Time** - the departure of a train

1.5.1 The Relational Model

The relational data model was developed in the 1970s and is currently the most common database model. Example database management systems based on this model include Oracle, Ingres, IBM DB2, mySQL and Microsoft Access.

In the relational model, data is stored as records in **tables** and is interrogated using the Structured Query Language (SQL).

Before a designer can create a database using any model, an understanding of the requirements of the system is necessary. This analysis of the data will lead to the choice of tables, identification of **keys**, and an understanding of the requirements of users of the database. At this early stage of the design process the system is considered in terms of concepts and ideas to produce an abstract model of the required system.

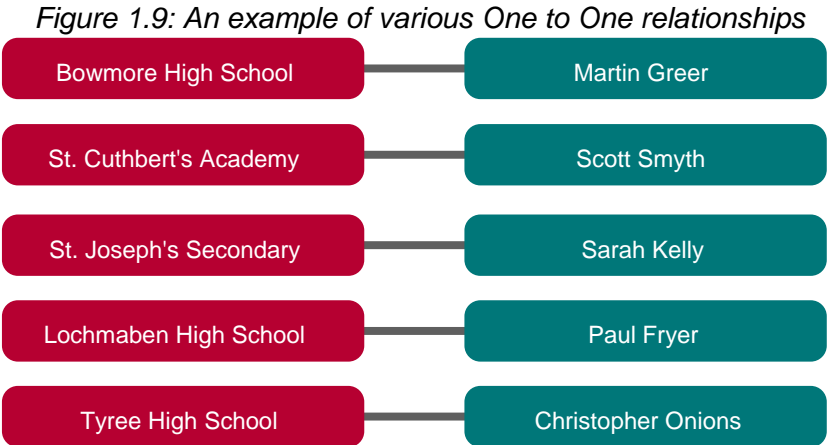
In the relational model, this design makes use of **Entity-Relationship** (E-R) modelling. This modelling is independent of the choice of relational database used to implement the database so it does not matter at this stage whether the system is to be implemented on a large scale using something like Oracle or Microsoft SQL server, or on a small scale using a system such as mySQL, Microsoft Access or Filemaker.

The modelling of the requirements is separate from the final implementation so that important design decisions can be taken independently of a particular product.

1.5.2 Relationships

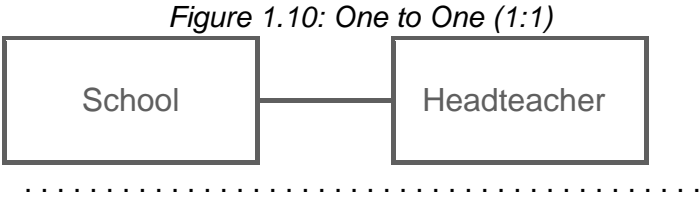
Entities can be associated with each other by relationships. The number of entities that another entity relates to is known as the mapping **cardinality**, or the degree of the relationship, and can be one of the following:

- **One-to-one** An entity is related to at most one other entity, written as a **1:1** relationship
- **One-to-many** An entity is related to more than one other entity, written as a **1:M** relationship
- **Many-to-one** A number of entities are related to one entity, written as a **M:1** relationship
- **Many-to-many** Many entities relate to many other entities, written as a **M:M** relationship.



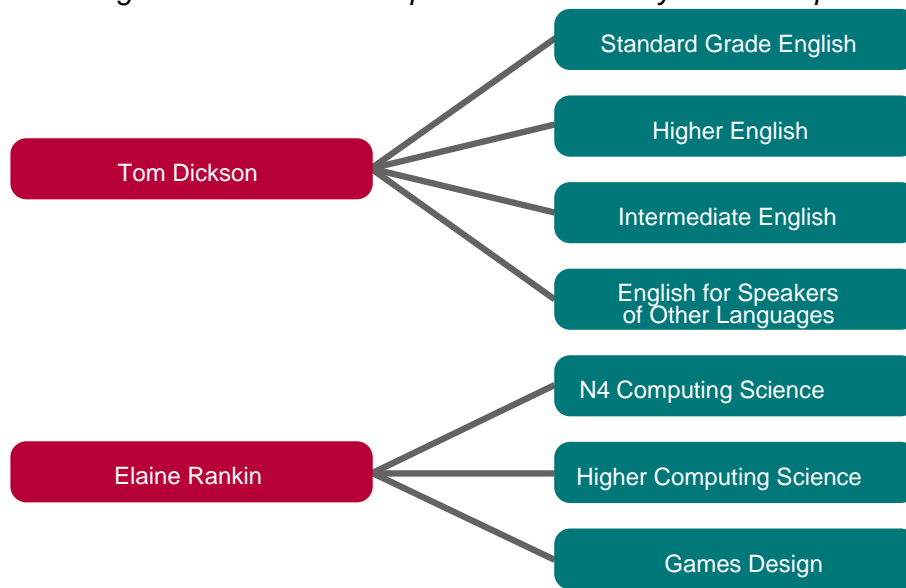
Each School has one Head teacher and each head teacher has one school.

This can be represented in an ERD (Entity - Relationship Diagram) as 1:1



One to Many Examples

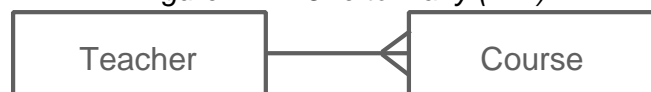
Figure 1.11: A few examples of One to Many relationships



.....

Here a teacher can teach many courses and each course is taught by one teacher. This can be represented in an ERD as a one to many relationship

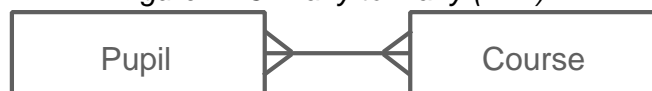
Figure 1.12: One to Many (1:M)



Many to Many Example

On the online course you can see an interactive example of a Many-to-Many relationship. This can be represented in an ERD as a many-to-many relationship.

Figure 1.13: Many to Many (M:N)



1.5.3 Keys

It is important to be able to distinguish between different entities. This is achieved by using one or more attributes of that entity as the key that identifies an entity. A **key** is a set of attributes that uniquely identifies an entity.

For example, the *order number* attribute of an order entity is a unique identifier thus *order number* is a key.

When looking for the key to an entity, there may be more than one attribute or combination of attributes that could uniquely identify the entities. These attributes under consideration are referred to as candidate keys. Once a key is chosen from the candidate keys as the principal identifier for an entity set, this becomes the **primary key**.

1.5.4 Data Dictionary

Once a data model has been completed, it is necessary to design the database. This involves deciding on the properties of each field. The structure of tables and the properties of fields are usually represented in the form of a **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 (**domain constraints**). *The domain of an attribute is the set of permitted values for that attribute. E.g. A name must only contain letters, Age must be between 17-25.* The data dictionary is important because this information can then be used to create a database.

Properties of fields

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

- The field 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 Memo** e.g. Smith, EH991AB, £13+VAT. Holds letters and numbers, can't be used in a calculation. Stores up to 255 characters. Stores text up to 65,536 characters.
 - Numeric** Either **integer** (whole numbers) or **real/double** (decimal numbers) 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 (short) 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. True or False										
hyperlink	A reference to a file located outside the database.										
OLE Object	Data such as a picture or sound file.										
• <u>The key</u>	Whether the field is a primary key (PK), part of a compound key, or a foreign key (FK).										
• <u>Validation</u>	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.										
• <u>Field size</u>	<i>Allows the user to restrict the number of characters in a field. E.g. First name size = 30 characters.</i>										
• <u>Formatting</u>	Some fields may require input to follow a certain pattern e.g. a postcode G73 7HT. In input mask can be used to encourage a particular format. The postcode input mask >La09 0LL the symbols have the following meaning. <table border="0"> <tr> <td>0</td> <td>Digit (0 through 9, entry required; plus [+] and minus [-] signs not allowed).</td> </tr> <tr> <td>9</td> <td>Digit or space (entry not required; plus and minus signs not allowed).</td> </tr> <tr> <td>L</td> <td>Letter (A through Z, entry required).</td> </tr> <tr> <td>></td> <td>Causes all characters that follow to be converted to uppercase.</td> </tr> <tr> <td>a</td> <td>Letter or digit (entry optional).</td> </tr> </table>	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.	a	Letter or digit (entry optional).
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.										
a	Letter or digit (entry optional).										

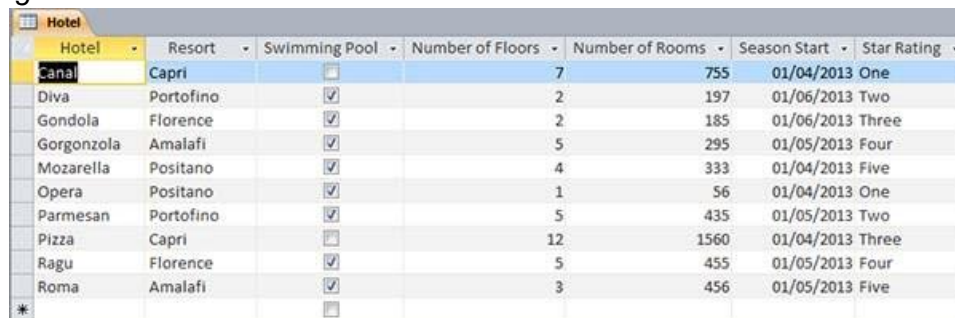
Data Dictionary DVD Rental Example

Entity	Attribute	Key	Data Type	Required	Unique	Format	Validation
DVD	DVD Code	PK	Integer	Y	Y		
	Film Code	FK	Integer	Y	N		Lookup value from FILM table
	Cost		Currency	Y	N		Range Check >=1 and <=3
MEMBER							
MEMBER	Member Number	PK	Integer	Y	Y		
	Title		Text	Y	N		Restricted Choice Mr/Mrs/Miss/Ms/Rev/Dr/Prof
	Forename		Text (15)	Y	N		
	Surname		Text (20)	Y	N		
	Address 1		Text (20)	Y	N		
	Address 2		Text (20)	N	N		
	Address 3		Text (20)	N	N		
	Post Code		Text (8)	Y	N		
	Telephone Number		Text (14)	Y	N		
	E-mail		Text (30)	N	N		
	FILM						
FILM	Film Code	PK	Integer	Y	Y		
	Title		Text (30)	Y	N		
LOAN							
LOAN	Member Number	PK	Integer	Y	N		Lookup value from MEMBER table
	DVD Code	PK	Integer	Y	N		Lookup value from DVD table
	Date Hired	PK	Date	Y	N	Short Date	
	Date Due		Date	Y	N	Short Date	

1.6 Complex data operations

A **user interface** is what the user sees when they are using a program or application. When creating a database there are different interfaces that can be created, depending on the purpose of the database. For a very simple database the user may use a simple table, the advantages of a table are that if there is not a huge amount of data in the database then it can all be displayed on the one screen. Tables are usually used for entering and editing data.

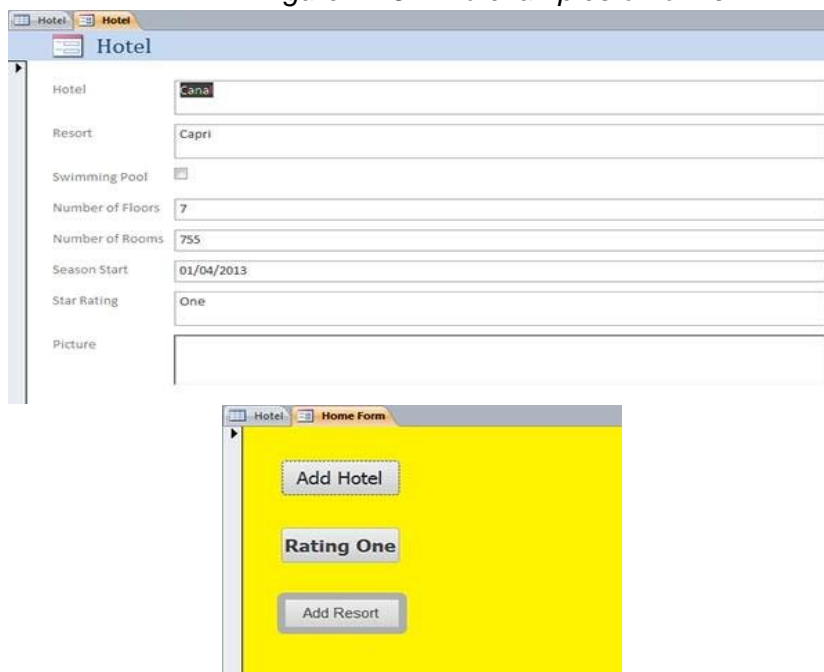
Figure 1.14: A table



Hotel	Resort	Swimming Pool	Number of Floors	Number of Rooms	Season Start	Star Rating
Canal	Capri	<input type="checkbox"/>	7	755	01/04/2013	One
Diva	Portofino	<input type="checkbox"/>	2	197	01/06/2013	Two
Gondola	Florence	<input checked="" type="checkbox"/>	2	185	01/06/2013	Three
Gorgonzola	Amalafi	<input checked="" type="checkbox"/>	5	295	01/05/2013	Four
Mozarella	Positano	<input checked="" type="checkbox"/>	4	333	01/04/2013	Five
Opera	Positano	<input checked="" type="checkbox"/>	1	56	01/04/2013	One
Parmesan	Portofino	<input checked="" type="checkbox"/>	5	435	01/05/2013	Two
Pizza	Capri	<input type="checkbox"/>	12	1560	01/04/2013	Three
Ragu	Florence	<input checked="" type="checkbox"/>	5	455	01/05/2013	Four
Roma	Amalafi	<input checked="" type="checkbox"/>	3	456	01/05/2013	Five
*		<input type="checkbox"/>				

A second alternative is to create a **form**. A form has the advantage that it is graphical and may look more professional, or interesting. A form might be easier for novice users to enter information. The drawback to forms is that they can only display one record at a time. Forms are used to provide a 'front end' to the user. They can be used to add records to a database, or to display individual records. They can have buttons, which run different queries or searches when pressed. Forms are useful to help avoiding mistakes when transferring data from paper forms which have been filled in; the database form can be laid out in the same way as the paper one.

Figure 1.15: Two examples of forms



The figure shows two examples of database forms. The top form is a data entry form with the following fields: Hotel (text input), Resort (text input), Swimming Pool (checkbox), Number of Floors (text input), Number of Rooms (text input), Season Start (text input), Star Rating (text input), and Picture (text input). The bottom form is a yellow 'Home Form' with three buttons: 'Add Hotel', 'Rating One', and 'Add Resort'.

The usability of forms can be increased by using drop down menus to allow users to select from a choice of options, or using buttons that allow the users to enter yes or no.

A third option could be to create a **report**. Reports are an excellent way of arranging the data in an easy to read format. Reports can show some, or all of the information in a database and you can have more than one record showing at a time. Reports can also include graphics to make them more appealing to users. Reports are most commonly used to generate a printable version of a database table or query.

Figure 1.16: A report

Title	Director	Certificate	Rating	Library Ref	Cost
A Nightmare on Elm Street	Wes Craven	18	5	14	£2.99
Conair	Simon West	18	5	9	£2.99
Dirty Dancing	Emile Ardolino	15	5	5	£5.99
Evan Almighty	Tom Shadyac	PG	3	2	£11.99
Fantastic Four	Tim Story	PG	4	3	£9.99
Harry Potter: Goblet of Fire	Mike Newell	12	5	4	£9.99
Harry Potter: Order of the Phoenix	David Yates	PG	3	8	£9.99
In Her Shoes	Curtis Hanson	PG	5	10	£4.99
Jay and Silent Bob Strike Back	Kevin Smith	18	5	13	£1.99
Michael Myers (Halloween)	Fred Walton	18	5	11	£4.99
Moulin Rouge	Baz Luhrmann	15	4	6	£3.99
Romeo and Juliet	Baz Luhrmann	15	4	15	£2.50
Spiderman 3	Sam Raimi	12	4	7	£15.99
Surf's Up	Ash Brannon	PG	4	1	£4.99
The Simpsons Movie	David Silverman	PG	5	12	£9.99
					£103.36

Page 1 of 1

A **query** is when you extract information from a database using searching and sorting tools or language built into the database management software. Queries often involve searching for specific items and/or sorting the resultant data into order. In the example above, a query could be used to compile a list of all the films rated PG, only showing the title and rating in order of title. In a hotel booking system, a query could be used to find a list of all today's expected 'check-ins' for the staff. Queries are ultimately what makes a database useful: users can get access to the specific information they need quickly and have it in a useful layout and order for their purposes.

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:

Total Mark: [Test 1] + [Test 2] + [Test 3]

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

Other calculations are **vertical** calculations otherwise known as Summary fields. 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 =Average([Test 1]).

	Name	Test 1	Test 2	Test 3	Total Mark	
record 1	J Bloggs	8	9	10	→ 27	<i>Calculated field</i>
record 2	J Public	6	7	8	→ 21	
	Average	↓ 7	↓ 8	↓ 9		

Summary field (usually in the report footer)

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). This saves storage space as the derived values do not need to be stored.

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

Report Header	Text/data to appear at the head of the report
Page Header	Text/data to appear at the top of each page of the report
Main Detail Header	Text/data to appear above each entry in the main detail section
Main Detail Section	Data from selected records in a table or query
Main Detail Footer	Text/data to appear below each entry in the main detail section (this may use a summary function)
Page Footer	Text/data to appear at the foot of each page of the report
Report Footer	Text/data to appear at the bottom of the report. Can include a summary field. E.g. =sum([Total DVDs rented to date])

DVD Rental Statistics		
Page 1		
<u>Details for J Bloggs</u>		
<u>Memb No</u>	<u>Address</u>	<u>Tel No</u>
142312	Main Street	123456
Total DVDs rented to date: 26		
<i>End of page 1</i>		
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.

Example using the sales field.

Category	Description	Example of functions using the field below
Sum	Used to total a field.	=SUM([Sales]) answer =
Count	Used to count the number of occurrences.	=COUNT([Sales]) answer =
Avg	Used to work out the average of a field.	=AVERAGE([sales]) answer =
Max	Used to work out the maximum value in a field.	=MAX([sales]) answer=
Min	Used to work out the minimum value in a field.	=MIN([sales]) answer=

Sales
£100
£200
£400
£500

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.

Member : Report				
Report Header				
<i>Member Rentals</i>				
Page Header				
Number	Name	Address	Post Code	Telephone N
Member Number Header				
Member	Name	Address	Post Code	Telephone N
Detail				
Date Hired	Title	Cost		
Member Number Footer				
<i>Total Cost of Rentals:</i>		=Sum([Cost])		

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.

Member Number Footer	
<i>Total Cost of Rentals:</i>	=Sum([Cost])
<i>Number of Rentals:</i>	=Count([Title])
Page Footer	
<i>Cost of Rentals for this page:</i>	=Sum([Cost])
<i>Number of Rentals for this page:</i>	=Count([Title])
Report Footer	
<i>Grand Total Cost of Rentals:</i>	=Sum([Cost])
<i>Total Number of Rentals:</i>	=Count([Title])

For example, the figure above 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.

1.6 How to answer exam questions about Complex Data Operations

Simple Example – Technology Company

Manufacturer	Item	Colour	Available	Price
Spingfast	Phones	Blue	Yes	£112
Alcafuz	Tablets	Black	Yes	£156
Mac Prods	Consoles	Pink	No	£342
iFuture	Phones	Green	No	£444
Stewson	Smart Watch	Black	Yes	£522

EXAMPLE QUESTION:

How could you find out what manufacturer sells phones for under £150?

EXAMPLE ANSWER:

Create a query with the following fields.

- Manufacturer
- Item
- Price and criteria <150

Multiple Table Query Example

Dog Walkers is a company that walks dogs when their owners are at work. The company has a database to store details of the dogs, their owners and the walkers. The data is stored in the following tables.

Dog	Owner	Walk	Walker
<u>Dog ID</u>	<u>Owner ID</u>	<u>Walk ID*</u>	<u>Walker ID</u>
Dog name	Owner name	Dog ID*	Walker name
Dog type	Owner address	Walker ID*	Walker phone number
Gender	Owner phone	No. Of days per week	
Walks well with others		Cost	
Photo			
Owner ID*			

EXAMPLE QUESTION:

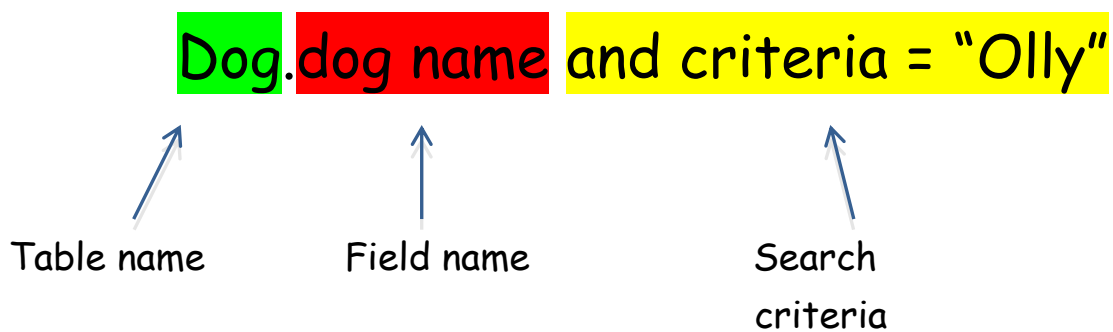
Describe how you would find the dogs called "Olly", dog type and gender, the walker name and walker phone number. Put the data into alphabetical order of dog name.

EXAMPLE ANSWER:

Create a query with the following fields.

- Dog.dog name and criteria = "Olly" and order = ascending
- Dog.dog type
- Dog.gender
- Walker.walker name
- Walker.walker phone number

The above solution shows you **how to set out an answer in an exam** if you are asked to explain how a query was performed.



Multiple Table Query Exam Question Structure

Try to mention some of the following points in your answer.	Example of your answer
<ul style="list-style-type: none"> • Mention that you are creating a query. 	<p><i>You should start by creating a query.</i></p>
<ul style="list-style-type: none"> • Mention the field names and which tables the fields will come from to create the query. 	<p><i>Customer. Name Customer. Second name Customer. Address Order.Order Number</i></p>
<ul style="list-style-type: none"> • Mention what field names have been used in a search (if any). <ul style="list-style-type: none"> ○ Mention the criteria for a search (e.g. >=20) 	<p><i>Customer.Second name and criteria = Reilly</i></p>
<ul style="list-style-type: none"> • Mention the field names that have been used in sort (if any). <ul style="list-style-type: none"> ○ Mention what order the sort may be in (e.g ascending/descending) 	<p><i>Customer.Second name sort = ascending</i></p>
<ul style="list-style-type: none"> • Mention if a calculated field has been used. <ul style="list-style-type: none"> ○ Mention what formula may have been used to create the computed field. 	<p><i>Calculated Field E.g.Total:[weekly payment] * [no of weeks]</i></p>
<ul style="list-style-type: none"> • Mention if any summary functions have been used in the report (e.g. Sum, Average, Count, Min, Max) and which fields they have been used with. 	<p><i>Use the COUNT summary feature with the account number field and put the formula into the report footer. E.g. =COUNT([account number])</i></p>

