Attributes

The attributes of an entity are the items of data that belong to that entity and that we need to store for the purposes of the database. The attributes of STUDENT as seen from the table (Figure 2) on page 2 are Name,YearGrp, Set, Tutor Group, Tutor and Grades.

The attributes of an entity is a list of the data items that we need to store for the entity class. It must include all the data that belongs to the entity and that is also needed for the purposes of the database. If we needed to analyse student performance by gender then sex would be needed as an additional attribute of STUDENT.

The actual list of attributes for a given entity will depend on the entity itself and the information that the database must provide. It would be quite possible for two different database applications to contain the same entities but to have different attributes associated with them.

If you are familiar with flat-file databases then the entities can be thought of as corresponding to records and their attributes to fields.

There are two commonly used ways of showing the attributes of an entity. The first way is to write the list of attributes in brackets after the entity name. The entity STUDENT could then be written:

STUDENT (Name,YearGrp, Set, TutorGroup, Tutor, Grades)

An alternative method is to show the information in diagram form as shown in Figure 3.

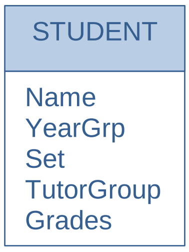


Figure 3: Student Entity

Relationships

Normally a database will store data about more than one entity. In the library database, for example, we had BOOK, MEMBER, LOAN and FINE. We can identify relationships between these entities. A relationship, if it exists, will be one of three types.

One-to-One Many Relationships

A one-to-many relationship exists when it is possible that one instance of entity A might be linked to many instances of entity B but, when the view is reversed, one instance of B links to only one instance of A. This is easier to understand when applied to some actual entities.

In the library system, one particular member may have paid many fines. One particular fine will have been paid by one particular member. The relationship between MEMBER and FINE is therefore a one-to-many relationship. It would be shown in an entity-relationship diagram in the following way.

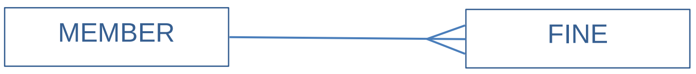


Figure 4: Representing a One-to-Many Relationship

The 'crow's foot' lines indicate the many side of the relationship. An alternative method of showing the many side of the relationship is to use the symbol for infinity (∞) to indicate the many side of the relationship and a 1 to indicate the one side. In this case the E-R diagram would look like this:

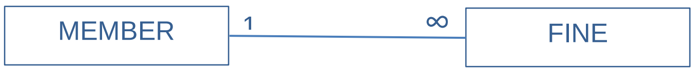


Figure 5: Alternative Representation of One- to-Many

Chapter 4: Tables and First Normal Form

Representing Entities

We are now almost at a stage where our entities can be represented by tables that can be directly implemented in a relational database. The table used to represent an entity should obey the following rules.

(1) The table must have a unique name. Usually this will be the plural of the corresponding entity, so that the entity STUDENT is represented by the table tblStudents. The *tbl* is added to the front of the name to indicate that this is a table rather than any other type of database object such as a query or a form.

(2) The table must be made up of columns and rows. Each column must have a unique name and the order of the columns must be unimportant. The column names will be the attributes that we have identified for the entity

(3) No two rows can be identical. We have ensured this by selecting one or more attributes to be an entity identifier. In the table we will call this the primary key.

(4) The order of the rows must be unimportant and each row must stores data about one instance of the entity.

(5) Each cell in the table must contain a single indivisible data value. We have not addressed this requirement yet. It will mean that we have to remove lists of data – such as the list of days that a doctor attends a particular surgery – from our entities.

The set of rules above come from mathematics and defines a mathematical concept called a relation. The correct name for our table is in fact relation. We say that we implement the entities in a relation. It is this that gives the relational database its name – because we use relations to represent the entities. You probably thought up to now (quite naturally) that we called the database relational because of the relationships between the entities. But it is the mathematical relation, with its rows and columns and strict mathematical definition that gives this type of database its name.

Tables, Access and Relational Databases

As we start to represent the entities of the data model in the form of relations or tables, our design begins to match up with what will happen at implementation. Each entity will be represented as a table in the database. The attributes will be represented as columns and the combination of attributes that define an instance of the entity will be represented by a row.

Figure 12 on page 18 shows the student data from the table on page 2 converted to an Access table. This table does not comply with the definition of a relation. There is no guarantee that each row will be unique which breaks rule 3 above. Also the values in the Grades column can be broken down into a number of separate values, which breaks rule 5.

Chapter 6 Third Normal Form

Third Normal Form

Third Normal Form	A table is in third normal form if it is in second normal form and there is no functional dependency between non-key items.

Third normal form requires that each of the purely informational or non-key item columns of the table is independent of any other. Put another way, informational items must be dependent on the primary key and on nothing else.

There are two ways in which one informational item can be dependent on another. Firstly if one column value is calculated from another then there is obviously a functional dependency. Secondly we may have a situation where there is a non-calculated dependency between two items.

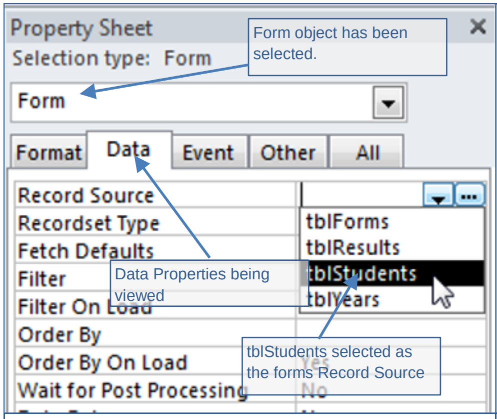
Calculated Dependency

Suppose we had an entity CUSTOMER (AccountNo, Name, Address, DOB, Age). A table representing this entity would be in 2NF since Name, Address, Date of Birth and Age are functionally dependent on AccountNo. (Assuming that Address is atomic for the purposes of the application).

Age can be calculated from DOB and there is therefore a calculated dependency between them. A calculated dependency is removed by removing the calculated value. It is not usually necessary to store a calculated value since it can be calculated from the underlying data as and when required.

There are some situations where we would want to store a calculated value. We may need to keep the original calculated value when the underlying value changes – perhaps for audit purposes. Another case where we might want to store calculated values is when a large number of calculations have to be done in order to display information. This might degrade database performance and so we might decide that it is better to store the calculated values in defiance of 3NF.

This raises an important issue. Normalisation is a guide to developing a database structure that results in minimum data duplication and maximum data independence. There is no point in achieving these goals if the resulting structure does not fulfil the needs of the end user. There will be situations where the best design strategy will be to have tables that are not fully normalised. Database design is therefore not an exact mathematical process – it relies on the designer's experience to know when to break the rules.



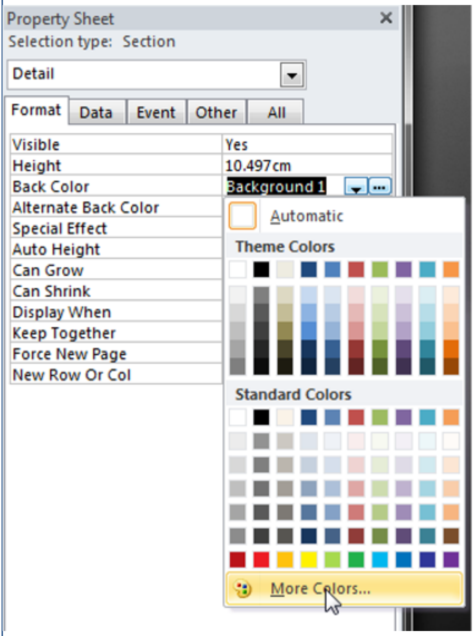
This form will edit data stored in tblStudents, so our first task is to link the form to the table.

Make sure the form is selected, either by choosing it in the selection type combo box at the top of the property sheet or by clicking the small square button at the left-hand end of the horizontal ruler on the form itself.

Click the *Data* tab on the Properties sheet and click on the empty space on *Record Source* property line.

There are two buttons at the end of the Record Source line. One presents a drop down list and the other (the three dots) allows you to design a query as the form's record source.

This form will be based on a table, so click the drop-down button and select tblStudents from the list.

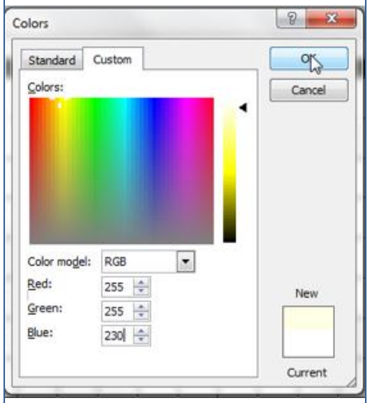


Now we are going to create a custom colour and apply it as the background colour for the detail section of the form.

Click anywhere on the detail section to select it. Alternately you can select *Detail* in the drop-down list at the top of the properties sheet.

Click the *Format* tab on the properties sheet so that the format properties are displayed.

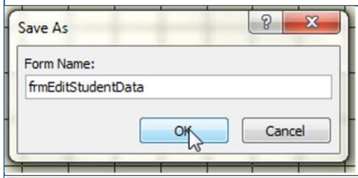
Click on the *Back Color* property and click the button with three dots on it. When the colour chart appears, click *More Colors* at the bottom of the chart.



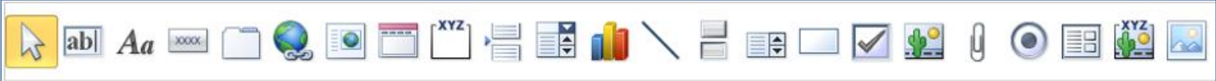
Click the *Custom* tab on the dialogue box and set a custom colour with the properties:

Red	255
Green	255
Blue	230

And then click *OK* to apply this custom colour as the background to your form detail.



Before going on with the form design, save the form, giving it the name *frmEditStudentData* as shown on the design sheet. Make sure that you follow the naming convention by making sure that there are **NO SPACES** in the name.



The picture above shows the *Design* tab of the *Form Design* ribbon.as it appears in *Form Design View*. A similar ribbon, but with fewer tools available, appears when *Layout View* is chosen for the form. This ribbon is not displayed when *Form View* is selected to display the form. Some of the more commonly used tools are shown below. You can find more details about different types of control in Table 7 on pages 71 - 73

Aa	The label tool places a label on the form. A label is used to display text that does not normally change. When other controls are added to the form a label control is automatically added as well and linked to the new control.	ab|	The text box tool is used to place a text box on the form. A text box can be used to display data on the form and also to allow the user to edit existing or add new data.
↖	The pointer tool is used to select individual objects on the form or groups of objects within rectangular areas of the form.	XXXX	The command button control is most commonly used to provide a link to macros or code.
	The combo box tool places a combo box on the form. This is a drop-down list that can be used for data entry.		The list box tool places a list box on the form. The list will be scrollable if there are more items than can fit in the space occupied by the control.

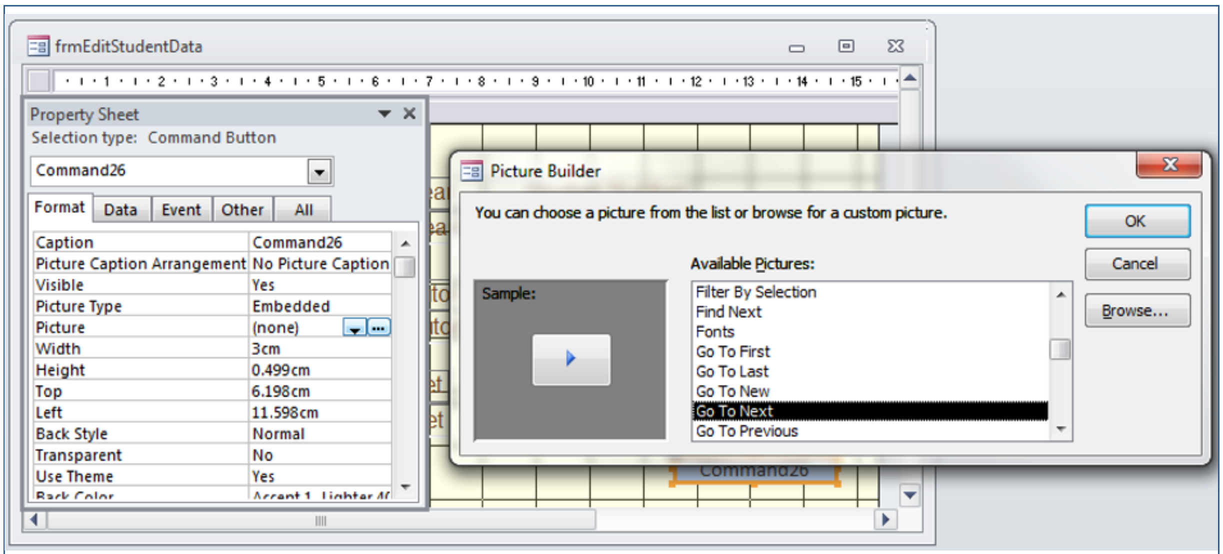
acNext defaults to a move of one record forward then we do not need the value 1 at the end. The method call then simplifies to:

DoCmd.GoToRecord , , acNext

Notice that the two commas are needed before acNext since otherwise Access has no way of knowing that this is the third value. If optional values are missed out on the left of a value that is supplied then their position in the list must be marked with a comma.

In fact, since acNext is the default value for this method, we could leave it out too. However it is then no longer obvious what the code is doing, so it is better to leave the acNext in place.

Practical 11.04



frmEditStudentData

Property Sheet
Selection type: Command Button

Command26

Format | Data | Event | Other | All

Caption	Command26
Picture Caption Arrangement	No Picture Caption
Visible	Yes
Picture Type	Embedded
Picture	(none)
Width	3cm
Height	0.499cm
Top	6.198cm
Left	11.598cm
Back Style	Normal
Transparent	No
Use Theme	Yes
Back Color	Accent 1, Lighter 40

Picture Builder

You can choose a picture from the list or browse for a custom picture.

Sample:

Available Pictures:
Filter By Selection
Find Next
Fonts
Go To First
Go To Last
Go To New
Go To Next
Go To Previous

In this practical work you will add four navigation buttons to move forward and backwards through the list, to jump to the first and last records.

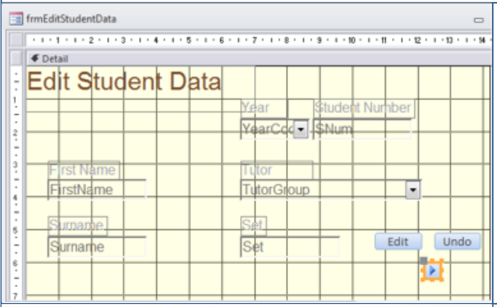
Use the copy of the database that you had at the end of practical 11.03. If this is not available then use the database file db11P04.accdb, which you will find in the chapter 11 folder.

Open frmEditStudentData in design view and, with the Control Wizard turned off, create a command button. Set its *Name* property to cmdNext.

Select the Format tab in the properties window for the button. Ignore the caption property and place your cursor on the *Picture* property line.

Click on the three dots at the end of the line to start up the Picture Builder.

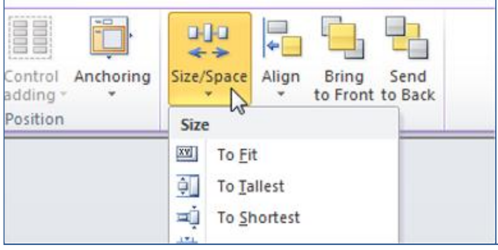
Select *Go To Next* from the list of available picture since this is the icon that the user will most easily recognise as moving on to the next record.



Select the *Event* tab for the properties window and use the Code Builder to enter the following code for the *Click* event of the button.

```
DoCmd.GoToRecord , , acNext
```

Finally resize the button so that it is the same height as the other two buttons on your form and position roughly as shown.



A simple way to make two controls the same height is to select both of them and then use the Size/Space control to make them equal sizes using the *To Tallest* or *To Shortest* option.

The tool can be found in the *Sizing and Ordering* group on the design *ribbon's* Arrange tab.

Customising the Access Environment

There are many things that an inexperienced user might accidently or innocently do within the database that would corrupt data or damage the database to the point where it would not work. For example, an action query could be run directly from the Navigation Pane and might cause changes or deletions that should be performed only under specific conditions.

In this section, you will learn how to hide the both the Navigation Pane and the ribbon from the user. The methods used do not make your system secure since, if the database is opened while the SHIFT key is held down, the customisation will be ignored and Access will open with its normal settings.

Hiding the Ribbon

The ribbon can be customised easily using *Customize Ribbon* section of the *Options* from backstage view. This allows you to specify which tabs and tools will display. Although this is easy to use, the changes made apply to Access in general, not just the file you are working on. In many environments you would not want to make changes that affect all Access files.

If we want to change the ribbon for one file only then we need to create a custom ribbon. Custom ribbons are defined using XML code so we will restrict ourselves to the simplest case of defining a custom ribbon that is empty. Using this ribbon will mean that the user sees an empty ribbon, i.e. no ribbon at all.

The XML code for the ribbon must be stored in a special system table called *USysRibbons*, which we must create. Each entry in this table represents one custom ribbon that will be available to the database file that contains the table.

USysRibbons must have two fields. One called *RibbonName* and the other *RibbonXML*. These contain the ribbon's name and its defining code.

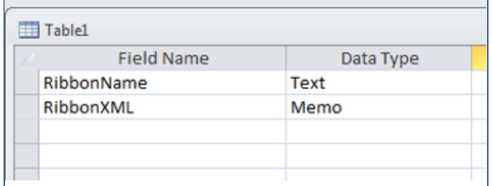
XML code is case-sensitive and it easy to make a mistake when entering it. A single mistake in the XML code will result in the default ribbon loading. Access will not normally report the error and it is easy to assume that the method being described just does not work when in fact an error in the code is preventing it from doing so.

Here is the code needed to define an empty ribbon.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
<ribbon startFromScratch="true"/>
</customUI>
```

Practical 17.05 will take you through the steps of setting up the ribbon definition table, entering the code and applying it to ensure that the empty ribbon is used when your Access file loads.

Practical 17.05 Hiding the Ribbon



Field Name	Data Type
RibbonName	Text
RibbonXML	Memo

Use either your own file from the previous practical or file 1db7P05.accdb.

Open you database file. The menu form will open automatically. Right click on the form and close it.

Use the Table Design tool on the Create tab to open a new table in design view. Add two fields.

RibbonName	(Data Type *Text*)
RibbonXML	(Data Type *Memo*)