

Extending the design of a Relational Database

Aims:

You should be able to:

- Describe in detail 1st, 2nd and 3rd normal form
- Independently develop a relational database
- Use a lookup wizard
- Compare and Contrast Flat File and Relational models
- Design, Implement and Develop a working user interface

More on Normalisation

It is worth reiterating that the coverage of the topic here represents a relatively informal overview. It is stressed that whilst relational databases are extremely popular and widespread, they are not the only type of database currently in fashion. Relational databases are particularly suited to applications where data is structured and, for example, are not necessarily the best tools for mass text retrieval.

When relational databases were first developed by Codd & Date in the mid 1970's there was a premium on disk storage space. This partly informed the development process. The process of normalising data results in many small (i.e. records with few fields) tables. This means that there is an overhead for each query processed because if the data is held across multiple tables then the DBMS has to JOIN those tables together to produce the RELATION. Such systems are particularly suited to applications which have high levels of transactions e.g. student record systems and banking systems.

The rapid growth in the power of computing hardware and the corresponding fall in the cost of computing hardware means that other software techniques are being developed which would not have been previously feasible. During the 1990's there has been a rapid and sustained growth in the production of OBJECT ORIENTATED systems. Students wishing to further their knowledge of database design are recommended to explore object orientated database techniques.



Notes

Throughout this exercise we will be asking what happens if we **delete**, **modify** and **add** data. If we discover anomalies we can suspect that our data design is in need of modification.

Applying the rules of first, second and third normalisation takes us through the following stages.

First Normalisation

This rule was designed to prohibit multivalued attributes (fields). Fields must not be allowed to contain a set of values. If we represent the student data in a slightly different form (Figure 7.1) we can see that the fields REGNO, MODEL and COLOUR are not single value fields (i.e. they are not ATOMIC or indivisible).

STUDREC

SREF	INIT	SNAME	DOB	GENDER	RES	KIDS	DISTANCE	HTOWN	REGNO	MODEL	COLOUR
1	TJ	OSMAN	1953	M	False	0	8	MGREEN	F 28 CWE C579 CUA N 609 UWJ TJO 1 TJO 2	HONDA CAVALIER CARINA FERRARI RANGE ROVER	WHITE WHITE MERCURY RED GREEN
...

Figure 7.1: Back to an un-normalised design

We need to **remove all repeating fields**. This can be done by normalising Figure 7.1; examine REGNO, MODEL and COLOUR. They contain sets of values (e.g. Model contains {HONDA, CAVALIER, MAZDA}, therefore, for this domain (record) the set MODEL > 1, therefore it is not atomic.

This can be remedied by redesigning Figure 7.1 to the schema shown in Figure 7.2.



Notes

STUDREC

SREF	INIT	SNAME	DOB	GENDER	RES	KIDS	DIST	HTOWN	REGNO	MODEL	COLOUR
1	TJ	OSMAN	1953	M	False	0	8	MGREEN	F 28 CWE	HONDA	WHITE
1	TJ	OSMAN	1953	M	False	0	8	MGREEN	C579 CUA	CAVALIER	WHITE
1	TJ	OSMAN	1953	M	False	0	8	MGREEN	N 609 UWJ	CARINA	MERCURY
1	TJ	OSMAN	1953	M	False	0	8	MGREEN	TJO 1	FERRARI	RED
1	TJ	OSMAN	1953	M	False	0	8	MGREEN	TJO 2	RANGE ROVER	GREEN

Figure 7.2: First Normal Form

The relation (table) in Figure 7.2 is now in first normal form (1NF). Notice that there is also REDUNDANCY as details such as SREF, SNAME, DOB etc. are stored more than once.

Let us consider what deficiencies there are with this design with respect to deleting, amending and adding data.

If we add a record, perhaps because T. OSMAN has bought yet another car, we duplicate data that is already held (INIT, SNAME, DOB, HTOWN etc.). We also run the risk of introducing internal inconsistencies into our database, it is possible for one record to have T. Osman's address as Millhouse Green, and another to have T. Osman's address as Thurlstone.

Similar problems exist if we amend records. T. Osman moves from Millhouse Green to Thurlstone, we would need to amend each and every record for T. Osman. This introduces uncertainty, we may well miss a record.

Suppose that T. Osman sells all of his cars and starts walking to work. We would delete a record for each car. What happens when the last car is sold? We would lose all records for T. Osman.

Data in first normal form is only a start.



Second Normal Form

This form is designed to ensure that all non primary attributes (non key fields) are fully functionally dependent on the primary key (key field). In this case we know that the key field for STUDREC is SREF. It is evident that INIT, SNAME, DOB, GENDER, RES and KIDS are fully functionally dependent upon SREF. However, we may well suspect that COLOUR and MODEL are not fully functionally dependent upon SREF. COLOUR and MODEL are in fact fully functionally dependent upon REGNO.

We can decompose the table (relation) STUDREC into two tables (relations) STUDREC and CARS as shown in Figure 7.3. This will be in second normal form because all non primary attributes (non-key fields) will be functionally dependant upon the primary attributes (key fields).

STUDREC

SREF	INIT	SNAME	DOB	GENDER	RES	KIDS	DISTANCE	HTOWN
------	------	-------	-----	--------	-----	------	----------	-------

CARS

REGNO	MODEL	COLOUR	OID
-------	-------	--------	-----

Figure 7.3: Second Normal Form

Notice that CARS now has another field (attribute) OID. This is so that each car's details can be cross referenced with its owners details. OID in the table CARS is a foreign key and matches values in the SREF field in STUDREC

There is a problem with this design. Consider how we may know if Millhouse Green is 8 miles from the college given our test data. What happens if, next year, we have no students from Millhouse Green? How would we know how far Millhouse Green is away from the college?

If the record for T. Osman were to be deleted then our only means of knowing how far Millhouse Green is away from the College also disappears. We surely can't depend upon having a student from Millhouse Green enrolled at the College in order to store Millhouse Green related details.

Note: at this stage of normalisation, STUDREC is in second normal form whilst CARS is in third normal form.



Third Normal Form

This is designed to remove any **transitive dependencies**. A relation is in third normal form if every non prime attribute of a relation is

fully functionally dependant upon the key attributes (SREF and REGNO in this example) and

non transitively dependant upon the key attributes.

The table CARS in Figure 7.3 is in third normal form (3NF), but the table STUDREC is not. This is because although the fields (attributes) INIT, SREF, SNAME, DOB, GENDER, RES, KIDS, DIST and HTOWN do depend upon SREF, the field (attribute) DISTANCE in turn depends upon HTOWN. This is a transitive dependency.

The table STUDREC will need to be decomposed into two further tables, STUDREC and TOWNS as shown in Figure 7.4. Both of these tables will now be in third normal form (3NF).

STUDREC

SREF	INIT	SNAME	DOB	GENDER	RES	KIDS	HTOWN
------	------	-------	-----	--------	-----	------	-------

TOWNS

HTOWN	DIST
-------	------

CARS

REGNO	MODEL	COLOUR	OID
-------	-------	--------	-----


Figure 7.4: Third Normal Form

This design will allow us to store data on various town related details independently of the fact that a student from that town may or may not be enrolled at the College at any point in time.



Amending the Student Records database into 3NF

So much for the theory, we now have to make it work in practice. We shall amend the underlying table structures in the Student Records database to comply with the design that we have just worked through. When we've done that, we'll create a query and a report based on the new structure.

 Ensure that the database window for the Student Records database is open

 Select the tables tag

 Click on the NEW button

 Select the Design View option

 Click on the OK button


 Create a table with the following fields

TNAME	text (width = 16)
DISTANCE	number (integer)

 Set TNAME as the key field

 Save the table as "TOWNS" (no quotation marks)

 Select the VIEW mode

 Enter the data shown in Figure 7.5



Notes

TNAME	DISTANCE
MILLHOUSE GREEN	9
HUDDERSFIELD	14
BARNSLEY	3
LEEDS	34
SHEFFIELD	14
SILKSTONE	1
THURLSTONE	7
SUNDERLAND	120
HENDON	128
ROTHERHAM	20
DONCASTER	26

Figure 7.5: TOWN data

- ➔ Add any other town details that you have included in your STUDREC table that are not listed here

Notice that we have stored details for the City of Sunderland. Our design allows us to do this independently of the fact that we may or may not have any students who come from Sunderland.




- ➔ Save the table TOWNS and close it.

We will now define the relations between the three tables (STUDREC, CARS and TOWNS).

- ➔ Click on the relationships icon in the toolbar
- ➔ Delete any tables that appear in the relationships window (highlight them and press the DELETE key)
- ➔ Click on the SHOW TABLE icon (this will display the SHOW TABLE window)
- ➔ Add the TOWNS table to the RELATIONSHIPS window








Notes



-  Next, add the **STUDREC** table to the **RELATIONSHIPS WINDOW**
-  Close the **SHOW TABLE** window
-  Drag the **TNAME** field in **TOWNS** and place it over the **HTOWN** field in **STUDREC** (this will establish a link between the two tables)

If you get an error message at this point the most likely cause is that you have got a town listed in **STUDREC** that isn't in **TOWNS**. This breaks the rules of relational integrity. If this happens, you should check the data in your tables and if you find you have town in **STUDREC** that isn't in **TOWNS**, edit **TOWNS** and add the missing town.



You should now see a further relationships window that will ask whether you wish to enforce referential integrity - we do, so

-  ensure that the appropriate check box is ticked.
-  Click on the **JOIN TYPE** button to access the **JOIN PROPERTIES** window
-  Ensure that the **JOIN TYPE** is option 1 (inner join) and press the **OK** button to return to the relationships window
-  Click on the **CREATE** button to return to the **RELATIONSHIPS** window
-  Close and save the relationships layout









We are now in a position to remove the **DISTANCE** field from **STUDREC**.

-  Open **STUDREC**
-  Move to **DESIGN VIEW**



-  Delete the **DISTANCE** field (you will get a warning message to the effect that if you delete this field all the data contained within it will permanently be lost - that is OK because we've got the distances safely stored in the **DISTANCE** field in the **TOWNS** table)
-  Save and quit from the **STUDREC** table

Just to prove that we have not lost any data we are going to create a query based on the relationship between **TOWNS** and **STUDREC**.

-  Ensure that the database window is open and select the **Queries** tag
-  Click on the **NEW** button
-  Select **DESIGN VIEW**
-  Add **TOWNS** followed by **STUDREC** from the **SHOW TABLE** window
-  Close the **SHOW TABLE** window
-  Add the fields **TNAME**, **DISTANCE**, **INIT** and **SNAME** to the query as shown in Figure 7.6
-  Save the query as "**DISTANCE**" (no quotation marks)
-  View the results of the query

You will find that each student's name is listed along with their home town and the distance of their home town from the College



Notes

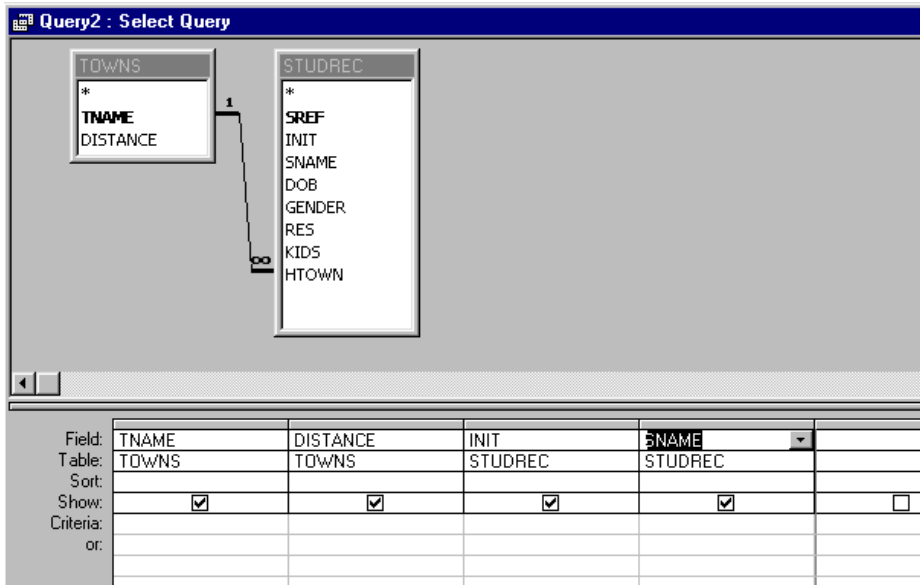





Figure 7.6: Defining the “distances” query

Lookup Fields

The rules of referential integrity demand that any value we now add in the HTOWN field of STUDREC has to be an existing value in TNAME in the table TOWNS. If we try to add a student from Newcastle, we will get an error message to the effect that we can't change or add a record because a related record is required in the table TOWNS. This is all very well, but it could get a bit frustrating if we keep trying to add students and keep forgetting which towns we have details on in the TOWNS table. Is there any way in which we could make things easier for the user?

We could define the data type for the HTOWN field in STUDREC to be a lookup field. A wizard will guide us through the process.

-  Open the STUDREC table in design view
-  Click in the DATA TYPE column of the HTOWN field
-  Select the LOOKUP WIZARD data type (this will display the lookup wizard window - Figure 7.7)



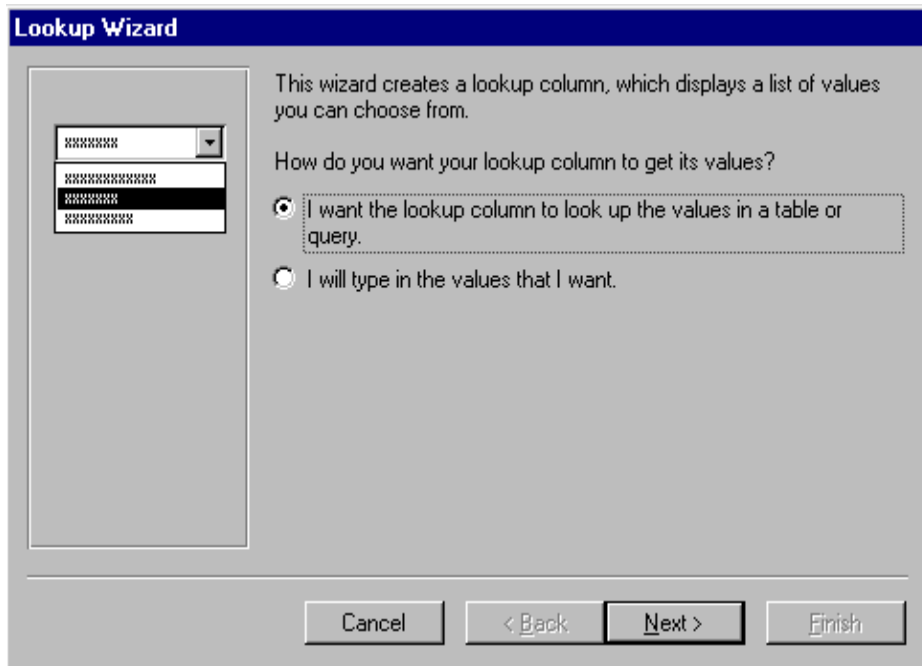


Figure 7.7: Lookup Wizard (1)

The choices we have here are whether to type in a limited set of permissible values for the user to choose from, or to get the values from another table or query.

We are going to choose to get our values from another table (TOWNS)..

- ☞ Select the “I want the look up column to look up the values in a table or query” option
- ☞ Click on the NEXT button
- ☞ Choose the TOWNS table
- ☞ Click on the NEXT button
- ☞ Select the TNAME field (Figure 7.8)



Notes

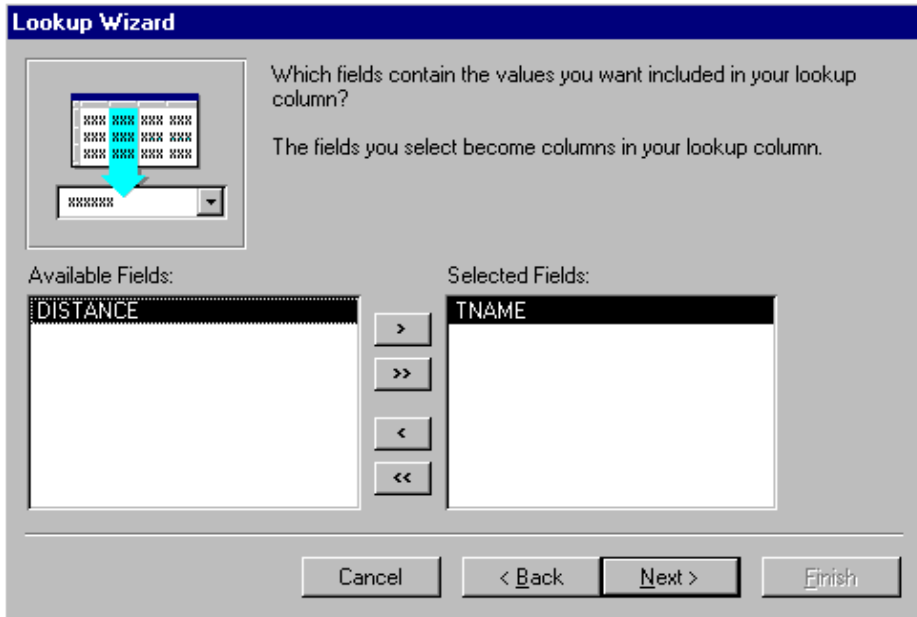


Figure 7.8: Lookup Wizard (Choose which field)

- ☞ Click on the NEXT button
- ☞ Adjust the width of the lookup column so that it displays the town names fully
- ☞ Click on the NEXT button
- ☞ Call the lookup column "Towns" (no quotation marks)
- ☞ Click on the FINISH button
- ☞ Save the changes
- ☞ Select the Datasheet View



Notes



Add a new record with the following details:

SREF = <the next number>
 INIT = J
 SNAME = MARSHALL
 DOB = 03/07/50
 GENDER = F
 RES = No
 KIDS = 2
 HTOWN = THURLSTONE

What happened when you tried to add the home town? You should have been able to choose from the lookup table (Figure 7.9). The lookup table is getting its values from the TNAME column in the TOWNS table.

	16	G	WHITE	03-Mar-65	M	Yes	3	BARNLEY
	17	J	GREEN	06-Aug-69	F	No	0	HENDON
	18	J	GREEN	09-Aug-45	M	No	0	HUDDERSFIELD
	19	F	WATSON	03-Mar-58	M	Yes	1	LEEDS
	20	L	HARVEY	03-Aug-54	F	No	2	MILLHOUSE GREEN
	21	T	MOSLEY	31-Oct-75	M	Yes	2	SHEFFIELD
	22	J	POWERS	30-Aug-45	M	Yes	0	SILKSTONE
	23	J	CHESTER	15-Mar-60	M	No	0	SUNDERLAND
	24	J	MARSHALL	03-Jul-50	F	No	2	THURLSTONE
*						No		

Figure 7.9: The HTOWN Look up table



Save the changes to STUDREC



Close STUDREC



Notes

Exercise 7.1: Creating Class lists

This exercise requires you to develop the Student Records database to be able to store lists of Classes that are taught. In addition, you will need to produce reports of the names of students attending each class (a class register).

The two tables shown here (Figure 7.10 and Figure 7.11) are your starting point.

CLASSREF	TITLE	SEMESTER	ROOM	DAY
1	Introduction to Database Design	1	1	Monday
2	Introduction to Database Design	1	1	Tuesday
3	Key Skills - Internet	1	1	Monday
4	Key Skills - Internet	1	2	Wednesday
5	Key Skills - Statistics	1	8	Wednesday
6	Computer Systems	2	3	Friday

Figure 7.10: The CLASSES table

The table CLASSES holds details of each class, the title, which semester it is taught in, which room it is in and which day of the week it is timetabled for (is this a look up field?).

Many students can and (hopefully) do attend many classes, each class has many students. We need another table. This table will be a sort of class register table (in any event, it will hold details of which students attend which classes).

The REGISTER table consists of two foreign keys, CLASSID and SREF. SREF cross references with SREF in the table STUDREC and CLASSID links to the CLASSREF field in the CLASSES table.

NOTE: it is possible to have two (or more) fields combined as a key field. If you do this in the REGISTER table with CLASSID and SREF the Access DBMS will prevent you from accidentally entering the same student on the class more than once. Both fields are selected by holding the SHIFT key down and then clicking on each field in turn (then click on the key field icon as usual).

Create the two tables CLASSES and REGISTER.

Enter the data shown in Figure 7.10 and Figure 7.11.



Set up the appropriate relationships. You have to list all classes whether or not they have any students in them, and you must only list students if they are members of a class.

Create a query called CLASS LISTS to list all classes and students in those classes

Create a report called CLASS LISTS that is similar to the one shown in the following pages.

Provide a brief account of how relational techniques enable you to implement the class lists report. You should contrast this with the the difficulties inherent in using a flat file design. Do not write more than 600 words (**be clear that, in order to achieve at NOC Level 3, your written work must meet the overarching Level 3 criteria - in particular, you should show evidence of reading outside of the taught sessions and you should provide a bibliography**).

Note:

That the report lists students in alphabetical order within each class.

The report is divided into semesters.

There is a calculated field that gives the total number of students on each class.

There is a new page after each class.

You should submit a printed copy of your report for marking.



CLASSID	SREF
1	1
1	2
1	3
1	4
1	5
1	6
1	7
2	8
2	9
2	10
2	11
2	12
2	13
2	14
3	1
3	2
3	3
3	13
3	14
3	17
5	1
5	4
5	18
6	12
6	18
6	19

Notes

Figure 7.11: The REGISTER table



Notes

Class Lists

SEMESTER 1

TITLE	Introduction to database design		
ROOM	1	DAY: Monday	

SNAME	INIT	DOB
-------	------	-----

ARNOTT	K	01-Aug-60
--------	---	-----------

CARTER	J	21-Mar-54
--------	---	-----------

ISHAMO	S	05-Dec-50
--------	---	-----------

JONES	A	10-Nov-48
-------	---	-----------

LANGLEY	S	21-Aug-57
---------	---	-----------

OSMAN	TJ	29-Sep-53
-------	----	-----------

WILSON	H	07-Jul-62
--------	---	-----------

Total Number of Students for Introduction to database design = 7

18 December 2001

Page 1 of 5



Notes

Class Lists

SEMESTER 1

TITLE Introduction to database design
 ROOM 1 DAY: Tuesday

SNAME	INIT	DOB
ARNOTT	A	23-Aug-54
ARNOTT	B	23-May-62
BROWN	J	29-Sep-58
GREEN	N	30-Sep-58
HEY	N	10-Oct-55
JACKSON	H	21-Apr-41
WILSON	K	13-Mar-65

Total Number of Students for Introduction to database design = 7



Notes

SEMESTER 1

TITLE Key Skills - Internet
 ROOM 1 DAY: Monday

SNAME	INIT	DOB
BROWN	J	29-Sep-58
GREEN	J	06-Aug-69
LANGLEY	S	21-Aug-57
OSMAN	TJ	29-Sep-53
WILSON	H	07-Jul-62
WILSON	K	13-Mar-65

Total Number of Students for Key Skills - Internet = 6

18 December 2001

Page 3 of 5



Notes

SEMESTER 1

TITLE Key Skills - Statistics
 ROOM 8 DAY: Wednesday

SNAME	INIT	DOB
CARTER	J	21-Mar-54
GREEN	J	09-Aug-45
OSMAN	TJ	29-Sep-53

Total Number of Students for Key Skills - Statistics = 3

18 December 2001

Page 4 of 5



Notes

SEMESTER 2

TITLE	Computer Systems
ROOM	3 DAY: Friday

SNAME	INIT	DOB
GREEN	J	09-Aug-45
HEY	N	10-Oct-55
WATSON	F	03-Mar-58

Total Number of Students for Computer Systems = 3

18 December 2001

Page 5 of 5



Sub-Forms - Creating a viable User Interface

Many of us can think of computer applications that we rate highly or, conversely, detest using. What makes the difference?

If we assume that most, if not all, modern and mature applications work well on a technical level then the user's experiences of "computing" are differentiated by the design of the user interface. Consider the "class lists" report you are required to do for exercise 7, in itself this is a useful report; it clearly communicates which students are enrolled on which classes.

The problem arises when we consider exactly how the relevant data was input into the database in the first place. At a push, we can imagine that it is just about feasible for a technically competent ACCESS developer (such as yourself?) to add students to the STUDREC table and to add class details to the CLASSES table (Figure 7.10). However, it is not feasible to expect even an experienced developer to successfully populate and maintain a table such as REGISTER (Figure 7.11).

Database developers need to develop workable and intuitive user interfaces that shield the user from the complexities of maintaining what are essentially 'background' housekeeping processes. Ensuring data integrity is also an important consideration in designing a successful user interface. Access provides a mechanism whereby the twin goals of intuitive use and promotion of data integrity can be met, namely the use of Sub-forms.

The successful use of nested forms depends upon:

- the database having more than one table
- a set of relationships established between the tables (figure 7.12)
- the sub form needs to have a corresponding foreign key (to the primary key in the "main" table).

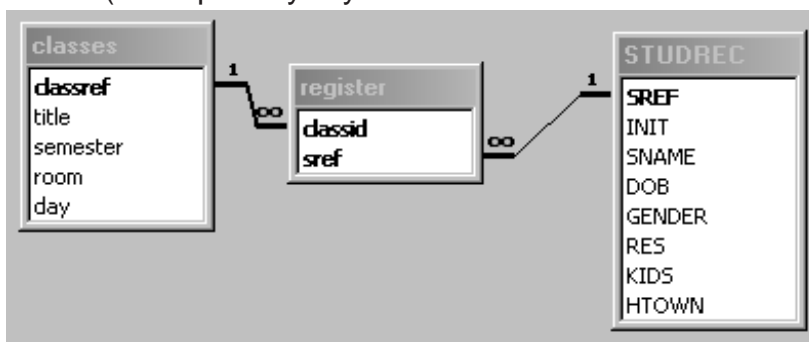


Figure 7.12: CLASSES, REGISTER and STUDREC



Consider the following data input interface (Figure 7.13)

The screenshot shows a window titled 'classes' with a sub-form titled 'Introduction to Database Design'. At the top, there are input fields for 'semester' (value: 1), a dropdown for 'Monday', and 'room' (value: 1). Below this is a section labeled 'register' containing a list of student names in a table-like structure. Each row has a dropdown menu on the left and a text field on the right. The names listed are: OSMAN, LANGLEY, WILSON, CARTER, jones, ISHEMO, ARNOTT, HEY, HARVEY, and an empty row. At the bottom of the register list, there are navigation controls: 'Record: [back] [left] [9] [right] [next] [end] of 9'. Below the register list is a blue bar with four navigation buttons (back, left, right, next) and a red 'STOP' button. The text 'Select Class' is centered below the blue bar.

Figure 7.13: A register update user interface (a form with a sub-form).

Figure 7.13 illustrates the type design considerations that a designer needs to bear in mind. The fact that “Introduction to Database Design” has 8 students enrolled is instantly apparent as is the way in which a user can browse through each class register.

Implementing a user interface to enable efficient updating of registers.

This section will take you through the process of constructing the form shown in figure 7.13.

Note: before beginning this process, you MUST have successfully completed exercise 7, this will ensure that you have the tables CLASSES, REGISTER and STUDREC successfully related as shown in figure 7.12.



Notes

We could use the forms wizard to accomplish the entire task, however a more efficient way to approach this task is to set up a query with the required fields and relationships properly defined; this means we can reuse the query in future tasks.



Create a query called qry_registers as shown in figure 7.14

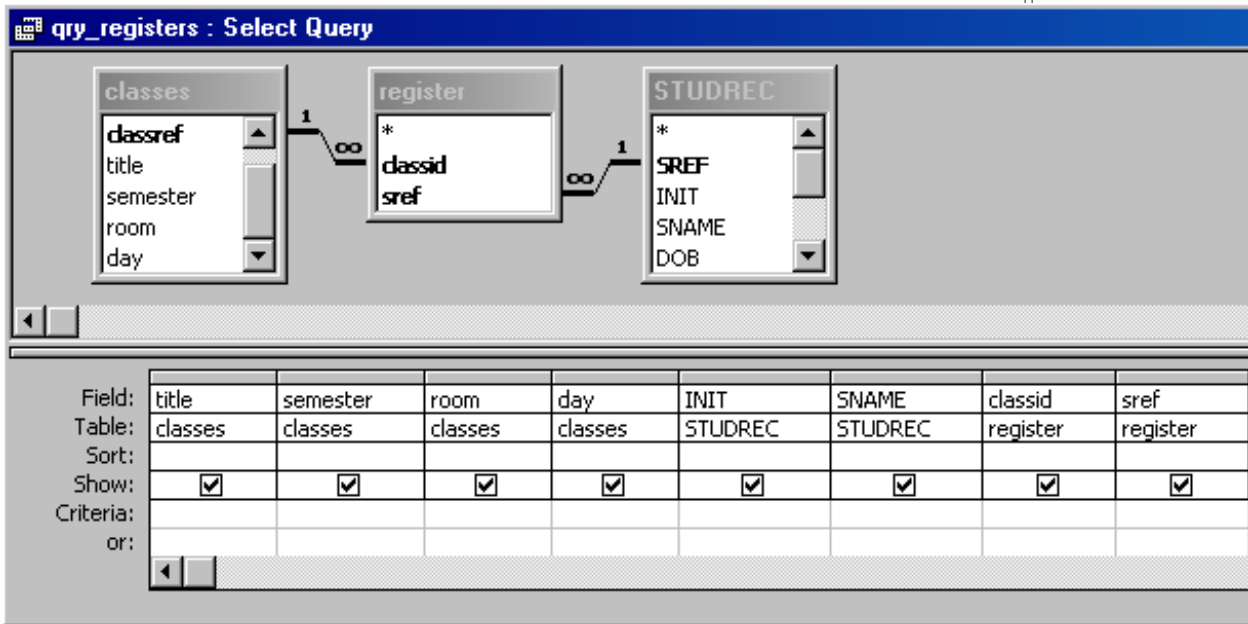


Figure 7.14: qry_Registers



Select the “Create form by using Wizard” option



Select the query qry_registers as the data source and ensure that you then select all fields to be included in the form(s) (Figure 7.15).

Note that the SREF field is from the REGISTERS table not from the STUDREC table (i.e. register.sref).



Notes

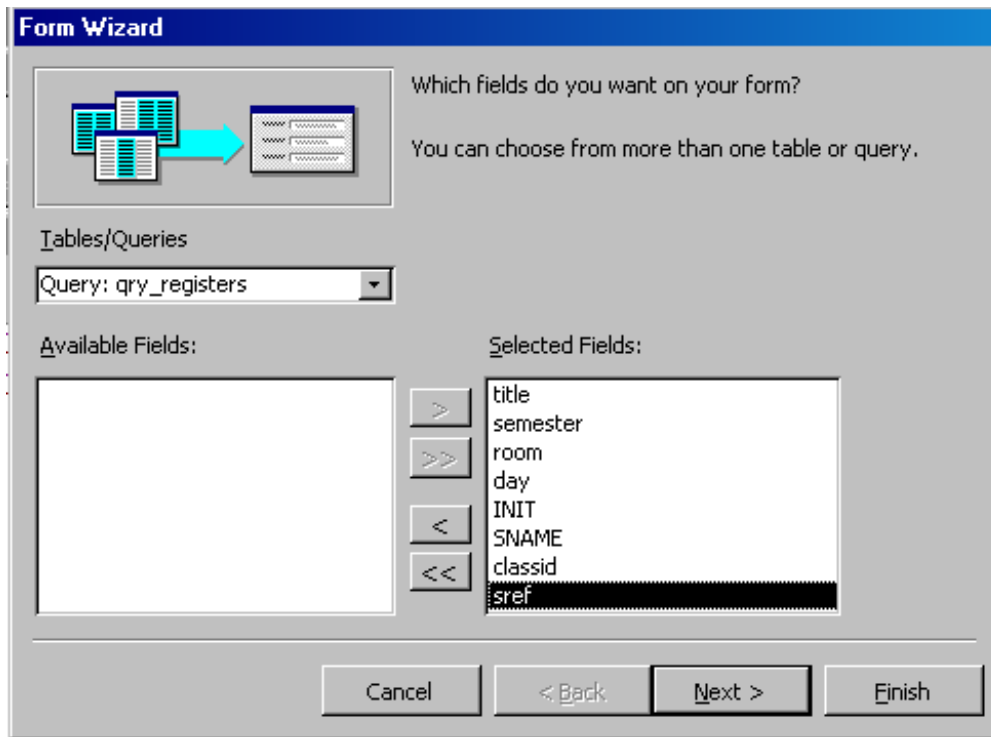


Figure 7.15: Selecting the fields.



Click on the Next button, this will display the dialogue box shown in figure 7.16 (below).

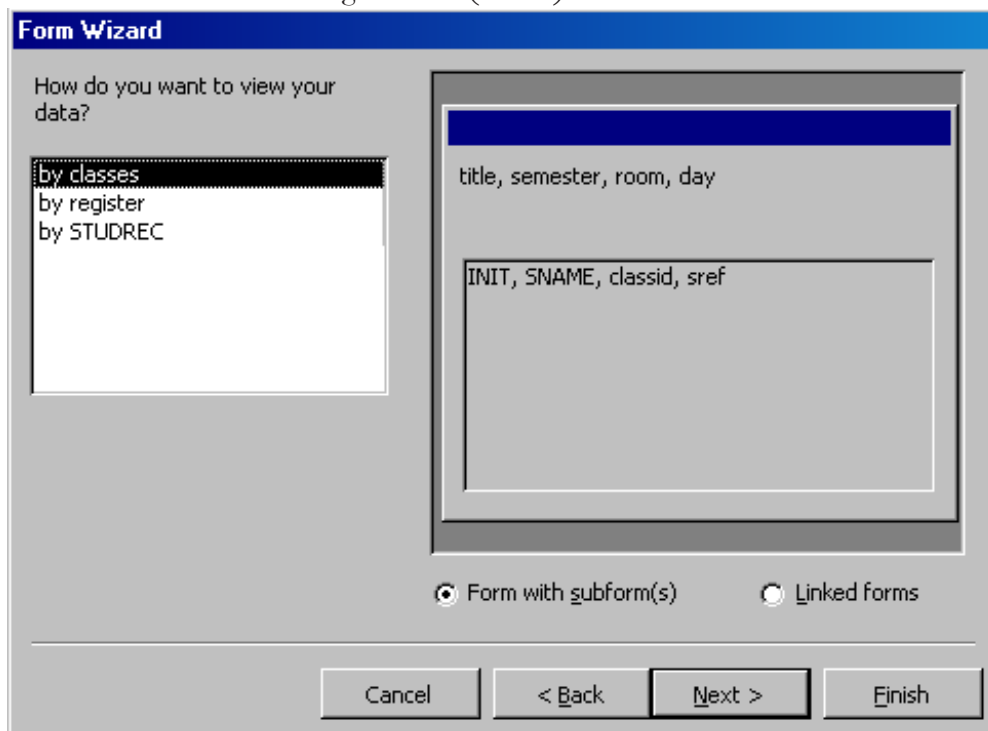


Figure 7.16: How do you want to view your data



Access has automatically picked up the fact that the data originates from different tables and therefore “knows” that it needs to develop two forms to display the data. You are being prompted to select which table will populate the Main Form and (by implication) which data will be displayed in the sub-form.

We wish to display registers, therefore we need to ensure that the data is displayed by CLASSES. Checking your screen should show that the data is set up for “display by classes”. Note the “Form with subform(s)”/“Linked forms” options; a linked form will display a button on the main form which, when clicked will open a sub form. By contrast, the subforms option will automatically display the open sub form(s) within the main form.

☛ Ensure that you have selected the “by Classes” and “Form with Subform(s)” options and click on the Next button.

☛ Select the Datasheet option (Figure 7.17)

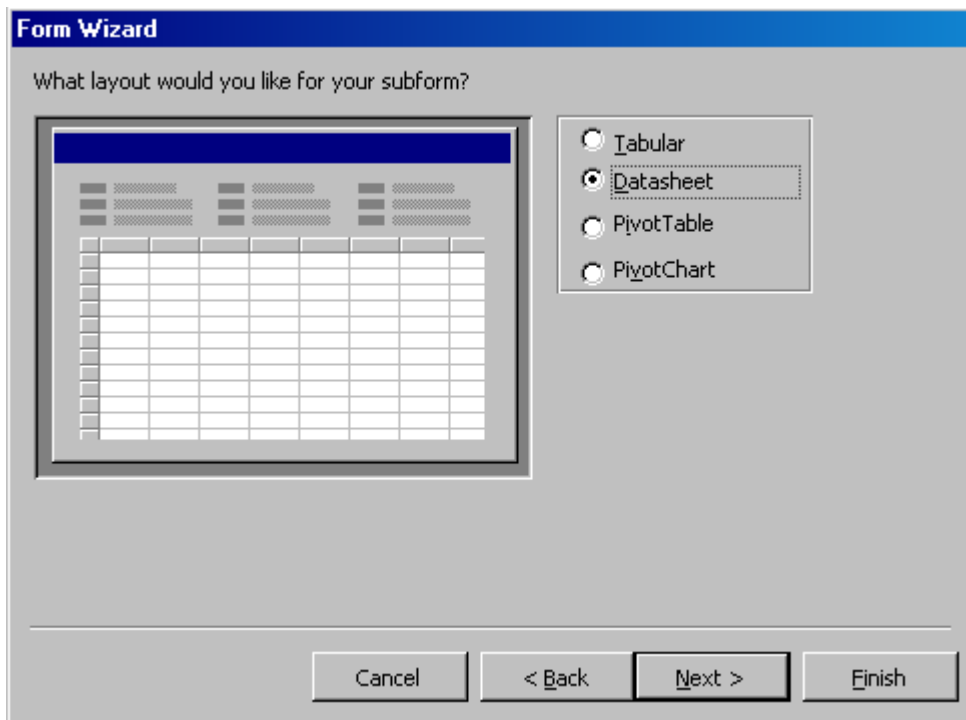


Figure 7.17: Form layout (choose Datasheet)

☛ Click on the Next button

☛ Select the form style you require (I'd suggest Standard)



- Click on the Next button to move to a dialogue box that invites you to accept or change the names of the main form and sub form that the wizard has generated (figure 7.18).

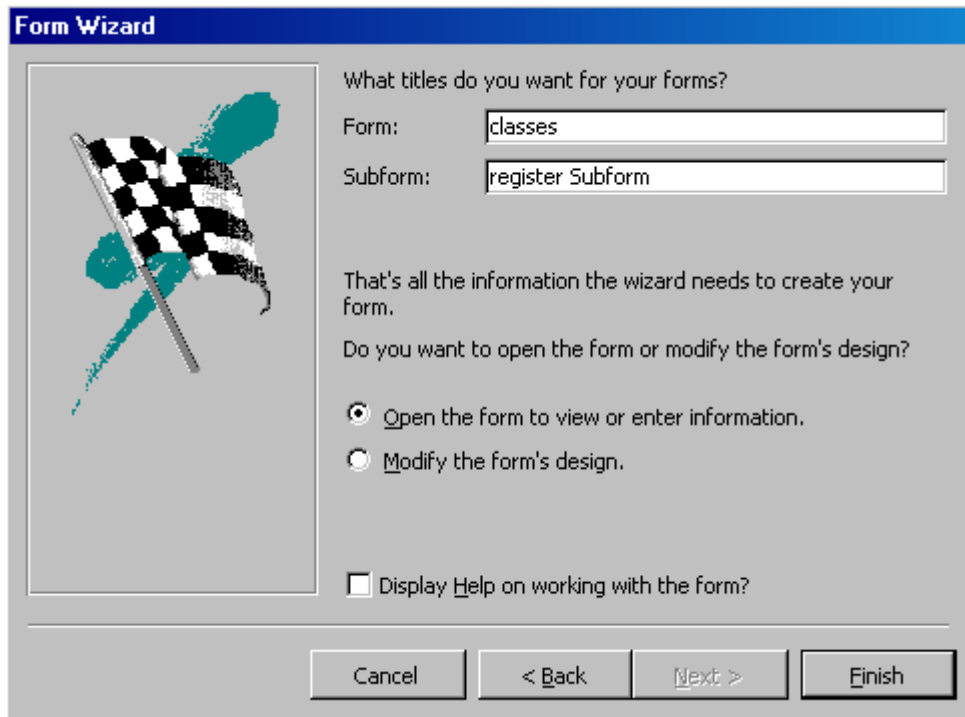


Figure 7.18: Naming the Main Form and Sub form

- Make a note of these names and accept them by clicking on the Finish button.

You should see a form (and sub form) on your screen similar to that shown in figure 7.19.



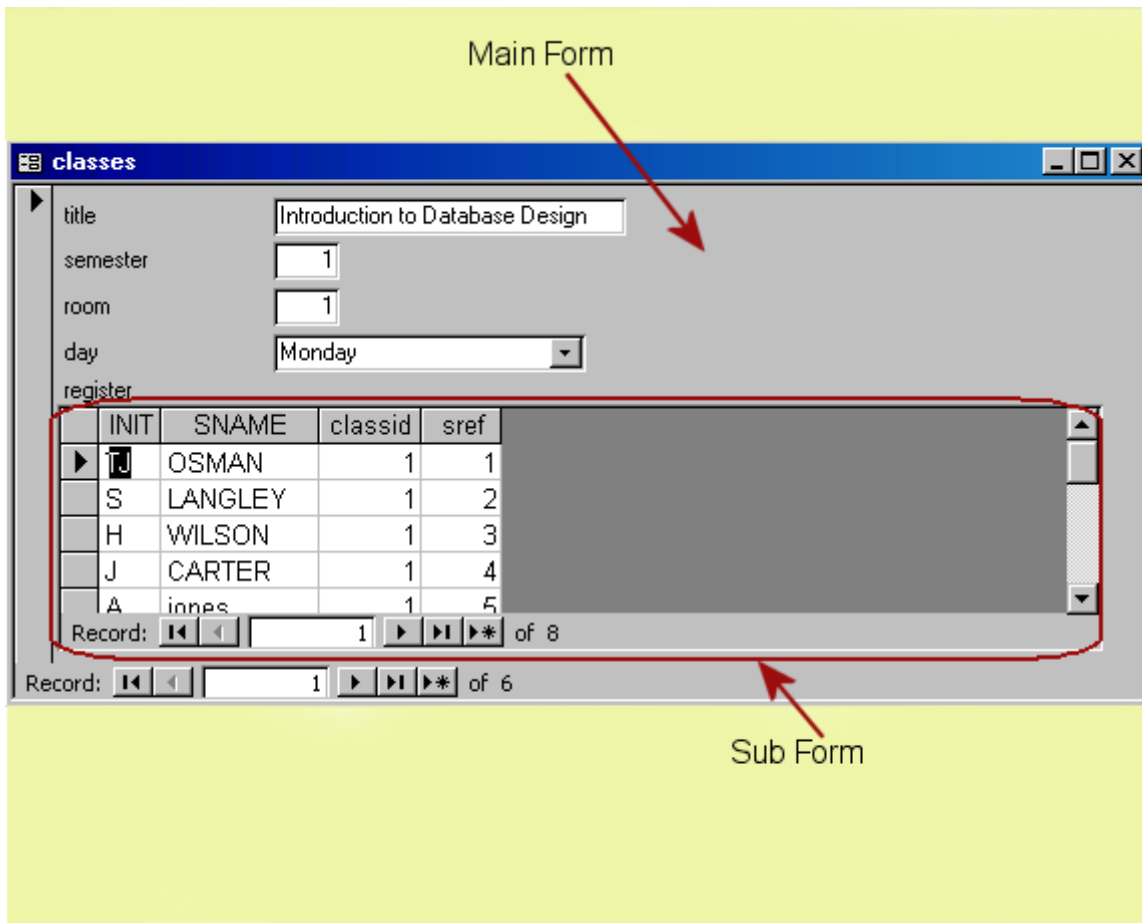


Figure 7.19: The Classes form with the register sub form

Notice the two sets of record selector buttons (figure 7.20).



Figure 7.20: Record Selectors

The outer set will step through the records in the Main form (i.e. will step through each class in turn) whilst the inner set of Record selector buttons will step through each student on the register for a particular class.

Redesigning the forms.

The major advantage of this form/sub-form interface is that the “junction table” REGISTER (which as you will remember, consists of only two foreign keys) will automatically be updated as students are added to or deleted from the class register.



Amendments to data (e.g. to student names) can also be successfully written back to the appropriate table via this form. However, it is not advisable to allow this in this example. Consider if we really want a teacher to be able to change the name of a student on that register, especially as the change would be reflected throughout all other registers that happened to have that student enrolled. Perhaps such a change should be made centrally in an “edit student’s details” section of the database.

The first change we’ll make to the user interface is to present the student names in a linear fashion (see Figure 17.13).

- ☛ Close the form CLASSES
- ☛ Open the form “register Subform”
- ☛ Ensure that you are in design mode
- ☛ Delete the field labels (we don’t need them)
- ☛ Arrange the fields in a straight line (REGISTER.SREF, INIT, SNAME, REGISTER.CLASSID) (see Figure 17.21)

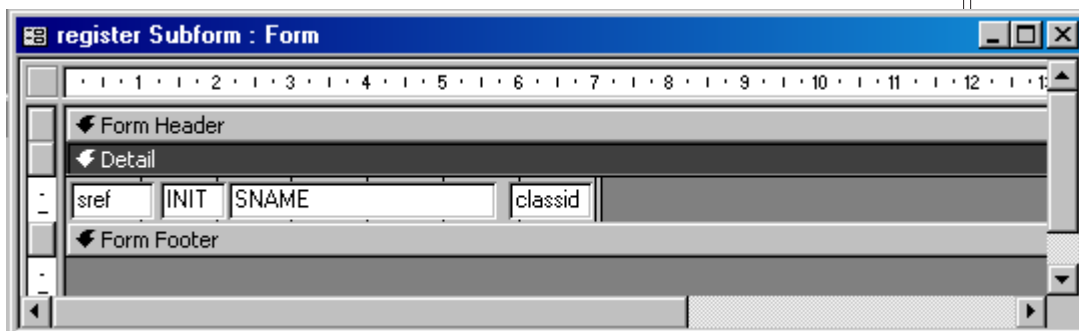


Figure 17.21: The redesigned “register Subform”

- ☛ View the form data; you should see something similar to Figure 17.22.



INIT	SNAME	classid	sref
TJ	OSMAN	1	1
TJ	OSMAN	3	1
TJ	OSMAN	5	1
S	LANGLEY	1	2
S	LANGLEY	3	2
H	WILSON	1	3
H	WILSON	3	3
J	CARTER	1	4
J	CARTER	5	4

Notes

Figure 17.22: Datasheet view of “register Subform”

The data shown in figure 17.22 is not in the form we might have expected (see Figure 17.19), so what went wrong? The answer is that nothing went wrong, rather, the form is been displayed in “DataSheet” view rather than in a “Form” view, we need to ensure that the form is displayed in “Continuous Forms” view.

- ☛ Switch back to the Design view and display the properties of the form (ensure that the Format tag is selected in the properties window)
- ☛ Set the “Default View” to “Continuous Forms” (Figure 17.23)



Notes

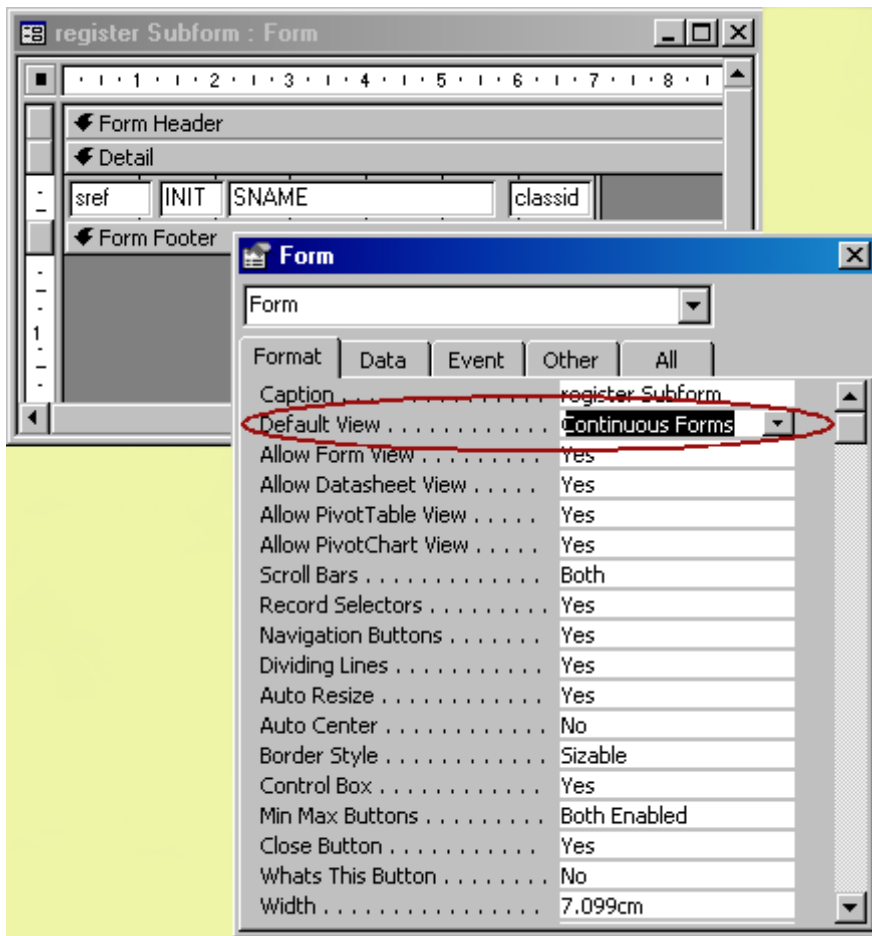


Figure 17.23: Setting the Default View of a Form

☛ Switch back to “Form View” mode

Your finished form should be similar to the one shown in figure 17.24.

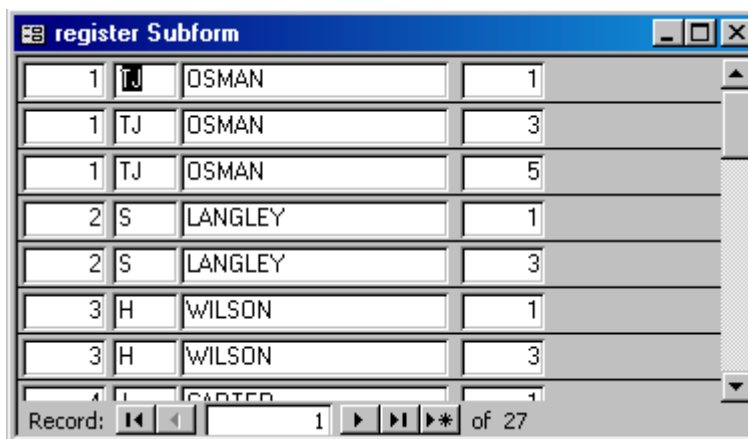


Figure 17.24: The “register Subform” in “Continuous Forms” mode.



Note that we could choose to hide the fields REGISTER.SREF and REGISTER.CLASSID but we can't remove those fields from the form as they are needed to "link" the sub form to the main form.

Hiding a field on a form

We are going to hide only the REGISTER.CLASSID field as we need access to the REGISTER.SREF field later in this example.

- ☛ Switch to design view (of "register Subform")
- ☛ Select the field REGISTER.CLASSID
- ☛ Ensure that the properties are displayed (select the Format Tab if necessary)
- ☛ Change the "Visible" property to "No".

Your form should now look like that shown in figure 17.25 (below)

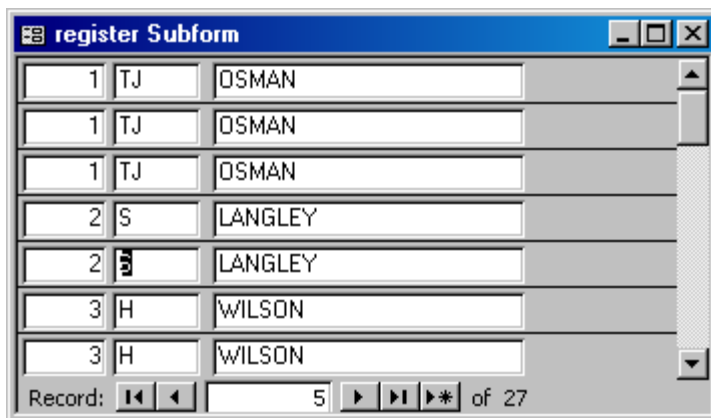


Figure 17.25: REGISTER.CLASSID is hidden

- ☛ Save and close "register Subform"
- ☛ Open the form "Classes", it should look similar to figure 17.26



Notes

classes

title: Introduction to Database Design

semester: 1

room: 1

day: Monday

1	TJ	OSMAN
2	S	LANGLEY
3	H	WILSON
4	J	CARTER

Record: 1 of 8

Record: 1 of 6

Figure 17.26: The form "Classes"

Exercise 7.2

Use the form design skills you have to amend the form "Classes" from that shown in figure 17.26 to something similar to figure 17.13

In doing so, you could consider some of the following form property attributes of the two forms:

1. Allow Datasheet View
2. Allow PivotTable View
3. Allow PivotChart View
4. Scroll Bars
5. Record Selectors
6. Navigation Buttons
7. Dividing Lines



Adding and deleting students from a register

If you have developed an answer to exercise 7.2, you will perhaps realise that you have developed a potentially useful user interface to the part of the student records database that deals with assigning students to particular classes. Perhaps a college registry department could use it to prepare registers for the teachers.

However, the interface as it stands is not robust enough to ensure the integrity of the data and smooth running of the database.

There are two problems we face:

1. Student's names can be amended directly from the register
2. Adding a new student to a register – this is because we can only add an existing student (i.e. one who has a record in STUDREC) to the register; how does anyone using this interface know which students are recorded on STUDREC?

The answer to problem 1 is to make all fields except REGISTER.SREF read only, whilst the answer to problem 2 is to make REGISTER.SREF a lookup field (similar to the TOWNS example earlier in this chapter).

Ensuring fields are 'read only'

The two field properties that could be useful in this scenario are Enabled and Locked.

Consider figure 17.27 (below), this shows the property settings of the TITLE field.

Setting "Enabled" to "No" will prevent the field being selected on the form, whilst setting "Locked" to "Yes" will ensure that the field cannot be changed. This two pronged approach will doubly ensure that the TITLE field is not changed by a user of this interface.



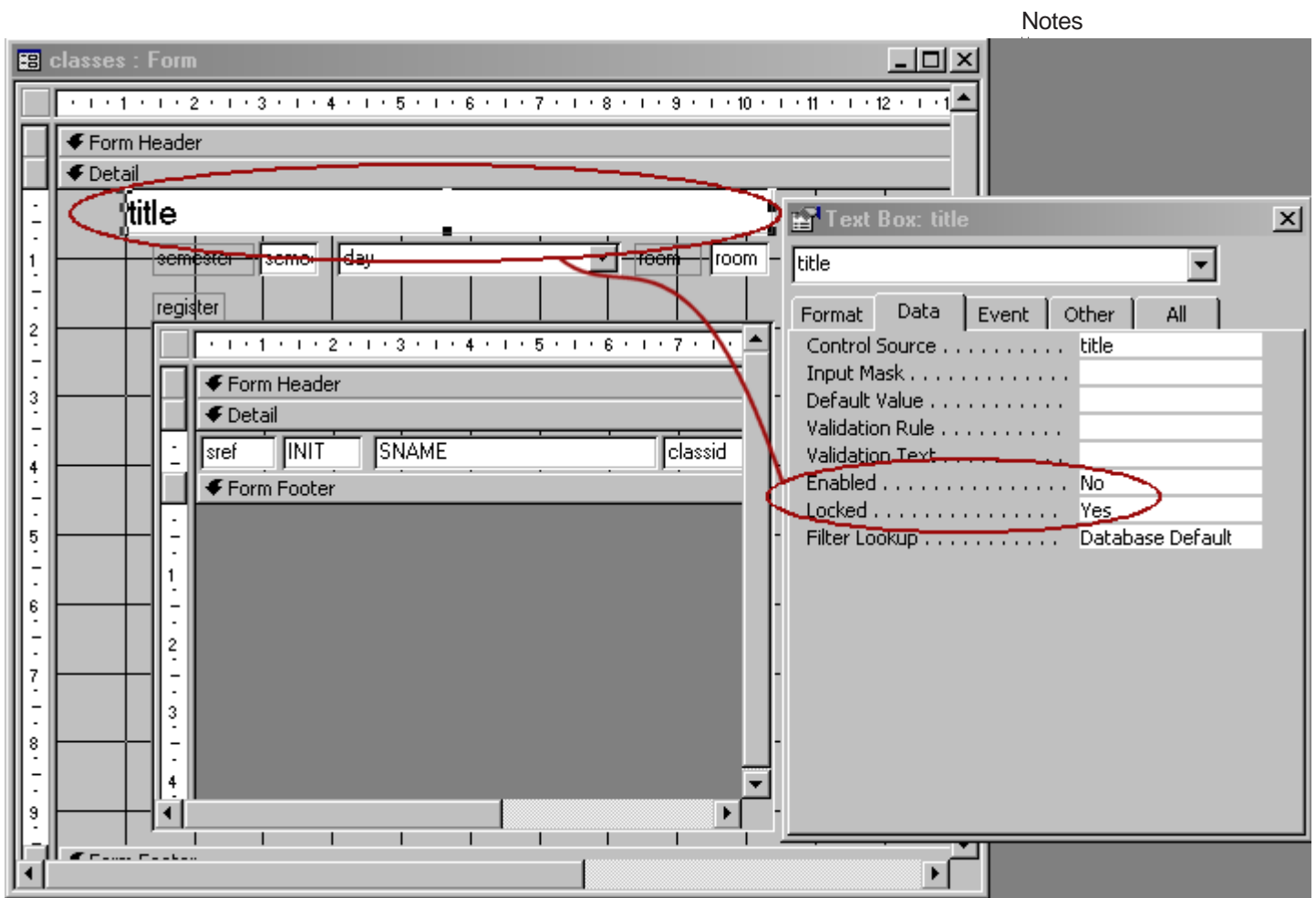


Figure 17.27: Disabling and Locking the TITLE field

Exercise 7.3

Ensure that all of the fields on the two forms of this user interface are both locked and disabled (except for the REGISTER.CLASSID field - which is hidden and the REGISTER.SREF field - which we will use in the next section).

Changing REGISTER.SREF to a lookup field

In practice REGISTER.SREF would have been defined as a lookup field before starting to design the user interface, however for instructional purposes we'll take a look at the interface and see what happens when we try to add a student to a register.

Note that amending a student's data is no longer possible and also that deleting a student from a register is a straightforward process that would not invalidate the database.



- ☛ Assuming that exercise 7.3 has been successfully completed, load the CLASSES form
- ☛ Try selecting any field other than REGISTER.SREF, you should find that you cannot (Figure 17.28)

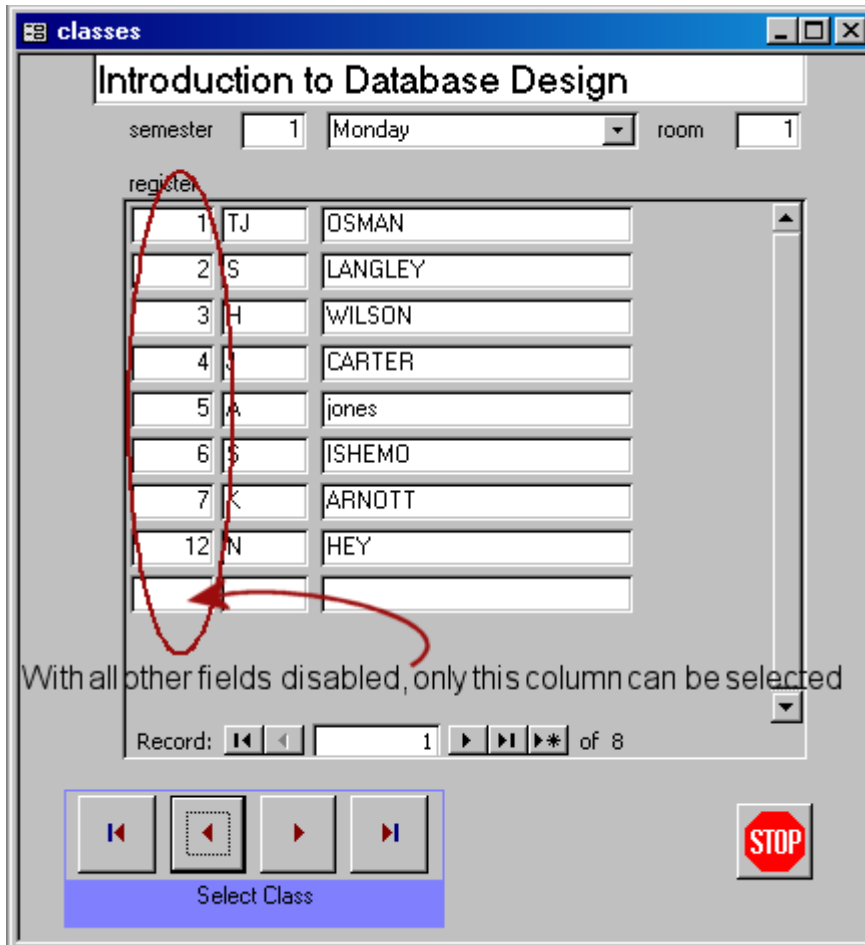
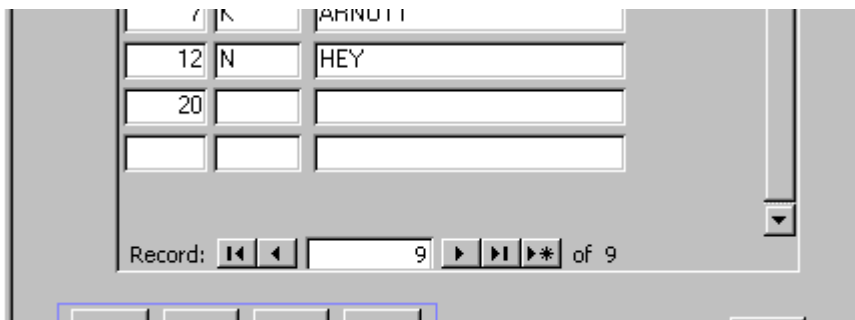


Figure 17.28: Only REGISTER.SREF is enabled

Consider the case where we wish to add a student to the Semester 1 “Introduction to Database Design” register. The only thing we can do is to add the student’s reference number (REGISTER.SREF) to the register. This has all sorts of human interface related problems and drawbacks associated with it. Consider figure 17.29, this shows an attempt to add the student L HARVEY to the register.





Notes

Figure 17.19: Adding a new student (stage 1)

As you can see, we need to know the reference number of the student (how likely is this?). Secondly, there is no immediate confirmation that student number 20 is in fact L HARVEY until we enter the record (pressing the return key will do this). Only at this stage do we know we have added L HARVEY to the register (figure 17.20).

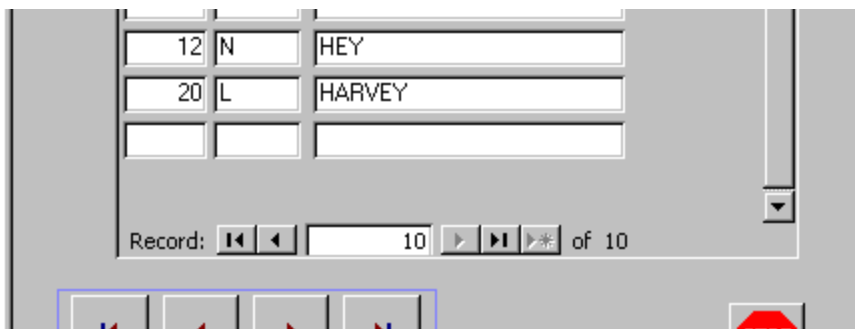


Figure 17.20: Adding a new student (stage 2)

Notice that the details of the student's initials and family name are displayed in the disabled fields.

It is true that we can always delete L HARVEY if this proves to be a mistaken entry on the register, but this way of doing things does seem a bit hit and miss.

Things get even worse when we consider that we could enter a value for REGISTER.SREF that does not exist in the table STUDREC, this would violate the rules of referential integrity: as a result of this breach ACCESS would inform us of that fact with an error message (Figure 17.21)

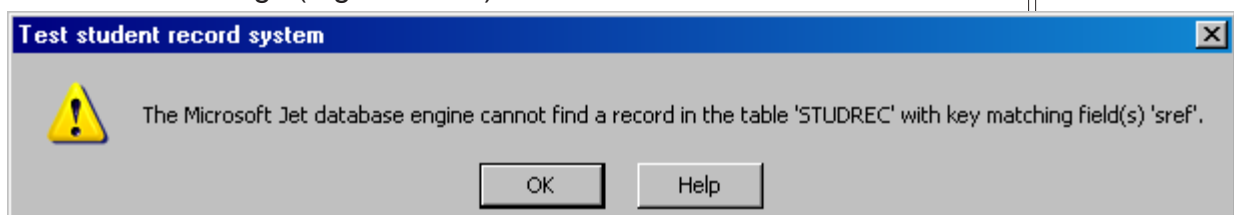



Figure 17.21: Invalid key field message



The resolution of our difficulties lies in redefining the field REGISTER.SREF as a lookup field.

Defining REGISTER.SREF as a Lookup field

As Figure 17.12 shows, REGISTER.SREF is currently involved in joining REGISTER with STUDREC. Before we can redefine REGISTER.SREF we need to delete the join (relationship) on that field.

- ☛ If you haven't already done so, save and close the form "Classes", together with the sub form.
- ☛ Open the relationships window 
- ☛ Right click on the line representing the join between REGISTER and STUDREC on REGISTER.SREF (Figure 17.22)

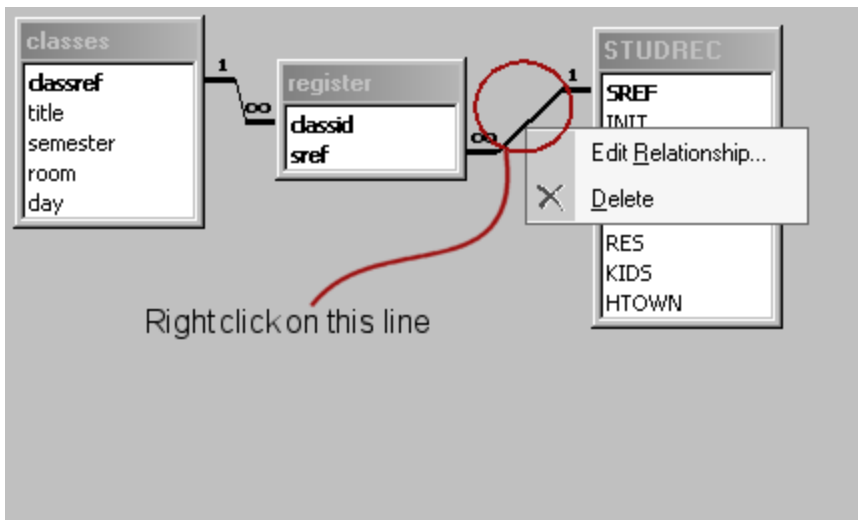







Figure 17.22: Deleting a Join










- ☛ Select the delete option from the pop up menu.
- ☛ Confirm that you want to delete the relationship
- ☛ Close the Relationships window
- ☛ Open the table REGISTER in design view



Notes

-  Select the SREF field and change the data type to “Lookup Wizard”
-  From the table STUREC, select the following fields: SREF, INTI, SNAME
-  Finish running the wizard (Make sure that the “Hide Key Field” box is ticked – it will be by default).
-  Save and close the REGISTER table
-  Open the “Relationships” window

You should see that the join between REGISTER and STUDREC on SREF has been re-created. However, referential integrity is not enforced at this stage, we need to do this.

-  Right click on the line representing the join between REGISTER and STUDREC
-  Select the “Edit Relationship” option
-  In the “Edit Relationships” dialogue box, ensure that the “Enforce Referential Integrity” box is ticked.
-  Click on the OK button to accept the changes and close the dialogue box.
-  Close the Relationships window.
-  Open the form “register Subform” in design view
-  Delete the existing field REGISTER.SREF (we need to replace it with the updated version)
-  Using the “Field List” icon  insert REGISTER.SREF back into the form.



Notes

- ☛ Delete the INIT field
- ☛ Ensure that the form is adequately laid out (figure 17.23).

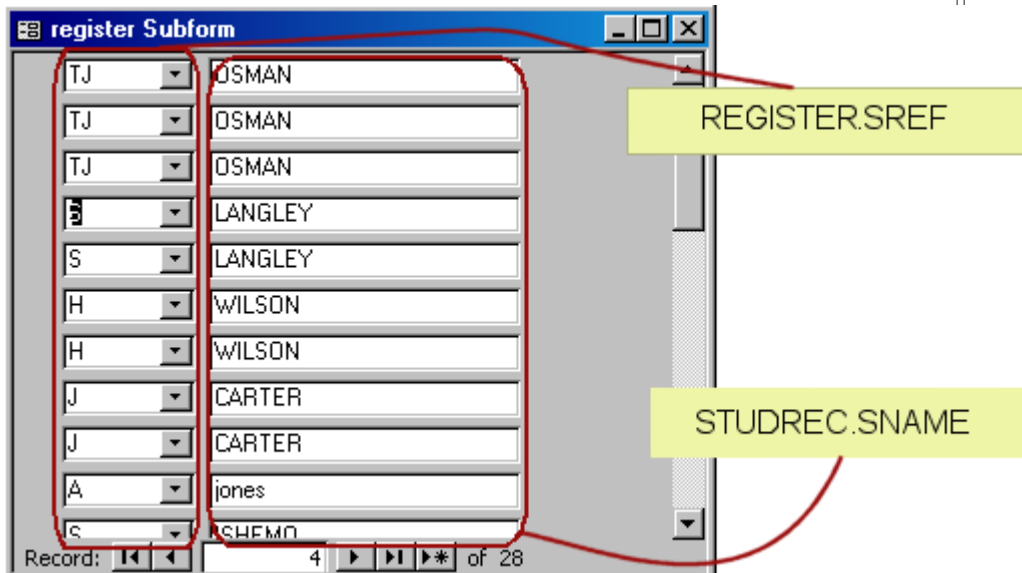


Figure 17.23: The redesigned “register Subform”

- ☛ Ensure that the form you are working on is similar to figure 17.23 and then save and close the form.
- ☛ Test the redesigned user interface by opening the “Classes” form (this should be similar to the user interface shown in figure 17.13).



Notes

Exercise 7.3:

Try adding and deleting one or two students to one or more classes.

Try adding the same student to the same class twice (note what happens).

Using your observations, complete the following assignment.

Assume you are a Technical Author and that you have been asked to write user instructions for people who are going to use this interface to maintain class registers. Ensure that you illustrate your instructions with appropriate screen shots of your user interface. Produce the instructions on no more than 2 sides of A4 .

Submit your instructions to your tutor for marking.

