

Essential Relational Technique & Theory

Aims of Session

To enable you to:

- Describe the limitations of a flat file
- Describe in general outline the process of normalisation
- Use Access to create a simple relation (under tutorial guidance)
- Base a query on two related tables (Join)
- Base a report on a relational query
- Base a form on a relational query

What is a relational database?

The model of a relational database was introduced in 1970 by E. Codd. The relational model is concerned with looking at data and how that data is represented. A relational database consists of a set (two or more) of tables that hold data and can be cross referenced (joined) to produce complex views of that data.

Most database designs produced by inexperienced designers consist of one data table (a FLAT FILE database). Users commonly expect this single data table to hold all of the data to be stored in the database, however it should be noted that this approach will lead to problems. It should be remembered that a database can and often does consist of more than one data table.

Let us assume that we have the task of designing a database that will list all students who have cars. The idea being that users can type in a car registration number (a good key field?) and find out who the owner is - a sort of mini DVLC. Also, it will be useful to type in a student's name and find out which car(s) they own.

At this point in the course we will make significant changes to the design and structure of our student records database. You will need to retain the existing version so that your previous exercises will still work.



At this stage we're going to change the structure of some of the tables in your "Student Records" database. In order that you can still access your earlier work on queries, reports and forms you need to make a backup copy of the database.

- ☛ Make a copy of your Access database (Student Records) and save the copy as

OLD STUDENT RECORDS

We'll continue this course using the database version called **Student Records**.

In order to record car details we could amend the design of STUDREC so that is as shown in Figure 6.1.

SREF	INIT	SNAME	DOB	GENDER	RES	KIDS	HTOWN	DISTANCE	REGNO	MODEL	COLOUR
------	------	-------	-----	--------	-----	------	-------	----------	-------	-------	--------

Figure 6.1: STUDREC with car details (ver. 1)

The additional fields are:

- REGNO: (The cars registration number)
- MODEL: (What type of Car is it?)
- COLOUR: (What colour is the car?)

For the purpose of this exercise we shall assume that these three fields will suffice: if we were interested in calculating traveling expenses then it would also be necessary to include engine size and mileage.

The design in Figure 6.1 seems at first glance to work. However if we consider the situation where a student has more than one car (I know it's not likely, but it is possible) things become more complicated. Suppose T Osman owns a C registration Cavalier and then buys a Honda Accord (F registration). How will the current design accommodate this?

We could amend the Structure of STUDREC by adding REGNO2, MODEL2 and COLOUR2 as shown in Figure 6.2.



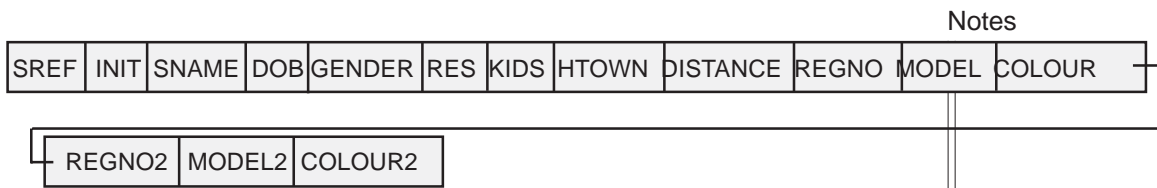


Figure 6.2: STUDREC with car details (Ver. 2)

It doesn't take much thought to realise that this is a very inflexible solution. Suppose that T. Osman purchases a third car, or even a fourth or fifth, how would we store the data? It is (I hope) evident that we cannot go on adding more fields to the record.

A more sophisticated attempt at solving the problem would be to suggest adding a new record for each car, i.e. if T Osman owned one car there would be one record, two cars would produce two records and three cars would produce three records and so on. This is shown Figure 6.3.

SREF	INIT	SNAME	DOB	GENDER	RES	KIDS	HTOWN	REGNO	MODEL	COLOUR
1	T	OSMAN	29/09/53	M	F	0	MGREEN	C579 CUA	CAVALIER	WHITE
1	T	OSMAN	29/09/53	M	F	0	MGREEN	F28 CWE	HONDA	WHITE
1	T	OSMAN	29/09/53	M	F	0	MGREEN	123 ABC	MAZDA	RED

Figure 6.3: The Studrec Table with car details added

Consider the problems that this would cause.

- i There would be unwanted duplication of data. The owners name and address and other details would be repeated for each car they own.
- ii What would happen if we deleted a record which contained details of a car that a student had sold? If it was the students only car, we would no longer have any reference to that student.
- iii There would be a danger of inconsistency between duplicate records (ROWS), e.g. different details of (say) different dates of birth may be entered by mistake when amending a record.



If T Osman moved from Millhouse Green to Thurlstone we would need to scan the entire data table to ensure that we updated each record (ROW) which applied to T Osman. This generates ample scope for error - we could easily miss a record. Doing so would lead to internal inconsistencies within the database i.e. the integrity of the data would be compromised.

If T Osman were to give up driving and walk to work everyday, we would need to delete all records. If we do this we will delete all reference to T Osman.

The problem areas can be summed up in three broad categories, **insert**, **delete**, and **update**.

Rule of Thumb: Always ask what happens when you add data to a database, delete data from a database and amend data in a database.

How can the design be altered to overcome or sidestep these problems?

The answer to this problem is to split the data table into two or more smaller files (TABLES) this is a process known as *normalisation*. There are a number of rules which act as guidelines to a successful split of the data (for those of you who wish to know more, refer to CJ Date¹ on relational databases and make sure you understand the processes of first, second, and third normalisation of data.

Rule of thumb: a good design principle is "one fact in one place"



First, Second and Third Normal Forms

The rules of normalisation can be paraphrased as follows:

First Normal Form:

Make sure that your data is laid out in ROWS and COLUMNS (Shown in Figure 6.4).

THIS

Name	Car
Tom Osman	Cavalier
Tom Osman	Honda
Tom Osman	Mazda
Tom Osman	Fiat

NOT THIS

Name	Car
Tom Osman	Cavalier
	Honda
	Mazda
	Fiat

Figure 6.4: First Normal Form

Second Normal Form:

Ensure that all of the data in each record is fully dependent upon the primary key field.



Third Normal Form:

Ensure that there is no *transitive dependency*, i.e. field A depends upon field B which in turn depends upon the key field. For a good (but extremely detailed) example please refer to “An Introduction to Database Systems Vol: 1 CJ Date - Chapter 17.) We will look at transitive dependency in detail in the next section of this course.

Normalising STUDREC

Because STUDREC is already laid out in rows and columns (Figure 6.4) it can be regarded as being in first normal form. There are no repeating fields.

To get it into second normal form we need to decompose the table (STUDREC) into two tables, one dealing with the students, the other dealing with cars. We need to ensure that the values in the fields in each table depend upon the key field for each table. It is evident that only the fields INIT, SNAME, DOB, GENDER, RES, KIDS and HTOWN depend upon the value of the key field SREF (the student reference number). A blue car would still be blue regardless of who owned it. The fields MODEL and COLOUR however depend upon the key field REGNO, the cars registration number.

Therefore we can decompose the data into two tables as shown Figure 6.5.

STUDREC

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

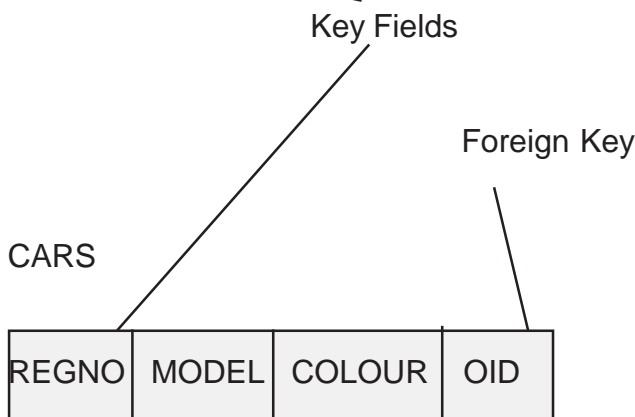


Figure 6.5: Key Fields in the STUDREC and CARS tables



Notice that an additional field, OID (Owners IDentification) has been added to the CARS data table, this is known as a Foreign Key, i.e. it provides a means of cross referencing the CARS table to the STUDREC table. In the relational model there are restrictions on the set of values that the foreign key (OID) may have. OID MUST point to an existing ROW in STUDREC or be a NULL value.





The CARS data table

REGNO	OID	COLOUR	MODEL
CUA 579 C	1	WHITE	CAVALIER
F 28 CWE	1	WHITE	HONDA ACCORD
TJO 1	1	RED	FERRARI
TJO 2	1	GREEN	RANGE ROVER
GXN 732 Y	2	BLUE	ESCORT
ABC 123	3	WHITE	ESCORT
D 111 ABC	4	RED	GOLF
H 123 WW	4	YELLOW	2CV
G 123 NC	5	BLACK	FIESTA
UWJ 189 Y	10	RED	MAZDA
N 609 UWJ	1	MERCURY	CARINA

Figure 6.6: The CARS data table

As always, things will become clearer if you work through a practical example. We are going to create a table (CARS), populate it with the data shown in Figure 6.6 and relate CARS to STUDREC.

To create the CARS table in our student records database follow through the worked example.

-  Ensure that the database window for your STUDENT RECORDS database is open
-  Select the TABLES tab
-  Click on the NEW button
-  Select DESIGN VIEW



Notes

☞ Click on the OK button (this will place you in the design view of a new data table) - we now need to define the data fields

☞ Define the following fields:

Field Name	Data Type	Field width/number type
REGNO	TEXT	10
OID	NUMBER	INTEGER
COLOUR	TEXT	20
MODEL	TEXT	20

You should end up with something similar to Figure 6.7 (without the field descriptions).

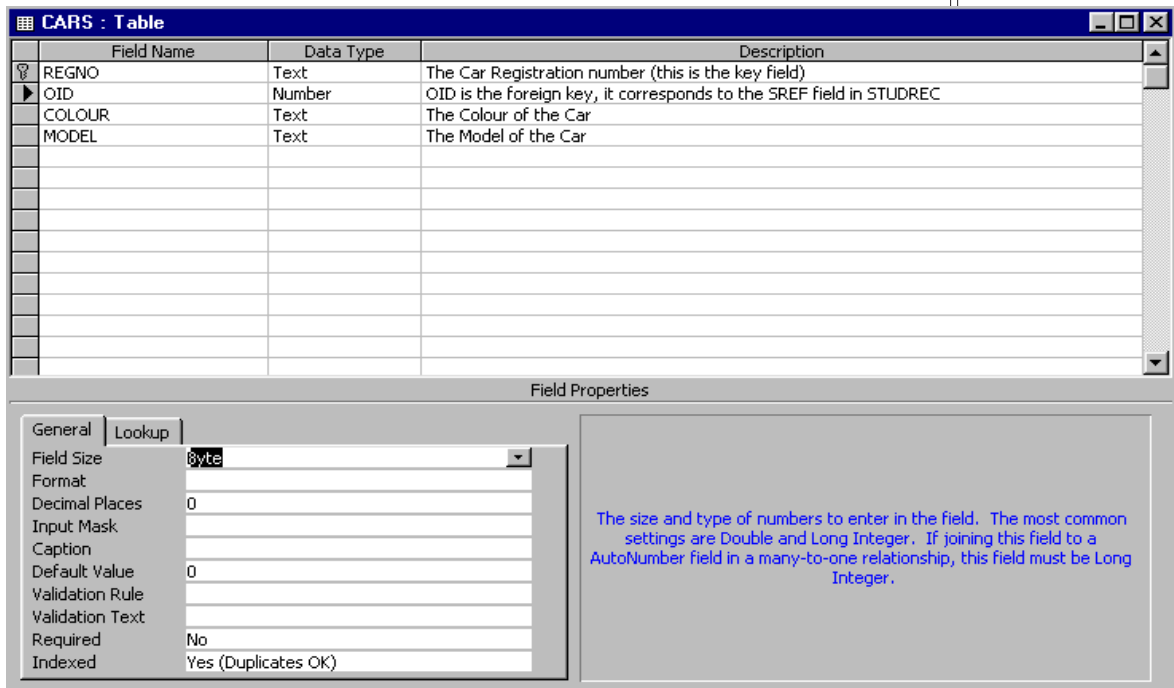


Figure 6.7: The Design View of the CARS table

☞ Ensure that you set the key field to REGNO

☞ Save the table as "CARS" (no quotation marks)

☞ Switch to datasheet view



Notes



Enter the data exactly as it is shown in Figure 6.6



Save the table CARS

Abandoned cars

Let us assume that we have found an abandoned car and that it is a black Trabant and that its registration number is SAD 1. We can enter the details onto the CARS table, because OID is allowed to be empty (NULL). This does NOT mean that OID contains the number 0 (zero), on the contrary, be very careful to delete the 0 (zero) that is the default value for the OID filed in CARS.



Add a new record to the CARS table with the following details:

REGNO = SAD 1

OID = null

COLOUR = BLACK

MODEL = TRABANT



Close the Table CARS

Note: Null means that we enter nothing. **DO NOT** enter 0 (zero); zero is not null.

Entering 0 (zero) will result in the database trying to find an owner whose ID is 0 (zero, as we don't have such an owner an error of consistency will have occurred. Most RDBMS systems enforce referential integrity and will not allow such an error to happen.



Relationships in Access

We now have two tables, STUDREC and CARS. How do we get them to work with each other?

The answer to this question is that we must define a relationship between the two tables.

Relationships can be of three types:

- One to One
- One to Many
- Many to Many

One to One relationships

TABLE A

SREF	INIT	SNAME
1	T	Osman
2	J	Smith
3	H	Jones
4	K	Granger

TABLE B

REF	PHONE No.
1	01226 123456
2	01484 123456
3	01484 234567
4	01226 234567

Figure 6.8: One to One Relationship

Each record in table A (Figure 6.8) has only one matching record in table B. The two are linked by the values of SREF and REF. In practice we wouldn't use this design, we would simply add an extra field in table A to hold the students' telephone number.



One to Many Relationships

Each record in table A can have one or more matching records in table B (Figure 6.9). Tutor ID in table A cross references with TID in Table B.

TABLE A

TABLE B

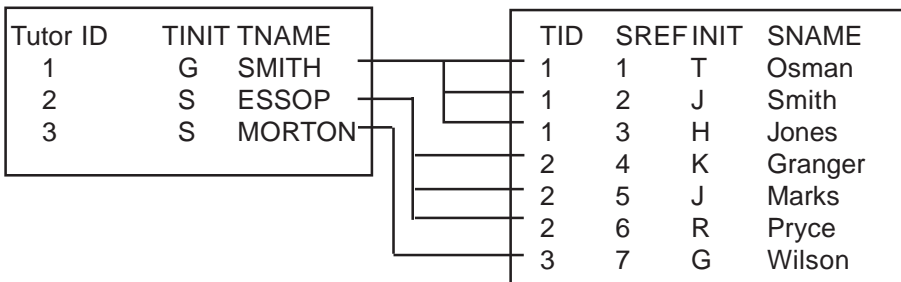


Figure 6.9: One to Many Relationship

Each record in table A can have one or more matching records in table B. In this example, table A is a list of personal tutors and table B is a list of students. We can state with certainty that each personal tutor may have one or more tutees and each tutee can only have one personal tutor (indeed, MUST have a personal tutor). Table A is related to table B on the field Tutor ID via the TID (tutors ID) field in table B.

This is a commonly used design.

Many to Many Relationships

Understand this from the very beginning, in an implementation of the relational model you can't have a many to many relationship. However, it is possible to *imitate* the effect of a many to many relationship by creating a third table and using two "one to many" relationships.

The effect of a many to many relationship is to enable each record from table A to have one or many matching records in table B and each record in table B to have one or many matching records in table A.



Imagine that we have a series of tutorial groups (A, B & C) taught by a number of tutors, each tutor can teach each group. This in effect means that many tutors can teach many students and that each student can have many tutors. We can't model this by having only a TUTORS table and a STUDENTS table, another (Junction) table is needed. In our example, the Junction table is recording details about the dates that each class will run, so we'll call the table TIMETABLE. This is modeled in Figure 6.10.

It is possible to relate table B to the Junction table on a one to many basis and then to relate the junction table (TIMETABLE) to table A. This would have the effect of listing each student together with each tutor who teaches that student.

If we wanted to, we could relate table A to the Junction table (TIMETABLE) and then relate the Junction table to table B. This would have the effect of listing each tutor together with each student that that tutor teaches.

In the case of G. SMITH, it is possible to obtain:

G. SMITH - Tutor
 (12/11/98)
 T. OSMAN
 T. FLEMING
 A. KENNEDY
 I. FOSTER

(26/11/98)
 D. GREEN
 E. INCE
 C. BURNIP

This is done by following the relations through from Table A to the junction Table (TIMETABLE) through to Table B, i.e. one tutor can have many students.

Working the other way round would (in the case of T. OSMAN) produce

T. OSMAN
 12/11/98 G. SMITH
 26/11/98 S. MORTON

i.e. one student has many tutors.



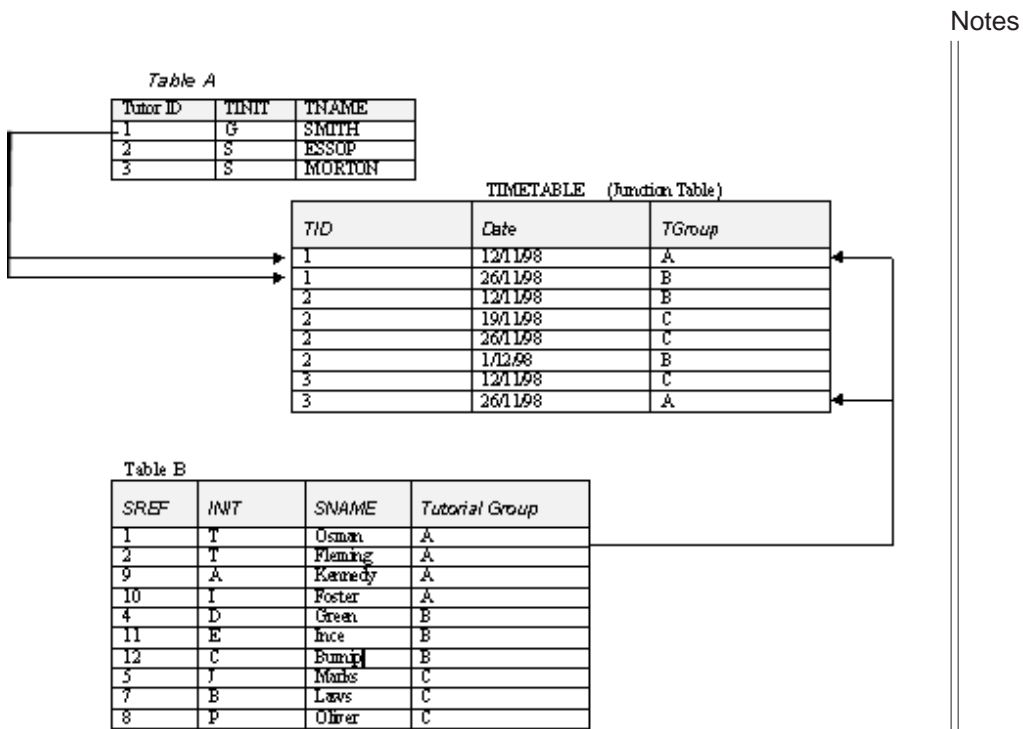


Figure 6.10: Implementing a Many to Many relationship using a JUNCTION Table



Setting Up a Relationship Between the CARS table and the STUDREC table

There are now two data tables (CARS and STUDREC), how do we establish relationships between them?

Again, it is instructive to work through a concrete example. The task is to construct a relational design that will enable us to flexibly list Students, Cars and, most importantly, Students and the cars they own.

☞ Ensure that the database window is open (in this case it doesn't matter which tag is selected)

☞ Click on the RELATIONSHIPS button (it's in the toolbar at the top of your screen)

This will display the "RELATIONSHIPS" window, which should be empty at this stage.

We now need to add the required tables to this window.

☞ Click on the SHOW TABLE icon (again, this is located in the toolbar at the top of your screen)

This will display a SHOW TABLE window similar to Figure 6.11.

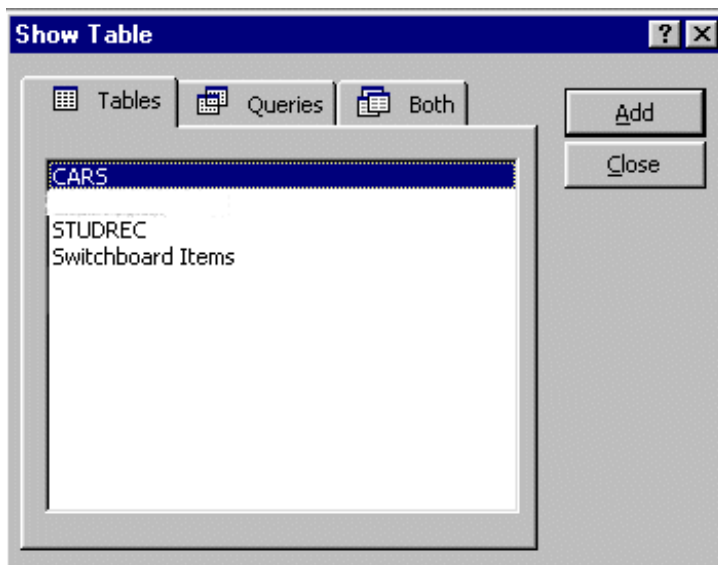


Figure 6.11: The SHOW TABLE window



Let's make a decision that we are going to build a query that will list all students, together with the cars that they own (or do not own in many cases). Any cars without owners will not be listed. The technique for achieving this involves first of all defining which tables we are going to use, defining the relationships between them, and then basing a query on the relationship.

We may then choose to build a report based on the query.

- ☞ Select the **STUDREC** table in the **SHOW TABLE** window and click on the **ADD** button.
- ☞ Select the **CARS** table and click on the **ADD** button
- ☞ Click on the **CLOSE** button to close the **SHOW TABLE** window

This should leave you with a screen similar to Figure 6.12.

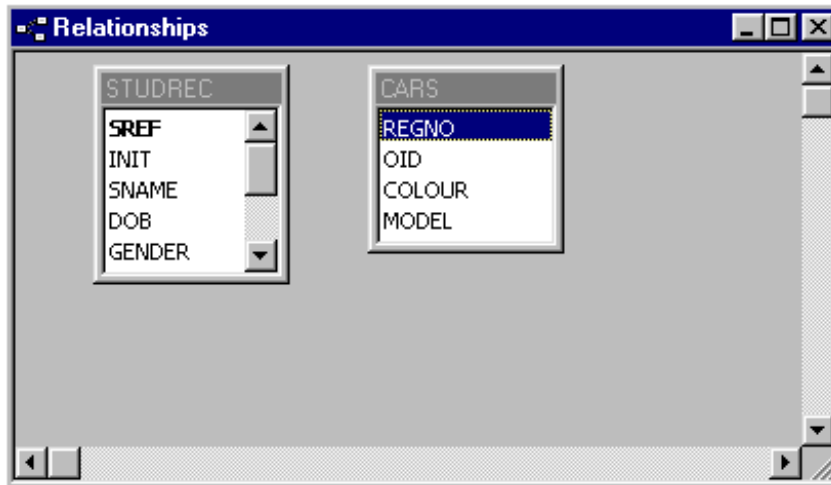


Figure 6.12: Relating STUDREC & CARS (1)

We now need to define how the two tables are to be related (cross referenced). We need to be aware of two special types of field, a **KEY FIELD** and a **FOREIGN KEY**. We've covered key fields in detail and we know that they are unique identifiers (SREF is the key field for STUDREC and REGNO is the key field for CARS). What is OID?



OID is an example of a foreign key. Under relational rules (designed to protect the integrity of the database) OID can only contain a value that matches an existing value of SREF in STUDREC. Similarly, we can't let OID contain a value that doesn't exist in the SREF field of STUDREC, if we did allow that we would corrupt the database (we would be looking for a non-existent owner). In principle we could have a NULL value for the OID foreign key, in our example we could have a car that was abandoned in the car park and we might not be able to trace the owner (we do have such a car, the black Trabant).

When we cross reference the two tables we will use the key field in STUDREC and cross reference it (relate it) to the CARS table via the OID field.

- ☞ Point to the SREF field in STUDREC with the mouse
- ☞ Click on the SREF field and drag the mouse over to the OID field in the CARS table (release the mouse button)

A second RELATIONSHIPS window should appear (Figure 6.12)

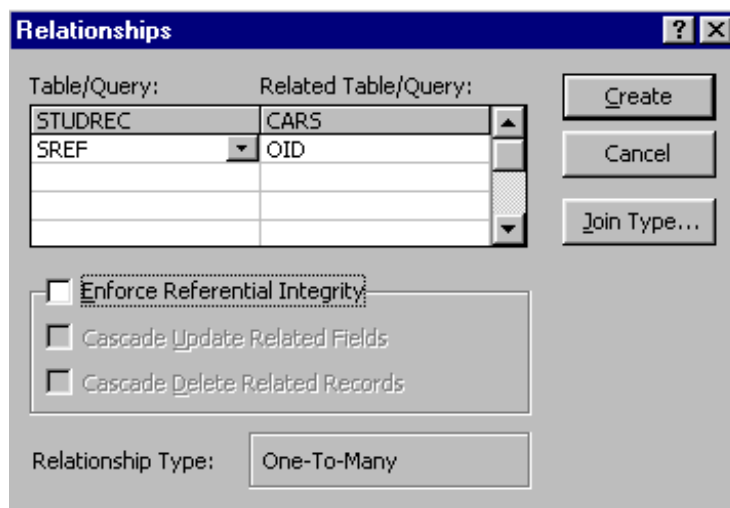


Figure 6.12: Defining the relationship

The window shows us that we have related (joined) the two tables (STUDREC & CARS) using the fields SREF & OID, we can if we wish change the fields at this stage (we don't wish to).



JOINS

At this point it would be useful to introduce more relational terminology. A join is a relational algebraic operator. Its function is to merge together different sets of data (in our case different tables or relations as they are more properly known in relational terminology). A JOIN operator will work on two tables (A & B) and combine them to produce a logical table (C) that contains an amalgamation of the data held in table A and table B. Exactly what data table C contains depends upon the type of JOIN used.

There are three common types of JOIN:

INNER JOIN
LEFT JOIN
RIGHT JOIN

An inner join (sometimes known as an equijoin) will join rows (records) in tables A & B only where the values of the related fields are equal. In other words, a student in STUDREC will be listed only if they own a car and a car will be listed only if it has an owner (see Figure 6.13).

A left join is where all records (rows) for the first table (Table A) are listed together with only those records in table B that have a matching value (see Figure 6.14).

A right join is where all records (rows) for the second table (Table B) are listed together with only those records in Table A that have a matching value (see Figure 6.15).



Notes

SREF	INIT	SNAME	REGNO	OID	COLOUR	MODEL
1	TJ	OSMAN	CUA 579 C	1	WHITE	CAVALIER
1	TJ	OSMAN	F 28 CWE	1	WHITE	HONDA ACCORD
1	TJ	OSMAN	TJO 1	1	RED	FERRARI
1	TJ	OSMAN	TJO 2	1	GREEN	RANGE ROVER
1	TJ	OSMAN	N 609 UWJ	1	MERCURY	CARINA
2	S	LANGLEY	GXN 732 Y	2	BLUE	ESCORT
3	H	WILSON	ABC 123	3	WHITE	ESCORT
4	J	CARTER	D 111 ABC	4	RED	GOLF
4	J	CARTER	H 123 WW	4	YELLOW	2CV
5	A	JONES	G 123 NC	5	BLACK	FIESTA
10	H	JACKSON	UWJ 189 Y	10	RED	MAZDA

Figure 6.13: STUDREC & CARS inner join

SREF	INIT	SNAME	REGNO	OID	COLOUR	MODEL
1	TJ	OSMAN	CUA 579 C	1	WHITE	CAVALIER
1	TJ	OSMAN	F 28 CWE	1	WHITE	HONDA ACCORD
1	TJ	OSMAN	TJO 1	1	RED	FERRARI
1	TJ	OSMAN	TJO 2	1	GREEN	RANGE ROVER
1	TJ	OSMAN	N 609 UWJ	1	MERCURY	CARINA
2	S	LANGLEY	GXN 732 Y	2	BLUE	ESCORT
3	H	WILSON	ABC 123	3	WHITE	ESCORT
4	J	CARTER	D 111 ABC	4	RED	GOLF
4	J	CARTER	H 123 WW	4	YELLOW	2CV
5	A	JONES	G 123 NC	5	BLACK	FIESTA
6	S	ISHMO				
7	K	ARNOTT				
8	B	ARNOTT				
9	N	GREEN				
10	H	JACKSON	UWJ 189 Y	10	RED	MAZDA
11	A	ARNOTT				
12	N	HEY				
13	K	WILSON				
14	J	BROWN				
15	A	ARNOTT				
16	G	WHITE				
17	J	GREEN				
18	J	GREEN				
19	F	WATSON				
20	L	HARVEY				
21	T	MOSLEY				
22	J	POWERS				
23	J	CHESTER				
... and more...						

Figure 6.14: STUDREC & CARS left join



							Notes
SREF	INIT	SNAME	REGNO	OID	COLOUR	MODEL	
10	H	JACKSON	UWJ 189 Y	10	RED	MAZDA	
1	TJ	OSMAN	CUA 579 C	1	WHITE	CAVALIER	
1	TJ	OSMAN	F 28 CWE	1	WHITE	HONDA ACCORD	
2	S	LANGLEY	GXN 732 Y	2	BLUE	ESCORT	
3	H	WILSON	ABC 123	3	WHITE	ESCORT	
4	J	CARTER	D 111 ABC	4	RED	GOLF	
5	A	JONES	G 123 NC	5	BLACK	FIESTA	
1	TJ	OSMAN	TJO 1	1	RED	FERRARI	
4	J	CARTER	H 123 WW	4	YELLOW	2CV	
1	TJ	OSMAN	TJO 2	1	GREEN	RANGE ROVER	
1	TJ	OSMAN	N 609 UWJ	1	MERCURY	CARINA	
			SAD 1		BLACK	TRABANT	

Figure 6.15: STUDREC & CARS right join



Back to the Practicalities of Defining a Relationship in Access

If you've been reading the last few pages whilst working through the practical example, you should have Figure 6.12: *Defining the Relationship* displayed on your screen. Let's pick things up from there.



Click on the JOIN TYPE button

This will give you the JOIN PROPERTIES window shown in Figure 6.16.

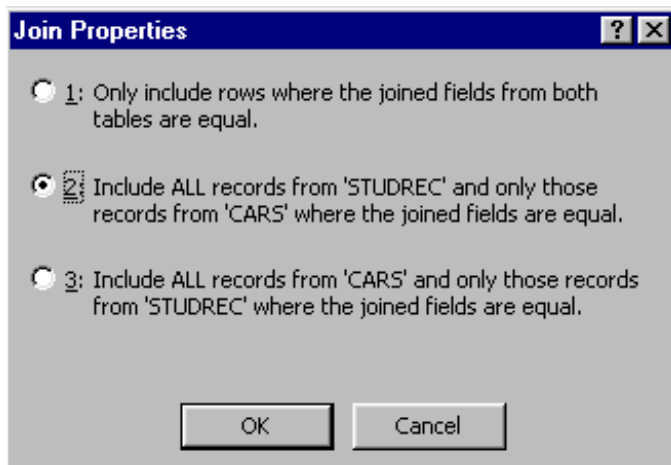


Figure 6.16: The JOIN PROPERTIES window

We are going to list all students (whether or not they own a car) together with all cars (alongside their owners). This means that we need to select a left join.

Options 1, 2 & 3 correspond to inner join, left join and right join respectively.



Select option 2



Click on the OK button

This will close the JOIN PROPERTIES window and return to the RELATIONSHIPS window.



There are three other options to be considered, whether to enforce referential integrity and if we do, whether to cascade updates or deletes.

Referential integrity refers to the rules that a relational database will enforce when joining tables. There are three points to bear in mind:

A foreign key **MUST** have either a matching value in the referring (primary) table **OR** be NULL.

It is not permitted to delete a record (row) in the primary table if there are matching records in the secondary table. If we attempted to delete the record for T. OSMAN in the (primary) table STUDREC that was related to a secondary table (CARS), we would be unable to do so because that would mean leaving a number of cars with an undefined owner.

It is not permitted to change the value of the key field in the primary table if there are matching values in the secondary table. If we tried to change the value of SREF for T. OSMAN from 1 to 101, we would not be allowed to because that would again have the effect of leaving a number of cars (all with an OID value of 1) with an undefined owner.

If we choose to tick the check box to require Access to enforce referential integrity, we are faced with two other choices, **CASCADE UPDATED RELATED FIELDS** and **CASCADE DELETE RELATED FIELDS**.

If the **CASCADE UPDATED RELATED FIELDS** box is ticked, any change to a key field in the primary table will be cascaded into the secondary table. If the value of SREF for T.OSMAN is changed from 1 to 101, then all matching values of OID in the CARS table will also be changed from 1 to 101.

If the **CASCADE DELETE RELATED FIELDS** box is ticked when a record is deleted in the primary table, all matching records in the secondary table are also deleted. If the record for T.OSMAN were to be deleted from the STUDREC table, then so would all of the records in CARS that have an OID value of 1.

We will play safe and enforce referential integrity but not allow any cascades.



☞ Tick the **ENFORCE REFERENTIAL INTEGRITY** box and press the **CREATE** button.

Your screen should be similar to Figure 6.17: Defining the relationship between **STUDREC** & **CARS**.

If you get a warning message that reads “**Relationship must be on the same number of fields with the same data types**” you will need to edit the underlying structure of your data tables (either **CARS** or **STUDREC** or both depending on what’s wrong). The most likely cause of such an error is that the data type of **SREF** in the table **STUDREC** is of a different data type to the field **OID** in the table **CARS**. Both fields ought to be **NUMBERS** and they should both be defined as **INTEGERS**.

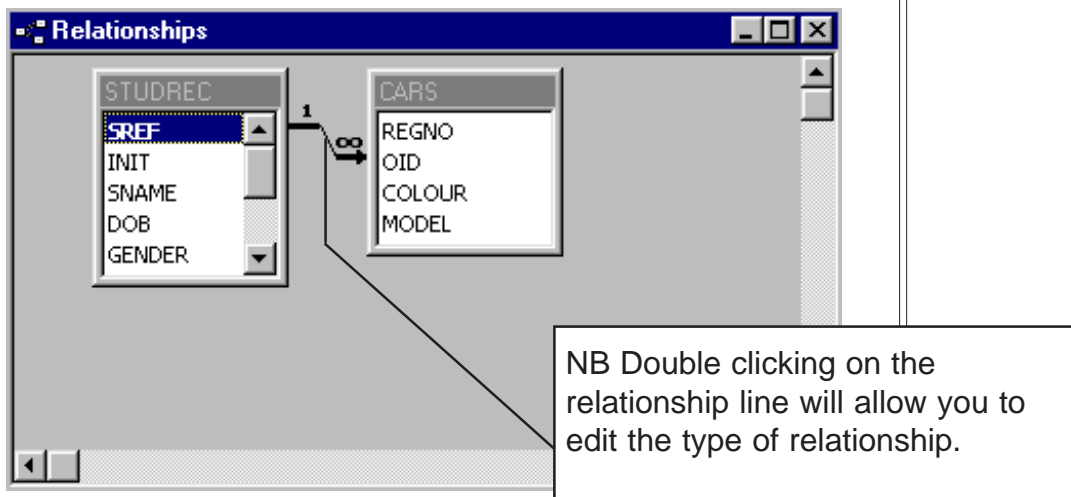


Figure 6.17 Defining the relationship between **STUDREC** & **CARS**

☞ Close the Relationships window

*Building a query on the **STUDREC/CARS** relationship*

How do we use the relationship we have just defined?

The first step (in Access) is to build a query. We are going to build a query called **CAR OWNERS**.



- ☞ Select the **QUERY** tag of the database window
- ☞ Click on the **NEW** button
- ☞ Select the **DESIGN VIEW** (no wizards this time)

You should see a **SHOW TABLE** window similar to Figure 6.18.

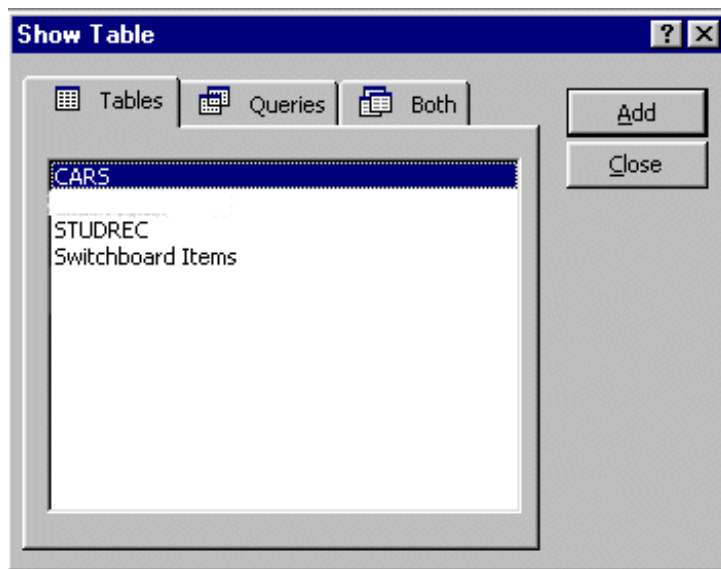


Figure 6.18: The **SHOW TABLE** window

- ☞ Highlight the **STUDREC** table and click on the **ADD** button
- ☞ Add **CARS** by highlighting the **CARS** table and clicking on the **ADD** button
- ☞ Close the **SHOW TABLE** window

Your screen should be similar to Figure 6.19. As the **CARS** table was added to the query design, the relationship that we have defined is automatically implemented. If we define another relationship later as we progress through the building of the Student Records Database, we don't need to worry about losing the current definition because that will be preserved with the query when we save it.



Notes

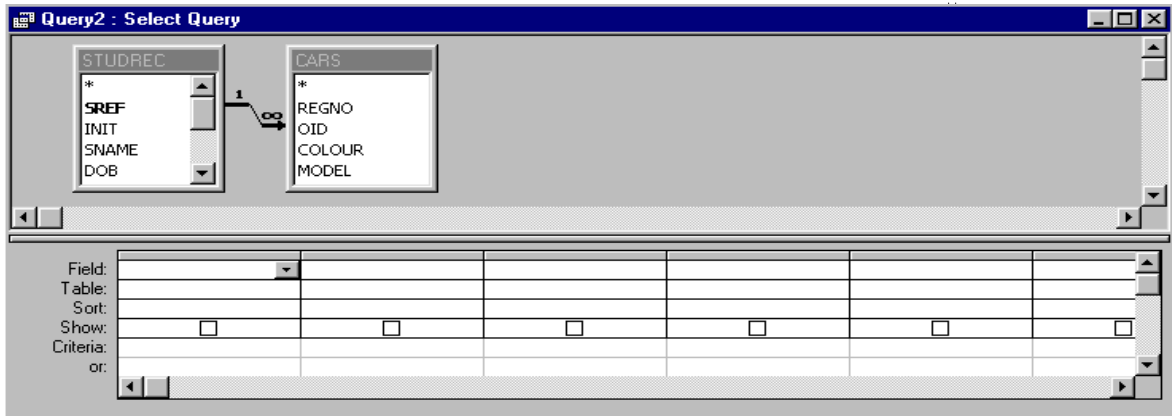


Figure 6.19: Basing a query on a relationship

We now need to add the required field to the query. This can be done either by using the drop down arrows in the “Field” row of each column, or by dragging each field from the tables in the top part of the window and dropping them in the appropriate field/ column intersection in the lower part of the window.

- ☞ Refer to Figure 6.20 and insert these fields into your query
- ☞ Save the query as “CAR OWNERS” (no quotation marks)

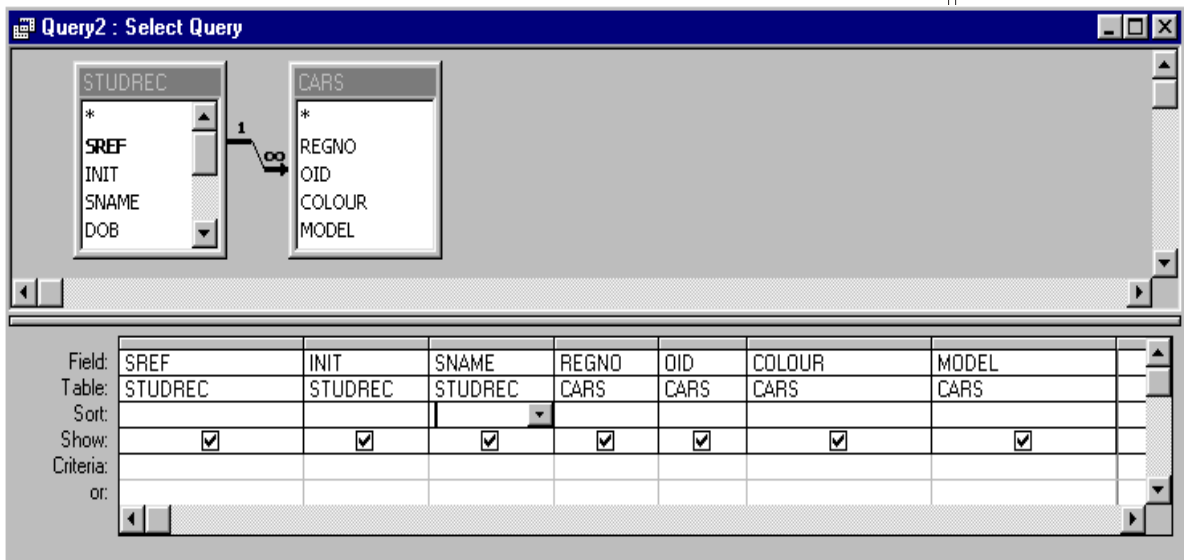


Figure 6.20: Defining which fields to use

- ☞ Switch to VIEW mode and inspect the results of your work to date. (You should see the listing that is shown in Figure 6.14: *STUDREC & CARS left join*)



Exercise 6

Question 1:

Submit for marking a printout of the results of joining STUDREC to CARS with:

- 1: INNER JOIN
- 2: LEFT JOIN
- 3: RIGHT JOIN

Ensure that you mark your papers clearly with the words "INNER JOIN", "LEFT JOIN" and "RIGHT JOIN" as appropriate.

Question 2:

Consider the following report (CAR PARK DETAILS). Produce a similar report (it will be based on LEFT JOIN).

Save your report as "CAR PARK DETAILS"

Please keep in mind that the aim of the exercise is to test if you can independently produce a report that is based on a query that is in turn based on a relationship.



Notes

CAR PARK DETAILS

SREF	INIT	SNAME	REGNO	OID	COLOUR	MODEL
1	TJ	OSMAN				
			CUA 579	1	WHITE	CAVALIER
			F 28 CWE	1	WHITE	HONDA ACCORD
			N 609	1	MERCURY	CARINA
			TJO 1	1	RED	FERRARI
			TJO 2	1	GREEN	RANGE ROVER
2	S	LANGLEY				
			GXN 732	2	BLUE	ESCORT
3	H	WILSON				
			ABC 123	3	WHITE	ESCORT
4	J	CARTER				
			D 111	4	RED	GOLF
			H 123	4	YELLOW	2CV
5	A	JONES				
			G 123 NC	5	BLACK	FIESTA
6	S	ISHAMO				
7	K	ARNOTT				
8	B	ARNOTT				
9	N	GREEN				
10	H	JACKSON				
			UWJ 189	10	RED	MAZDA
11	A	ARNOTT				
12	N	HEY				
13	K	WILSON				
14	J	BROWN				
15	A	ARNOTT				



Notes

SREF	INITSNAME	REGNO	OID	COLOUR	MODEL
35	H	KNIGHT			
36	I	ARNOTT			
37	A	WHITE			
38	G	DAVIDSON			
39	U	CAPRA			
40	R	AKBAR			
41	S	OSMAN			

Total Number of Cars = 11

14 December 2001 Page 3 of 3

Note: Page 2 of this report is omitted to conserve space - no students who are listed on this page own a car.

