University of BRISTOL

IT SERVICES

Access 2013: Intermediate to advanced queries

Practical workbook

Aims and Learning Objectives

This document shows you how to do things with queries that take you slightly 'beyond the norm'. It is not supposed to be exhaustive – no Access query workbook ever could be – but it shows you some of the less-trodden paths. The intention is to give you some hints and tips as to what can be done with queries so as to make you experiment further on your own.

The core learning objectives for this course are to:

- explain what you can do with Access queries beyond viewing subsets of data;
- create queries using a range of operators, functions and expressions;
- create a query to calculate totals;
- customise a crosstab query in the Query Design window;
- use different table joins to alter query results;
- create queries to sort data into ranges (e.g. age: 1 – 10, age: 11 – 20 and so on);
- join two sets of query results together using SQL.

Document information

Course files

This document and associated practice file are available on the web. To find these, go to http://www.bristol.ac.uk/it-services/learning/resources and in the **Keyword** box, type the document code given in brackets at the top of this page.

Related documentation

Other related documents are available from the web at:

http://www.bristol.ac.uk/it-services/learning/resources

Contents

Introduction

This document takes you through some intermediate to advanced queries. The assumption is that you are able to build simple one and multi-table queries using wildcards, such as *, and simple operators, such as =, > and so on.

The further assumption is made that you understand how to create both AND and OR queries – for example creating a query that must match multiple criteria at the same time or match one of two (or more) sets of criteria.

The yet further assumption is that you know how to create basic expressions, parameter and concatenation queries.

If you haven't got a clue what all this means, then work through the documents listed below in the **Prerequisites** section before attempting to work through this document.

.

Prerequisites

You will get more out of this document if you have done one of the following:

- Attended "Access 2013: An introduction" (ITS-ACC13-1) and "Access 2013: Building Access databases" (ITS-ACC13-2).
- Worked through the course documents for the above two courses. These documents can be found at http://www.bristol.ac.uk/it-services/learning/resources/. For the 'An introduction' document, type acc-45 in to the search box. For the 'Building' document, type acc-46 in to the search box.

Task 1 Opening the practice database

Objectives To open the database and look at tables, relationships and data types.

Comments It is a good idea to become familiar with both the database relationships and data/data types before trying to do complex things with the data.

1.1 To open the practice database:

- Navigate to wherever you saved the practice file.
- Double-click on the file name to open the database.
- On the **Security Warning** message line, select **Enable Content**.
- (If a further **Security Warning** opens asking if you want to make the file a "Trusted Document", select **No**.)

1.2 To get to know the database:

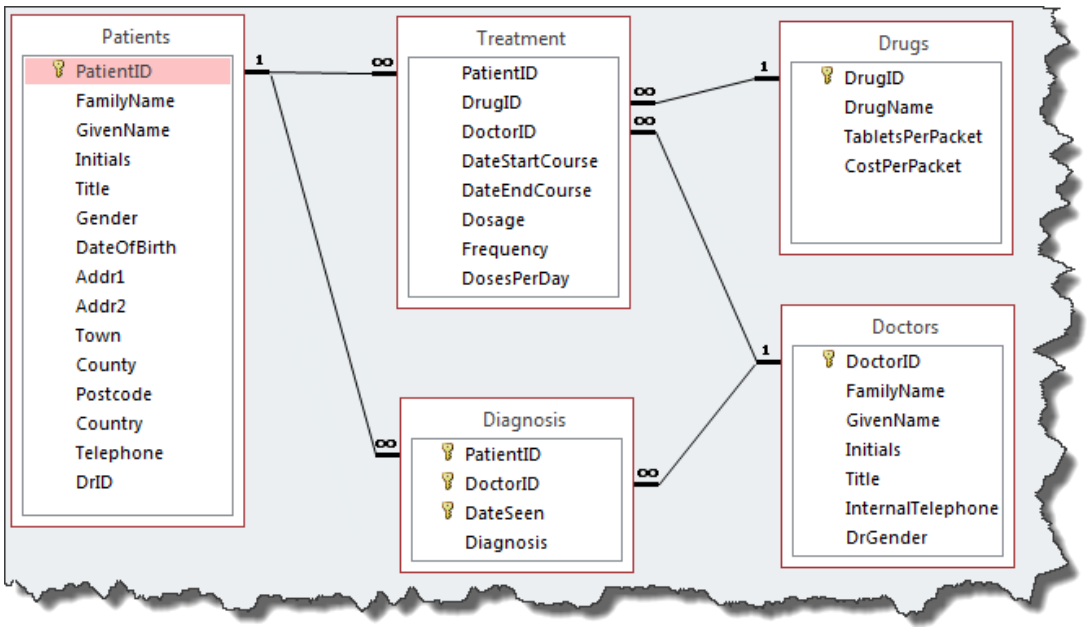


Figure 1 – tables and relationships in the practice database

- Open up each table in turn and look at the data each contains.
- Work out the different data types, text (aligns left), numeric (aligns left), yes/no (tickbox), date and time and so on
- Look at Figure 1 above to identify primary keys (1), foreign keys (∞) and the relationships between tables.
- Close all tables when finished.

Task 2 Query wizards

Objectives To use the Find Unmatched Query Wizard and the Crosstab Query Wizard.

Comments See Unmatched queries in Task 14 for other ways of finding unmatched data.

2.1 To display subjects where there are no students:

- In the **Create** tab of the **Ribbon**, select **Query Wizard**.
- Click **Find Unmatched Query Wizard** and **OK**.
- Select **Table: Patients**, followed by **Next**.
- Select **Table: Diagnosis** as the referring table, followed by **Next**.
- Make sure that **Patient ID** is highlighted on each side, and click **Next**.

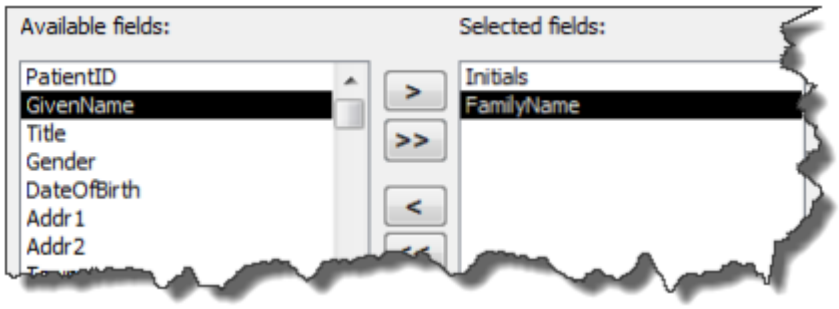


Figure 3 – you should choose these two fields

- Select **Initials** and **FamilyName**, then click on **Next** and **Finish**.
- After viewing the query results look at the query in **Design View** to see how it has achieved this result (look at the join between the two tables – more will be revealed in Task 14) then close the query.

Crosstab query using the Crosstab Query Wizard

Crosstab queries show your data with both row and column headings based on the values in two selected fields. This is a good way to summarise large amounts of data: for example, viewing the number of male and female patients from each town.

2.2 To create a crosstab query using the Crosstab Query Wizard:

- In the **Create** tab of the **Ribbon**, select **Query Wizard**, followed by **Crosstab Query Wizard**, then click **OK**.
- Select **Table: Patients** and click **Next**.
- Double-click on **Town** as the row heading (since it has more values than **Gender**) and click **Next**.
- Now select **Gender** as the column heading and click **Next**.
- Click once on **PatientID** in the **Field** box and click once on **Count** in the **Functions** box and then click **Next**.
- Name it ***PatientCrosstab*** and click **Finish** to view the results.

Task 3 Simple expressions and functions

Objectives To create expression queries that create new fields based on calculations or of joining two or more fields together. To create a simple function query that turns text into uppercase.

Comments When you use an expression to create a new field, you have to give the new field a name, otherwise it is arbitrarily named as Expr1, Expr2 or similar.

Numerical expression

3.1 To create a numerical expression that works out how much each tablet costs:

- Create a new query and add **DrugName**, **TabletsPerPacket** and **CostPerPacket** from the **Drugs** table.
- In the next field enter ***[CostPerPacket]/[TabletsPerPacket]*** and click outside the field where you've just entered the text – notice that Access adds **Expr1:** in front of what you have just entered.

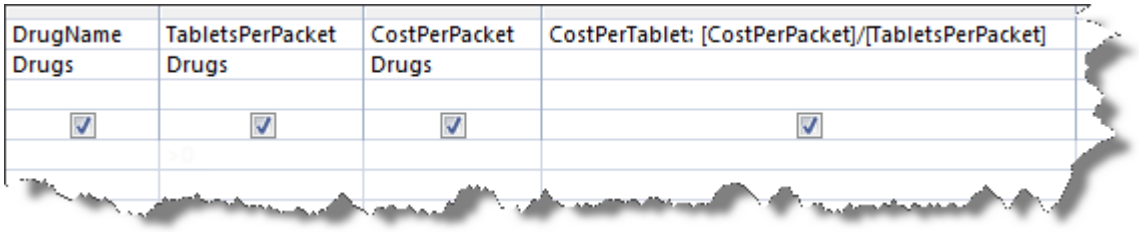


Figure 2 – the finished query

- Alter **Expr1:** to ***CostPerTablet:*** and view the query results.

Note With numerical expressions if a record has a null value or a zero for one of the fields used in the calculation, then no result is shown for that particular record.

Set properties on the above expression

3.2 To set properties on the above expression:

- In **Design View**, right-click the **CostPerTablet** column, open the **Property Sheet**.
- In the **General** tab, change **Format** to **Currency** and **Decimal Places** to **7** and look at the result of the query.
- Save the query as ***Task2Query1*** and close it.

Note If the number of decimal places is set to 2, the cost is rounded up.

Concatenation expression

3.3 To create a concatenation expression that joins first and last name together:

- Create a new query and add **GivenName** and **FamilyName** from the **Patients** table.
- Sort by **FamilyName** ascending and remove the tick from the **Show** tickbox.

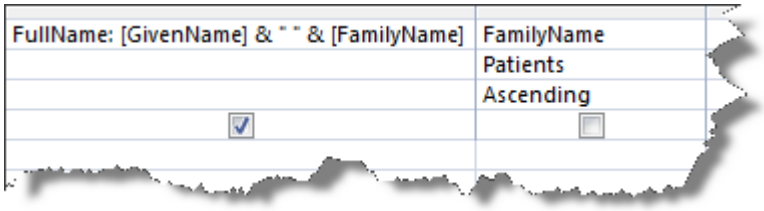


Figure 3 – the finished query

- Update the **GivenName** field so that it reads ***FullName: [GivenName] & " " & [FamilyName]***.
- Run the query to see the results, then close, saving the query as ***Task2Query2***.

Note The & symbol joins one thing to another and the two speech marks mean that there is some text involved – the text in this case is the space that we want to put between the two fields. The first & joins GivenName to the space and the second & joins the space to FamilyName. Note that FamilyName is used as a sort field, but does not show when you run the query.

Date expressions

3.4 To use date expressions in a query:

- Create a new query based on the **Patients** table and add **GivenName**, **FamilyName** and **DateOfBirth** – then add **DateOfBirth** again.
- Alter the second **DateOfBirth** to ***DateOfBirth +7***.
- Change **Expr1:** to ***NextWeek:*** (don't delete the colon) and view the results.
- In the next column type ***NextYear:[NextWeek] +365*** and view the query results.
- In the design grid, right-click on the **Field** row for **NextWeek** and open **Properties…** to look at the display options under **General** / **Format**.
- Select several and look at the way these display the date when you run the query.

Simple functions

3.5 To convert the text in a field to uppercase using the UCase function:

- Create a new query and add **Town** from the **Patients** table.

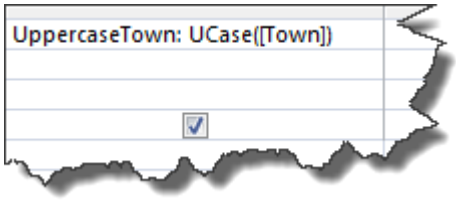


Figure 4 – the finished query

- Alter the **Town** field so that it reads ***UppercaseTown: UCase([Town])***.
- **Run** the query to see the results, then close, saving the query as ***Task2Query3***.

3.6 To capitalise/un-capitalise words in a field using the Strconv function:

- Open up the **Patients** table and in **Addr1** change **The Old Vicarage** to ***the old vicarage***, then close the table.
- Create a new query based on the **Patients** table and add **Addr1**, then alter this so that it says ***strconv(Add1, 1)*** to the grid and look at the results.

- Change the final **1**, first to ***2*** and then to ***3***, and check the results each time.

Note See the training document, "Access: using operators and functions in queries" (acc-4) to see what other functions you can use.

Make a sentence using a concatenation query

3.7 To practice creating concatenation queries:

Note Refer back to Figure 1 to check table and field names.

- From the **Drugs** table, concatenate **DrugName** and **TabletsPerPacket**.
- Use ***&*** and ***“”*** with relevant text so that the query result looks the same as below.

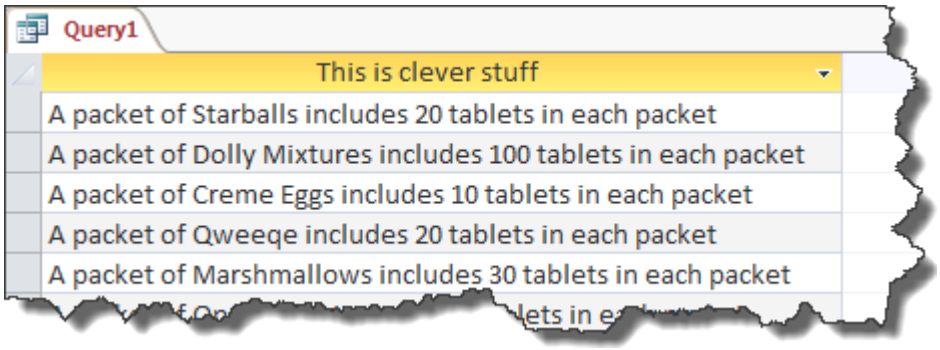


Figure 5 – create the above concatenation query on your own!

Task 4 Grouped queries

Objectives To create two grouped queries, one of which matches against specific criteria and to create a grouped multi-table query with an added function.

Comments You can do sophisticated calculations such as working out the number of patients that come from each town – which we will see in the first example below.

Calculation on a group

4.1 To find out how many patients come from each town, ordered by largest population followed by town name:

- Create a new query and add **Town**, **PatientID** and **Town** (again) from the **Patients** table.
- Select the **Total** icon from the **Ribbon** and in the **Totals** row for the first **Town** select **Group By**.
- Update **PatientID** so that it says ***NumberOfPatients: PatientID***, select **Count** from the **Total** row and sort descending.

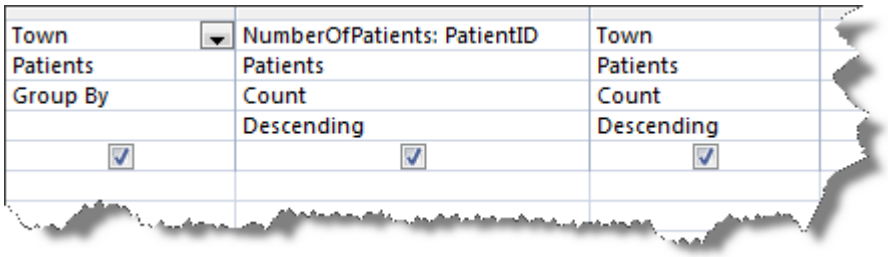


Figure 6 – the finished query

- In the **Totals** row for the second **Town** select **Count** and sort descending.
- **Run** the query to see the results, then close, saving the query as ***Task3Query1***.

Note Group By finds the unique values of town. The Count function on Patient ID divides the patients into groups based on the town in which they live. Count is applied on the Primary key (PatientID) to give the number of patients for each town including those patients where the town is unknown (notice that the count on Town shows 0 for unknown towns, so if you want to get a count of how many unknown values there are, count on the primary key).

Setting criteria on aggregates

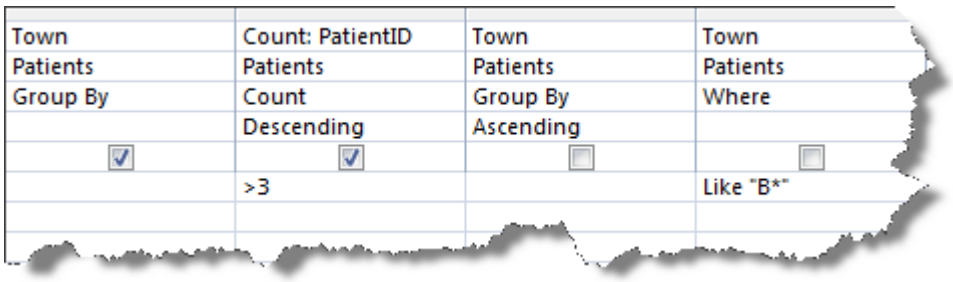
4.2 To find towns beginning with "B" and which have more than three patients:

- Create a new query and add **Town**, **PatientID**, **Town** (again) and **Town** (yet again) from the **Patients** table.

Note Town appears three times because the ordering is done on the Count and then the town, but the town is to be displayed first. Town then appears so that the criterion of towns starting with the letter "B" can be set.

- Select the **Total** icon from the **Ribbon** and in the **Totals** row for the first **Town** select **Group By**.
- Update **PatientID** so that it says ***Count: PatientID***, select **Count** from the **Total** row, sort descending and enter ***>3*** in the **Criteria** row.

- In the **Totals** row for the second **Town** select **Group By**, remove the tick from the **Show** tickbox and sort ascending.



Town	Count: PatientID	Town	Town
Patients	Patients	Patients	Patients
Group By	Count	Group By	Where
	Descending	Ascending	
☑	☑	☐	☐
	>3		Like "B*"

Figure 7 – the finished query

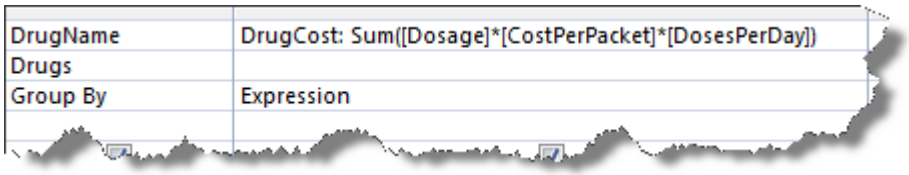
- In the **Totals** row for the third **Town** select **Where**, remove the tick from the **Show** tickbox and add ***B**** in the **Criteria** row.
- **Run** the query to see the results, then close, saving the query as ***Task3Query2***.

Grouped two table query (with a function for good measure!)

4.3 To create a grouped two-table query with function to work out the cost per drug:

- Create a new query and add **DrugName** from the **Drugs** table and **Dosage** from the **Treatment** table.
- Select the **Total** icon from the **Ribbon** and in the **Totals** row for the first **DrugName** select **Group By**.
- Update **Dosage** so that it says:

  DrugCost: Sum([Dosage]*[CostPerPacket]*[DosesPerDay]).



DrugName	DrugCost: Sum([Dosage]*[CostPerPacket]*[DosesPerDay])
Drugs	
Group By	Expression

Figure 8 – the finished query

- In the **Totals** row for **DrugCost** select **Expression**.
- **Run** the query to see the results, then close, saving the query as ***Task3Query3***.

Task 5 Relational operators

Objectives To view how SQL handles usage of several relational operators.

Comments We have already used Boolean operators such as AND and OR. Relational operators such as >, <, In, Between and Like are used to compare one field value with a value.

Relational operators

Operator(s)	Description	Example(s)
=, <, <=, <>, >, >=	Test for equality, greater than, less than, not equal to and so on	>10 <>"A"
Between And	Tests whether a value falls within a range	Between 0 And 10
Like	Tests whether the entry in a text field matches a pattern – it allows comparisons between strings using wild card characters and so is different to = • * represents any number of any character • ? represents a single character • # represents a single digit • [A - S] represents any character between A and S • W[ae*] – means starting with W followed by a or e, then any other characters • W[!ae]* starting with W and not followed by a or e • *[*]* contains an asterisk • *[?]* contains a question mark	Like "Ch*"
In	Tests whether an entry is an item in a set	In (50, 100, 200) In ("Bristol", "Bath")
Not	Negates an expression	Not <100
Is Null	Tests whether a field is empty (used as a selection criterion)	Is Null
Is Not Null	Tests whether a field is not empty	Is Not Null

The Between operator

5.1 To create a query using the Between operator on three different data type fields:

- Create a new query with **Initials**, **FamilyName**, **DateOfBirth** and **PatientID** from the **Patients** table.
- In the **Criteria** row for **FamilyName** enter ***between b and i***. Check the result.
- In the **Criteria** row for **DateOfBirth** enter ***between 1/1/1960 and 12/31/1969***. Check the result again.

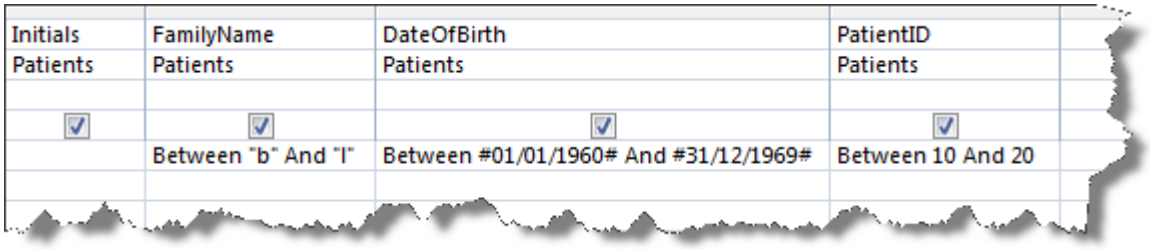


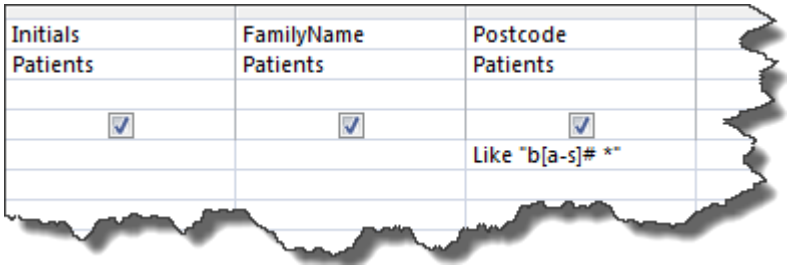
Figure 9 – the finished query

- In the **Criteria** row for **PatientID** enter ***between 10 and 20***.
- **Run** the query to see the final result, then close, saving the query as ***Task4Query1***.

The Like operator

5.2 To create a query using the Like operator with wildcards:

- Create a new query with **Initials**, **FamilyName** and **Postcode** from the **Patients** table.
- In the **Criteria** row for **Postcode** enter ***b[a-s]# ****.



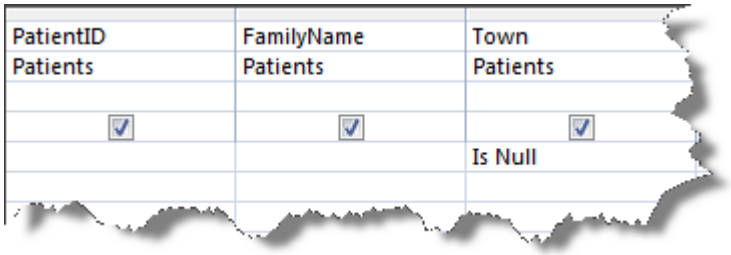
- Notice that when you click outside of the **Criteria** row, Access automatically adds ***Like*** and speechmarks.
- **Run** the query to see the results, then close, saving the query as ***Task4Query2***.

The Is Null operator

Is Null is used to test for an unknown value. Unknown values are distinguished from 0, so when averages, etc., are calculated any blank values are ignored.

5.3 To create a query using the Is Null operator to find where there is no value in the Town field:

- Create a new query with **PatientID**, **FamilyName** and **Town** from the **Patients** table.



- In the **Criteria** row for **Town** enter ***is null***.
- **Run** the query to see the results, then close, saving the query as ***Task4Query3***.

The not operator

5.4 To create a query using the Not operator:

- Create a new query based on the **Patients** table and add **FamilyName** and **Town** to the grid.
- In the **Criteria** box for **Town**, enter ***not Bristol*** to list all patients not living in Bristol.
- Run it to see the results and then close it. Don't bother to save the query.

Task 6 Parameter queries

Objectives To build a parameter query that includes a wildcard search.

Comments Parameter queries are useful if you run the same query often but have to alter the match criteria each time you run it. Instead you can get Access to ask you what you want to match against each time.

Simple parameter query

6.1 To create a simple parameter query that asks which town you want to view:

- Create a new query with **Initials**, **GivenName**, **FamilyName** and **Town** from the **Patients** table.

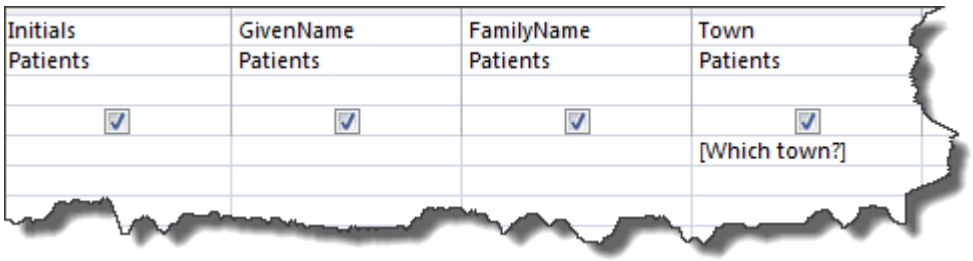


Figure 10 – enter [Which town?]

- In the **Criteria** row for **Town** enter ***[Which town?]***.
- **Run** the query to see the results – when prompted enter ***bristol***.

Note With parameter queries the prompt is enclosed in square brackets, as in [Which town?] above. It doesn't matter what words you put in the square brackets as long as you don't put in anything that Access recognises as a field name.

6.2 To add a wildcard parameter on surname to the query:

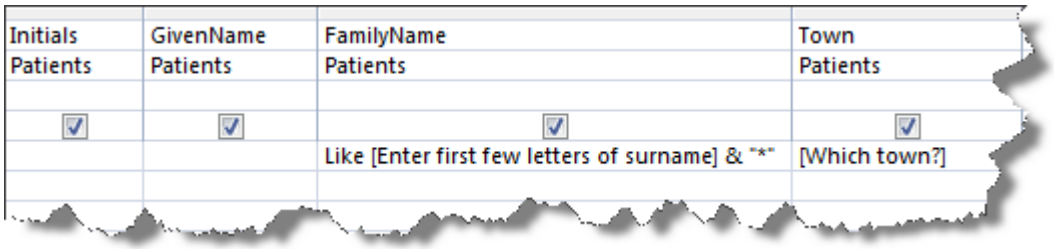


Figure 11 – the finished query

- In the **Criteria** row for **FamilyName** enter ***Like [Enter first few letters of surname] & **.*** Access will add speechmarks around the wildcard when you click outside the **Criteria** row.
- **Run** the query to see the results – when prompted enter ***bet*** in the first parameter prompt and enter ***bristol*** in the second.
- Close the query and save as ***Task5Query1***.

6.3 To practice using the parameter query:

- Open the **Patients** table so that you have an idea of the data held in the **FamilyName** and **Town** fields.
- Have a play!

Task 7 Action queries

Objectives To find out what the three types of action queries do; to create an action query that runs a calculation.

Comments Action queries can amend or delete a lot of data at the same time and so can be dangerous if you don't know what you are doing! Before running an action query that changes or deletes data, you are advised to save a backup copy of your database.

Type	Description
Delete	This type of query deletes records from a table, based on specified criteria.
Append	This type of query adds one or more records by specifying values or records from one or more tables to another table.
Update	This type of query changes the data in a specified group of records. Update can only be used on one table at a time.

Update query

7.1 To create an update query that increases drug costs by 10% where there are more than 50 tablets in a packet:

- Create a new query with **CostPerPacket** and **TabletsPerPacket** from the **Drugs** table.
- Click on the **Update** icon in the **Ribbon** and notice that a new row has appeared in the query window.

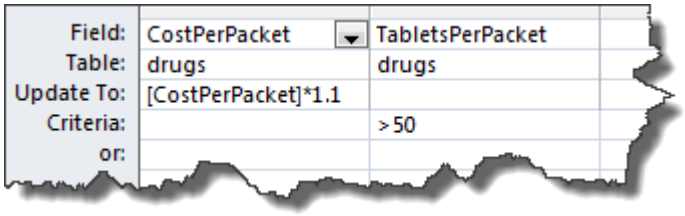


Figure 12 – the finished query

- In the **Update To** row for **CostPerPacket** enter ***[CostPerPacket]*1.1*** and in the **Criteria** row for **TabletsPerPacket** enter ***>50***.
- Look at the query in **Datasheet View** (this shows you what will be updated when you run the query, whereas selecting **Run** updates the data)

- Now **Run** the query then look again at **Datasheet View**.
- Now **Run** the query again then look again at **Datasheet View** – what is the query doing?
- Close the query and save as ***Task6Query1*** (because it is a different type of query to normal (select) queries, you will find it at the bottom of the **Navigation Pane** under the union query – depending on type, some appear at top and some at bottom).

Warning! It is a very good idea not to run update or other action queries until you know what they do – always open first in design view to work out what the action query does.

Task 8 The iif function

Objectives To create basic iif function queries followed by a grouped iif function query.

Comments The iif function requires the name of a field or expression, followed by the criteria, and the values to be returned if true and false.

Note If you use a mathematical operator to find a value and one of the fields is blank, the result is a null value. To prevent this you can use the iif function to check and produce an error message, for instance, if a date is missing from a field used in a calculation the message might ask you to check for a missing date.

iif query

8.1 To create an iif query to create a new field to show full gender (e.g. “Male” instead of “M”):

- Create a new query and include **Gender**, followed by **Gender** again and **GivenName** from the **Patients** table.

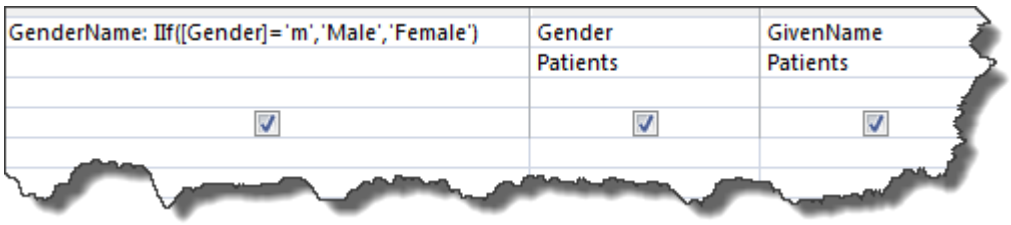


Figure 13 – the finished query

- Amend the first **Gender** field so that it says ***iif([Gender]='m','Male','Female')***.
- Click in the row underneath then amend **Expr1:** to say ***GenderName***.
- **Run** the query to see the results, then close, saving the query as ***Task7Query1***.

8.2 To create an iif query to find null values in a field and enter text based on this:

- Create a new query and include **Town**, followed by **Town** again from the **Patients** table.

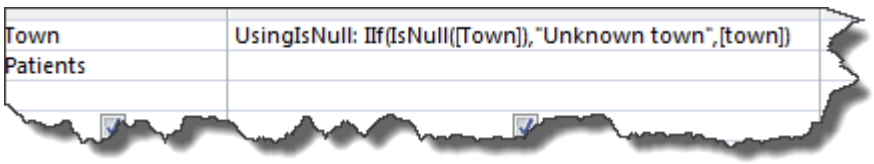


Figure 14 – the finished query

- Amend the second **Town** to read

  UsingIsNull: Iif(IsNull([Town]),"Unknown town",[town]).
- **Run** the query to see the results, then close, saving the query as ***Task7Query2***.

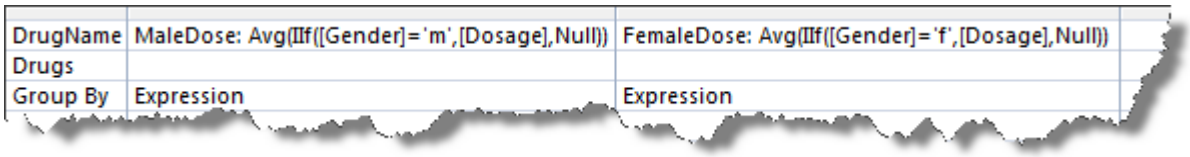
Grouped iif query

8.3 To create a grouped iif query that shows average dosage of drug per gender:

- Create a new query and select the **Patients**, **Treatment** and **Drugs** tables from the **Show Table** window.

Note Although we are only going to use fields from the Patients and Drugs tables, we need the Treatment table so that there is a relational link between the two tables.

- Add **DrugName** from the **Drugs** table, click on the **Totals** icon in the **Ribbon** and in the **Total** row select **Group By**.
- In the next field enter ***MaleDose: Avg(IIf([Gender]='m',[Dosage],Null))*** into the **Field** field and choose **Expression** from the **Total** dropdown.



DrugName	MaleDose: Avg(IIf([Gender]='m',[Dosage],Null))	FemaleDose: Avg(IIf([Gender]='f',[Dosage],Null))
Drugs		
Group By	Expression	Expression

Figure 15 – the finished query

- In the next field enter ***FemaleDose: Avg(IIf([Gender]='f',[Dosage],Null))*** into the **Field** field and choose **Expression** from the **Total** dropdown.
- **Run** the query to see the results, then close, saving the query as ***Task7Query2***.

Note This query finds, for each drug, the dosage by each gender. The clause: iif(gender,"m", dosage, null) selects the dosage if gender = "m"; otherwise the value is set to the null value and is not included in the calculation of the average. This type of query calculates an average on records then groups the result by two different ways - one down the left side (the group by clause) and the other across the top (the aggregated functions). The field aliases "MaleDose" and "FemaleDose" are used to specify the field names.

Task 9 Crosstab query

Objectives To create a query on which to base a crosstab query and to then create the crosstab query itself.

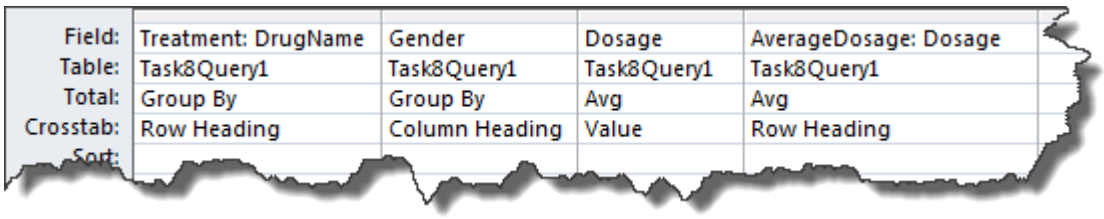
Comments Crosstabs are used to produce summaries of data.

9.1 To create the query on which the crosstab is to be based:

- Create a new query based on the **Drugs**, **Treatment** and **Patients** tables.
- Add **Gender** from **Patients**, **Dosage** from **Treatment** and **DrugName** from **Drugs**.
- **Run** the query to see the results, then close, saving the query as ***Task8Query1***.

9.2 To create a crosstab query that shows average drug dosage grouped by drug and gender:

- In the **Create** tab, select **Query Wizard** followed by **Crosstab Query Wizard**, then select **OK**.
- Select **Queries** and highlight **Query: Task8Query1**, then click on **Next**.
- Add **DrugName** when asked to choose a row heading, then click **Next**.
- Add **Gender** when asked to choose a column heading, then click **Next**.
- Leave **Dosage** with the default **Avg** and click **Next**.
- Save the query as ***Task8Query2*** when prompted and click on **Finish**.
- View the query and note that the column headings could be more user-friendly.
- Open the query in **Design View**.



Field:	Treatment: DrugName	Gender	Dosage	AverageDosage: Dosage
Table:	Task8Query1	Task8Query1	Task8Query1	Task8Query1
Total:	Group By	Group By	Avg	Avg
Crosstab:	Row Heading	Column Heading	Value	Row Heading
Sort:				

Figure 16 – the finished query

- Alter **[DrugName]** to ***Treatment: DrugName***, alter **Total Of Dosage: [Dosage]** to ***AverageDosage: Dosage***.
- You can also remove the square brackets from **Gender** and **Dosage** if you want.
- **Run** the query to see the results, then close, saving the query as ***Task8Query2***.

Task 10 Date handling

Objectives To use the Format, Day, Month, Year and DateDiff functions to format dates.

Comments See also Access: using operators and functions in queries (acc-4) at http://www.bristol.ac.uk/it-services/learning/documentation/acc-4/acc-4r.pdf.

Format function

Note The Format function specifies the date format using keywords, e.g. "dddd". The case used specifies the case of the output, for example "MMMM" outputs the month name in capital letters.

10.1 To create a query using the format function to show date elements in different ways:

- Create a new query and, from the **Patients** table, add **FamilyName** once followed by **DateOfBirth** three times.
- Amend the second **DateOfBirth** to read:

  NoYear: Format([DateOfBirth],"dddd mmmm d ").

Figure 17 – the finished query

- Amend the third **DateOfBirth** to read ***Year: Format([DateOfBirth],"yyyy")***.
- **Run** the query to see the results, then close, saving the query as ***Task9Query1***.

Another Format function example

10.2 To create a query grouped by day number that shows day name and number of patients that start their treatment on that day

- Create a new query and add **DateStartCourse** (twice) and **PatientID** from the **Treatment** table.
- Select the **Total** icon from the **Ribbon**.
- Amend the first **DateStartCourse** to read:

  DayNumber: DatePart('w',[DateStartCourse]) and select **Group By** from the **Totals** dropdown.
- Amend the second **DateStartCourse** to read:

  Day: Format([DateStartCourse],'dddd') and select **Group By** from the **Totals** dropdown.

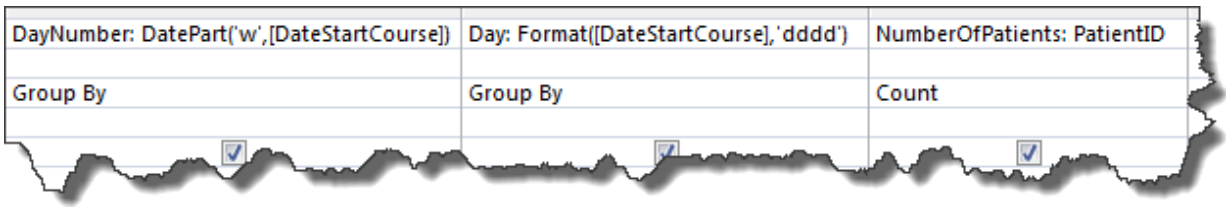


Figure 18 – the finished query

- Amend **PatientID** to read: ***NumberOfPatients: PatientID*** and select **Count** from the **Totals** dropdown.
- **Run** the query to see the results, then close, saving the query as ***Task9Query2***.

Day, Month and Year functions

10.3 To create a query using the Day, Month and Year functions:

- Create a new query and, from the **Patients** table, add **DateOfBirth** four times.

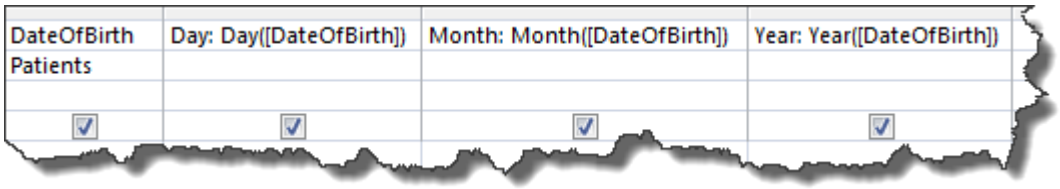


Figure 19 – the finished query

- Leave the first **DateOfBirth** as it is, but amend the other three to:

  Day: Day([DateOfBirth])

  Month: Month([DateOfBirth])

  Year: Year([DateOfBirth]) as in the screenshot above.

- Run the query to see the results, then close, saving the query as ***Task9Query3***.

DateDiff function

Note DateDiff returns the difference in the given period (for example, "m" for months) between two dates. If two dates are subtracted using the minus (-) operator, the answer is given in days. The Int function rounds the age down. Using "yyyy" as the interval does not give the correct age as only the year is considered, not the day and month.

10.4 To create a query showing several methods of using the DateDiff function:

- Create a new query and add **DateOfBirth** four times from the **Patients** table.

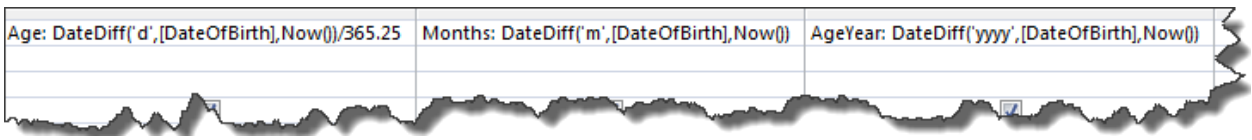


Figure 20 – the finished query (excluding the first DateOfBirth field)

- Leave the first **DateOfBirth** as is, but amend the other three to:

  Age: DateDiff('d',[DateOfBirth],Now())/365.25

  Months: DateDiff('m',[DateOfBirth],Now())

  AgeYear: DateDiff('yyyy',[DateOfBirth],Now()) as in the screenshot above.

- **Run** the query to see the results, then close, saving the query as ***Task9Query4***.

Task 11 Define your own group values

Objectives To create a new table to hold the description and upper and lower bounds of each range of values and to create a query to group data into specified ranges.

Comments Ranges can be easily changed if necessary by altering the values in the table rather than by having to amend or create a new query.

Create a table to specify age ranges

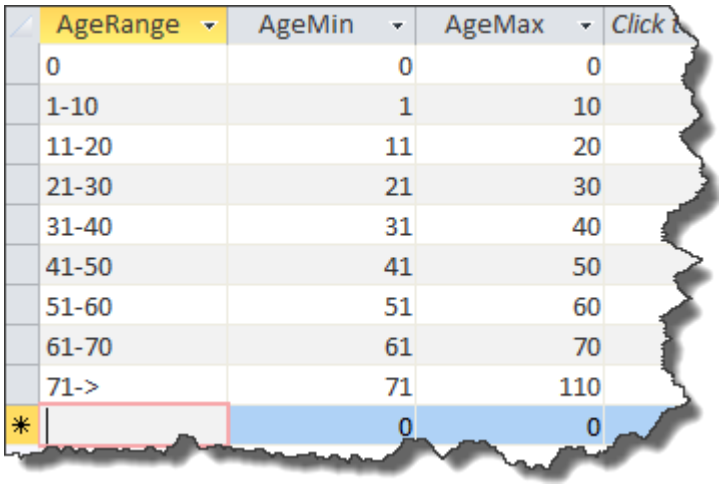
11.1 To create a table to specify age ranges:

- Create a new table with the following fields. Ensure that you apply the correct data types and field sizes.

Fieldname	Data type	Field size/format
AgeRange	Short Text	15
AgeMin	Number	Integer
AgeMax	Number	Integer

Note The AgeRange field is text so as to allow dashes and other non-numeric characters, whilst the other two fields are Integer so as to allow just whole numbers.

- Make **AgeRange** the primary key and save the table as ***AgeRange***.



AgeRange	AgeMin	AgeMax	Click t
0	0	0	
1-10	1	10	
11-20	11	20	
21-30	21	30	
31-40	31	40	
41-50	41	50	
51-60	51	60	
61-70	61	70	
71->	71	110	
	0	0	

Figure 21 – add data to the table exactly as above

- Open the table and enter data exactly as in the above screenshot.

Create a query that groups by age on the AgeRange table

11.2 To create a query that groups by age on the AgeRange table:

Note To find patient age the query uses the DateDiff function.

- Create a new query based on the **AgeRange** and **Patients** tables.
- Add **AgeRange** from the **AgeRange** table and **PatientID** and **DateOfBirth** from the **Patients** table.
- Click on the **Totals** icon in the **Ribbon** and select **Group By** from the **Total** row dropdown for **AgeRange**.

- Select **Count** from the **Total** row dropdown for **PatientID**.
- Alter **DateOfBirth** to read ***DateDiff('m',[DateOfBirth],Now())/12***, remove the tick from the **Show** tickbox and select **Where** from the **Total** row dropdown.

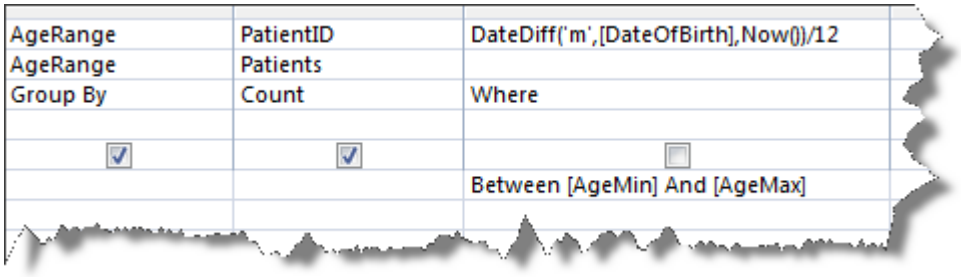


Figure 22 – the finished query

- In the **Criteria** row underneath the **Where** clause enter ***Between [AgeMin] And [AgeMax]***.
- **Run** the query to see the results, then close, saving the query as ***Task10Query1***.

Task 12 Nested queries

Objectives To find a value based on the results of another query using only one query. To use nested field expressions.

Comments The nested SQL statement must be enclosed in parentheses and Access carries out this clause first. If the nested query produces more than one value then the 'outer' query will not work. For introductory information about SQL see Task 15.

12.1 To create a nested query to find the treatment with the maximum dose:

- Create a new query using the **Drugs** and **Treatment** tables.
- Add **DrugName** from the **Drugs** table and **Dosage** from the **Treatment** table.

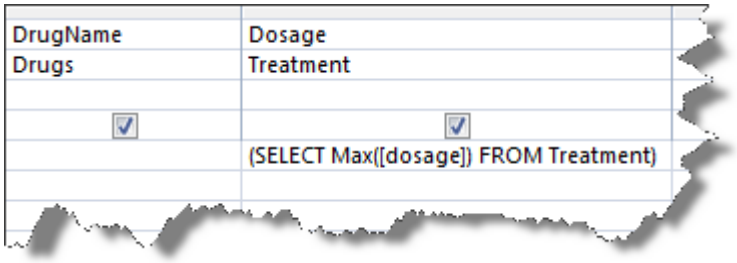


Figure 23 – the finished query

- In the **Criteria** row for **Dosage** enter ***(Max([dosage]) FROM Treatment)***.
- **Run** the query to see the results, then close, saving the query as ***Task11Query1***.

Note Just in case there is still confusion about what this query is doing – the SELECT clause uses the Max function to find the highest value in the dosage field. The highest dosage value is 4. The query then uses 4 to match against Dosage and shows DrugName where the dosage is 4. In this instance this returns 2 records.

12.2 To create a query with nested field expressions that show how long a treatment has lasted and how the total number of does in that time:

- Create a new query using the **Drugs** and **Treatment** tables.
- Add **DrugName** and **DosesPerDay** from the **Drugs** table, **DateStartCourse** from the **Treatment** table and **DosesPerDay** from the **Drugs** table again.
- Amend **DateStartCourse** to read:

  NumberOfDays: DateDiff('d',[DateStartCourse],[DateEndCourse]).

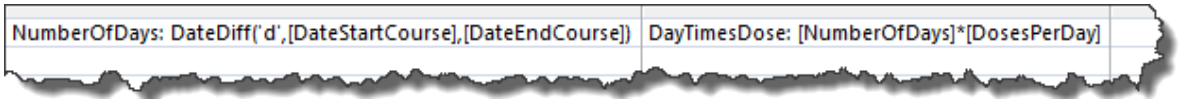


Figure 24 – the two expressions, the output from the first being used in the second

- Amend the second **DosesPerDay** to read

  DayTimesDose: [NumberOfDays]*[DosesPerDay].
- **Run** the query to see the results, then close, saving the query as ***Task11Query2***.

Task 13 Calculating percentages

Objectives To create three queries to calculate a percentage, where the third query uses the first two queries.

Comments The first query finds the number of doses that have been taken for each drug, the second query finds the number of doses in total, whilst the third query calculates the percentage of the total number using the results of the first two queries.

Stage 1: Create a query to find the number of doses per drug

13.1 To create a query to find the number of doses per drug:

- Create a new query based on the **Treatment** and **Drugs** table.
- Add **DrugName** and **CostPerPacket** from the **Drugs** table and click on the **Totals** icon on the **Ribbon**.
- In **DrugName** select **Group By** from the **Total** dropdown.

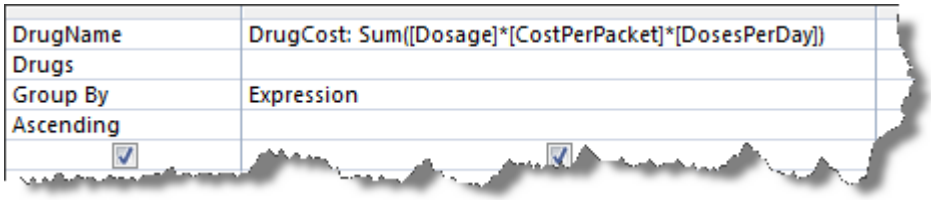


Figure 25 – the finished query

- Amend **CostPerPacket** so that it reads:

  DrugCost: Sum([Dosage]*[CostPerPacket]*[DosesPerDay]) and select **Expression** from the **Total** dropdown.
- **Run** the query to see the results, then close, saving the query as ***Task12Query1***.

Stage 2: Create a query to find the total number of doses

13.2 To create a query to find the number of doses in total:

- Create a new query based on the **Treatment** and **Drugs** table and add **CostPerPacket** from the **Drugs** table.

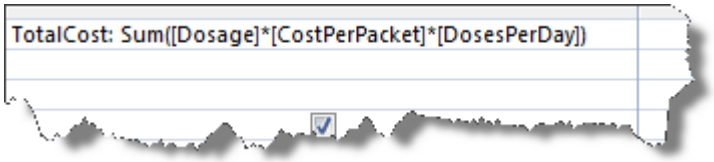


Figure 26 – the finished query

- Amend **CostPerPacket** to read:

  TotalCost: Sum([Dosage]*[CostPerPacket]*[DosesPerDay]).
- **Run** the query to see the results, then close, saving the query as ***Task12Query2***.

Stage 3: Create a query to find percentage based on the first two queries

13.3 To create a query to find percentage based on the first two queries:

- Create a new query, but instead of selecting any tables click on the **Queries** tab in the **Show Table** window and select **Task12Query1** and **Task12Query2**.

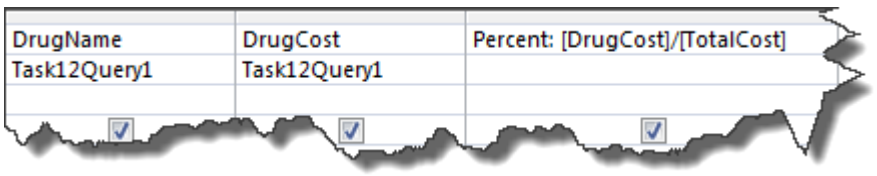


Figure 27 – the finished query

- Add **DrugName** and **DrugCost** from **Task12Query1** and in the next field enter ***Percent: [DrugCost]/[TotalCost]***.
- **Run** the query to see the results, then close, saving the query as ***Task12Query3***.

Task 14 Unmatched queries

Objectives To create two queries using different methods to find data that is in one table but not in another and to introduce different table joins in queries.

Comments There are three different types of join between tables – the default is to only include records from both tables that share a common value for a field, for example a query on the Drug and Treatment tables only lists drugs used in treatments.

Using the Is Null operator

14.1 To create a query to find drugs that have not been used as a treatment using the Is Null operator:

- Create a new query based on the **Drugs** and **Treatment** tables.
- Add **DrugID** and **DrugName** from the **Drugs** table and **DrugID** from the **Treatment** table.

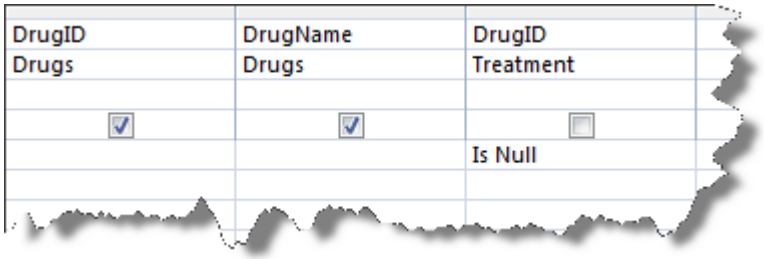


Figure 28 – the finished query

- In the **Criteria** row for **DrugID** from the Treatment table enter ***Is Null*** and remove the tick from the **Show** tickbox.
- **Run** the query to see the results, then close, saving the query as ***Task13Query1***.

Using the Not Exists operator

14.2 To create a query to find drugs that have not been used as a treatment using the Not Exists operator:

- Create a new query based on the **Drugs** and **Treatment** tables.
- Add **DrugID** and **DrugName** from the **Drugs** table and in the next free field enter:

  Exists (SELECT * FROM Treatment WHERE Drugs.DrugID = Treatment.DrugID).

Figure 29 – the finished query

- In **Criteria** row for **Expr1:** enter ***False*** and remove the tick from the **Show** tickbox.
- **Run** the query to see the results, then close, saving the query as ***Task13Query2***. You will notice that the results of both of these queries are exactly the same.

Table joins

Note We all take it for granted that when we run a multi-table query the results will be exactly as required, but the default only returns results where there are matching values in the linked key fields, whereas we might want to see all data from one table irrespective of matches.

Join type	Explanation
Equi-Join or Inner Join	The default includes records from both tables that share a common value for a field. Thus a query on the Drug and Treatment tables will only list those drugs that have been used in treatments.
Left Outer Join	This join lets you include records from the primary table regardless of whether or not a record matches a record in the other table. Thus, drugs can be listed regardless of whether they have been used in treatments or not.
Right Outer Join	This join lets you include records from the other table regardless of whether or not a record matches a record in the primary table. Thus, the query can display all treatment records regardless of whether they have details in the drug table. (This would break referential integrity since it should not be possible to prescribe a drug if it is unknown. Thus unmatched queries can also be useful when trying to set up referential integrity between two tables.) If your database is correctly set up, then running a multi-table query with a right outer join should return the same results as the default query setting.

14.3 To view the results of a default (equi-join/inner-join) two table query:

- Create a new query based on the **Patients** and **Treatment** tables.

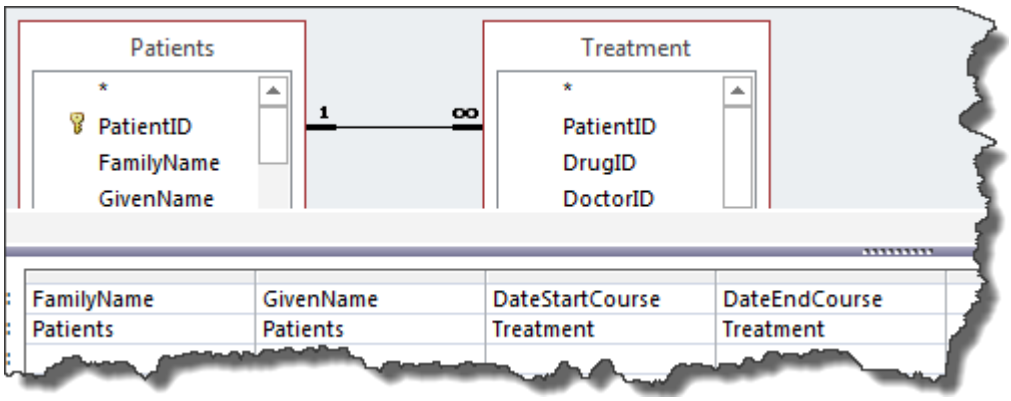
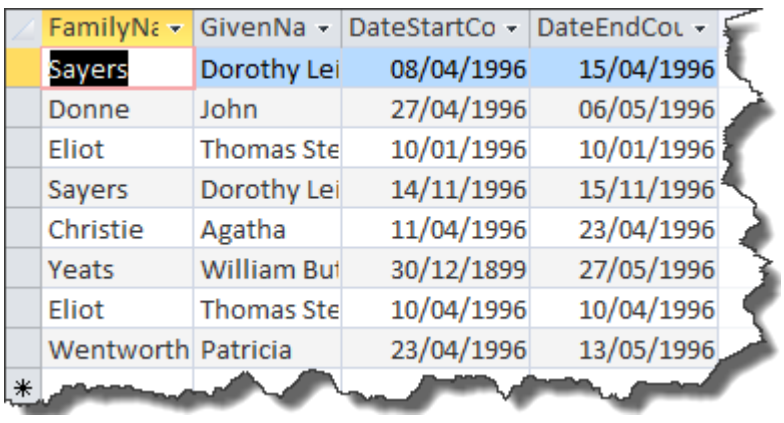


Figure 30 – the finished query … for a moment, at least

- Add **Familyname** and **GivenName** from the **Patients** table and **DateStartCourse** and **DateEndCourse** from the **Treatment** table.



FamilyNa	GivenNa	DateStartCo	DateEndCou
Sayers	Dorothy Lei	08/04/1996	15/04/1996
Donne	John	27/04/1996	06/05/1996
Eliot	Thomas Ste	10/01/1996	10/01/1996
Sayers	Dorothy Lei	14/11/1996	15/11/1996
Christie	Agatha	11/04/1996	23/04/1996
Yeats	William But	30/12/1899	27/05/1996
Eliot	Thomas Ste	10/04/1996	10/04/1996
Wentworth	Patricia	23/04/1996	13/05/1996

Figure 31 – you should receive 8 results

- **Run** the query and note that there are 8 results.

14.4 To run the same query with a left outer join:

- In **Design View** right-click on the relationship between the two tables and select **Join Properties**.
- Select the second option – ***Include ALL records from 'Patients'*** …and click on **OK**.

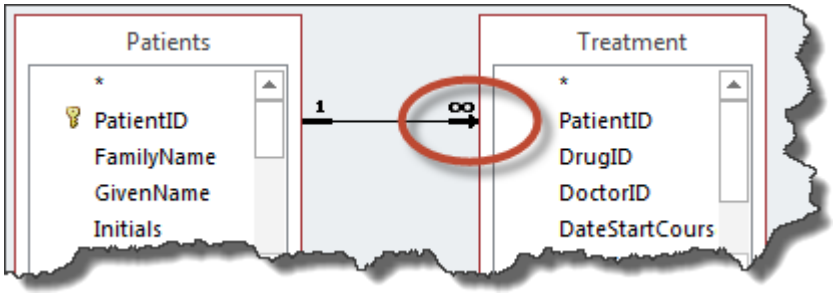


Figure 32 – the amended query with arrow at many end of relationship

- Note that the relationship now shows an arrow at the many end to denote that the query has a left outer join.

FamilyNa	GivenNa	DateStartCo	DateEndCou
Eliot	Thomas Ste	10/01/1996	10/01/1996
Eliot	Thomas Ste	10/04/1996	10/04/1996
Sayers	Dorothy Lei	08/04/1996	15/04/1996
Sayers	Dorothy Lei	14/11/1996	15/11/1996
Christie	Agatha	11/04/1996	23/04/1996
Marsh	Ngaio		
Wentworth	Patricia	23/04/1996	13/05/1996
Betjeman	John		
Macneice	Louis		
Allingham	Marjorie		
Heyer	Georgette		
Wilde	Oscar		
Brooke	Rupert		
Rendall	Ruth		
Donne	John	27/04/1996	06/05/1996
James	PD		
Browning	Robert		
Simpson	Dorothy		
Moyes	Patricia		
Yeats	William But	30/12/1899	27/05/1996
Auden	Wystan		

Figure 33 – you should now see 26 returned query results

- **Run** the query and now notice that you see all patients irrespective of whether they are or have been receiving treatment.
- Close the query and save it as ***Task13Query3***.

Task 15 Using SQL to join two sets of query results

Objectives To understand the basics of Access SQL basics and to create a union query that joins the outputs of two queries together.

Comments SQL is not case sensitive, but it is best to write clauses in upper case for ease of identification. Note that Access SQL has differences to Oracle SQL.

What is SQL

SQL commands are used to create, query and maintain a database. Viewing specific subsets of data from tables is the most common use of SQL. You don't see the SQL statement (unless you look specifically) but when you create a query, Access, in the background, writes an Access SQL *SELECT* statement.

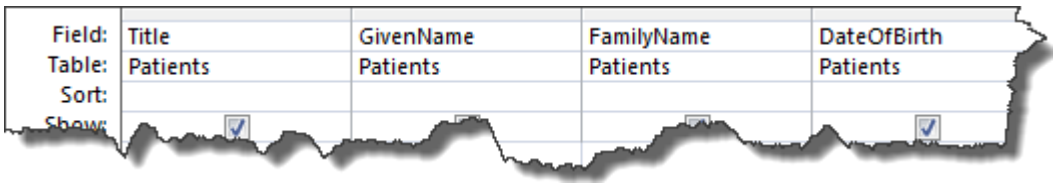


Figure 34 – how a query looks in the Query By Example grid

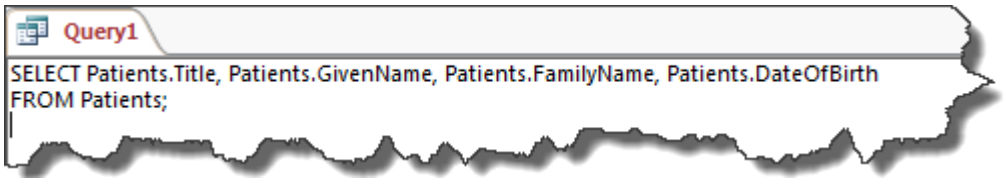


Figure 35 – the same query in the SQL design window

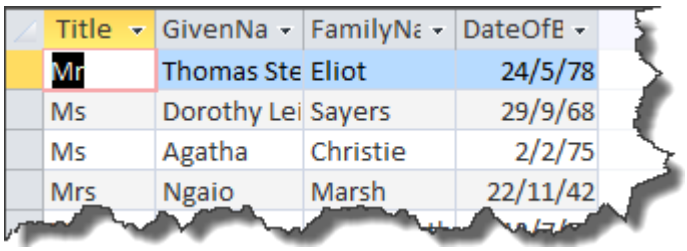


Figure 36 – the query results

You can also use Access SQL to create (and delete) tables and set field properties.

Basic SQL statement structure

A basic SQL statement consists of two clauses which must be specified:

1. *SELECT* – the fields you want to display.
2. *FROM* – which tables the fields come from.

All SQL queries include the above two clauses, but further clauses can follow these. SQL queries must finish with a semi colon.

What does a Union query do?

This combines fields from two or more SELECT queries into a single query (and eliminates duplicate rows whilst it's at it). For example, if you have one query that lists individual treatments and another that lists the totals for each drug, you can combine these lists into one result set.

Do I have to write Union queries in SQL from scratch?

No, you can create the two select queries as normal in the QBE grid, then go to SQL View and copy the SQL. You can then paste this into Notepad. Once you have both sets of SQL pasted into Notepad, you can copy this back into the SQL View window with "UNION" added between the two. To get to an empty SQL View window, create a new query, but shut down the Show Table window without choosing any tables. You will then see the SQL View option in the Ribbon.

15.1 To view an existing Union query (no, we're not so mean as to get you to type in the whole lot!):

- Open **Task4Query1** in **Design View** (note that this query is close to the bottom of the **Navigation Pane** – note also the different icon).

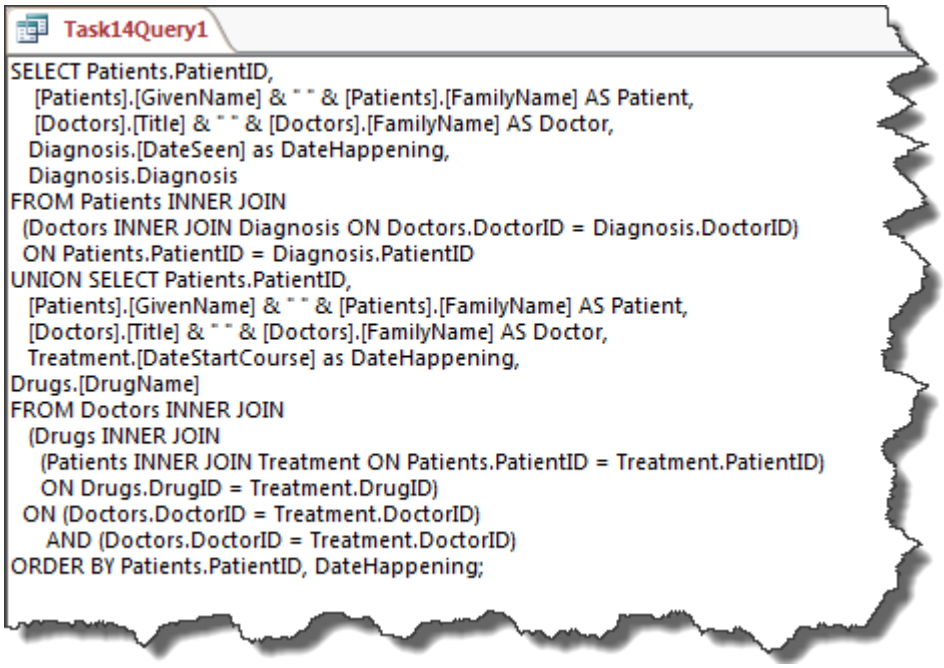


Figure 37 – a union query in the SQL design view window

- Note that it opens in **SQL View** – Union queries cannot be opened in the QBE grid (though the component parts can be created in the grid and the SQL copied).

Note The number of fields in the two select clauses must be equal and the data types of the fields must match. The AS clause can be used to change the name of the resulting fields. By default, the resulting field should be the name of the field of the first SELECT clause. Thus, the last field is called Diagnosis, the penultimate field is called DateHappening. The last clause is the ORDER BY clause. This applies to the result of the whole query, not the integral parts so there is only one ordering clause.

- **Run** the query to see the results then close the query.

15.2 To create your own Union query:

- Create a single Union query to list the names of patients and the names of doctors.