Chapter 7
Database Queries in Access

Chapter Outline

7.1 What you'll be learning

Databases play a number of roles within a business. We've seen our Brokerage database act as a repository of data (held as a collection of related tables), we've seen it act as a user interface (through the use of forms) and we've seen it as a report generator. It is also a wonderful tool for asking questions about your data. Who bought IBM for less than $45 a share during the past month? What is the commission made on each sale by broker Smith? What are the total commissions made by each broker for just the current month? Can I see the customers sorted according to last name? Which customers have made the most money – perhaps a list of customers with the top 20% of gains? Can we delete Customer 1234 and all of their related material? Who hasn't bought any stocks during the past year? One of our Brokers has left the firm, can we reassign all of their customers to a broker we just hired? Can I see a grid that details the total value of stocks – with customers listed along the rows and brokers listed in the column headings? Most of these questions will get answered in this chapter.

Each of these questions represents a database query. In many databases these queries are issued as SQL statements. You might recall this Structured Query Language from the overview we provided in Chapter one. Here's a simple SQL statement that finds all of the data we have for each customer living in Rhode Island.

```
SELECT * FROM Customers WHERE State = 'RI'
```

While most industrial strength databases use SQL as their query mechanism, Access provides a wonderful design tool that makes queries a snap – it's called the ***Query Design Grid***. Access certainly understands SQL statements as well; in fact, the query you design with the design grid will be translated into SQL before Access will execute that query. This is good news since it shields us from having to know SQL at this point. In our next chapter we'll spend some time learning the basics of SQL, since one of our goals is to

interact with this database through a web site and the neat little design grid won't exist on the web! At that point, if you don't know a bit of SQL, you won't be able to talk to your database.

7.2 Overview of Query Design – Basic queries

Let's begin with a simple query that illustrates field selection, the order of those fields, sorting and setting simple criteria. I'll design a query that finds all customers living in Rhode Island and will display the customer's name, City, State, the names of the stocks they own and the stock's current price. The data will get sorted by the customer's last name. For this query to work, we'll need data from the Customer, Trans and Stocks tables.

There are two options for building a query – you can use a wizard *or* move directly into design mode. To me, the wizard isn't the huge help it is with Forms and Reports. Creating the design by yourself is probably easier, so select ***Create Query in Design View*** (Fig. 7.1)**.**

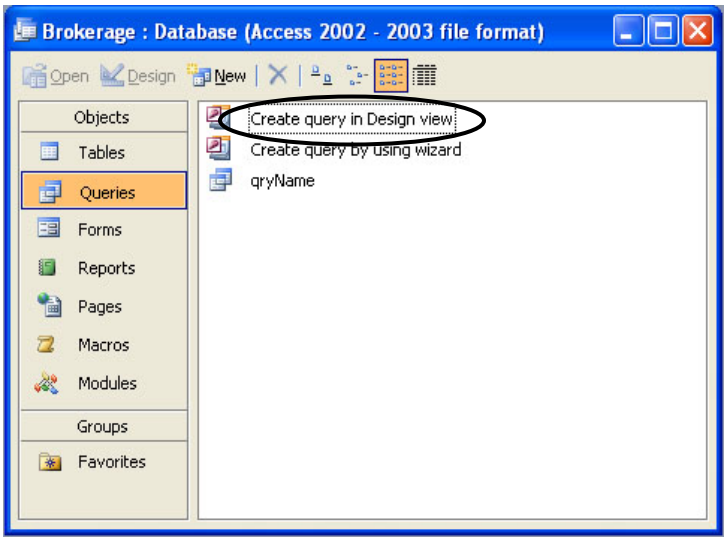


Figure 7.1 – Launching the query design grid

Once you select "Create query in Design View", Access will display the ***Show Table*** dialog box. Begin the design by selecting the tables involved in your question. You can click on a table and select ADD – or you can simply double-click each table you need. As you select the tables, they are added to the query design. Once you've selected the tables, close the Show Table dialog box.

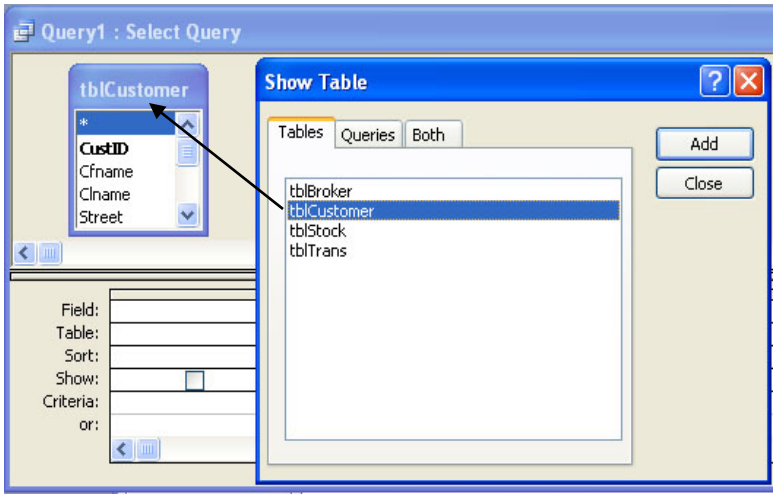


Figure 7.2 – The Show Table dialog lets you select table involved in your query

If you forgot to add one of these tables to the design grid, there is a tool on the main toolbar that will display the Show Table screen. That tool looks like a database table and an ADD (Plus) symbol (Fig. 7.3).

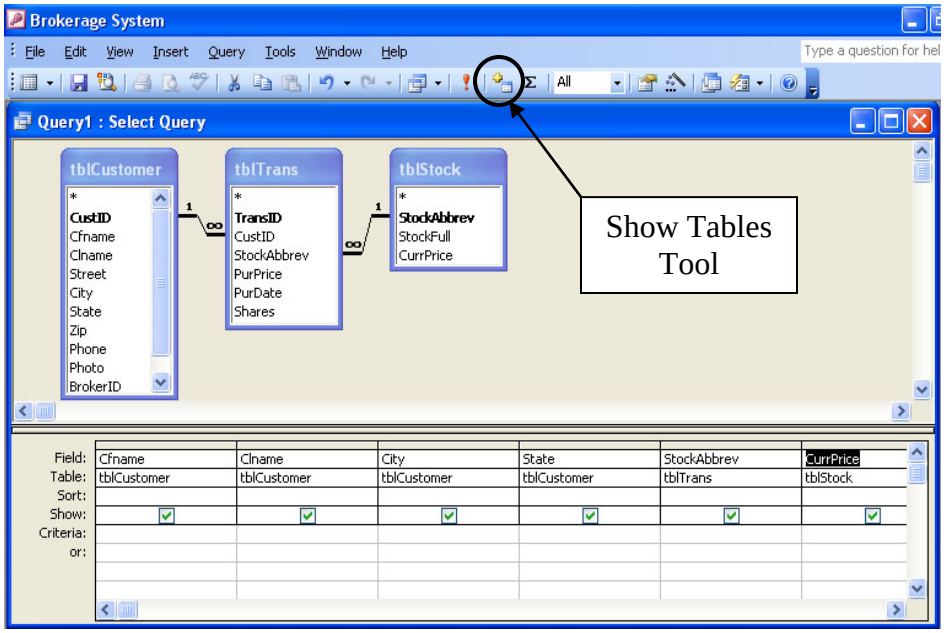


Figure 7.3 – The Show Tables tool brings displays the Show Tables dialog

Since our query asks for the stocks owned by each customer living in Rhode Island, I've *double-clicked each field* I wanted in the answer (names, addresses and stock holdings) and Access moved them into the Field and Table rows of the query grid. When we execute this query Access will display the first name, last name, city, state, stock abbreviation and current price.

Getting the data sorted by last name is simple. Beneath the Table names in our grid there is a row for sorting (Fig. 7.4). Click in the sort area for the column with last name in it and select *Ascending* as the sort order. Ascending order is normal alphabetical or numeric order. Select *Descending* if you want the sort order to start at the letter "Z" or to go from large numbers to small.

The row labeled ***Criteria*** is where we set the conditions for the query. For this simple query, the only condition is that the customer must be from Rhode Island. I entered **RI** as the criterion (Fig. 7.4) and Access helped me out by placing quotes around it – criteria involving words need to be inside quotes. By the way, never put quotes around numbers if they are involved in a query.

What are the check boxes for? Well, we might need the State field so we can specify that only Rhode Islanders should be in the query results – but we might not want to display the State field for some reason. If that's the case, just take the check mark off the State field.

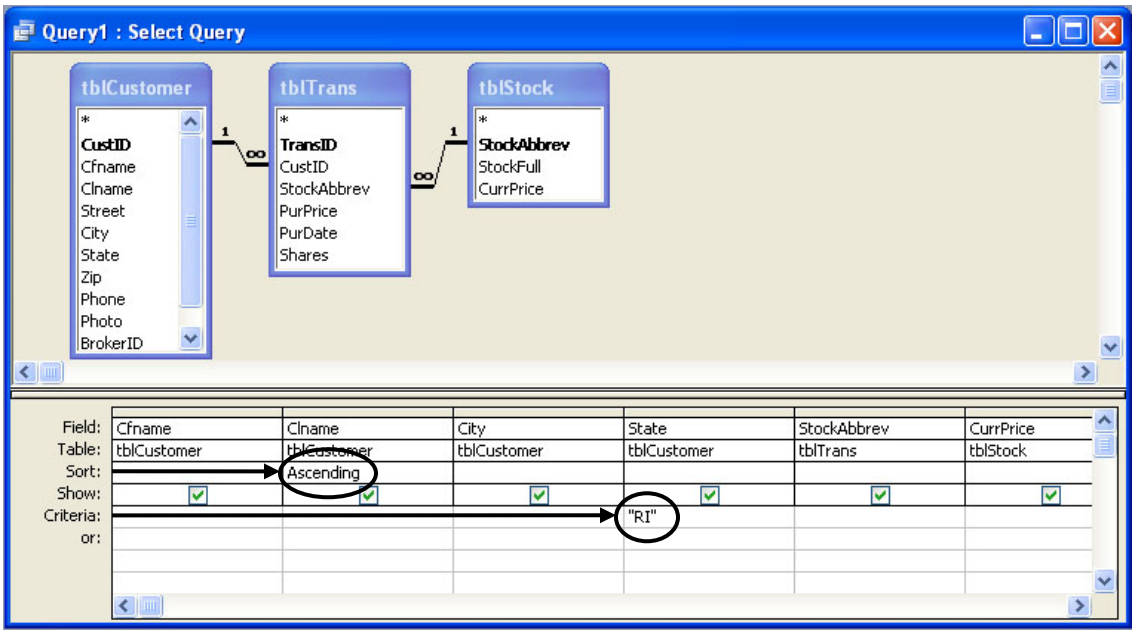


Figure 7.4 – The design grid lets you select tables and fields, sort your answer and specify conditions

Moving Columns to New Locations

Before we run this query, decide if the fields are in the right order. If you want to re-position any column of data, grab the column with your mouse and drag it into the new position. When you grab the column, hold your left mouse button down as you position the mouse within the thin gray box just above the field name. The mouse will change into an arrow with a small box at its base. Once you see this shape, just drag the column until it reaches the new location.

Running the Query

The easiest way to execute (run) this query is to select the *Exclamation Mark* up in the toolbar. Another approach would be to click the 'down arrow' next to the Design tool in the upper left. From there you'll see a number of views – you'll need to pick *Datasheet* if you want the query to run. A third alternative is to select *Open* when you are at the screen that lists all of your queries.

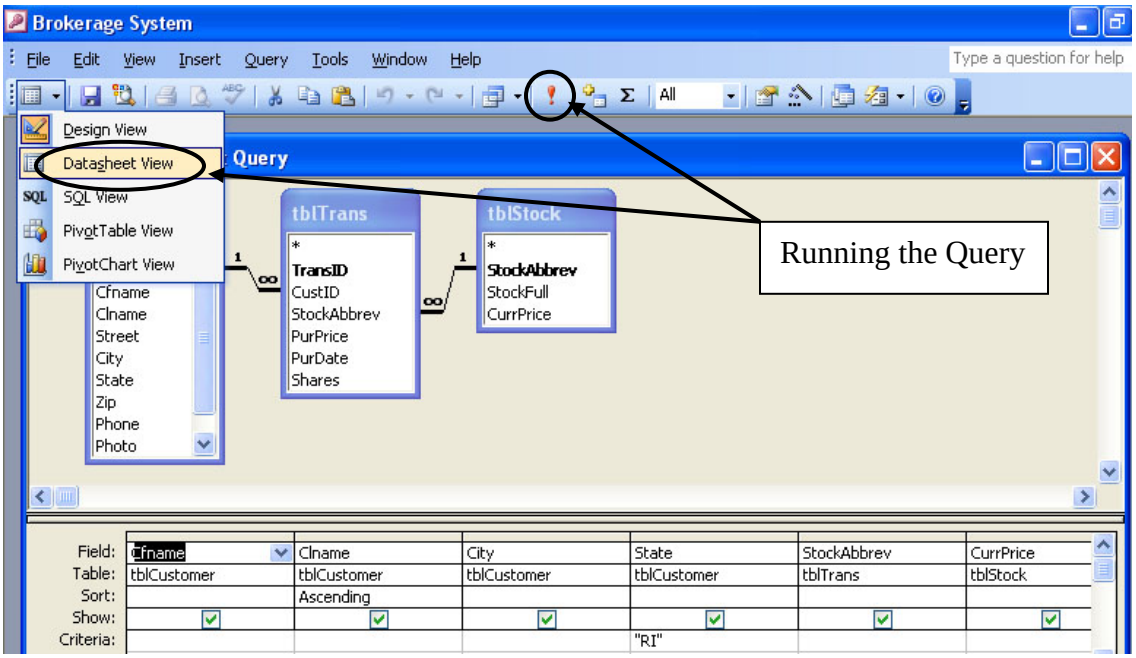


Figure 7.5 – There are several ways to Run a query

My database only has two customers from RI (Adams and Snyder) – hopefully we'll pick up a few more!

CustStocks : Select Query

Cfname	Clname	City	State	StockAbbrev	CurrPrice
Sue	Adams	Barrington	RI	BnkAm	$27.00
Sue	Adams	Barrington	RI	Apple	$25.00
Harry	Snyder	Bristol	RI	Apple	$25.00

Figure 7.6 – Query listing RI customers sorted by last name

Once the query runs you can shift back into design mode by clicking the Design Tool in the upper left corner of the query screen.

If you are ready to save this query, just close the query by clicking the "X" in the upper right of the query – be careful, there are also "X"s for the entire database and for Access. Access will prompt you to save the query and provide it a name. If you are following our scheme that prefaces each object with three letters that identify the type of object, then use qry for all of your queries. I'll name my query **qryCustStocks**.

Multiple Criteria (think AND)

What if there are a number of conditions that must exist as criteria in the query? Let's say we want to see people living in RI who bought IBM for less than $45. These conditions are logically connected with the word *AND* – I want people who lived in RI **AND** who bought IBM **AND** who paid less than $45. Dealing with multiple criteria like this is a simple matter of specifying the conditions for each field (Fig. 7.7).

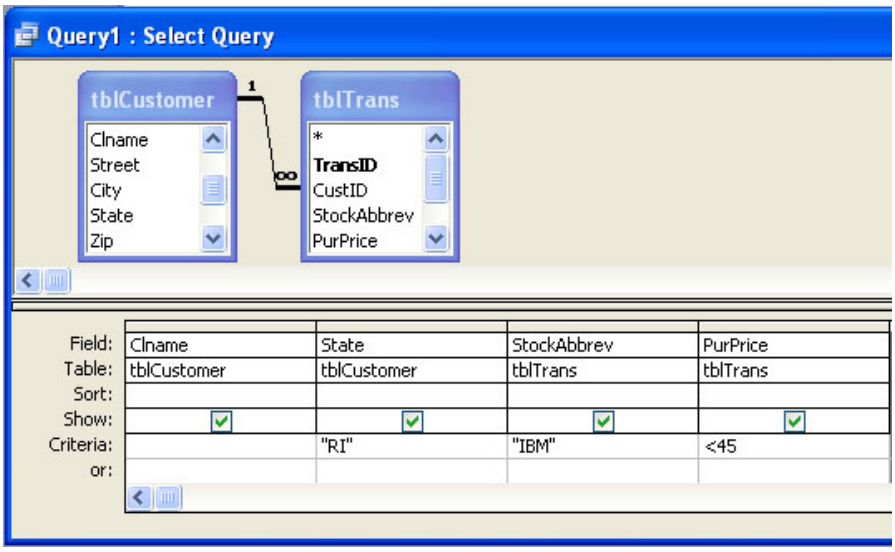


Figure 7.7 – Query to select RI customers who bought IBM for less than $45 per share

Here's the run:

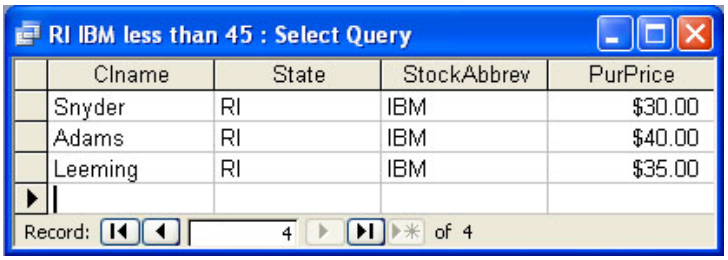


Figure 7.8 – Run showing RI customers who bought IBM for less than $45 a share

Working with Ranges

Finding a range of values is an extremely common query condition. Who bought IBM for between $30 and $50? There are two ways of expressing a range. Most computer people would write the condition as:

>=30 and <=50

You would read this as "greater than or equal to 30 and less than or equal to 50". In plain English, think of ">=" as "**At least**" and think of "<=" as being "**no more than**". So in English we might think of the formula as "At least 30 **and** no more than 50".

Access provides an even easier way to express a range – you simply use the word "Between". So the formula would be:

Between 30 and 50

Using OR conditions

What if we want to find people who own Apple or IBM? This is simple if we connect the two conditions with the word "OR". You would go to the criteria for StockAbbrev and set the condition to

"Apple" or "IBM"

Using OR conditions with a List of items

Suppose you are looking for people who own one of 5 different stocks. You could string the list of possibilities together with OR, or you could use the ***IN*** operator that we played with when we were doing Validation tests on our forms – remember when we set the condition that the customer had to live in CT, MA or RI? We did this by saying that the

state had to be IN the following list. For our simple example of Apple and IBM, the condition could be written:

IN (“Apple”, “IBM”)

Complex OR conditions (be careful…)

One of the managers at the brokerage has asked us to locate all customers who live in RI and own either Apple or IBM and all customers living in MA who own Apple or Microsoft (MSFT).

These are really two different conditions separated with an OR. Either you live in RI and own the appropriate stock **OR** you live in MA and own the appropriate stock. Notice how the query design grid provides a set of criteria rows (Fig. 7.9). You can be in the results if you meet this condition OR you meet this condition OR you meet this condition etc. *Each line in the query grid represents a different condition.*

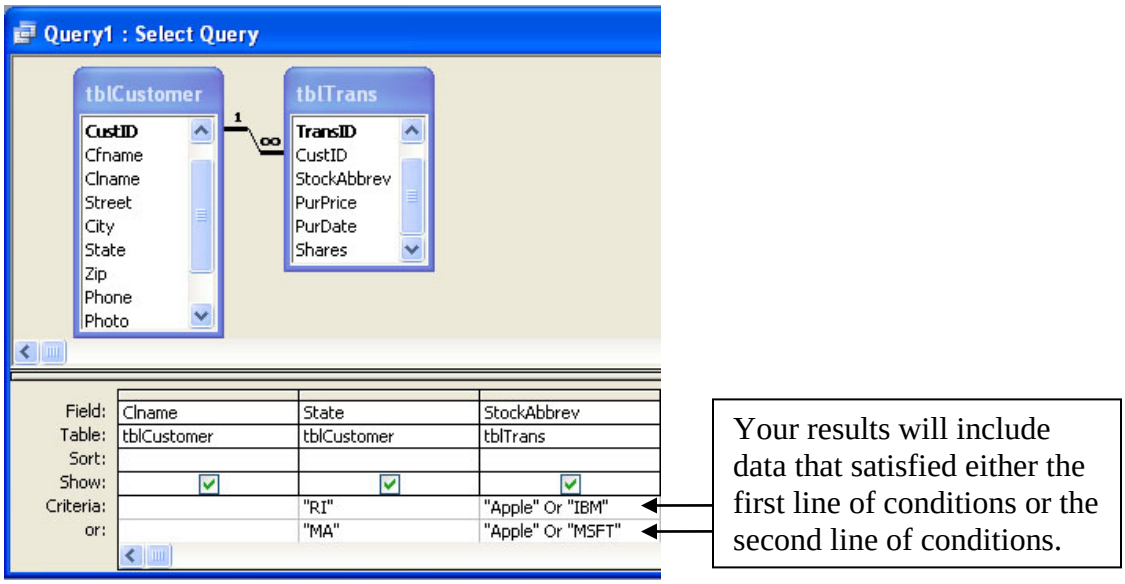


Figure 7.9 – A query can include several independent conditions

Check your understanding of this use of the OR condition. If we had the following design grid (Fig. 7.10), what do you think the query results would be? I’ve placed the answer *just under the figure* – don’t peek until you have your answer!

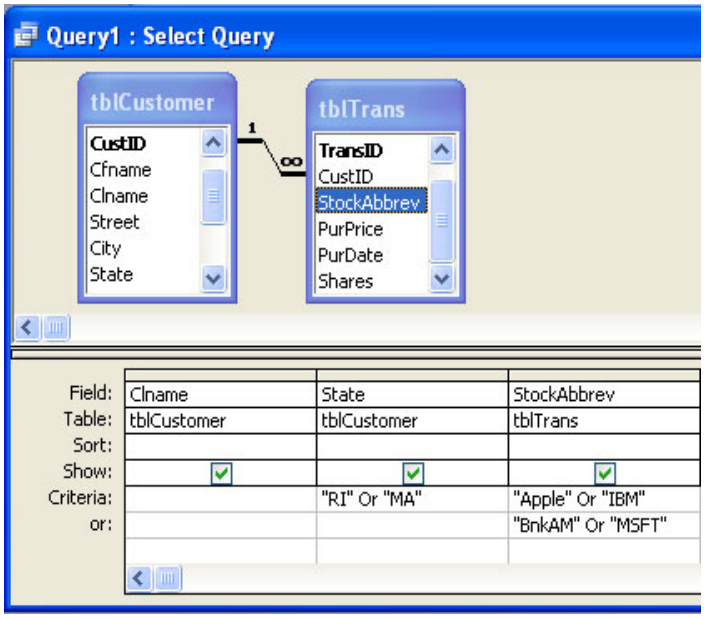


Figure 7.10 – Test your understanding of independent conditions. What will this query list?

The query has two condition lines. One condition is that customers must live in RI or MA **and** they must own Apple or IBM. The second line says nothing about the state you live in – it only cares that you own Bank of America or Microsoft. The query will list everyone who owns those two stocks, even people living in RI or MA!

7.3 Using Wildcards

There are times when you want the condition to match a pattern of some sort. Perhaps you need to see a list of all customers whose names begin with an "A", or maybe all customers whose names start in the range from "A to M". We went over some of these wildcards when the Brokerage database was being designed – but this would be a good time to work with those again and add a few new ones (you already know * and ?).

Microsoft has a group of wildcards that make finding patterns easy. Check out the list in the table below, then we'll go over some examples to get you comfortable.

Wildcard	Use
*	Substitutes for any number of characters
?	Substitutes for a single character
[ ]	Match any character in these brackets
-	Use to indicate a range of characters
!	Stands for "NOT"
#	Like a question mark, but for numbers

Wildcard Examples

Asterisk Wildcard – matches a collection of characters	
Last names that start with "S", remaining characters could be anything	S*
Names that end in "s", but could start with anything	*s
Names that start with "S", end with "s" but could be anything in the middle	S*s

Question Mark Wildcard – Replaces a Single Character	
First character could be anything, rest of word is "all" – e.g. ball, call, fall	?all
Know how the name starts and ends – middle character could be anything	Sm?th
Don't care what first 2 characters are, then an "m", then any characters	??m*

Bracket, Hyphen and Exclamation Wildcards – to specify more than one character	
Find Smith or Smyth – note brackets specify the substitute as "i" or "y"	Sm[iy]th
Name starts with anything from A to M; rest of characters don't matter	[A-M]*
Starts with "S", but make sure second character is NOT an "e"	S[!e]*

Number Sign Wildcard – Substitutes single digit character	
Be careful, we're talking about *Text fields* that hold digits – not real Numeric fields	
Find zip codes that start with **0280**. Note: zip would need to be a text field	0280#

7.4 Creating New Fields (Calculated Fields)

One of the guidelines that database developers follow is that you normally don't store something that is easy enough to create when you need it. For example, each broker will be paid a 5% commission on the purchase of each stock. I suppose we could have added a field in the Trans table that stored that number – but why store it when we can make it on demand! Creating these "fields" within a query is fairly simple – you declare the

name of the new thing and use a **Colon** to separate it from the calculation that builds the new data (Fig. 7.11). Our calculation determines the purchase cost (PurPrice x Shares) and takes 5% of that amount. Check out the query (Fig. 7.12) - I've also created a calculation for the Cost of each stock purchase.

For our Commission field we would click in a new column of the design grid and type:

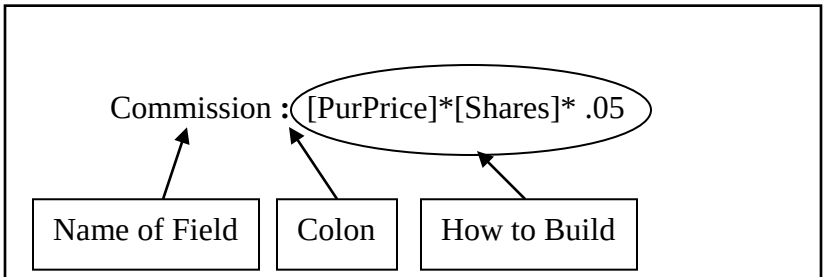


Figure 7.11 – Creating a calculated field

If typing a long formula in that small field box is tough on your eyes, right click in the field area and select Zoom from the context menu. That will open a nicely sized window for you to enter the description of the new field.

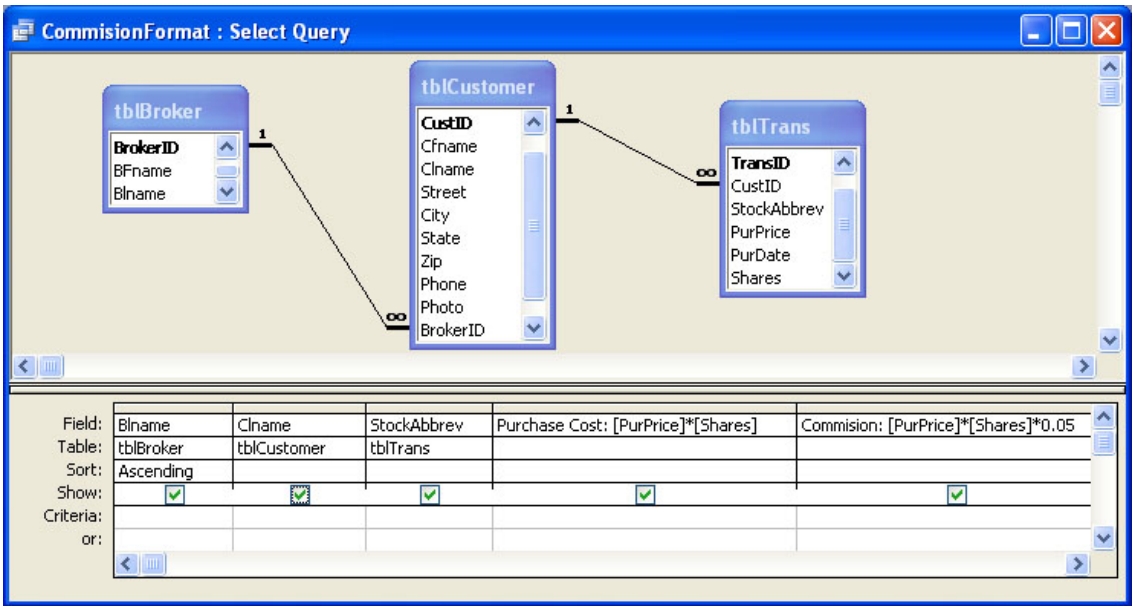


Figure 7.12 – Calculated fields for Purchase Cost and Commission
– generally you don't store something that you can build on the fly

You might be disappointed with the formatting when you run this query. I was hoping for the results to be formatted as currency – if the results display as ordinary numbers, I can force the formatting to be currency by right clicking on the new field, selecting Properties, Format and then Currency (Fig. 7.13).

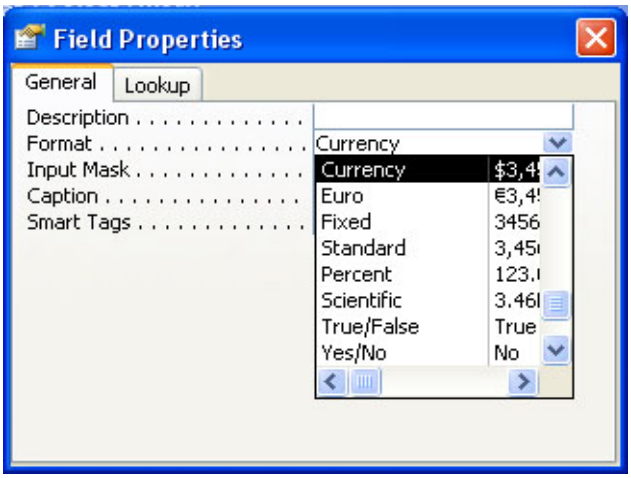


Figure 7.13 – Right-click on a field and select "Properties" when you want to adjust the format

Here's what the run looks like:

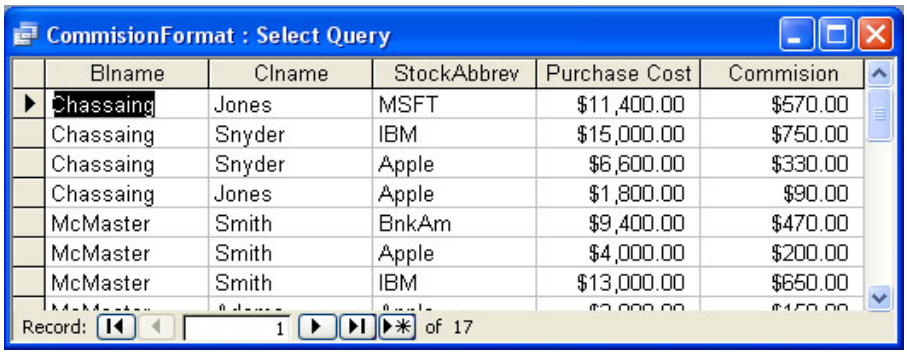


Figure 7.14 – Purchase Cost and Commission calculated and formatted as Currency

Once the new field exists, you can include it in the query condition. For example, list the sales made by broker McMaster that gave him a commission between $500 and $1000. You would simply set the criterion for the "Commission" column to "Between 500 and 1000" and set the Broker to "McMaster".

7.5 Parameter Queries – More Bang for the Buck

We've been working with queries that are created by *you*, the database person. You might have built a query to locate people who own IBM, or maybe Apple within a certain price range, or sold by a particular broker.

What if a broker needs to see MSFT rather than IBM? What if they are interested in who bought MSFT and in a price range different from the one that finds a range for Apple? Do we want to train our brokers to create a different query for every common question they have? How many different queries would that be?

The solution is something called a ***parameter query***. Rather than specify a particular condition, such as the stock being IBM, a parameter query will prompt the user for the name of the stock – and that becomes the criterion.

Let's start with a very simple example. Our brokers often need to see the list of people who have bought particular stocks. Rather than have the brokers build a new query for each stock they have an interest in, we'll design a single query and have that query ask the broker for the name of a stock. The trick is to replace the criteria with questions enclosed within square brackets. Here's a parameter query that prompts the user for the stock name (Fig. 7.15). Since the prompt is located in the Criteria row of the design grid, the answer to the question becomes the thing Access will search for.

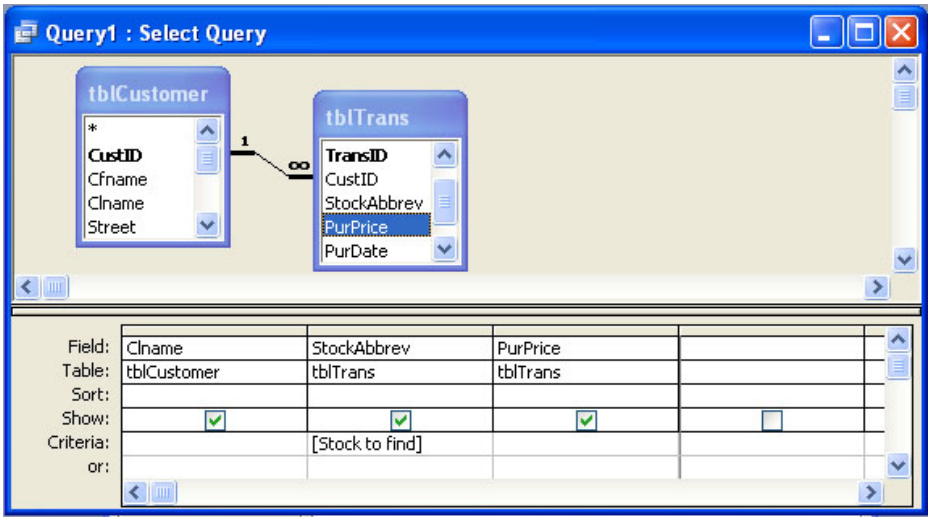


Figure 7.15 – Parameter queries let the user specify a condition at "run time"

When the parameter query runs, Access displays the question about the stock, the user answers the question and that answer becomes the criterion for the query. Here's what it looks like when the parameter query runs (Fig. 7.16) – I've entered MSFT as the stock I want to find. Once the user clicks OK to the parameter dialog screen, the query executes.

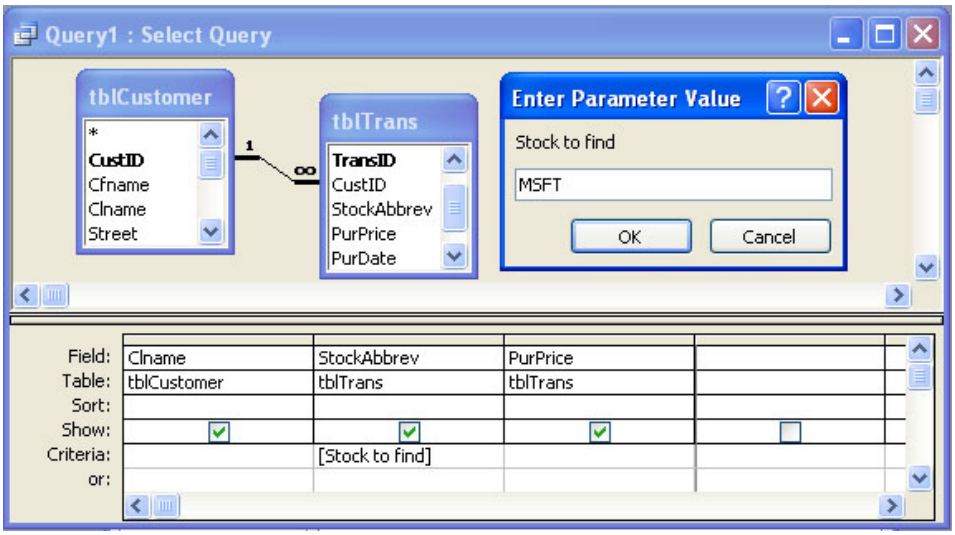


Figure 7.16 – A parameter queries prompts the user for a stock name

Let's try one that finds any stock the broker wants and lets them specify a price range. In this case we'll need three questions – one for the name of the stock, one for the low end of the price range and one for the high end. We can use a "Between" to get the range that goes from the answer to the low and high end questions. Here's what it will look like (Fig. 7.17).

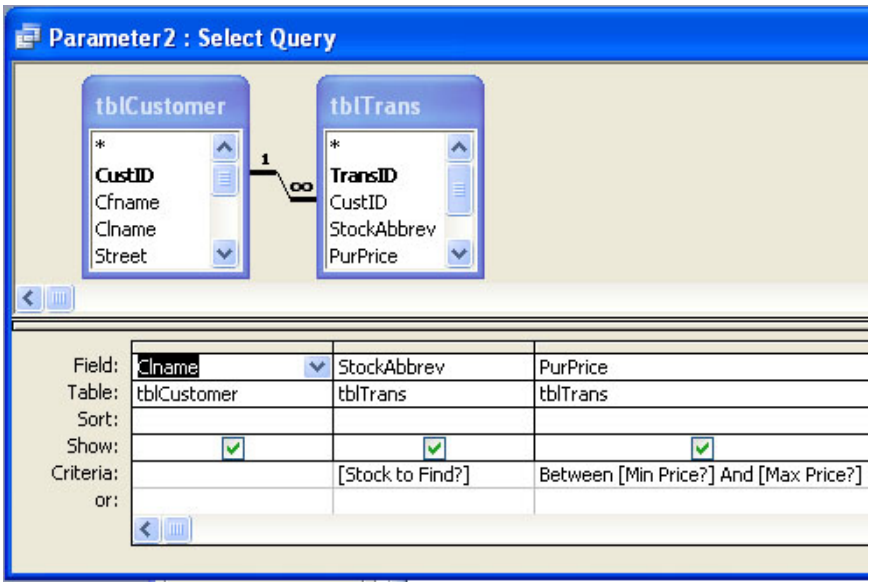


Figure 7.17 – Using multiple parameters in a query

When this query runs, Access will prompt the user with the first question, then move to the next questions – as Access finds each question going from left to right. Once Access has the three answers, the query will run.

Accidental parameters!

You may have accidentally created a parameter query in the past. When Access sees square brackets it assumes it is looking at either a field name or a parameter question. Access first checks to see if there is a field name that matches what's in the brackets. If there is no field that matches, it assumes you are doing a parameter query. You can check this out for yourself – enter a formula that misspells a field name…you should find Access treating it as if it were a parameter query! In the example below (Fig. 7.18) I put an extra "r" in PurPrice. Access couldn't find a field of that name, so it assumed I was trying a parameter query!

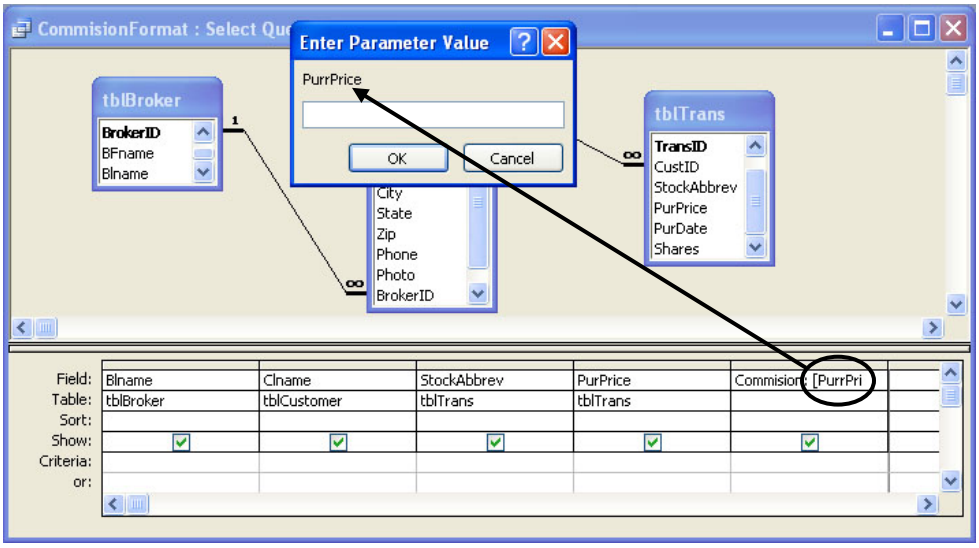


Figure 7.18 – Misspelled field names will generate a parameter question

7.6 Aggregate (Group) Queries

The management of our brokerage firm likes to see summary information. They are interested in the total sales by each broker, the average price paid for a particular stock, the number of customers they have from each state. The managers need to see data at a level that is often more general than the view our brokers need.

Let's start with a simple problem – how many customers do we have in each state? The process involves gathering each customer from CT into a group of CT customers, then each customer from RI into their own RI group and then does the same for every other state. Access doesn't have to know the names of the states involved. Once we've told Access that the states are forming our groups, Access will check out the State data and determine what their names are and how many states we have. After the customers have been grouped into states, we will count the number of customers who fall into each group.

Let's begin a new query design and select the two fields involved in the problem. In our case, we need to count customers and have them grouped by their State – so I've added the Customer's last name and the State to the QBE design grid. To add the grouping activity to this query, I clicked the ***summation*** symbol on the toolbar (if you pause your mouse over this symbol for a moment, it will say "***Totals***". That's how I got a row in the query grid that says "***Total***".

It is very important to avoid adding unnecessary fields to the grid. If the field isn't part of the grouping activity or involved in creating totals of some sort, then that field should not be in the grid – if you have extraneous fields, Access will generate an error saying that the field is *not part of the aggregate function* (aggregate is a fancy way of saying 'grouping').

Once you have the Totals row, it's time to determine which field will form the groups and which field will have some math performed on it. In our case (Fig. 7.19), I've chosen State as the ***Group By*** field. I've chosen to count the customers by counting their last names. Frankly, it would have worked if I had decided to count any field related to the customer – I could have counted their telephone numbers! I got to Count by clicking the drop down list in the Total row under Clname. My options were things like Sum, Average, Max, Min and Count.

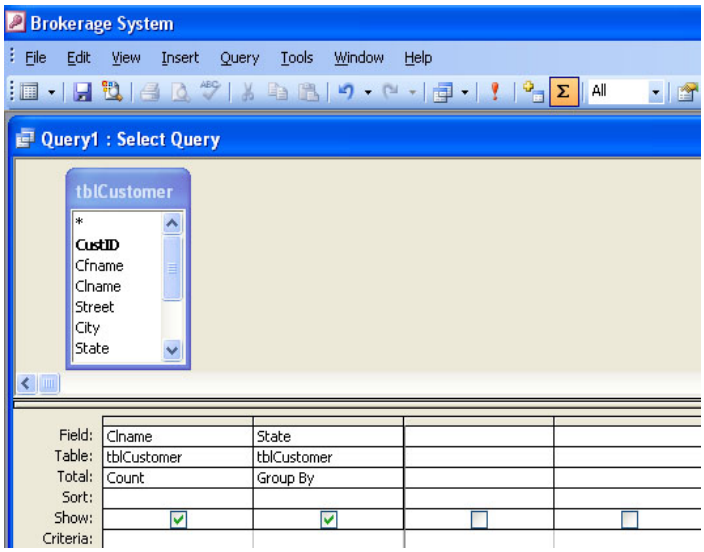


Figure 7.19 – Data is grouped by state and customers are counted

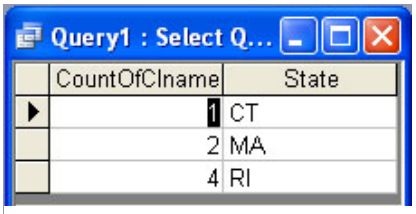


Figure 7.20 – An aggregate query counts customers in each state

When this query runs, it will count the customers in each state (Fig. 7.20). If you had wanted that list alphabetized, you could have selected an ascending sort for the State field. I have very few customers at this point so my totals are low – but here's what the group query found:

Here's a somewhat more complicated query. What if management wanted to see the *total gains or losses for each customer?* I've created a calculated field to determine the gain/loss for each customer – then I grouped the data for each customer (so all of Smith's data is together, all of Jones' data is together etc.) and then summed the gain/loss data (Fig 7.21).

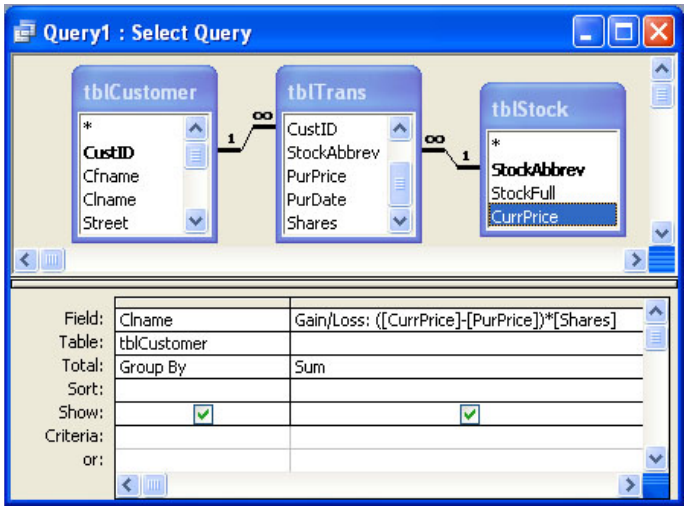


Figure 7.21 – An aggregate query based on a calculated field

…and here's the run:

Figure 7.22 – Data aggregated by Customer,
with the gains/Losses summed

Letting Access form the groups and do the Sum for our Gain/Loss field makes a lot more sense than doing the math by ourselves.

Hint: Whenever you see key words such as Total, Average or Count… start thinking of an aggregate query. In most cases the calculations will be for groups of data (total sales for each broker, average purchase price of each stock, total sales for each day).

7.7 Finding your Top (or Bottom) Performers

The management at our brokerage has decided to invite their most successful customers to a year-end party – partly to recognize the success of these customers and partly to push a new investment vehicle that these investors might have an interest in. The decision has been made that "successful" means the top 20% in terms of gains in their portfolio. Our database will be used to identify these people.

Access provides two ways to find top or bottom performers. You can ask to see the top X records or the top X percent. You see this type of problem all the time… perhaps the students with the top 5 GPAs are invited into an honor society, or a teacher decides that students in the top 10% of their class will receive an "A".

We need to build a query that calculates the Gain/Loss for each customer. That will involve creating a calculated field in the query. The data will then need to be grouped by Customer so that we can SUM the Gain/Loss data for the group of stocks each customer owns. Once all this is in place, we'll add the top 20% feature.

Here's the design (Fig. 7.23). Notice that I have included only those fields involved in the aggregate/group activity – the customer is there because I need the data grouped by customer. The Gain/Loss is there because I need this summed for each group of customer records.

The TOP 20% feature will involve two steps.

1. To the right of the Summation symbol you'll see a drop down list where you can select either a number or a percentage. If you don't like the limited choices, just type your own value into the box.

2. In most cases you'll want the results sorted. If you want to see the TOP 20%, then the sort will need to be Descending – since the data will go from large numbers to small ones. To see the bottom 20%, just set the sort to Ascending.

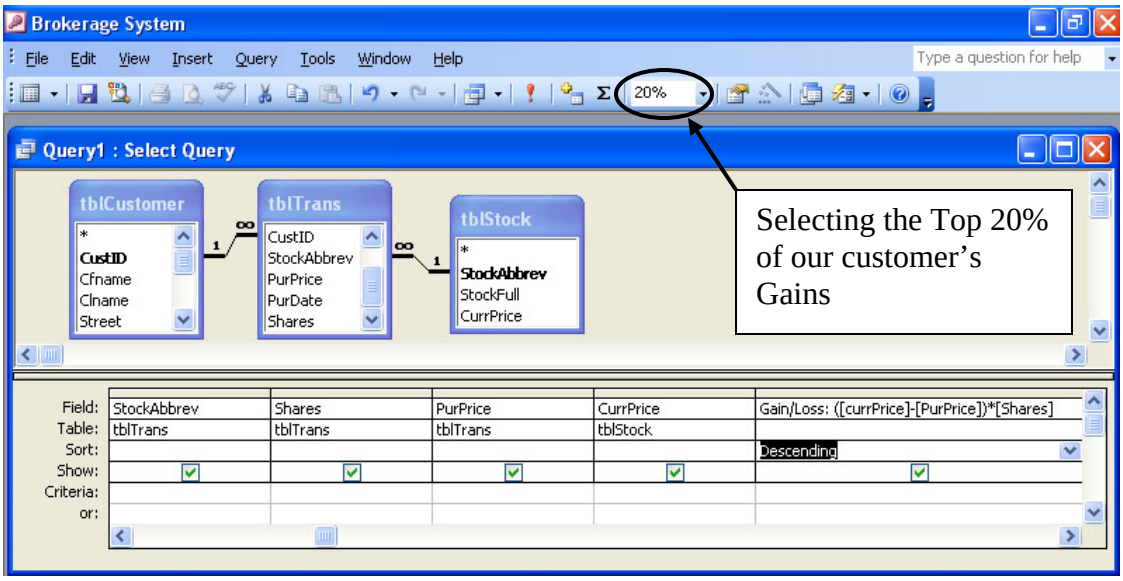


Figure 7.23 – Access provides a simple approach to finding your top or bottom performers

7.8 Delete queries

Databases are not static – we gain new customers and lose others. There has to be a way to clean the database of records that no longer belong there. If you have a single customer to delete, it might be easier to just open the customer table, select the customer and hit the delete key. Our brokerage has decided against deleting the customers one by one. We are implementing a process that involves listing each customer we have lost and having a manager analyze that list before deleting these customers from the database – perhaps we can call these customers and get them to change their mind about leaving.

To implement this policy, you have decided to add a new field to the Customer database that identifies the customers who are now "inactive". Once the manager has looked these customers over, the manager will pass the list to you. Before running the delete, you will check off those customers from the list the manager handed you. A query will then delete the entire batch of inactive customers at once.

Let's add the new field to the customer table (Fig. 7.24). The field name will be "Inactive" and the data type will be "Yes/No". Notice that I've set the Default to "NO". Checking the boxes off will now set them to the "YES" position.

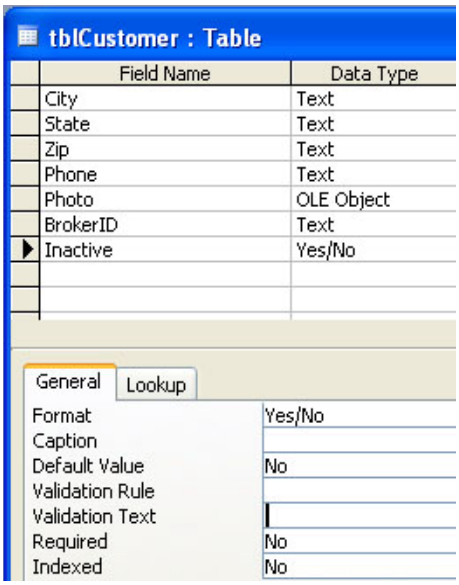


Figure 7.24 – A Yes/No field will let us identify customers who are inactive – management will make the decision and we'll check off who should be deleted from the Customer table.

The Customer table now has Check Boxes for the Inactive field (Fig. 7.25). Once management makes the decision, we'll check off the customers who are definitely leaving and then run a Delete query to delete any customer records where the Inactive field is set to "YES".

Figure 7.25 – Our new table design has check boxes that management will use to select customers who need to be deleted from the customer table

Designing the Delete query

We need to let Access know that this isn't another Select Query. There are special query designs for Delete Queries, Update Queries and a few others (Fig. 7.26). Before we continue with the design of this delete query, let's change the query type to ***Delete***. Just to the left of the "Run" tool there is a drop down that lists the various types of queries we

can use. So far, every query has been a SELECT query. We now need to use a DELETE query.

Once the query type has been set to a Delete query, drag the asterisk at the top of the Customer table into place as your first column in the design grid. That asterisk says that we are grabbing all of the fields in the Customer records. Now grab the Inactive field for the next column in the grid. Set the delete condition to "YES" since we want to delete every customer who is currently inactive. Take a look at the design – we are deleting ***from the tblCustomer*** **table for every customer** ***Where the Inactive field is set to Yes***.

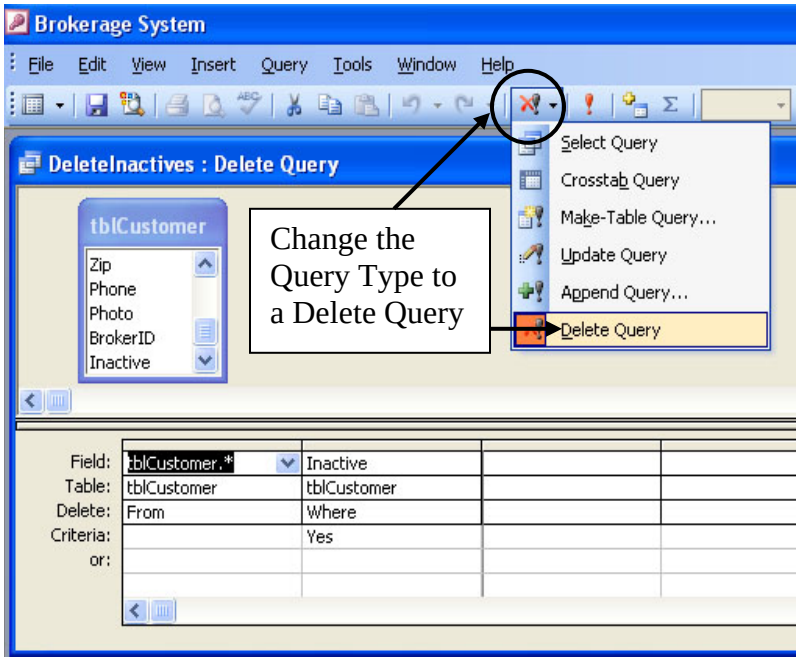


Figure 7.26 – You can select from several types of queries

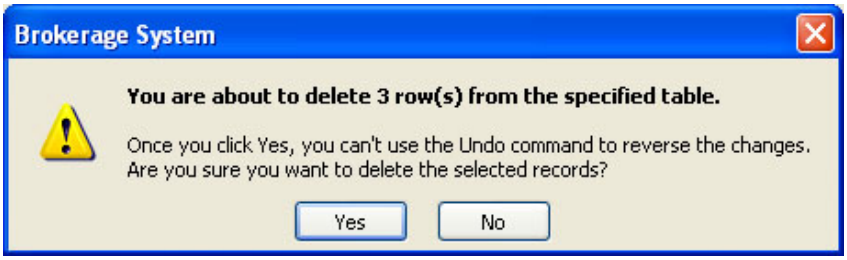


Figure 7.27 – Deletes require confirmation. Once the data is gone, it's gone. You DO have backups?

You should check the Customer table and confirm for yourself that these customers are gone. Is there anything else that needs deleting? How about all the stock purchases I recorded for this customer – do they need to be deleted? Well, actually they should all be

gone by now since we designed the relationship between the Customer and Trans tables to use ***Cascading Deletes***. If you recall, this means that deletions on the one-side of a relationship will automatically remove the associated data from the many-side.

If you didn't set the relationship between the tblCustomer and tblTrans tables to include the Cascading Deletes setting, you can set it before the delete query runs. Simply go to the Relationships screen, right click the line describing the relationship between Customer and Trans tables and select EDIT. Now you can check off the Cascading Deletes box and save the relationship diagram. Once cascading deletes are set, the deletion of a customer will automatically delete all references to stocks they bought.

7.9 Update queries

Like Delete queries, Update queries are most useful when you are updating a whole collection of records at once. If I simply wanted to change Bob Smith's name to Robert, I probably would open the form with his data and type Robert as his first name. Since the controls on a form are *bound* to the underlying tables, the change I make on the form will be reflected in the table as well,

It happens that one of our brokers has decided to join a new firm. We will need to shift all of his customers to Alice Chassaing who we hired to replace the departing broker, Mark Leeming. I suppose we could do this by hand, but Mr. Leeming has quite a few customers and it would be better if we automated the process with an Update Query.

Begin the query by selecting the Customer table for the query and adding the BrokerID to the design grid. Be sure to select Update Query as the query type. Notice that we are updating the BrokerID to B3456 for any customers that currently have broker B2345 (Fig. 7.28).

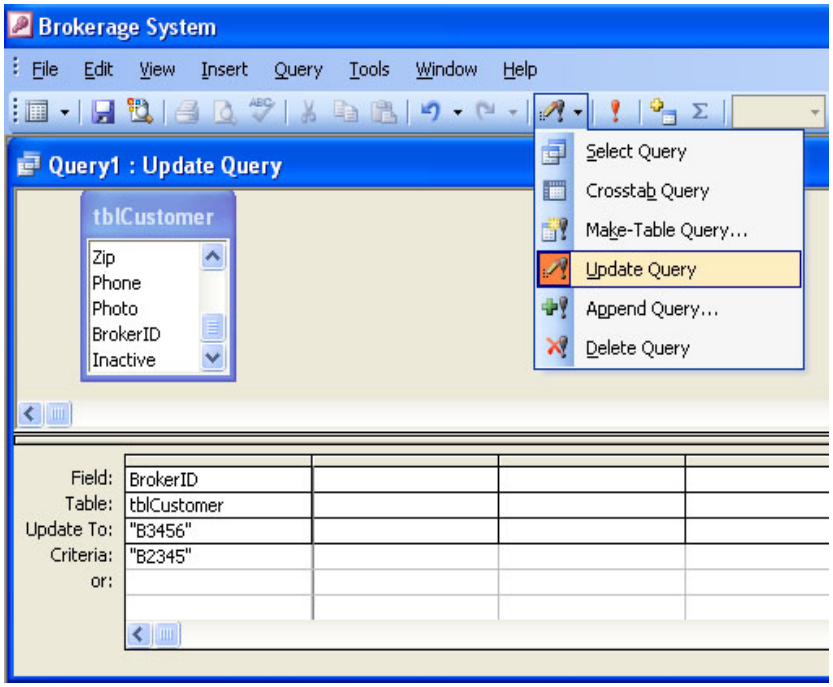


Figure 7.28 – Updating every customer who currently has Broker B2345 *to* Broker B3456

There's a warning that we are about to change a certain number of records. Once you click OK, the job is done.

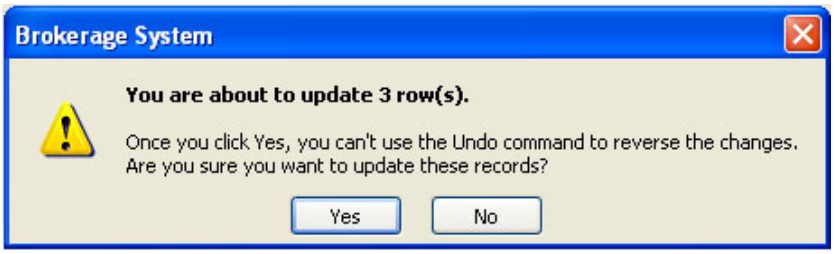


Figure 7.29 – As with Deletes, Access will warn you about an update

HINT: When running an update query, the design grid should only include the table being updated.

7.10 Working with Dates

A great deal of activity at our Brokerage firm revolves around dates. When did a customer buy that stock? For tax purposes, how long has each stock been owned? Who bought stocks last week, last month or last year? Who hasn't bought stocks for a long time?

Dates would be very difficult to work with if we had to work with them in our heads. If I told you my birthday was July 14, could you determine how many days I had left before my next celebration? It wouldn't be hard, but it would certainly be tedious. You would need to know how many days were left in the current month, then add the days in each month through June and then add the 14 days in July. The months between this one and June have different numbers of days and if February is included, you'll have to determine whether this is a leap year.

Microsoft takes some of the tedium out of dates by converting every date into a Serial Number. The serial numbers are based on January 1, 1900 which becomes day 1. January 2, 1900 becomes day 2 and so on…

Finding the days between two dates now becomes a simple subtraction problem.

Let's use November 8, 2005 and July 14, 2006 as our example.

Within Access these dates are converted to serial numbers

July 14, 2006	38912
November 8, 2005	38664
Days Between (just subtract)	248

What if you wanted to know how many weeks there are until my birthday? Simple, just divide the days by 7. For years? Divide by 365 – unless there are more than four years involved, in which case you might want to take the leap years into account by dividing by 365.25.

Months can be tricky. You might think of dividing the days into 30-day chunks. That will get you close to months – but as you know, more months have 31 days. Microsoft suggests a fairly involved formula for working with months. I've always gotten very close to their answers by calculating years (which will be an answer with a few decimal places) and then multiplying the answer by 12 (12 months for every year). So, for example, if the answer is 1.5 years, that will be 1.5 times 12 = 18 months.

Involving Today's Date

Access provides two functions that get you the current date. The Now( ) function will get both the date and time – which is great if you need to timestamp some transaction. If you just need the date, Access has the Date( ) function. As with all dates, these are stored internally as serial numbers.

Working with Specific Dates

If we need to ask about a particular date within a formula, be sure to enclose the date within number signs (#). Let's say we want to know who bought stocks during December of 2005. You would start a query and set the criteria for date purchased to:

Between #12/1/2005# and #12/31/2005#

Let's try a few date queries to be sure you're comfortable.

Example 1: How many weeks has each stock been held by customers? The question hasn't set any conditions for the query – it just wants to see the number of weeks. Our Trans table includes the Purchase Date (PurDate), but does not store the weeks. That makes sense, since the weeks would be different every time you looked at the data. Not a problem – we'll create a calculated field and name it "Weeks Held". It may display more decimal places than we want, but that's easy to fix by setting the Format property. The formula for the Week calculation will figure out the number of days between today's date and the date a stock was bought. That answer will be sliced into 7-day chunks to create weeks (Fig. 7.30). To format the answer, right click the formula (once you've entered it), select Properties and Format. To set the decimals to a fixed number of decimal places, choose Fixed as the format and enter the number of decimals you want in the answer.

Weeks Held : (Date() – PurDate) / 7

Here's the design:

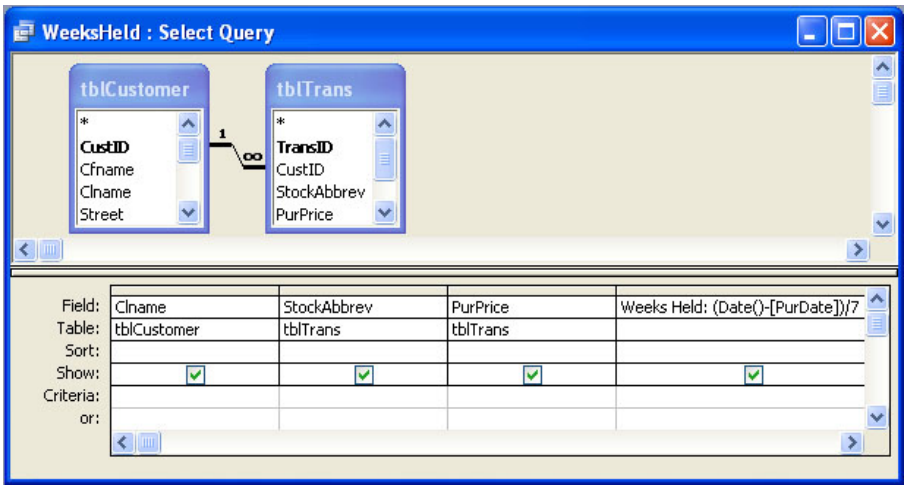


Figure 7.30 – This query determines the number of days since a stock was bought and then calculates weeks by dividing the days into 7 day 'chunks'

…and here's the run.

Figure 7.31 – The number of weeks each stock has been held

Example 2: Management has their eye on a particular broker who hasn't been very active recently – it's the new guy, Smolsky. They'd like you to check activity within the past two weeks. Since dates are stored as simple numbers, we can take today's date and just subtract 14 from it to get the date for two weeks ago (Fig. 7.32). For the date to be within 2 weeks, the Purchase Date has to be equal or greater than the date two weeks ago.

Here's the design:

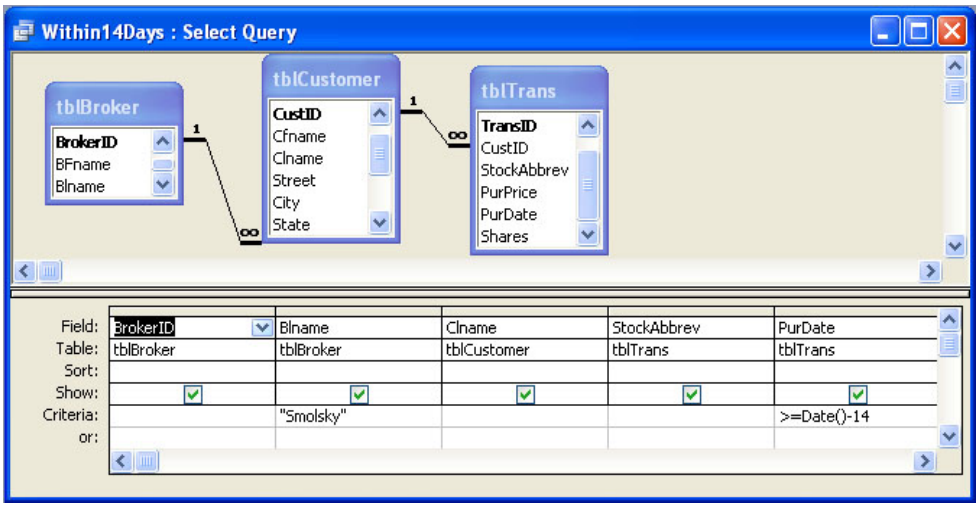


Figure 7.32 – Purchases handled by broker Smolsky within the past two weeks

…and the run:

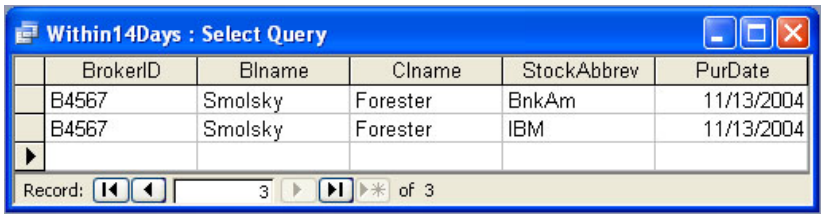


Figure 7.33 – Purchases handled by Smolsky within the past two weeks

Example 3: Here's a problem that doesn't relate to our brokerage firm, but one that falls into the category of "How long until something happens?" For example, how many weeks before a newspaper subscription expires, who is within 3 days of a doctor's appointment (need to call them), how long does someone have before their car lease expires? Let's look at a company that maintains the birthdates of their employees in a field named "BirthDate". The company is in contract negotiations with the employee union and a proposal for early retirement incentives has been floated by the union. Before responding to that proposal, the company decides to run a quick query to see a list of employees who are within 5 years of retirement (age 65). This is an easy problem if you know someone's age – a 60 year old will have 5 years to go. We just need to subtract the ages. So our formula needs to calculate the age of each employee and subtract that from 65. Then we need to set the criterion such that the answer needs to be less than or equal to 5. Calculating the age of these employees is a simple matter of figuring out how many days old they are (the difference between today's date and their birthdate) and slicing those days into 365 day chunks. I tossed the ".25" in to account for

leap years. If we hadn't done that, the answer would be off by one day for every four years of someone's age. This formula would go in the criteria row under the birthdate field.

```
65 - (Date()-[BirthDate] ) /365.25 <= 5
```

7.11 A glimpse of SQL (Structured Query Language)

At the start of this chapter I mentioned that databases talk a language called SQL. So far, Access has shielded us from needing to know SQL. That is about to change as we prepare to learn HTML in order to create web pages and then attempt to get our web pages to talk to our databases. The neat query grid won't be available to us once we are outside of Access.

When I say that Access has shielded us from SQL, that doesn't mean Access hasn't been using SQL. If you look at the properties of any form or report that displays data, you'll find that the source of data for that form/report is stated as a SQL statement (Fig. 7.34). The wizard asked us about our tables and the fields we wanted. It asked us about sorting and grouping. All of that activity led to the generation of a SQL statement. As an example, here's the Customer Form we built in Chapter 5. I right clicked on that square in the upper left of the form (in design view) and selected Properties. From there, I moved to the Data tab. Notice the SQL statement sitting in the "Record Source". You can only see a bit of it since it goes on for miles, selecting every field needed on the form.

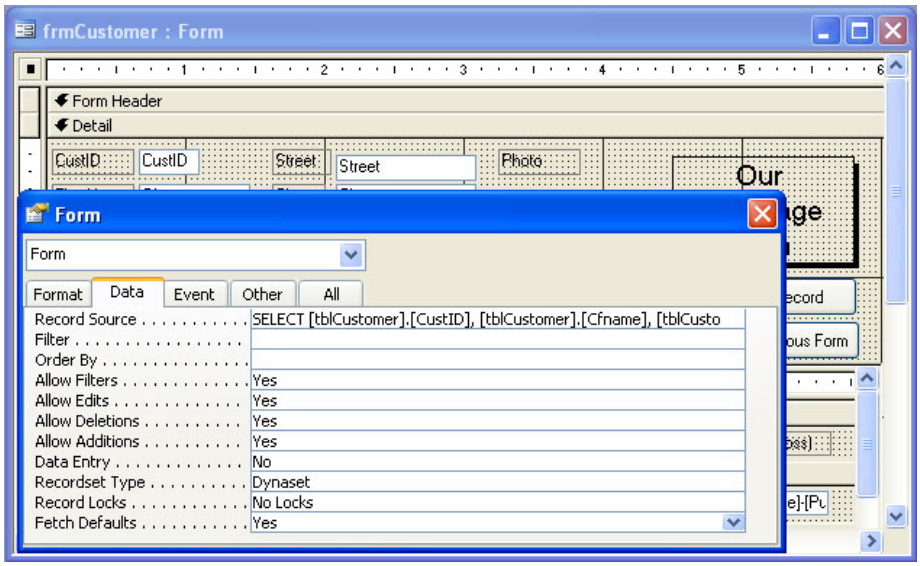


Figure 7.34 – Access uses SQL queries to obtain data for both Forms and Reports

Access has been writing SQL for each of our queries in this chapter. Let's check out the SQL with a simple query that selects a few fields from the Customer table and sets the condition to people living in Rhode Island. The list is sorted by Last Name. In the upper left there's a drop down list that provides a number of different views of this query – including a SQL view (Fig. 7.35). The SQL for this query is shown in Figure 7.36.

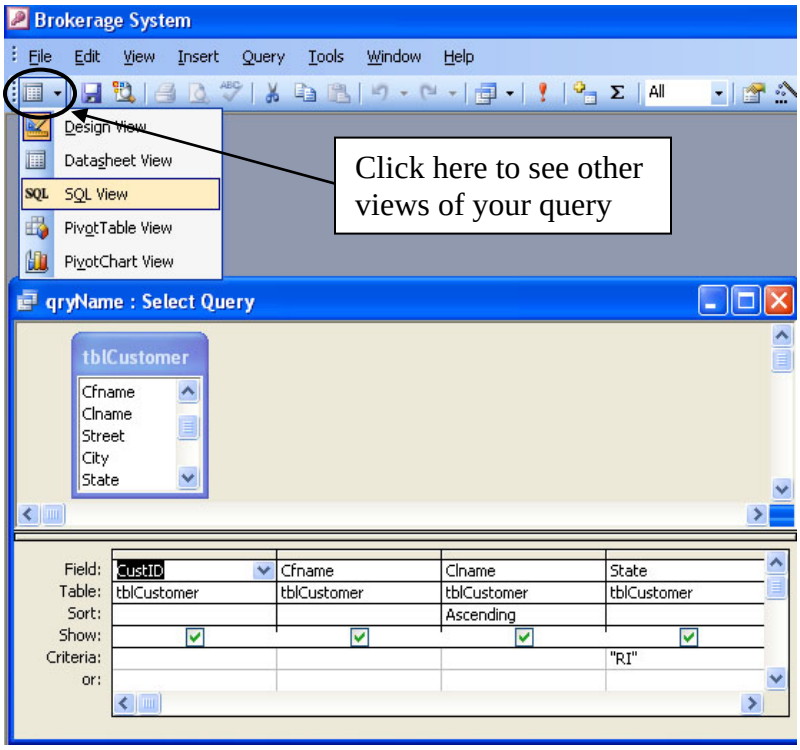


Figure 7.35 – Moving into SQL view is a great way to learn a bit of SQL

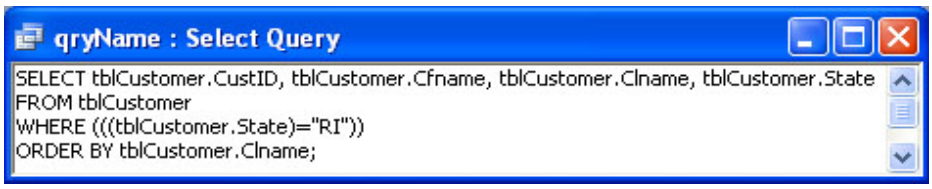


Figure 7.36 – Selecting customers from RI and sorting by last name

It might look complicated because Access prefaces every field name with the name of the table it comes from. Without this extra information, the SQL statement is fairly readable.

```
SELECT CustID, Cfname, Clname, State
FROM tblCustomer
WHERE State = "RI"
ORDER BY Clname ;
```

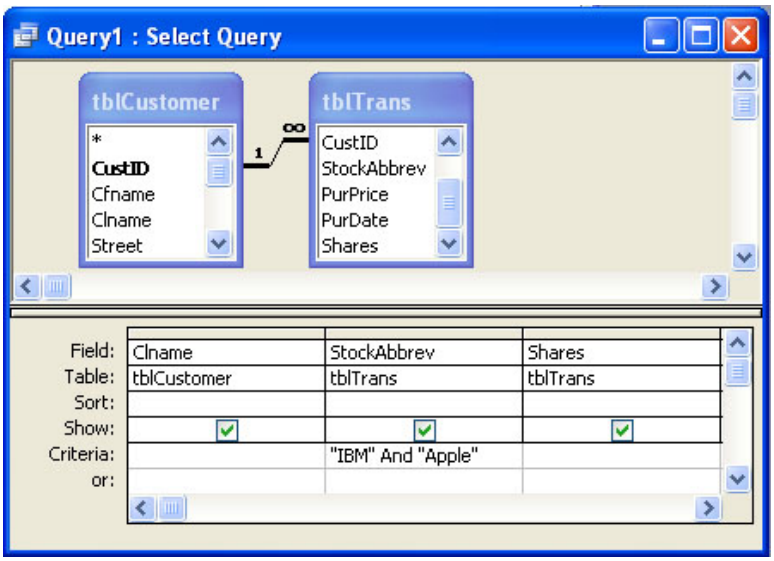
In Chapter 8 we'll show you just enough SQL to provide the power you need to create that Web/Database link that is driving so many activities on the web.

Keywords

Aggregate Query	Wildcards…
Cascading Deletes	*
Delete Query	?
Parameter Query	[ ]
Structured Query Language (SQL)	!
Update Query	#

Review Questions

1. Your manager has asked to see a list of customers who own IBM and people who own Apple. You create the following query.
   a) The query lists people who own both IBM and Apple
   b) The query lists people who own IBM and people who own Apple
   c) The query lists nothing, since the Stock field can't contain both IBM and Apple at the same time
   d) The query generates an error because both stock names should be uppercased



2. Which of the following will create a new column in a query that computes the "extended cost" of items purchased by customers? An 'extended cost' is simply the price of an item multiplied by the number purchased. So a customer who buys 3 candy bars for 60 cents each has an extended cost of $1.80. Let the new column read "Total cost"
   a) Total Cost = Price x Quantity
   b) Total Cost : [Price] * [Quantity]
   c) [Price * Quantity] = Total Cost
   d) Place "Total Cost" in the cell labeled 'field' and set the criteria to [Price] * [Quantity]

3. Which wildcard pattern would find customers whose names begin with "Ke" and have either a "V" or "W" as the third character?
   a) Like "Ke [VW] *"
   b) Like "Ke [V or W]"
   c) Like "Ke [V-W] *"
   d) Like "Kev*" or "Kew*"

4. Which wildcard would list every name EXCEPT those starting with an "S"?
   a) Like "[A-R]* or [T-Z]*"
   b) Like "[Not S] *"
   c) Like "[xS] *"
   d) Like "[!S] *"

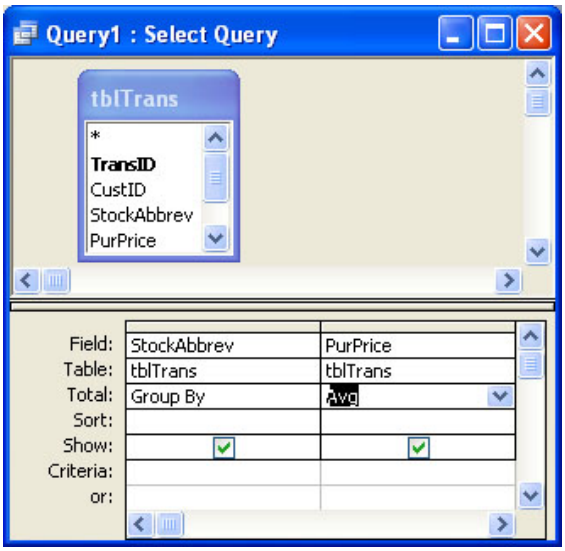
5. Axelrod Computers has gone out of business and no longer supplies parts to your store. You have been asked to delete Axelrod from the 'Supplier' table and to remove from the 'Items' table any items that Axelrod might have supplied to us. You decide to
   a) verify that the relationship between the Supplier and Items tables has "Cascading Deletes" selected. It is then a simple matter of deleting Axelrods from the Supplier table – and any item provided by Axelrods will automatically be deleted from the Items table
   b) delete Axelrods from the Supplier table, then turn **off** Referential Integrity. Once Axelrods is gone from the one-side of the relationship, the absence of Referential Integrity will require that all Axelrod items automatically get deleted from the Items table.
   c) delete Axelrods from the Supplier table, then turn **on** Referential Integrity. Once Axelrods is gone from the one-side of the relationship, the absence of Referential Integrity will require that all Axelrod items automatically get deleted from the Items table.
   d) delete Axelrods from the Supplier table, then delete Axelrods from the Items table

6. The brokerage firm had a problem with any stock purchase handled by one of our new brokers during April of 2005 – the broker has been taught how to correctly enter the purchases, but we need to find each of the transactions he was involved with for that time period. As part of the query, we need to locate stocks purchased in April. You should set the criterion for PurDate to:
   a) Between April 1, 2005 and April 30, 2005
   b) Between 4/1/2005 and 4/30/2005
   c) Between ?4/1/2005? and ?4/30/2005?
   d) Between #4/1/2005# and #4/30/2005#

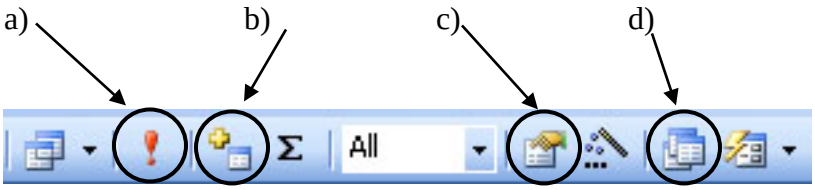
7. To list stock transactions made within the past 5 weeks, you could set the PurDate criterion to:
   a) Date() – ([PurDate] / 7) <= 5
   b) Date() – ([PurDate] / 7) >= 5
   c) (Date() – [PurDate]) / 7 <= 5
   d) (Date() – [PurDate]) / 7 >= 5

8. Queries that allow the user to enter criteria at 'run time' are known as
   a) interactive queries
   b) parameter queries
   c) run time queries
   d) referential queries

9. What does the following query do?
   a) groups the stocks together and determines the overall average price
   b) groups each stock's purchases and lists the average price of each stock
   c) determines the average price of each individual transaction
   d) averages the stock prices, but only for people who bought a stock many times



10. You have begun to design a query and have forgotten to include one table in the design grid. What tool on the toolbar will display the "Show Table" dialog box that lets you select tables for the query?



Exercises

Let's continue working on the Flights database you began developing in Chapter 4. A group of managers for the airline have asked for your help. Their interests lie mainly with scheduling and finances.

1. List all the passengers flying from Boston to Miami. Your list should include the Departure and Destination city, the passenger's last name, the date of departure and the ticket price.

2. Create the same list of passengers but, this time, only include passengers flying from Boston to Miami within the next 30 days.

3. Airport security has decided to pull some passengers aside for a closer look, based on a pre-established list. The list will include people whose names begin with D through those whose names start with K. Once again, do this only for passengers on the Boston – Miami flights.

4. Create a list that shows the total revenue for each flight.

5. Create a list that shows the total revenue – but this time, make it be the total revenue for each day.

6. Management wants a list that shows the security taxes collected on each flight. The security tax will be 5% of the ticket price.

7. A special security warning has required us to list passengers going from Boston to Miami on October 5 of this year – as well as anyone traveling from Boston on October 6 (regardless of their destination).

8. Flight 1234 has been cancelled due to a mechanical problem. Every passenger on that flight is being moved to flight 5432. Solve this problem by making sure you have a flight 5432. If you don't have this flight, just create that row of data. For the problem, the departure and destination city and date can be anything – just be sure the flight exists. Once you have

the flight, run a query that changes the flight number for every passenger currently on flight 1234 to be flight 5432.

9. Sorry, more mechanical problems. Flight 5432 will need to be cancelled and all passengers will get a refund. Assume the refunds have already been processed. It is now time to delete flight 5432 and all of its passengers.