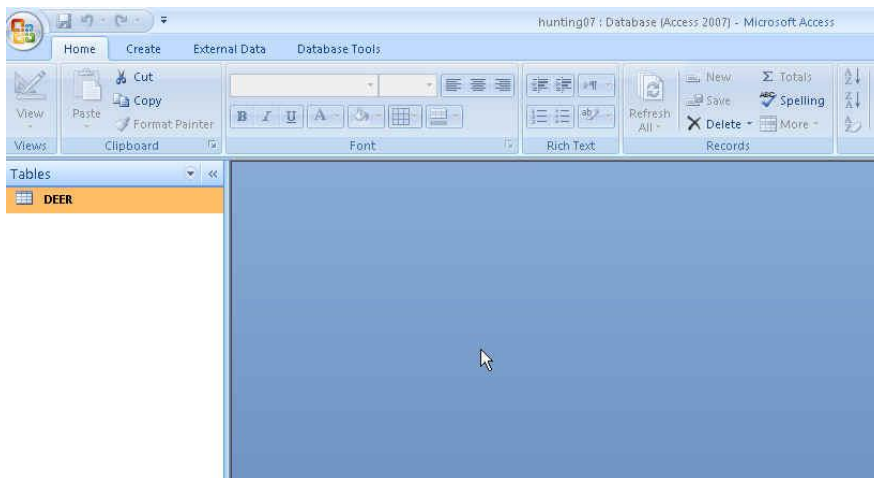


# A quick look around Access

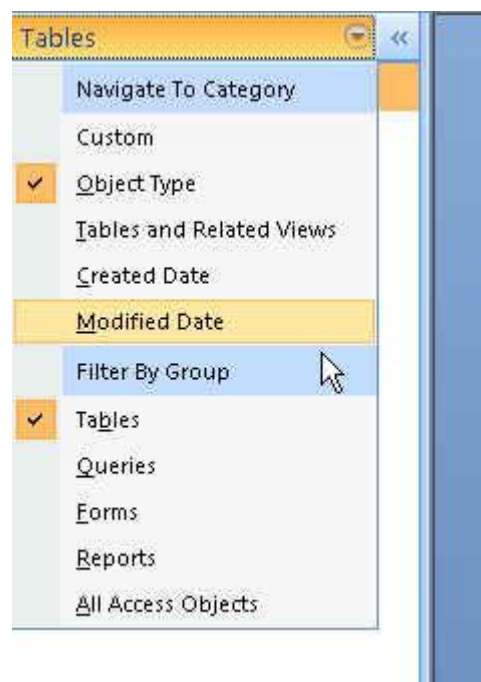
(versions 2007 & newer)

**Database:** A collection of related data objects, including tables, saved queries, and items for advanced programming. Access stores all of these items within one file (the newest versions use the “.accdb” extension and older versions use “.mdb”) Most often you will not get data in this format, but instead will need to import data from something like Excel or a delimited text file.

When you first open an Access database -- whether it's a new blank one or one that has 1 or more tables in it -- you'll see a menu of objects on the left. This picture shows the Hunting database, which has one table called “Deer”

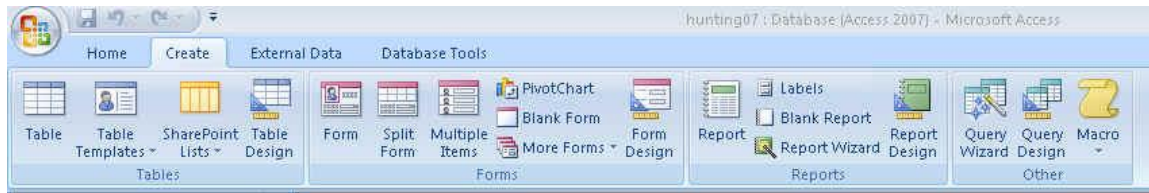


If you click on the little pull-down arrow to the left of the word “Tables”, you see a menu that allows you to select what will appear in this left-hand navigation box. You might want to change it to “object type” – this way it will group all the tables together, and all the queries together. Then it will give you different options and you'll need to put a check mark next to “Queries” so that it will display any queries that you save.



Across the top of the screen are the “ribbons”: Home, Create, External Data, Database Tools. Most of the time we will be using the tools under the “Create” and “External Data” ribbons.

## CREATE:



This ribbon contains the tools we’ll need to start a query or create a new table. It also has tools for building “forms,” which are useful for inputting data, and “reports”, which are essentially fancy versions of query results (think: business report). We’re not going to cover how to use either of those tools.

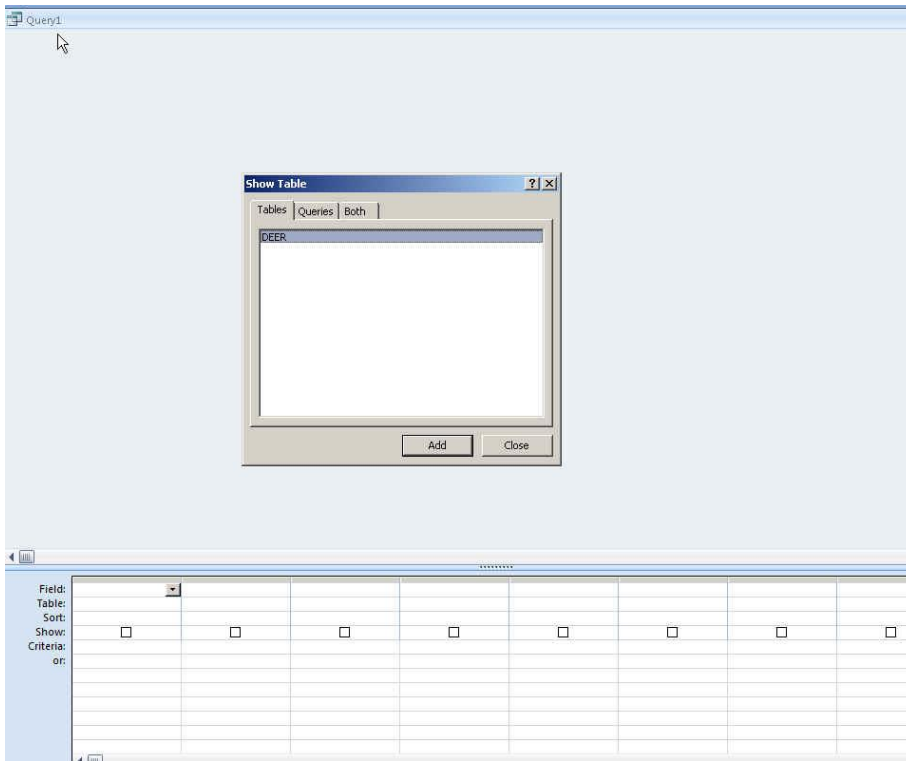
The create a new table tool (far left of the ribbon) will create a blank table where you can type information in directly, much like a spreadsheet. I find there’s limited use for this in data journalism, but on a rare occasion it is something that I use.

The “Query Design” button is the one I use most often. This is where we go to start building the questions we’re going to ask the data. You’ll see there is also a “Query Wizard”. The two buttons do many of the same things, although the wizard sort of babysits you through the process. The wizard does have some specialty queries (such as a crosstab query) that I use occasionally and I’ll talk about that more later.

We’re going to use the Query Design button because it allows us to build our queries from scratch using Structured Query Language (SQL). I think this is the best route to go because it forces you to learn and truly understand WHAT you’re doing, and not just learn which buttons to push.

## Running a query:

Start by going to the Create ribbon and pushing the Query Design button. Two new windows (one larger in the background and a small one in the foreground) will appear, like this:



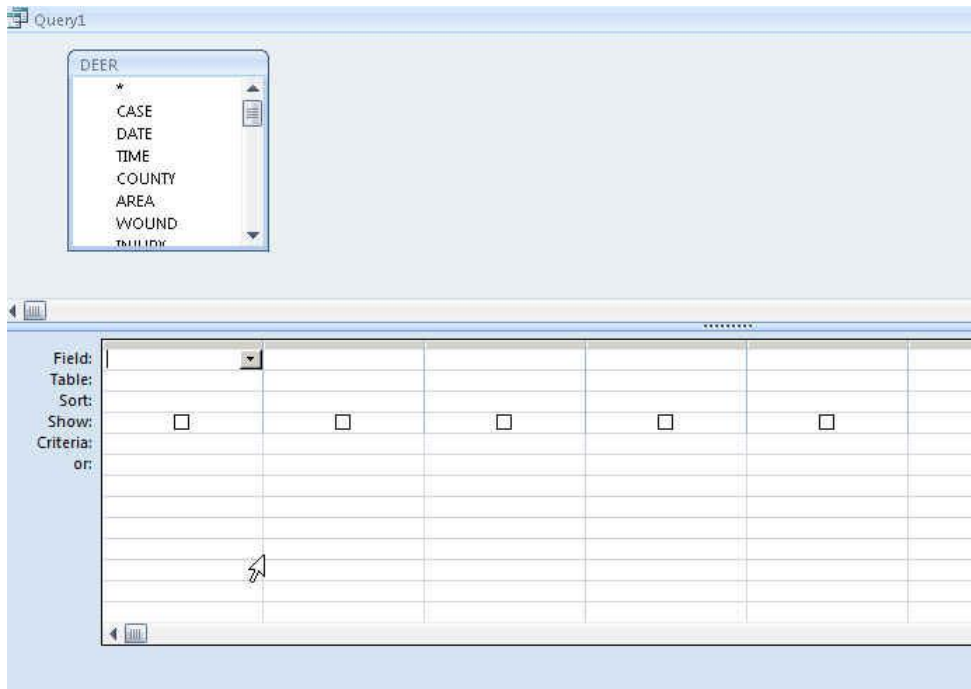
The smaller window is the “Show Table” box and it lists the available tables and saved queries in your database (notice the tabs “Tables”, “Queries” and “Both”).

Our Hunting database here only contains the one table called “Deer”

This is the point at which you choose which table(s) or query(s) you want to base your query on. In this case, it’s easy because we only have one table. When we get to databases with multiple tables that join together, we will need to choose more than one item here.

For this one, though, you’ll see that Deer is already highlighted (in blue), so you just need to push the “Add” button and then push the “Close” button to get rid of the Show Table window.

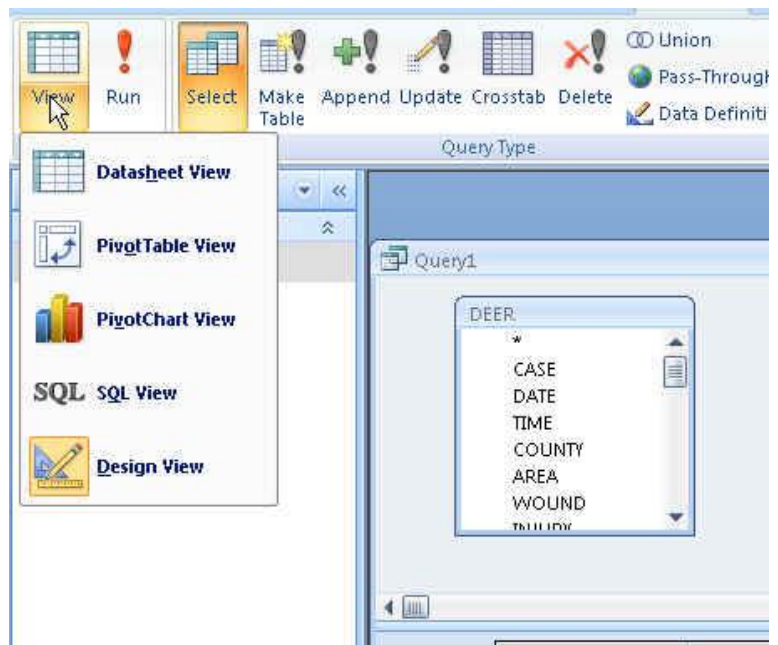
This will leave you with the Grid View, like this:



There are 2 primary ways to build a query in Access. The first is this Design View (sometimes I call it the Grid View).

The second is known as the SQL View.

To switch between the two views, click the “View” button in the upper left corner of the ribbon banner and you’ll see the different view options (Datasheet View is the one that comes up with the answer to your query, which we’ll see in a minute)



In this class, we're going to learn how to use the SQL View for a couple reasons: the first is that it will force you to learn Structured Query Language, which then enables you to use other database software (such as SQL Server or MySQL); the second is that if you're having trouble running a query, it's simple to copy and paste the SQL code into an email to send to the instructor or a friend to get help. You can't copy and paste the Design view.

Once you write your query in SQL, you can flip over to the Design View and you'll see that Access automatically populates the grid, too. As a result, this allows you to learn both of them at the same time.

And there are a couple cases where I find the Design view is much easier to use and we'll be able to build part of our query there and then flip over to the SQL view to finish it off. Likewise, I find there are some things that are nearly impossible in the Design View, but are quite easy in the SQL View.

Let's build a basic query and go over the various components/terminology and then we'll look closer at Structured Query Language.

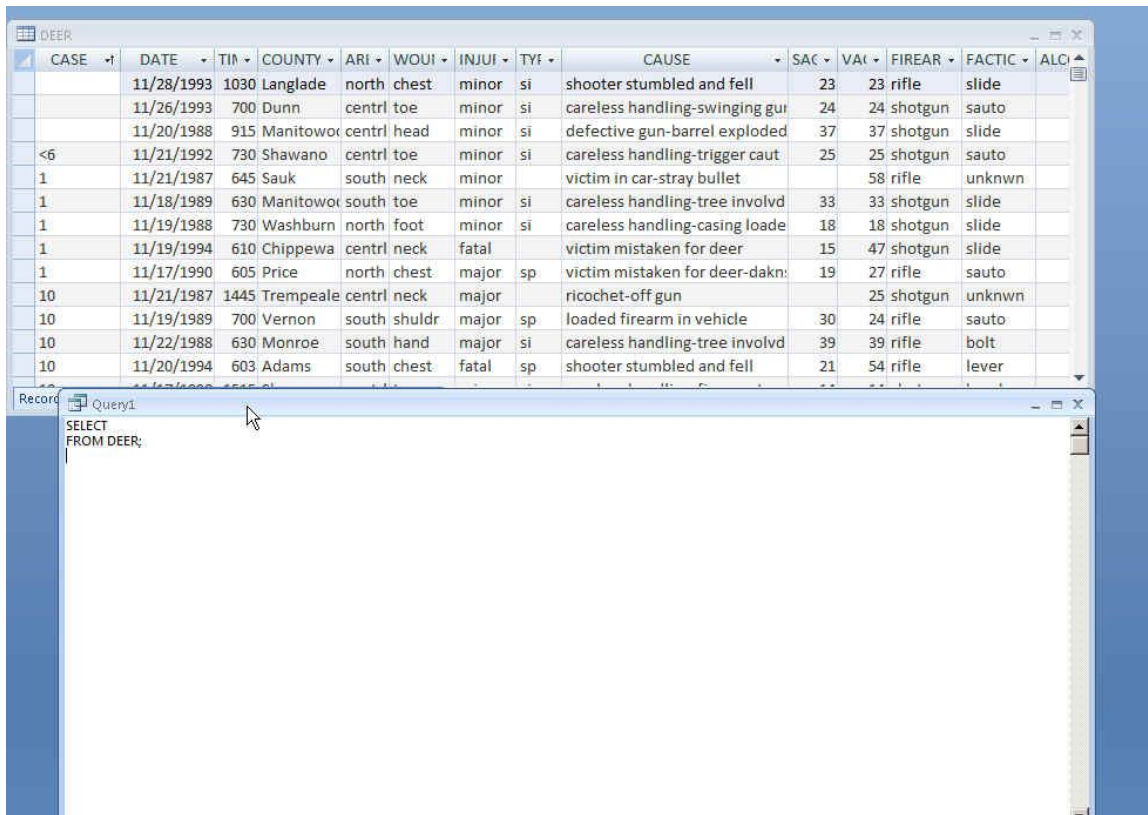
So we've already started our query. Go to the View button and switch to the SQL View and you'll get this:

A screenshot of a Microsoft Access query window titled "Query1". The window is in SQL View, and the text "SELECT FROM DEER;" is visible in the query design grid area. The window has a standard Windows-style title bar and a scrollable area below the text.

In a query we can tell Access to show us certain columns (i.e. fields) or all of the columns, and we can choose which rows we want returned based on criteria found in one or more of the columns.

What I find helps most is to be able to see the table columns while I'm writing my query. So -- while leaving the query window open -- double-click on the table, "Deer" and then move the table and query windows around so that you can see both at the same time. (put your cursor on

any corner or side of a window and then “drag” it in so that it’s smaller, in terms of width and/or depth). Here’s what mine looks like:



Now back to the query....Right now it says “Select” on the first line and then “From Deer” on the second line.

The SELECT line is going to be where we choose which columns we want in our answer.

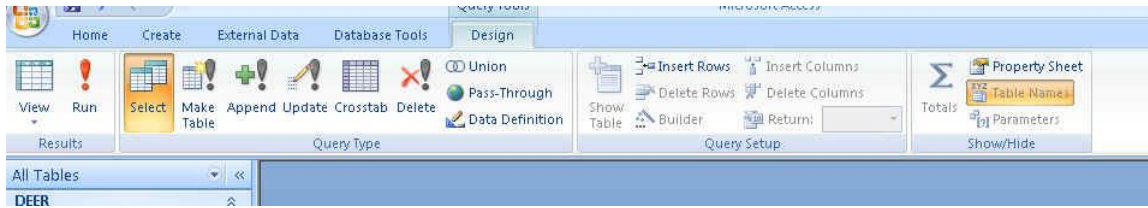
The FROM line is where we tell it what table(s) to use. We’ve already done that by selecting the Deer table when we first created our query. So we don’t need to mess with the FROM line at all.

Let’s start with something simple. I’d like to see a list of all the hunting accidents in our database, showing the date, the county where it occurred and the seriousness of the injury. In other words, I only want to see 3 of the columns (fields) in our database.

To do that, we’ll add to the select line:

```
SELECT DATE, COUNTY, INJURY  
FROM DEER;
```

In order to “run” the query, we need to push the Run button (red exclamation point), which is on the Design ribbon. Sometimes I find that the Design ribbon disappears and I have to click on the “Design” header at the top of the page to get it back. Here’s what the ribbon looks like:



When you click the “Run” button, it will switch the Query window over to the Datasheet View and you will have your answer.

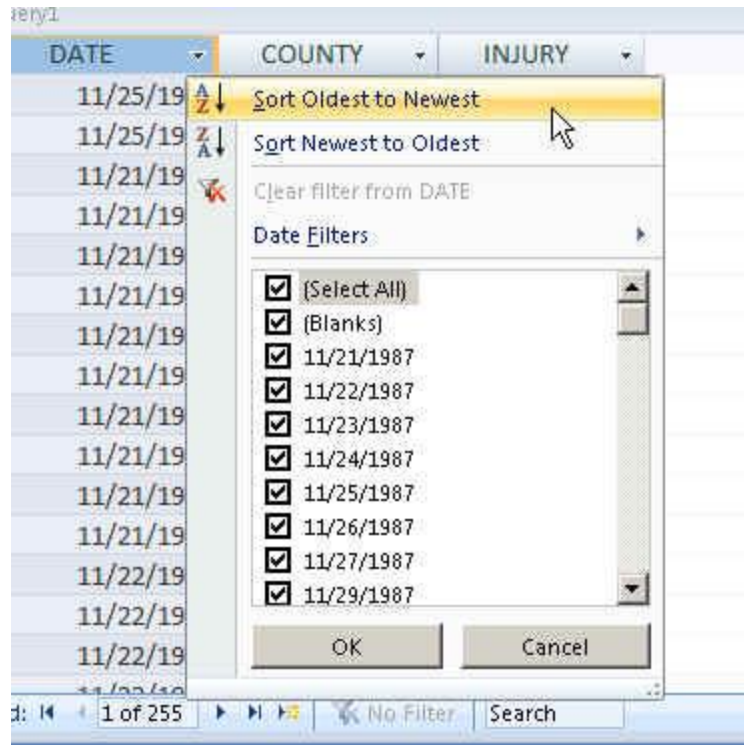
Note that the columns appear in the order that you typed them in the SELECT line: DATE, COUNTY, INJURY

Also note that it has returned all of the records in the table (Note in the example below that the bottom of the query window says “13 of 255 records” -- this means the cursor is on the 13th record)

DATE	COUNTY	INJURY
11/28/1993	Langlade	minor
11/26/1993	Dunn	minor
11/20/1993	Door	minor
11/20/1993	Marinette	major
11/20/1993	Sauk	major
11/20/1993	Crawford	minor
11/20/1993	Columbia	major
11/20/1993	Washburn	minor
11/20/1993	Vernon	minor
11/21/1993	Sheboygan	minor
11/21/1993	Dunn	major
11/21/1993	Portage	minor
11/21/1993	Waukesha	major
11/22/1993	Walworth	minor
11/23/1993	Sauk	fatal

Record: 13 of 255

There are a couple ways we can SORT our results. The easiest is to click on the pulldown arrow next to the field name in our query result. In this case, let's say we want to order it by date -- click on the arrow next to DATE and you'll get a window like this:



Also note that this window allows you to Filter your results, just like in Excel.

The second way is to go back to our SQL View (click on the View button in the far upper left corner of the screen) and add to the SQL like this:

```
SELECT DATE, COUNTY, INJURY  
FROM DEER  
ORDER BY DATE
```

Note that I removed the semi-colon after the table name (Deer) and then added the ORDER BY line of SQL. The default is that it will sort it ascending (oldest to newest, in this case). If we want it to go descending (newest to oldest), you simply say "ORDER BY DATE DESC"

We can make our query a little more interesting by choosing to limit which records we get back. Let's say we only want to see accidents in a particular county. Or we only want to see fatal accidents. Or we only want to see self-inflicted accidents. There are so many different possibilities here.

These types of queries require us to add another line to our SQL, called the WHERE clause.

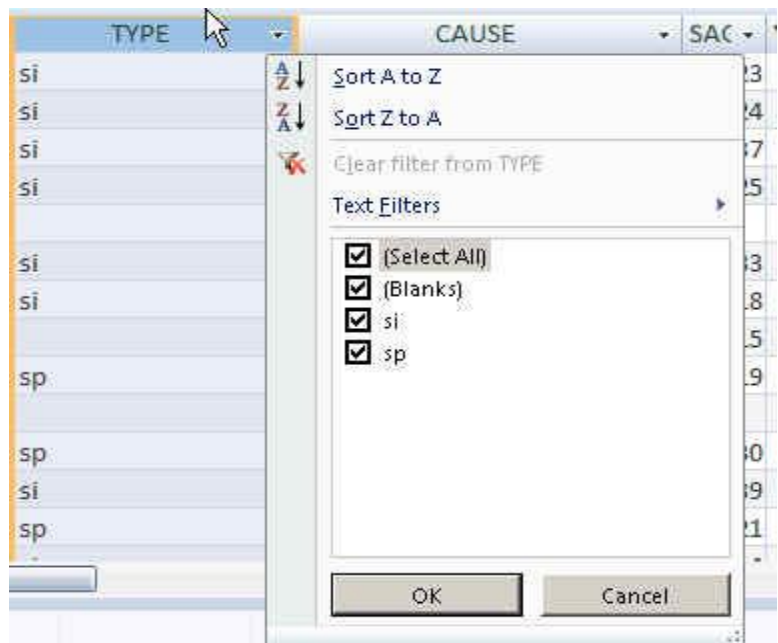


The WHERE clause is essentially a filter -- just like the filter I just pointed out on that menu and just like the ones in Excel. However, we're going to write it out in SQL.

The key to a WHERE clause is that you need to figure out which column contains the criteria you want to filter by and then you also need to know how that criteria is referred to in the column.

For example, let's say we wanted to see only accidents where the injury was self-inflicted. First we need to figure out which column contains that information. In this case, it's the column called "TYPE." Then we need to know how the data flags self-inflicted ones versus those where the shooter is not also the victim. So we need to look at our data.

An easy way to do this is to go to the table and find the column -- in this case "TYPE" -- and click on the filter arrow next to the header. It will bring up the sort and filter menu. Notice that it displays each of the potential values that exist in this column. In this case, it's telling us that the values are either "si", "sp" or that it's been left blank.



The record layout tells us that "si" stands for self-inflicted and "sp" stands for 'second person'

So if we wanted to only see the self-inflicted injuries, we would write a query that looks like this:

```
SELECT DATE, COUNTY, INJURY, TYPE
FROM DEER
WHERE TYPE= "SI"
ORDER BY DATE
```

Notice that my WHERE clause says the name of the field (type) and then uses the equal operator to say we only want records that exactly match the criteria. In this case the criteria is "SI". Note that "SI" is in quotes -- this is how we tell the computer that the criteria we're looking for is a text value stored in the database.

I've typed everything in capital letters, but that's not necessary. As long as everything (field names and criteria you are referencing) are spelled right, the case doesn't matter.

There are about a zillion ways we could set up the WHERE clause to filter our records. The syntax varies depending on whether we're dealing with text, dates or numbers. (More on that in a bit)

We can also use a wildcard in cases where we either don't know exactly how something is spelled or in cases where it might be very different.

Here's a good example of a wildcard. Notice our Deer table has a field called "CAUSE." This gives a sort of short narrative of what led to the accident. In the first few records we can see that there are several that appear to involve careless handling of a weapon. But just looking through the first few you can see there are lots of variations:

careless handling-trigger caut  
careless handling-swinging gun  
careless handling-casing loaded

Let's say we want to see all the records that have the term "careless handling" in the cause field.

To do that, we'll use a wildcard operator. In Access, the wildcard operator is an asterisk. And instead of the equal sign, we use the word "LIKE".

Here's what the SQL looks like:

```
SELECT DATE, COUNTY, INJURY, CAUSE  
FROM DEER  
WHERE CAUSE LIKE "CARELESS HANDLING*"
```

Our answer gives us lots of variations:



INJURY	CAUSE
	careless handling-swinging gun
	careless handling-trigger caut
	careless handling-rifl on foot
	careless handling-hand at muzz
	careless handling-hand on muzz
	careless handling-trigger caut
	careless handling-fell asleep
	careless handling-tree involvd
	careless handling-dropped gun
	careless handling-tree involvd
	careless handling-safety off
	careless handling-tree involvd
	careless handling-tree involvd
	careless handling-gloves, tree
	careless handling-tree involvd
	careless handling-loaded rifle

But what if there are others that we've missed? We can widen our net, so to speak, by putting wildcards on both sides of the word "careless", like this:

```
SELECT DATE, COUNTY, INJURY, CAUSE
FROM DEER
WHERE CAUSE LIKE "**CARELESS**"
```

In this case, we end up with the same number of records -- 55 -- as we did with the first query. But it's worth casting a wider net like this just to make sure you don't miss something.

My favorite gem in here is the accident on 11/18/1990 in Trempealeau County. 😊

11/17/1990	Dallam	major	careless handling-treadmill
11/17/1990	Shawano	minor	careless handling-finger on trg
11/18/1990	Clark	major	careless handling-glove invold
11/18/1990	Trempealeau	major	careless handling-wife involved
11/20/1990	Jefferson	fatal	careless handling-tree involv
11/23/1990	Sheboygan	major	careless handling-ricochet
11/25/1990	Green Lake	major	careless handling-gun on shldr

We can also filter by more than one criteria in the WHERE clause. These two or more criteria could be in the same field or they could be in different fields. The syntax is different depending on which it is.

The syntax for all this is called Boolean Logic. We primarily use the AND and OR operators to set the limits.

So for example, let's say we want to see all the accidents that were in St. Croix and Pierce counties. In this case, the two criteria we're looking for are both contained in the same column (but never together in the same record). So we need to use the OR operator. Because we are asking it to show us the records where the accident is either in St. Croix County OR in Pierce County.

The syntax for that would be:

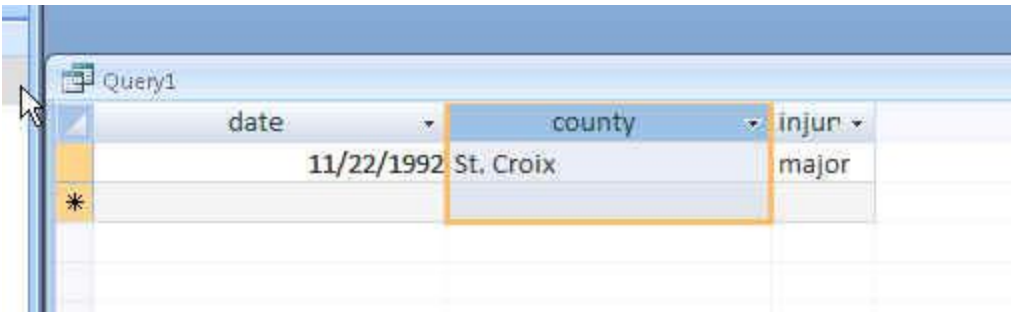
```
SELECT DATE, COUNTY, INJURY
FROM DEER
WHERE COUNTY= "ST. CROIX" OR COUNTY= "PIERCE"
```

Note that I repeated the field name for the second criteria (this is required -- you'll get an error if you forget to include it!). Also note that you need to spell the county names exactly as they appear. In this case, I've listed St. Croix with the period after "St". That's how it shows up in the database. If the records in the database did not include the period, I wouldn't get any results from St. Croix with the query I've written above.

Now let's say I want to see all the accidents in St. Croix County that were self-inflicted. In this case, I'm looking for two criteria that are contained in separate columns -- and I want both criteria to exist in the same record. So this is a situation where I need to use the AND operator.

```
SELECT DATE, COUNTY, INJURY, TYPE
FROM DEER
WHERE COUNTY= "ST. CROIX" AND TYPE= "SI"
```

Here's what my results look like:



The screenshot shows a database query result window titled "Query1". The window displays a table with three columns: "date", "county", and "injur". The first row of data shows the date "11/22/1992", the county "St. Croix", and the injury type "major". A yellow asterisk is visible in the left margin of the first row. The "county" and "injur" columns are highlighted with a blue selection bar.

date	county	injur
11/22/1992	St. Croix	major

Now what if I had screwed up that query and used the OR operator instead of the AND operator. Try it out and see the results. Instead of 1 record, you'll get 88 records -- because you'll get all the St. Croix records (regardless of whether it was self-inflicted or not) and you'll get all the records from other counties where the accident was self-inflicted (but you won't get the records for those other counties where it was not self-inflicted).

The key takeaway there is that one little word change can give you a completely wrong answer. So you need to not only pay close attention to your SQL syntax, but also look at your answer carefully to assess whether it's what you had expected.

## CHEAT SHEET:

**To start a new query:** Go to the Create ribbon and click “Query Design”

**To get to SQL view:** Go to the View button in upper left corner.

**To save a query:** With the results open, right-mouse click on the table and choose “Save”. Once saved, it will appear in the list along the left where you now see the “Deer” table.

### Various WHERE options:

COMMAND	SYMBOL USED	EXAMPLE
Wildcard:	Asterisk and Like	Cause like “careless*”
Text:	Quotes	county= “Washington”
Date:	Hashmarks (pound sign)	Date = #12/1/04#
Time:	Hashmarks (pound sign)	Time = #2:00 pm#
Number:	no special symbols	SAGE = 5
Blanks:	NULL	date is NULL
Don’t include Blanks:	NOT NULL	date is NOT NULL
Does not equal:	<>	county <> “Washington”
Does not equal:	NOT	County is not “Washington”
Greater than:	>	Time >#2:00 pm#
Less than:	<	Date < #12/1/04#
Greater than or equal to:	>=	Sage >= 20
Less than or equal to:	<=	Date <= #12/1/04#

### SQL SYNTAX:

Here are the five basic lines in Structured Query Language, in the order they need to be used in a query, and denoted in parentheses is whether it is required or optional.

SELECT (required)  
FROM (required)  
WHERE (optional)  
GROUP BY (optional)  
ORDER BY (optional)

About each one:

SELECT -- this is where you list the fields (columns) you want displayed in your answer. If you want to see all the fields, you can simply type “SELECT \*” (asterisk represents all the fields). The things you put here don’t necessarily need to be a field exactly as you have it displayed in the table. It can be a calculation based on a field (the sum of all the values in a field or the average of all the values), it can be a math calculation between two or more fields (i.e. number of students divided by number of teachers to get a student-teacher ratio). Note: if any of your field names have spaces in them or start with a number or are the same as a “reserved” word in Access -- such as the word “select” -- then you will need to put brackets around the field name. Here’s an example:

SELECT county, [accident type], date, injury

FROM -- this is the name of the table(s) or query(s) that holds the data you are querying. If you are just pulling data from one table it will be simply the word FROM and the name of your table. If you are pulling data from more than one table -- this is referred to as a "join" -- the syntax for the FROM line will be more complex. That will be covered later in the semester.

WHERE -- this is how you filter which records you want to appear in your answer or be calculated as part of your answer. The syntax of this line will vary greatly depending on what criteria you are using to filter.

GROUP BY -- this is used when you want to summarize or aggregate your data. For example, let's say you want to count how many accidents occurred in each county. You would want an answer that shows each county (listed once) and the number of accidents in the 2nd column. In this instance you would "group by" the field called County. We'll address this in greater detail in another handout.

ORDER BY -- this is an optional line that you can use for setting the order of the records in your answer. For example, you can order them by date (oldest to newest), or you can order them by the county name, etc. The default is that it will sort ascending (youngest to oldest or 1 to 100 or A to Z). If you want it to go the opposite way -- descending -- you simply add the word "DESC" after the field name.

You can also do a multiple order by using more than one field. For example, the following query would first order them by the date and then when it finds multiple records on the same date, it would then order those records (on that one date) by the county name, alphabetically:

```
select county, date, injury, type
from deer
order by date, county
```