HUNTING ACCIDENT DATA:

When Bob Imrie, an Associated Press reporter in Wausau, WI, went searching for information about hunting accidents, he ended up right on target. The state filed hardcopy reports on every accident detailing information such as weather, topography and the number of pieces of orange the victim was wearing. Imrie had no way of digging out trends from the voluminous hard-copy reports and he had never used a computer database. But with some help over the phone from some nerdy colleagues, he created his own database to look at trends in hunting accidents in Wisconsin.

The record layout, which is the map to any database, is below. It shows the fields Imrie created in the database, whether they were characters or numbers and the width of each field:

1	CASE	Character 10	This is an identifier he created
2	DATE	Date 8	Date of accident
3	TIME	Numeric 4	Time of accident
4	COUNTY	Character 11	County of accident
5	AREA	Character 6	Region of state (he created the areas)
6	WOUND	Character 6	Part of body wounded
7	INJURY	Character 5	Severity of injury
8	TYPE	Character 2	Self-inflicted, or other
		(si=self-inflicted, sp=second person)	
9	CAUSE	Character 30	Description of cause
10	SAGE	Numeric 2	Age of shooter
11	VAGE	Numeric 2	Age of victim
12	FIREARM	Character 7	Type of firearm
13	FACTION	Character 6	Faction of gun
14	ALCOHOL	Character 3	Alcohol involved
15	ALCOLEV	Numeric 5	Alcohol level
16	WEATHER	Character 11	Weather
17	TOPOGRO	Character 10	Typography
18	SEXPER	Numeric 2	Years of shooter's experience
19	VEXPER	Numeric 2	Years of victim's experience
20	SGRADUATE	Character 3	Shooter graduate of safety school
21	VGRADUATE	Character 3	Victim graduate of safety school
22	SSEX	Character 1	Sex of shooter
23	VSEX	Character 1	Sex of victim
24	GUNBRND	Character 13	Brand of gun
25	GUNGUAGE	Character 10	Gauge of gun
26	TEMP	Numeric 2	Temperature
27	MUZDIS	Numeric 5	Muzzle distance
28	LAND	Character 3	Private or public

29	VORANGE	Numeric 1	Number of pieces of orange worn by victim
30	GUNSIGHT	Character 5	Open, scope or other
31	VACTIVTY	Character 10	Victim's activity
32	LOCATION	Character 6	Location of victim
33	PRECIP	Character 3	Precipitation

Some questions to get us started using basic SQL:

1) How many people were injured or killed in incidents that occurred in Jackson County?
2) How many were injured or killed in incidents that occurred in St. Croix County and Pierce County combined?
3) How many people were injured or killed in incidents where the cause indicates some type of careless use of the weapon?
4) Do a query to show all the injuries occurring physically below, and including, the ankle. How many injuries were there?
   Hint: here's a list of all possible wounds: abdomen, ankle, arm, back, butt, chest, ear, eye, face, finger, foot, groin, hand, head, hip, knee, leg, neck, scrotum, shoulder, stomach, thigh, thumb, toe.
5) How many injuries or fatalities were self-inflicted, caused by shooters older than 50?
6) How old was the oldest victim in Jackson County? What wound did they have? How old was the youngest? What wound? (hint: this might require more than one query)
7) How many injuries/fatalities occurred overall on the first day of deer hunting season (November 19, 1994)?
8) How many injuries/fatalities involved a shooter under age 18? Of those, how many were fatal?
9) How many injuries/fatalities involved either a female victim or a female shooter?
10) How many self-inflicted incidents involved a shooter who did NOT take a gun safety class ?
11) How many fatalities involved alcohol?

Group By Queries:
So far we've merely filtered our data to get answers to our questions, but most of the time you'll want to ask a question that requires you to summarize either all of your data or part of it. This is where you need to do what's called a group by query. It's essentially the same thing as a pivot table in Excel, but is immensely more powerful and flexible.

Let's step back a minute and talk about when we would need a group by query:

Typically the records in your database will each represent one thing (1 hunting accident, 1 baseball player, 1 crime incident, 1 campaign finance donation) and you may want to count them or summarize them or "group" them and compare the groups.

So questions that might require a group by query would be like:

Hunting accidents -- Which county had the <u>greatest number</u> of injuries and fatalities? Or <u>How many</u> victims were women, compared to how many were men?
Baseball salaries – Which team paid <u>the most</u> in salaries last year? Or What are the <u>average</u> salaries for each position?
Crime incidents – What type of crime <u>occurred most often</u> last year? Did they occur <u>more often</u> at night than during the day?
Campaign finance – Which donor gave <u>the most</u> money? Or which candidate received <u>the most</u> money? Or which candidate received <u>the greatest number</u> of contribution checks?

I've underlined the key words that should tip you off that you might need a group by query to get the answer. Notice how many of these questions ask you for one thing in your answer – i.e. "which team paid the post". Although you're asking a question that implies you need just one thing in your answer, you really will need to see ALL the teams to put them in order based on the total money they paid out. Then whichever team lands at the top of your list, is the answer to your question.

Before we do a group by query, let's look at some of the FUNCTIONS that we are going to use as part of our group by queries.

The main ones that you will find yourself using are:
COUNT – this will count the number of records
SUM – this will sum the values in whatever field you've selected
AVG – this will average the values in whatever field you've selected

Let's put these into action in the Hunting database, just to see how they work.

Remember that the Hunting database is one record for each person injured or killed in a hunting accident in Wisconsin between 1987 and 1994.

So we can use the COUNT function to count how many injuries/deaths there were overall, by using this query:

```
SELECT COUNT(*)
FROM DEER
```

This query will return one number (87) because that’s all the query asked for. It doesn’t ask to see any of those records. Notice that I put an asterisk inside the parentheses.

The COUNT function doesn’t require us to specify a particular column (unlike the SUM and AVG functions). You can do that, however I’d advise against it. If you did put a column name in there instead of the asterisk, it will only count the records where that column has a value (it will ignore the NULL or blank values).

This database doesn’t have a good field for using the SUM function (a dollar field is what you’ll use it for most often). But the AVERAGE function works the same way. So let’s find out the average age of the victims (the field VAGE in the hunting database)

```
SELECT AVG(VAGE)
FROM DEER
```

This query also returns one number. In this case, it’s 30.8. In other words, the average age of the victims was about 31 years.

The two queries we just run generated numbers based on all the records in the database.

We can limit which records it does it’s work on by adding a WHERE clause:

```
SELECT AVG(vage)
FROM DEER
WHERE county=’Marathon’
```

Let’s say we want to count the number of injuries/fatalities in each county. In other words, we want one record for each county, showing the number of victims in each. For this, we’ll use the COUNT function, along with a group by.

Let’s try this one out first and then we’ll explain how it works. Here’s the SQL syntax:

```
SELECT COUNTY, COUNT(*)
FROM DEER
GROUP BY COUNTY
```

You’ll see that we got 64 records back – that’s the number of counties in Wisconsin.

You’ll also see that we only have 2 columns in our answer. Notice that our SELECT line only has two columns – the name of the county and the result of the COUNT.

Here’s what we just did: Because we did NOT have a where clause, it looked at all the records in our database and then it divvied them up into virtual “piles”; all the Adams county ones in a pile; all the Ashland county ones in another, etc.
Then it went through and counted the number of records in each “pile” and returned that number in our second column.

Notice that “county” (the name of the column) appears in BOTH the SELECT line and the GROUP BY line. Why? That’s one of the hard and fast rules of SQL. Whatever criteria (or field) you are using to dictate your groups or “piles” must appear in both the Group By and the Select lines. AND …you can’t have any other fields in the select line (only a function like COUNT or SUM or AVG)

It returned the answer in alphabetical order by the county field (the first field) because that’s the default when you don’t include an ORDER BY statement.

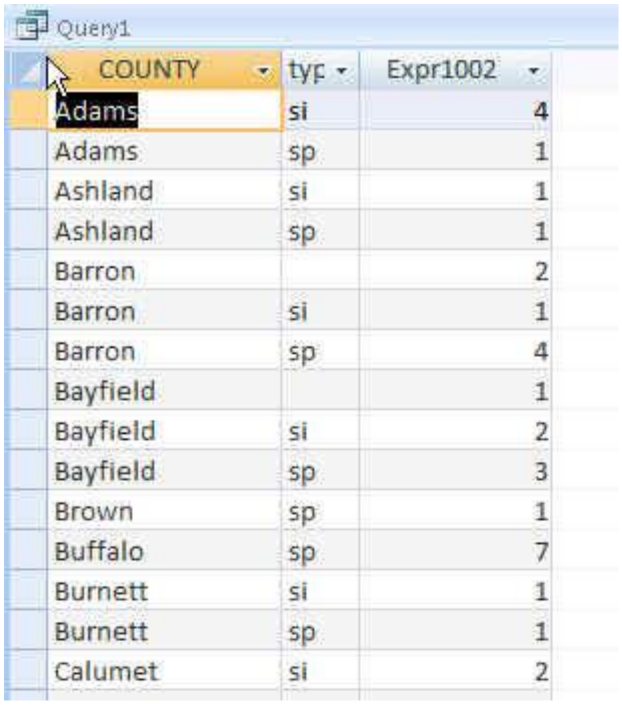
So if we wanted to adjust this query so that we see which county had the MOST injuries/fatalities at the top of our list, simply add an ORDER BY, like this:

```
SELECT COUNTY, COUNT(*)
FROM DEER
GROUP BY COUNTY
ORDER BY 2 DESC
```

Why does it say ‘ORDER BY 2 DESC’?
We could say “ORDER BY COUNT(*) DESC”, but database programs often allow you to use a little shortcut here and just refer to the column number – the column in our answer. In this case, the name of the county is column one (it’s the first item in our SELECT line) and the COUNT result is our second column. So we want the answer to be displayed largest to smallest, according to that COUNT. You can use this shortcut anytime you use ORDER BY. It doesn’t need to be a group by query.

What if we want to create our “piles” based on more than one criteria? For example, what if I wanted to see the number of victims in each county that were self-inflicted and the number that were second person. A cross-tab query would be a fancier way to do this, but let’s just see what happens when we put these two things together in a regular group by query:

```
SELECT COUNTY, TYPE, COUNT(*)
FROM DEER
GROUP BY COUNTY, TYPE
ORDER BY 1
```



Query1

COUNTY	typ	Expr1002
Adams	si	4
Adams	sp	1
Ashland	si	1
Ashland	sp	1
Barron		2
Barron	si	1
Barron	sp	4
Bayfield		1
Bayfield	si	2
Bayfield	sp	3
Brown	sp	1
Buffalo	sp	7
Burnett	si	1
Burnett	sp	1
Calumet	si	2

You'll see now we have more than 1 record for each county. In the case of Barron County, we have 3 – because there were 2 records where the TYPE field was not filled in, 1 where the injury was self-inflicted and 4 where it was a second person.

Probably one of the most confusing things for a newbie learning SQL is this idea that with a group by query you can't see everything in one swoop.

I've seen reporters flummoxed by this because they want to find out which county had the most victims, but they also want to be able to see – at the same time –whether the majority of them were self-inflicted or not.

The problem is that the query we just ran won't give you both answers at the same time. Notice that if you sort this last query by the COUNT field, you'll end up with Buffalo County at the top – but this isn't showing ALL the victims in Buffalo; it's only showing the 7 that were second-person.

You either need to run the two queries separately or get into more sophisticated queries (like a cross tab query). As a beginner, I'd recommend you stick with simple queries and just constantly remind yourself that each group by query is only going to tell you one thing.

The best way to approach Group By queries is to first draft your question, then think about what you expect the answer to look like.

Let's try out another question….
Which county had the most fatalities?

Let's deconstruct our question. First, we're asking it to ONLY look at fatal accidents. So that means we need to filter our data – in other words, use a WHERE statement.

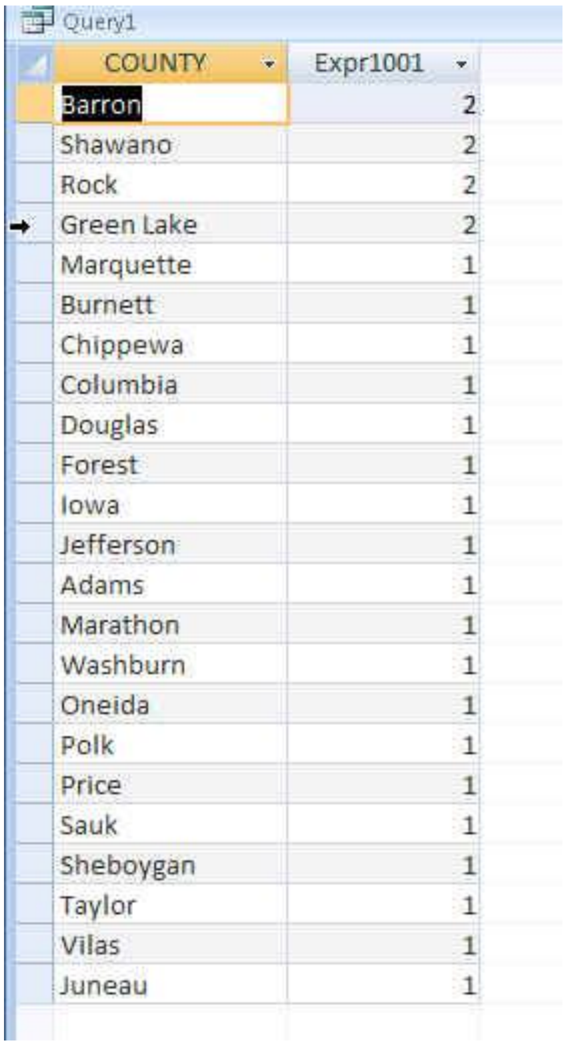
Second, we're asking for the name of a county. And in order to see which county had the MOST, we need to see all the counties in a list. So we're going to need the field COUNTY in our SELECT line.

Finally, we want a number for each county – so we need to COUNT the records. And because we're counting, we need to use GROUP BY.

And we'll need an ORDER BY putting the records in descending order by the COUNT field so we can quickly see which county had the most fatalities.

So that means our SQL would look like this:

```
SELECT COUNTY, COUNT(*)
FROM DEER
WHERE INJURY=  'FATAL'
GROUP BY COUNTY
ORDER BY 2 DESC
```



COUNTY	Expr1001
Barron	2
Shawano	2
Rock	2
Green Lake	2
Marquette	1
Burnett	1
Chippewa	1
Columbia	1
Douglas	1
Forest	1
Iowa	1
Jefferson	1
Adams	1
Marathon	1
Washburn	1
Oneida	1
Polk	1
Price	1
Sauk	1
Sheboygan	1
Taylor	1
Vilas	1
Juneau	1

The best way to learn Group By queries is to practice, practice, practice. So here are some other questions to answer:

Include both the answer and the SQL you used for the query or queries used to get that answer.

12) What was the most common wound?

13) What were the 3 most common self-inflicted wounds?

14) What was the most common cause? Is this answer reliable? Why or why not?

15) Which brand of gun was involved most often?

16) What is the average age of victims who shot themselves?

17) What is the average age for the shooters in butt-injury incidents?

18) On which date were the most victims hurt/killed? After getting this answer, then run a query to find out where those accidents occurred. Which county(s) had the most on that date?

19) What percentage of victims were wearing 2 or more pieces of orange?

Created by MaryJo Webster
@MaryJoWebster
Mjwebster71@gmail.com