ILDEBTORS

FIRST STEP when opening this database in Access 2007 is to click on the Security Warning and choose “Enable this content”

This Access file contains two databases that the St. Louis Post Dispatch obtained for a story back in the 1990s. Yeah, I know it’s old data but it’s a great example of the things you would have to deal with when trying to match two databases that weren’t intended to go together.

In this case, they were writing about the state of Illinois’ not doing very well at collecting money from parents who had not paid their child support obligations. A reporter wondered that if the state wasn’t doing so well overall, could there be state employees who had outstanding child support debt? So the reporter got the database of “orders” identifying each time a judge issued an order against an individual for an unpaid child support debt. Then they got the state payroll database showing all state employees.

Your job is to figure out how best to match the two databases together to see how many state employees have outstanding child support debt.

First step is to get to know the two tables …ORDERS and PAYROLL.

Run some basic group by queries to get a sense of what’s there. Scroll through the tables to see what’s included in the data.

When joining tables one of the first things you want to look for is whether or not there are multiple records per person in either or both of the tables.

Let’s check out the ORDERS table first. How many orders are there against each person?

Select name, count(\*)  
from orders  
group by name  
Order by 2 desc

Yep, there are plenty of people with more than one record. But let’s dig a little deeper.

Michael Williams comes up with 12 orders. Let’s look at those records:

Select \*  
from orders  
where name=”WILLIAMS MICHAEL”

What’s wrong? Clearly there is more than one deadbeat dad with the name Michael Williams. So how could we adjust our group by query to be more precise?

We’ve got a couple options, one would be street address, another would be date of birth. Generally date of birth will provide a more precise result since your DOB doesn’t change over time – but your address probably will.

So let’s try that query again, slightly changed:

Select name, dob, count(\*)  
from orders  
group by name, dob  
order by 3 desc

We can also take advantage of another line of SQL that we haven’t used yet. Let’s say we want to return only those records where there is more than 1 order against the person. Remember that the number of orders is reflected in the count(\*) that is generated when we run the query – it’s not something that’s stored in a field in the table. So we have to use something other than WHERE to tell the computer how to limit our results.

In this case, we can use HAVING. It comes after group by, but before the order by line.

Select name, dob, count(\*)  
from orders  
group by name, dob  
Having count(\*)>5  
order by 3 desc

So now we know that there are quite a few people with more than one order against them. That’s a piece of information we’ll need to know later when we get ready to join to the Payroll table.

In the meantime, we also need to know this if we want to figure out who owes the most money among all the deadbeat parents? Since we have more than one order for some people, that means we have to do a group by query and sum the debt to get the correct answer.

Select name, dob, city, sum(debt)  
from orders  
group by name, dob, city  
order by 4 desc

We can also learn another new trick in Access – how to do math in a query (other than sum, average and count). In this case, we’d like to find out how much each person owes per child.

Note a couple things in the query:

1. the debt/num\_kids must be included in the group by line as well as the select line because it is not an “aggregate” function. It’s just doing math on each record. The rule of thumb is that anything in your select line that is not an aggregate function (sum, average, count) must also be in your group by line.
2. We’ve got a WHERE statement. There are some records where it lists zero as the number of kids. This is likely cases where the child is over 18 but the parent still owes the money. Our division won’t work properly – the computer can’t divide by zero – so we need to limit the results of our query. Remember this means that not all records will be returned in our answer.
3. Notice the “as PerKid” after the debt/num\_kids. This is a way of labeling the column in our answer. You can name the column anything. Just make sure it’s all one word, no symbols or spaces.

Select name, dob, city, sum(debt), DEBT/num\_kids as PerKid  
from orders  
where num\_kids>0  
group by name, dob, city, debt/num\_kids  
order by 5 desc

As noted above, joins tend to work best if there aren’t any duplicate records in either of the tables. So in this case, we want just one record per person who has order(s) against them.

So let’s make a new table where we get rid of some of those duplicates and sum the debt from the various orders.

This requires running a Make Table Query. Up until now, we’ve only been using Select queries. If you’re in the Design View or SQL View of your query, you’ll see in the menu that there’s a section of query types. The one called “Select” is most likely highlighted. Next to it you’ll find, Make Table, Update, Append, Delete.

Click on “Make Table” – it will ask you to name your new table. Name it “OrdersTotal” (all one word). Then go back into your SQL to adjust things a bit. You’ll see that it has added “into OrdersTotal” at the end of the select line. That’s how you do Make Table in SQL (without having to push the button)

Adjust your query to match this:

Select name, street, city, state, zip, dob, num\_kids, sum(debt) as TotDebt into OrdersTotal  
from orders  
group by name, street, city, state, zip, dob, num\_kids

Let’s look at that table.. Run a query to see if we have any duplicates here.

Select name, zip, count(\*)  
from orderstotal  
group by name, zip  
Order by 1

We’ve got some problems, huh? What went wrong with our group by that made this new table?

Answer: dirty data, particularly in the street and DOB fields.

If you were doing this for a real story, your next step would be to figure out which record is the correct one and then clean up the data – resulting in one record that combines all the debt for that person. (There are various ways to do this).

We’re going to proceed even though it isn’t quite perfect. Actually having the duplicates will allow you to see what happens.

Look at Payroll table.

Let’s first look for duplicate records here.

SELECT NAME, ZIP, count(\*)  
FROM Payroll  
group by name, zip  
order by 3 desc

We’ve got four people who have duplicate records. Adjust your query a bit and save this as a query called “dups”

SELECT NAME, ZIP, count(\*)  
FROM Payroll  
group by name, zip  
having count(\*)=2  
order by 3 desc

Start a new query using the Dups query and the Payroll table as your two tables. Join them using Name and Zip. Have it display all the records from the Payroll table by simply putting “payroll.\*” in the select line. Your SQL will look like this:

SELECT Payroll.\*  
FROM Payroll INNER JOIN dups ON (Payroll.ZIP = dups.ZIP) AND (Payroll.NAME = dups.NAME)

This returns all of the records in the payroll table for those four men. What does it show?

Dirty data once again. In this case it looks like they have slight variations in each of the addresses. Unfortunately we can’t delete records when the query is displaying a join, so we need to open the Payroll table and go find the bad records and delete them manually. Luckily there aren’t that many. So go into Payroll and delete one of the records for each of these four men (since it’s just variations in the address, it doesn’t matter which one you delete)

Once you’re done deleting records, let’s move on to matching the two tables – Payroll and the OrdersTotal table that we created earlier.

What fields do we have that we could match to the dead-beat parents data? We don’t have date of birth here, do we? Unfortunately, that’s something a lot of states don’t give out for their state employees. But we do have street and zip code.

Let’s try matching just on name:

Select \*  
from payroll, orderstotal  
where orderstotal.name=payroll.name

What’s wrong?

Take a look at Michael Adams. We probably have different people with the same name.

Let’s tighten our join by also matching on the street fields.

Select \*  
from orderstotal, payroll  
where orderstotal.name=payroll.name and orderstotal.street=payroll.street

We end up with 214 matches. (Remember that when you join tables, it only returns records where it finds a match – unless you do a special kind of join that allows you to return all records from one table and only matches from another, but that’s more advanced than we need here).

Let’s do a check to see if that join didn’t miss anybody. Instead of matching on street, match on zip

Select \*  
from orderstotal, payroll  
where orderstotal.name=payroll.name and orderstotal.zip=payroll.zip

Now we get 238 records. At the top of the list should be Charles Abron – he’s a good example of several problems we have here.

First, notice that there are two records and one has a different address. Then look across to the right where the fields from the Payroll table show up. The address for payroll matches one of those address, but it doesn’t the other (one says “Place” and the other says “Pl”). So one of Charles Abron’s records wouldn’t have matched in our earlier query when we joined on name and street.

But why do we have two Charles Abron records in the Orders table? That is one of the ones we didn’t get cleaned up. Notice there are two different dates of birth. We’d have to find out whether that’s just a data entry error or mistake, or if these are two different people who happened to have lived at the same address (highly unlikely, but that’s the kind of caution you need to take).

You’ll see that each Charles Abron record from the OrdersTotal table, matched to the same Charles Abron record from the Payroll table. That’s how joins work when you’ve got duplicate records in one of your tables.

Once you have all these matches, the next thing you’d have to do is go through and look for other problems. Flag any of the matches that appear to be at all suspicious.

Here are a couple things you could do to look for suspicious matches.

FIRST LET’S MAKE A TABLE OF OUR POTENTIAL MATCHES:

Select payroll.name, payroll.street, payroll.zip, mjorderstotal.name, orderstotal.street, mjorderstotal.zip, dob, debt, agency, job\_title, ytd\_gross INTO Matches  
from payroll, orderstotal  
where (mjorderstotal.name=payroll.name) and (mjorderstotal.zip=payroll.zip)

Then run a query on the table looking for records where the two address fields don’t match up:

SELECT \*  
FROM matches  
where orderstotal\_STREET<>payroll\_STREET

Notice how the field names in Matches table are different. It added “orderstotal\_” to the front of all the fields from the OrdersTotal that had the same name as in the Payroll table. And then it did the same thing with “Payroll\_” for all the payroll fields (such as name, street, zip, etc)

This shows us 24 records where the street doesn’t match up. Some of them look like just variations of the same address. But how about George Wilson? There’s one I’d definitely investigate and/or throw out of the analysis.

One last thing to keep in mind … with an analysis like this, especially where you’re accusing people of something, you’re better off being conservative. So if you’re not able to track down whether the George Wilson that is a deadbeat parent is the same George Wilson who works for the state, then you’re better off leaving him out of the list. The story you ultimately write will have to say something like “at least X number of state employees owed child support”

NOW USE OUR NEW TABLE TO RUN SOME ANALYSIS:  
Include the SQL with your answer…

1. Which state employee owes the most child support?
2. What is the total amount of support owed by Illinois state employees?
3. From which agency do employees owe the most?
4. Which type of employee (job title) owes the most in back child support?