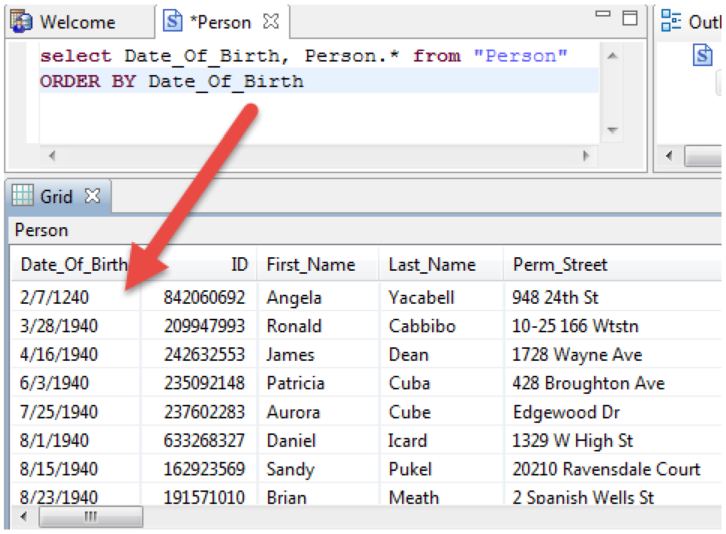
Tech Tip #14: Accessing and Fixing Bad Data from the PCC

Overview: Using queries and the grid to update fields

Once you've seen how to access the SQL interface from the **Pervasive Control Center** and write some simple queries, what can you do with this? Well, to put it mildly – just about whatever you want! The availability of the manual data grid editing capability within the PCC, as well as the UPDATE statement capability in SQL, allows you to use SQL to detect problems in your data and even correct those problems.

Remember the **ORDER BY** clause from the last Tech Tip? We can very rapidly use this to isolate bad data in our database that might be out of bounds or otherwise defective. Here's an example of finding bad data after adding an ORDER BY clause. Notice that by ordering the data by *Date_Of_Birth* field, the record from the year 1240 bubbles right up to the top! We can then use the data grid to fix the date manually from this screen.



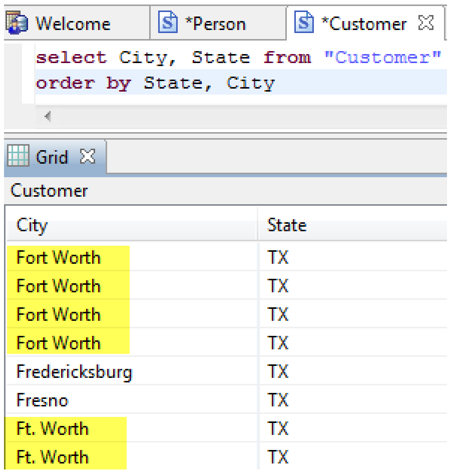
```
select Date_Of_Birth, Person.* from "Person"
ORDER BY Date_Of_Birth
```

Person

Date_Of_Birth	ID	First_Name	Last_Name	Perm_Street
2/7/1240	842060692	Angela	Yacabell	948 24th St
3/28/1940	209947993	Ronald	Cabbibo	10-25 166 Wtstn
4/16/1940	242632553	James	Dean	1728 Wayne Ave
6/3/1940	235092148	Patricia	Cuba	428 Broughton Ave
7/25/1940	237602283	Aurora	Cube	Edgewood Dr
8/1/1940	633268327	Daniel	Icard	1329 W High St
8/15/1940	162923569	Sandy	Pukel	20210 Ravensdale Court
8/23/1940	191571010	Brian	Meath	2 Spanish Wells St

What about the other end of the sort key? There are two ways to handle this. First, you can drag the scroll bar all the way down or press *Ctrl-End* to jump to the end of the data set, then work backwards from the bottom. Or, you can add the key word **DESC** (for DESCending) after the field name and re-run the query to display the data in reverse order. Once you've done that, anything at the other end of the scale should appear in the first row or two.

Another common issue with databases is inconsistency in data entry. This is especially common when multiple people are entering data into the same system. Here's an example where the differences in entering a City name are quite obvious. In this case, finding all the Customers from Fort Worth cannot be done with a simple **WHERE** clause, but rather you have to include TWO clauses, one for each spelling. Ugh.



```
select City, State from "Customer"
order by State, City
```

Customer

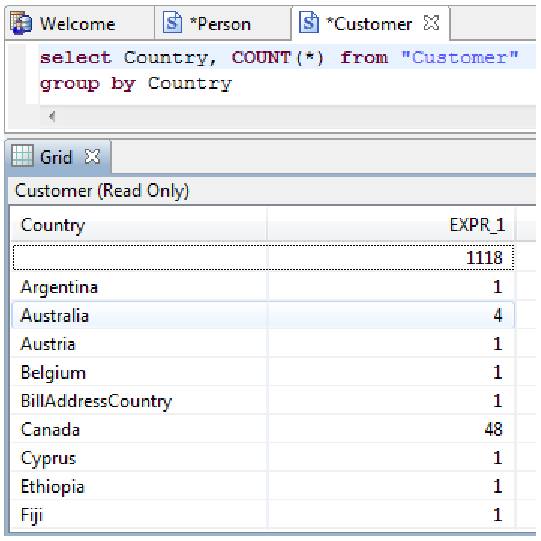
City	State
Fort Worth	TX
Fort Worth	TX
Fort Worth	TX
Fort Worth	TX
Fredericksburg	TX
Fresno	TX
Ft. Worth	TX
Ft. Worth	TX

While you could use the grid to change these records manually so that they all look the same, it is often easier to use the **UPDATE** statement. The **UPDATE** statement is used to **SET** one or more fields in an entire set of records at once. In this case, we want to find all of those records that use "Ft. Worth" and change the *City* field:

```
UPDATE Customer
SET City = 'Fort Worth'
WHERE City = 'Ft. Worth'
```

See how simple this is? We execute ONE query and the SQL engine takes care of finding ALL of the records that match the WHERE clause, and then makes the appropriate change to the *City* field.

We can also use the **GROUP BY** operator to group data into sets, and then grab aggregate data from those sets. Here's an example from the same database.



```
select Country, COUNT(*) from "Customer"
group by Country
```

Customer (Read Only)

Country	EXPR_1
	1118
Argentina	1
Australia	4
Austria	1
Belgium	1
BillAddressCountry	1
Canada	48
Cyprus	1
Ethiopia	1
Fiji	1

What does this query do? First, it uses the **GROUP BY** clause to group all of the records into smaller subsets based on the value of the *Country* field. Then, for each subset, it counts up the number of records in that group. Finally, it displays a list of field values and counts to the screen.

This query shows us a few things. First, many records have a blank or NULL for the *Country* field. Second, we see a smattering of other countries listed, as expected. However, we also see that someone has entered a country called "BillAddressCountry" – which is obviously bad data! Notice, though, that because individual records are not being displayed here, the grid is NOT editable, and there is no way to edit this data directly. In order to find (and thus fix) this data, you must do a second query, like this one:

```
SELECT * FROM Customer WHERE Country = 'BillAddressCountry'
```

Running this query will show you all of the records that match this criteria (which we already know is only one record), and present the ability to edit the data at the same time.

With these basic building blocks of the PCC and the SQL language, you can quickly go through large blocks of data looking for errors, typos, and other intentional bad data entries. By leveraging the grid and UPDATE statements, you can even fix up the data as you find it, and everyone who users your database will be grateful.