Roger's Access Blog

Thoughts, opinions, samples, tips, and tricks about Microsoft Access

http://rogersaccessblog.blogspot.com/2011/03/problem-of-repeated-columns.html

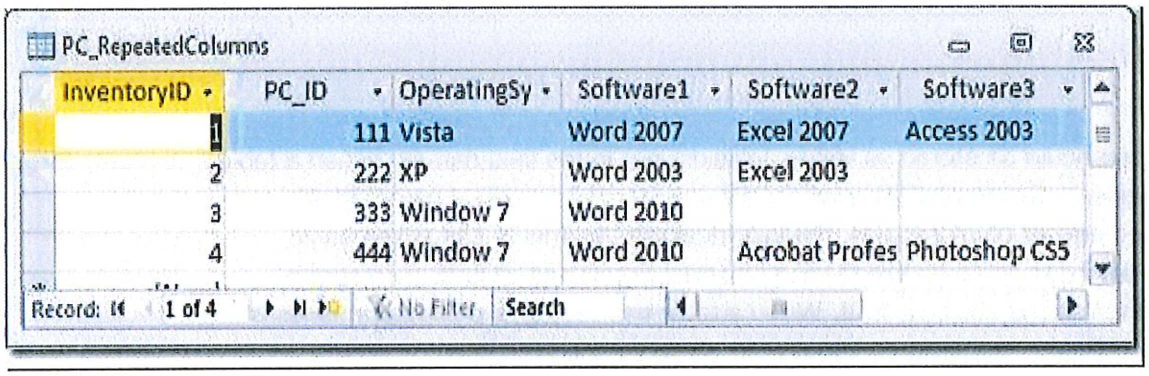
Monday, March 7, 2011

The Problem of Repeated Columns

I've discussed this topic briefly in another post: Why Normalization?, but the issue of Repeated Columns comes up so often that I think it deserves a post of its own.

First of all, what are repeated columns?

A table has repeated columns when the same category of information is stored in multiple columns (or fields). For instance, if I have workstation inventory table, I would want to store all the software installed on a given PC. I could solve this problem with fields like this: Software1, Software2, Software3, and so on. Anytime you have fields which need a suffix to differentiate one field from another, you have repeated columns.

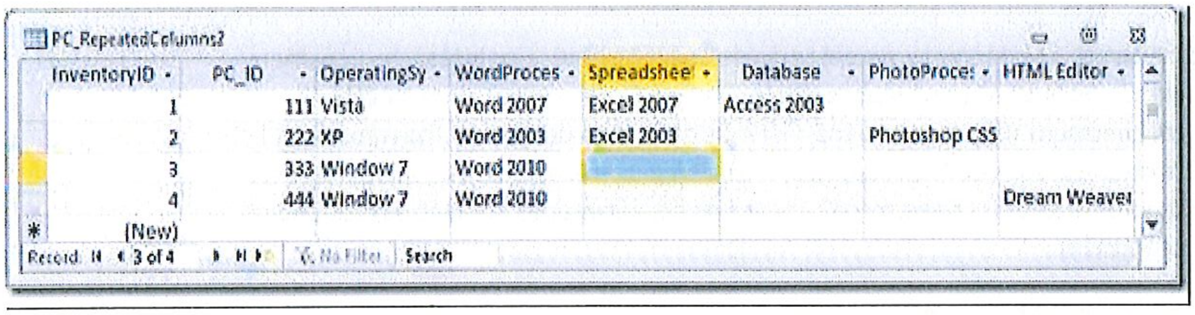

PC_RepeatedColumns

InventoryID	PC_ID	OperatingSy	Software1	Software2	Software3
1	111	Vista	Word 2007	Excel 2007	Access 2003
2	222	XP	Word 2003	Excel 2003	
3	333	Window 7	Word 2010		
4	444	Window 7	Word 2010	Acrobat Profes	Photoshop CS5

Record: 1 of 4 No Filter Search

Figure 1: Inventory table with repeated fields.

But although fields with suffixes are the easiest to spot, there are other cases. Suppose I had these fields: OS, Wordprocessor, Spreadsheet, PhotoManager, and so forth. This is essentially the same as Software1, Software2. I've just substituted the name of software type for a generic name. This reduces the number of fields from the above example, but it still has problems of its own.


PC_RepeatedColumns2

InventoryID	PC_ID	OperatingSy	WordProces	Spreadshee	Database	PhotoProce	HTML Editor
1	111	Vista	Word 2007	Excel 2007	Access 2003		
2	222	XP	Word 2003	Excel 2003		Photoshop CS5	
3	333	Window 7	Word 2010				
4	444	Window 7	Word 2010				Dream Weaver
(New)							

Record: 3 of 4 No Filter Search

Figure 2: Inventory table with repeated fields with different names.

Yet a third type is Multiple Yes/No fields. Suppose I have a Patient table with fields for symptoms (Cough, Fever, Sneeze, etc) which are Yes/No fields. At first glance, these are not repeated fields, but in fact they are. They store the same category of information, that is, each is a symptom.

Querying:

Querying with repeated columns often turns a simple task into an exercise in frustration, requiring massive, nested IIF statements, multiple OR conditions, or both. Aggregate queries like Sum, Count, and Average are particularly difficult.

These problems are magnified by the "How Many, How Few" issue. If I have a maximum possible number, I need to add columns to my query that may never be used. But if I only have the minimum, adding a new column requires me to modify every query. Either way, it's a lot of unnecessary work.

Aggregating Across Repeated Columns: Averaging

'n the first post in this series (The Problem of Repeated Columns), I defined repeated columns and talked about the data integrity problems associated with them. If you haven't read that yet, it would be worthwhile to read first. Similarly, if you are not familiar with the concept of Normalization, you should read my blog series What is Normalization?

So far in this series, I've discussed the problem with querying textual or Boolean (Yes/No) data from repeated columns. But numeric data offers new challenges because we often want to do math on them. The most common kind of math is aggregation, that is, summing, counting, and averaging numeric values. This time, I'll talk about counting data in repeated columns.

Others in this series:

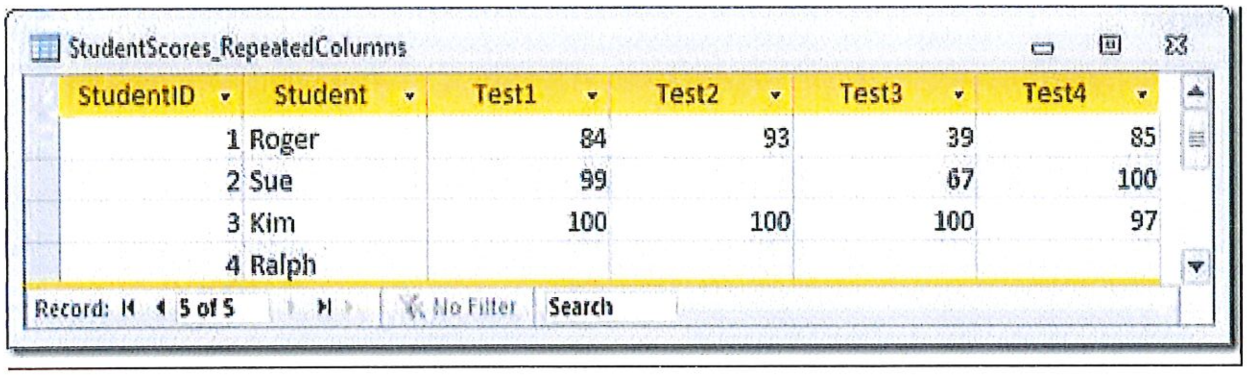
- Multiple ORs
- Multiple Unions
- Multiple Joins
- Multiple Ifs
- Impossible Joins
- Summing
- Counting

Averaging Across Columns

The most common type of data aggregation, perhaps, is in calculating an Average where you divide the sum of the values by the count of the values.

In Excel, there is an AVERAGE function which will average the cells that have a value. The Excel function will work for any range of cells, across or down. In Access, however, the Average() function only works down columns, not across columns. So, averaging values in repeated rows is easy (see below), but just like summing and counting, averaging across repeated columns is more challenging.

For instance, suppose I had a table of student test scores:

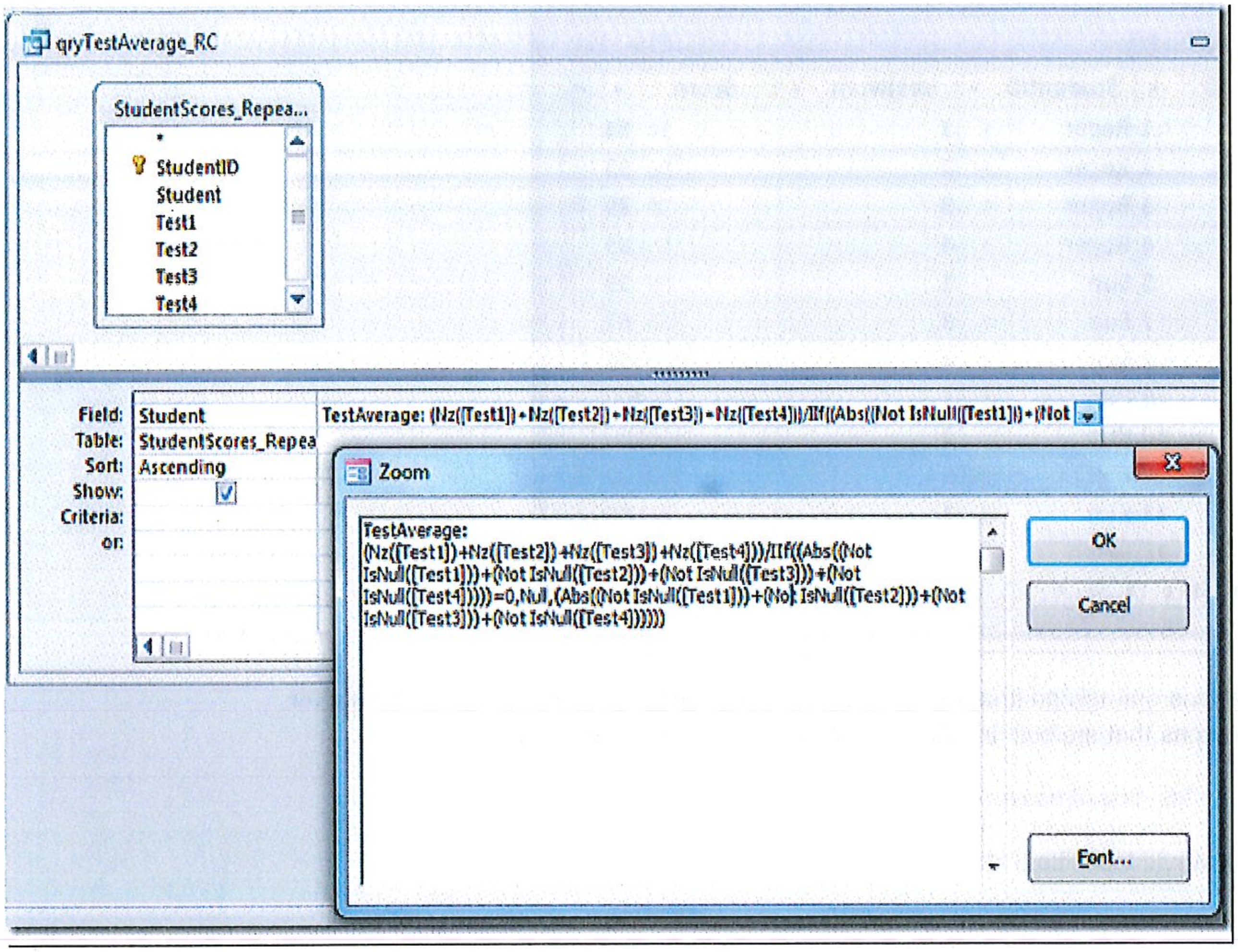

StudentScores_RepeatedColumns

StudentID	Student	Test1	Test2	Test3	Test4
1	Roger	84	93	39	85
2	Sue	99		67	100
3	Kim	100	100	100	97
4	Ralph				

Record: 5 of 5 No Filter Search

If I wanted to average the test values for each student, I need to create an expression that sums the values for the numerator and counts the values for the denominator. For details on how to sum the values, see Aggregating Across Repeated Columns: Summing. To count the values, seeAggregating Across Repeated Columns: Counting

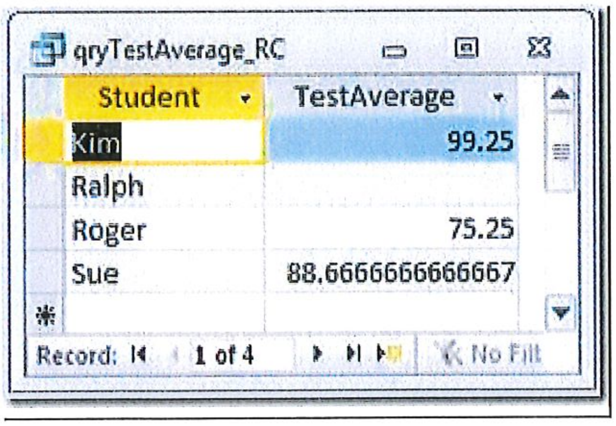
To do this, I need 4 stages.



Field:	Student	TestAverage: (Nz([Test1])+Nz([Test2])+Nz([Test3])+Nz([Test4]))/IIf((Abs((Not IsNull([Test1]))+(Not
Table:	StudentScores_Repea	
Sort:	Ascending	
Show:	✓	
Criteria:		
or:		

```
TestAverage:
(Nz([Test1])+Nz([Test2])+Nz([Test3])+Nz([Test4]))/IIf((Abs((Not
IsNull([Test1]))+(Not IsNull([Test2]))+(Not IsNull([Test3]))+(Not
IsNull([Test4]))))=0,Null,(Abs((Not IsNull([Test1]))+(Not IsNull([Test2]))+(Not
IsNull([Test3]))+(Not IsNull([Test4])))))
```

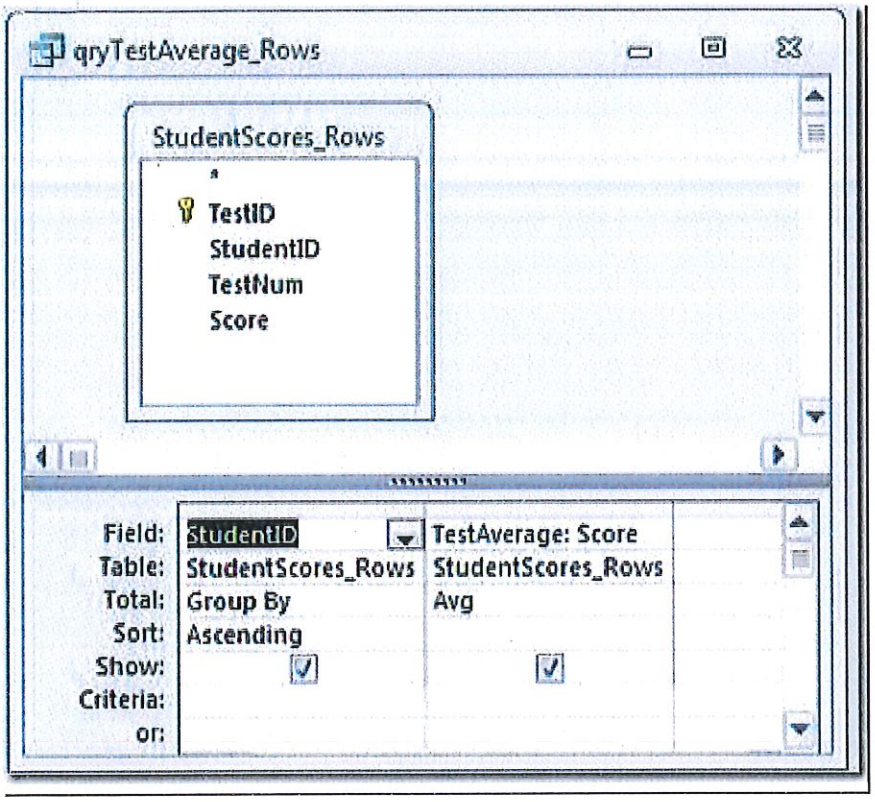
The result would look like this:



Student	TestAverage
Kim	99.25
Ralph	
Roger	75.25
Sue	88.6666666666667

Averaging Down Rows

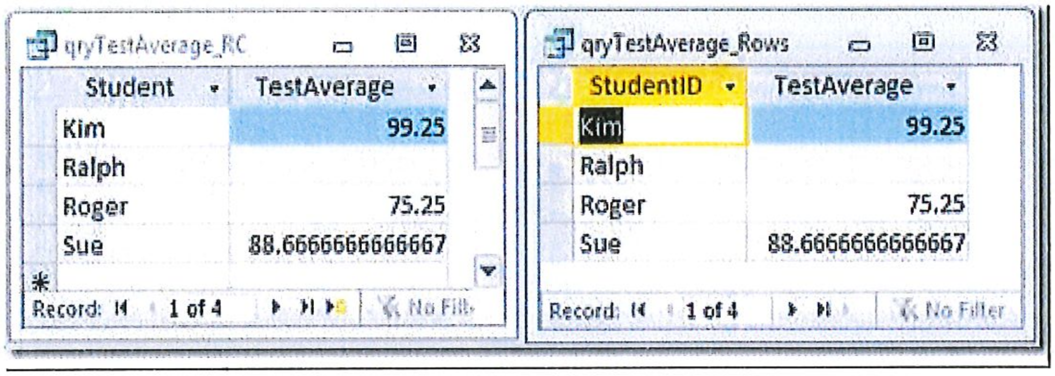
By contrast, suppose I normalize the table to remove the repeated columns. The table should look something like this:


qryTestAverage_Rows

StudentScores_Rows
- *
- TestID
- StudentID
- TestNum
- Score

Field:	StudentID	TestAverage: Score
Table:	StudentScores_Rows	StudentScores_Rows
Total:	Group By	Avg
Sort:	Ascending	
Show:	✓	✓
Criteria:		
or:		

Once again, the results of the queries are identical:


qryTestAverage_RC

Student	TestAverage
Kim	99.25
Ralph	
Roger	75.25
Sue	88.6666666666667

Record: 1 of 4

qryTestAverage_Rows

StudentID	TestAverage
Kim	99.25
Ralph	
Roger	75.25
Sue	88.6666666666667

Record: 1 of 4

Now, with only 4 test scores, the expression to average repeated columns is manageable. But what if there were 20 or 50? The expression quickly becomes long and cumbersome. But with the normalized structure, the query doesn't change no matter how many test values there are.