



# Advanced Excel Reporting (Data Mining)

**PERENNIAL SOFTWARE**  
Makers of SedonaOffice and AlarmBillr

---

Table of Contents

<b>Overview</b> .....	<b>4</b>
Each Company is a Database.....	4
Databases Contain Tables, Views, and Stored Procedures.....	4
Tables Contain Fields.....	5
<b>Linking Tables</b> .....	<b>5</b>
Link Types.....	5
<b>Customer Structure</b> .....	<b>6</b>
<b>Invoice Structure</b> .....	<b>7</b>
<b>Cash Structure</b> .....	<b>8</b>
<b>Vendor Structure</b> .....	<b>9</b>
<b>Vendor Bills Structure</b> .....	<b>10</b>
<b>Check Structure</b> .....	<b>11</b>
<b>Job Structure</b> .....	<b>12</b>
<b>Service Ticket Structure</b> .....	<b>13</b>
<b>Inventory structure</b> .....	<b>14</b>
<b>General Ledger Structure</b> .....	<b>15</b>
<b>Open Database Connectivity (ODBC)</b> .....	<b>16</b>
Creating an ODBC Connection with Excel.....	16
<b>Building a Query Using Excel and MS Query</b> .....	<b>18</b>
<b>Using Microsoft Access to Review Your Data</b> .....	<b>22</b>
Connecting Access via ODBC.....	22
Writing a query with Access.....	27
Creating a Report with Access.....	32
Creating a Grouped and Sub Totaled Report.....	36
<b>Using SedonaOffice Data in a Microsoft Mail Merge</b> .....	<b>44</b>

Creating the List of Customers .....	44
Creating the Letter .....	48
Merging and Creating the Letters .....	53
<b>Basic SQL Language .....</b>	<b>54</b>
Select Keyword .....	54
From Keyword .....	55
Join Keyword .....	55
Where Keyword .....	59
Order By Keyword .....	66
<b>Advanced SQL Language .....</b>	<b>67</b>
Sub Queries .....	67
Union Keyword .....	68
Aggregates and Group By .....	69
Variables .....	74
If, While and Case .....	76
Virtual tables .....	79
<b>Even More Advanced SQL .....</b>	<b>83</b>
Views .....	83
Select Into .....	86
Sample queries .....	88

## Overview

This guide is intended to teach you how to access data from a SedonaOffice database. Data extracted from a database can be used for many different purposes both internally and externally for an organization. While this guide will review a variety of different techniques, it is impractical to detail each and every method that can be used to extract data.

### Each Company is a Database

Each SedonaOffice company is its own unique database within the SQL server. In addition to the various company databases, there is an additional database that helps to control access to the company databases. This access control database is named SedonaMaster.

SedonaMaster contains a list of company names and the database associated with each name. All other data about a company is contained within the company database. All of the setup information, names, addresses, part numbers, service tickets, etc., for a company, are stored within the same database. The structure of the database will remain the same for all companies. The differences in how companies operate are contained in the setup tables. If a feature of SedonaOffice is not used, the data structure will still exist but may be empty of data.

### Databases Contain Tables, Views, and Stored Procedures

The main structures in a database are tables, views and stored procedures. Tables contain the raw data, the actual names, addresses, etc.

Views are premade queries that will return sets of data automatically. If there is a set of data you are going to regularly extract, you may want to think about making a view. SedonaOffice uses several views in supplying data to the client. Do NOT alter these or your system may cease to operate correctly<sup>1</sup>.

Stored procedures are routines containing SQL code. They can be created to act as a view but are usually used to manipulate the data. Stored procedures can also take parameters; values that modify how the stored procedure will operate. Most of the business logic in SedonaOffice is handled by stored procedures. They are encrypted and locked for safety and security. Do NOT delete or replace a stored procedure or your system will cease to operate correctly<sup>2</sup>.

---

<sup>1</sup> Unless directed to by a SedonaOffice support person.

<sup>2</sup> Unless directed to by a SedonaOffice support person.

## Tables Contain Fields

Fields contain your actual data. There are different types:

- Text including varchar, nvarchar and char. The length of the field in characters (including spaces and punctuation) is defined when the field is created.
- Numeric including integer, double and money. What range, and if a fractional decimal amount is supported, is defined when the field is created.
- Datetime. Microsoft SQL server does not contain a field type for date and a separate field for time. All date and time related fields are Datetime fields.

## Linking Tables

Tables are linked via fields that end in Id. In each table, the first field is the Identity field for that table. Identity fields are not editable nor should you try. Identity fields are unique. This number is automatically assigned by the SQL server. Once assigned, a number is never reused, not even if it was previously deleted. This Identity field is the “Address” of the record. Other tables that point to this table will have an Id that matches the “Address” of the record. IE Customer\_Id in the AR\_Customer\_Bill record will point to the Customer\_Id field in the AR\_Customer table. The Customer\_Id in the AR\_Customer table is the Identity or “Address” of that record. Notice that the name of an Id is the same as the table name in our example. This is true of all ID’s with very few exceptions.

## Link Types

Table links are defined by the relationship of records in one table to the records in another table. There are three basic link models.

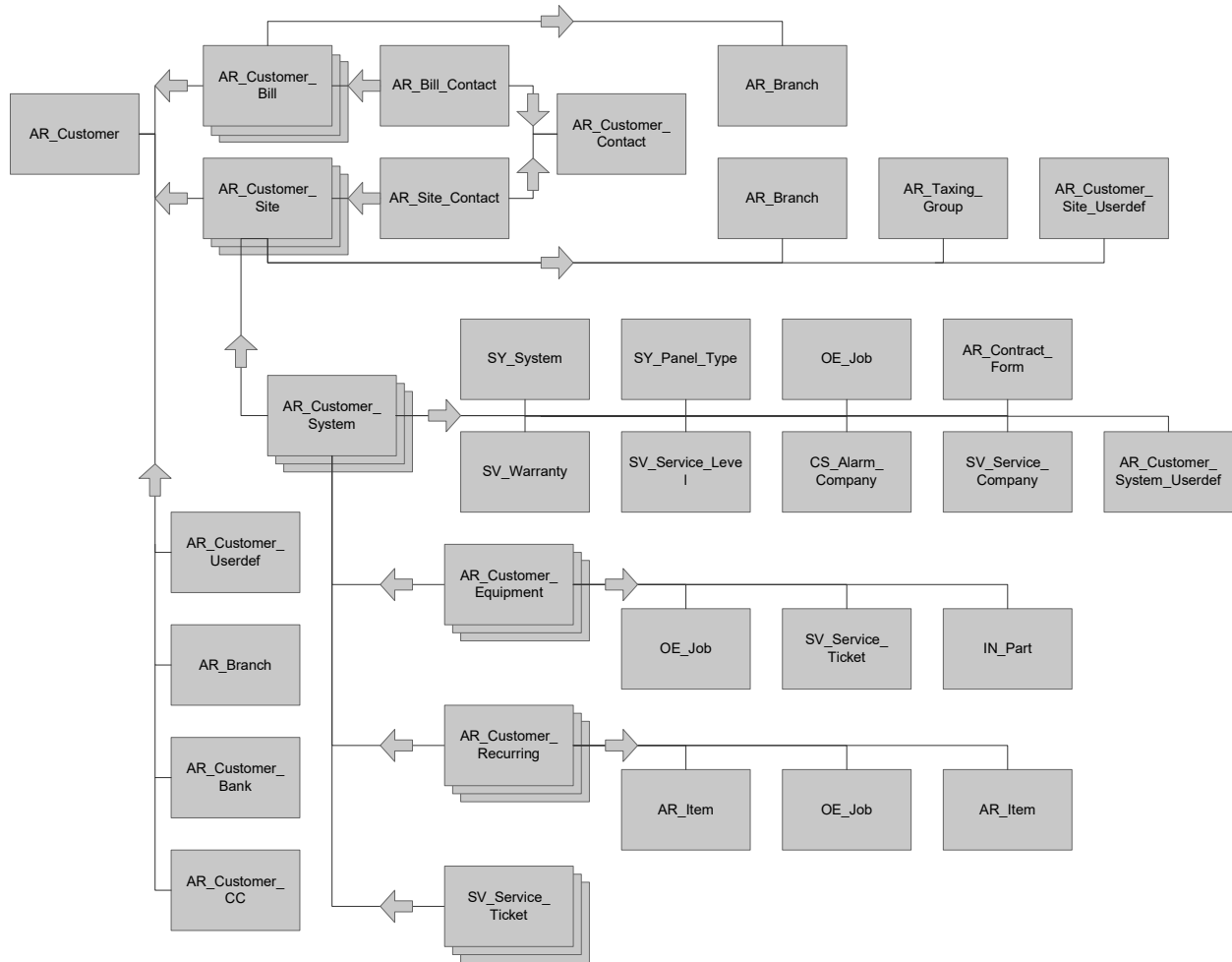
One to one: Each record in one table matches to exactly one record in the other table. IE AR\_Customer and AR\_Customer\_Userdef

One to many: Each record in one table matches to many records in the other table. IE AR\_Invoice and AR\_Invoice\_Item

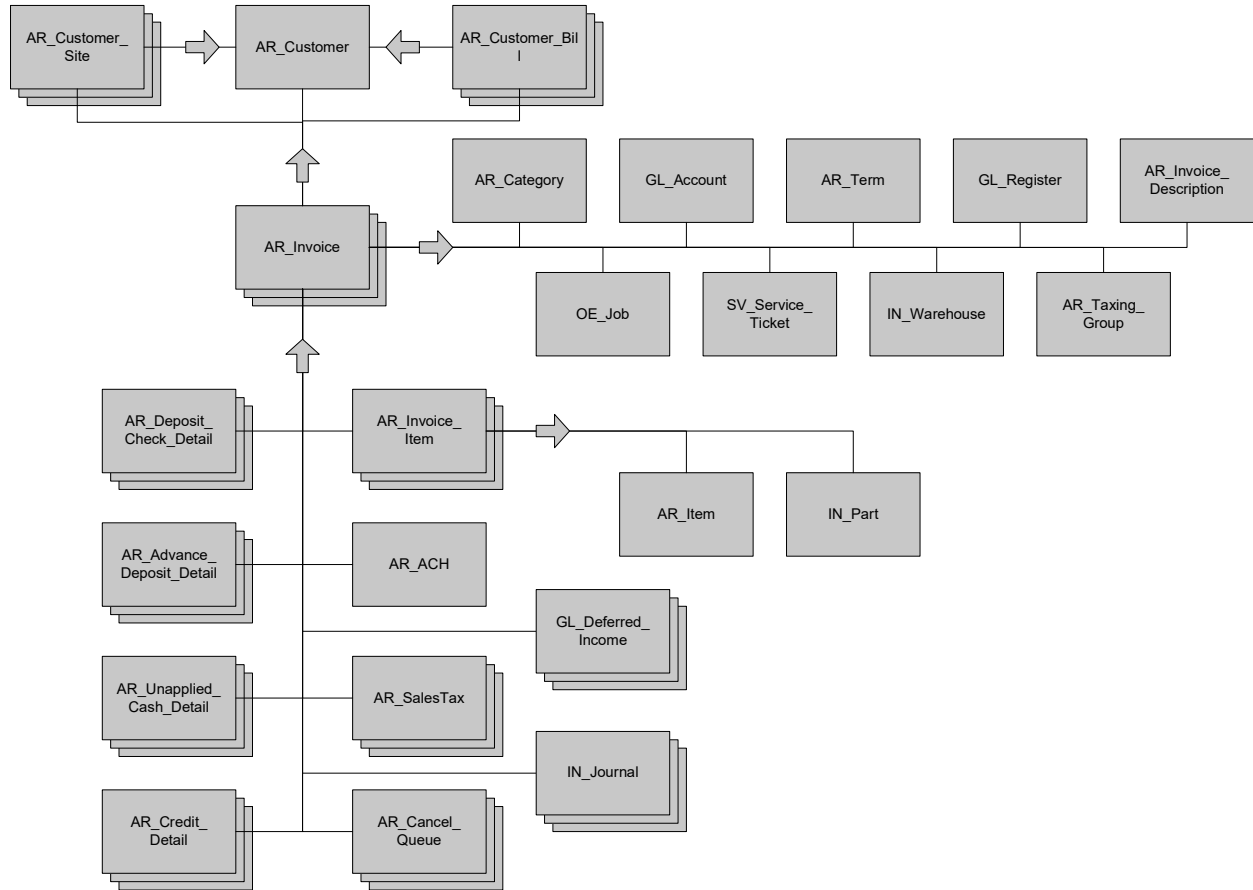
Many to one: Many records in one table match to one record in the other table. IE AR\_Customer and AR\_Branch

The following diagrams are not meant to be completely accurate or to be used as a definition of the database structure. They are simplified diagrams to give an outline of the relationship of the various tables that combine to make up a data structure.

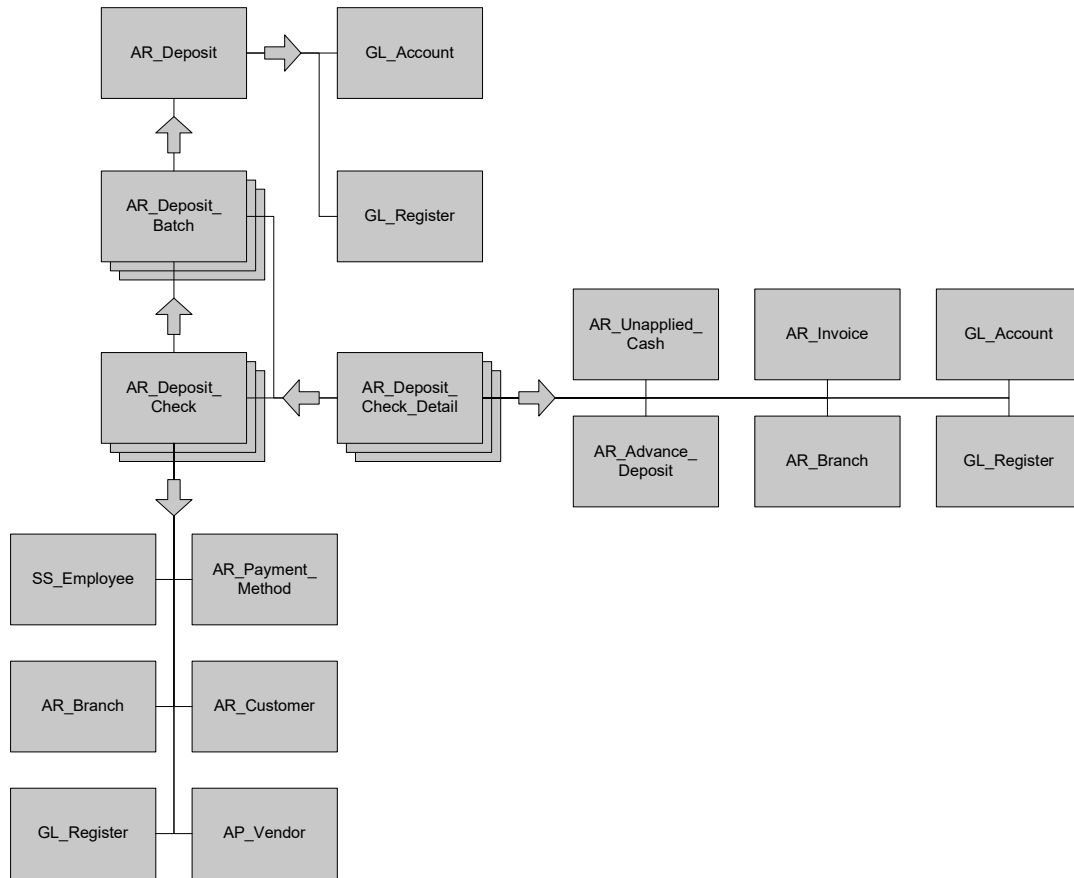
## Customer Structure



## Invoice Structure

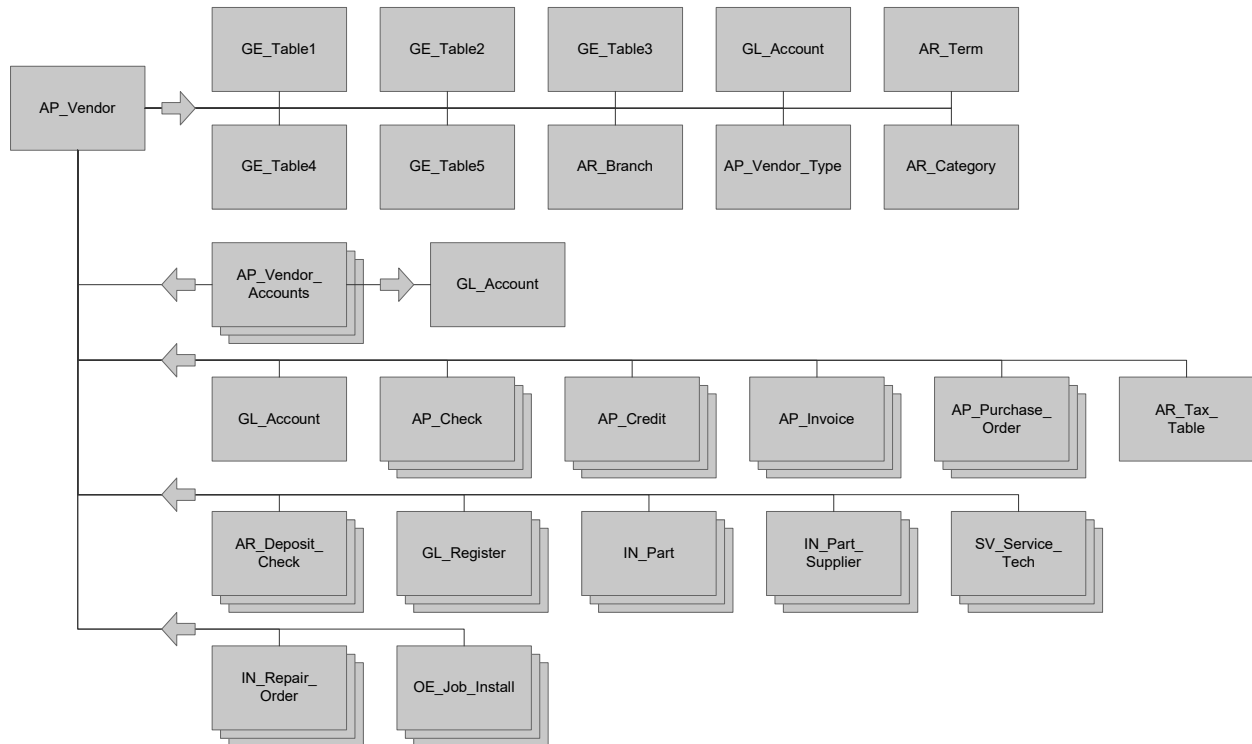


## Cash Structure

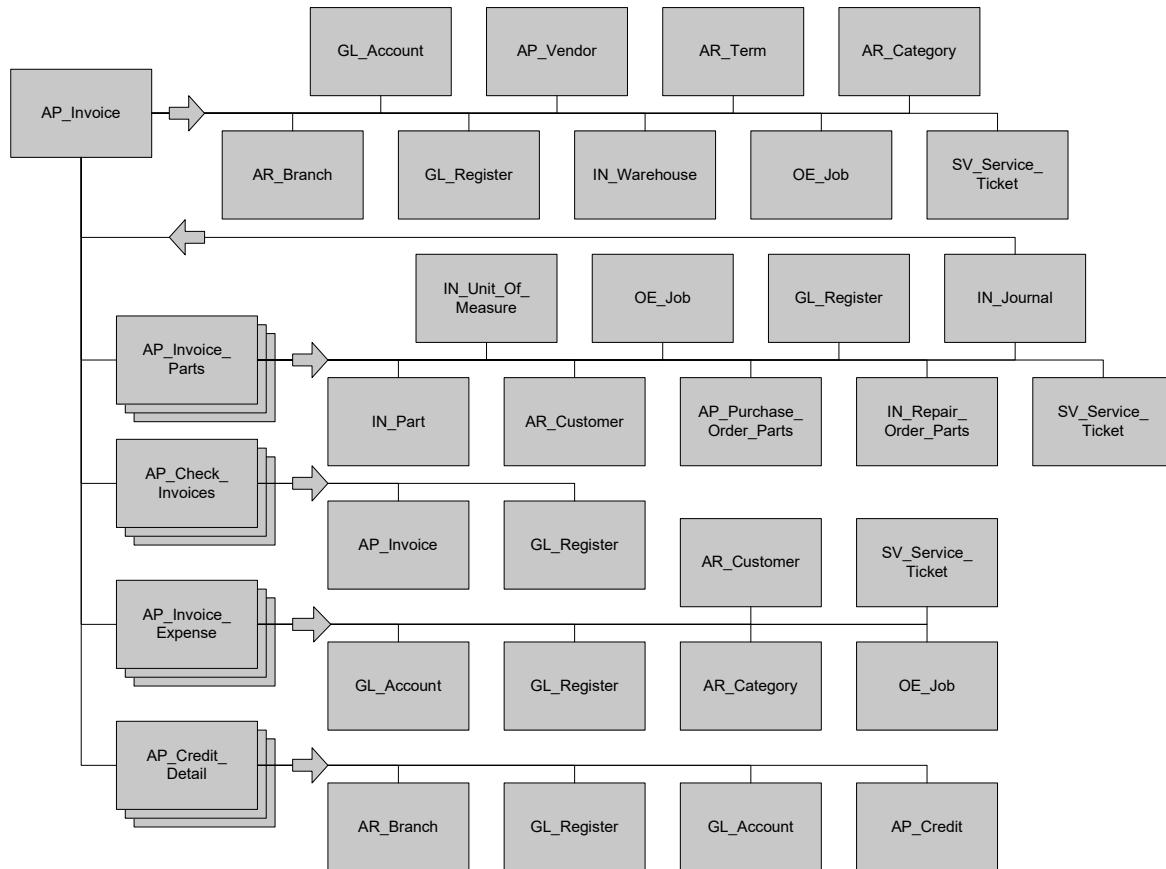




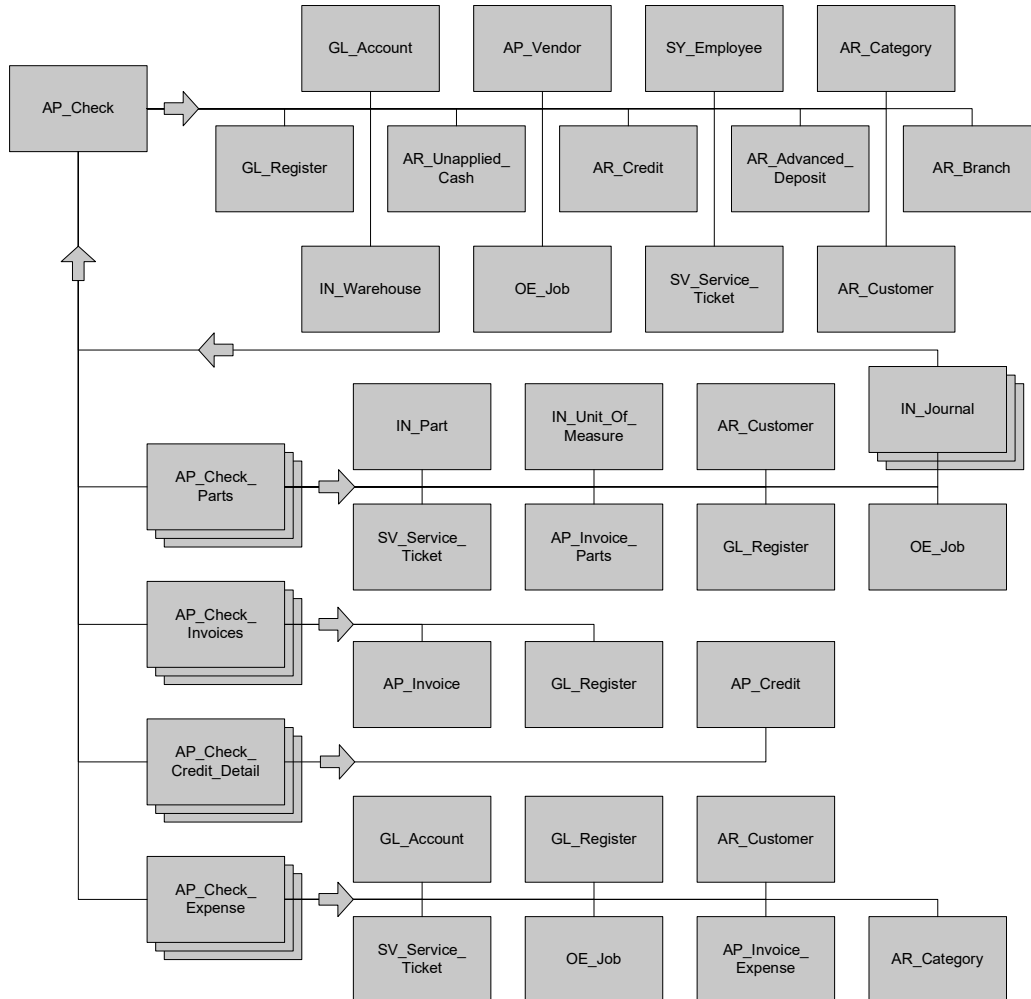
## Vendor Structure



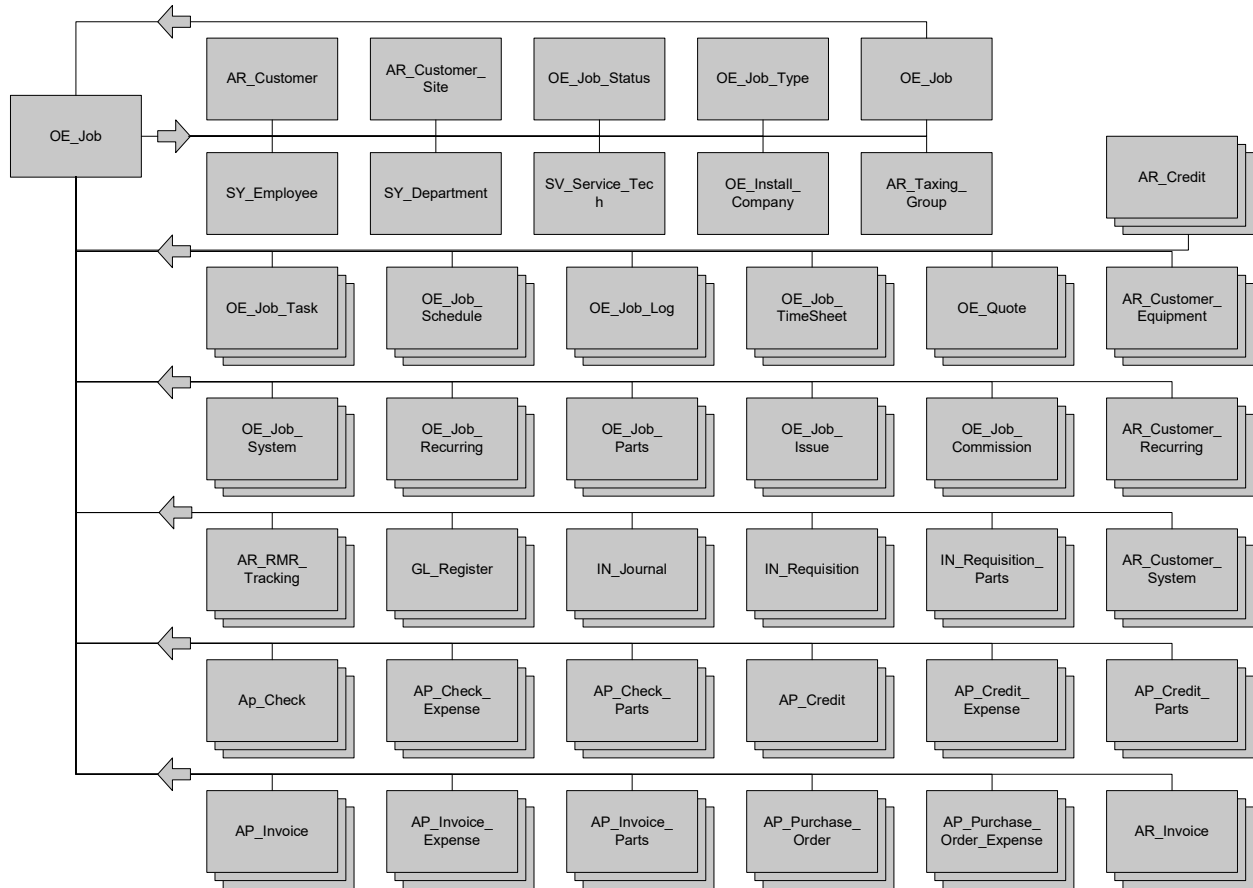
## Vendor Bills Structure



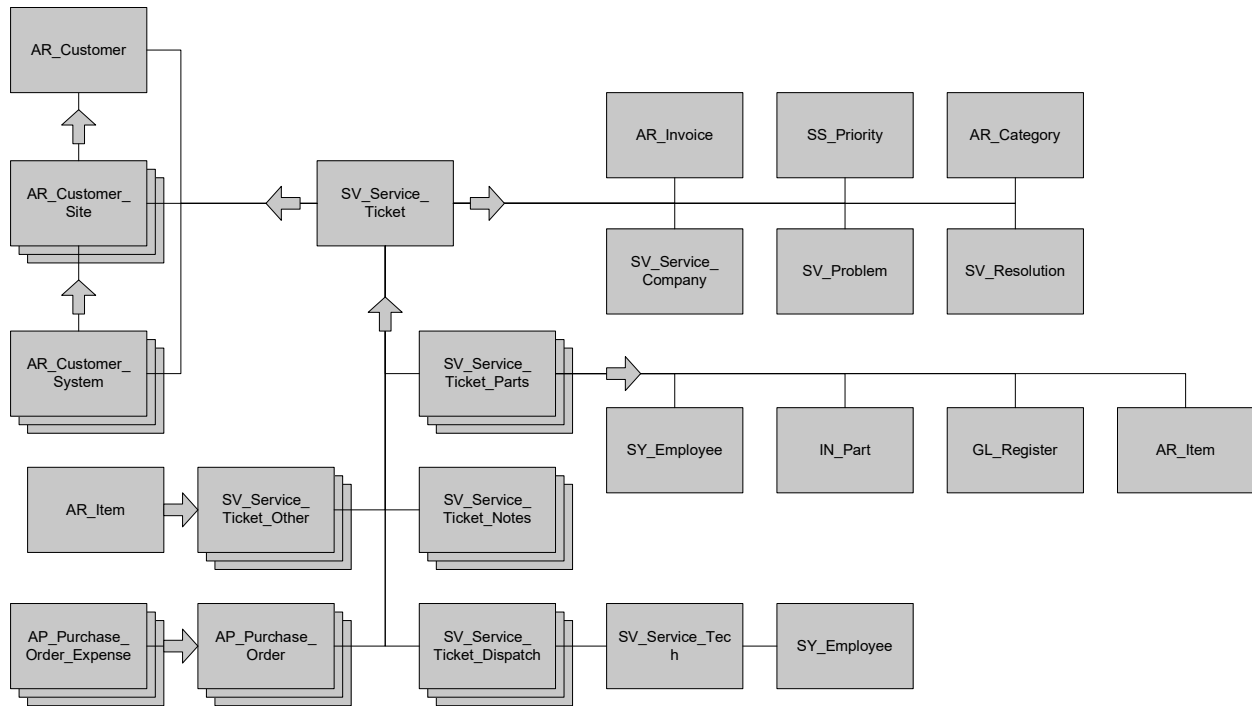
## Check Structure



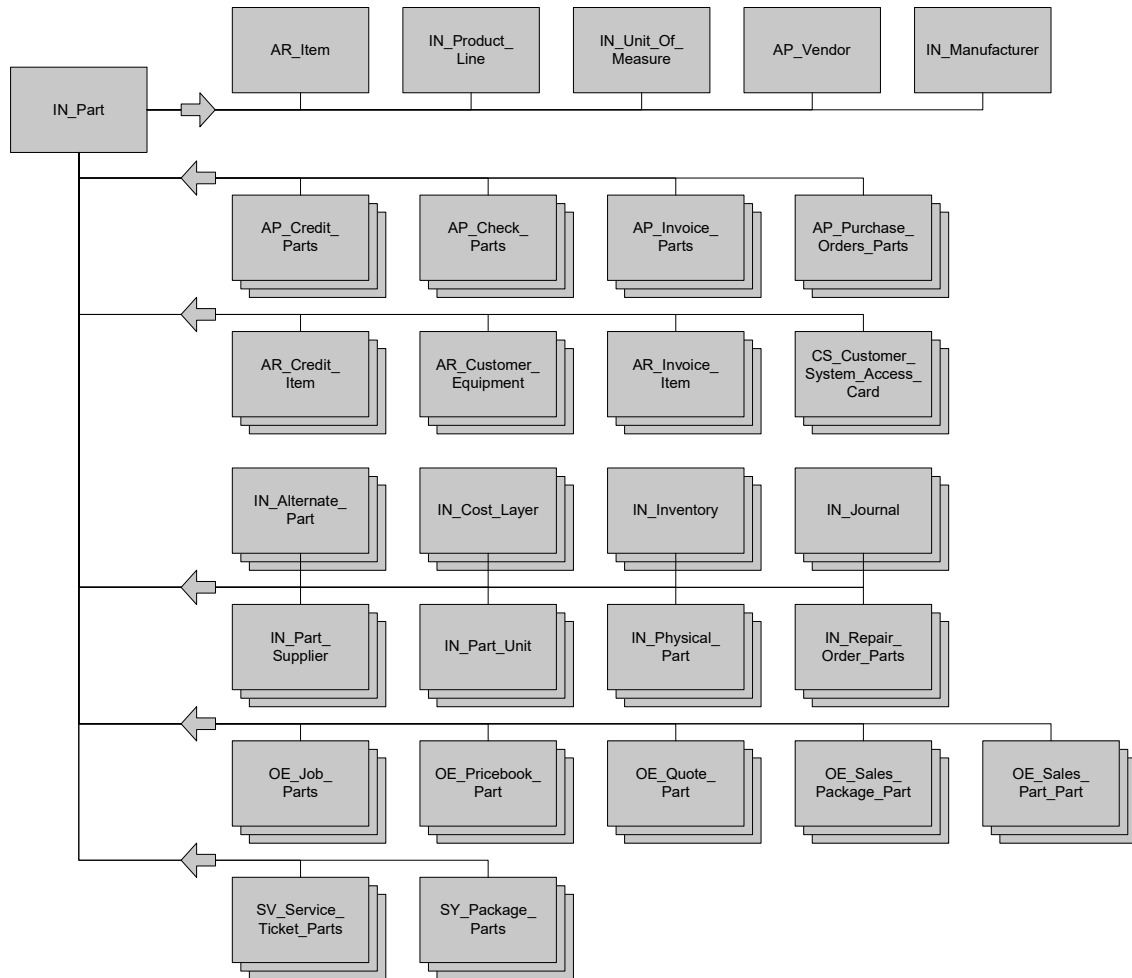
## Job Structure



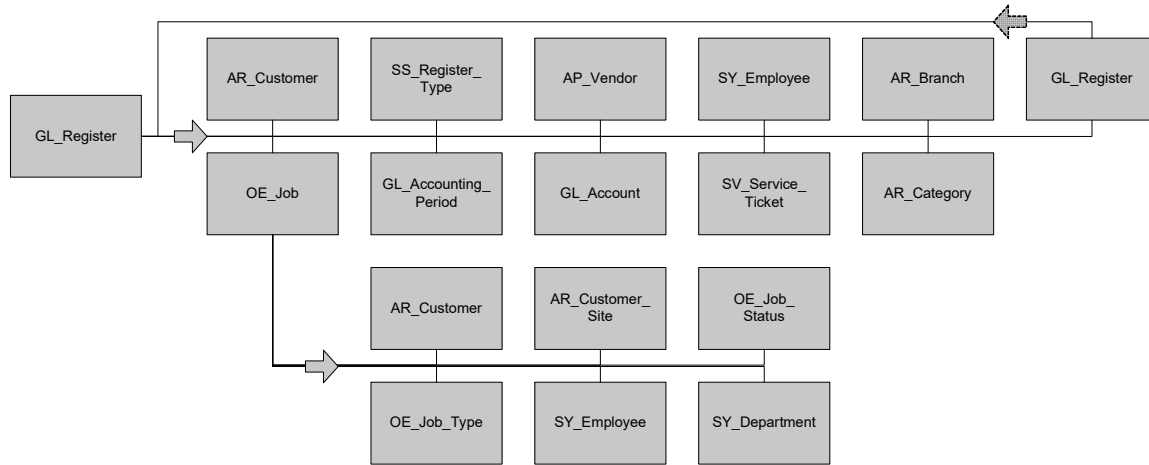
## Service Ticket Structure



## Inventory structure



## General Ledger Structure

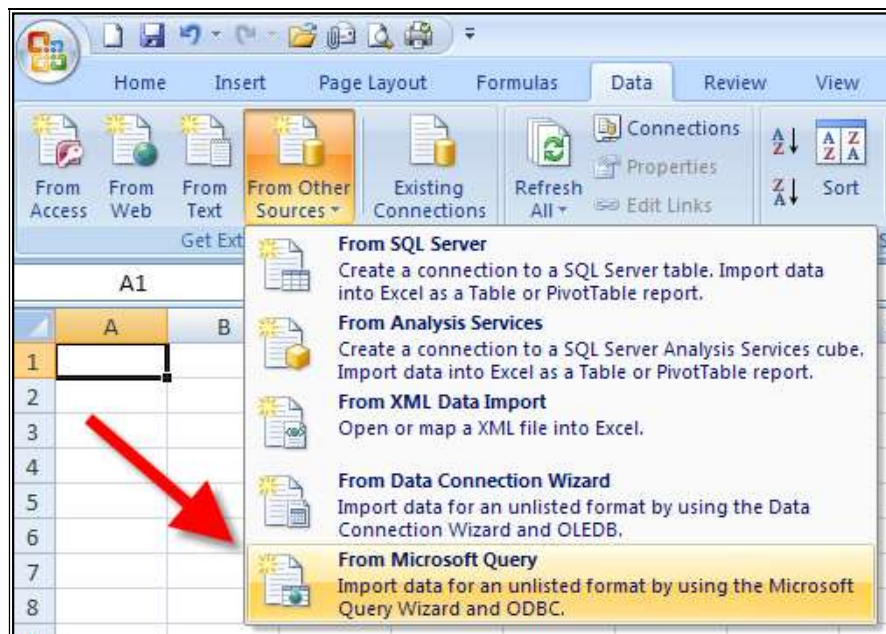


## Open Database Connectivity (ODBC)

Open Database Connectivity is the methodology created by Microsoft for different applications to talk to different kinds of databases. With ODBC you can connect Excel to Microsoft SQL server or MS Word to Excel for example. The first step in connecting any application to your Microsoft SQL database is to create an ODBC connection. There is a utility for setting up ODBC connections. It is located in the Control Panel under ODBC. Many applications contain an implementation of the ODBC Data Source Administrator. In our example we are going to use Excel to create an ODBC connection.

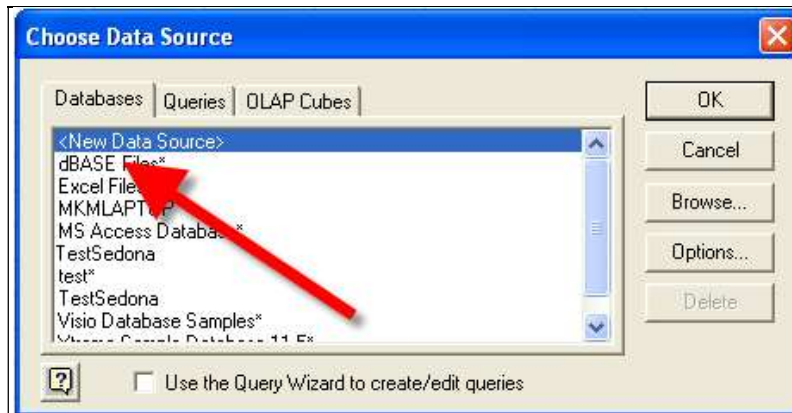
### Creating an ODBC Connection with Excel

Let's now review how to import Data into Microsoft Excel. In this example we are going to use the feature in Excel to Query an External Data Source using Microsoft Query. This feature is available in most recent versions of Excel but may need to be installed as Excel does not install it by default in the standard install.





If you have not already done so, you will need to create a Data Source connection to your SedonaOffice database.



To create the new Data Source:

- 1) Name the data source appropriately (Here we are using “SedonaOffice GL Data” but the same connection can be used for all of your queries so you might want a more general name).
- 2) Select ‘SQL Server’ as the driver to connect to the database.
- 3) Press the Connect button.
  - a. On the SQL Server Login Screen select the name of the SQL Server for SedonaOffice.
  - b. Use “SedonaReports” as the Login ID, no password is needed.
  - c. Select the Options tab and select the name of your production SedonaOffice database.
- 4) Press OK.

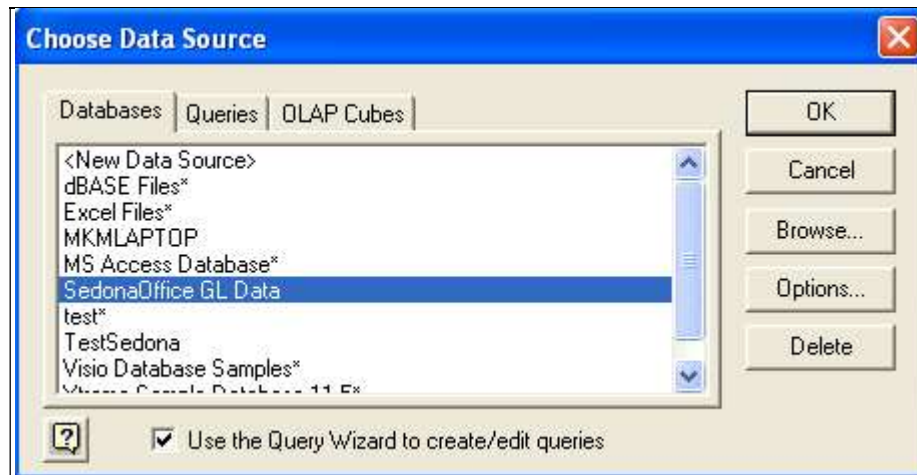




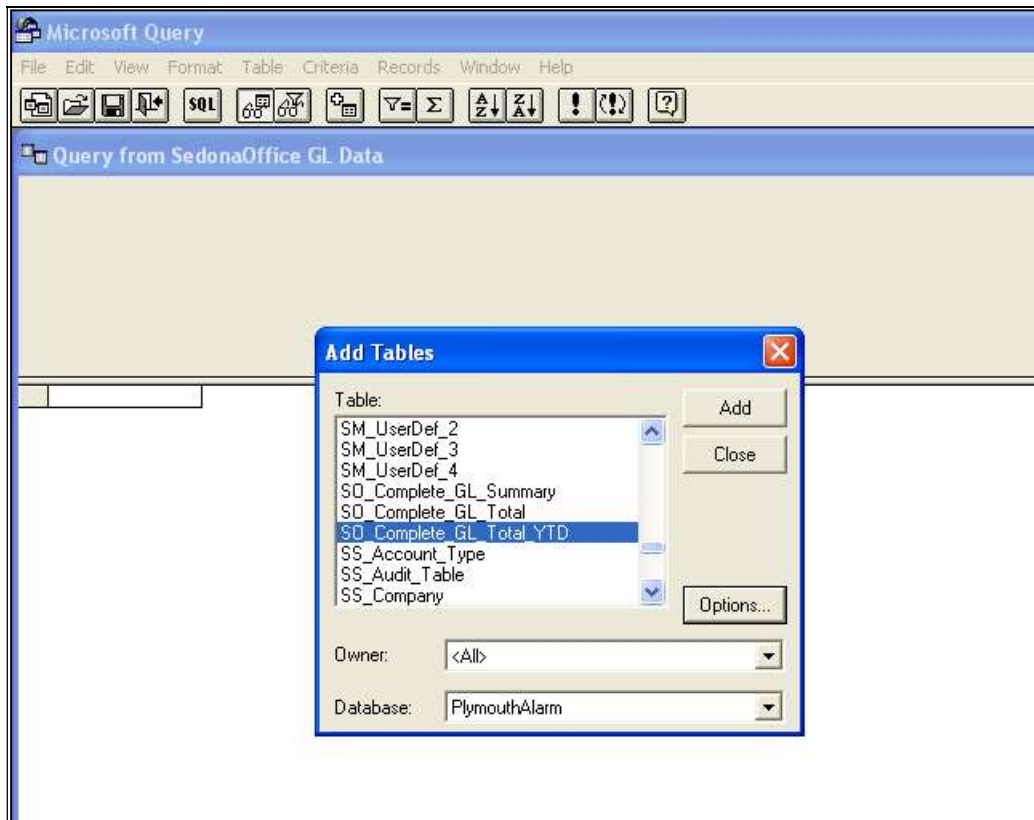
You now have an ODBC connection to your database.

## Building a Query Using Excel and MS Query

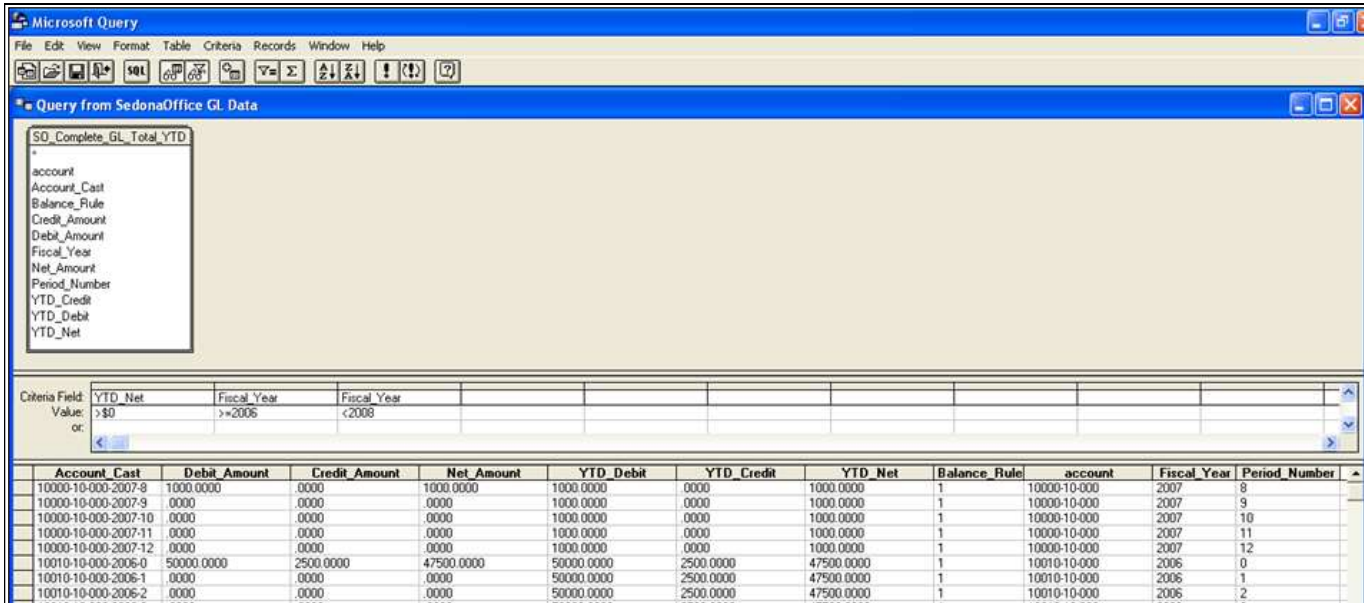
Select the data source you have previously created to create the Query. Uncheck the “Use the Query Wizard...” this will take you directly to Microsoft Query to create the Query.



To begin, you need to select the Table file to use in the Query. Select the “SO\_Complete\_GL\_Total\_YTD” table. Then click Close.



The next step is to select the data fields and criteria for the data to be returned. Select all the data elements in the Table. While it doesn't really matter what order to display the data fields, using the order as shown below will be more logical when viewed with Excel.



Since this table can contain thousands (or hundreds of thousands) of records, it is best to use some criteria to limit the data that is returned.

**Criteria Selections:**

- 1) YTD\_Net <> \$0 – By selecting this option only data with values will be returned.
- 2) Fiscal Year >= 2006 – In this case only years 2006 and 2007 are needed so limit the data to only these fiscal years.
- 3) Fiscal Year < 2008 – In this case since 2008 has been created we can remove these entries since we are only reporting on 2006 and 2007.
- 4) Net\_Amount <> \$0 – This is included as an 'OR' selection. This is necessary to return the Retained Earnings account (more on this later).

Criteria Field:	YTD_Net	Fiscal Year	Net_Amount	
Value:	<>\$0	>=2006		
or:	\$0	>=2006	<>\$0	

Now that we have completed the Query, click the Return Data icon, and the GL Data will be returned to Excel.



	A	B	C	D	E	F	G	H	I	J	K
1	Account_Cast	Debit_Amount	Credit_Amount	Net_Amount	YTD_Debit	YTD_Credit	YTD_Net	Balance_Rule	account	Fiscal_Year	Period_Number
29	10010-10-000-2007-9	0	19744	-19744	50100	22364	27736	1	10010-10-000	2007	9
30	10010-10-000-2007-10	0	0	0	50100	22364	27736	1	10010-10-000	2007	10
31	10010-10-000-2007-11	0	0	0	50100	22364	27736	1	10010-10-000	2007	11
32	10010-10-000-2007-12	750000	0	750000	800100	22364	777736	1	10010-10-000	2007	12
33	10010-20-000-2007-9	750000	980	749020	752425	5325	747100	1	10010-20-000	2007	9
34	10010-20-000-2007-10	0	0	0	752425	5325	747100	1	10010-20-000	2007	10

## Using Microsoft Access to Review Your Data

Why use Access instead of Excel? Excel has a row limit. The maximum number of rows you can have in a spreadsheet varies with what version you use, from as little as 32,767 rows for older versions to 1,048,576 rows in Excel 2007. This may seem like a lot of rows and for most queries it will be sufficient. But, queries involving the GL\_Register for a company that has several years of history can easily exceed these limitations.

Excel treats all fields containing only numeric characters (0-9) as numbers unless prefaced with a ' character. By treating things like postal codes as numbers postal codes starting with a 0 are truncated. Thus a postal code of 01234 becomes 1234.

Finally, Access has a built in report generator. With Access you can make complex reports with groups, subtotals, totals, etc.

**\*\*\* Caution \*\*\*** *ONLY use SedonaReports for an ODBC connection to Access. Otherwise changes you make in Access can change your SQL Server data and corrupt your database.*

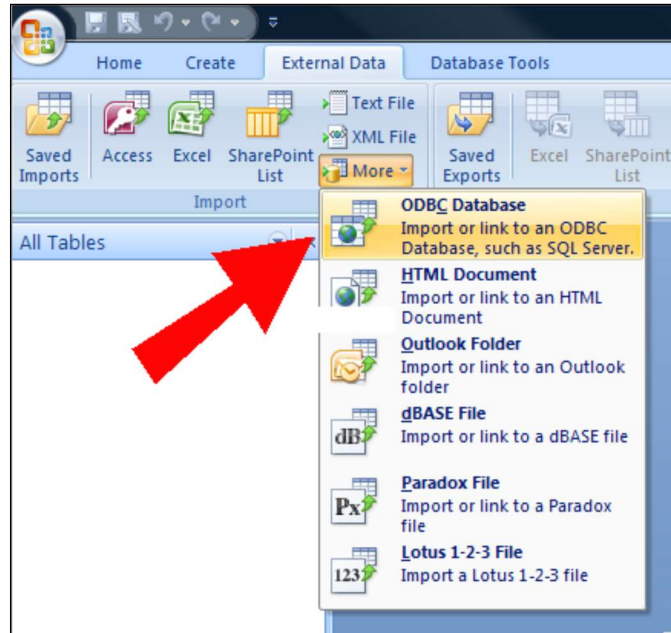
### Connecting Access via ODBC

When using an ODBC connection with Access you have two options on how to connect the data; Import or Link.

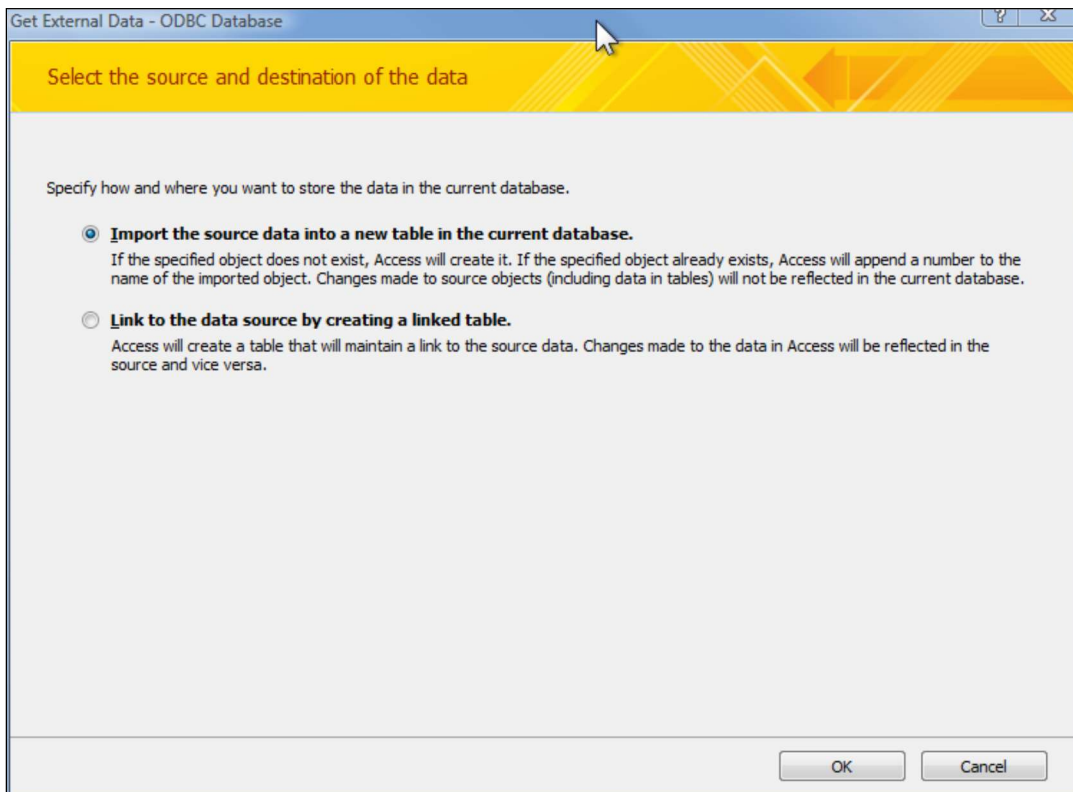
When you Import data into Access, you create a copy of the data stored within the Access database. This allows you to review the data when not connected to the database. Like Excel, you have to periodically refresh the data to keep it up to date.

When you Link data to Access, the data remains in the SQL Server but Access can use it in queries and reports. This method constantly refreshes as the data in the SQL server changes but it will not function if it is disconnected from the SQL Server.

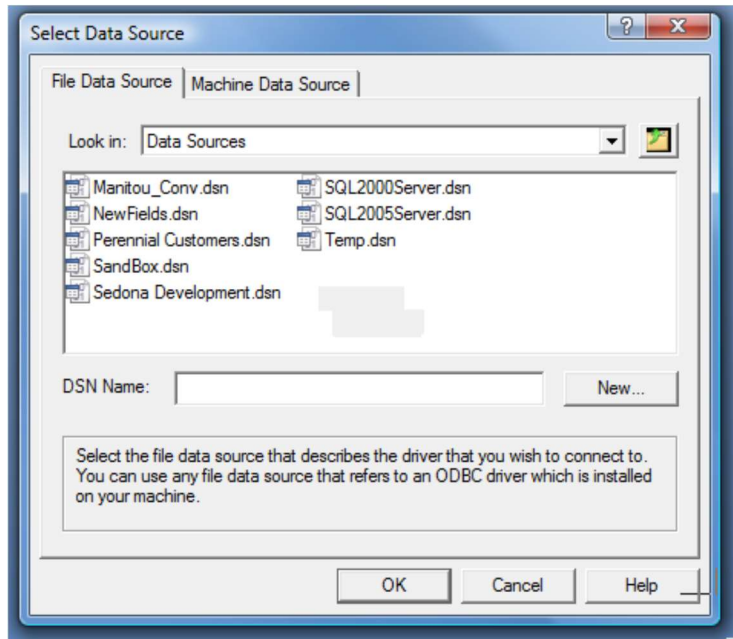
Choose the External Data tab. Then choose More. Finally choose ODBC Database.



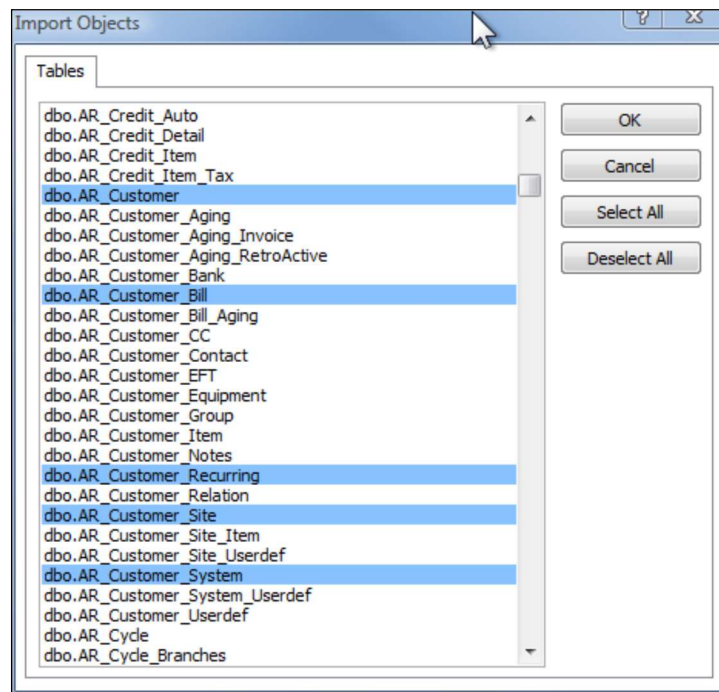
Choose Import or Link and then click OK.



Choose your Data Source.

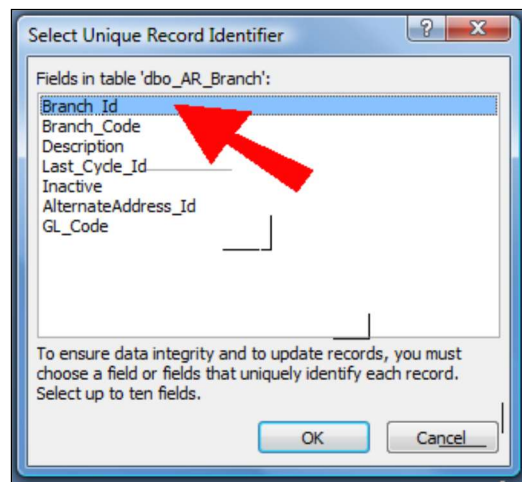


Then choose the tables you wish to Import or Link and click OK. You can choose multiple tables but do not select all. Access is not as large or as powerful as SQL Server. Choosing all will probably crash Access.

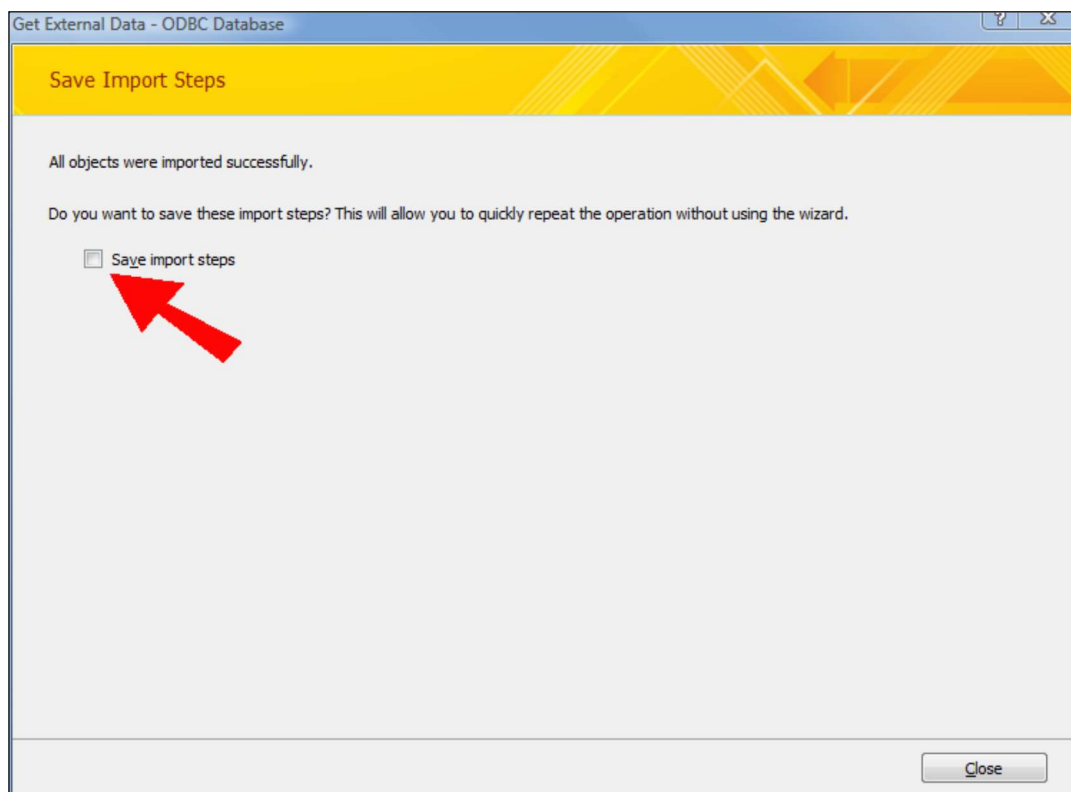




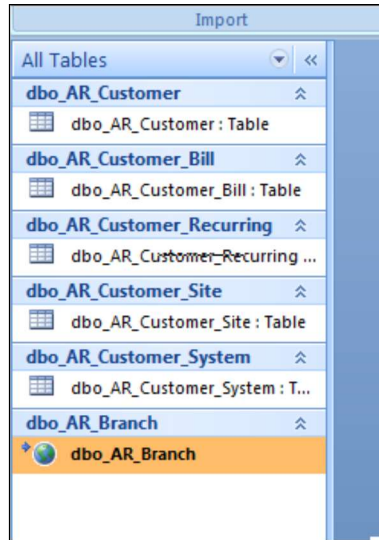
If you chose to Link, you will be asked to “Select Unique Record Identifier”. This is always the top item in SedonaOffice.



If you chose Import, when the operation is complete a window will be displayed showing the success of the operation. Here you can also Save the steps you just completed so that refreshing the data will be easier.



Your tables will then be accessible in Access. You may mix Import and Link in the same Access database. In the example, I have Imported several customer tables and linked the branch table. Notice the different icons for imported versus linked tables. The highlighted table is the linked branch table.

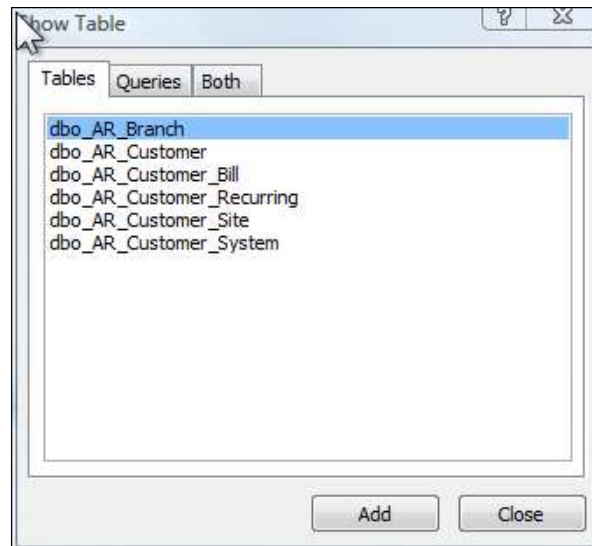


## Writing a query with Access

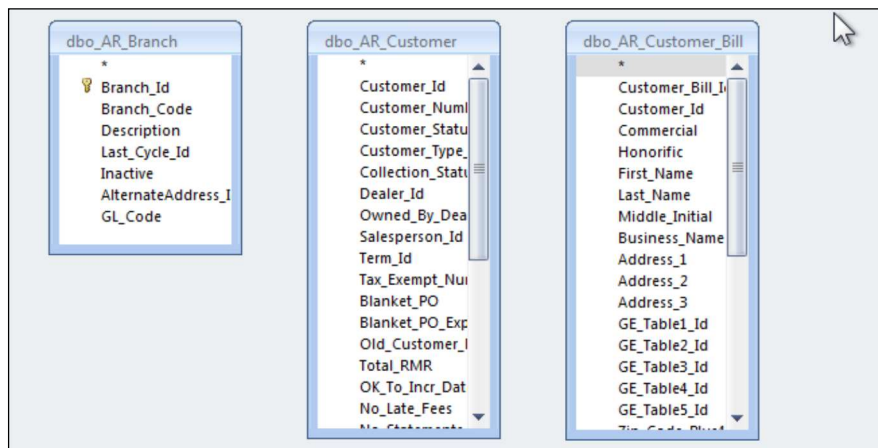
Click on the Create tab and then on Query design.



Choose the tables you wish to include in your query. A table can be selected more than once if you need to join it to more than one Id. For our example we are going to choose all of the tables.



Delete all of the joins that access automatically creates.

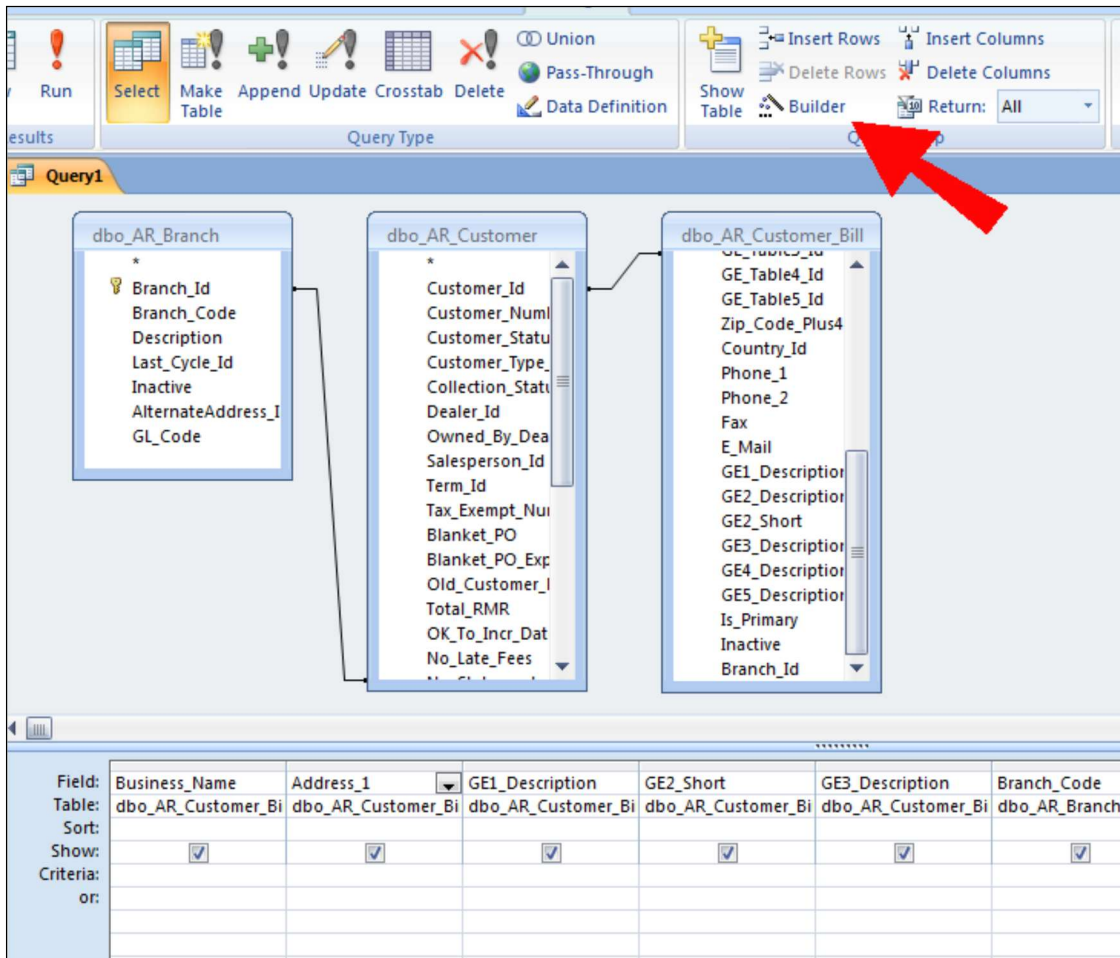


Create the joins according to the structure of SedonaOffice. In this case AR\_Branch.Branch\_Id to AR\_Customer.Branch\_Id and AR\_Customer.Customer\_Id to AR\_Customer\_Bill.Customer\_id.

We are going to create a mailing list, so we need to drag the name and address information to the lower pane. We are also going to drag down the branch code so we can sort on branch.

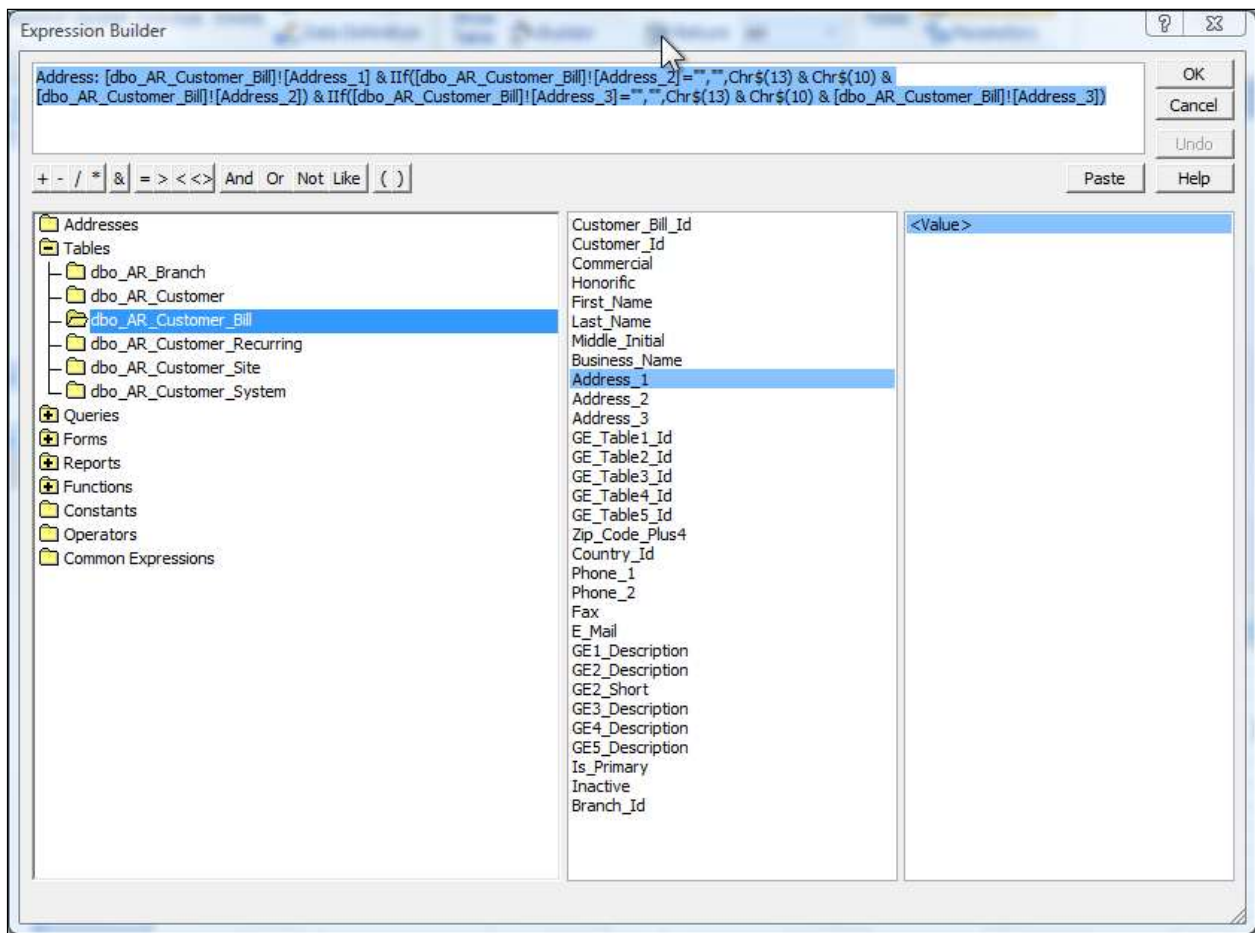
Field:	Business_Name	Address_1	GE1_Description	GE2_Short	GE3_Description	Branch_Code
Table:	dbo_AR_Customer_Bi	dbo_AR_Customer_Bi	dbo_AR_Customer_Bi	dbo_AR_Customer_Bi	dbo_AR_Customer_Bi	dbo_AR_Branch
Sort:						
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:						
or:						

Address\_1 may not be all of the address information needed, but if there is no address\_2 we don't want to add a blank line. So we create a formula. Click in the Address\_1 cell and then click on the formula button.

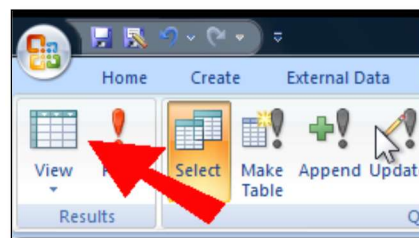


Enter the following into the builder window.

```
Address: [dbo_AR_Customer_Bill]![Address_1] & If([dbo_AR_Customer_Bill]![Address_2]=
"", "", Chr$(13) & Chr$(10) & [dbo_AR_Customer_Bill]![Address_2]) &
If([dbo_AR_Customer_Bill]![Address_3]= "", "", Chr$(13) & Chr$(10) &
[dbo_AR_Customer_Bill]![Address_3])
```



Click view to test our results.



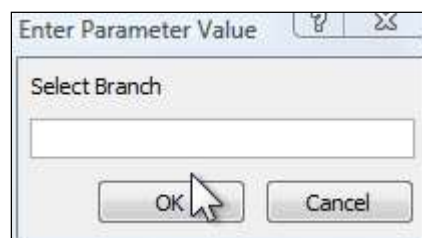
Business_Name	Address	GE1_Description	GE2_Short	GE3_Descrip	Branch_Codi
N/A		N/A	N/A	N/A	Main Division
George Washington	1234 Mount Vernon Lane	Colorado Springs	CO	80919	Major
John Adams	5411 2nd Street	Colorado Springs	CO	80919	Major
Mega Mart	2154 Mountain Springs Road	Colorado Springs	CO	80919	Major
Mega Mart #200	7415 Union Blvd	Colorado Springs	CO	80919	Major
John Wayne	4521 Mountain View Terrace Apartment # 315	Colorado Springs	CO	80919	Major
Roy Rogers	7411 Bullet Lane	Colorado Springs	CO	80919	Major
Gene Autry	7466 Carter Vall Road	Colorado Springs	CO	80919	Major
Clint Eastwood	12445 Happy Acres Drive	Colorado Springs	CO	80919	Major
Andrew Marriott	123 Main Street	Colorado Springs	CO	80919	Major
Rocky Mountain High School	421 Falcon Way	Colorado Springs	CO	80919	Major
TELUS	1234 Main Street	Colorado Springs	CO	80919	Major
Win-Pak	421 Windchime Pl	Colorado Springs	CO	80919	Major
Dealer X	1234 Main Street	Colorado Springs	CO	80919	Major

Now, let's remove the N/A row and add a method to select which branch we want.

Under business\_Name add <>"N/A". Then under Branch\_Code add =[Select Branch].

Business_Name	Address: [dbo_AR_Cu	GE1_Description	GE2_Short	GE3_Description	Branch_Code
dbo_AR_Customer_Bi		dbo_AR_Customer_Bi	dbo_AR_Customer_Bi	dbo_AR_Customer_Bi	dbo_AR_Branch
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<> "N/A"					=[Select Branch]

Now when we return the results we are asked to select a Branch.



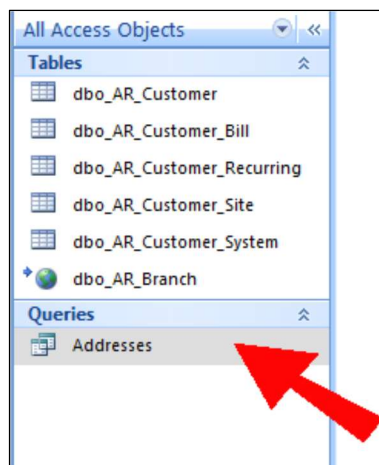
Entering a branch we get results with no N/A.

Business_Name	Address	GE1_Description	GE2_Short	GE3_Descrip	Branch_Codi
TELUS	1234 Main Street	Colorado Springs	CO	80919	Major
Dealer X	1234 Main Street	Colorado Springs	CO	80919	Major
George Washington	1234 Mount Vernon Lane	Colorado Springs	CO	80919	Major
John Adams	5411 2nd Street	Colorado Springs	CO	80919	Major
Mega Mart	2154 Mountain Springs Road	Colorado Springs	CO	80919	Major
Mega Mart #200	7415 Union Blvd	Colorado Springs	CO	80919	Major
John Wayne	4521 Mountain View Terrace Apartment # 315	Colorado Springs	CO	80919	Major
Roy Rogers	7411 Bullet Lane	Colorado Springs	CO	80919	Major
Gene Autry	7466 Carter Valley Road	Colorado Springs	CO	80919	Major
Clint Eastwood	12445 Happy Acres Drive	Colorado Springs	CO	80919	Major
Andrew Marriott	123 Main Street	Colorado Springs	CO	80919	Major
Rocky Mountain High School	421 Falcon Way	Colorado Springs	CO	80919	Major
Win-Pak	421 Windchime Pl	Colorado Springs	CO	80919	Major

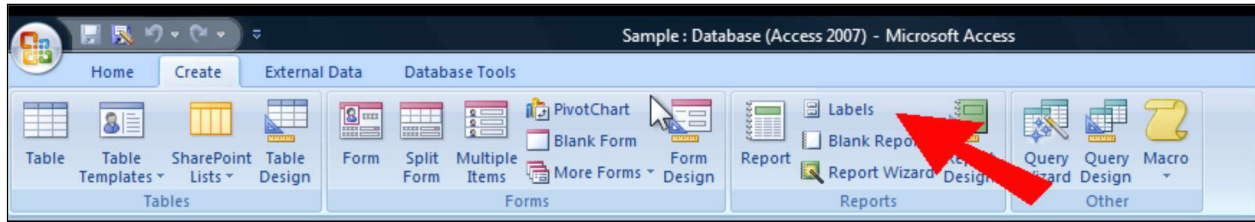
### Creating a Report with Access

Displaying the results on the screen is useful but Access allows us to create reports. The report we are going to create will be used to create mailing labels.

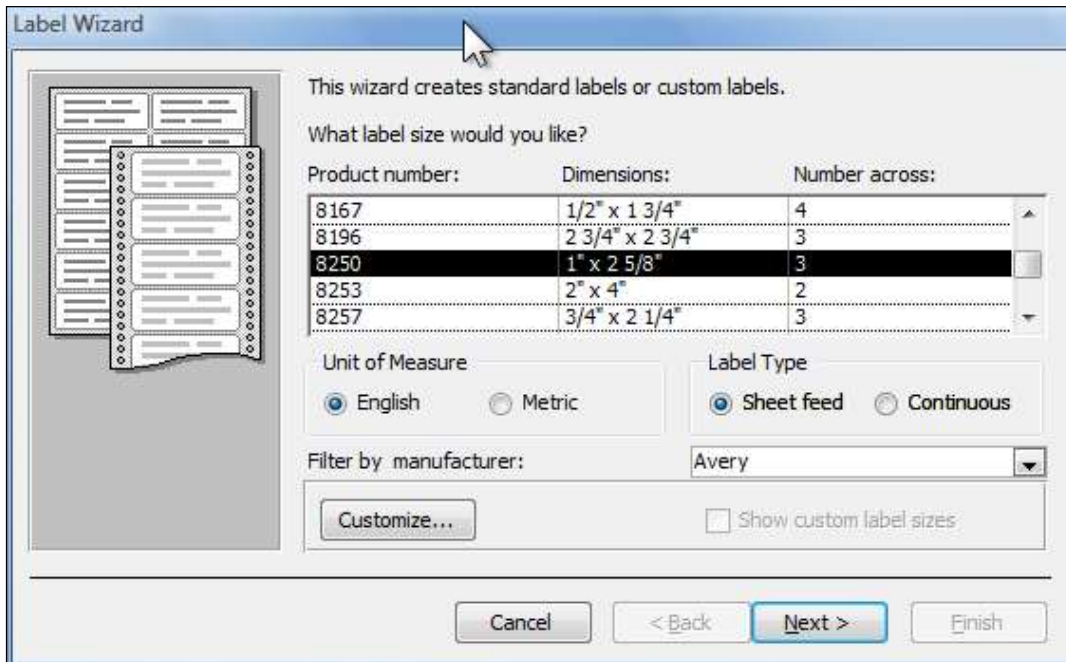
First make sure the new query you created is selected and then launch the label wizard.



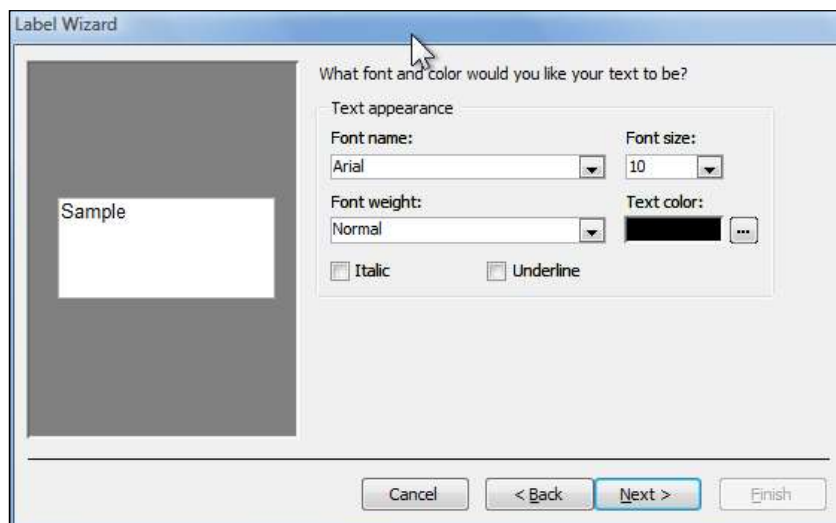




Choose your label. You can choose by the form number if you bought labels from a major manufacturer or just choose a label of the same size as the ones you are using.



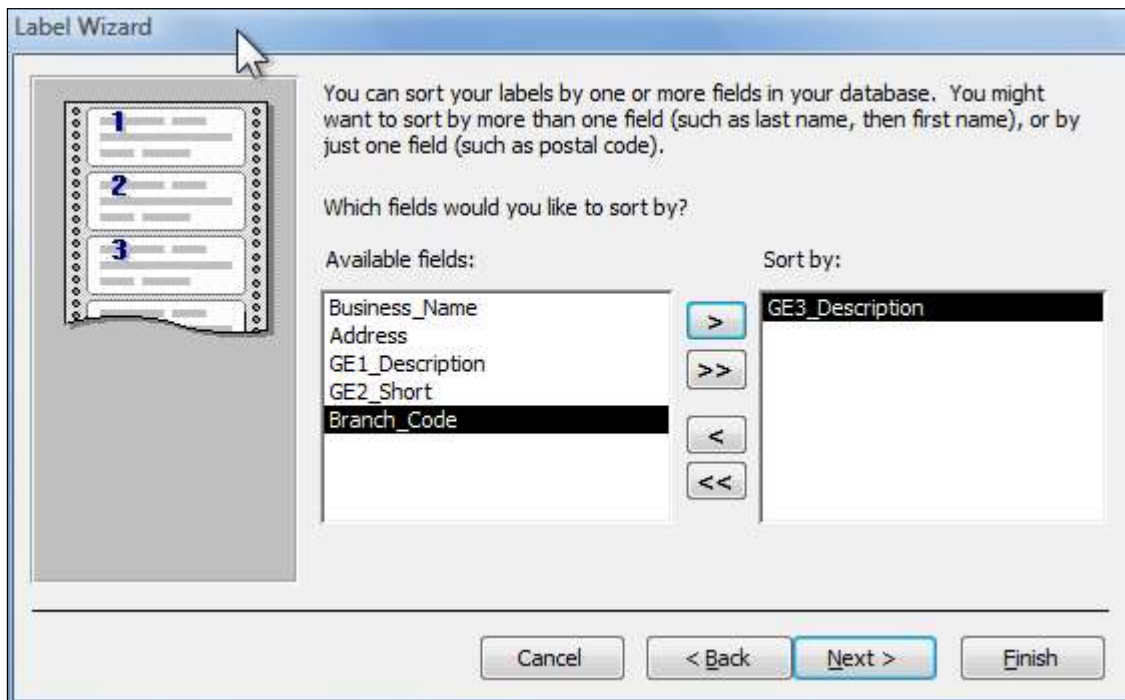
Choose your font.



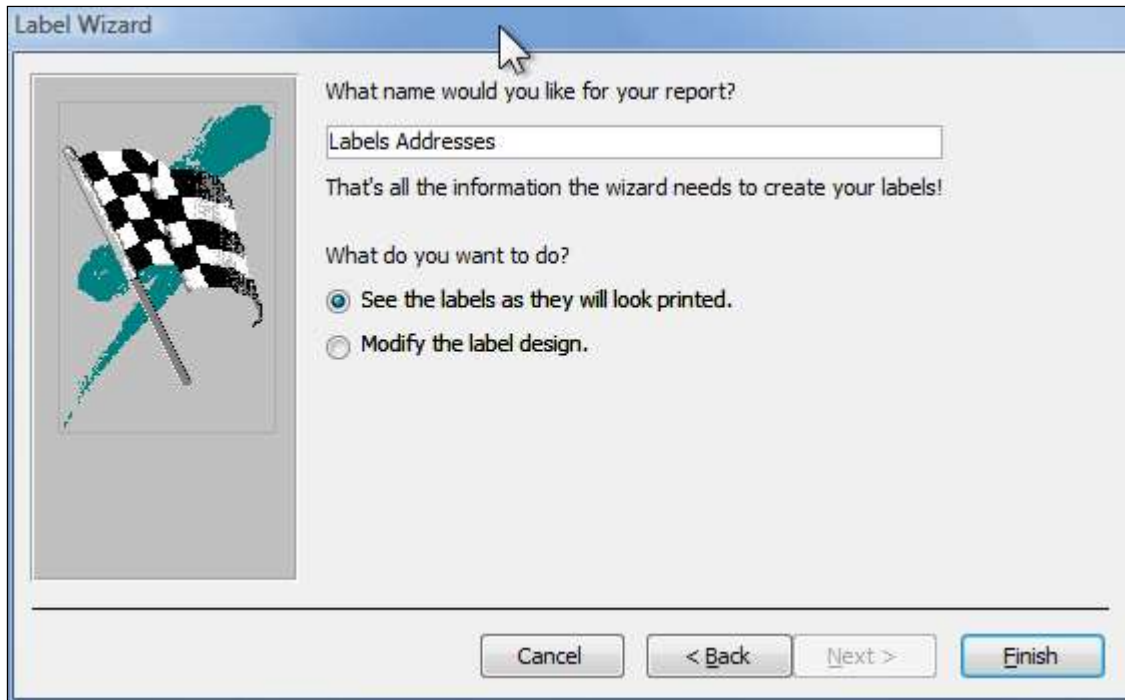
Setup the fields how you want them to appear on the label.



Select any fields you want to sort on. Here I've selected the GE3\_Description so we can get a presorted discount from the post office.



Give your report a name and save it.



Click Finish and see a preview.

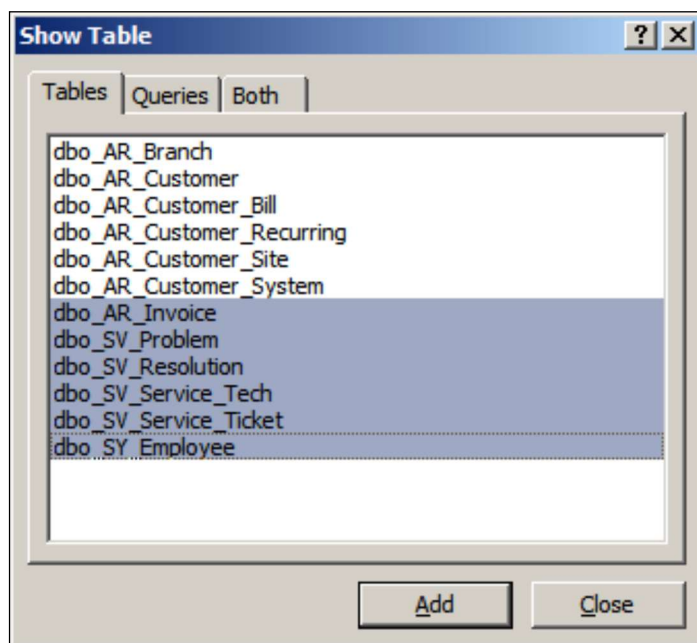
Win-Pak 421 Windchime Pl Colorado Springs CO 80919	Rocky Mountain High School 421 Falcon Way Colorado Springs CO 80919	Andrew Marriott 123 Main Street Colorado Springs CO 80919
Clint Eastwood 12445 Happy Acres Drive Colorado Springs CO 80919	Gene Autry 7466 Carter Valley Road Colorado Springs CO 80919	Roy Rogers 7411 Bullet Lane Colorado Springs CO 80919
John Wayne 4521 Mountain View Terrace Apartment # 315 Colorado Springs CO 80919	Mega Mart #200 7415 Union Blvd Colorado Springs CO 80919	Mega Mart 2154 Mountain Springs Road Colorado Springs CO 80919
John Adams 5411 2nd Street Colorado Springs CO 80919	George Washington 1234 Mount Vernon Lane Colorado Springs CO 80919	Dealer X 1234 Main Street Colorado Springs CO 80919
TELUS 1234 Main Street Colorado Springs CO 80919		

## Creating a Grouped and Sub Totaled Report

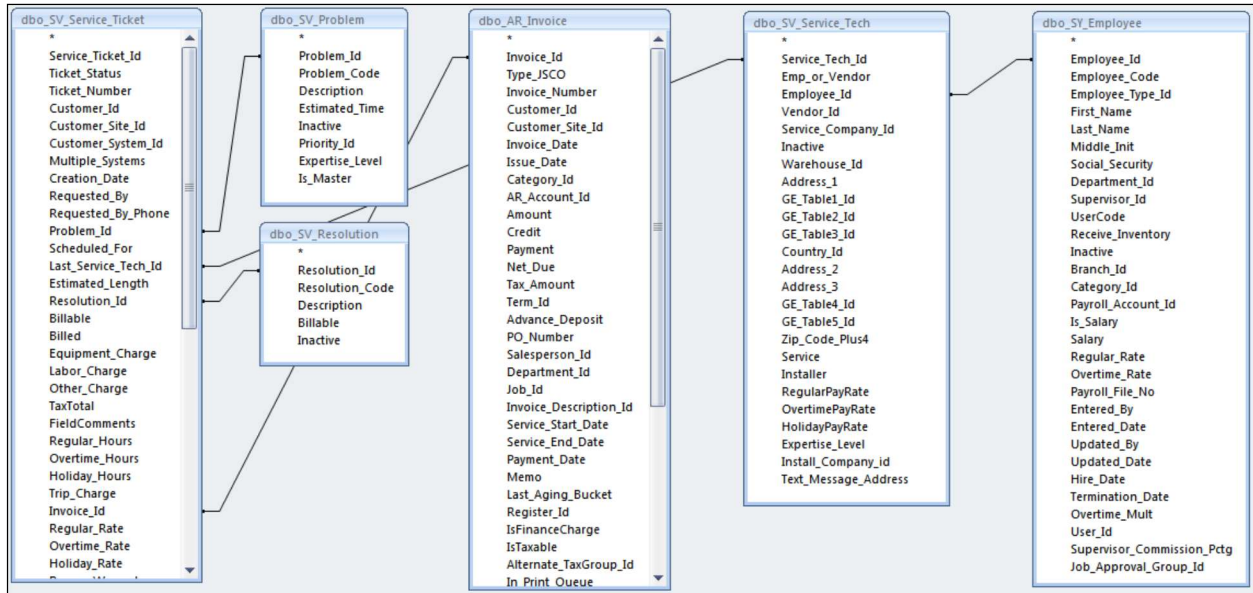
First we will need some additional data. Again select the ODBC database import item. Add these additional tables:

- SV\_Service\_Ticket
- SV\_Problem
- SV\_Resolution
- AR\_Invoice
- SV\_Service\_Tech
- SY\_Employee

Create a new Query and add these tables.



Link the tables as shown.

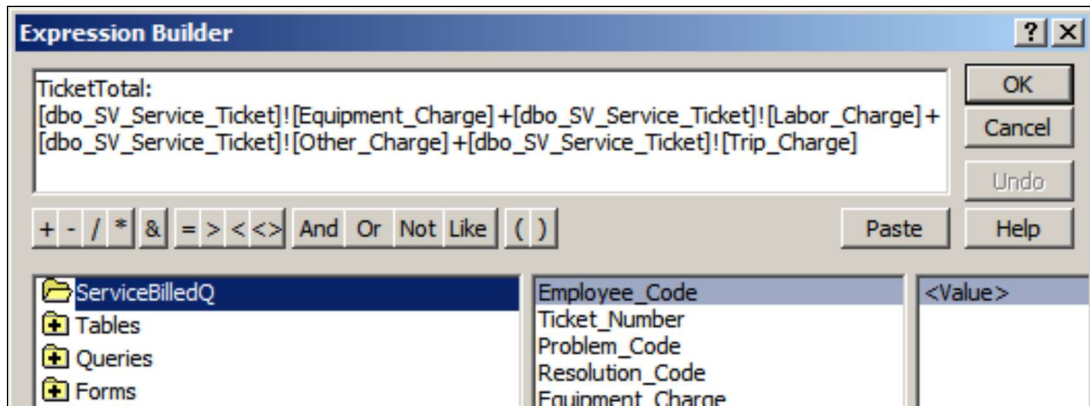


Add these fields.

- `dbo_SY_Employee.Employee_Code`
- `dbo_SV_Service_Ticket.Ticket_Number`
- `dbo_SV_Problem.Problem_Code`
- `dbo_SV_Resolution.Resolution_Code`
- `dbo_SV_Service_Ticket.Equipment_Charge`
- `dbo_SV_Service_Ticket.Labor_Charge`
- `dbo_SV_Service_Ticket.Other_Charge`
- `dbo_SV_Service_Ticket.Trip_Charge`
- `dbo_AR_Invoice.Invoice_Number`
- `dbo_AR_Invoice.Amount`
- `dbo_SV_Service_Ticket.Ticket_Status`
- `dbo_SV_Service_Ticket.Service_Ticket_Id`

Now we need to create a calculated field. We want a field that will be the sum of all of the charges on the ticket. So open the Build dialog and enter these fields.

- `dbo_SV_Service_Ticket.Equipment_Charge`
- `dbo_SV_Service_Ticket.Labor_Charge`
- `dbo_SV_Service_Ticket.Other_Charge`
- `dbo_SV_Service_Ticket.Trip_Charge`

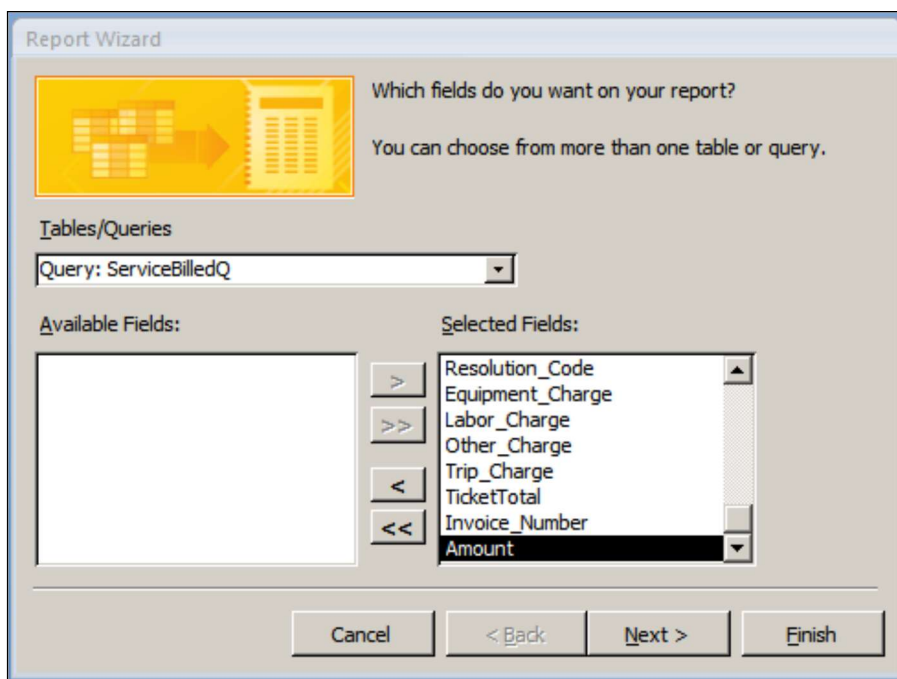


And select OK.

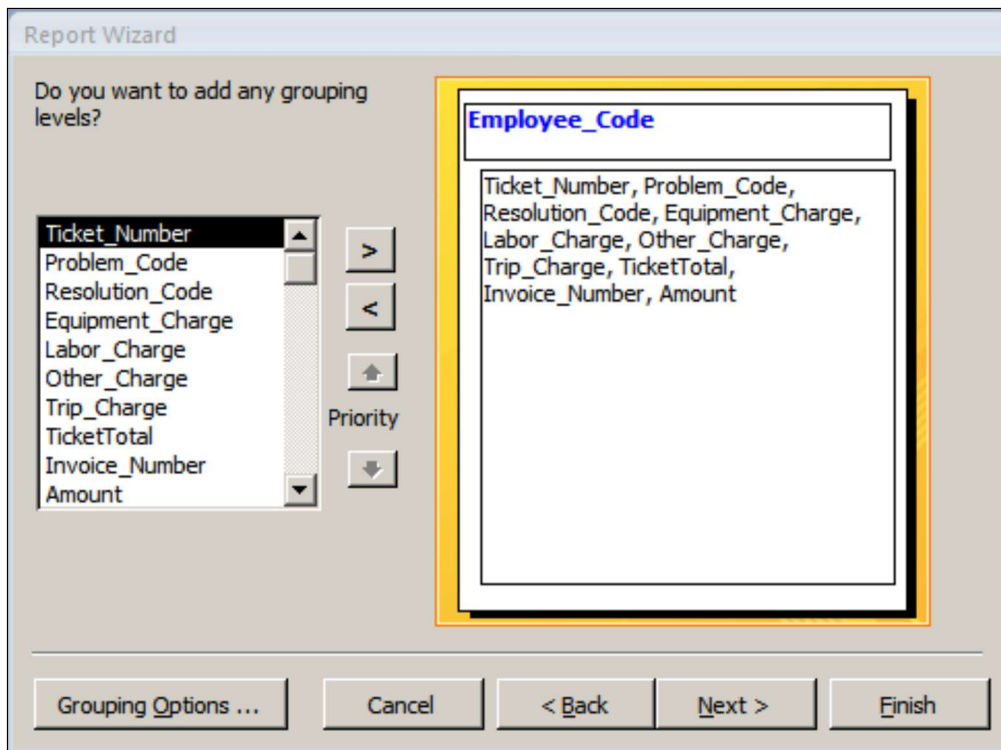
Also add criteria for Ticket\_Status and Service\_Ticket\_Id.

Ticket_Status	Service_Ticket_Id
dbo_SV_Service_Ticke	dbo_SV_Service_Ticke
<input type="checkbox"/>	<input type="checkbox"/>
"CL"	<>1

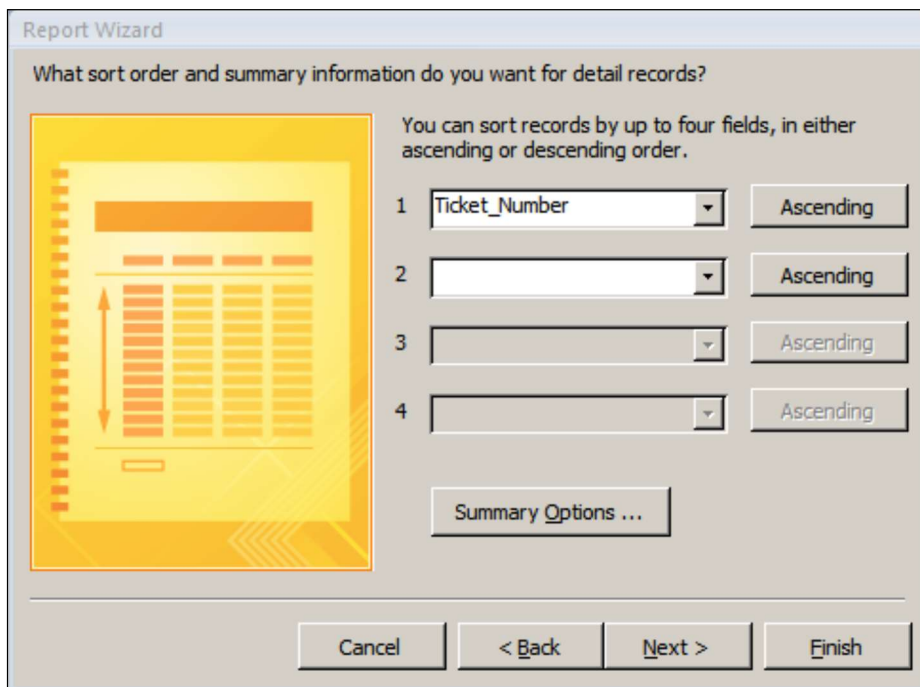
Save the query as ServiceBilledQ. Create a new report and select ServiceBilledQ as the data source, select all of the fields and press next.



We are going to group by Employee\_Code. Select the right pointing arrow while Employee\_Code is highlighted. The result should look like the image below.

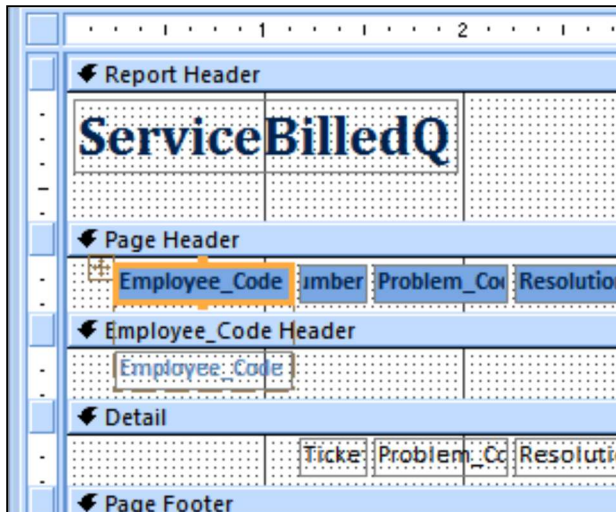


Press Next. We want to sort the tickets by Ticket\_Number for each Service\_Tech which we can do on the next page of the wizard.

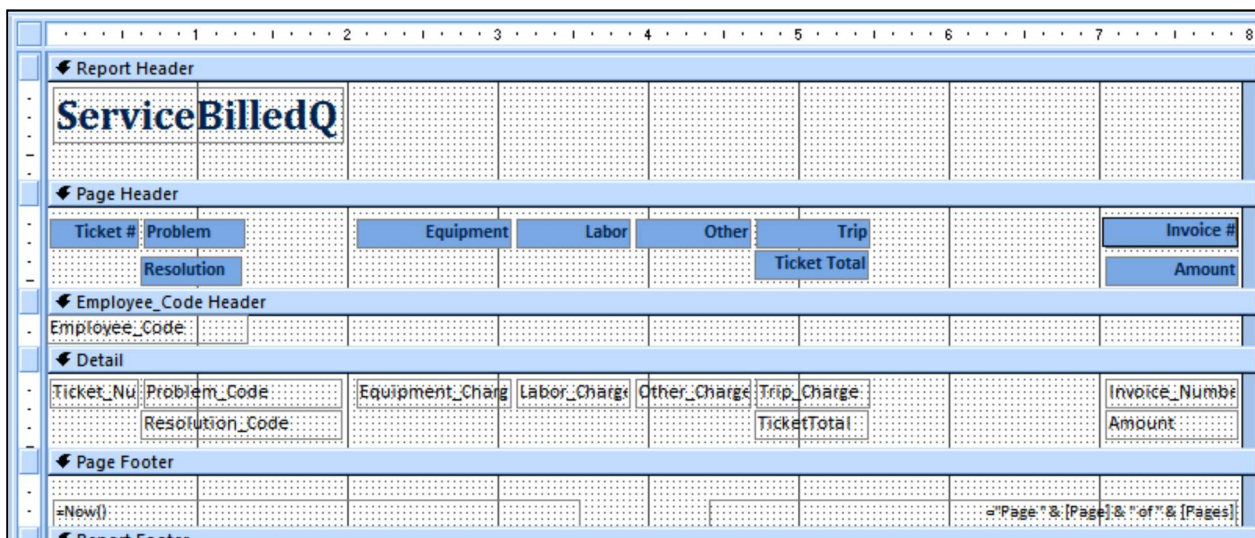


Select Finish. A preview of the report will display. Close the preview for now and you will get the report designer. We can now make changes to the basic report to suit our needs.

Click on any blank spot on the report to deselect everything. Now select the Employee\_Code label.

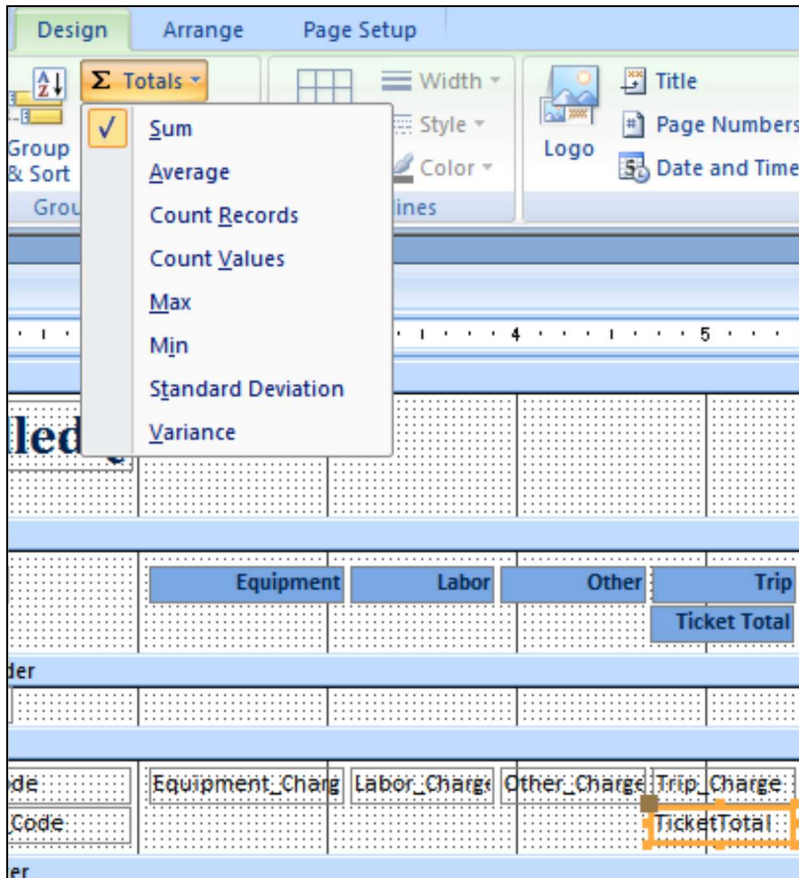


Delete the label. If you delete the Field just drag it back from the fields list. Continue moving fields till your layout looks like the image below.





Now we are going to add subtotals. Select the TicketTotal field in the report detail section, then choose the Totals menu and Sum in the Design bar.



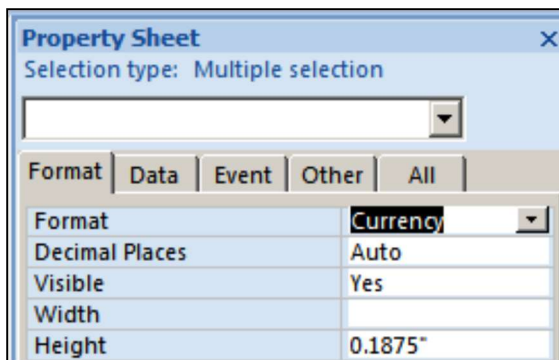
Do the same for the Amount field from the invoice. We will now have a subtotal by service tech and a grand total for all service techs.

Charge	Trip_Charge		Invoice_Numb
	TicketTotal		Amount
	=Sum([Ticket		=Sum([Amount
		= "Page " & [Page] & " of " & [Pages]	
	=Sum([Ticket		=Sum([Amount

We are almost done. Choose all of the currency fields.

	Equipment_Charge	Labor_Charge	Other_Charge	Trip_Charge		Invoice_Number
				TicketTotal		Amount
				=Sum([Ticket		=Sum([Amount
						= "Page " & [Page] & " of " & [Pages]
				=Sum([Ticket		=Sum([Amount

And then in the Properties dialog, choose "Currency" for the format.



Add a line above the subtotals and set it to black. Add one line above and two lines below the grand total and set their color to black. Save the report and preview it.

<b>ServiceBilledQ</b>						
Ticket #	Problem	Equipment	Labor	Other	Trip	Invoice #
	Resolution				Ticket Total	Amount
<b>Barney Barber</b>						
7007	Keypad Trouble	\$542.10	\$770.00	\$0.00	\$0.00	\$1,007.00
	Add Equipment				\$1,312.10	\$1,393.60
					<b>\$1,312.10</b>	<b>\$1,393.60</b>
<b>Ben Bainbridge</b>						
7000	Keypad Trouble	\$275.85	\$30.00	\$65.00	\$65.00	\$1,001.00
	Replace Equipment				\$435.85	\$397.06
7001	Inspection	\$22.02	\$90.00	\$65.00	\$65.00	\$1,003.00
	Insp Comp.				\$242.02	\$179.11
7016	Keypad Trouble	\$0.00	\$0.00	\$0.00	\$0.00	\$0.00
	Replace Equipment				\$0.00	\$0.00
					<b>\$677.87</b>	<b>\$576.17</b>
<b>Cain Cabe</b>						
7013	Keypad Trouble	\$125.00	\$30.00	\$65.00	\$65.00	\$1,014.00
	Replace Equipment				\$285.00	\$231.88
7014	Keypad Trouble	\$202.20	\$90.00	\$65.00	\$65.00	\$1,038.00
	Replace Equipment				\$422.20	\$376.41
					<b>\$707.20</b>	<b>\$608.29</b>
					<b>\$2,697.17</b>	<b>\$2,578.06</b>

There are a number of improvements that could be added to the report. The title should be changed. Dates could be added. Perhaps some additional Customer information could be included.

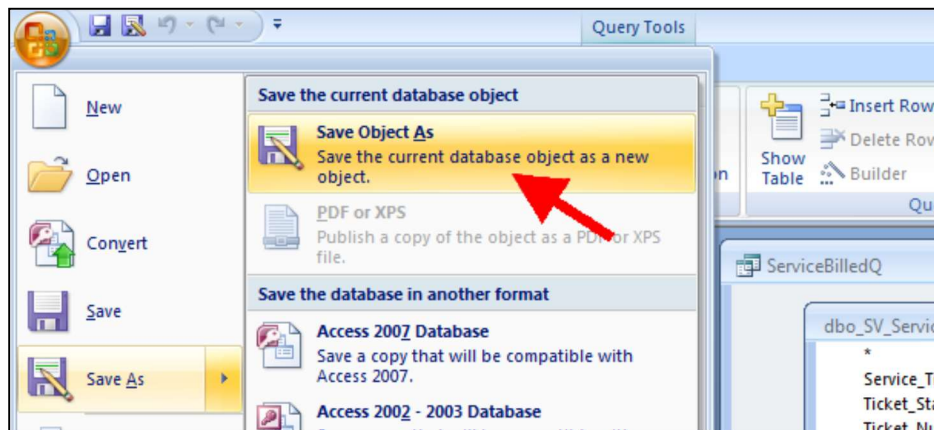
In this report we have learned how to group and total. We have learned how to expand the detail section to show more data than will fit on one line.

## Using SedonaOffice Data in a Microsoft Mail Merge

Now we will create a mail merge using Access and Word. For our example, let's create a query of customers that do not have a service contract and were charged for a service call last year. From this data we will create letters offering these customers a service contract.

### Creating the List of Customers

We will use the query we created in our last example as a starting point. Open the ServiceBilledQ query from the last example and save it as MailMergeQ.



Again select the ODBC database import item. Add these additional tables:

- AR\_Customer\_Recurring
- AR\_Item

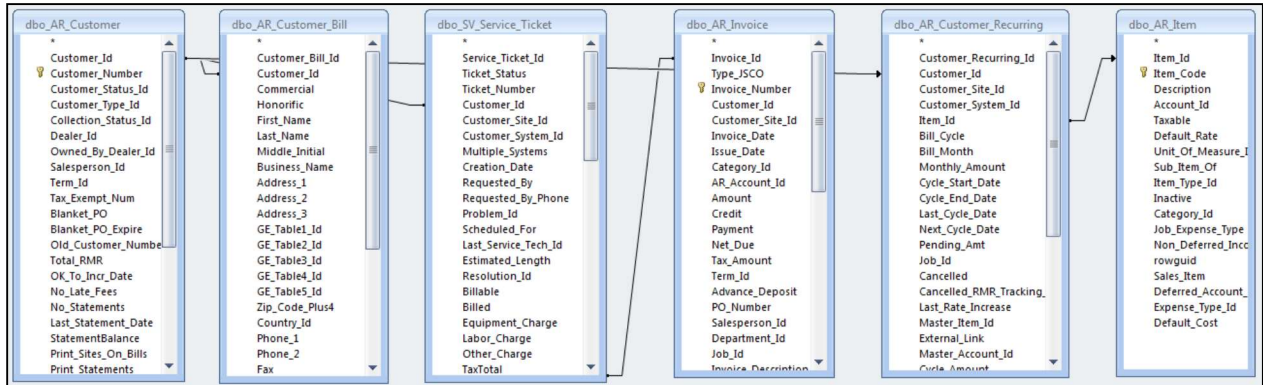
Then, using the table list, add these tables to the new query:

- AR\_Customer
- AR\_Customer\_Bill
- AR\_Customer\_Recurring
- AR\_Item

Remove these tables:

- SV\_Problem
- SV\_Resolution
- SV\_Service\_Tech
- SY\_Employee

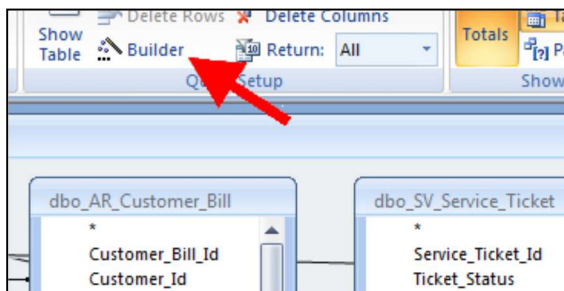
Link the tables as shown:

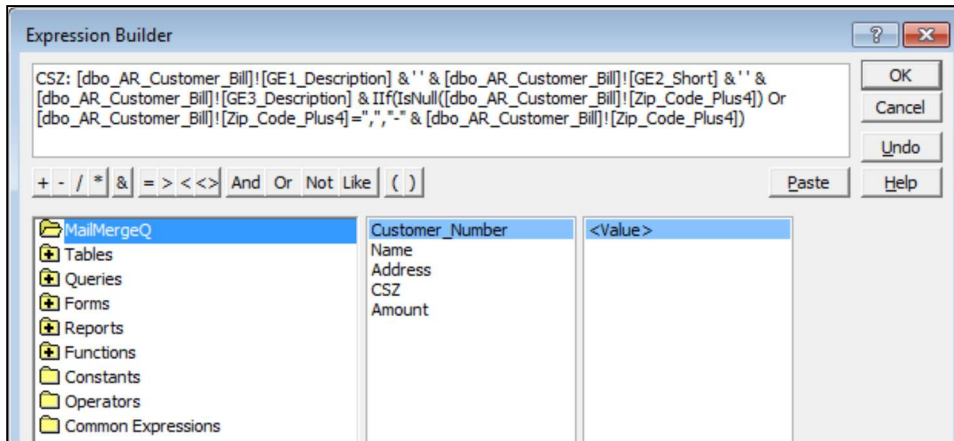


Add and remove fields till you have this list:

- AR\_Customer.Customer\_Number
- AR\_Customer\_Bill.Business\_Name
- AR\_Customer\_Bill.Address\_1
- CSZ
- AR\_Invoice.Amount
- AR\_Customer\_Bill.Customer\_Bill\_Id
- SV\_ServiceTicket.Creation\_Date
- AR\_Item,Item\_Code

Name the Business\_Name field “Name” and the Address\_1 field “Address”. The CSZ field is a calculated or formula field. You create it with the Builder function.

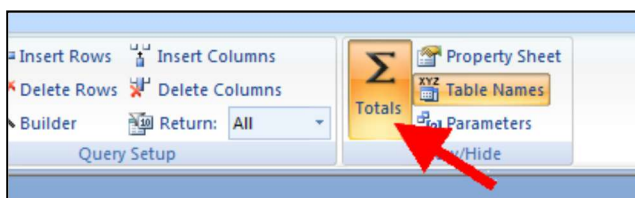




This formula could be simpler but I added an IIf function to format the City State and Zip differently if there was a Zip Plus 4. Add the following “Where” entries:

Amount: Amount	Customer_Bill_Id	Creation_Date	Item_Code
dbo_AR_Invoice	dbo_AR_Customer_Bi	dbo_SV_Service_Ticket	dbo_AR_Item
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<>0	<>1	>=CDate("01/01/2010") And <=CDate("01/01/2011")	Is Null
<>0	<>1	>=CDate("01/01/2010") And <=CDate("01/01/2011")	<>"Service Contract"

Under the item code we have the “Is Null” check which is true if they have no AR\_Customer\_Recurring records and the <> “Service Contract” which is only true if they do not have a Service Contract. Of course this will have to be changed to the code or codes that you use for service contracts. Finally turn on grouping.



Unlike the last report, here we want to use some of the special features of grouping. So for each column, select the grouping actions as follows:

Field:	Customer_Number	Name: Business_Name	Address: Address_1	CSZ: First[(dbo_AR_Cu	Amount: Amount	Customer_Bill_Id	Creation_Date	Item_Code
Table:	dbo_AR_Customer	dbo_AR_Customer_Bi	dbo_AR_Customer_Bi		dbo_AR_Invoice	dbo_AR_Customer_Bi	dbo_SV_Service_Ticket	dbo_AR_Item
Total:	Group By	First	First	Expression	Sum	Where	Where	Where

Let's discuss these options. "Group By" is the default and basically it says create a record for each unique row. "First" means to not create a record for each unique row, but create a record only for the first row generated for that group. Notice that "First" can be either in the Total row or in the formula as it is in the CSZ column. When this appears in the formula, then "Expression" is used on the total line to show that the option is in the Formula. "Sum" means to create a sum of the values rather than a separate record or row for each value.

This allows us to get a record with the sum of all of the invoices rather than a record for each invoice. Finally we have the "Where" option. This tells the query not to create a row based on these values but to use these values to see if it is included at all. Thus we can exclude invoices from other years, people with service contracts or people that did not receive an invoice. Once this is completed, save your query, and open Word.

## Creating the Letter

I like to start my mail merges by creating my letter as if it were not a mail merge. I used a template downloaded from Microsoft as my base letter. Below is the basic letter after I modified the Microsoft template:

Sandbox Alarm Company  
1234 Main St  
Plymouth, MI 48170  
January 8, 2011

[Company Name]  
[Street Address]  
[City, ST ZIP Code]

[Recipient Name]:

Our records show that last year you were invoiced \$0.00 for service calls. You might like to know more about our service contract, which covers the cost of parts and labor for normal repairs. The actual cost varies, depending on your alarm system, but it is usually less than the time and materials costs of a service call without a service contract.

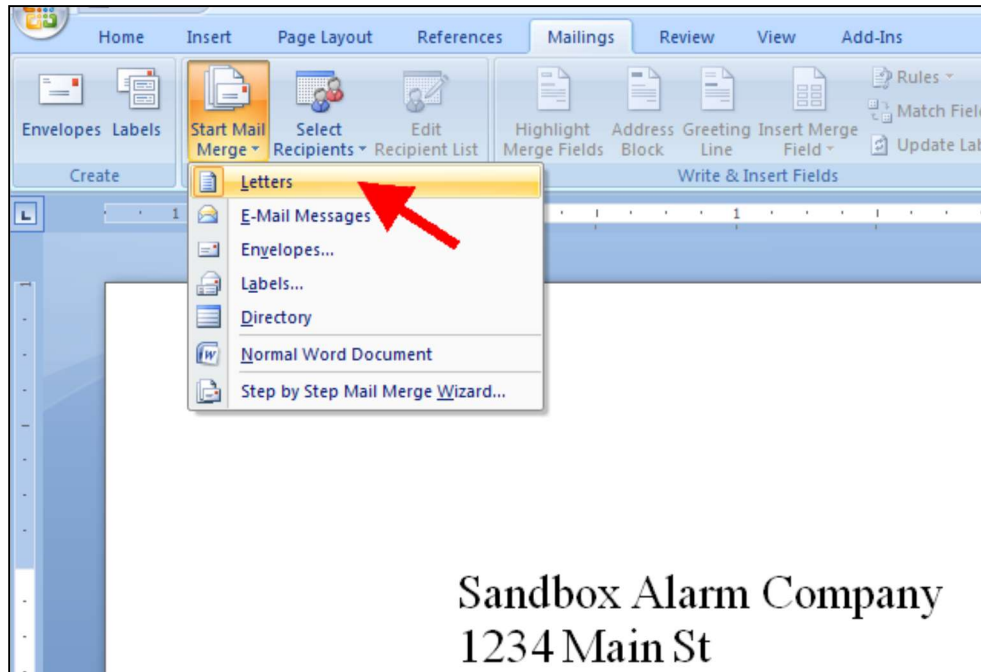
Mr. Jonathan Haas, the professional responsible for your alarm installation, will be happy to meet with you and give you a free estimate.

Sincerely,

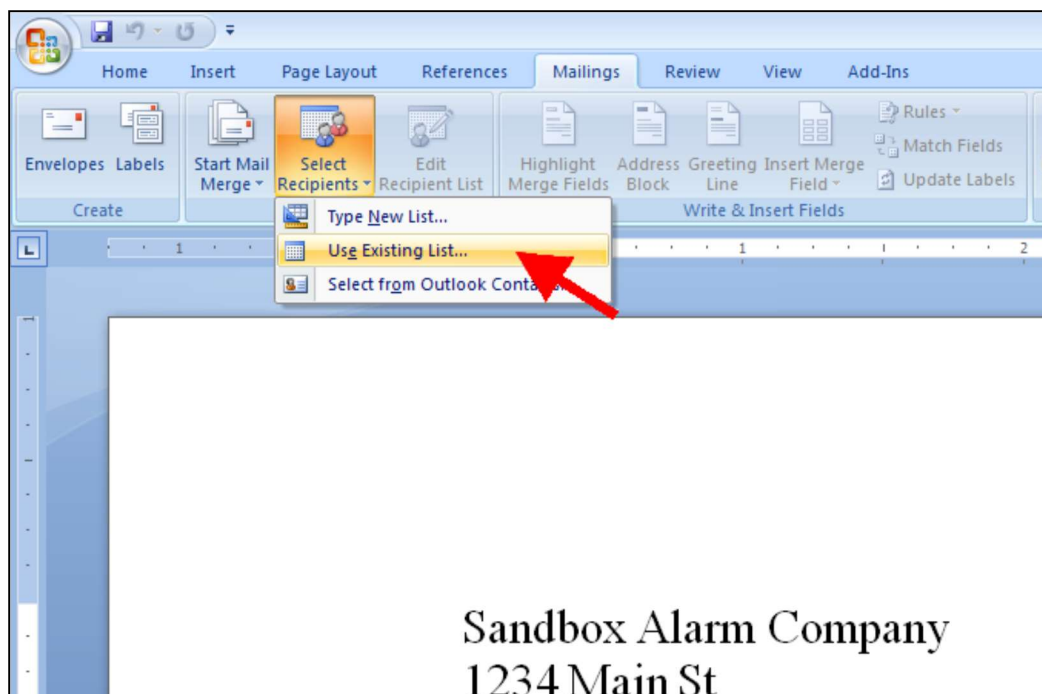
Mathew S. Howe  
General Manager



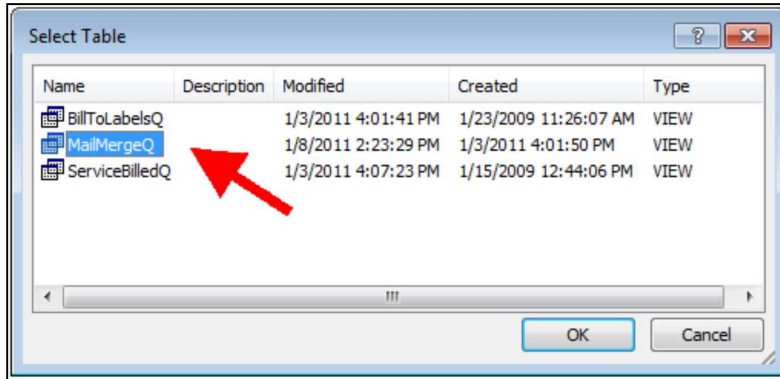
Now, go to the mailings tab and choose “Start Mail Merge” and “Letters”:



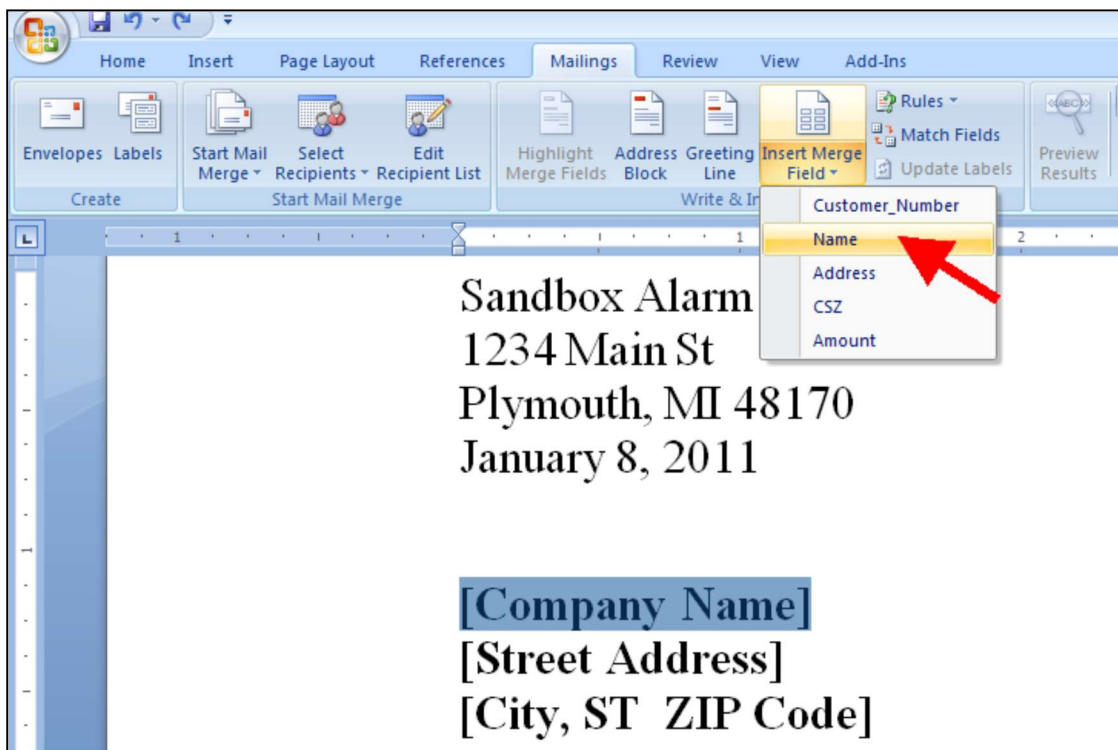
Next choose your recipients:



Locate and select your Access database and choose the MailMergeQ query:



Highlight the text you wish to replace with a field from your query, click on “Insert Merge Field” and choose the field you want:



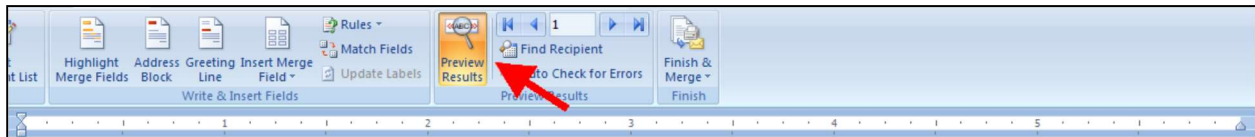
The selected text will be replaced with the name of the field:

<p>«Name» [Street Address] [City, ST ZIP Code]</p> <p>[Recipient Name]:</p> <p>Our records show that last year you</p>
--

Insert the rest of the fields till your letter looks like this:

<p>Sandbox Alarm Company 1234 Main St Plymouth, MI 48170 January 8, 2011</p> <p>«Name» «Address» «CSZ»</p> <p>«Name»:</p> <p>Our records show that last year you were invoiced \$«Amount» for service calls. You might like to know more about our service contract, which covers the cost of parts and labor for normal repairs. The actual cost varies depending on your alarm system, but it is usually less than the time and materials costs of a service call without a service contract.</p> <p>Mr. Jonathan Haas, the professional responsible for your alarm installation, will be happy to meet with you and give you a free estimate.</p> <p>Sincerely,</p>
--

Notice I replaced the 0.00 with the Amount field but left the dollar sign in place. Now we can preview our letter:



Sandbox Alarm Company  
 1234 Main St  
 Plymouth, MI 48170  
 January 8, 2011

A & L Renna Service  
 66 N Park St  
 E Orange NJ 07017

A & L Renna Service:

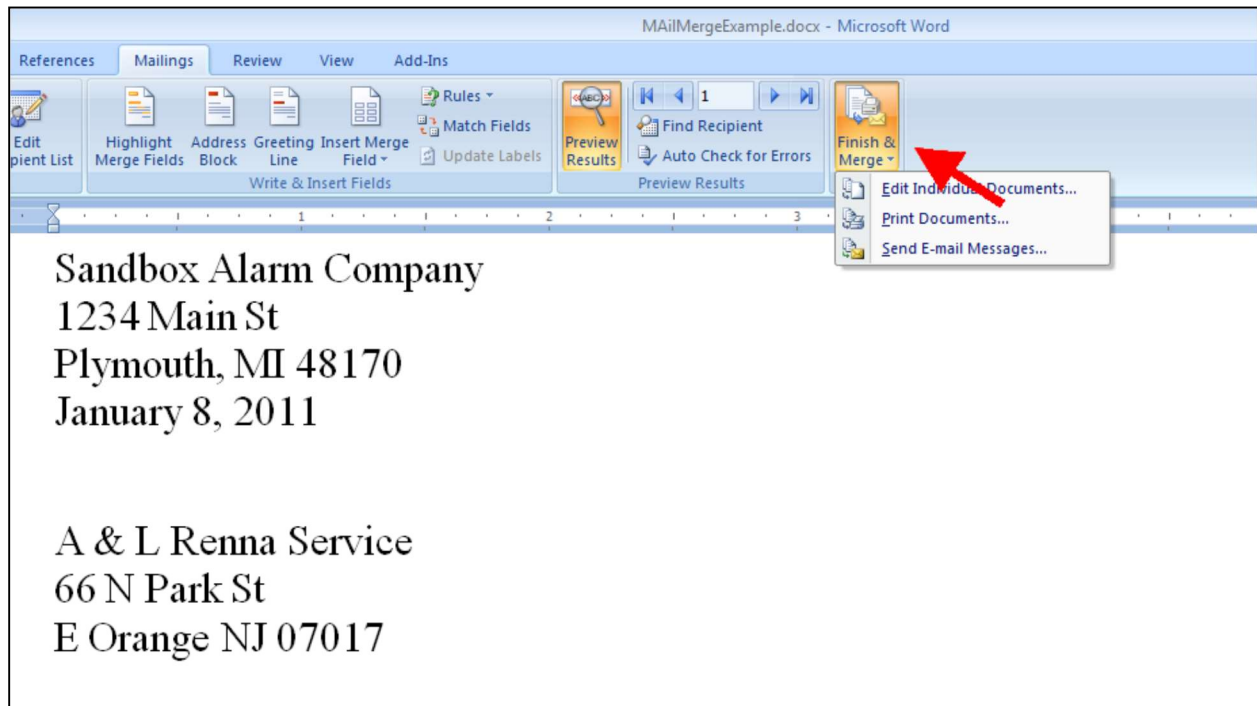
Our records show that last year you were invoiced \$606.24 for service calls. You might like to know more about our service contract, which covers the cost of parts and labor for normal repairs. The actual cost varies depending on your alarm system, but it is usually less than the time and materials costs of a service call without a service contract.

Mr. Jonathan Haas, the professional responsible for your alarm installation, will be happy to meet with you and give you a free estimate.

Sincerely,

We are now ready to print our form letters. You have three choices. If you want to keep a copy of the letters on your computer choose the first choice. That will create a document containing all of the merged letters. Or you can just print them using the second option. Finally there is a function to Email the letters but we won't be covering that here.

## Merging and Creating the Letters



In this section we have learned more about grouping in queries. We have also learned how to use our queries in mail merges to create form letters.

## Basic SQL Language

The majority of SQL queries can be created with just four commands; Select, From, Where and Order By.

### Select Keyword

The “Select” keyword prefaces the list of data to return. This data can be; fields, calculated fields, constants or sub queries. Each of these data items must be separated by a comma.

Fields are the individual fields from the records in the tables. They are collected and displayed as they are in the database. This is the most common use for the select keyword; examples might be Quantity, Rate, Part\_Code, Business\_Name, etc.

Calculated fields are fields that have had a process applied to them. For instance a calculated field might be  $\text{Quantity} * \text{Rate}$  to get the extended price. Another example would be  $\text{GE3\_Description} + '-' + \text{ZipPlus4}$  to get a complete U.S. zipcode.

Constants are numbers or characters that you want to show in your query and will be the same for the entire column. The ‘-’ above is a constant. So that  $\text{GE3\_Description}$ , ‘-’ and  $\text{ZipPlus4}$  could be in separate columns if separated by commas instead of being joined together with plus signs. Calculated fields will have no name unless you assign one with the “As” keyword. An example of using the “As” keyword would be  $\text{Quantity} * \text{Rate}$  As  $\text{Extended\_Price}$ . Notice the underscore in  $\text{Extended\_Price}$ . Names must be one word unless you place them in single quotes, ‘Extended Price’ The “As” keyword will also work with regular fields for example  $\text{Business\_Name}$  As  $\text{Name}$ .

An example of a Select clause would be:

```
Select cu.Customer_Number, cb.Business_Name, cb.Address_1, cb.GE1_Description,  
cb.GE2_Short, cb.GE3_Description
```

### Distinct and Top modifiers

The “Select” keyword has two modifiers, “Distinct” and “Top”. “Distinct” causes the “Select” keyword to only display unique rows. If there are more than one identical rows exactly the same only one will be displayed. If I select  $\text{GE2\_Short}$  from my  $\text{AR\_Customer\_Site}$  table in my test database I get 10185 rows. I get a row for every Site I have. If I add the “Distinct” keyword I get back 26 rows, the 26 states that I have sites in whether it is the 6402 I have in NJ or the one I have in OK it only returns one row.

The “Top” keyword controls how many rows to return. Top 3 would return the first three rows from the rows meeting our criteria, even if 10,000 rows met our criteria, only three would be returned. To control which rows are first see the “Order By” keywords below.

## From Keyword

The “From” keyword prefaces the list of tables and how they are joined. IE

```
From AR_Customer cu Inner Join AR_Customer_Bill cb On cu.Customer_Id = cb.Customer_Id
```

“From” is the keyword, followed by the first table name and an optional short nickname or alias. Next comes the type of Join, which will be discussed below. Then the second table is added, also followed by an optional alias. After the two tables are named comes the “On” keyword. After the “On” keyword are the conditions of how the tables relate to one and another, in this case only return rows where for each Customer\_Id in AR\_Customer there is a matching Customer\_Id in AR\_Customer\_Bill.

## Join Keyword

Joins come in different types. The most common type and the type that is used by default if no other type is specified is the Inner Join.

### Inner Join

Inner joins only return rows where both tables are equal. Using the example below, records 2, 4 and 6 are returned because those are the only records present in both tables.

Table 1	Table 2	Join
1	2	2,2
2	4	4,4
3	6	6,6
4	7	
5	8	
6	9	

### Left Outer Join

Left and Right Outer joins return rows containing all of the records from one table and only the matching records from the other table. So in our example below, all of the records from Table 1 are returned but only 2, 4, and 6 are returned from Table 2 as they are the only records that match. The Left Outer Join and Right Outer Join differ only in which table is on the left side of the Join keyword and which is on the right side of the Join keyword. Our Left Outer Join example would look like this:

Table 1 Left Outer Join Table 2

Table 1	Table 2	Join
1	2	1
2	4	2,2
3	6	3
4	7	4,4
5	8	5
6	9	6,6

### Right Outer Join

Using the same example data, a Right Outer Join would return rows containing all of the records from Table 2 and only 2, 4 and 6 from Table 1. Again, which side of the Join Keyword a table is on is the determining factor. Our Right Outer Join example would look like this:

Table 1 Right Outer Join Table 2

Table 1	Table 2	Join
1	2	2,2
2	4	4,4
3	6	6,6
4	7	,7
5	8	,8
6	9	,9



**Full Outer Join**

Full Outer Joins result in all of the records from both tables. It would be as if you added a Left Outer Join and a Right Outer Join together. A Full outer Join would look like this:

Table 1	Table 2	Join
1	2	1,2
2	4	2,4
3	6	3,6
4	8	4,8
5	10	5,10
6	12	6,12
		,8
		,10
		,12

Outer joins of any type are slower than inner joins. Replacing an inner join with a full outer join can change a query that runs in two minutes to one that takes 20 or 30 minutes to run. Only use outer joins when it is necessary.

Alias's can be used to shorten the join clause, for example:

```
From AR_Customer_System
```

```
Inner Join AR_Customer_System_Userdef On
```

```
AR_Customer_System.Customer_System_Id =  
AR_Customer_System_Userdef.Customer_System_Id
```

Can be shortened to:

```
From AR_Customer_System s
```

```
Inner Join AR_Customer_System_Userdef u On s.Customer_System_Id = u.Customer_System_Id
```

Aliases are needed in order to refer to a table in more than one join.

```
From AR_Customer_Recurring r
```

```
Inner Join AR_Item i on r.Item_Id = i.Item_Id
```

```
Inner Join AR_Item m on r.Master_Item_Id = m.Item_Id
```

Allowing you to see both the recurring item and the master recurring item for a recurring record.

A final word on outer joins. If you create a query joining tables from AR\_Customer to AR\_Item it would look something like this"

```
From AR_Customer c
```

```
Inner Join AR_Customer_Site s on c.Customer_Id = s.Customer_Id
```

```
Left outer join AR_Invoice I on s.Customer_Site_Id = i.Customer_Site_Id
```

```
Left outer join AR_Invoice_Item ii on i.Invoice_Id = ii.Invoice_Id
```

```
Left outer join AR_Item it on ii.Item_Id = it.Item_Id
```

We use an inner join between the customer and customer site tables because every customer must have at least one site. For the invoices we use a left outer join because not every site will have invoices, simple. But from there on it gets a little odd. If every invoice must have at least one invoice item, why do we use a left outer join between the invoice and invoice item? This is one of those “little secrets” of SQL. Once you start using outer joins all subsequent joins must be an outer join even if it would qualify for an inner join. So, both the invoice to invoice item join and the invoice item to item joins must be outer joins even though the table structure would allow inner joins. The next question you have is what happens if I use and inner join anyway? Microsoft says it is “undefined” meaning the results may or may not be correct. To avoid unreliable results, just don’t do it. If a table is joined with an outer join, all tables connected to that table must be an outer join also.

### Where Keyword

Where clauses control what rows are returned by matching the records against a set of conditions or filters connected by logical operators. Each condition or filter results in a “True” or “False” condition. Examples of “True” conditions are:

$5 = 5$

$'A' < 'B'$

$3 + 4 = 7$

$4 <> 9$

Examples of “False” conditions are:

$5 <> 5$

$'A' > 'B'$

$4 + 4 = 7$

$4 = 9$

Of course these examples would not do us much good, but we can substitute Fields for the numbers and characters in the conditions, for example:

Amount = 5

BusinessName < 'B'

InvoiceTot - CreditTot = 7

The query will return every row in which the conditions of the Where clause are true. For example, the following query will return only invoices for \$5.00. No other value invoice would be included in the returned rows.

Select

```
Invoice_Number,  
Amount,  
Net_Due  
From  
AR_Invoice  
Where  
Amount = 5
```

Notice we have included the Net\_Due. The value of Net\_Due will not affect what rows are returned. It will only be displayed. If we wanted to include only invoices with an outstanding balance we would change the query to look like this:

Select

```
Invoice_Number,  
Amount,  
Net_Due  
From  
AR_Invoice  
Where  
Amount = 5  
And  
Net_Due > 0
```

This brings up the next concept, logical operators.

### Logical Operators

The most common logical operators are And, Or, Not, Xor, Nand and Nor.

And

Value 1	Value 2	Result
False	False	False
False	True	False
True	False	False
True	True	True

Or

Value 1	Value 2	Result
False	False	False
False	True	True
True	False	True
True	True	True

Not (Reverses any result)

Value 1	Result
False	True
True	False

Xor (Exclusive or)

Value 1	Value 2	Result
False	False	False
False	True	True
True	False	True
True	True	False

Nand (the same as Not(Value 1 And Value 2))

Value 1	Value 2	Result
False	False	True
False	True	True
True	False	True
True	True	False

Nor (the same as Not(Value 1 Or Value 2))

Value 1	Value 2	Result
False	False	True
False	True	False
True	False	False
True	True	False

Another thing to know is the precedence of logical operators. Everyone knows that  $2 + 5 * 3$  is 17 and not 21 because we know that you multiply before you add, this is the precedence of arithmetic operators. Take for example the following data:

ID	Amount	City
1	5.00	Flint
2	5.00	Detroit
3	0.00	Flint
4	0.00	Detroit

If our Where clause is `Amount = 5 And City = 'Flint' Or City = 'Detroit'` we might expect to get rows 1 and 2. In reality we would get rows 1, 2, and 4. Just as  $2 + 5 * 3$  should be thought of being written as  $2 + (5 * 3)$  so the  $5 * 3$  is done first, Our Where clause should be thought of as being written as `(Amount = 5 And City = 'Flint') Or City = 'Detroit'`. The And operator is processed first just like the multiplication operator in arithmetic. If our where clause were written as `Amount = 5 And (City = 'Flint' Or City = 'Detroit')` we would get rows 1 and 2. So the order of precedence is Not, And then Or but the order of precedence should not be relied on. Like the example, to be sure, use parentheses.

**Other Where Clause Filters**

So far we have looked at filters, the true false statements that use the simple comparators listed below:

Comparator	True Examples	False Examples
=	5 = 5, 'A' = 'A'	5 = 7, 'X' = 'R'
<>	5 <> 6, 'AB' <> 'CD'	5 <> 5, 'A' <> 'A'
<	5 < 6, 'A' < 'G'	5 < 5, 6 < 5, 'A' < 'A', 'G' < 'A'
>	6 > 5, 'G' > 'A'	5 > 5, 5 > 6, 'A' > 'G', 'A' > 'A'
<=	5 <= 6, 'A' <= 'G', 5 <= 5, 'A' <= 'A'	6 <= 5, 'G' <= 'A'
>=	6 >= 5, 'G' >= 'A', 5 >= 5, 'A' >= 'A'	5 >= 6, 'A' >= 'G'

Now we will look at Is Null, In, Like and Between. Is Null returns a true if the Field being examined is a Null. Remember, a Null is not the same as a blank "" or a space " ". A Null means not defined or never entered. So using our Right Outer Join example:

Table 1	Table 2	Join
1	2	2,2
2	4	4,4
3	6	6,6
4	7	,7
5	8	,8
6	9	,9

And a where clause something like this:

Where Table1.Field Is Null

We would get the following records returned:

,7
,8
,9

Is Null should not be confused with IsNull. IsNull is a function that allows you to replace nulls with a default value. It replaces only the Nulls, otherwise it uses the value of the Field. If we wanted the nulls to be replaced with a 0 we would write a function like this:

```
IsNull(FieldName,0)
```

Like uses characters and wild cards to create a pattern matching filter. An example of a Like filter:

```
Select
Customer_Number,
Customer_Name
From AR_Customer
Where Customer_Name Like 'A_a%'
```

This would return all customers whose name started with an “A” then contained another character of any sort including spaces, contained an “a” in the third spot followed by zero or more characters of any kind. Below is a chart of the wild cards and what they mean.

Wildcard character	Description	Example
%	Any string of zero or more characters.	WHERE title LIKE '%computer%' finds all book titles with the word 'computer' anywhere in the book title.
_ (underscore)	Any single character.	WHERE au_fname LIKE '_ean' finds all four-letter first names that end with ean (Dean, Sean, and so on).
[]	Any single character within the specified range ([a-f]) or set ([abcdef]).	WHERE au_lname LIKE '[C-P]arsen' finds author last names ending with arsen and starting with any single character between C and P, for example Carsen, Larsen, Karsen, and so on. In range searches, the characters included in the range may vary depending on the sorting rules of the collation.
[^]	Any single character not within the specified range ([^a-f]) or set ([^abcdef]).	WHERE au_lname LIKE 'de[^l]%' all author last names starting with de and where the following letter is not l.



Between takes two parameters and does exactly as one would expect. Here is an example:

Select

Invoice\_Number,

Amount

From AR\_Invoice

Where Amount Between 0.00 AND 15.00

This returns all invoices where the amount is 0.00 through 15.00 inclusive. If you run this it will even return the 1 record which should be the only invoice with a 0.00 Amount. Please notice the “AND” portion of the Between filter. This is not the same as a normal And. It is NOT evaluated with the other And’s, and Or’s. It is just part of the Between filter and should be considered only as part of the Between. Between also works with character values IE:

Where Customer Name Between ‘A’ AND ‘D’

Again this would include all names starting with “A” through name of “D”, not starting with “D” as there are no wild cards here. If you want all of the “D”s, use something like this:

Where Customer\_Name Between ‘A’ AND ‘Dzzz’

If we wanted to look at all of our customer sites in the mid-west region we could do something like this:

Where s.GE2\_Short = ‘OH’

OR s.GE2\_Short = ‘MI’

OR s.GE2\_Short = ‘IN’

OR s.GE2\_Short = ‘IL’

OR s.GE2\_Short = ‘WI’

OR s.GE2\_Short = ‘MN’

Or we could use the “IN” keyword:

Where s.GE2\_Short IN ('OH', 'MI', 'IN', 'IL', 'WI', 'MN')

The “In” keyword checks the value in the field against a list of accepted values. If the field value is in the list it returns true. If not, it returns false. “In” can be used to check against a list as above or the results of a sub query.

### Order By Keyword

Order By controls what order the records are returned in. If no Order By is included, then often the record set will be in the order they were entered, often, but not always. If the order records are returned in is important, use a Order By clause. Here is an example:

```
Select  
Customer_Number,  
Customer_Name  
From AR_Customer  
Where Customer_Name Like 'A_a%'
```

Order By Customer\_Name

Order By can be either ascending (asc) or descending (desc). You can also mix fields and directions. For example:

```
Order By Customer_Number asc, Amount desc, Invoice_Type asc
```

This Order By would take the records and put them in order by Customer\_Number from least to greatest, then the invoices for those customers in order by the amount from Largest to least and finally by Invoice\_Type from first to last. Fields included in the Order By do NOT need to be in the Select clause.

---

## Advanced SQL Language

### Sub Queries

Sub queries have two basic uses. The first is to return lists of values for the “In” check.

```
Where s.GE2_Short in (Select Distinct s.GE2_Short from AR_Customer_Site s
Inner Join AR_Branch b On s.Branch_Id = b.Branch_Id Where b.Branch_Code =
'Midwest')
```

This looks more complicated than the simple list above but it has a major advantage, the list is built dynamically. If someone adds Iowa to the Midwest branch, “IA” will be automatically added to the list for the “In”.

The second use for sub queries is to get aggregated data. This example will list the customer number, site name and how many systems each site has:

```
Select c.Customer_Number,
s.Business_Name,
(Select Count(y.Customer_System_Id) From AR_Customer_System y
Where y.Customer_Site_Id = s.Customer_Site_Id) As 'Number of Systems'
From AR_Customer c
Inner Join AR_Customer_Site s on c.Customer_Id = s.Customer_Id
```

Notice the “As” keyword, without it that column would not have a name. The “Count” keyword will be discussed later in the section on aggregates.

Sub Queries have a few rules; first, they must only return one value. In this case it is the count of systems. Second, notice in the sub query where clause we have “Where y.Customer\_Site\_Id = s.Customer\_Site\_Id”. This is how the sub query knows which systems to count. The y.Customer\_Site\_Id of the sub query is compared to the s.Customer\_Site\_Id from the main query. Again this is something that can only be done through the use of aliases. Were we to leave the where clause out of the sub query we would see the count of systems would be the total number of systems in the database repeated for each customer site. Also sub queries must be surrounded by parenthesis “()”.

## Union Keyword

Sometimes, we need to create a list of records that combines two separate queries. For example, we want a list of open invoices and open credits in order by customer\_number and then date meshed into one recordset.

```
Select
```

```
Customer_Number,
```

```
Customer_Name,
```

```
Invoice_Date,
```

```
Net_Due,
```

```
'I'
```

```
From AR_Customer C Inner Join
```

```
AR_Invoice I On C.Customer_Id = I.Customer_Id
```

```
Where net_Due > 0
```

```
Union
```

```
Select
```

```
Customer_Number,
```

```
Customer_Name,
```

```
Credit_Date,
```

```
-1 * (Amount - Used_Amount),
```

```
'C'
```

```
From AR_Customer C Inner Join
```

```
AR_Credit I On C.Customer_Id = I.Customer_Id
```

```
Where Amount - Used_Amount > 0
```

```
Order By Customer_Number, Invoice_Date
```

Let's look at this query from the top. First we have a select clause. Notice the last item in the list is 'I'. This literal will place a column in our recordset that has an "I" in every row that is an invoice. In the From clause we use an Inner Join to connect the two tables. Notice the use of aliases here, the C and I. This is done just to make the lines a bit more manageable in length. Next we have the Where clause that returns only records that still have a Net\_Due. Now we get to the new clause, the Union keyword. A Union keyword joins the Select query above it with the Select query below it. The number and order of columns must be the same and the data types

for the columns of each query must be compatible. Below the Union we have another query that returns the open credits. In order to find open credits we had to take the Amount – Used\_Amount. We also multiply the result times a negative one so that the credits will be negative compared to the invoices. Also notice that the 'I' field from the top query is now a 'C' and that the Where clause contains a calculated value. Lastly we have an Order By clause. The order by clause must exist after the two queries but the fields must be named from the first query. Also, you may Union as many queries as you want as long as you follow the rules about number, order and type of columns.

There is one option to the Union keyword. If you use the Union key word between two queries and some of the records returned by the first query exactly match some of the records returned by the second query, the final recordset will have only one copy of any record. In other words, all duplicates are reported only once. If you want to see the duplicates, use the All keyword after the Union keyword, IE. Union All

In our example query, this would not be a problem for two reasons. First, we are returning records where the invoices are marked by an 'I' and credits by a 'C'. Secondly, all of the invoices will be positive amounts and all of the credits will be negative amounts.

### Aggregates and Group By

Sometimes in queries you don't want a large list of records, you just want the total. We do this with Aggregates and grouping. We will look at the "Group By" keywords first.

Group By changes the way records are returned. Like records are combined into a single record line. For instance, a simple list of customers with recurring might look like this:

```
Select Customer_Id From AR_Customer_Recurring Where Terminated_RMR = 'N'
```

And return this:

Customer\_Id

384

384

384

384

384

384

384

384

384

384

384

384

384

384

384

384

1374

1374

1374

1374

1374

1374

1374

1758

1758

1758

1758

1758

1758

If we had many more customers in our database this would get out of hand in a hurry. We only want to see one row per customer, so we add a group by like this:

```
Select Customer_Id From AR_Customer_Recurring Where Terminated_RMR = 'N'  
Group By Customer_Id
```

And we get this in return:

Customer\_Id

384

1374

1758

If we wanted to include Customer\_Site\_Id's it would look like this:

```
Select Customer_Id, Customer_Site_Id From AR_Customer_Recurring Where  
Terminated_RMR = 'N' Group By Customer_Id, Customer_Site_Id
```

And we get this in return:

Customer\_Id Customer\_Site\_Id

384 596

384 597

384 599

384 601

1374 1816

1758 2247

Notice we get each Customer\_Id and Customer\_Site\_Id combinations but no duplicates. Also note we added Customer\_Site\_Id to our Group By clause. All fields must be part of the Group By or an aggregate which we will see next.

A list of Id's is nice and we can see how many customers and sites, but it doesn't tell us how many recurrings or provide us with the Monthly\_Amount, for that we need aggregates. Aggregates contain, among others, functions like AVG (average of a column), COUNT (count number of items in a grouped column), MAX (maximum value in a column), MIN (minimum value in a column) and SUM (the sum of the values in a grouped column). Let's look at how we would use the COUNT function first. If we add COUNT functions to our query it will look like this:

```
Select Customer_Id, Customer_Site_Id, COUNT(Customer_Recurring_Id) as  
Recurrings From AR_Customer_Recurring Where Terminated_RMR = 'N' Group By  
Customer_Id, Customer_Site_Id
```

And would return this:

Customer_Id	Customer_Site_Id	Recurrings
384	596	8
384	597	2
384	599	4
384	601	2
1374	1816	7
1758	2247	6

Notice we have to supply a name for the aggregate (as Recurrings) but it simply counts how many rows are grouped together.



We can also count the “Distinct” systems by adding a COUNT containing the DISTINCT key word.

```
Select Customer_Id, Customer_Site_Id, COUNT(distinct Customer_System_Id) as  
Systems, COUNT(Customer_Recurring_Id) as Recurrings  
From AR_Customer_Recurring Where Terminated_RMR = 'N' Group By Customer_Id,  
Customer_Site_Id
```

Would give us:

Customer_Id	Customer_Site_Id	Systems	Recurrings
384	596	1	8
384	597	1	2
384	599	1	4
384	601	1	2
1374	1816	1	7
1758	2247	1	6

The final aggregate that we are going to look at is the Sum function. First we will place a Sum at the end of the select list.

```
Select Customer_Id, Customer_Site_Id, COUNT(distinct Customer_System_Id) as  
Systems, COUNT(Customer_Recurring_Id) as Recurrings , SUM(Monthly_Amount) as  
Monthly  
From AR_Customer_Recurring Where Terminated_RMR = 'N' Group By Customer_Id,  
Customer_Site_Id
```

Which returns:

Customer_Id	Customer_Site_Id	Systems	Recurrings	Monthly
384	596	1	8	411.75
384	597	1	2	104.50
384	599	1	4	203.00
384	601	1	2	57.50
1374	1816	1	7	111.10
1758	2247	1	6	82.50

Each “Monthly” is the sum of all of the monthlies on that system.

## Variables

Variables are temporary storage that can be used like fields except they don’t affect the database. A variable must be declared showing the variable name starting with the “@” character and the variable type. Multiple variables can be created by the same Declare by separating them with commas:

```
Declare @remove_flag char
```

```
Declare @remove_flag char,  
        @credit_amount money,  
        @credit_type nvarchar(15)
```

Common variable types are:

Exact Numerics	money
numeric	
decimal	Approximate Numerics
int	float

### **Date and Time**

date

datetime

time

### **Character Strings**

char

varchar

text

### **Unicode Character Strings**

nchar

nvarchar

ntext

### **Other Data Types**

timestamp

uniqueidentifier

Variables exist as long as the query runs. They can be used every where a data field would be used. They can be assigned a value by using the Select or Set keywords:

```
Select @remove_flag = 'Y'  
Set @credit_amount = Monthly_amount  
Set @credit_type = 'Service Credit'
```

## If, While and Case

If, While and Case control how the query flows and what the query returns to us. They use the same logical methods as the Where clause but can change the entire way a query works.

### If

If we want to make a simple decision, left or right; positive or negative; add or subtract; then we want to use If and possibly Else. In its simplest form the If statement contains the “If” keyword followed logical expression like we would use in a where clause and finally a statement to execute if the logical expression is true.

```
IF monthly_amount < 0  
    SELECT @remove_flag = 'Y'
```

In this example if the monthly\_amount is less than zero, the remove\_flag is set to equal Y. But what if we want to do more than one thing if the test is true? We use a code block. Code blocks are created by placing one or more SQL statements between the “Begin” and “End” keywords.

```
IF monthly_amount < 0  
    BEGIN  
        SELECT @remove_flag = 'Y'  
        SELECT @credit_amount = monthly_amount  
    END
```

In the new example if the `monthly_amount` is less than zero, the `remove_flag` is set to equal Y and the `credit_amount` is set to the `monthly_amount`. What is the value of `remove_flag` if the test is false? What it was before if it has already been used or null if it has not been used. Using the “Else” keyword we can execute statements when the test is false also.

```
IF monthly_amount < 0
    BEGIN
        SELECT @remove_flag = 'Y'
        SELECT @credit_amount = monthly_amount
    END
ELSE
    BEGIN
        SELECT @remove_flag = 'N'
        SELECT @invoice_amount = monthly_amount
    END
```

## While

While loops will perform a task repeatedly as long as the logical expression following the “While” keyword is true. Again, the task can be a single statement or a code block. This example finds the last day of the previous month.

```
set @enddate = @middledate
While DATEPART(day, @enddate) <= DATEPART(day, @middledate)
    Begin
        SELECT @enddate = DATEADD(day, -1, @enddate)
    End
```

This example uses “DATEPART” to get the day of the month for a given date. Also note that a code block can consist of a single statement. This is often done to make the code easier to read. In the above example we set the value of the @enddate equal to the value of @middledate to make sure the loop starts. If the logical expression starts as false, the loop will not be executed at all.

### Case

Case statements allow you to execute different statements based on logical expressions. The case statement acts similar to a series of If statements except in a Case statement only the first true logical expression is executed where as in a series of If statements all of the statements with true logical expressions would execute. This example sets the plus or minus value of the GL\_Register.Amount based on the value of the GL\_Register.Credit\_Or\_Debit value.

```
Select Sum(Amount *  
    Case When Credit_Or_Debit ='C' Then -1  
    When Credit_Or_Debit ='D' Then 1  
    ELSE 1  
    End)  
From GL_Register
```

Each When/Then pair returns a value and finally if for some reason the Credit\_Or\_Debit field contains some other than a C or D, the optional Else part sets a default value. Notice the End statement. This is required for Case statements.

## Virtual tables

Wouldn't it be great if we could create tables with just the information we wanted and then use them in queries? With virtual tables and views we can do exactly that.

Virtual tables start as a normal select query. They have Select, From and Where clauses. They can also contain Group By, Unions and Sub Queries. Once you have the records being returned how you want, place parenthesis “()” around the query. After the closing parenthesis give the virtual table a name or alias. Here is a simple example:

```
(Select
AR_Customer.Customer_Number As 'CustNum',
AR_Customer.Customer_Id as 'CustId',
AR_customer_Bill.Business_Name + AR_Customer_Bill.Commercial As
'Bill_Postal_Name',
AR_customer_Bill.Address_1 As 'Bill_Address_1',
AR_customer_Bill.Address_2 As 'Bill_Address_2',
AR_customer_Bill.GE1_Description As 'Bill_City',
AR_customer_Bill.GE2_Short As 'Bill_State_Abbreviation',
AR_customer_Bill.GE3_Description As 'Bill_Postal_Code',
AR_Customer_Bill.Zip_Code_Plus4 As 'Bill_Zip_Plus4'
From
AR_Customer
Inner JOIN AR_Customer_Bill On AR_Customer.Customer_Id =
AR_Customer_Bill.Customer_Id
Inner JOIN SS_Customer_Status On AR_Customer.Customer_Status_Id =
SS_Customer_Status.Customer_Status_Id
Where
AR_Customer.Customer_Id <> 1 And
Customer_Status_Code = 'AR') MailAddr
```

This will return all active customer Bill To addresses. To see a field in our virtual table we would use the alias followed by the name we assigned to the field. IE. MailAddr.CustNum.

We can join virtual tables just like regular tables, see the example below:

Select

MailAddr.CustNum,

i.Invoice\_Number

From AR\_Invoice i

inner join

(Select

AR\_Customer.Customer\_Number As 'CustNum',

AR\_Customer.Customer\_Id as 'CustId',

AR\_customer\_Bill.Business\_Name + AR\_Customer\_Bill.Commercial As  
'Bill\_Postal\_Name',

AR\_customer\_Bill.Address\_1 As 'Bill\_Address\_1',

AR\_customer\_Bill.Address\_2 As 'Bill\_Address\_2',

AR\_customer\_Bill.GE1\_Description As 'Bill\_City',

AR\_customer\_Bill.GE2\_Short As 'Bill\_State\_Abbreviation',

AR\_customer\_Bill.GE3\_Description As 'Bill\_Postal\_Code',

AR\_Customer\_Bill.Zip\_Code\_Plus4 As 'Bill\_Zip\_Plus4'

From

AR\_Customer

Inner JOIN AR\_Customer\_Bill On AR\_Customer.Customer\_Id =  
AR\_Customer\_Bill.Customer\_Id

Inner JOIN SS\_Customer\_Status On AR\_Customer.Customer\_Status\_Id =  
SS\_Customer\_Status.Customer\_Status\_Id

Where

AR\_Customer.Customer\_Id <> 1 And

Customer\_Status\_Code = 'AR') MailAddr

on i.Customer\_Id = MailAddr.CustId



We can even join two virtual tables to create a complex query:

```
Select
MailAddr.CustNum,
MailAddr.Bill_City,
inv.Amount,
inv.Net_Due,
inv.PastDue
From
(Select
i.Customer_Id as 'CustId',
i.Invoice_Date as 'Date',
DATEADD(d,t.Days_Net_Due,i.Invoice_Date) as 'Due_Date',
i.Amount as 'Amount',
i.Net_Due,
DATEDIFF(d,DATEADD(d,t.Days_Net_Due,i.Invoice_Date),GETDATE()) as 'PastDue'
From AR_Invoice i
Inner Join AR_Term t on i.Term_Id = t.Term_Id
Where
DATEDIFF(d,DATEADD(d,t.Days_Net_Due,i.Invoice_Date),GETDATE()) > 10
and i.Net_Due > 0) inv
inner join
(Select
AR_Customer.Customer_Number As 'CustNum',
AR_Customer.Customer_Id as 'CustId',
AR_customer_Bill.Business_Name + AR_Customer_Bill.Commercial As
'Bill_Postal_Name',
AR_customer_Bill.Address_1 As 'Bill_Address_1',
AR_customer_Bill.Address_2 As 'Bill_Address_2',
AR_customer_Bill.GE1_Description As 'Bill_City',
AR_customer_Bill.GE2_Short As 'Bill_State_Abbreviation',
```

```
AR_Customer_Bill.GE3_Description As 'Bill_Postal_Code',
AR_Customer_Bill.Zip_Code_Plus4 As 'Bill_Zip_Plus4'

From

AR_Customer

Inner JOIN AR_Customer_Bill On AR_Customer.Customer_Id =
AR_Customer_Bill.Customer_Id

Inner JOIN SS_Customer_Status On AR_Customer.Customer_Status_Id =
SS_Customer_Status.Customer_Status_Id

Where

AR_Customer.Customer_Id <> 1 And
Customer_Status_Code = 'AR') MailAddr

on inv.CustId = MailAddr.CustId
```

Here we have created two virtual tables. The first is called inv and contains information about invoices. The second virtual table, MailAddr, contains mailing information. Now this may seem like a lot of work to accomplish the same thing as a normal query would do. But there are times when virtual tables are absolutely required. For example, Using Top, If, Case or When in an aggregate query (Group By) is not allowed. If you needed to use this combination you would need to first create a virtual table containing the Top, If, Case or When. Then using the virtual table you could create your Aggregate query.

---

## Even More Advanced SQL

Up until now, everything we've done only looks at the database. Now we are going to look at some commands that will save your queries for later use. Also Excel with Microsoft query has some serious limitations, especially with outer joins. By using views and Select Into you can create structures that place a wrapper around the data returned from your favorite complex query so that Excel thinks it is dealing with just one table.

**These commands can damage your database if not done correctly. ALWAYS do a back up before running these commands.**

### Views

What if you have a query you use quite often, like our MailAddr query above. Wouldn't it be nice to have it always available without having to type it in each time? Views allow you to do that. A view is a query that is precompiled by the SQL Server and stored under a name you give it. SedonaOffice has created a number of Views for your use. They are listed in the dbExplorer on the Views tab. But you can make your own also. To create a view you use the Create View command:

```
Create View [dbo].[MailingAddr]
as
Select
AR_Customer.Customer_Number As 'CustNum',
AR_Customer.Customer_Id as 'CustId',
AR_customer_Bill.Business_Name + AR_Customer_Bill.Commercial As
'Bill_Postal_Name',
AR_customer_Bill.Address_1 As 'Bill_Address_1',
AR_customer_Bill.Address_2 As 'Bill_Address_2',
AR_customer_Bill.GE1_Description As 'Bill_City',
AR_customer_Bill.GE2_Short As 'Bill_State_Abbreviation',
AR_customer_Bill.GE3_Description As 'Bill_Postal_Code',
AR_Customer_Bill.Zip_Code_Plus4 As 'Bill_Zip_Plus4'
From
```

```
AR_Customer
```

```
Inner JOIN AR_Customer_Bill On AR_Customer.Customer_Id =  
AR_Customer_Bill.Customer_Id
```

```
Inner JOIN SS_Customer_Status On AR_Customer.Customer_Status_Id =  
SS_Customer_Status.Customer_Status_Id
```

```
Where
```

```
AR_Customer.Customer_Id <> 1 And
```

```
Customer_Status_Code = 'AR'
```

This creates a view named MailingAddr. It can be used just like any table. There are a few considerations though.

- The [dbo] insures that the view will e available to all valid SQL users.
- You can not create a view named the same as an existing view. You must “Drop” the other view first.
- Do not use any name already in use by SedonaOffice. We will over-write it during the next update. We recommend you use your name or your company name as part of the view’s name. IE. Matt\_Mail\_Addr or Acme\_Mail\_Addr.
- Rarely, because it is precompiled, a view may not work the same as the query it is based on. Always check it before using it for anything serious.
- Views can not contain an Order By clause. This is not as bad as it seems because when you use a query, you can add the Order By there. IE.

```
Select * from MailingAddr order by CustNum
```

To over write an existing view, add the four lines below to the Create View script. This will Drop the existing view first before the Create script.

```
IF EXISTS (SELECT * FROM sys.views WHERE object_id =  
OBJECT_ID(N'[dbo].[MailingAddr]'))
```

```
DROP VIEW [dbo].[MailingAddr]
```

```
GO
```

```
Create View [dbo].[MailingAddr]
```

```
as
```

**Select**

```
AR_Customer.Customer_Number As 'CustNum',
AR_Customer.Customer_Id as 'CustId',
AR_customer_Bill.Business_Name + AR_Customer_Bill.Commercial As
'Bill_Postal_Name',
AR_customer_Bill.Address_1 As 'Bill_Address_1',
AR_customer_Bill.Address_2 As 'Bill_Address_2',
AR_customer_Bill.GE1_Description As 'Bill_City',
AR_customer_Bill.GE2_Short As 'Bill_State_Abbreviation',
AR_customer_Bill.GE3_Description As 'Bill_Postal_Code',
AR_Customer_Bill.Zip_Code_Plus4 As 'Bill_Zip_Plus4'
```

**From**

```
AR_Customer
Inner JOIN AR_Customer_Bill On AR_Customer.Customer_Id =
AR_Customer_Bill.Customer_Id
Inner JOIN SS_Customer_Status On AR_Customer.Customer_Status_Id =
SS_Customer_Status.Customer_Status_Id
```

**Where**

```
AR_Customer.Customer_Id <> 1 And
Customer_Status_Code = 'AR'
```

## Select Into

What if we create our masterpiece query with everything we need at the end of the month but it is going to take a week to analyze everything in it. We could just have everyone stop working till we are done but I doubt that is practical. So we can use Select Into to create a table that is a snapshot of our query at that instant. So a View is constantly changing as the data changes. But the table created by a Select Into is static as of the time it was created.

To create our snapshot table we merely add the line “into *TableName*” right before the From clause.

### Select

```
tg.Taxing_Group_Code,  
tg.Description as 'Group Desc',  
tt.Tax_Table_Code,  
tt.Description as 'Tax Desc',  
tt.Tax_Rate,  
tt.Collect_Tax  
into TestTable  
From AR_Taxing_Group tg  
inner join AR_Taxing_Group_Taxes tgt on tg.Taxing_Group_Id = tgt.Taxing_Group_Id  
inner join AR_Tax_Table tt on tgt.Tax_Table_Id = tt.Tax_Table_Id  
Order by tg.Taxing_Group_Code, tt.Tax_Table_Code
```

After we run this we can examine the contents anytime we want with select queries like:

```
select * from TestTable
```

Now next month you are going to want to run this again. (Or anytime you want a refresh.) You can't run the same Select into query again as it will error and tell you the table already exists. You either need to change the “Into” table name or drop the table first.

```
DROP Table [dbo].[TestTable]
```

Or you could add the drop command to the beginning of your Select Into command.

```
IF EXISTS (SELECT * FROM sys.tables WHERE object_id = OBJECT_ID(N'[dbo].[TestTable]'))
DROP TABLE [dbo].[TestTable]
GO
```

Select

```
tg.Taxing_Group_Code,
tg.Description as 'Group Desc',
tt.Tax_Table_Code,
tt.Description as 'Tax Desc',
tt.Tax_Rate,
tt.Collect_Tax
into TestTable
From AR_Taxing_Group tg
inner join AR_Taxing_Group_Taxes tgt on tg.Taxing_Group_Id = tgt.Taxing_Group_Id
inner join AR_Tax_Table tt on tgt.Tax_Table_Id = tt.Tax_Table_Id
Order by tg.Taxing_Group_Code, tt.Tax_Table_Code
Go
```

If you do a manual table drop be very very careful. If you drop the wrong table you can only restore it by loading a back up. So remember to do a back up before you drop a table.

## Sample queries

Get all customers whose annual is between \$239.00 and \$245.00:

Select

```
c.Customer_Number,  
b.Business_Name,  
  
(Select Sum(r.Monthly_Amount*12) From AR_Customer_Recurring r where  
r.Cycle_Start_Date <= GETDATE() And (r.Cycle_End_Date <= {d'1900-01-01'} Or  
r.Cycle_End_Date > GETDATE()))  
And r.Customer_Id = c.Customer_Id) as Annual  
  
From AR_Customer c  
  
Inner Join AR_Customer_Bill b on c.Customer_Id = b.Customer_Id  
  
Where b.Is_Primary = 'Y' And (Select Sum(r.Monthly_Amount*12) From  
AR_Customer_Recurring r where  
r.Cycle_Start_Date <= GETDATE() And (r.Cycle_End_Date <= {d'1900-01-01'} Or  
r.Cycle_End_Date > GETDATE()))  
And r.Customer_Id = c.Customer_Id) Between 239.00 and 245.00
```

Get a range of service appointments and dispatch times:

```
SELECT t.Ticket_Number,  
d.Schedule_Time,  
d.Dispatch_Time,  
e.Employee_Code,  
c.Customer_Number,  
c.Customer_Name  
  
FROM SV_Service_Ticket t  
  
INNER JOIN SV_Service_Ticket_Dispatch d ON t.Service_Ticket_Id =  
d.Service_Ticket_Id  
  
INNER JOIN SV_Service_Tech tech ON d.Service_Tech_Id = tech.Service_Tech_Id
```



```
INNER JOIN SY_Employee e ON tech.Employee_Id = e.Employee_Id
INNER JOIN AR_Customer c ON t.Customer_ID = c.Customer_ID
WHERE d.Schedule_Time >= {d'2013-01-01'} AND d.Schedule_Time < {d'2013-01-31'}
ORDER BY d.Schedule_Time
```

Get how much they paid last year in monitoring, service and installations:

```
Select
c.Customer_Number,
b.Business_Name,
(select IsNull(SUM(i.Amount), 0) From AR_Invoice i
Where i.Type_JSCO = 'C' And i.Invoice_Date Between {d'2012-01-01'} And {d'2012-12-31'}
And i.Customer_Id = c.Customer_Id) as 'Monitoring',
(select IsNull(SUM(i.Amount), 0) From AR_Invoice i
Where i.Type_JSCO = 'S' And i.Invoice_Date Between {d'2012-01-01'} And {d'2012-12-31'}
And i.Customer_Id = c.Customer_Id) as 'Service',
(select IsNull(SUM(i.Amount), 0) From AR_Invoice i
Where i.Type_JSCO = 'J' And i.Invoice_Date Between {d'2012-01-01'} And {d'2012-12-31'}
And i.Customer_Id = c.Customer_Id) as 'Installs',
(select IsNull(SUM(i.Amount), 0) From AR_Invoice i
Where i.Type_JSCO = 'O' And i.Invoice_Date Between {d'2012-01-01'} And {d'2012-12-31'}
And i.Customer_Id = c.Customer_Id) as 'Other'
From AR_Customer c
Inner Join AR_Customer_Bill b on b.Customer_Id = c.Customer_Id
```

Get a detailed list of cancelled accounts for sales to do follow up calls:

**Select**

```
cu.Customer_Number,  
cu.Customer_Name,  
st.Customer_Status_Code,  
cb.Branch_Code as Customer_Branch,  
ty.Type_Code,  
cs.Business_Name,  
cs.Address_1,  
cs.Address_2,  
cs.GE1_Description,  
cs.GE2_Short,  
cs.GE3_Description,  
cs.Zip_Code_Plus4,  
sb.Branch_Code as Site_Branch,  
ts.System_Code,  
pt.Panel_Type_Code,  
cq.CS_Cancelled_Date,  
cu.Customer_Since,  
cq.Effective_Date,  
cq.Reference,  
cq.Memo,  
it.Item_Code,  
cr.Monthly_Amount,  
cq.Balance_Of_Contract,  
cq.Full_Cancel
```

```
From AR_Customer cu
```

```
Inner Join AR_Type_Of_Customer ty On cu.Customer_Type_Id = ty.Type_Id
```

```
Inner Join SS_Customer_Status st On cu.Customer_Status_Id =  
st.Customer_Status_Id
```

```
Inner Join AR_Customer_Site cs On cu.Customer_Id = cs.Customer_Id
```

```
Inner Join AR_Branch cb On cu.Branch_Id = cb.Branch_Id
```



```
Inner Join AR_Branch sb On cs.Branch_Id = sb.Branch_Id
Inner Join AR_Customer_System sy On cs.Customer_Site_Id = sy.Customer_Site_Id
Inner Join SY_System ts On sy.System_Id = ts.System_Id
Inner Join SY_Panel_Type pt On sy.Panel_Type_Id = pt.Panel_Type_Id
Inner Join AR_Customer_Recurring cr On sy.Customer_System_Id =
cr.Customer_System_Id
Inner Join AR_Item it on cr.Item_Id = it.Item_Id
Inner Join AR_Cancel_Queue cq On cu.Customer_Id = cq.Customer_Id
Inner Join AR_Cancel_Queue_Site qs On cq.Cancel_Queue_Id = qs.Cancel_Queue_Id
Where cr.Cycle_End_Date = cq.Effective_Date And cs.Customer_Site_Id =
qs.Customer_Site_Id
And cq.Effective_Date Between {d'2012-01-01'} And {d'2012-12-31'}
Order By cu.Customer_Number
```