SedonaOffice

13TH ANNUAL USERS CONFERENCE

January 12-14, 2015 | Coronado Bay, CA

Data Mining 2

Presented By:

Matt Howe

Table of Contents

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 Quantity * Rate to get the extended price. Another example would be GE3_Description + '-' + 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 GE3_Description, '-' and 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 Quantity * Rate As Extended_Price. Notice the underscore in 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 Business_Name As 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 GE2_Short from my 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	4	2,2
3	6	3,
4	8	4,4
5	10	5,
6	12	6,6
		,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.

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 bused 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 then 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 se 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

numeric

decimal

int

money

Approximate Numerics

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 for 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 and Views

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

Virtual tables start as a normal select query. They have Select, From and Where clauses. The 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.

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.

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'
```

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
```