SedonaOffice Users Conference

San Francisco, CA | January 21 – 24, 2018

Data Mining Part 1

Presented by:
Matt Howe

PERENNIAL SOFTWARE

This Page Intentionally Left Blank

Table of Contents

Overview

In these two sessions, we are going to learn to write queries to extract data from a SedonaOffice Database. We will be using SQL or Structured Query Language. In order to use what you have learned from this session you will need access to the SQL Management Studio. There is a copy that gets installed with the SQL Server install on the SQL Server. However, additional copies can be installed on client computers and linked to your SQL Server. We will not be covering the installation of additional client as it varies from version to version and is more of an IT function that Data Mining.

Note: All SQL keywords, queries and code snippets will be in *italics* to make them stand out from the commentary.

Basic Queries

First, we will start with a few basics. Each SedonaOffice company is stored in a separate SQL database. These databases are made of a number of different items but for now, we are interested in the tables. Tables contain fields and the fields are where the data is actually stored. These fields are arranged in records or rows. Their structure is the same for all records in a given table. Besides the data stored in fields there are fields that are used as pointers to other tables and their records.

SedonaOffice has a basic structure in table and field naming to assist in writing queries. Most of the tables (All of the ones we will be working with) start with a two-letter code and an underscore. These two letter code designations help to identify the area of SedonaOffice to which the table name pertains. Here is a list of some of the most common codes:

Two Letter Code	Description
AP	Accounts Payable
AR	Accounts Receivable
CS	Central Station
GE	Geographic
GL	General Ledger
IN	Inventory
OE	Jobs
SM	Sales Management
SS	SedonaOffice Settings (Usually you should not change these)
SV	Service
SY	Your Settings

These codes are then followed by the rest of the table name, i.e. AR_Customer. In nearly every table, the first field is the record identity. These are number fields that are assigned by the SQL Server and should not ever be changed. In fact, the SQL server makes it extremely difficult to change one. These identity fields also have a naming structure. The main part of the table name followed by an underscore and the letters ID. So, the identity field for the table AR_Customer would be Customer_ID. There can be other pointers in a record that are not an identity but point to or join to other tables. IE. AR_Customer contains a pointer to the SS_Customer_Status table called Customer_Status_ID. Notice this is identical to the identity of the SS_Customer_Status table. We will discuss this further as we build queries.

Select

Data mining queries start with the *select* keyword. Then a list of fields to display and the tables from which to get the fields. Or first query is:

*Select * from AR_Customer*

We have the *select* keyword followed by a wildcard "*" this tells the SQL Server to return all fields. This is followed by our next keyword *from*. This tells the SQL Server we are done listing fields and are going to list the tables that the fields come from. We can also list the specific fields we want replacing the "*" with a comma separated list of the fields we want.

Select Customer_Number, Customer_ID from AR_Customer

Notes

Joins

A join is how we include multiple tables in the *from* clause.There are four types of joins inner, left outer, right outer and full outer. Inner joins only include rows where the records are matched in both tables. Left outer joins show all the records in the left table but only matching records from the right table. As you would expect, a right outer join is the reverse of a left outer join showing all records from the right table and only matching records from the left table. Finally, a full outer join includes all of the records from both tables. Below is a chart of the results of different joins based off of:

From table1
Join table2 on table1.link = table2.link

Where the “link” is one of the previously mentioned pointers.

Table 1	Table 2	Inner	Left outer	Right outer	Full outer
A	B	B,B	A,	B,B	A,
B	C	C,C	B,B	C,C	B,B
C	D		C,C	,D	C,C
					,D

Notes

So to add a join to our previous query, we get”

Select Customer_Number, Customer_Status_Code
from AR_Customer
inner join SS_Customer_Status on AR_Customer.Customer_Status_ID =
SS_Customer_Status.Customer_Status_ID

Notice in the join we wrote out table_name.field_name. Anytime a field with the same name exists in two different tables with in a query, you must specify the table name. With some pretty long table names IE. CS_Onboarding_Test_Received_Status and long field names IE. Refresh_Empty_Message_Text_Before_Max_Hours, joins could become very long. However, there is a short cut.

Select Customer_Number, Customer_Status_Code
From AR_Customer c
Inner join SS_Customer_Status s on c.Customer_Status_ID = s.Customer_Status_ID

Notice the small c following AR_Customer and the small s following SS_CustomerStatus. These are called aliases and once set replace the full table name. Aliases can be any combination of letters, numbers, underscores and dashes but they must start with a letter.

Notes

Where

The final basic we are going to look at is how to filter our returned list to bring only the records we want to see. The query we have been working on brings back a list of all customers and their status. What if we only wanted to see the cancelled customers? We would use a *where* clause. The *where* key word starts the clause which is then followed by a series of conditions or tests. Each of these tests need to be a true or false test. Things like

Amount = 0
Count >10
Name = 'John'

So, adding a *where* clause to our query gives us.

```
Select Customer_Number, Customer_Status_Code
From AR_Customer c
Inner join SS_Customer_Status s on c.Customer_Status_ID = s.Customer_Status_ID
Where s.Customer_Status_Code = 'CANC'
```

This will only return records of cancelled customers.

Tests are connect by logical operators. Part 1 here we are only going to look at two logical operators, *And* and *Or.* If we have two tests amount = 10 and count = 5, and we connect them with an *And* the *Where* clause would look like this.

```
Where amount = 10 And count = 5
```

Both conditions would need to be true in order for that record to be displayed. If our clause used an *Or* it would look like this.

```
Where amount = 10 Or count = 5
```

If either test or both tests are true the record will be returned.

There is one last thing about logical operators before we create something useful.

What does 3 + 2 * 4 = ? 11 or 20? Well the rules of math say 11 because you multiply before you add so 2 * 4 is 8 plus 3 is 11. Simple!

Let us say we have two branches called Acme and Solo and we have customer in the cities Pontiac, Flint and Detroit. If we needed to get a list of customers in the Acme branch customers in Flint and Detroit we could do something like.

```
Where Branch = 'Acme' And City = 'Flint' Or City = 'Detroit'
```

However, when we ran it we would get Acme customers from Flint but every customer in Detroit. Why? Because SQL does the *And* before the *Or*. Not because it comes first but because like math does multiply before add, logic does And before *Or.* And like math where if we wanted the 20 not the 11 we would write it like (2 + 3) * 4, we could write our query like.

Where Branch = 'Acme' And (City = 'Flint' Or City = 'Detroit')

So, a quick recap of the rules for *Where* Clauses.

- All test must be able to be evaluated as True or False
- And is evaluated before Or
- If there are parenthesis what is in the parenthesis is evaluated first
- Finally, all test and evaluated parenthesis are evaluated from left to right

Notes

Example Query

Let us create a query to show every service call run by a particular tech for the last 30 days.

We will start by deciding what we want to display.

Customer number
Customer name
Site name
Site address with city, state and zip code
System account
System type
Arrival time
Departure time
Call resolution

From this we can get a list of tables.

AR_Customer
AR_Customer_Bill
AR_Customer_Site
AR_Customer_System
SS_System
SV_Service_Ticket
SV_Service_Ticket_Dispatch
SV_Resolution
SV_Service_Tech
SY_Employee

Notes

From

We will start by creating a simple bring back everything query using the * operator so we can get our *From* clause right.

```
select
*
from AR_Customer c
inner join AR_Customer_Bill b on c.Customer_Id = b.Customer_Id and b.Is_Primary =
'Y'
inner join AR_Customer_Site si on c.Customer_Id = si.Customer_Id
inner join AR_Customer_System csy on si.Customer_Site_Id = csy.Customer_Site_Id
inner join SY_System sy on csy.System_Id = sy.System_Id
inner join SV_Service_Ticket st on csy.Customer_System_Id = st.Customer_System_Id
inner join SV_Service_Ticket_Dispatch std on st.Service_Ticket_Id =
std.Service_Ticket_Id
inner join SV_Service_Tech stt on std.Service_Tech_Id = stt.Service_Tech_Id
inner join SV_Resolution r on std.Resolution_Id = r.Resolution_Id
inner join SY_Employee e on stt.Employee_Id = e.Employee_Id
```

A few comments here, first because we used all inner joins, where records must be present in both tables we have eliminated all customers that have never had a service tech at their location. Also notice the *b.Is_Primary = 'Y'*.in the join. This insures we only get the primary bill to. This is a handy extension to the join clause to be able to place some logic tests in the join, but even with these two provisions, our record set contains over 50,000 records and hundreds of fields we do not want. So let us add the *select* clause next.

Notes

Select

```
select
c.Customer_Number,
b.Business_Name,
si.Business_Name,
si.Address_1, si.GE1_Description, si.GE2_Description, si.GE3_Description,
csy.Alarm_Account,
sy.Description as 'System Type',
std.Arrival_Time,
std.Departure_Time,
r.Description
from AR_Customer c
inner join AR_Customer_Bill b on c.Customer_Id = b.Customer_Id and b.Is_Primary =
'Y'
inner join AR_Customer_Site si on c.Customer_Id = si.Customer_Id
inner join AR_Customer_System csy on si.Customer_Site_Id = csy.Customer_Site_Id
inner join SY_System sy on csy.System_Id = sy.System_Id
inner join SV_Service_Ticket st on csy.Customer_System_Id = st.Customer_System_Id
inner join SV_Service_Ticket_Dispatch std on st.Service_Ticket_Id =
std.Service_Ticket_Id
inner join SV_Service_Tech stt on std.Service_Tech_Id = stt.Service_Tech_Id
inner join SV_Resolution r on std.Resolution_Id = r.Resolution_Id
inner join SY_Employee e on stt.Employee_Id = e.Employee_Id
```

First, notice each field is appended to the table aliases we set up. It makes the code much easier to read and write. Next we added and optional *as* clause and a name to the *sy.Descripition* field. This appears in the column header and again is added to make it easier to read. Again, we got over 55,000 records but it came back much more quickly without the extra fields.

Notes

Where

Finally we are going to add our *Where* clause.

```
select
c.Customer_Number,
b.Business_Name,
si.Business_Name,
si.Address_1, si.GE1_Description, si.GE2_Description, si.GE3_Description,
csy.Alarm_Account,
sy.Description as 'System Type',
std.Arrival_Time,
std.Departure_Time,
r.Description
from AR_Customer c
inner join AR_Customer_Bill b on c.Customer_Id = b.Customer_Id and b.Is_Primary =
'Y'
inner join AR_Customer_Site si on c.Customer_Id = si.Customer_Id
inner join AR_Customer_System csy on si.Customer_Site_Id = csy.Customer_Site_Id
inner join SY_System sy on csy.System_Id = sy.System_Id
inner join SV_Service_Ticket st on csy.Customer_System_Id = st.Customer_System_Id
inner join SV_Service_Ticket_Dispatch std on st.Service_Ticket_Id =
std.Service_Ticket_Id
inner join SV_Service_Tech stt on std.Service_Tech_Id = stt.Service_Tech_Id
inner join SV_Resolution r on std.Resolution_Id = r.Resolution_Id
inner join SY_Employee e on stt.Employee_Id = e.Employee_Id
Where std.Arrival_Time > {d'2016-06-01'} and std.Arrival_Time < {d'2016-07-01'}
and e.Employee_Code = 'Ralph'
```

We added something new in the date part of the where clause. First we used a special notation for the date. The date I chose was 6/1/2016 in the US, 1/6/2016 in Canada and most of Europe. Using the {d' notation followed always by 4 digit year, 2 digit month and 2 digit date allows this same query to work in the US, Canada, Europe, everywhere. Also, because Arrival time is a date time field, it could look like 2016-06-05 16:04:32.002. A lot of typing so I used date starting with any date after 2016-06-01 and before 2016-07-01 that way I do not have to type out the entire time portion. This returns 90 rows. Not a bad showing for Ralph.

Notes