Today’s Agenda

• Finding Data
• Report Development Standards
• Writing Efficient Code
• T-SQL Tips
• Report Development
Finding Data

• If you know the NPR structure then finding data will be much easier

• In general you can think of a Detail Segment as a table

Tools

1. Meditech Website
2. SysDrTables/SysDrColumns
3. Shift F9 and Shift F8 for Magic
4. DR Application Menu
Overview of the data repository

* Table Structure

Comparing NPR to M-AT to SQL

<table>
<thead>
<tr>
<th>NPR Components</th>
<th>M-AT Components</th>
<th>SQL Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPM</td>
<td>Object</td>
<td>Tables</td>
</tr>
<tr>
<td>Segment</td>
<td>Record</td>
<td>Table</td>
</tr>
<tr>
<td>Element</td>
<td>Field</td>
<td>Column</td>
</tr>
</tbody>
</table>

Meditech Website
1. Shows the equivalent NPR – Parent/Child relationships
2. Interactive Primary Keys that displays other tables with foreign keys
# Meditech 6.0 – Data Model

<table>
<thead>
<tr>
<th>Application</th>
<th>OM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table Name</td>
<td>System</td>
</tr>
<tr>
<td>OmAccess_AomProcessFunctions</td>
<td>Primary Keys</td>
</tr>
<tr>
<td>OmAccess_ClinicalDataFunctions</td>
<td>SourceID, OmAccessID</td>
</tr>
<tr>
<td>OmAccess_Main</td>
<td>SourceID, OmAccessID</td>
</tr>
<tr>
<td>OmAccess_ProcessFunctions</td>
<td>SourceID, OmAccessID</td>
</tr>
<tr>
<td>OmAccess_AmbOrderPrintForms</td>
<td>SourceID, OmAccessID, AmbulatoryOrderPrintForm_OmFormatDictID</td>
</tr>
<tr>
<td>OmAccess_AomCatGrpSortOrder</td>
<td>SourceID, OmAccessID, AomCategoryGroup_OmGrpID</td>
</tr>
<tr>
<td>OmAccess_AomControlSchedules</td>
<td>SourceID, OmAccessID, AomControlScheduleID</td>
</tr>
<tr>
<td>OmAccess_AomForms</td>
<td>SourceID, OmAccessID, AomFormID</td>
</tr>
<tr>
<td>OmAccess_HomeMedicationForms</td>
<td>SourceID, OmAccessID, HomeMedicationForm_OmFormatDictID</td>
</tr>
<tr>
<td>OmAccess_Identifiers</td>
<td>SourceID, OmAccessID, IdentifierTypeID, IdentifierID</td>
</tr>
<tr>
<td>OmAccess_OmAckCategories</td>
<td>SourceID, OmAccessID</td>
</tr>
<tr>
<td>OmAccess_OmAckMedTypes</td>
<td>SourceID, OmAccessID</td>
</tr>
<tr>
<td>OmAccess_OmCatGrpSortOrder</td>
<td>SourceID, OmAccessID</td>
</tr>
<tr>
<td>OmAccess_OmReviewCategories</td>
<td>SourceID, OmAccessID</td>
</tr>
<tr>
<td>OmAccess_OmReviewMedTypes</td>
<td>SourceID, OmAccessID</td>
</tr>
<tr>
<td>OmCat_Main</td>
<td>Application: OM</td>
</tr>
<tr>
<td>OmCat_Facilities</td>
<td>Column Name</td>
</tr>
<tr>
<td>OmCat_ConnectionOrderRules</td>
<td>SourceID</td>
</tr>
<tr>
<td>OmCat_ID</td>
<td>OmCatID</td>
</tr>
<tr>
<td>RowUpdateDateTime</td>
<td>RowUpdateDateTime</td>
</tr>
<tr>
<td>Mnemonic</td>
<td>Mnemonic</td>
</tr>
<tr>
<td>Active</td>
<td>Active</td>
</tr>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>Group_OmGrpID</td>
<td>Group_OmGrpID</td>
</tr>
<tr>
<td>Type</td>
<td>Type</td>
</tr>
<tr>
<td>ConnectTo_OmConnID</td>
<td>ConnectTo_OmConnID</td>
</tr>
<tr>
<td>ConnectionMnemonic</td>
<td>ConnectionMnemonic</td>
</tr>
<tr>
<td>CategoryLookup</td>
<td>CategoryLookup</td>
</tr>
<tr>
<td>IncludeAsPartOfString</td>
<td>IncludeAsPartOfString</td>
</tr>
<tr>
<td>UsedIn</td>
<td>UsedIn</td>
</tr>
<tr>
<td>AomGroup_OmGrpID</td>
<td>AomGroup_OmGrpID</td>
</tr>
<tr>
<td>AomCategoryLookup</td>
<td>AomCategoryLookup</td>
</tr>
<tr>
<td>AomIncludeAsPartOfString</td>
<td>AomIncludeAsPartOfString</td>
</tr>
<tr>
<td>CopyFromId_OmCatID</td>
<td>CopyFromId_OmCatID</td>
</tr>
</tbody>
</table>
Table Information in livedb and livefdb

**livedb**

---
- A general search by DR Field Name
- '%Comment%' is a wildcard search for any field with Comment
- You can modify the name as needed for your search
---

```sql
SELECT T.Name, C.*
FROM livedb.dbo.SysDrColumns C
INNER JOIN livedb.dbo.SysDrTables T
  ON C.TableID = T.TableID
WHERE C.Name like '%Comment%'
ORDER BY 1
```

---
- A search by specific NPR field
---

```sql
SELECT T.Name, C.*
FROM livedb.dbo.SysDrColumns C
INNER JOIN livedb.dbo.SysDrTables T
  ON C.TableID = T.TableID
WHERE C.NprElement = 'BAR.PAT.account'
ORDER BY 1
```

**livefdb**

---
- A general search by DR Field Name
- '%Comment%' is a wildcard search for any field with Comment
- You can modify the name as needed for your search
---

```sql
SELECT DT_M.TableName, DT_C.*
FROM livefdb.dbo.DrTable_Main DT_M
INNER JOIN livefdb.dbo.DrTable_Columns DT_C
  ON DT_M.SourceID = DT_C.SourceID
  AND DT_M.DrTableID = DT_C.DrTableID
WHERE DT_C.ColumnName like '%Comment%'
ORDER BY 1
```

---
- A search by specific NPR field
---

```sql
SELECT DT_M.TableName, DT_C.*
FROM livefdb.dbo.DrTable_Main DT_M
INNER JOIN livefdb.dbo.DrTable_Columns DT_C
  ON DT_M.SourceID = DT_C.SourceID
  AND DT_M.DrTableID = DT_C.DrTableID
WHERE DT_C.ColumnObjectClass = 'OmOrd'
ORDER BY 1
```
Examples

```
SELECT T.Name, C.*
FROM livedb.dbo.SysDrColumns C
INNER JOIN livedb.dbo.SysDrTables T
ON C.TableID = T.TableID
WHERE C.Name like '%Comment%'
order by 1

SELECT DT_M.TableName, DT_C.*
FROM livefoc.dbo.DrTable_Main DT_M
INNER JOIN livefoc.dbo.DrTable_Columns DT_C
ON DT_M.SourceID = DT_C.SourceID
AND DT_M.DrTableID = DT_C.DrTableID
WHERE DT_C.ColumnName  like '%Comment%'
ORDER BY 1
```

Shows table name, column, data type along with the DPM, NprSegment and NprElement.

Shows table name, column, data type, length along with ObjectClass, Column Record and Column Field.

<table>
<thead>
<tr>
<th>DrTableID</th>
<th>TableName</th>
<th>ColumnName</th>
<th>ColumnObjectClass</th>
<th>ColumnRecord</th>
<th>SortOrder</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCG0000040</td>
<td>DrTableTest_TestDataTypes</td>
<td>(DataType=YnComment)</td>
<td>DrTableTest</td>
<td>TestDataTypes</td>
<td>13</td>
</tr>
<tr>
<td>FCG0000062</td>
<td>DrTableTest_TestKeyedTimeFile</td>
<td>(Comment)</td>
<td>DrTableTest</td>
<td>TestKeyedTimeFile</td>
<td>5</td>
</tr>
<tr>
<td>FCG0000043</td>
<td>DrTableTest_TestTimeFile</td>
<td>(Comment)</td>
<td>DrTableTest</td>
<td>TestTimeFile</td>
<td>5</td>
</tr>
<tr>
<td>FCG000127</td>
<td>EmnAcctm_Mar</td>
<td>(MarScheduleComments)</td>
<td>EmnParam</td>
<td>Mar</td>
<td>44</td>
</tr>
<tr>
<td>FCG000127</td>
<td>EmnParam_Mar</td>
<td>(MarCommentPopUp)</td>
<td>EmnParam</td>
<td>Mar</td>
<td>45</td>
</tr>
<tr>
<td>FCG000127</td>
<td>EmnParam_Mar</td>
<td>(MarCommentRmoveHours)</td>
<td>EmnParam</td>
<td>Mar</td>
<td>46</td>
</tr>
<tr>
<td>FCG000169</td>
<td>EmnStationStatus_Main</td>
<td>(Comment)</td>
<td>EmnStationStatus</td>
<td>Main</td>
<td>5</td>
</tr>
<tr>
<td>FCG000212</td>
<td>EmnAcctm_BloodReactionComments</td>
<td>(BloodReactionComment)</td>
<td>EmnAcctm</td>
<td>BloodReactionComments</td>
<td>10</td>
</tr>
<tr>
<td>FCG000212</td>
<td>EmnAcctm_BloodReactionComments</td>
<td>(BloodReactionComment)</td>
<td>EmnAcctm</td>
<td>BloodReactionComments</td>
<td>13</td>
</tr>
</tbody>
</table>
Shift F9 and Shift F8 for Magic
Likewise, in AT applications, information about the location of a field in the DR can be garnered from the help option.
I rarely use these tools on the DR Menu – They can be helpful but I find the previously covered options to be the best.

Magic 5.66 Screen
Report/SQL Development Standards

- Stored procedures
- Data and Database Organization
- Documentation
1. Don’t save stored procedures and/or tables in live databases.
2. Create a database to keep your stored procedures, views and tables.
   - Recovery Mode is set to simple
   - You can set up the database files similar to livendb
3. You will want to include the database you create in your backup plan.
What is a stored procedure?

A stored procedure is a saved set of code on the SQL Server that allows you to run:

EXEC spBarAccountsByAccountType

Rather than......

SELECT BVFD.AccountType, BV.PostalCode, BV.AccountNumber
FROM livedb.dbo.BarVisits BV
INNER JOIN livedb.dbo.BarVisitFinancialData BVFD ON BV.SourceID = BVFD.SourceID
AND BV.BillingID = BVFD.BillingID
Organize your Stored Procedures

• Name your stored procedures so that you can easily locate them.
  • Ex: spErDepartVolumesByHour
  • Ex: spAdmRevisitsTable

• Re-name stored procs no longer in use.
  • Ex: x_spErDepartVolumesByHour

• Use Header information to describe a stored procedure’s use.

• Only save useable code as a stored procedure.

• Save test code and research code as a text file or label appropriately.
**Stored Procedures**

To save a stored procedure you CREATE PROCEDURE. This saves the stored procedure on the server/database you’ve selected.

```sql
CREATE PROC [dbo].[spErVolumes]
( @Begin datetime, @End datetime )
AS

SELECT *
FROM tbErVolumeResults
WHERE Day_Date between @Begin and CONVERT(DATE, @End)
ORDER BY 1,2
```

Once created – you can Modify and View Dependencies.
Saving code as a text file

**Items to save as a text file**
1. Research queries
2. Testing queries

Default Location – this can be changed
Documentation

• Documenting through out your code!
• Document on any piece of code that is the least bit out of the ordinary. Not only what by why.
• Notate in each step of your code what you are doing.
/*
 * Created by Acmeware, Inc., All Rights Reserved
 * Title: spMU_ED_1
 * Version: 1.0
 * Author: Jamie Gerardo
 * Description: This stored procedure generates ARRA HITSP Quality Measure output for the ED-1
 *
 * Testing Code: EXEC [spMU_ED_1] '02/1/2012','04/30/2012 23:59:59'
 *
 * Revision History:
 * 4/12/11 - Created stored procedure
 * 5/10/11 - Review all ed depart dates
 * 5/20/11 - Add nursing queries for depart date time
 * 6/25/11 - Fix duplicates
 *
 * Questions:
 * 1. Which ED depart date is the most accurate
 * 2. What date to use if null
 */
Example code documentation

-----------------------------
-- Getting all days between Admit and DischargeDateTime
----------------------------------
-- Using this method to include Discharge Date Location as another row easily
-- a row per inpatient day

-- First we are getting only patients with the Diabetes diagn and then looping
-- through to populate Table A with a date for each day the patient is in hosp
IF OBJECT_ID('tempdb.dbo.#Patients')IS NOT NULL
DROP TABLE #Patients

SELECT PD.*
INTO #Patients -- select * from #Patients ORDER BY 2,7
-- SELECT VisitID, COUNT(*) FROM #Patients GROUP BY VisitID HAVING COUNT(*) > 1
FROM #PatDiag PD
WHERE (PD.PrimaryDiag IS NOT NULL
OR PD.SecondaryDiag IS NOT NULL)
AND PD.DiagnosisSeqID = (SELECT MIN(PD1.DiagnosisSeqID)
  FROM #PatDiag PD1
  WHERE PD.SourceID = PD1.SourceID
  AND PD.VisitID = PD1.VisitID)

IF OBJECT_ID('tempdb.dbo.#TableA')IS NOT NULL
DROP TABLE #TableA
Select VisitID, AdmitDateTime, DischargeDateTime, AdmitDateTime as TheDay
Into #TableA -- select * from #TableA order by VisitID, TheDay
From #Patients

Acmeware
Creating Efficiencies

• Indexing and Primary Keys
• Execution Plan
• Joining on Primary Keys
• Filters
• Where Exists
• Functions
Table Structure

- Each MEDITECH table is implemented with one Index – the tables clustered index.
- Additional indexes can be built to increase query efficiency. (and should be)
Table Indexing

• Clustered
  • Every MEDITECH table has a clustered index, which is the physical order of the table by **primary key(s)**. *Never modify or delete*
  • There is only 1 per table

• Non-Clustered
  • A non-clustered index creates a separate 'internal' table that stores only the selected key values of the table in order. Each 'record' in this index contains the key value from one record in the table along with a pointer to either the data record itself or to a value in the clustered index.
What are primary keys?

- Fields (columns) in a table that are special.
- The primary key values make a record unique to the table.
- Every MEDITECH table will have at least two primary keys per table. SourceID is always the first key.
# Common Table indexes

**livedb**

- BarChargeTransactions
  - Ix_ServiceDateTime
  - Ix_TransactionProcedureID
  - Ix_ProcedureChargeDept

- BarVisits
  - Ix_VisitID
  - Ix_AdmitDateTime
  - Ix_ServiceDateTime

- AdmVisits
  - Ix_ServiceDateTime
  - Ix_Status

- BarCollectionTransactions
  - Ix_ReceiptDateTime
  - Ix_InsuranceID

- AdmittingData
  - Ix_AdmitDateTime

- AbstractData
  - Ix_VisitID

- DMisUserStatisticsDetail
  - Ix_AccountNumber (Field4)
  - Ix_UnitNumber (Field3)

**livefdb**

- RegAcctQuery_Results
  - ix_DateTime
  - ix_InstanceID
  - ix_Query_MisQryID

- RegAcct_Main
  - ix_ArrivalDateTime
  - ix_ServiceDateTime
  - ix_AdmitDateTime

- OmOrd_Main
  - ix_SourceID_VisitID
  - ix_OrderDateTime
Creating an Index

Table name: BarVisits
Index name: ixVisitID
Index type: Nonclust

Index key columns:
- Name
- VisitID

Indexes:
- mtpk_visits (Clustered)
- ixVisitID (Non-Unique, Non-Clustered)
Example of Execution Plan

```
SELECT AV.VisitID, AV.LocationID, AV.[Name], AD.AdmitDateTime
FROM livedb.dbo.AdmVisits AV
INNER JOIN livedb.dbo.AdmittingData AD on AV.SourceID=AD.SourceID and AV.VisitID=AD.VisitID
where Status='ADM IN'
```

Display Estimated and Actual Execution Plan
Primary Keys

• Joining on the primary keys will make your report run more efficiently.

• Omitting the primary key could slow down your query and can skew your intended output.

• Each application has a unique identifier (primary key) that will allow you to join to other applications.

• All primary keys will end in either ID or DateTime

• In the M-AT 6.1 release, VisitID is the most commonly used primary to join from one application to another using the _Main tables

• Typically, all primary keys should be addressed in your Query
Primary Key Example

```sql
SELECT AV.VisitID,
    AV.LocationID,
    AV.[Name],
    AD.AdmitDateTime
FROM livedb.dbo.AdmVisits AV WITH (NOLOCK)
INNER JOIN livedb.dbo.AdmittingData AD WITH (NOLOCK)
ON AV.SourceID = AD.SourceID
AND AV.VisitID = AD.VisitID
WHERE Status='ADM IN'
```

Even if there is only one SourceID, you will want to use the Clustered Index for faster processing.
Primary Key Example 2

SELECT AV.VisitID,
    AV.LocationID,
    AV.[Name],
    AD.AdmitDateTime,
    BV.PrimaryInsuranceID,
    BVFD.Balance
FROM livedb.dbo.AdmVisits AV

INNER JOIN livedb.dbo.AdmittingData AD
    ON AV.SourceID = AD.SourceID
    AND AV.VisitID = AD.VisitID

LEFT JOIN livedb.dbo.BarVisits BV
    ON AV.SourceID = BV.SourceID
    AND AV.VisitID = BV.VisitID

LEFT JOIN livedb.dbo.BarVisitFinancialData BVFD
    ON BV.SourceID = BVFD.SourceID
    AND BV.BillingID = BVFD.BillingID

VisitID is in a number of tables but you’ll only want to use it to join to a parent type table - BarVisits, AbstractData, Lab Specimens, SchAppointments

Use the application’s primary key (unique identifier) within the application tables.

- Adm - VisitID
- Bar – BillingID
- Abs – AbstractID
- Oe – OrderID
- Sch – AppointmentID
- Reg - VisitID
In the M-AT 6.+ releases, VisitID is the most commonly used primary to join from one application to another using the _Main tables.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Application</th>
<th>Parent tables (patient data)</th>
<th>PrimaryKey To use within application tables</th>
<th>Foreign Key for joining from other applications</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS_Magic</td>
<td>ADM</td>
<td>AdmVisits</td>
<td>VisitID</td>
<td>VisitID or PatientID</td>
<td></td>
</tr>
<tr>
<td>CS_Magic</td>
<td>BAR</td>
<td>BarVisits</td>
<td>BillingID</td>
<td>VisitID</td>
<td></td>
</tr>
<tr>
<td>CS_Magic</td>
<td>ABS</td>
<td>AbstractData</td>
<td>AbstractID</td>
<td>VisitID</td>
<td></td>
</tr>
<tr>
<td>CS_Magic</td>
<td>LAB</td>
<td>LabRequisitions</td>
<td>RequisitionID</td>
<td>VisitID</td>
<td></td>
</tr>
<tr>
<td>CS_Magic</td>
<td>LAB</td>
<td>LabSpecimens</td>
<td>SpecimenID</td>
<td>VisitID</td>
<td></td>
</tr>
<tr>
<td>CS</td>
<td>ITS</td>
<td>ItsOrders</td>
<td>OrderID</td>
<td>VisitID or OeOrderID</td>
<td></td>
</tr>
<tr>
<td>CS_Magic</td>
<td>OE</td>
<td>OeOrders</td>
<td>OrderID</td>
<td>VisitID</td>
<td></td>
</tr>
<tr>
<td>MAT</td>
<td>OM</td>
<td>OmOrd_Main</td>
<td>OmOrdID</td>
<td>VisitID or PatientID</td>
<td></td>
</tr>
<tr>
<td>CS_Magic</td>
<td>PHA</td>
<td>PhaRx</td>
<td>PrescriptionID</td>
<td>VisitID</td>
<td></td>
</tr>
<tr>
<td>Magic</td>
<td>RAD</td>
<td>RadExams</td>
<td>PatientID</td>
<td>PatientID</td>
<td>* This is one of the exceptions</td>
</tr>
<tr>
<td>CS_Magic</td>
<td>SCH</td>
<td>SchAppointments</td>
<td>AppointmentID</td>
<td>VisitID</td>
<td></td>
</tr>
<tr>
<td>CS_Magic</td>
<td>SCH</td>
<td>SchPatOrCaseMain</td>
<td>CaseID</td>
<td>VisitID or PatientID</td>
<td>* Patient may not have VisitID</td>
</tr>
</tbody>
</table>
You can use this tool but you still need to join on the primary keys. This tool will not automatically do that for you.
WHERE Clause  (filtering your data)

Filter data from the most restrictive to the least restrictive

```
SELECT
    AV.Name,
    AV.AccountNumber,
    AV.UnitNumber AS MedicalRecordNumber,
    AV.LocationName,
    OO.OrderDateTime,
    OO.Category,
    OO.CategoryName,
    OO.OrderedProcedureMnemonic,
    OO.OrderedProcedureName
FROM
    livedb.dbo.AdmVisits AV
INNER JOIN livedb.dbo.OeOrders OO
    ON AV.SourceID = OO.SourceID
    AND AV.VisitID = OO.VisitID
WHERE
    AV.Status = 'ADM IN'
    AND OO.Status NOT IN ('CANC', 'CANCEL', 'CNC', 'UNCOL', 'UNV', 'UNVER')
ORDER BY
    AV.Name,
    OO.OrderDateTime
```
Using EXISTS

EXISTS in your WHERE clause allows you to return data that’s in another table without directly joining to the table.

```
SELECT AV.AccountNumber, AV.LocationID FROM dbo.AdmVisits AV
WHERE EXISTS (SELECT 1 FROM dbo.AbsSpecialCareUnits ASCU
    WHERE AV.SourceID = ASCU.SourceID AND AV.VisitID = ASCU.VisitID
    AND ASCU.LocationID = 'ICU')
AND AV.LocationID <> 'ICU'
```
User Defined Function

What is a User Defined Function?

Functions are subroutines used to encapsulate frequently performed logic. Any code that must perform the logic incorporated in a function can call the function rather than having to repeat all of the function logic.

- **Built-in functions** operate as defined in the Transact-SQL Reference and cannot be modified. The functions can be referenced only in Transact-SQL statements using the syntax defined in the Transact-SQL Reference.
  - Examples AVG, SUM, COUNT, DATEADD, DATEDIFF, NAME, ETC..

- **User-defined functions** allow you to define your own Transact-SQL functions using the CREATE FUNCTION statement. For more information about these built-in functions.
  - This is what we'll looking at today.
FUNCTIONS

Useful Acmeware functions

• fxAge
• fxProperCase
• fxConvertGramsToLbs
• fxMeditechTimeIDToDateTime
• fxIsNumeric
Function - fx.Age

--Created by Acmeware, Inc., All Rights Reserved
--This function returns a computed Age in years between two dates.

CREATE FUNCTION [dbo].[fxAge] (@DOB datetime, @CheckDate datetime)

RETURNS int AS
BEGIN
RETURN DATEDIFF(Year, @DOB, @CheckDate) --
CASE
    WHEN Month(@CheckDate) * 31 + Day(@CheckDate) >= Month(@DOB) * 31 + Day(@DOB) THEN 0
    ELSE 1
END
END

<table>
<thead>
<tr>
<th>Name</th>
<th>BirthDateTime</th>
<th>CalcAge</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLEN, APPLE W</td>
<td>11/14/78</td>
<td>32</td>
</tr>
<tr>
<td>ALLEN, BABY GIRL</td>
<td>05/25/11</td>
<td>0</td>
</tr>
<tr>
<td>KNABEL, ORANGE L</td>
<td>01/11/43</td>
<td>68</td>
</tr>
<tr>
<td>RICHARDSON, RED W</td>
<td>11/20/38</td>
<td>72</td>
</tr>
<tr>
<td>SHORT, LINDA E</td>
<td>02/25/67</td>
<td>44</td>
</tr>
<tr>
<td>THOMAS, BARBARA A</td>
<td>10/10/48</td>
<td>62</td>
</tr>
</tbody>
</table>
Function - fxProperCase

Selecting the data:

```sql
SELECT
  [Name],
  dbo.fxProperCase(Name) AS ProperName,
  ProviderGroupName,
  dbo.fxProperCase(ProviderGroupName)AS ProperGroupName
FROM livedb.dbo.DMisProvider
```

This takes any value and converts it to upper and lower case. Works great for creating consistencies in your reports.

<table>
<thead>
<tr>
<th>Name</th>
<th>ProperName</th>
<th>ProviderGroupName</th>
<th>ProperProviderGroupName</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANDERSON,PATRICK J DO</td>
<td>Anderson,Patrick J Do</td>
<td>OXFORD MEDICAL GROUP</td>
<td>Oxford Medical Group</td>
</tr>
<tr>
<td>ANDERSEN,ROLF L MD</td>
<td>Andersen,Rolf L Md</td>
<td>HEART GROUP</td>
<td>Heart Group</td>
</tr>
<tr>
<td>ANDERSON,THOMAS W MD</td>
<td>Anderson,Thomas W Md</td>
<td>LITTLE FAMILY MEDICINE</td>
<td>Little Family Medicine</td>
</tr>
<tr>
<td>ANDERSEN, WILLIAM K MD</td>
<td>Andersen,William K Md</td>
<td>SMITH SKIN CENTER PC</td>
<td>Smith Skin Center Pc</td>
</tr>
<tr>
<td>ANDREJKO,CONSTANCE</td>
<td>Andrejko,Constance</td>
<td>ONSITE NEONATAL PARTNERS</td>
<td>Onsite Neonatal Partners</td>
</tr>
</tbody>
</table>
Function - fxMeditechTimeIDToDate_time

There are various fields throughout Meditech that are in seconds. This function will calculate the date for you.

```sql
ALTER FUNCTION [dbo].[fxMeditechTimeIDToDate_time] (@TimeID int)
RETURNS DATETIME AS
BEGIN
    RETURN DATEADD(ss, CONVERT(int, @TimeID), '3/1/1980')
END
```
Function - fxIsNumeric

There will be times where you need to ensure that a field strictly has numeric values. Using the System.IsNumeric does not always work.

```sql
-- Created by Acmeware, Inc., All Rights Reserved
ALTER Function [dbo].[fxIsNumeric]
( @StrNumeric varchar(80) )
RETURNS bit
AS
BEGIN
    RETURN
    CASE
        WHEN Len(@StrNumeric) > 18 THEN 0
        WHEN @StrNumeric = '.' THEN 0
        WHEN @StrNumeric = '-' THEN 0
        WHEN @StrNumeric not like '%' -- Check for percent wildcard
            AND Len(@StrNumeric) = 1
            AND Charindex('-', @StrNumeric) = CASE
                WHEN Left(@StrNumeric, 3) = '1,2'
                THEN 1
                ELSE 0
            END
        ELSE 1
        END
END
```

 DECLARE @a varchar(10)
 SET @a = '1,2'

 SELECT IsNumeric(@a)
 SELECT dbo.fxIsNumeric(@a)

-- IF IsNumeric(@a) = 1 SELECT Convert(decimal(11,2), @a)
IF dbo.fxIsNumeric(@a) = 1 SELECT Convert(decimal(11,2), @a)
CTRL + Z

Remember you can always (almost) undo your last command
SQL Tips

- Temp Tables
- Row_Number
- Multiples to a single column
- Dates
- Using WITH (NOLOCK)
What is a temp table?

- There are two types of Temp tables:
  - Active within the same window - #TempTable
  - Active with your connection to the server - ##TempTable

- Temp tables are created on the fly to store data temporarily

- The temp tables are then joined to other SQL tables for further analysis or for calculating aggregates

- To avoid taking up excess space, you typically will not order data being put into a temp table (there are exceptions)

- Temp tables are deleted when the connection to the database is closed (query window is closed) or the table is dropped
  - CAUTION: When querying data, open SQL windows will retain the allocated space being used
Code for Dropping Temp Tables

When using temp tables enter this before each temp table and it will save you a lot of time and hassle with continuously dropping the table.

IF OBJECT_ID('tempdb.dbo.#TableName') IS NOT NULL
DROP TABLE #TableName

SELECT
Fields
INTO #TableName
FROM MyTables

You can also enter the code at the end of your stored procedure or query to make sure the temp table has been dropped.
Using **ROW_NUMBER**

```sql
SELECT C.VisitID, Query_MisQryID, Text, Value, ActivityDateTime
FROM dbo.tbSCIP_Catheters C
WHERE C.ActivityDateTime = (SELECT MIN(C2.ActivityDateTime)
FROM dbo.tbSCIP_Catheters C2
WHERE C.SourceID = C2.SourceID
AND C.VisitID = C2.VisitID)
```

--- Added **RowNumber**

```sql
SELECT C.VisitID, Query_MisQryID, Text, Value, ActivityDateTime,
ROW_NUMBER() OVER(PARTITION BY VisitID ORDER BY ActivityDateTime) AS SeqID
FROM dbo.tbSCIP_Catheters C
WHERE C.ActivityDateTime = (SELECT MIN(C2.ActivityDateTime)
FROM dbo.tbSCIP_Catheters C2
WHERE C.SourceID = C2.SourceID
AND C.VisitID = C2.VisitID)
```

```
<table>
<thead>
<tr>
<th>VisitID</th>
<th>Query_MisQryID</th>
<th>Text</th>
<th>Value</th>
<th>ActivityDateTime</th>
<th>SeqID</th>
</tr>
</thead>
<tbody>
<tr>
<td>V0-20130905101149763</td>
<td>GU.VOIDM</td>
<td>Voiding Method</td>
<td>Indwelling Catheter</td>
<td>2014-03-04 18:35:00.000</td>
<td>1</td>
</tr>
<tr>
<td>V0-20131204135110496</td>
<td>GU.VOIDM</td>
<td>Voiding Method</td>
<td>Indwelling Catheter</td>
<td>2014-04-07 13:37:00.000</td>
<td>1</td>
</tr>
<tr>
<td>V0-20140117081708679</td>
<td>GU.VOIDM</td>
<td>Voiding Method</td>
<td>Indwelling Catheter</td>
<td>2014-03-31 18:15:00.000</td>
<td>1</td>
</tr>
<tr>
<td>V0-20140120093109647</td>
<td>GU.VOIDM</td>
<td>Voiding Method</td>
<td>Indwelling Catheter</td>
<td>2014-03-31 17:30:00.000</td>
<td>1</td>
</tr>
<tr>
<td>V0-20140121343708729</td>
<td>GU.VOIDM</td>
<td>Voiding Method</td>
<td>Indwelling Catheter</td>
<td>2014-03-18 18:22:00.000</td>
<td>1</td>
</tr>
<tr>
<td>V0-2014021162649721</td>
<td>GU.VOIDM</td>
<td>Voiding Method</td>
<td>Indwelling Catheter</td>
<td>2014-03-04 17:30:00.000</td>
<td>1</td>
</tr>
<tr>
<td>V0-20140214135700034</td>
<td>GU.VOIDM</td>
<td>Voiding Method</td>
<td>Indwelling Catheter</td>
<td>2014-03-07 17:30:00.000</td>
<td>1</td>
</tr>
<tr>
<td>V0-20140219085714894</td>
<td>GU.VOIDM</td>
<td>Voiding Method</td>
<td>Indwelling Catheter</td>
<td>2014-03-04 23:54:00.000</td>
<td>1</td>
</tr>
<tr>
<td>V0-20140219085714894</td>
<td>GU.VOIDM</td>
<td>Voiding Method</td>
<td>Indwelling Catheter</td>
<td>2014-03-04 23:54:00.000</td>
<td>2</td>
</tr>
<tr>
<td>V0-20140219085714894</td>
<td>GU.VOIDM</td>
<td>Voiding Method</td>
<td>Indwelling Catheter</td>
<td>2014-03-04 23:54:00.000</td>
<td>3</td>
</tr>
</tbody>
</table>
```
Output based on Row_Number field

-- Using RowNumber
SELECT C.VisitID, Query_MisQryID, Text, Value, ActivityDateTime,
ROW_NUMBER() OVER(PARTITION BY VisitID ORDER BY ActivityDateTime) AS SeqI
INTO #TempTable
FROM dbo.tbSCP_Catheters C
WHERE C.ActivityDateTime = (SELECT MIN(C2.ActivityDateTime)
FROM dbo.tbSCP_Catheters C2
WHERE C.SourceID = C2.SourceID
AND C.VisitID = C2.VisitID)
ORDER BY C.VisitID

SELECT TT.*
FROM #TempTable TT
WHERE SeqID = (SELECT MIN(TT2.SeqID)
FROM #TempTable TT2
WHERE TT.VisitID = TT2.VisitID)

<table>
<thead>
<tr>
<th>VisitID</th>
<th>Query_MisQryID</th>
<th>Text</th>
<th>Value</th>
<th>ActivityDateTime</th>
<th>SeqID</th>
</tr>
</thead>
<tbody>
<tr>
<td>V0-201401230931095647</td>
<td>GU VOIDM</td>
<td>Voiding Method</td>
<td>Indwelling Catheter</td>
<td>2014-03-31 17:30:00.000</td>
<td>1</td>
</tr>
<tr>
<td>V0-20140123143708729</td>
<td>GU VOIDM</td>
<td>Voiding Method</td>
<td>Indwelling Catheter</td>
<td>2014-03-18 16:22:00.000</td>
<td>1</td>
</tr>
<tr>
<td>V0-20140203150301530</td>
<td>GU VOIDM</td>
<td>Voiding Method</td>
<td>Indwelling Catheter</td>
<td>2014-03-04 17:30:00.000</td>
<td>1</td>
</tr>
<tr>
<td>V0-20140211162649721</td>
<td>GU VOIDM</td>
<td>Voiding Method</td>
<td>Indwelling Catheter</td>
<td>2014-03-29 17:30:00.000</td>
<td>1</td>
</tr>
<tr>
<td>V0-20140214135700034</td>
<td>GU VOIDM</td>
<td>Voiding Method</td>
<td>Indwelling Catheter</td>
<td>2014-03-07 17:30:00.000</td>
<td>1</td>
</tr>
<tr>
<td>V0-20140219085714894</td>
<td>GU VOIDM</td>
<td>Voiding Method</td>
<td>Indwelling Catheter</td>
<td>2014-03-04 23:54:00.000</td>
<td>1</td>
</tr>
</tbody>
</table>
Create a single column list
(from multiple value columns)

-- get procedures

IF Object_ID ('tempdb..dbo.#Procedures') IS NOT NULL
DROP TABLE #Procedures

SELECT
ACP.VisitID,
ACP.ProcedureSeqID,
ACP.ProcedureCode,
ACP.ProcedureCodeName

INTO #Procedures -- SELECT * FROM #Procedures
FROM Acmeware_MUST.dbo.AmOperationProcedures AOP
ORDER BY 1,2

<table>
<thead>
<tr>
<th>VisitID</th>
<th>ProcedureSeqID</th>
<th>ProcedureCode</th>
<th>ProcedureCodeName</th>
</tr>
</thead>
<tbody>
<tr>
<td>V0-20100607130016211</td>
<td>1</td>
<td>99.29</td>
<td>INJECT/INFUSE NEC</td>
</tr>
<tr>
<td>V0-20100810104655375</td>
<td>1</td>
<td>81.08</td>
<td>LUMBAR AND LUMBO SACRAL FUSION POSTERIOR TECHNIQUE</td>
</tr>
<tr>
<td>V0-20100810104655375</td>
<td>1</td>
<td>81.62</td>
<td>FUSION/REFUS OF 2-3 VERTEBRAE</td>
</tr>
<tr>
<td>V0-20100810104655375</td>
<td>1</td>
<td>80.51</td>
<td>EXCISION INTERVET DISC</td>
</tr>
<tr>
<td>V0-20100816083747291</td>
<td>1</td>
<td>88.58</td>
<td>CLOSURE SKIN &amp; SUBCUTANEOUS NEC</td>
</tr>
<tr>
<td>V0-20100816080531391</td>
<td>1</td>
<td>86.53</td>
<td>CLOSURE SKIN &amp; SUBCUTANEOUS NEC</td>
</tr>
<tr>
<td>V0-20100816101550022</td>
<td>1</td>
<td>88.70</td>
<td>UD RECONSTRUCTION NOS</td>
</tr>
<tr>
<td>V0-20100816122177835</td>
<td>1</td>
<td>98.51</td>
<td>[ESWL] OF THE KIDNEY, URETER AND/OR BLADDER</td>
</tr>
<tr>
<td>V0-20100816124307439</td>
<td>1</td>
<td>66.07</td>
<td>INSERTION OF TOTALLY IMPLANTABLE VASC ACCESS DEVIC</td>
</tr>
<tr>
<td>V0-20100816124307439</td>
<td>1</td>
<td>59.23</td>
<td>INJECTION OR INFUSION ERIM AS ANTI-INFLAMMATORY AGENT</td>
</tr>
<tr>
<td>V0-20100816124307439</td>
<td>1</td>
<td>87.33</td>
<td>THORAC SFT TISS XRAY NEC</td>
</tr>
</tbody>
</table>

Some examples are cpt codes, diagnosis codes, procedure codes, and allergies
Using FOR XML to create a single list

--- Create list

IF Object_Id ('tempdb.dbo.#List') IS NOT NULL
DROP TABLE #List

SELECT DISTINCT
P.VisitID,
ISNULL(( SELECT P1.ProcedureCode + ';' AS 'data()'
FROM #Procedures P1 WHERE P1.VisitID = P.VisitID FOR XML PATH('')),'') AS ProcedureList

INTO #List
FROM #Procedures P

SELECT * FROM #List

<table>
<thead>
<tr>
<th>VisitID</th>
<th>ProcedureList</th>
</tr>
</thead>
<tbody>
<tr>
<td>V0-20100507135016212</td>
<td>99.29;</td>
</tr>
<tr>
<td>V0-20100810104655376</td>
<td>81.08; 81.62; 80.51;</td>
</tr>
<tr>
<td>V0-20100816094734729</td>
<td>86.59;</td>
</tr>
<tr>
<td>V0-20100816095313981</td>
<td>86.59;</td>
</tr>
<tr>
<td>V0-20100816115150222</td>
<td>08.70;</td>
</tr>
<tr>
<td>V0-20100816122217785</td>
<td>98.51;</td>
</tr>
<tr>
<td>V0-20100816124307499</td>
<td>86.07; 99.28; 87.39;</td>
</tr>
</tbody>
</table>
TSQL Tips - Dates

• SQL Date Default
  • ‘5/26/17’ defaults to 5/26/17 00:00:00

• Getdate()
  • Gets Current date and time

• DateDiff
  • Calculates the difference between two dates

• DateAdd
  • Adds a period of time to a date (or subtracts)
    • Years, Months, Days, Hours, Minutes or Seconds

These three functions will create any date you need to automate a stored procedure.
SQL Date Time Default

Because SQL defaults to a time of 00:00:00. We code for that with a DateAdd.

Keep this in mind when creating data range parameters so that you include the full last day of the search.
DateAdd Calculations

First Day of Last Month:
SELECT DATEADD(MM, DATEDIFF(MM,0,DATEADD(MM, -1,GETDATE())), 0)

Explanation:
1. 0 = 19000101
2. The DATEDIFF calculates the number of months since 19000101
3. The DATEADD adds the same number of months back to 19000101 to give you the beginning of the current month

Last Day of Current Month:
SELECT DATEADD(SS,-1,DATEADD(MM,DATEDIFF(MM,0,GETDATE())+1,0))

Explanation:
1. DATEDIFF(MM,0,GETDATE())+1 - calculates the number of months from the current date since 19000101 and adds 1
2. DATEADD(MM,DATEDIFF(MM,0,GETDATE())+1,0) - adds the above number of months to 19000101 (this will give you the first day of next month)
3. The last DATEADD substracts 1 second to give you the last day of the current month (ie. 9/30/09 23:59:59 )

First Day of Last Month:
SELECT DATEADD(MM, DATEDIFF(MM,0,DATETIMEADD(MM,-1,GETDATE())),0)

Explanation:
1. DATEADD(MM,-1,GETDATE()) - Subtracts 1 month from current date
2. DATEDIFF(MM,0,DATETIMEADD(MM,-1,GETDATE())) - calculates the number of months since 19000101
3. The DATEADD adds the calculated number of months back to 19000101 to give you the beginning of the previous month
DateAdd Calculations

Last Day of Last Month:
SELECT DATEADD(SS,-1,DATEADD(MM, DATEDIFF(MM, 0, GETDATE()), 0))

Explanation:
DATEADD(MM, DATEDIFF(MM, 0, GETDATE()), 0) - same code as getting the first day of the current month
DATEADD substracts 1 second to give you the last day of previous month

First Day of Current Year:
SELECT DATEADD(YY, DATEDIFF(YY, 0, GETDATE()), 0)

Explanation:
1. 0 = 19000101
2. The DATEDIFF calculates the number of years since 19000101
3. The DATEADD adds the same number of years back to 19000101 to give you the beginning of the current year
4. This is the same as the month calculations but instead of mm for month you use the yy for year

Last Day of Last Year:
SELECT DATEADD(SS,-1,DATEADD(YY, DATEDIFF(YY, 0, GETDATE()), 0))

Explanation:
1. 0 = 19000101
2. The DATEDIFF calculates the number of years since 19000101
3. The DATEADD adds the same number of years back to 19000101 to give you the beginning of the current year
4. The next DATEADD substracts 1 second to reflect the day before just before midnight.
Examples using DateAdd

SELECT DATEADD(MM, -6, GETDATE())
   -- Subtracting 6 months from now
SELECT CONVERT(DATETIME, CONVERT(CHAR, DATEADD(MM, -6, GETDATE()), 101))
   -- Subtracting 6 months from right now then removing time factor
SELECT DATEADD(MM, -6, DATEADD(MM, DATEDIFF(MM, 0, GETDATE()), 0))
   -- Getting the beginning of the month 6 months ago

Understanding how the data functions work will help you write the appropriate code for your particular needs.
Using WITH (NOLOCK)

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

When data in a database is read or modified, the database engine uses special types of controls, called locks, to maintain integrity in the database. Locks basically work by making sure database records involved in a transaction cannot be modified by other transactions until the first transaction has committed, ensuring database consistency.

The benefit of using WITH (NOLOCK) is that it allows you to keep the database engine from issuing locks against the tables in your queries; this increases concurrency and performance because the database engine does not have to maintain the shared locks involved.
SSRS – Reporting Services

- Stored procedures and Reports are typically developed by someone in IS.

- The report is highly customizable with various options for display.
  - Tables, Matrix tables, charts and gauges are all reporting options.

- The reports are developed to run with or without input parameters.

- Reports are deployed and access given to groups and users.

- End Users access and run the report but can not modify.

- Modifications are done in IS.

- SSRS Reports are the best option for more complicated SQL queries.
Example Stored Procedure

```
ALTER PROC spMuseLabTests
AS

SELECT BV.VisitID,
    BV.AccountNumber, BV.Name,
    BV.FinancialClassID,
    BV.InpatientOrOutpatient,
    BV.Sex, BV.BirthDateTime,
    BV.PrimaryInsuranceID,
    TestMnemonic,
    TestName,
    ResultDateTime,
    NormalRange,
    ResultRW,
    AbnormalFlag,
CASE WHEN AbnormalFlag = ' *' THEN 'Other'
    WHEN AbnormalFlag like '%H%' THEN 'High'
    WHEN AbnormalFlag like '%L%' THEN 'Low'
END AS AbnormalFlagText
FROM TestMdb.dbo.LabSpecimenTests LST
INNER JOIN TestMdb.dbo.BarVisits BV
ON LST.SourceID = BV.SourceID
AND LST.VisitID = BV.VisitID
WHERE DATEDIFF(MM, ResultDateTime, GETDATE()) < 3
ORDER BY 1, 8, 10
```

Example only
Lab results for the past 3 months.

© 2020 Acmeware, Inc.
SSRS Report Development

Solution Explorer with Design View.

**Data Source** – defines database connection

**Reports** – contains all developed reports
SSRS Development

Report Design view with available data fields from previous stored procedure
SSRS Development

There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.
SSRS Deployed report

Report example grouped by patient and lab test with details regarding test results

<table>
<thead>
<tr>
<th>BAYNE, SCOTTIE</th>
<th>N00014332</th>
</tr>
</thead>
<tbody>
<tr>
<td>Financial Class: U</td>
<td>DOB: 11/16/1952</td>
</tr>
<tr>
<td>Primary Insurance: NON</td>
<td>Sex: F</td>
</tr>
<tr>
<td>Test Name: CHOLESTEROL TOTAL SERUM</td>
<td>Result Date Time</td>
</tr>
<tr>
<td></td>
<td>5/8/2012 8:33:00 AM</td>
</tr>
<tr>
<td>Test Name: GLUCOSE SERUM FASTING</td>
<td>Result Date Time</td>
</tr>
<tr>
<td></td>
<td>5/8/2012 8:33:00 AM</td>
</tr>
<tr>
<td>Test Name: HDL CHOLESTEROL</td>
<td>Result Date Time</td>
</tr>
<tr>
<td></td>
<td>5/8/2012 8:33:00 AM</td>
</tr>
<tr>
<td>Test Name: LDL CHOLESTEROL</td>
<td>Result Date Time</td>
</tr>
<tr>
<td></td>
<td>5/8/2012 8:33:00 AM</td>
</tr>
</tbody>
</table>
Look for our MUSE sessions

• Tuesday, May 29
  • 702 - Custom BCA Dashboards with Visual Insight
  • 703 - The Alphabet Soup of Clinical Quality Measures Reporting and Reimbursement: 2018 Updates
  • 704 - Soup to Nuts - Data Repository 101
  • 802 - Report Designer Fundamentals
  • 804 - Soup to Nuts – Data Repository 102

• 1010 - Revenue Cycle Optimization: Tools and Strategies for Success
  Wednesday May 30 at 2:30 pm

• 1087 - HIE: Effective Integration and Interoperability
  Thursday May 31 at 1:45 pm

• 1104 - The DR Overnight DBA
  Thursday May 31 at 2:45 pm

• 1091 - Electronic Reporting: Quality Management Cycle Concepts that Achieve Reliable Results
  Friday June 1 at 9:00 am

• 1103 - The Report Request Lifecycle
  Friday June 1 at 10:00 am