Today’s Agenda

- Finding Data
- Report Development Standards
- Writing Efficient Code
- T-SQL Tips
- Report Development
  - SSRS Reports
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
Meditech DR Home Page

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
Meditech – Data Model

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

### Application: OM

<table>
<thead>
<tr>
<th>Table Name</th>
<th>System</th>
<th>Primary Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>OmAccess_AomProcessFunctions</td>
<td></td>
<td>SourceID, OmAccessID</td>
</tr>
<tr>
<td>OmAccess_ClinicalDataFunctions</td>
<td></td>
<td>SourceID, OmAccessID</td>
</tr>
<tr>
<td>OmAccess_Main</td>
<td></td>
<td>SourceID, OmAccessID</td>
</tr>
<tr>
<td>OmAccess_ProcessFunctions</td>
<td></td>
<td>SourceID, OmAccessID</td>
</tr>
<tr>
<td>OmAccess_AmbOrderPrintForms</td>
<td></td>
<td>SourceID, OmAccessID, AmbulatoryOrderPrintForm_OmFormatDictID</td>
</tr>
<tr>
<td>OmAccess_AomCatGrpSortOrder</td>
<td></td>
<td>SourceID, OmAccessID, AomCategoryGroup_OmGrpID</td>
</tr>
<tr>
<td>OmAccess_AomControlSchedules</td>
<td></td>
<td>SourceID, OmAccessID, AomControlScheduleID</td>
</tr>
<tr>
<td>OmAccess_AomForms</td>
<td></td>
<td>SourceID, OmAccessID, AomFormID</td>
</tr>
<tr>
<td>OmAccess_HomeMedicationForms</td>
<td></td>
<td>SourceID, OmAccessID, HomeMedicationForm_OmFormatDictID</td>
</tr>
<tr>
<td>OmAccess_Identifiers</td>
<td></td>
<td>SourceID, OmAccessID, IdentifierTypeID, IdentifierID</td>
</tr>
<tr>
<td>OmAccess_OmAckCategories</td>
<td></td>
<td>SourceID, OmAccessID</td>
</tr>
<tr>
<td>OmAccess_OmAckMedTypes</td>
<td></td>
<td>SourceID, OmAccessID</td>
</tr>
<tr>
<td>OmAccess_OmCatGrpSortOrder</td>
<td></td>
<td>SourceID, OmAccessID</td>
</tr>
<tr>
<td>OmAccess_OmReviewCategories</td>
<td></td>
<td>SourceID, OmAccessID</td>
</tr>
<tr>
<td>OmAccess_OmReviewMedTypes</td>
<td></td>
<td>SourceID, OmAccessID</td>
</tr>
<tr>
<td>OmCat_Main</td>
<td></td>
<td>SourceID, OmCatID</td>
</tr>
<tr>
<td>OmCat_Facilities</td>
<td></td>
<td>SourceID, OmCatID</td>
</tr>
<tr>
<td>OmCat_ConnectionOrderRules</td>
<td></td>
<td>SourceID, OmCatID</td>
</tr>
<tr>
<td>OmCat_RuleEvaluateAt</td>
<td></td>
<td>SourceID, OmCatID</td>
</tr>
<tr>
<td>OmCat_Rules</td>
<td></td>
<td>SourceID, OmCatID</td>
</tr>
</tbody>
</table>

**Primary key and foreign keys**

- **Primary Keys**:
  - SourceID, OmAccessID
  - SourceID, OmAccessID, AmbulatoryOrderPrintForm_OmFormatDictID
  - SourceID, OmAccessID, AomCategoryGroup_OmGrpID
  - SourceID, OmAccessID, AomControlScheduleID
  - SourceID, OmAccessID, AomFormID
  - SourceID, OmAccessID, HomeMedicationForm_OmFormatDictID
  - SourceID, OmAccessID, IdentifierTypeID, IdentifierID
  - SourceID, OmAccessID
  - SourceID, OmAccessID
  - SourceID, OmAccessID
  - SourceID, OmAccessID, AomCategoryGroup_OmGrpID
  - SourceID, OmAccessID, AomCategoryLookup
  - SourceID, OmAccessID, AomIncludeAsStringOfString
  - SourceID, OmAccessID, CopyFromOld_OmCatID
  - SourceID, OmCatID
  - SourceID, OmCatID
  - SourceID, OmCatID
  - SourceID, OmCatID
  - SourceID, OmCatID

- **Foreign Key Joins**:
  - OmCat_ConnectionOrderRules
  - OmCat_Facilities
  - OmCat_Identifiers
  - OmCat_LocationContents
  - OmCat_PrintTexts_PrintText
  - OmCat_RuleEvaluateAt
  - OmCat_Rules
  - OmCat_Words
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

<table>
<thead>
<tr>
<th>Name</th>
<th>TableID</th>
<th>Data Type</th>
<th>Length</th>
<th>SortKey</th>
<th>NprDpm</th>
<th>NprSegment</th>
<th>NprElement</th>
</tr>
</thead>
<tbody>
<tr>
<td>abpaed</td>
<td>PatientStatusComment</td>
<td>varchar</td>
<td>75</td>
<td>0</td>
<td>ABS.PAT</td>
<td>abc.data</td>
<td>ABS.PAT.abc.pt.status.com</td>
</tr>
<tr>
<td>abpaicq</td>
<td>YnComment</td>
<td>varchar</td>
<td>70</td>
<td>0</td>
<td>ABS.PAT</td>
<td>ins.cd.queries</td>
<td>ABS.PAT.ins.cd.yn.comment</td>
</tr>
<tr>
<td>abpProjectorQueries</td>
<td>YesNoComment</td>
<td>varchar</td>
<td>70</td>
<td>0</td>
<td>ABS.PAT</td>
<td>projects.queries</td>
<td>ABS.PAT.projects.yn.comment</td>
</tr>
<tr>
<td>absurDenialAppeals</td>
<td>YesNoComment</td>
<td>varchar</td>
<td>75</td>
<td>0</td>
<td>ABS.PAT</td>
<td>ur.denial.appeal</td>
<td>ABS.PAT.ur.denial.appeal.comment</td>
</tr>
<tr>
<td>absurEventQueries</td>
<td>YnComment</td>
<td>varchar</td>
<td>70</td>
<td>0</td>
<td>ABS.PAT</td>
<td>ur.event.cds.queries</td>
<td>ABS.PAT.ur.event.cds.query.yn.comt</td>
</tr>
<tr>
<td>absurLevelsOfCare</td>
<td>Comment</td>
<td>varchar</td>
<td>75</td>
<td>0</td>
<td>ABS.PAT</td>
<td>ur.levels.of.care</td>
<td>ABS.PAT.ur.level.of.care.comment</td>
</tr>
<tr>
<td>AdminBedReservations</td>
<td>Comment</td>
<td>varchar</td>
<td>30</td>
<td>0</td>
<td>ADM.PAT</td>
<td>bed.reservations</td>
<td>ADM.PAT.bed.reservation.comment</td>
</tr>
<tr>
<td>AdminClinDepartureDate</td>
<td>Comment</td>
<td>varchar</td>
<td>50</td>
<td>0</td>
<td>ADM.PAT</td>
<td>cli.departure.data</td>
<td>ADM.PAT.cli.departure.comment</td>
</tr>
</tbody>
</table>

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>ColumnObjClass</th>
<th>ColumnRecord</th>
<th>SortOrder</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCG0000043</td>
<td>DrTable Test ToolDataTypes</td>
<td>DataTypesYnComment</td>
<td>DrTableTest</td>
<td>TestDataTypes</td>
<td>13</td>
</tr>
<tr>
<td>FCG0000062</td>
<td>DrTable Test KeyedTimeFiles</td>
<td>UseSetComment</td>
<td>DrTableTest</td>
<td>TestKeyedTimeFiles</td>
<td>5</td>
</tr>
<tr>
<td>FCG0000043</td>
<td>DrTable Test ToolTimeFiles</td>
<td>Common</td>
<td>DrTableTest</td>
<td>TestTimeFiles</td>
<td>5</td>
</tr>
<tr>
<td>FCG00003127</td>
<td>EdinParam_Mar</td>
<td>MarScheduleComments</td>
<td>EdinParam</td>
<td>Mar</td>
<td>44</td>
</tr>
<tr>
<td>FCG00003127</td>
<td>EdinParam_Mar</td>
<td>MarCommentPopUp</td>
<td>EdinParam</td>
<td>Mar</td>
<td>45</td>
</tr>
<tr>
<td>FCG00003127</td>
<td>EdinParam_Mar</td>
<td>MarCommentRemoveHours</td>
<td>EdinParam</td>
<td>Mar</td>
<td>46</td>
</tr>
<tr>
<td>FCG00003669</td>
<td>EdinStationStatus_Main</td>
<td>Comment</td>
<td>EdinStationStatus</td>
<td>Main</td>
<td>5</td>
</tr>
<tr>
<td>FCG00002212</td>
<td>EmrAcctItem_BloodReactionComments</td>
<td>BloodReactionCommentUnId</td>
<td>EmrAcctItem</td>
<td>BloodReactionComments</td>
<td>10</td>
</tr>
<tr>
<td>FCG00002212</td>
<td>EmrAcctItem_BloodReactionComments</td>
<td>BloodReactionComment</td>
<td>EmrAcctItem</td>
<td>BloodReactionComments</td>
<td>13</td>
</tr>
</tbody>
</table>

AcmeWare
Shift F9 and Shift F8 for Magic
Identifying AT Data Fields in the 6.0 DR

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.
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:

```sql
EXEC spBarAccountsByAccountType
```

Rather than......

```sql
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(D...
ORDER BY 1,2
```
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
Document in your code!

• Documenting throughout your code is a huge help with understanding later...just what you were thinking at the time.
• 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.
/******CAUTION - DO NOT ALTER THE CODE BELOW*******/

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 diag 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 #PatDiaDig PD
WHERE (PD.PrimaryDiag IS NOT NULL
OR PD.SecondaryDiag IS NOT NULL)
AND PD.DiagnosisSeqID = (SELECT MIN(PD1.DiagnosisSeqID)
FROM #PatDiaDig 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
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.
  ▪ A max of 249 non-clustered indexes per table – You will want to be selective with indexes you build
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

Insert screenshot of an interface for creating an index. The interface shows a table named `BarVisits` and an index named `ixVisitID`. The index is non-unique and non-clustered. The screenshot also includes a list of columns and their properties.
Example of Execution Plan

Display Estimated and Actual Execution Plan
Primary Keys

• Joining on the primary keys will make your report run faster.
• Omitting the primary key will not only slow down your query but many times your output will not be correct.
• Each application has a unique identifier (primary key) that will allow you to join to other applications.
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

```sql
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
WHERE Status='ADM IN'
```

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
## Application Parent tables
(with patient data)

<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

```sql
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

```sql
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'
```

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

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

SELECT Name, BirthDateTime, CalcAge
FROM livedb
INNER JOIN
ON AV.Source = 1
AND AV.Visit = 1
WHERE Status = 'active'

<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:

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.
Function - fxMeditechTimeIDToDateTime

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

• Temp Tables
• Row_Number
• Multiples to a single column
• Dates
• Creating a #Dates and populating
• Using WITH (NOLOCK)
What is a temp table?

- 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.
- Temp tables are deleted when the connection to the database is closed.
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.

```sql
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)
ORDER BY C.VisitID
```

**Results**

<table>
<thead>
<tr>
<th>VisitID</th>
<th>Query_MisQryID</th>
<th>Text</th>
<th>Value</th>
<th>ActivityDateTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>V0-20130905101149763</td>
<td>GU.VOIDDM</td>
<td>Voiding</td>
<td>Indwelling Catheter</td>
<td>2014-03-04 18:35:00.000</td>
</tr>
<tr>
<td>V0-20131204135110496</td>
<td>GU.VOIDDM</td>
<td>Voiding</td>
<td>Indwelling Catheter</td>
<td>2014-04-07 13:37:00.000</td>
</tr>
<tr>
<td>V0-20140117081708679</td>
<td>GU.VOIDDM</td>
<td>Voiding</td>
<td>Indwelling Catheter</td>
<td>2014-03-31 18:15:00.000</td>
</tr>
<tr>
<td>V0-20140120093109647</td>
<td>GU.VOIDDM</td>
<td>Voiding</td>
<td>Indwelling Catheter</td>
<td>2014-03-31 17:30:00.000</td>
</tr>
<tr>
<td>V0-20140123143708739</td>
<td>GU.VOIDDM</td>
<td>Voiding</td>
<td>Indwelling Catheter</td>
<td>2014-03-18 18:22:00.000</td>
</tr>
<tr>
<td>V0-201402203150301530</td>
<td>GU.VOIDDM</td>
<td>Voiding</td>
<td>Indwelling Catheter</td>
<td>2014-04-04 17:30:00.000</td>
</tr>
<tr>
<td>V0-20140211162649721</td>
<td>GU.VOIDDM</td>
<td>Voiding</td>
<td>Indwelling Catheter</td>
<td>2014-03-29 17:30:00.000</td>
</tr>
<tr>
<td>V0-20140214135700034</td>
<td>GU.VOIDDM</td>
<td>Voiding</td>
<td>Indwelling Catheter</td>
<td>2014-03-07 17:30:00.000</td>
</tr>
</tbody>
</table>

**-- 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)
ORDER BY C.VisitID
```

**Results**

<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>V0-20140219085714984</td>
<td>GU.VOIDDM</td>
<td>Voiding</td>
<td>Indwelling Catheter</td>
<td>2014-03-04 23:54:00.000</td>
<td>1</td>
</tr>
<tr>
<td>V0-20140219085714984</td>
<td>GU.VOIDDM</td>
<td>Voiding</td>
<td>Indwelling Catheter</td>
<td>2014-03-04 23:54:00.000</td>
<td>2</td>
</tr>
<tr>
<td>V0-20140219085714984</td>
<td>GU.VOIDDM</td>
<td>Voiding</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

```sql
SELECT C.VisitID, Query_MisQryID, Text, Value, ActivityDateTime,
       ROW_NUMBER() OVER(PARTITION BY VisitID ORDER BY ActivityDateTime) AS SeqID
INTO #TempTable
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)
ORDER BY C.VisitID
```

```sql
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>V0-20140123093109647</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>V0-20140213085714884</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>

Sequencing rows are useful when your output needs to be a single row per patient, visit or other value.
Create a single column list
(from multiple value columns)

-- get procedures

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

SELECT
    AOP.VisitID,
    AOP.ProcedureSeqID,
    AOP.ProcedureCode,
    AOP.ProcedureCodeName

INTO #Procedures

FROM Acmeware.MUSE.dbo.AbsOperationProcedures AOP

ORDER BY 1,2

<table>
<thead>
<tr>
<th>VisitID</th>
<th>ProcedureSeqD</th>
<th>ProcedureCode</th>
<th>ProcedureCodeName</th>
</tr>
</thead>
<tbody>
<tr>
<td>V0-20100807135016212</td>
<td>1</td>
<td>59.23</td>
<td>INJECT/INFUSE NEC</td>
</tr>
<tr>
<td>V0-20100810104858375</td>
<td>1</td>
<td>81.08</td>
<td>LUMBAR AND LUMBOSacral FUSION POSTerior TECHNIQUE</td>
</tr>
<tr>
<td>V0-20100810104858375</td>
<td>1</td>
<td>81.02</td>
<td>FUSION/REFUS OF 2 VerteBRAE</td>
</tr>
<tr>
<td>V0-20100810104858375</td>
<td>1</td>
<td>80.51</td>
<td>EXcision INTERVert DISC</td>
</tr>
<tr>
<td>V0-201008160913734729</td>
<td>1</td>
<td>86.59</td>
<td>CLOSURE SKIN &amp; SUBCUTANEOUS NEC</td>
</tr>
<tr>
<td>V0-20100816095033991</td>
<td>1</td>
<td>86.59</td>
<td>CLOSURE SKIN &amp; SUBCUTANEOUS NEC</td>
</tr>
<tr>
<td>V0-20100816119150022</td>
<td>1</td>
<td>88.70</td>
<td>UID RECONSTRUCTION NOS</td>
</tr>
<tr>
<td>V0-2010081612217785</td>
<td>1</td>
<td>98.51</td>
<td>[ESWL] OF THE KIDNEY, URETER AND/OR BLADDER</td>
</tr>
<tr>
<td>V0-20100816124307499</td>
<td>1</td>
<td>98.37</td>
<td>Insertion of Totally Implantable Vasc Access Devic</td>
</tr>
<tr>
<td>V0-20100816124307499</td>
<td>1</td>
<td>99.23</td>
<td>INJECTION or INFUSION SRM, ANTIneoplASMATIC AGENT</td>
</tr>
<tr>
<td>V0-20100816124307499</td>
<td>1</td>
<td>87.39</td>
<td>thorAX 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>V0-2010081612217785</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
  ▪ ‘5/26/15’ defaults to 5/26/15 00:00:00

• 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
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.
DateDiff calculations have countless uses in your code development.

SELECT GETDATE()
-- Gets today's date and time
SELECT DATEDIFF(YY,'05/01/15',GETDATE())
-- Difference in years
SELECT DATEDIFF(MM,'05/01/15',GETDATE())
-- Difference in months
SELECT DATEDIFF(DD,'05/01/15',GETDATE())
-- Difference in days
SELECT DATEDIFF(HH,'05/01/15',GETDATE())
-- Difference in hours
SELECT DATEDIFF(MINUTE,'05/01/15',GETDATE())
-- Difference in minutes
SELECT DATEDIFF(SECOND,'05/01/15',GETDATE())
-- Difference in seconds

* Be aware of the time values and rounding specifically with Hours
### DateDiff Calculations for Elapsed Time

```sql
SELECT
  CONVERT(DECIMAL(20, 7), DATEDIFF(minute, '05/01/14 06:00:00', '05/2/14 12:25:30')) / 60 AS HoursIn,
  DATEDIFF(HH, '05/01/14 06:00:00', '05/2/14 12:25:30') AS HoursIn2,
  DATEDIFF(HH, '05/01/14 06:00:00', '05/2/14 12:55:30') AS HoursIn3,
  CONVERT(VARCHAR(3), FLOOR(CONVERT(DECIMAL(20, 7), DATEDIFF(minute, '05/01/14 06:00:00', '05/2/14 12:25:30')) / 60)) + '.' +
  CONVERT(VARCHAR(3), FLOOR(CONVERT(DECIMAL(20, 7), DATEDIFF(SS, '05/01/14 06:00:00', '05/2/14 12:25:30')) / 3600) * 60) AS HoursAndMinutes
FROM
```

<table>
<thead>
<tr>
<th>HoursIn</th>
<th>HoursIn2</th>
<th>HoursIn3</th>
<th>HoursAndMinutes</th>
</tr>
</thead>
<tbody>
<tr>
<td>30.4166666666</td>
<td>30.0000000000</td>
<td>30.0000000000</td>
<td>30.2500000000</td>
</tr>
</tbody>
</table>

Be aware of using standard SQL functions, they may not work the way you expect.
DateAdd Calculations

First Day of Current Month:
SELECT DATEADD(MM, DATEDIFF(MM,0,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. DATETIMEADD(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:

<table>
<thead>
<tr>
<th>SQL Expression</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| `SELECT DATEADD(SS,-1,DATEADD(MM,DATEDIFF(MM,0,GETDATE()),0))` | 1. DATEADD(MM,DATEDIFF(MM,0,GETDATE()),0) - same code as getting the first day of the current month  
2. DATEADD substracts 1 second to give you the last day of previous month |

## Last Day of Last Year:

<table>
<thead>
<tr>
<th>SQL Expression</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| `SELECT DATEADD(SS,-1,DATEADD(YY,DATEDIFF(YY,0,GETDATE()),0))` | 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 |

## First Day of Current Year:

<table>
<thead>
<tr>
<th>SQL Expression</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| `SELECT DATEADD(YY,DATEDIFF(YY,0,GETDATE()),0)` | 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 |
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.
Creating a #Dates Temp Table

The #Dates table works great for Summaries that need to include every day regardless of data values.
Using WITH (NOLOCK)

`ALTER PROC [dbo].[spErAlertNoDepart]
AS

SELECT EAD.*,
AVE.EventUserID AS LastUserID,
DMU.Name AS LastUserName

FROM tbErAlertDetails EAD WITH (NOLOCK)
LEFT JOIN livedb.dbo.AdmVisitEvents AVE WITH (NOLOCK)
ON EAD.SourceID = AVE.SourceID
AND EAD.VisitID = AVE.VisitID
AND EventDateTime = (SELECT MAX(AVE1.EventDateTime)
FROM livedb.dbo.AdmVisitEvents AVE1
INNER JOIN livedb.dbo.DMisUsers DMU
ON AVE1.SourceID = DMU.SourceID
AND AVE1.EventUserID = DMU.UserID
WHERE AVE1.SourceID = AVE1.SourceID
AND AVE1.VisitID = AVE1.VisitID)
AND EventSeqID = (SELECT MAX(AVE2.EventSeqID)
FROM livedb.dbo.AdmVisitEvents AVE2
INNER JOIN livedb.dbo.DMisUsers DMU
ON AVE2.SourceID = DMU.SourceID
AND AVE2.EventUserID = DMU.UserID
WHERE AVE2.SourceID = AVE2.SourceID
AND AVE2.VisitID = AVE2.VisitID
AND AVE2.EventDateTime = AVE2.EventDateTime)

LEFT JOIN livedb.dbo.DMisUsers DMU WITH (NOLOCK)
ON AVE.SourceID = DMU.SourceID
AND AVE.EventUserID = DMU.UserID

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.
Reporting Services - SSRS

• Stored procedures and Reports are 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.
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

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.
### Report Example

- **Patient:** BAYNE, SCOTTIE
- **Social Security Number:** N00014332

<table>
<thead>
<tr>
<th>Test Name</th>
<th>Result Date Time</th>
<th>Result</th>
<th>Normal Range</th>
<th>Abnormal Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHOLESTEROL TOTAL SERUM</td>
<td>5/8/2012 8:33:00 AM</td>
<td>146</td>
<td>50-200</td>
<td></td>
</tr>
<tr>
<td>GLUCOSE SERUM FASTING</td>
<td>5/8/2012 8:33:00 AM</td>
<td>133</td>
<td>74-106</td>
<td>H</td>
</tr>
<tr>
<td>HDL CHOLESTEROL</td>
<td>5/8/2012 8:33:00 AM</td>
<td>26</td>
<td>40-60</td>
<td>L</td>
</tr>
<tr>
<td>LDL CHOLESTEROL</td>
<td>5/8/2012 8:33:00 AM</td>
<td>94.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*SSRS Deployed report*

Report example grouped by patient and lab test with details regarding test results.
Discussion, Questions & Answers
Thank you.

Our other MUSE sessions!

• Tuesday 5/26 9:30-12:00
  ▪ 701 - Soup to Nuts – Data Repository 101
  ▪ 703 - Converting reports from NPR to SQL

• Tuesday 5/26 1:00-3:30
  ▪ 801 - Soup to Nuts – Data Repository 102
  ▪ 803 - Alphabet Soup of Clinical Quality Reporting

• Wednesday 5/27 10:30
  ▪ 1095 - Using Meditech Data to Drive Clinical Decision Support

• Thursday 5/28 9:15
  ▪ 1094 - Meaningful Use Audit, Is Your Organization Ready?
  ▪ 1133 – Click Here to Upgrade your DR to 6.1