Using The Meta Schema To View The CDW Translations Of Data Structures In VistA

Richard Pham

Enterprise Architect – BISL

Production 1.0

20-December-2012

Introduction

This paper is an introduction to how to use the Meta schema in CDW to translate tables and columns in the CDWWork SQL to the data's origins in VistA. This paper aims to present a couple of issues in brief:

1. How to use the Meta schema queries in CDWWork to look at design history and patterns

2. Where is the Metadata Report for a visual of the data in the Meta schema.

Using Queries In The Meta Schema

On VHACDWA01 in the CDWWork database, a schema available to anyone who has any permission level can see the Meta schema. The Meta schema has several tables:

Two tables that are the versions of the VistA data dictionary that CDW uses:

1. Meta.FileManFile – A table that contains a file list from VistA
2. Meta.FileManField – A table that contains field list from VistA

Five tables that are representations of what is in CDW, and where applicable, their translations in the VistA:

1. Meta.DWSchemaV – A list of the CDW schema, and what is generally contained in there. This table has a one-to-many relationship with Meta.DWTableV (One schema could have many tables).

2. Meta.DWTableV – A list of the tables in the CDW. This table has a one-to-many with Meta.DWFieldV (one table could have many fields (columns)). This table has a one-to-many with Meta.DWIndexV (one table could have many indexes (columns)).

3. Meta.DWFieldV – A list of the fields (columns) in the CDW. This table has

4. Meta.DWIndexV – A list of indexes that are placed on the CDW tables.

5. Meta.DWForeignKeyV – A list of foreign keys, and the tables that contain the referencing primary key.

Meta.FileManFile

Background

Think of this as a data dictionary pull from DHCP. This is a listing of all the files and definitions where available in DHCP. The version that is displayed is PUG (Puget Sound) – Station 663.

Table Structure - Meta.FileManFile

Column	DataType	Definition
SourceEntitySID	Int	The PK for the table. Defined as unique files
FileNumber	Varchar(50)	The File Number in FileMan.
FileName	Varchar(50)	The File Name in colloquial English in FileMan. Note that these don't have to make sense (look for FileName 401 for example)
ParentFileNumber	Varchar(50)	If this is a child table, what is the parent?
ParentEntitySID	Int	Recursive pointer to the SourceEntitySID
Description	Varchar(8000)	Description in FileMan

Useful Queries Using This Table

```
--Change to the right database
USE CDWWork
GO

--Look at the top 1000 Files

SELECT TOP 1000 *
FROM Meta.FileManFile
;

--Find files dealing with LOINC in the filename

SELECT *
FROM Meta.FileManFile
WHERE Description LIKE '%LOINC%'
;

--Find files that have "drug" in them

SELECT *
```

```
FROM Meta.FileManFile
WHERE FileName LIKE '%DRUG%'
;

--Find File 9000010 - VISIT

SELECT *
FROM Meta.FileManFile
WHERE FileNumber LIKE '9000010'
;

--Find Files and their parents

SELECT A.*, B.FileName AS ParentFileName
FROM Meta.FileManFile AS A INNER JOIN
Meta.FileManFile AS B
ON A.ParentEntitySID=B.SourceEntitySID
;
```

Pointers and Pointer Logic

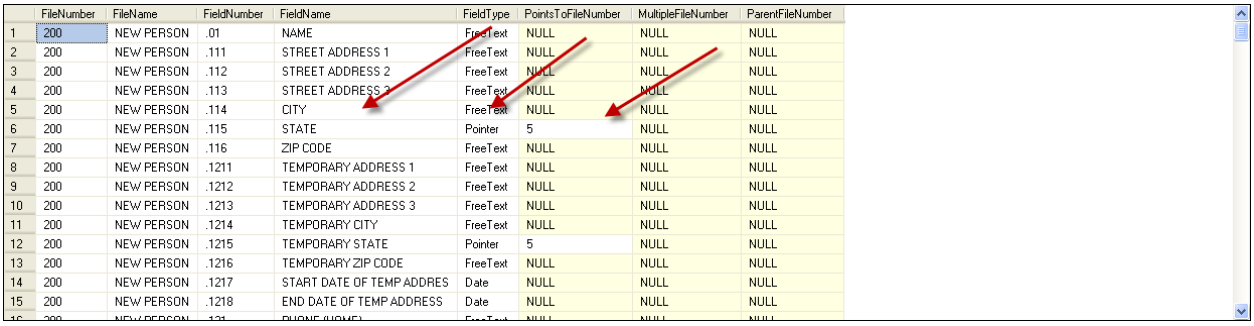
There are a couple ways that files can have relationships defined in VistA. The following is an explanation of the business context for why these relationships in the data are formed.

File 200 NEW PERSON to File 5 STATE – One-to-One Pointer

Let's take a simple example, what state is an employee from in the data warehouse.

```
--Your Entry in File 200
SELECT StaffName, StateName
FROM SStaff.SStaff
WHERE StaffName = 'ZZTEST, VA DOD ONE'
AND StateName = 'NEVADA'
;
```

You will see a row with the name and state. What follows is what actually goes on behind the scenes.



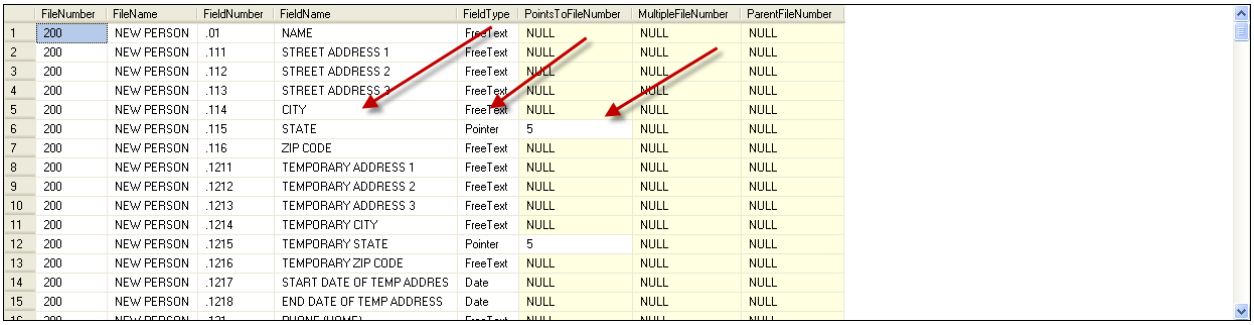
	FileNumber	FileName	FieldNumber	FieldName	FieldType	PointsToFileNumber	MultipleFileNumber	ParentFileNumber
1	200	NEW PERSON	.01	NAME	FreeText	NULL	NULL	NULL
2	200	NEW PERSON	.111	STREET ADDRESS 1	FreeText	NULL	NULL	NULL
3	200	NEW PERSON	.112	STREET ADDRESS 2	FreeText	NULL	NULL	NULL
4	200	NEW PERSON	.113	STREET ADDRESS 3	FreeText	NULL	NULL	NULL
5	200	NEW PERSON	.114	CITY	FreeText	NULL	NULL	NULL
6	200	NEW PERSON	.115	STATE	Pointer	5	NULL	NULL
7	200	NEW PERSON	.116	ZIP CODE	FreeText	NULL	NULL	NULL
8	200	NEW PERSON	.1211	TEMPORARY ADDRESS 1	FreeText	NULL	NULL	NULL
9	200	NEW PERSON	.1212	TEMPORARY ADDRESS 2	FreeText	NULL	NULL	NULL
10	200	NEW PERSON	.1213	TEMPORARY ADDRESS 3	FreeText	NULL	NULL	NULL
11	200	NEW PERSON	.1214	TEMPORARY CITY	FreeText	NULL	NULL	NULL
12	200	NEW PERSON	.1215	TEMPORARY STATE	Pointer	5	NULL	NULL
13	200	NEW PERSON	.1216	TEMPORARY ZIP CODE	FreeText	NULL	NULL	NULL
14	200	NEW PERSON	.1217	START DATE OF TEMP ADDRES	Date	NULL	NULL	NULL
15	200	NEW PERSON	.1218	END DATE OF TEMP ADDRESS	Date	NULL	NULL	NULL

File 200, the NEW PERSON File, contains information regarding personnel.

```
--What the Pointer looks like in File 200
--State is a 1 to 1 relationship, this is a
pointer
```

```
SELECT FileNumber, FileName,FieldNumber,  FieldN
ame, FieldType, PointsToFileNumber,MultipleFileN
umber, ParentFileNumber
FROM CDWWork.MetaFilemanField
WHERE FileNumber = '200'
;
```

As you can see, there is a bunch of demographic information in here:



	FileNumber	FileName	FieldNumber	FieldName	FieldType	PointsToFileNumber	MultipleFileNumber	ParentFileNumber
1	200	NEW PERSON	.01	NAME	FreeText	NULL	NULL	NULL
2	200	NEW PERSON	.111	STREET ADDRESS 1	FreeText	NULL	NULL	NULL
3	200	NEW PERSON	.112	STREET ADDRESS 2	FreeText	NULL	NULL	NULL
4	200	NEW PERSON	.113	STREET ADDRESS 3	FreeText	NULL	NULL	NULL
5	200	NEW PERSON	.114	CITY	FreeText	NULL	NULL	NULL
6	200	NEW PERSON	.115	STATE	Pointer	5	NULL	NULL
7	200	NEW PERSON	.116	ZIP CODE	FreeText	NULL	NULL	NULL
8	200	NEW PERSON	.1211	TEMPORARY ADDRESS 1	FreeText	NULL	NULL	NULL
9	200	NEW PERSON	.1212	TEMPORARY ADDRESS 2	FreeText	NULL	NULL	NULL
10	200	NEW PERSON	.1213	TEMPORARY ADDRESS 3	FreeText	NULL	NULL	NULL
11	200	NEW PERSON	.1214	TEMPORARY CITY	FreeText	NULL	NULL	NULL
12	200	NEW PERSON	.1215	TEMPORARY STATE	Pointer	5	NULL	NULL
13	200	NEW PERSON	.1216	TEMPORARY ZIP CODE	FreeText	NULL	NULL	NULL
14	200	NEW PERSON	.1217	START DATE OF TEMP ADDRES	Date	NULL	NULL	NULL
15	200	NEW PERSON	.1218	END DATE OF TEMP ADDRESS	Date	NULL	NULL	NULL

We want to look at state. Well, state isn't actually in this table, you can access it as a pointer.

The two values that you will see when you have a pointer is the fact that the datatype is a pointer, and that the destination (pointedtofilenumber) is another file. In this case, the way I generally read this out is "File 200/Field .1215 points to File 5."

In a pointer relationship, both files are independently maintained. That means, entries exist independently of each other and pointers happen to link certain records from each of the independent files together. A pointer is also a "one-way" door. You know where the file goes, but when you arrive there, you do not know from where you came.

Moving from File 5 to File 5.01 – Parent/Child Relationships from the Parent Perspective

We now turn our attention to File 5, the State File. This contains information about states.

```
--The state file, File 5
--Many counties in a state,
SELECT FileNumber, FileName,FieldNumber,  FieldN
ame, FieldType, PointsToFileNumber,MultipleFileN
umber, ParentFileNumber
FROM CDWWork.MetaFilemanField
WHERE FileNumber = '5'
;
```

There is a lot of information about the state. Notice that we do not know that there is a pointer into this file from File 200. That is because this file stands on its own; it needs no reference to be able to be understood. Information in this table is state The pointer relationship does not define this table, and it would be prohibitively cumbersome to define the relationship at the other end.

(For a really good demonstration of impracticality of defining all possible departures to a destination table, try File 80 – ICD. There are at least 423 files that point to it, which makes projects like the ICD-10-CM conversion interesting to say the least.)

	FileNumber	FileName	FieldNumber	FieldName	FieldType	PointsToFileNumber	MultipleFileNumber	ParentFileNumber
1	5	STATE	.001	NUMBER	Numeric	NULL	NULL	NULL
2	5	STATE	.01	NAME	FreeText	NULL	NULL	NULL
3	5	STATE	1	ABBREVIATION	FreeText	NULL	NULL	NULL
4	5	STATE	2	VA STATE CODE	FreeText	NULL	NULL	NULL
5	5	STATE	2.1	AAC RECOGNIZED	SetOfCodes	NULL	NULL	NULL
6	5	STATE	2.2	US STATE OR POSSESSION	SetOfCodes	NULL	NULL	NULL
7	5	STATE	3	COUNTY	SubFile	NULL	5.01	NULL
8	5	STATE	5	CAPITAL	FreeText	NULL	NULL	NULL

Now, we come into an entry that needs further definition, COUNTY. COUNTY is a subfile. When you see “subfile” in the DAR,

you look for a value in the “multiple file number” that tells you where this relationship ought to go. This means that from one record in the parent goes to multiple records in the child (or more commonly, “multiple”). In this example, there are many records in the COUNTY that correspond to a single record in STATE. The way that this is generally referred in English to this is “File 5 has a subfile/multiple at Field 3 which goes into File 5.01.”

5.01 – Child/Subfile/Multiple of File 5 – Parent/Child Relationships From The Child Perspective

We are now at COUNTY. You know this is a child of STATE, because now you see an entry in the ParentFileNumber.

```
--County
SELECT FileNumber, FileName,FieldNumber,  FieldN
ame, FieldType, PointsToFileNumber,MultipleFileN
umber, ParentFileNumber
FROM CDWWork.Meta.FilemanField
WHERE FileNumber = '5.01'
;
```

	FileNumber	FileName	FieldNumber	FieldName	FieldType	PointsToFileNumber	MultipleFileNumber	ParentFileNumber
1	5.01	COUNTY	.01	COUNTY	FreeText	NULL	NULL	5
2	5.01	COUNTY	1	ABBREVIATION	FreeText	NULL	NULL	5
3	5.01	COUNTY	2	VA COUNTY CODE	FreeText	NULL	NULL	5
4	5.01	COUNTY	3	CATCHMENT CODE	FreeText	NULL	NULL	5
5	5.01	COUNTY	4	ZIP CODE	SubFile	NULL	5.02	5
6	5.01	COUNTY	5	INACTIVE DATE	Date	NULL	NULL	5
7	5.01	COUNTY	663001	CITY	SubFile	NULL	5.03	5

Unlike the previous example, where you did not know that File 200 could go to File 5, you definitely know that File 5.01 COUNTY is a destination from its parent File 5 STATE. This is because records in the child/multiple/subfile are defined by their parent. This is somewhat hard to understand without seeing some data. So in this example, records at COUNTY file are a multiple of the STATE, and need the STATE file to mak sense of the data.

Here is a partial list of the 67 counties in Alabama.
Tuscaloosa
Jefferson
Lamar
Autauga

Tuscaloosa and Autauga counties are probably unique across the country, but I know that there is a Lamar County in Texas, and that there are several Jefferson counties across the country. The point is that the COUNTY file alone does not make sense. You need both the STATE and COUNTY records together to make sense. So, while Tuscaloosa and Lamar are ambiguous alone, Alabama – Tuscaloosa and Alabama – Lamar are definitive. A child table's data does not make sense without a parent. The CDW always gets parent data when a child table is used.

One consequence as well for this relationship, a child record (meaning a county record in this case) deletion will not do anything more than sever the relationship with the parent. Those records are called orphan records. However, if the parent is deleted (someone accidentally deletes a state), all corresponding child records are deleted. That means if the entry for File 5 in Alabama is deleted, all child records in File 5.01 are correspondingly deleted at the same time.

The reason why I tend to use "subfile" over "multiple" for the child table is that there are some "multiples" that have only one child for a parent in the present day. However, at anytime multiple records are entered, this parent-child relationship allows for that situation. Subfiles support 0, 1, or more than one possible relationship to the parent file. On the other hand, pointer relationship only allow for 0 or 1 matches between files.

Variable Pointers

Particularly in older central package tables, you run into something that is called a variable pointer. This is the trickiest relationship to deal with as the relationship always contains coding logic.

The example is from laboratory chemistry.

```
--Multiple Pointers
SELECT FileNumber, FileName, FieldNumber,  Field
Name, FieldType, PointsToFileNumber,MultipleFile
Number, ParentFileNumber
FROM CDWWork.MetaFilemanField
WHERE FileNumber = '63.04'
ORDER BY SourceAttributeSID ASC
;
```

	FileNumber	FileName	FieldNumber	FieldName	FieldType	PointsToFileNumber	MultipleFileNumber	ParentFileNumber
1	63.04	CHEM, HEM, TOX, RIA, SER, etc.	.01	DATE/TIME SPECIMEN TAKEN	DateTime	NULL	NULL	63
2	63.04	CHEM, HEM, TOX, RIA, SER, etc.	.02	DATE/TIME OBTAINED INEXACT	SetOfCodes	NULL	NULL	63
3	63.04	CHEM, HEM, TOX, RIA, SER, etc.	.03	DATE REPORT COMPLETED	DateTime	NULL	NULL	63
4	63.04	CHEM, HEM, TOX, RIA, SER, etc.	.04	VERIFY PERSON	Pointer	200	NULL	63
5	63.04	CHEM, HEM, TOX, RIA, SER, etc.	.05	SPECIMEN TYPE	Pointer	61	NULL	63
6	63.04	CHEM, HEM, TOX, RIA, SER, etc.	.06	ACCESSION	FreeText	NULL	NULL	63
7	63.04	CHEM, HEM, TOX, RIA, SER, etc.	.07	*VOLUME	FreeText	NULL	NULL	63
8	63.04	CHEM, HEM, TOX, RIA, SER, etc.	.08	METHOD OR SITE	FreeText	NULL	NULL	63
9	63.04	CHEM, HEM, TOX, RIA, SER, etc.	.09	SUM REPORT NUM.	Numeric	NULL	NULL	63
10	63.04	CHEM, HEM, TOX, RIA, SER, etc.	.1	REQUESTING PERSON	Pointer	200	NULL	63
11	63.04	CHEM, HEM, TOX, RIA, SER, etc.	.11	REQUESTING LOCATION	FreeText	NULL	NULL	63
12	63.04	CHEM, HEM, TOX, RIA, SER, etc.	.111	REQUESTING LOC/DIV	VariablePointer	NULL	NULL	63
13	63.04	CHEM, HEM, TOX, RIA, SER, etc.	.112	ACCESSIONING INSTITUTION	Pointer	4	NULL	63
14	63.04	CHEM, HEM, TOX, RIA, SER, etc.	.12	NEW PERSON CONVERSION	SetOfCodes	NULL	NULL	63
15	63.04	CHEM, HEM, TOX, RIA, SER, etc.	.31	UID	FreeText	NULL	NULL	63
16	63.04	CHEM, HEM, TOX, RIA, SER, etc.	.32	ORDERING SITE	Pointer	4	NULL	63
17	63.04	CHEM, HEM, TOX, RIA, SER, etc.	.33	COLLECTING SITE	Pointer	4	NULL	63
18	63.04	CHEM, HEM, TOX, RIA, SER, etc.	.34	HOST UID	FreeText	NULL	NULL	63
19	63.04	CHEM, HEM, TOX, RIA, SER, etc.	.342	ORDERING SITE UID	FreeText	NULL	NULL	63
20	63.04	CHEM, HEM, TOX, RIA, SER, etc.	.35	ORDERED TEST	SubFile	NULL	63.07	63
21	63.04	CHEM, HEM, TOX, RIA, SER, etc.	.99	COMMENT	SubFile	NULL	63.041	63
22	63.04	CHEM, HEM, TOX, RIA, SER, etc.	2	GLUCOSE	Numeric	NULL	NULL	63
23	63.04	CHEM, HEM, TOX, RIA, SER, etc.	3	UREA NITROGEN	Numeric	NULL	NULL	63
24	63.04	CHEM, HEM, TOX, RIA, SER, etc.	4	CREATININE	Numeric	NULL	NULL	63
25	63.04	CHEM, HEM, TOX, RIA, SER, etc.	5	SODIUM	FreeText	NULL	NULL	63
26	63.04	CHEM, HEM, TOX, RIA, SER, etc.	6	POTASSIUM	Numeric	NULL	NULL	63

Variable pointers are a pointer without explicit (i.e. variable) destination. Variable pointers are the data equivalent of going to the airport, putting down a grand, and telling the ticketing agent “Surprise me” when the agent asks for your destination.

Variable pointer destinations are not known *a priori*. CDW Architecture must research the issue with the CDW VistA Extract Team by looking at the code.

In this example, this location can point to a:

1. Ward Location – In hospital
2. Location – “Clinic”
3. Room Bed – Bed in medical center

Meta.FileManField

Background

This is a listing of all the fields, their datatypes, and definitions where available in DHCP. The version that is displayed is PUG (Puget Sound) – Station 663.

Table Structure – Meta.FileManField

Column	DataType	Definition
SourceAttributeSID	Int	The PK for the table. Defined as a unique field within a file in FileMan
SourceEntitySID	Int	An FK to Meta.FileManFile
FileNumber	Varchar(50)	The File Number in FileMan.
FieldNumber	Varchar(50	The Field Number in FileMan
FieldName	Varchar(50)	The Field in colloquial English in FileMan. Note that these don't have to make sense (look for FieldName DIE for example)
FieldType	Varchar(50)	What kind of field is it? Refer to documentation above.
FieldLength	Int	When relevant, what is the length of the field
FieldCodes	Varchar(500)	The codes inside the field. When the field is a SetofCodes, then this will be in English. If this is a

		pointer, then the fieldcodes will be an array of what is supposed to go in place of the field
PointstoFileNumber	Varchar(50)	If this is a pointer, which file does it point to? Note that this does not work for MultiplePointers
GlobalNode	Varchar(500)	The global node that this field comes from. This can be an array value.
Piece	Int	The piece where this field comes from. Note that multiple columns can come from one piece and multiple values be stuffed into one field due to the nature of hierarchical databases.
MultipleFileNumber	Varchar(50)	If the field has multiple values (subfield/subfile/multiple) then this will take you to the file where the values can be found. This is separated in the CDW
ParentFileNumber	Varchar(50)	If this file is a child, what file is the parent?
WholeNumberDigits	Int	If this field is a numeric, how many numbers to the LEFT of the decimal could be there
DecimalDigits	Int	If this field is a numeric, how many numbers to the RIGHT of the decimal could be there
Description	Varchar(8000)	The FileMan documentation of the

		field. This is missing or nonsensical sometimes.

Useful Queries Using This Table

```
--Change to the right database
USE CDWWork
GO

--Look at the top 1000 Fields

SELECT TOP 1000 *
FROM Meta.FileManField
;

--Find files dealing with LOINC in the field descriptions

SELECT *
FROM Meta.FileManField
WHERE Description LIKE '%LOINC%'
```

```
;

--Find files that have fields with "drug" in
them

SELECT *
FROM Meta.FileManField
WHERE FieldName LIKE '%DRUG%'
;

--Find all columns in File 9000010 - VISIT

SELECT *
FROM Meta.FileManField
WHERE FileNumber LIKE '9000010'
;

--Find all pointers within a file (using File 50
as the example file)

SELECT *
FROM Meta.FileManField
```

```
WHERE FieldType = 'POINTER'
;
```

Field Types Within VistA

IEN (Internal Entry Number)

The Internal Entry Number is a specialized field that records a unique identifier for each record. In most cases, this is what makes a record unique at a VistA instance. Most of the time, this is a cardinal integer, however, there are files that use decimal numbers as part of the IEN, and there cases where the IEN uses letters. In the CDW, this is modeled as a varchar(50) in SQL Server.

Freetext

Freetext entries allow any combination of letters, numbers, and symbols to be embedded within it up to a 255 character limit. These columns are treated as strings in programming. Using the limit placed in in FileManField, freetext columns are varchars up to 255 characters.

Numeric

Numbers are rendered in VistA as fixed-width decimal or numeric types. Numeric fields are stored as integers for whole number entries, and as fixed-width decimal or numeric values. Because ANSI M does not follow IEEE 754 guidance on floating point numbers, the CDW does not store any numbers in single or double precision format.[1]

---

[1] This is one of the features that differentiate the Cache version of M from the Kernel version. Cache allows for fp calculation, while Kernel does not as ANSI M does not conform to the IEEE 754 standard.

In case you are interested in the difficulties of storing single and double precision numbers, read http://download.oracle.com/docs/cd/E19957-01/806-3568/ncg_goldberg.html. This design choice is a catch-22 explanation. If you cannot understand the proof in Theorem 9, you probably don't want to deal with the headaches of floating point math, and the CDW choice of fixed-width numeric values will make the math you do come out the way you expect. . If you can read this and understand

Date and DateTime

Allows one to enter dates and times. DHCP allows the entry of imprecise dates (June 2011), which cannot be rendered in SQL Server 2008. CDW leaves these dates alone for translation.

Pointer

Pointers join two different files together. A pointer can be considered a foreign key to another file in relational terms. All pointers are IENs, and the CDW translates the dependency on the other file as a These fields are rendered as varchar(50) columns in CDW.

VariablePointer

These are special pointers. While the field itself contains an IEN, other logic is necessary to be able to figure out where the pointer goes.

Computed

The value of the field is calculated at the time the field is accessed. The data dictionary stores the code needed to calculate this. To replicate the behavior of a computed field, a stored procedure or trigger would have to be installed. However, this would adversely impact performance. At this time, no computed field is rendered in the CDW due to this problem.

Theorem 9 on why the approximate rounding error is always less than the 1 + 2/Even Base choice, a dba writing a numeric representation in double precision introduces all sorts of calculation issues that would cause users to be confused about why the math isn't quite precise. Of course, this doesn't stop anyone from casting a numeric as float if the use requires it in SQL….

MCode

This field contains an M routine. To replicate the behavior of a computed field, a stored procedure or trigger would have to be installed. However, this would adversely impact performance. At this time, no computed field is rendered in the CDW due to this problem. No MCode field is rendered in the CDW due to this problem.

Subfield/Subfile/Multiple

A subfield is a field that can have multiple values for any one record. FileMan parses this out. A subfile is the group of subfields. A subfile is also called a multiple. This cannot be modeled strictly in relational terms. The way this a subfield/subfile/multiple relationship is translated into a relational model is that the subfield/subfile/multiple is rendered as a parent-child relationship with the parent file IEN in every multiple entry as a one-to-many relationship.

WordProcessing

This field is used to write documents of unlimited size. This is always rendered as a varchar(8000) in the CDW.

Meta.DWSchemaV

Background

This table contains information about schema. Meta.DWSchemaV gives the schema name and a description (including securitization) for the domain. This is CDW-created, there is no equivalent in VistA for this.

Table Structure – Meta.DWSchemaV

Column	DataType	Definition
DWSchemaSID	Int	The PK for the table. Defined as a unique schema.
DWSchema	Varchar(50)	The schema name
SchemaDescription	Varchar(2000)	A description of the schema

Useful Queries Using This Table

```
--Change to the right database

USE CDWWork

GO


--The schema and what is supposed to be in there
SELECT DWSchemaSID, DWSchema, SchemaDescription
FROM Meta.DWSchemaV
;
```

Meta.DWTableV

Background

This table contains information about CDW tables. Meta.DWTableV gives the table name, and some useful information about special columns (partition date, cutoffdate).

Table Structure – Meta.DWTableV

Column	DataType	Definition
DWTableGUID	Varchar(32)	The PK for the table. Defined as a unique table.
DWSchemaSID	Int	FK to Meta.DWSchemaV. Defined as a unique schema. All schema names are unique.
DWSchema	Varchar(50)	Schema Name
DWTable	Varchar(50)	Table Name. All table names are unique.
DWTableDescription	Varchar(max)	Nontechnical description of the table.
DWTableTechnicalDescription	Varchar(max)	Technical description of the table (transformations and other unusual data features)
DWPartitionKey	Varchar(50)	The table

		partitioning date (how the table is broken up physically).
CutoffField	Varchar(50)	
TableVersion	Int	Version of the table in production at the moment.
SourceEntitySID	Int	Which VistaFile did this come from?
FileName	Varchar(50)	The English File Name from VistA.
FileNumber	Varchar(50)	The File Nmber from VistA.
SourceSystemSID	Int	The source system for the data (VistA, HDR, NPCD…)

Useful Queries Using This Table

```
--Change to the right database

USE CDWWork

GO


--The translated CDW to Source for Tables
SELECT DWSchemaSID, DWSchema, DWTableGUID, DWTab
le,
DWTableDescription, DWTableTechnicalDescription,
DWPartitionKey, CutoffField, TableVersion,
SourceEntitySID, FileName, FileNumber, SourceSys
temSID
FROM Meta.DWTableV
```

;

Partition and Cutoff Dates

All tables aside from dimensions have two specialized dates: a cutoff date and a partition date. The cutoff date is the date which a fact record may or may not be included. Unless otherwise stated, all records will be cutoff at the start of Fiscal Year 2000 (10/1/1999) and all NULL date records will be taken. The partition date is the date which is used to subdivide the table into manageable portions for queries. Searching using the partition date may be faster than without the date. This date is chosen to be the most searched upon date in the table. Ordinarily, the cutoff date and the partition date are the same. Here is a list of the fact tables, their cutoff dates, and their partition dates.

Meta.DWFieldV

Background

This table contains information about fields (columns). Meta.DWFieldV gives the table name, SQL data type, some useful information about special columns (partition date, cutoffdate, primary key (PK), and alternate key (AK)).

Table Structure – Meta.DWFieldV

Column	DataType	Definition
DWFieldGUID	Varchar(32)	PK for this table. This defines a field (column) within a table.
DWFieldName	Varchar(50)	This is the field name. Field names are unique within a table, but are not necessarily unique through the warehouse.
DWTableGUID	Varchar(32)	FK to Meta.DWTableV. Defined as a unique table.
DWSchemaSID	Int	FK to Meta.DWSchemaV. Defined as a unique schema.
DWSchema	Varchar(50)	Schema Name
DWTable	Varchar(50)	Table Name
DWFieldDescription	Varchar(max)	Nontechnical

		description of the table.
DWFieldTechnicalDescription	Varchar(max)	Technical description of the table (transformations and other unusual data features)
DWFieldDescriptionAuthor	Varchar(50)	Who wrote the field description
DWFieldDescriptionDate	Smalldatetime	When was the field description written
DWFieldDataType	Varchar(50)	Datatype in SQL for the field
DWFieldLength	Int	Length for the field where relevant
DWFieldScale	Int	Scale for the field where relevant
IdentityFlag	Char(1)	Is this field an identity (almost always a PK)
NullFlag	Char(1)	Can this column be NULL? Anything flagged as a business key cannot be NULL.
DWFieldSequence	Smallint	Where in the table is this field (1 means the first position in the physical table)
PKFlag	Char(1)	Is this part the Primary Key of the table
FKFlag	Char(1)	Is this a Foreign Key to another table
BusinessKeyFlag	Char(1)	Is this part of the

		Business Key (Alternate Key) for this table
SpecialHandlingFlag	Char(1)	Is there anything particularly unusual for the programmer for this field (variable pointer)
PartitionKeyFlag	Char(1)	Is this a partition key
Computed	Varchar(2000)	If the field is computed, what is the calculation (used in dates)
PersistedFlag	Char(1)	Is the computed field persisted (i.e. is this a deterministic calculation or an ad-hoc calculation upon query)
DataDomain	Varchar(50)	Domain
DefaultValue	Varchar(500)	Default value if specified
SourceAttributeSID	Int	FK to the Meta.FileManField table
ResolvedValue	Char(1)	Is this a resolved value from a pointer?
PointedToField	Varchar(50)	Which field contains the resolved value?
PrimaryKeyEntitySID	Int	FK to the Meta.FilemanFile table
SourceSystemSID	Int	Which system does

		this field originate from
FieldCodes	Varchar(500)	What output transform or code is placed here from VistA
Filename	Varchar(50)	VistA English FileName
FileNumber	Varchar(50)	VistA English File Number
FieldName	Varchar(50)	VistA English Field Name
FieldNumber	Varchar(50)	VistA English Field Number

Useful Queries Using This Table

```
--Change to the right database
USE CDWWork
GO


--The translated CDW to Source for fields
SELECT DWSchemaSID, DWSchema, DWTableGUID, DWTab
le,
DWFieldGUID, DWFieldName, DWFieldDescription,
DWFieldDescriptionAuthor, DWFieldDescriptionDate
,
DWFieldTechnicalDescription, DWFieldDatatype,
DWFieldLength, DWFieldScale, IdentityFlag, NullF
lag,
```

```
DWFieldSequence, PKFlag, FKFlag, BusinessKeyFlag
,
SpecialHandlingFlag, PartitionKeyFlag, Computed,
PersistedFlag, DataDomain, DefaultValue,
SourceAttributeSID, ResolvedValue, PointedToFiel
d,
PrimaryKeyEntitySID, SourceSystemSID, FieldCodes
,
FileName, FileNumber, FieldName, FieldNumber
FROM Meta.DWFieldV
;
```

Meta.DWIndexV

Background

This table contains information about indices/indexes. Meta.DWIndexV gives the index name, index type, and columns that are indexed or included.

Partition keys were discussed previously as a field that is used to physically subdivide a table. The CDW uses both clustered indexes (ones that physically order the table which is the pk index for Dimensions or the cdx for Fact tables), and unclustered indexes (ones that are stored as separate physical indexes which is every other index than the pk for Dimensions or the cdx for Fact tables).

A reasonable analogy to relate to storage in SQL Server is to think of a table as a physical encyclopedia set of books. A reader can only read with the same two page outlay at any given time (a database can only look at a row at a given time). A partition key divides that book into separate physical volumes that have range references. Once partitions are made, then the clustered index builds a table of contents and physically orders the encyclopedia entries (rows) in the defined order. Because clustered indexes really define an order, there can only be one of those. Unclustered indexes related to what we think of as book indexes, they are a separate section in the book that make references to the page order (the order defined by the clustered index). Depending on what topics chosen, you may need zero, one, or more than on unclustered indexes to organize the data for faster retrieval after the clustered index is defined.

The CDW does not use the same index values as VistA (the IENs). We define our own for speed in SQL Server. What happens is for a given table, we identify all the business keys, and all of the business keys together in hierarchical order from top level to bottom level are

combined and called the alternate key index (ak index). For every unique ak value combination, we create a primary key (pk) value, which is an int for Dimensions and a bigint for Facts. This pk value is what defines the pk index. For Dimensions, we physically order those tables in pk order, so the pk index is also the clustered index. For Facts, this is not viable, so we use the partition date so that physical table is arranged in partition order. Finally, we define unclustered indexes (idx indexes) for faster searches for certain commonly queried table fields (ETLBatchID-OpCode, PatientSID, and StaffSID are the most common).

Procedurally, with the advice of the CDW DBA team, the CDW Architecture Team proposes indexes. The CDW DBA Team is then responsible for deployment.

Table Structure – Meta.DWIndexV

Column	DataType	Definition
DWIndexNamedVersionSID	Int	Unique identifier on the index.
DWTableGUID	Varchar(32)	FK to Meta.DWTableV
DWSchema	Varchar(50)	The schema for the table
DWTable	Varchar(50)	The table name
ReleaseType	Varchar(50)	Initial is architecture proposed, Customer is DBA implemented
IndexName	Varchar(128)	The index name
ClusteredFlag	Char(1)	Is this index clustered
UniqueFlag	Char(1)	Is this index supposed to only

		have unique values
IndexColumns	Varchar(max)	Columns indexed
IncludeColumns	Varchar(max)	Columns included

Useful Queries Using This Table

```
--Change to the right database

USE CDWWork

GO


--The translated Indexes that are in CDW tables
SELECT DWIndexNamedVersionSID, DWTableGUID,
DWSchema, DWTable, ReleaseType, IndexName,
ClusteredFlag, UniqueFlag, IndexColumns, Include
Columns
FROM Meta.DWIndexV
;
```

Meta.DWForeignKeyV

Background

This table contains information about foreign keys.
Meta.DWForeignKeyV is the table that keeps the foreign keys and their linking primary tables.

Table Structure – Meta.DWForeignKeyV

Column	DataType	Definition
FKSchemaName	Varchar(50)	FK Originating schema
FKTableName	Varchar(50)	FK Originating table
FKFieldName	Varchar(50)	FK Originating Field
PKSchemaName	Varchar(50)	PK Reference Schema
PKTableName	Varchar(50)	PK Reference Table
PKFieldName	Varchar(50)	PK Reference Field

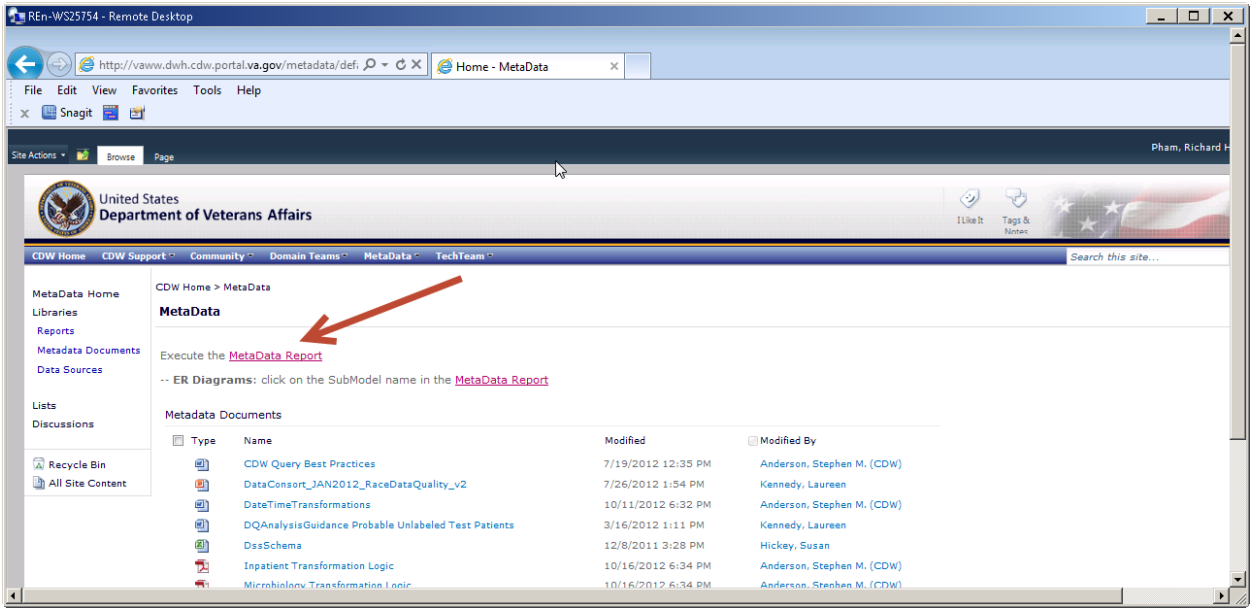
Useful Queries Using This Table

```
--Change to the right database

USE CDWWork

GO
```

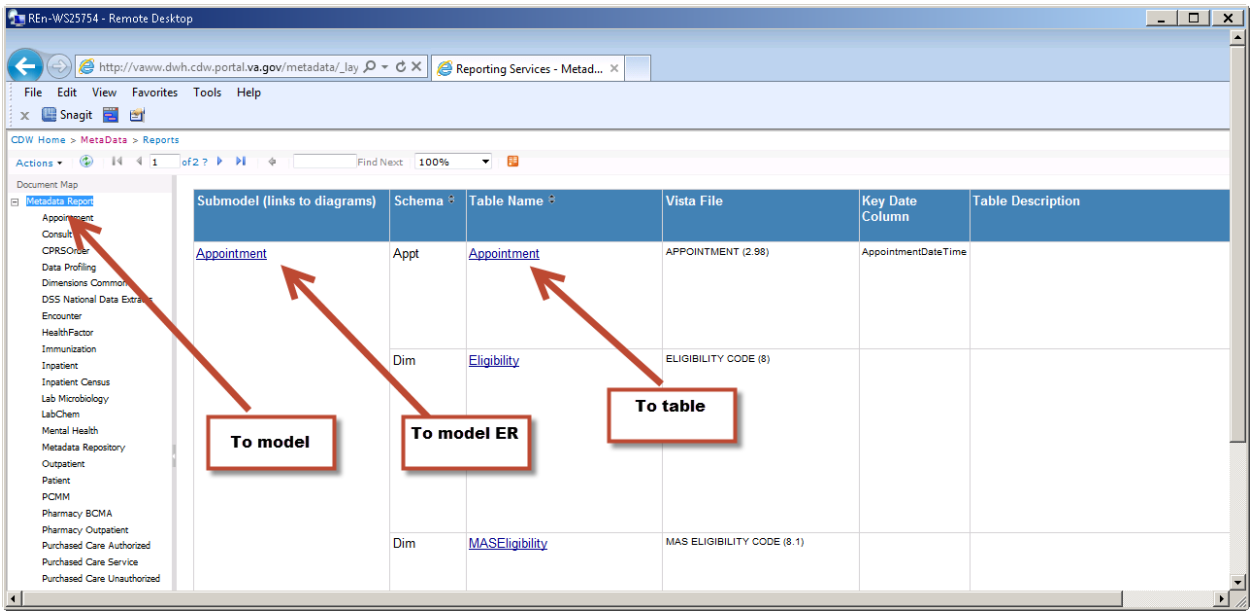
```
--The translated CDW to Source for Foreign Key
Relationships
SELECT FKSchemaName, FKTableName, FKFieldName, P
KSchemaName,
PKTableName, PKFieldName
FROM Meta.DWForeignKeyV
;
```

The Metadata Report

At http://vaww.dwh.cdw.portal.va.gov, there exists the CDW Metadata Report. To execute, click the MetaData Report link.



Once there, you can click on the Submodels in the left menu to jump to the submodel in the report. You can click on the submodel in the report itself to call up the E/R diagram. You can also click on a table to get a table column list.


REn-WS25754 - Remote Desktop
http://vaww.dwh.cdw.portal.va.gov/metadata/_lay
Reporting Services - Metad...
File Edit View Favorites Tools Help
Snagit
CDW Home > MetaData > Reports
Actions
Find Next
100%
Document Map
Metadata Report
Appointment
Consult
CPRSOrder
Data Profiling
Dimensions Common
DSS National Data Extract
Encounter
HealthFactor
Immunization
Inpatient
Inpatient Census
Lab Microbiology
LabChem
Mental Health
Metadata Repository
Outpatient
Patient
PCMM
Pharmacy BCMA
Pharmacy Outpatient
Purchased Care Authorized
Purchased Care Service
Purchased Care Unauthorized
Submodel (links to diagrams)
Schema
Table Name
Vista File
Key Date Column
Table Description
Appointment
Appt
Appointment
APPOINTMENT (2.98)
AppointmentDateTime
Dim
Eligibility
ELIGIBILITY CODE (8)
Dim
MASEligibility
MAS ELIGIBILITY CODE (8.1)
To model
To model ER
To table

Appendix I – Select * Queries Of Each Meta Table

```
/*

Queries in the Meta schema.

These are queries that you can write off to
learn about
CDW Metadata.


*/



--The translated CDW to Source for fields
SELECT DWSchemaSID, DWSchema, DWTableGUID, DWTab
le,
DWFieldGUID, DWFieldName, DWFieldDescription,
DWFieldDescriptionAuthor, DWFieldDescriptionDate
,
DWFieldTechnicalDescription, DWFieldDatatype,
DWFieldLength, DWFieldScale, IdentityFlag, NullF
lag,
DWFieldSequence, PKFlag, FKFlag, BusinessKeyFlag
,
SpecialHandlingFlag, PartitionKeyFlag, Computed,
PersistedFlag, DataDomain, DefaultValue,
SourceAttributeSID, ResolvedValue, PointedToFiel
d,
PrimaryKeyEntitySID, SourceSystemSID, FieldCodes
,
FileName, FileNumber, FieldName, FieldNumber
FROM Meta.DWFieldV
```

```
--The translated CDW to Source for Foreign Key
Relationships
SELECT FKSchemaName, FKTableName, FKFieldName, P
KSchemaName,
PKTableName, PKFieldName
FROM Meta.DWForeignKeyV
;

--The translated Indexes that are in CDW tables
SELECT DWIndexNamedVersionSID, DWTableGUID,
DWSchema, DWTable, ReleaseType, IndexName,
ClusteredFlag, UniqueFlag, IndexColumns, Include
Columns
FROM Meta.DWIndexV
;


--The schema and what is supposed to be in there
SELECT DWSchemaSID, DWSchema, SchemaDescription
FROM Meta.DWSchemaV
;

--The translated CDW to Source for Tables
SELECT DWSchemaSID, DWSchema, DWTableGUID, DWTab
le,
DWTableDescription, DWTableTechnicalDescription,
DWPartitionKey, CutoffField, TableVersion,
SourceEntitySID, FileName, FileNumber, SourceSys
temSID
FROM Meta.DWTableV
;
```

```
--All the Fields at all sites
SELECT     VistaFieldSID, VistaFileSID, VistaFil
eNumber,
VistaFieldNumber, Sta3n, VistaFileName, VistaFie
ldName,
FieldType, FieldLength, FieldCodes, PointsToFile
Number,
 GlobalNode, Piece, MultipleFileNumber, ParentFi
leNumber,
 WholeNumberDigits, DecimalDigits, VistaFieldDes
cription
FROM         Dim.VistaField
;


--All the Files at all sites
SELECT VistaFileSID, VistaFileNumber, Sta3n, Vis
taFileName,
ParentFileNumber, VistaFileDescription
FROM   Dim.VistaFile
;

--Data dictionary that CDW uses for targeting
SELECT SourceEntitySID, FileNumber, FileName,Par
entFileNumber,
ParentEntitySID, Description
FROM Meta.FileManFile
;

--Data dictionary that CDW uses for targeting
SELECT     SourceAttributeSID, SourceEntitySID,
FileNumber,
FieldNumber, FileName, FieldName, FieldType, Fie
ldLength,
FieldCodes, PointsToFileNumber, GlobalNode, Piec
e,
```

```
     MultipleFileNumber, ParentFileNumber,WholeN
umberDigits,
      DecimalDigits, Description
FROM Meta.FileManField
;
```

Appendix 2 - Basic Data Structures in VistA

This is a brief overview of the data structures in VistA for use in reading the CDW's implementation of the VistA instances' data dictionaries. For a more comprehensive introduction to the way VistA works, please read the VA Data Lifecycle presentation.

VistA Structural Information At The Programming Level

VistA has three types of variables: special, local, and global. Special variables are the system environment variables/parameters that enable operation. Local variables are stored to RAM or a memory partition which allow for user interaction and are destroyed at the end of a session. The global variables are array data structures that are stored on disk. The array has the global at its base, and subscripts which define positions of elements or subsets. Some of those subscripts result in further dependent global structures called nodes. To call values out of VistA, one must know the global node and the subscript to get the appropriate values from the array.

VistA Structural Information At the Database Management Level

Entries are the basic unit of VistA. An entry is a value stored about some characteristic. "Richard", "Pham", "05/03/1982" are values stored are entries in some area. A description of a particular entry is called a Field or Attribute. FirstName is a field/attribute that describes "Richard", LastName is a field/attribute that describes "Pham", and DateOfBirth is a field/attribute that describes "05/03/1982." A group of related entries is called a record. "Richard", "Pham, "05/03/1982" in the fields FirstName, LastName, and DateOfBirth describe the same object. A group of related records is called a File or Entity.

The aforementioned terminology is exposed to non-programmers as a Data Dictionary (DD) in the File Manager (FileMan). To look at data, one needs to know a File, a Field, and possibly a Record reference. The understand what one is looking at, the Metadata helps.