

Cerner Architectural Documentation

Created: Jan 1, 2018

Updated: Jan 1, 2018

Created in preparation for v3.00 and on Cerner Millennium 2015.01

**PROPRIETARY AND CONFIDENTIAL**

**FOR INTERNAL USE ONLY**

**NOT FOR RELEASE OUTSIDE MJ MORGAN CONSULTING**

Table of Contents

[How To Use This Document 3](#_Toc514595701)

[Formatting 3](#_Toc514595702)

[Bill Type Scenarios 4](#_Toc514595703)

[Cerner Patient Accounting Background 5](#_Toc514595704)

[Background 5](#_Toc514595705)

[Concepts 6](#_Toc514595706)

[Tables 6](#_Toc514595707)

[Retrieving Claims 10](#_Toc514595708)

[UB Codes 11](#_Toc514595709)

[Background 11](#_Toc514595710)

[How To Extract 11](#_Toc514595711)

[Value 12](#_Toc514595712)

[Occurrence 12](#_Toc514595713)

[Condition 13](#_Toc514595714)

[Future Development 13](#_Toc514595715)

[NDC Codes in Claims 14](#_Toc514595716)

[Charge Events 15](#_Toc514595717)

[Validating Table Data 16](#_Toc514595718)

[Assumptions 16](#_Toc514595719)

[Validation Queries 18](#_Toc514595720)

[Evaluating the Query Results 18](#_Toc514595721)

# How To Use This Document

## Formatting

|  |  |  |
| --- | --- | --- |
| Text | Example | Description |
| Sample | Hello world! | Descriptive narrative text to the reader. When appropriate, formatting may be used for emphasis, including *italics*, **bold**, and underline. |
| **Sample** | **m\_INFO** | Global variable by name (may be reserved words, environment variables, or declared variables). |
| Sample |  | Literal text. Text strings omit the enclosing quotes, except when part of a larger code snippet. |
| 0 |  | Integer number value |
| 0.0 |  | Float number value |
| 01-JAN-2010 00:00:00 |  | Date/Time expression |

It is presumed that when making reference to variables and literal values, that the data being provided, and the data type of the variable, match appropriately.

References to database tables and columns will always follow the convention TABLE.column where the table name is in all uppercase, and the column name is in lowercase. Table aliases will be defined the first time the table is referenced in a section of documentation. Table aliases will be consistent throughout the document (example: if PERSON\_ALIAS is aliased as PA once, this alias will be consistent in all uses of PERSON\_ALIAS. Later on, if PRSNL\_ALIAS is referenced, it will *not* use PA.).

# Bill Type Scenarios

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| From BILL\_RELTN… | Then BILL\_REC… | | | | | BR\_LONG\_BLOB\_RELTN… | LONG\_BLOB… | BENEFIT\_ORDER | | BO\_HP\_RELTN | |
| **Parent Entity** | **Bill Class** | **Bill Type** | **CM Status** | **Media Type** | **Subtype** | **Data Type** | **Parent Entity** | **Id** | **Fin Class** | **Id** | **Fin Class** |
| PFTENCNTR | Patient Statement | Patient Statement | *null* | Paper | Unassigned | 0 – Serial & Compr | BILL\_REC | **Req** | Self Pay, *null* | **Req** | *Any* |
| EDI | Print Image |
| Client Invoice | Client Invoice | Paper | Single Feed | 0 – Serial & Compr  5 – Cached PostScript | *null* | *null* | *null* | *null* |
| BENEFIT ORDER | Claim | HCFA 1450 | Pending Estimated Adjustment | Paper | UB04 | 1 – Claim Data XML  2 – Translated Claim XML  4 – Claim Data Lite XML  6 – Validation XML | PFT\_PENDING\_BILL | **Req** | *null* | **Req** | Any, null  (*not* Self Pay) |
| *various*, *null* | EDI | 837i\_5010 |
| HCFA 1500 | Pending Estimated Adjustment, *null* | Paper | CMS1500 0212 |
| *various*, *null* | EDI | 837p\_5010 |
| BO\_HP\_RELTN | Patient Statement | Patient Statement | *null* | Paper | Unassigned | 0 – Serial & Compr | BILL\_REC | **Req** | Self Pay | **Req** | Self Pay |
| EDI | Print Image |
| Claim | HCFA 1450 | Pending Estimated Adjustment | Paper | UB04 | 1 – Claim Data XML  2 – Translated Claim XML  4 – Claim Data Lite XML  6 – Validation XML | PFT\_PENDING\_BILL | **Req** | *null* | **Req** | Any  (*not* Self Pay) |
| *various*, *null* | EDI | 837i\_5010 | Any, null  (*not* Self Pay) |
| HCFA 1500 | Pending Estimated Adjustment, *null* | Paper | CMS1500 0212 |
| *various*, *null* | EDI | 837p\_5010 |

# Cerner Patient Accounting Background

## Background

In Cerner, the clinical model generally follows a hierarchical mode. That is: a patient can have zero-to-many encounters, an encounter can have zero-to-many orders, zero-to-many results, and zero-to-many charges. Some of these parent-child relationships have relationships among the children (e.g. orders to results, result to charges, etc). These are typically managed with “relation” tables to hold the primary keys of the two related items. These tables typically end with “\_RELTN”. Identifying the many-to-many relationships fairly straightforward since the primary key is typically a named foreign key in the relation table. You may have even encountered more complex relationships involving composite foreign keys, such as on CHARGE\_EVENT, CLINICAL\_EVENT, ORDER\_SENTENCE, or LONG\_TEXT. This will come in handy.

In the Patient Accounting data model, business logic changes often but re-engineering the data model and database is highly disruptive. Cerner originally used the hierarchical approach in the first generation of ProFit in the 2000’s and it resulted in a failed launch of the product. As clients requested flexible functionality, and as billing regulations became more complex, the data model itself could not adjust fast enough. Thus, the more recent new generation of Cerner Patient Accounting (CPA) has been built partially on the core data model from the first ProFit generation (denoted by tables that begin with “PFT\_”) and other tables that store dynamic relationships.

Note: Not all tables that exist in either CCLGLOS data dictionary, or that actually exist in the RDBMS, will actually by used. Many previous PFT tables are either not created in the Oracle RDBMS, or they still exist but they are not populated with data. Using CCLGLOS will not be as reliable in the CPA model as it has been in the clinical model. It is highly recommended that you have an *active live* client on CPA before building custom reports. Or, in lieu of this, have actual test cases entered into the Cerner domain. If you do not validate the data exists as you expect it, you may wind up building a report that is looking at the wrong tables.

Unlike the clinical model, tables ending in “\_RELTN” in the CPA data model frequently use the PARENT\_ENTITY\_NAME / PARENT\_ENTITY\_ID composite foreign key. Additionally, the “\_RELTN” table may have its own primary key and will store *multiple* many-to-many relationships.

For example: the table BILL\_RELTN stores relationships between a bill (e.g. claim, invoice, or patient statement) found on the BILL\_REC table (which uses CORSP\_ACTIVITY\_ID and BILL\_VRSN\_NBR as a composite primary key), and multiple other tables such as PFT\_ENCNTR, BENEFIT\_ORDER, BO\_HP\_RELTN, ACCOUNT, and PERSON. A single bill may be related to multiple of these tables, and even multiple records on the *same* foreign table! The BILL\_RELTN table has its own primary key, but this is not referenced anywhere else in Millennium. Instead, when a bill is generated, it will be added to BILL\_REC and then *all* of the relationships it has are written to BILL\_RELTN with one record for each relationship.

When embarking on a custom report in CPA, you will need to know a few things first:

1. What information do I have to start with?
2. What information do I want to end up with?
3. How can I get from here to there reliably?

*…and, to answer question 3, you must ask:*

1. What are the workflow scenarios that I should consider?

To answer question 4, you will find you are often connecting between the person asking for the report (who may be in management, finance, or somewhere other than Patient Accounting) and the Patient Accounting department.

## Concepts

NOTE: In CPA and patient accounting, the term “encounter” means a financial encounter as found on the PFT\_ENCNTR table). When referring to an encounter on the ENCOUNTER table, we will refer to this as a “clinical encounter”.

NOTE: In CPA and patient accounting, the term “account” has a specific meaning and should not be confused with either a financial encounter or clinical encounter.

Common Attributes – Like the clinical data model, you will find you frequently refer to certain key attributes in CPA. They are:

**Financial Class**, which may exist at the encounter level and also separately on each payor / insurance plan.

**Encounter Type**, which really refers to *encounter type class*, of which there are only five: Inpatient, Outpatient, Observation, Emergency, and Recurring.

**Bill Type** which may be a claim, client invoice, or patient statement. Each is a demand for payment from the guarantor, but serves a unique purpose.

Also, whether an encounter has *any* insurance will have a large effect on how the encounter is handled. When an encounter has multiple insurances, the billing can become quite complicated. Certain insurances have entirely separate rules, such as Medicaid and Workers Comp.

## Tables

BILLING\_ENTITY – this table stores a **billing entity** which is defined as an organization that provides billing services for one or many health care providers. You can think of a billing entity as being similar to an organization. In fact, a billing entity is a kind of organization. Think of a billing entity as a billing service provider (a company that provides billing services). Since hospitals typically do their own patient billing, it’s not uncommon for the health system to be the main billing entity. That same health system (which has purchased Cerner Millennium as a platform) may also provide billing services for other hospitals, clinics, and individual physicians. A billing entity can be configured to be a child of another billing entity, or to stand alone on its own. All of this can be configured within a single Millennium installation.

PFT\_ENCNTR – this table stores a **financial encounter** which relates to one and only one clinical encounter (ENCOUNTER.encntr\_id). Just like a clinical encounter will relate to one and only one org, a financial encounter will relate to one and only one billing entity. Typically, you will have only one financial encounter for each clinical encounter. There are two ways a clinical encounter will split into multiple financial encounters: charges generating to multiple billing entities in the same clinical encounter, and recurring clinical encounters that span more than one billing period (typically a calendar month). It is possible for *both* of these situations to occur in the same clinical encounter.

PFT\_CHARGE – this table stores one record for each charge that has been posted to CPA, on a one-for-one basis with the CHARGE table. The PFT\_CHARGE table relies on the existence of the CHARGE table and uses the information stored there. Thus, clients using CPA do not typically purge the CHARGE table. Charges generated by Charge Services that interface to a foreign billing system do not have a matching row in PFT\_CHARGE. It is possible for a client to have both types of charges, but each individual charge will be *either* sent outbound to a foreign system or to CPA. Refer to the processing status flag and interface file on the CHARGE table. This processing is defined on the charge tier and in CSMiscSetup.

BENEFIT\_ORDER – this table stores one record for each charge grouping. When charges post from Charge Services to CPA, they are typically grouped together into logical groupings. For example, professional charges are usually grouped separately from institutional charges, even though both kinds of charges may generate from a single clinical encounter. A client could setup CPA so that there is a single financial encounter for the clinical encounter, and provide “split billing” so that the charges are grouped separately on the same financial encounter and sent to two separate claims. Or, the client could setup CPA so that two separate financial encounters are created, each with one charge grouping and the charges are posted accordingly. There are other possibilities as well. The two indicators of a charge grouping are the CONS\_BO\_SCHED\_ID and BT\_CONDITION\_ID.

Another more common reason you would have multiple BENEFIT\_ORDER records for one financial encounter is different financial classes among the eligible payors. For example, someone who has no insurance will have just one financial class of Self Pay. However, if a patient has Medicare, then Commercial insurance, and then is responsible for the remainder, they would have at least two benefit order records (one for the insurance and one for the final self-pay financial class).

The rules for how many benefit orders you may have is complex. However from a query-building standpoint, you can consider the relationship from PFT\_ENCNTR to BENEFIT\_ORDER as being one-to-many. It is also possible for a PFT\_ENCNTR to have no benefit order, however this would mean that no insurance processing or billing has taken place (yet). You should always validate the benefit order records are active and within the beg/end effective dates.

NOTE: Many columns on the BENEFIT\_ORDER table are not routinely populated. Verify the data is present before using it. You will find most of the desired attributes on the BO\_HP\_RELTN table.

BO\_HP\_RELTN – sometimes called the “bill-hip” table by Cerner, this table stores the relationship between a benefit order and a health plan. Typically, the relationship between a benefit order and the BO\_HP\_RELTN table is one-to-one. However, it is possible for a single benefit order to have multiple related records on BO\_HP\_RELTN. This may occur when a single benefit order (charge grouping) may have multiple claim types. This is yet another scenario for CPA. For all intents, you can consider the relationship from BENEFIT\_ORDER to BO\_HP\_RELTN as being one-to-many.

Financial Class (FC) – this is not a table but an attribute found on many tables in CPA. The FC is stored on ENCOUNTER and PFT\_ENCNTR to reflect the *primary* plan’s financial class, and is stored on BO\_HP\_RELTN for the FC of *each* related plan. On BENEFIT\_ORDER, the FC columns is populated *only* for self pay, and is null for all others.

BILL\_REC – this table stores one record for each bill generated. A bill can be a claim, client invoice, or patient statement. Since each kind of bill serves a different purpose, only some of the details of the bill are stored on BILL\_REC. For more detail, you must join to other tables. However, these joins become diverse depending on the intent. The content of the bill itself is stored as a blob on LONG\_BLOB table, which can be reached by going through BR\_LONG\_BLOB\_RELTN. Bills are traditionally stored as XML, but for paper forms may be a compressed form of serial (stream) information for printing, as PostScript, or as a binary image. All of these are stored on the LONG\_BLOB table. There is *not* a table that stores the discrete line item information for a bill (PFT\_LINE\_ITEM table has been deprecated). The primary key for the BILL\_REC table is a composite key of the CORSP\_ACTIVITY\_ID and BILL\_VRSN\_NBR. You must use both to make a unique key. In general, you should always use the largest BILL\_VRSN\_NBR on the BILL\_REC table when you are provided only the CORSP\_ACTIVITY\_ID. You can do this by using a subquery in your CCL / SQL to look for the *max(bill\_vrsn\_nbr)* for that corsp\_activity\_id. This is a standard method for Cerner and will not be detrimental to performance so long as you qualify on a CORSP\_ACTIVITY\_ID.

BILL\_RELTN – this table stores the relationship between a bill on BILL\_REC table and the related tables. The relationship are intricate (see the Bill Relation Types section of this document).

ACCOUNT – this table stores the one record for each kind of a bookkeeping account. There are a handful that track the total amounts for the hospital (cash, account receivable, accounts payable, etc), but the overwhelming majority of records on the ACCOUNT table represent individual patient liabilities and their guarantors. Depending on various rules in CPA, a separate account will be created for each clinical encounter and its guarantor(s). Amounts charges, owed, paid, and written off (adjusted) will be tracked against these accounts.

TRANS\_LOG – this table stores every financial transaction that occurs in relationship to the billing entity. When charges post to the PFT\_CHARGE table, they create records on the TRANS\_LOG table so that the total dollars for the charges are increased in the respective accounts. Likewise, when a payment or adjustment is received by the hospital from the insurance company or the patient, these also create records on the TRANS\_LOG table to show money received. The TRANS\_LOG table stores only three types of transactions: Charge, Adjustment, Payment.

GL\_TRANS\_LOG – this table stores (typically) two records for every one TRANS\_LOG record. When a record is added to the TRANS\_LOG table, it must be assigned account numbers for the general ledger system (such as Lawson, Oracle, or some other major system foreign to Cerner). The account numbers assigned for the GL system are not the same accounts as in the ACCOUNT table. Instead, the GL\_TRANS\_LOG table stores what Cerner refers to as the GL *alias* which is just a systematic assignment of a four-part account number. These numbers are assigned dynamically based on various criteria, like encounter type, location, activity type, transaction type, etc. Ultimately, the purpose of the GL alias is so that when the daily transactions are sent to the GL system, they are assigned to the proper bookkeeping accounts. Think of the daily GL batch from Cerner as being the daily close at a restaurant. The bookkeeping system need not know who the customers were or what they ordered… it just needs to know how much was sold, how much cash was received, and how much inventory was used and received. This is what the GL aliasing process does.

It is possible for a transaction to be re-aliased in a subsequent process. This is called re-classing and is handled via the parent\_gl\_trans\_log\_id. A recursive query on the GL\_TRANS\_LOG table will reveal these reclassifications.

# Retrieving Claims

# UB Codes

## Background

When an insurance claim is sent to Medicare, Medicare may require certain supplemental information about the patient.

HPM has three kinds of records to receive this supplemental information, commonly called “UB” codes. Specifically:

|  |  |
| --- | --- |
| Kind of Information | HPM Record Type |
| Value | ENCUBVAL |
| Occurrence | ENCUBOCC |
| Condition | ENCUBCON |

If provided, this information is contained in a HCFA 1450 claim. A HCFA 1450 claim is submitted either on paper using a UB04 form, or electronically using the 837p format, version 5010.

Historically, claims to Medicare were only on paper, and the paper form always began with the letters “UB”, hence the common name of “UB codes” even though new installations of Cerner no longer use the UB form and submit entirely electronically.

The particular codes are contained in the 2300 loop of the HCFA 1450 claim as HI segments. The app to configure the application of which codes to use on claims in which situations is defined in profitclaimmanager.exe

## How To Extract

The desired data elements are contained within the XML structure of the claim in the 837p v5010 format.

To retrieve them, join BILL\_REC 🡪 BR\_LONG\_BLOB\_RELTN 🡪 LONG\_BLOB, retrieve the XML file where BLBR.data\_type\_flag = 2 (Translated Claim XML). Most BILL\_REC records will relate to 3 or 4 XML files via BLBR🡪LB. The data\_type\_flag is important because the other XML formats may not contain the UB codes, and even if they do, will not be in the standard 837p format.

Once you have retrieved the XML, search for the following strings:

|  |  |
| --- | --- |
| Code | XML Label |
| Value | <WPC837I5010\_2300\_HI\_ValueInformation> |
| *Sample* | <WPC837I5010\_2300\_HI\_ValueInformation> CR  <WPC837I5010\_2300\_HI01\_ValueInformation  TAB WPC837I5010\_2300\_HI0101\_ValueInformation\_CodeListQualifierCode="**BE**"  TAB WPC837I5010\_2300\_HI0102\_ValueCode="**02**"  TAB WPC837I5010\_2300\_HI0105\_ValueCodeAmount="**0.00**"  TAB /> CR  <WPC837I5010\_2300\_HI02\_ValueInformation  TAB WPC837I5010\_2300\_HI0201\_ValueInformation\_CodeListQualifierCode="**BE**"  TAB WPC837I5010\_2300\_HI0202\_ValueCode="**80**"  TAB WPC837I5010\_2300\_HI0205\_ValueCodeAmount="**7**"  TAB /> CR  </WPC837I5010\_2300\_HI\_ValueInformation> CR |
| Occurrence | <WPC837I5010\_2300\_HI\_OccurrenceInformation> |
| *Sample* | <WPC837I5010\_2300\_HI\_OccurrenceInformation> CR  <WPC837I5010\_2300\_HI01\_OccurrenceInformation  TAB WPC837I5010\_2300\_HI0101\_OccurrenceInformation\_CodeListQualifierCode="**BH**"  TAB WPC837I5010\_2300\_HI0102\_OccurrenceCode="**11**"  TAB WPC837I5010\_2300\_HI0103\_OccurrenceInformation\_DateTimePeriodFormatQualifier="**D8**"  TAB WPC837I5010\_2300\_HI0104\_OccurrenceCodeDate="**20171101**"  TAB /> CR  </WPC837I5010\_2300\_HI\_OccurrenceInformation> CR |
| Condition | <WPC837I5010\_2300\_HI\_ConditionInformation> |
| *Sample* | <WPC837I5010\_2300\_HI\_ConditionInformation> CR  <WPC837I5010\_2300\_HI01\_ConditionInformation  TAB WPC837I5010\_2300\_HI0101\_ConditionInformation\_CodeListQualifierCode="**BG**"  TAB WPC837I5010\_2300\_HI0102\_ConditionCode="**38**"  TAB />  </WPC837I5010\_2300\_HI\_ConditionInformation> CR |

Note that the 837p format contains carriage returns at the end of each line. The samples above contain additional carriage returns CR and tabs TAB to highlight the structure. The actual text is shown below (with soft line wrapping)

## Value

<WPC837I5010\_2300\_HI\_ValueInformation>

<WPC837I5010\_2300\_HI01\_ValueInformation WPC837I5010\_2300\_HI0101\_ValueInformation\_CodeListQualifierCode="BE" WPC837I5010\_2300\_HI0102\_ValueCode="02" WPC837I5010\_2300\_HI0105\_ValueCodeAmount="0.00"/>

<WPC837I5010\_2300\_HI02\_ValueInformation WPC837I5010\_2300\_HI0201\_ValueInformation\_CodeListQualifierCode="BE" WPC837I5010\_2300\_HI0202\_ValueCode="80" WPC837I5010\_2300\_HI0205\_ValueCodeAmount="7"/>

</WPC837I5010\_2300\_HI\_ValueInformation>

## Occurrence

<WPC837I5010\_2300\_HI\_OccurrenceInformation>

<WPC837I5010\_2300\_HI01\_OccurrenceInformation WPC837I5010\_2300\_HI0101\_OccurrenceInformation\_CodeListQualifierCode="BH" WPC837I5010\_2300\_HI0102\_OccurrenceCode="11" WPC837I5010\_2300\_HI0103\_OccurrenceInformation\_DateTimePeriodFormatQualifier="D8" WPC837I5010\_2300\_HI0104\_OccurrenceCodeDate="20171101"/>

<WPC837I5010\_2300\_HI02\_OccurrenceInformation WPC837I5010\_2300\_HI0201\_OccurrenceInformation\_CodeListQualifierCode="BH" WPC837I5010\_2300\_HI0202\_OccurrenceCode="18" WPC837I5010\_2300\_HI0203\_OccurrenceInformation\_DateTimePeriodFormatQualifier="D8" WPC837I5010\_2300\_HI0204\_OccurrenceCodeDate="20120101"/>

</WPC837I5010\_2300\_HI\_OccurrenceInformation>

## Condition

<WPC837I5010\_2300\_HI\_ConditionInformation>

<WPC837I5010\_2300\_HI01\_ConditionInformation WPC837I5010\_2300\_HI0101\_ConditionInformation\_CodeListQualifierCode="BG" WPC837I5010\_2300\_HI0102\_ConditionCode="38"/>

</WPC837I5010\_2300\_HI\_ConditionInformation>

Note that each kind of segment may occur multiple times (multiple value, occurrence, and/or condition codes) within a single claim. A separate ENCUBVAL, ENCUBOCC, ENCUBCON record should be created for each code.

Note that a code may also contain a begin and/or end date. The HPM record layout provides for these dates. The dates are optional within each element.

The CodeListQualifierCode correlates to the kind of data element: BE = Value Code, BH = Occurrence Code, BG = Condition Code. IT is not necessary to validate these codes relative to the XML node label.

## Future Development

Besides the three types of supplemental information provided above, there are other supplemental types of information in the 2300 loop of the HCFA 1450 and also the HCFA 1500 claim. In future development, we may extract this additional information to supply with an encounter, payor, or claim.

# NDC Codes in Claims

NDC codes are contained within the 2410 loop of the HCFA claims in XML format. See this article: <https://wiki.ucern.com/display/public/reference/National+Drug+Codes>

# Charge Events

# Validating Table Data

## Assumptions

The extraction program relies on certain assumptions on how data is organized in the database. These assumptions include:

1. Each record on a table has certain characteristics:
   1. active\_ind is 1 or 0 (not null)
   2. active\_status\_cd is 48\_ACTIVE (for active records), else any other value
   3. active\_ind and active\_status\_cd correlate.
   4. beg\_effective\_dt\_tm and end\_effective\_dt\_tm are populated
   5. beg\_effective\_dt\_tm is less than end\_effective\_dt\_tm
   6. primary key is unique and not null
2. Each record having a foreign key:
   1. The foreign key exists in only one foreign table.
   2. The foreign key is unique in the foreign table.
3. Each record having a ***composite*** foreign key:
   1. The foreign key id exists in the foreign table.
   2. The foreign key id is the primary key of the foreign table.
   3. The foreign table is consistently identified in the source table (usually PARENT\_ENTITY\_NAME).
   4. The two components of the composite foreign key are populated equally (if PARENT\_ENTITY\_ID is > 0, then PARENT\_ENTITY\_NAME is also populated, and vice versa).
4. If a table has a zero record (primary key = 0), it can be ignored and does not represent a valid record for end-users.
5. Each record having a categorical attribute : code\_value:
   1. The code value exists in the documented codeset.
      1. The code\_value may be inactive / end\_effective. This only affects whether a user can select the value in an app, not its current state in a related record. Always display foreign key code\_values regardless of their state on CODE\_VALUE table.
   2. All possible code values exist in the *same* codeset.
      1. Exceptions exist in generic columns or composite foreign keys, such as BILL\_ITEM\_MODIFIER, CHARGE\_MOD.
   3. Null (represented as zero in Discern Explorer) is an allowed value.
      1. Note that while zero and null are distinctly separate values in Oracle, Discern Explorer will convert nulls to zero in result sets, and that there is a zero record on CODE\_VALUE table. It is not typically necessary to differentiate between null and zero.
6. Each record having a categorical attribute : flag\_value:
   1. The flag value exists in the DM\_FLAGS table with same table and column name.
   2. All possible flag values exist in the same list of flag values for the same table\_name and column\_name on DM\_FLAGS.
   3. Null (represented as zero in Discern Explorer) is an allowed value.
      1. Note that while zero and null are distinctly separate values in Oracle, Discern Explorer will convert nulls to zero in result sets, and that there is a zero record on CODE\_VALUE table. It is not typically necessary to differentiate between null and zero.
7. Date/time columns:
   1. Have a realistic value (> Jan 1, 1800 and < Dec 31, 2100)
   2. May be null
      1. Unlike float columns, null date/time will return as null to Discern Explorer, not zero. Comparisons using “= 0” or “=0.0” will fail for this purpose. Comparison should instead be: “IS [NOT] NULL”
   3. May or may not include a time component.
   4. Will be stored in Oracle in the UTC value, and Discern Explorer will automatically convert to the local time zone unless otherwise handled by a UTC function.
   5. Date/time comparisons in qualification statements will require UTC conversion.
   6. Arbitrary future dates (e.g. end\_effective\_dt\_tm) may be >= Dec 30, 2100 00:00:00.
      1. Some Millennium apps use the 23:59:59 time and some do not.
   7. Systimestamp is used for current date/time, because it include the fractional time and can be used interchangeably on dq8 (date/time) and dm12 (timestamp with fractional seconds) columns.
      1. Older methods in CCL may use curdate, curtime3, or some combination of these. These are acceptable but will not work the same on timestamp dm12 columns.

Since the extract operates on these assumptions, we must sometimes verify these assumptions remain true. We have a variety of queries that will look for specific results using COUNT and GROUP BY functions to look for any instances of invalid combinations.

**Currently, we do not perform these checks on all clients, on all tables, or at all times. Likewise, the extract program does not always expect to handle these situations. If a client says they are missing data or have unusual data, you may want to run some of these queries to validate the assumptions have not been violated.**

Violation of our assumptions can occur when:

* The client installs a Millennium code package or upgrades the Millennium platform, that causes the Millennium system to store data differently. This can lead to:
  + New data is stored differently, but old data remains in the original state.
  + Both new data and old data is modified to fit the new state.
* The client begins using the application in a new or different way (sometimes as a result of new staff persons or optimization initiatives),
* The client configures a new inbound interface that causes data to be stored in a new way, or possibly changing existing data through update messages.
* An upstream system with an existing interface has a configuration change in that foreign system which causes interface messages to appear or process differently.
* The client modifies the extract configuration that now reveals an assumption violation that previously had had existed but had not been made symptomatic until now.

For the above, if you need to compare previous state to current state (after a change has occurred), you may wish to run the same query in the client’s non-production copy domain (usually Mock or “Copy prod”). While the copy prod domain may be out of date, this gives us a view into the past to see:

* Whether the assumption violation previously existed,
* Whether the extract configuration was recently changed,
* Whether a code install caused a data conversion process to modify old data to the new state.

These queries are capable of being run in any domain and then compare the results between domains. Of course, consider the quantity and purpose of data in the respective domain. A non-prod ref-only copy of production will contain very little activity data, and will likely contain a greater proportion of outlier vs normal data than a production domain.

## Validation Queries

The following are example queries to use for validating these assumptions.

Put sample queries here

## Evaluating the Query Results

The results must be considered in context for the client. For example: most of these queries will always have some outlier data that originated from the installation of the Millennium system, unusual test scenarios performed by the client, etc. It is best to consider the outliers with regard to the overall picture. To verify these outliers can be ignored, look at the data like:

* **Updt\_dt\_tm** Does it predate go-live of the system?
* **Updt\_id** Is it a real user on PRSNL table?
* **Updt\_task** What applications relate to this task? Are they in use? Does the app task exist?
* Are the other columns populated similar to the majority of data on the table?
* Can you view the record on the front-end app (will require relating it a patient, encounter, etc)? Does it look normal? Is it legitimate?