Introductions

Wednesday, September 27, 2023 9:11 AM

Emtelligent Pro documentation (Core concepts): https://docs.emtelligent.com/emtellipro-db-tutorial/v1.1.2/index.html
Username: optumrnd
Password: opT8m3nD!

Snowflake URL: https://uhg_optumcare.east-us-2.azure.snowflakecomputing.com/
ocemtel_ocdpprd@optum.com\4Ky6WwTohbaJVWHB

GitHub Repo: https://github.com/optum-care/pop-health-ccd-crosswalk/tree/main/

emtellipro_st_rd - going to be our playground

To run Snowflake query in parallel: https://doc.clickup.com/37466271/d/h/13qc4z-104/d4346819bd8d510

SDRP: https://github.com/optum-eeps/sdrp-support-intake/issues

How to Use the Multiprocessing Package in Python
++
Deployment guidance resources:

- Vakamundi Mohan Rao
- Vivek Singh

++
Ryan H.

[3:48 PM] Maharjan, David
rsa_emtelligent_key 1.p8
password: GnzLMzs6

snowsql -a uhg_optumcare.east-us-2.azure -u ocemtel_ocdpprd@optum.com --private-key-path rsa_emtelli_key.p8

Service account id needs access to: CCDA and ECDH_L1 schema
++

Costs - 2023

We will be on Optum Tech Finance rate card starting 2023

- **Credit rate**: $7.29 per Snowflake credit
- **Storage rate** : $29 per compressed TB per month
- **Egress rate** : $87 per TB transferred

Costs - 2024

Happy to announce that Optum Finance have reduced the rates significantly from 2023 rates.

- **Credit rate**: $4.99 per Snowflake credit
- **Storage rate** : $23 per compressed TB per month
- **Egress rate** : $87 per TB transferred

++

Use Azure Key Vault for Azure Databricks

Weekly status

Thursday, October 5, 2023 12:45 PM

- Weekly status, - Template updated with our details so we can refine before the meeting tomorrow
- List of broad items (aka usecases) that we are going to analyze for . i.,e rather than looking for tables and cross walk, i.e this is the reason we are looking for cross walk for i.e concept a., b, reason a, reason b (hope it makes sense)
- a detailed steps and outcomes for next week. now that we have the access

Does this makes sense doable ?

Design for processing steps to establish crosswalk across Availity and Emtelligent data

Recommend VW size and numbers to meet consumption patterns
Define table structures to house data in variant formats for easier data access and performant queries
Define approach to enhancement of "xml_document" table to include section level data
Define automation approach to execute the process at predefined intervals

- Completed and sign-off of design for the end-to-end data processing steps needed to establish the crosswalk of data across Availity and Emtelligent tables in Snowflake.
  - Identify any additional Snowflake capabilities (i.e. larger VW) needed to execute the crosswalk process in a performant and scalable manner
  - Define structured data formats (tables) for pre-processing and organizing vast amounts of data tied up in variant columns for more responsive query access for analysis.
  - Example: Emtellipro's documentstructuredmetadata table stores data in variant columns which could be pre-parsed out into separate structured data tables so parsing doesn't have to happen as part of the data query.
  - Determine the approach to enhancement of "xml_document" table to include section level data. Known options below:
  1. Done in the existing pipeline (before calling the Emtellipro Engine) when the "xml_document" table entries are created
  2. After the CCD XMLs have been run through the pipeline and the crosswalk is updated
  - Define the automation approach to execute the process at predefined intervals.
- Deploy fully signed-off crosswalk processing for automated execution
  - Coordinate any dependencies with IT team that owns the processing and load of Availity & Emtelligent data into Snowflake
  - Develop & unit test data processes to build crosswalk connecting Availity and Emtelligent data
  - Coordinate Snowflake capacity needs for on-going execution of crosswalk process
  - Perform integrated testing of end-to-end processing as needed, including IT team as needed
- Provide 2-weeks warranty period
  - Ensure productionized process is running as expected and address any potential bugs or issues as needed.

Notes

Wednesday, September 27, 2023 9:21 AM

Priorities:
1. Analysis
2. Create Crosswalk
3. Performance tune-up

++

EMR - Electronic Medical Records
SNOMED CT concepts represent clinical thoughts, ranging from |abscess| to |zygote|. Every concept has a unique numeric concept identifier. Within each hierarchy, concepts are organized from the general to the more detailed.

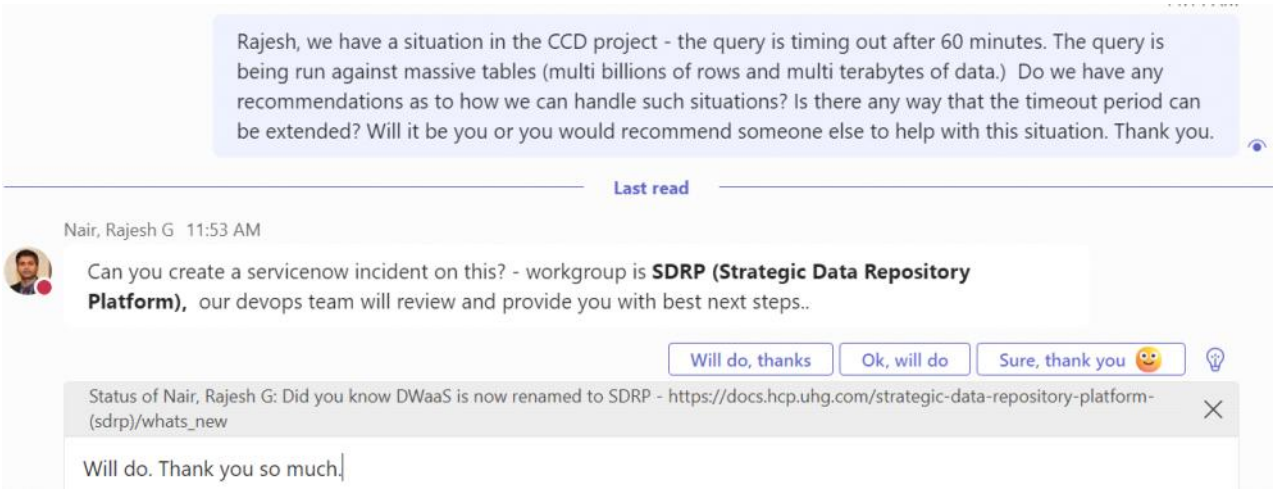
Snowflake indices and clusters: https://popsql.com/learn-sql/snowflake/how-to-create-an-index-in-snowflake
Clustering keys should be used only in case you are dealing with tables of the following properties:

- Tables containing huge amount of data (i.e. measured in terabytes).
- Data from tables are filtered on a regular basis.
  Only one clustering key per table!

Let's immediately clarify one thing: Snowflake doesn't support indices. Instead of creating or dropping an index in Snowflake, you can use clustering keys to accomplish query performance. This tutorial will show you how to define a clustering key for a particular table.

To create clustering key, use the `ALTER TABLE .. CLUSTER BY` command:

```
-- syntax
alter table table_name cluster by (column1, column2, .., columnN);

-- create clustering key on one column
alter table active_users
cluster by (id)

-- create clustering on multiple columns
alter table active_users
cluster by (id, active)
```



++

Our Understanding

- **'Facts' extracted from structured and unstructured data**
- A **'fact' must contain an identified patient, a date, and a Snomed concept or ICD-10 Diagnosis**.
- **Extraction of EMR unstructured data is done via the Emtelligent Natural Language Processing (NLP) Engine**.
- The Emtelligent NLP Engine extracts unstructured and some structured data from CCDs
- Emtelligent parsing results from the CCD data are stored in Emtelligent database tables on multiple Snowflake databases.
- **The Emtelligent Engine's date extraction is not robust enough**
- Same CCD is run thru **Availity, which normalizes, enrich, and optimize** structured clinical data, including dates
- **Availity** Engine **results** are **written to the Clinical Data Mart 'composition' tables**.
- **Availity dates** in the databases **seem to link up well with claims dates of service**.
- Therefore, there is a need to build a crosswalk process so that appropriate "dates" available in the Availity data can be used in conjunction with the "facts" from Emtelligent.
- HCE Population Health Analytics team has **attempted to build a prototype of the crosswalk process** but was **not able to fully run** the process due to **timeout that is enforced after 60 minutes** of execution. There is a **need to build a more robust and scalable process** and implement for automated on-going execution in coordination with the IT team that owns the Snowflake database.

Current state client concerns:

- We are not fully capturing all clinical data in our member's EMR
- Clinical signs/symptoms in the unstructured
- Clinical text are missed or incomplete

- NLP technology provides automated extraction and analysis of clinical data in the unstructured clinical text

- NLP can identify patterns and relationships in textual data

- Data from the clinical narrative help support RAF second-level review process.
- Member clinical insights will improve AI/ML model performance, allow improved targeting of members who can benefit from clinical interventions

Client Expectations

- **Completed and sign-off of design for the end-to-end data processing steps needed** to establish the crosswalk of data across Availity and Emtelligent tables in Snowflake.
- Identify any **additional Snowflake capabilities (i.e. larger VW) needed to execute the crosswalk** process in a performant and scalable manner
- **Define structured data formats (tables) for pre-processing and organizing data in variant columns** for more responsive query access for analysis.
  - Example: **Emtellipro's documentstructuredmetadata** table stores data in variant columns which could be pre-parsed out into separate structured data tables so parsing doesn't have to happen as part of the data query.
- **Determine the approach to enhancement of "xml_document" table to include section level data**. Known options below:
  a) Done in the existing pipeline (before calling the Emtellipro Engine) when the "xml_document" table entries are created
  b) After the CCD XMLs have been run through the pipeline and the crosswalk is updated
- **Define the automation approach** to execute the process at predefined intervals.
- **Deploy fully signed-off crosswalk processing** for automated execution
- **Coordinate any dependencies with IT team** that owns the processing and load of Availity & Emtelligent data into Snowflake
- **Develop & unit test data processes to build crosswalk** connecting Availity and Emtelligent data
- **Coordinate Snowflake capacity needs** for on-going execution of crosswalk process
- **Perform integrated testing of end-to-end processing** as needed, including IT team as needed

- Provide 2-weeks warranty period

- Ensure productionized process is running as expected and address any potential bugs or issues as needed.

++

Best practices for handling vast amounts of data in Snowflake:

- **Use multiple data models.** Snowflake supports multiple data models, including star schema, snowflake schema, and hybrid schema. Each data model has its own strengths and weaknesses, so it is important to choose the right model for your data and workload.
- **Use materialized views.** Materialized views are pre-computed views that can be used to improve query performance. They can be especially useful for queries that are frequently run on the same data.
- **Choose the right virtual warehouse size.** Virtual warehouses are Snowflake's compute resources. When you create a virtual warehouse, you specify the number of compute units and memory. It is important to choose the right size virtual warehouse for your workload. If you choose a virtual warehouse that is too small, your queries may not be able to complete in a timely manner. If you choose a virtual warehouse that is too large, you will be wasting money.
- **Use Snowpipe for continuous loading.** Snowpipe is a continuous data loading service that can be used to load data into Snowflake in real time or on a scheduled basis. This can be useful for workloads that require real-time data analysis or data warehousing.
- **Use partitioning.** Partitioning is a way to organize data into smaller, more manageable chunks. This can improve query performance and reduce storage costs.
- **Use compression.** Compression can be used to reduce the storage space required for your data. This can be especially useful for large datasets.
- **Use caching.** Caching can be used to store frequently accessed data in memory. This can improve query performance.
- **Use aggregations.** Aggregations can be used to pre-compute summary data. This can improve query performance for queries that filter and aggregate data.
- **Use role-based access control (RBAC).** RBAC can be used to control who has access to your data and what they can do with it. This is important for ensuring data security and compliance.
- **Monitor your workload.** It is important to monitor your workload to identify any performance bottlenecks. You can use Snowflake's monitoring tools to track query performance, resource utilization, and other metrics.

Here are some other tips for handling vast amounts of data in Snowflake:

- **Use a data lakehouse architecture.** A data lakehouse architecture combines the best features of data lakes and data warehouses. This can be a good option for workloads that require both high throughput and low latency.
- **Use cloud-native tools and services.** Snowflake is a cloud-native data warehouse, so you can take advantage of a variety of cloud-native tools and services to help you manage your data. For example, you can use cloud storage services to store your data and cloud orchestration services to manage your workloads.

++

Differences between operational and data warehouse models:

Operational models are designed to support real-time transaction processing. They are typically optimized for high throughput and low latency. Operational models typically store data in a normalized format, which means that the data is organized into multiple related tables. This helps to improve data integrity and reduce redundancy.

Data warehouse models are designed to support analytical queries on historical data. They are typically optimized for performance and scalability. Data warehouse models typically store data in a denormalized format, which means that the data is stored in fewer tables with more redundancy. This helps to improve query performance for analytical queries.

Here is a table that summarizes the key differences between operational and data warehouse models:

Feature	Operational Model	Data Warehouse Model
Purpose	Support real-time transaction processing	Support analytical queries on historical data
Optimization	High throughput and low latency	Performance and scalability
Data format	Normalized	Denormalized
Typical use cases	Order processing, customer relationship management (CRM), supply chain management	Business intelligence, data mining, predictive analytics

Here are some examples of operational and data warehouse models:

- **Operational model:** An order management system that stores data about customer orders, products, and inventory.
- **Data warehouse model:** A customer analytics data warehouse that stores data about customer demographics, purchase history, and product preferences.

Operational and data warehouse models can be used together to provide a complete solution for managing and analyzing data. Operational models can be used to capture and process real-time data, while data warehouse models can be used to store and analyze historical data. This allows organizations to make better decisions based on a complete view of their data.

+++

Best practices for Snowflake data models:

- **Use a star schema or a hybrid schema.** Star schemas are a good choice for most data warehouse workloads. They are easy to understand and maintain, and they perform well for analytical queries. Hybrid schemas can be used to combine the benefits of star schemas and snowflake schemas.
- **Normalize your dimension tables.** Dimension tables should be normalized to third normal form (3NF). This helps to improve data integrity and reduce redundancy.
- **Denormalize your fact tables.** Fact tables should be denormalized to support fast aggregations. This is especially important for fact tables that are used for business intelligence and reporting.
- **Use materialized views.** Materialized views are pre-computed views that can be used to improve query performance. They can be especially useful for queries that are frequently run on the same data.
- **Partition your fact tables.** Partitioning can improve query performance by allowing Snowflake to scan only the partitions that are relevant to the query.
- **Use compression and caching.** Compression can help to reduce storage costs, and caching can help to improve query performance.
- **Use role-based access control (RBAC).** RBAC can be used to control who has access to your data and what they can do with it. This is important for ensuring data security and compliance.
- **Document your data model.** It is important to document your data model so that other users can understand how your data is organized and how to query it.

Additional tips for Snowflake data modeling:

- **Use meaningful names for your tables, columns, and views.** This will make your data model more readable and maintainable.
- **Use consistent data types for your columns.** This will help to improve data integrity and reduce errors.
- **Use appropriate constraints on your tables.** Constraints can help to enforce data integrity and reduce redundancy.
- **Use a data dictionary to track the meaning of your data.** This will help to improve data quality and reduce errors.
- **Test your data model thoroughly before deploying it to production.** This will help to identify any potential problems and make sure that your data model meets your needs.

By following these best practices, you can create Snowflake data models that are efficient, scalable, and easy to use.

++

Best way to read a variant column in Snowflake:

- **Use the path syntax to access individual elements.** The path syntax is a shorthand for the GET or GET_PATH function, and it is the most efficient way to access individual elements in a variant column.
- **Use the FLATTEN function to flatten nested data structures.** The FLATTEN function can be used to flatten nested objects and arrays into a single column, which can make it easier to read and process the data.
- **Use type casting to convert the returned values to the desired data type.** By default, elements retrieved from a variant column are returned as string literals. To convert a returned element to a specific type, add the '::' operator and the target data type.

Here are some examples of how to read a variant column in Snowflake using the best practices above:

SQL

```
-- Access the first element of an object
SELECT variant_column:key1 FROM table;

-- Access the second element of an array
SELECT variant_column[1] FROM table;

-- Flatten a nested object
SELECT FLATTEN(variant_column) FROM table;

-- Convert the value of a string to an integer
SELECT variant_column::INTEGER FROM table;
```

In addition to the above, here are some other tips for reading variant columns in Snowflake:

- **Use the TYPEOF function to determine the data type of a value before casting it.** This can help to prevent errors.
- **Use the GET_PATH function to extract values from nested data structures with irregular paths.** The path syntax cannot handle all possible paths, so the GET_PATH function can be used to handle more complex cases.

Use the WHERE clause to filter the data before reading it. This can help to improve performance and reduce the amount of data that needs to be processed.

++

Duplicate rows in XML_DOCUMENT, which is a vital entity
Ryan and Dean will be the persons
Materialized view vs. dynamic table
multi-cluster warehouses, maximized vs. Auto-scale
CCDA xml_Document is the bridge between Availity and EmtelliPro output. Which data element in the Emtelligent Pro output joins the document ID/Document_root in XML_Document in CCDA?

+++++++++++++++++++++++++++++++++++++**WORKING QUERY FOR CCD CROSSWALK** ++

```
Select a.document_id, c.filepath, split_part(c.filepath, '/', -1) as filepath_leaf
, c.document_root, c.document_ext
, TO_TIMESTAMP (c.document_effective_time, 'yyyymmddHH24MISSTZHTZM' ) as document_effective_datetime
--, c.document_effective_time
, c.patient_root, c.patient_ext
, c.encounter_root, c.encounter_ext, c.create_date, c.file_md5, b.filename as emtelligent_filename_full
, split_part(b.filename, '/', -1) as emtelligent_filename_leaf, b.id as emtelligent_document_id
, parse_json(a.value):code::string  as section_code
, parse_json(a.value):class_code::string  as class_code
, parse_json(a.value):code_system::string  as section_code_system
, parse_json(a.value):code_system_name::string  as section_code_system_name
, parse_json(a.value):section::string  as section
, parse_json(a.value):label::string  as label
, parse_json(a.value):text_id::string  as text_id
, TO_TIMESTAMP ( parse_json(a.value):effective_time[0].value::string, 'yyyymmddHH24MISSTZHTZM' ) as eff_datetime
--, parse_json(a.value):effective_time[0].value::string  as eff_time_value
, parse_json(a.value):effective_time[0].low.value::string  as eff_time_low
, parse_json(a.value):effective_time[0].high.value::string  as eff_time_high
from OCDP_PRD_HEALTHCARE_ECONOMICS_DB.EMTELLIPRO_ST_FINAL.DOCUMENTSTRUCTUREDMETADATA a
Inner Join OCDP_PRD_HEALTHCARE_ECONOMICS_DB.EMTELLIPRO_ST_FINAL.DOCUMENT b
on a.document_id = b.id
Inner Join OCDP_PRD_OPTUMCARE_CORE_DB.CCDA.XML_DOCUMENT c
On split_part(b.filename, '/', -1) = split_part(c.filepath, '/', -1)
Inner Join OCDP_PRD_OPTUMCARE_CORE_DB.ECDH_L1.COMPOSITION d
On c.document_root = d.meta_source
Inner Join OCDP_PRD_OPTUMCARE_CORE_DB.ECDH_L1.COMPOSITION_CODE e
On d.composition_id = e.composition_id
Inner Join OCDP_PRD_OPTUMCARE_CORE_DB.ECDH_L1.COMPOSITION_NOTES_DH f
On e.composition_id = f.composition_id
limit 1000; --1M rows took 2m 29s
```

++**Get DOCUMENT with LATEST EFFECTIVE DATETIME** ++

```
use schema OCDP_PRD_OPTUMCARE_CORE_DB.CCDA;
CREATE or replace TEMPORARY TABLE test_table (id NUMBER, creation_date DATE);
INSERT into test_table values(1, '1/1/2010');
INSERT into test_table values(2, '1/2/2010');
INSERT into test_table values(1, '1/3/2010');
INSERT into test_table values(1, '1/4/2010');
INSERT into test_table values(2, '1/1/2009');

with
 unique_documents as
    (select id, max(creation_date) as creation_date
       from test_table where creation_date is not null group by id )
 select a.id, a.creation_date
 from test_table a
 Inner join unique_documents b
 On a.id = b.id
 And a.creation_date = b.creation_date
 order by id, creation_date;
```

+++

create or replace TABLE OCDP_PRD_OPTUMCARE_CORE_DB.ECDH_L1.COMPOSITION (
    COMPOSITION_ID VARCHAR(16777216),

META_SOURCE VARCHAR(16777216),
TYPE_CODING_SYSTEM VARCHAR(16777216),
TYPE_CODING_CODE VARCHAR(16777216),
TYPE_CODING_DISPLAY VARCHAR(16777216),
TYPE_CODING_TEXT VARCHAR(16777216),
CODE_CODING_SYSTEM VARCHAR(16777216),
CODE_CODING_CODE VARCHAR(16777216),
CODE_CODING_DISPLAY VARCHAR(16777216),
CODE_CODING_TEXT VARCHAR(16777216),
);

----

create or replace TABLE OCDP_PRD_OPTUMCARE_CORE_DB.ECDH_L1.COMPOSITION_CODE (
COMPOSITION_ID ,
PATIENT_ROOT,
PATIENT_EXT,
SRC_DOCUMENT_ROOT,
SRC_DOCUMENT_EXT ,
CODING_ID,
CODING_SYSTEM,
CODING_SYSTEM_NAME,
CODING_CODE,
CODING_DISPLAY,
TEXT

----

create or replace TABLE OCDP_PRD_OPTUMCARE_CORE_DB.ECDH_L1.COMPOSITION_NOTES_DH (
COMPOSITION_ID ,
PATIENT_ROOT ,
PATIENT_EXT,
SRC_DOCUMENT_ROOT,
SRC_DOCUMENT_EXT
++
OCDP_PRD_OPTUMCARE_CORE_DB.CCDA.XML_DOCUMENT (
FILEPATH,
CREATE_DATE ,
DOCUMENT_ROOT,
DOCUMENT_EXT,
DOCUMENT_EFFECTIVE_TIME ,
PATIENT_ROOT,
PATIENT_EXT ,
ENCOUNTER_ROOT,
ENCOUNTER_EXT,
FILE_MD5
\++
Approaches:
Create preprocessing steps, store results in temporary views
If multiple steps, create interim joins
Run the final query using interim joins outputs
Store results in dynamic tables
Serve data from dynamic tables

Databricks
Do all the cleansing, deduping, interim join steps, and final results in Databricks
Store final results in durable Snowflake tables and serve data from Snowflake

Cluster Keys
Use cluster keys on base tables, interim steps and final tables

Using limit option

Divide work into multiple chunks of work

Exploit Cache
++
Notes 10/06/2023:
XML Documents are important tables
get information from tables
Enitites must be related
Provide facts to machine learnng models
Need to put them in temporal aspects is the reason why dates are important
Aggregation level is to be determined
facts need to be putin appropriate time and place
CCDA has XML_DOCUMENT table
Entities are tied to concepts
Entities are tied to relations
Snomed entology is what we are concerned with
Most of what we're looking for would be found entities
JSON representation of the whole CCD (whole 51 tables)
E0 it patient
and E1 is the symptom e.g., heart attack
Text is the unstrutured text of the entry
ST Final Schema took 27 minutes
Brian and Bill will try to get create Materialized view capability for us
+++

Distributing Workloads That Fetch Results With the Snowflake Connector for Python

From <https://docs.snowflake.com/en/developer-guide/python-connector/python-connector-distributed-fetch>

For example:

```
with connect(...) as conn:
    with conn.cursor() as cur:
        # Execute a query.
        cur.execute('select seq4() as n from table(generator(rowcount => 100000));')

        # Return a Pandas DataFrame containing all of the results.
        table = cur.fetch_pandas_all()

        # Iterate over a list of Pandas DataFrames for result batches.
        for dataframe_for_batch in cur.fetch_pandas_batches():
          my_dataframe_processing_function(dataframe_for_batch)
```

++

Materialized views do work do not work well when too many changes. It goes unlimited. Dynamic tables is the recommendation.
Do streams on XML documents to only get changes, but only after the initial load is done.
Use create date to chunk workload

++

-- cluster by paths in variant columns
CREATE OR REPLACE TABLE T3 (t timestamp, v variant) cluster by (v:"Data":id::number);

++

1. Create a join on Emtelligent and Availity data
2. Break the big join into multiple interim views of sub joins
3. Defer joining to document/value column as much as practical
4. Experiment with different sizes of VW
5. Use multi-cluster VWs
6. Break workload into chunks by create_date
7. Distribute Workloads using Snowflake Connector for Python

++

COLUMNS TO ADD (11/16/2023):

COMPOSITION

- IDENTIFIER_SYSTEM
- IDENTIFIER_VALUE
- SRC_DOCUMENT_DTM

COMPOSITION_CODE

- CODING_SYSTEM
- CODING_CODE
- CODING_DISPLAY

COMPOSITION_NOTES_DH

- ORGTIME_VALUE
- SOURCE_ENCOUNTER_DATE

CCDA XML_DOCUMENT

- ENCOMPASSING_ENCOUNTER_EFFDT_LOW – parsed from document variant column
- ENCOMPASSING_ENCOUNTER_EFFDT_HIGH – parsed from document variant column

++

CCD Crosswalk build cutoff date: 1/1/2018

++

System Administrators - Warehouses	
☐ API_ONLY FULL	Administrator access to API_ONLY used only by APIs
☐ API_ONLY USAGE	API_ONLY used only by APIs (USAGE)
☑ OCDP Data Load 128 Warehouse FULL	Warehouse used for loading data (ETL/ELT)
☑ OCDP Data Load 32 Warehouse FULL	Warehouse used for loading data (ETL/ELT)
☑ OCDP Data Load 64 Warehouse FULL	Warehouse used for loading data (ETL/ELT)
☑ OCDP Data Load Warehouse FULL	Warehouse used for loading data into OCDP (ETL/ELT) for administrators
☐ OCDP Query 128 Warehouse FULL	Warehouse for executing queries and DDL statements

++

Reviewed CCD Crosswalk project's transition, deployment and operational artifacts to ensure minimum friction transition
Transition is ~75% complete

Leverage existing non-user account to run CCD Crosswalk build process to run as a service
Developed a parameterized Snowflake cost calculator to estimate the cost and time to process CCD initial and on-going workloadds
Migrated database and Python components to non-user account to run them as a service
Configured a Github repository to preserve CCD Crosswalk build components and associated artifacts

Updated CCD Crosswalk build components to reflect and validate CCD team's latest requests

++

To be prepared for 12/11/2023 meeting:
All union xml_documents: 145990660
All union xml_documents without the create_date: 145985303

Unmatched, discarded XM_DOcuments: 5357
Join this unique CCD set with EmtelliPro DOCUMENT table and to ProcessingDetails that contains submit_timestamp to determine if we've already processed a particular CCD to avoid duplicate CCD processing
The net CCDs to be processed are 46442 CCDs
EmtelliPro engine (latest version): 32,000 CCDs in 5.5 hours (139,000/day), revised/upgraded license

This is what I am thinking we will be presenting.

- Progress on operating model transition – Review of artifact, agreement on prod etc.
  - ~75% complete
- Findings on run time – Proposed infra suggestions / Recommendations
  - Initial workload: size: 51,000,000 CCDs, 1 VW, Size: 3X-Large, runtime: 8:00 hours, cost: $4,860
  - On-going (daily) workload: size: 130,000 CCDs, 1 VW, size: M, runtime: 20 minutes, cost: $14 ($4,200 annual (basis 300 days/year)

- Potential challenges – given that we still need to run the scalability test
  - EmtelliPro engine newer version and upgraded license still slow, e.g., 139,000 CCDs/day, 51,000,000/139,000 = 366.9065 days = 1 year

++

Joe Dale Dedupe process progress update:

In schema OCDP_PRD_OPTUMCARE_CORE_DB.CCDA two **tables** have been added

XML_DOCUMENT_DEDUP_MIN_FILEPATH
table where FILE_MD5 **is unique** (also added **IDENTITY for table**)
populates FILEPATH **with Min varchar** value
if more than 1 assoociated FILEPATH **for** given FILE_MD5,**MAX**(CREATE_DATE)
then will **add** a **row to** this **table**
XML_DOCUMENT_OAS_DEDUP_FILEPATHS
has a **reference to** its parent ablove via FILE_MD5
data correctly populated **for all** combinmed **rows** (XML_DOCUMENT_COMBINED **UNION** XML_DOCUMENT)
in above two **tables where** FILE_MD5 starts **with** 0, 3, 6 **or** 8
still populating, but slow process

++

CCD Build Run Stats using 1+M CCDs
As of: 2023-12-13

Warehouse Size: M
Chunk Size: 10K CCDs
Filtered CCDs Took: 2 minutes 2 seconds
Building Crosswalk: 25 minutes 36 seconds

Warehouse Size: M
Chunk Size: 50K CCDs
Filtered CCDs Took: 3 minutes 34 seconds
Building Crosswalk: 20 minutes 41 seconds

Warehouse Size: M
Chunk Size: 100K CCDs
Filtered CCDs Took: 2 minutes 2 seconds
Building Crosswalk: 16 minutes 45 seconds

Warehouse Size: L
Chunk Size: 10K CCDs
Filtered CCDs Took: 2 minutes 21 seconds
Building Crosswalk: 32 minutes 18 seconds

Warehouse Size: L
Chunk Size: 50K CCDs
Filtered CCDs Took: 2 minutes 27 seconds
Building Crosswalk: 17 minutes 23 seconds

Warehouse Size: L
Chunk Size: 100K CCDs
Filtered CCDs Took: 2 minutes 23 seconds
Building Crosswalk: 14 minutes 60 seconds

++

Crosswalk Column Updates and Order

NB Nystrom, Brian
To Cheema, Dave; Boinpally, Harish
Cc Ellis, William H
Retention Policy UHGInbox (90 days) Expires 3/13/2024
You replied to this message on 12/14/2023 1:05 PM.

Dave, Harish,

Here are the updates and new order for the crosswalk columns:

```
A1.FILEPATH                  as CCD_FILEPATH,
A1.src_document_root         as AV_DOCUMENT_ROOT,
A1.src_document_ext          as AV_DOCUMENT_EXT,
A1.patient_root              as AV_PATIENT_ROOT,
A1.patient_ext               as AV_PATIENT_EXT,
A1.COMPOSITION_ID            as AV_COMPOSITION_ID,
A1.coding_system             as AV_CODING_SYSTEM,
A1.coding_code               as AV_CODING_CODE,
A1.identifier_system         as AV_IDENTIFIER_SYSTEM,
A1.identifier_value          as AV_IDENTIFIER_VALUE,
A1.src_document_dtm          as AV_SRC_DOCUMENT_DTM,
A1.OBSTIME_VALUE             as AV_OBSTIME_VALUE,                   *** NEW
A1.ORGTIME_ORIGINAL_TEXT     as AV_ORGTIME_ORIGINAL_TEXT,
A1.AV_ORGTIME_VALUE          as AV_ORGTIME_VALUE,
A1.SOURCE_ENCOUNTER_DATE     as AV_SOURCE_ENCOUNTER_DATE,
A1.FILE_ID                   as AV_FILE_ID,                         *** REMOVE
A1.EM_DSM_ID                 as EM_DSM_ID,
A1.EM_DOC_ID                 as EM_DOC_ID,
xxxxx                        as EM_CLINICAL_DOCUMENT_ID_ROOT,       *** NEW
xxxxx                        as EM_CLINICAL_DOCUMENT_ID_EXT,        *** NEW
xxxxx                        as EM_CLINICAL_DOCUMENT_EFF_TIME,      *** NEW
xxxxx                        as EM_ENCOMPASING_ENCOUNTER_EFF_TIME_LOW,  *** NEW
xxxxx                        as EM_ENCOMPASING_ENCOUNTER_EFF_TIME_HIGH, *** NEW
A1.DISPLAY_NAME              as EM_DISPLAY_NAME,
A1.section_name_normalized   as EM_SECTION_NAME_NORMALIZED,
A1.SECTION_CODE              as EM_SECTION_CODE,
A1.SECTION_LABEL             as EM_SECTION_LABEL,
A1.SECTION_ID                as EM_SECTION_ID,
A1.SECTION_EXT               as EM_SECTION_EXT,
A1.text_id                   as EM_TEXT_ID,
E4.SECTION                   as EM_SECTION,
E4.ENTRY_CODE                as EM_ENTRY_CODE,
E4.ENTRY_ID                  as EM_ENTRY_ID,
E4.ENTRY_EXT                 as EM_ENTRY_EXT,
E4.ENTRY_TEXT_ID             as EM_ENTRY_TEXT_ID,
E4.EFF_TIME_VALUE            as EM_ENTRY_EFF_TIME_VALUE,            *** MODIFIED (added ENTRY)
E4.EFF_TIME_LOW              as EM_ENTRY_EFF_TIME_LOW,              *** MODIFIED (added ENTRY)
E4.EFF_TIME_HIGH             as EM_ENTRY_EFF_TIME_HIGH              *** MODIFIED (added ENTRY)
```

Please let me know if you have any questions.

Regards,

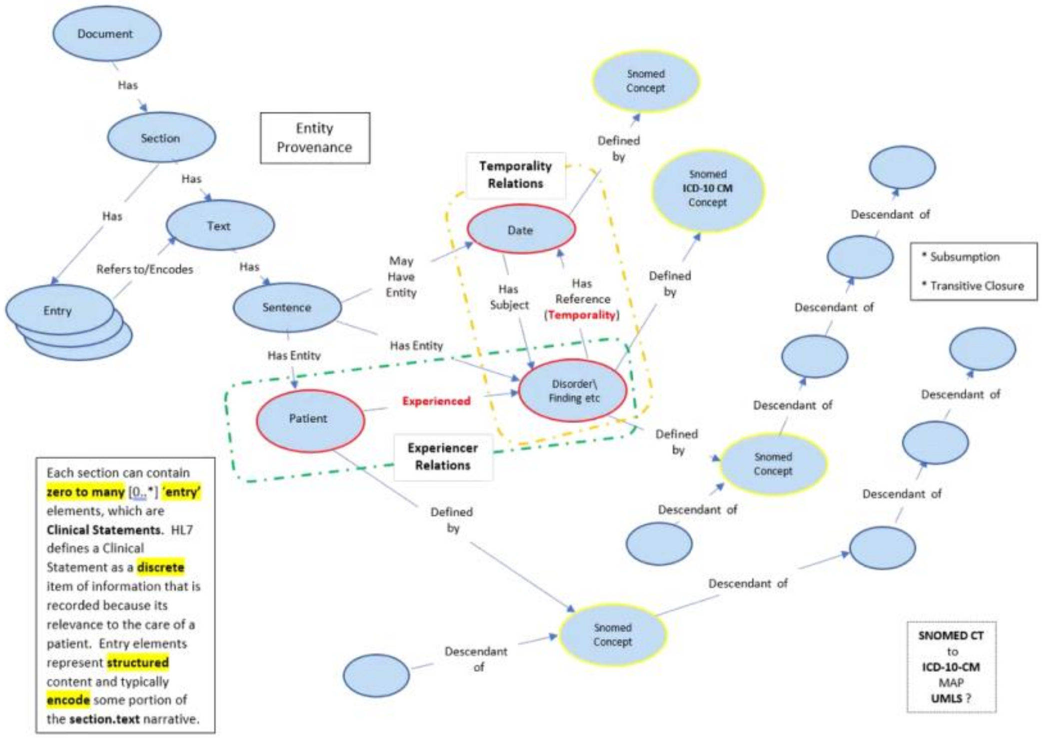
Brian Nystrom
Sr Healthcare Econ Cnslt, HCE-Population Health Analytics | Optum Health

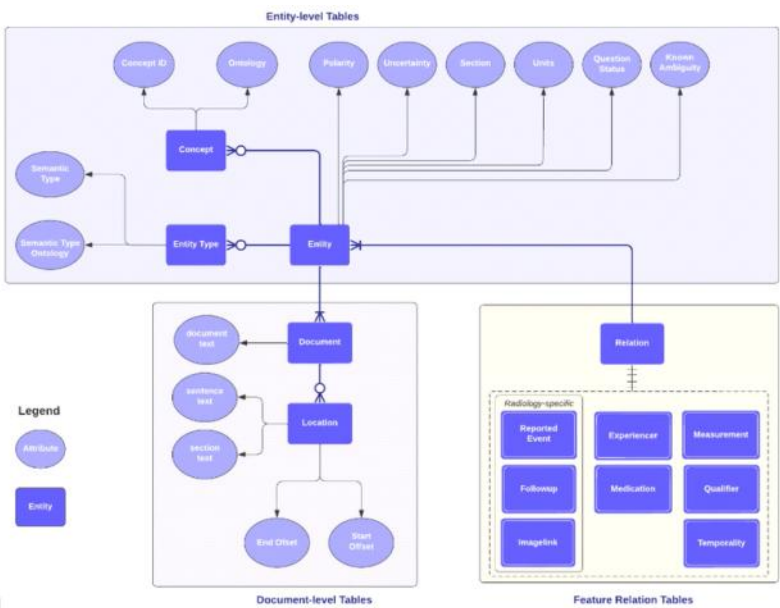
Add new: AV_OBSTIME_VALUE
Add before ORGTIME_VALUE: AV_OBSTIME_VALUE
++

Composition (Availity)			Emtelligent DocumentStructuredMetadata Table			
Compostion Table	**Composition column**		**FOR_ELT_TAG**	**CODE_ELT_TAG**	**value column (variant - JSON)**	**Comment**
COMPOSITION_NOTES_DH	src_document_root	↔	ClinicalDocument	code	PARSE_JSON(dm.value):id[0].root::string	CCD **document** level only
COMPOSITION_NOTES_DH	src_document_ext	↔	ClinicalDocument	code	PARSE_JSON(dm.value):id[0].extension::string	CCD **document** level only
COMPOSITION_CODE	coding_system	↔	section	code	PARSE_JSON(dm.value):code_system::string	
COMPOSITION_CODE	coding_code	↔	section	code	PARSE_JSON(dm.value):code::string	
COMPOSITION	identifier_system	↔	section	code	PARSE_JSON(dm.value):id[0].root::string	
COMPOSITION	identifier_value	↔	section	code	PARSE_JSON(dm.value):id[0].extension::string	

+++

TITLE





++

EM_ENCOMPASING_ENCOUNTER_EFF_TIME_LOW
EM_ENCOMPASING_ENCOUNTER_EFF_TIME_HIGH
EM_DISPLAY_NAME
EM_SECTION_NAME_NORMALIZED
EM_SECTION_CODE
EM_SECTION_LABEL
EM_SECTION_ID
EM_SECTION_EXT
EM_TEXT_ID
EM_SECTION
EM_ENTRY_CODE
EM_ENTRY_ID
EM_ENTRY_EXT
EM_ENTRY_DISPLAY_NAME
EM_ENTRY_EFF_TIME_VALUE
EM_ENTRY_EFF_TIME_LOW
EM_ENTRY_EFF_TIME_HIGH

++

Thank you Good foundation work. some suggestion (Of course I have to give, otherwise How will I show my presence right 🙂 !!)

- Added two templates, thought those would be good to explain CCD process than the one's we have used in MRIS DQ.
- Think of a theme for this, Our goal is to convey some of our work and not just reporting of what was done over the 16 weeks.

- I would suggest, themes, like ((leveraging points from Harish) and create a slide for each one. Pick a template and continue with it
    - optimization of process. (a slide for what was done, like synthesizing, paralle process etc)
    - Multi source data rationalization (utilization of right data, seamless alignment of information of temporality etc.)
    - Scalability (understanding the hardware need, optimal ruse of resource etc, configurability based on demand of volume..)
    - Maintainability (artifact, op transfer...)
    - Extendibility

Hope this makes sense.

Data Sharing

Sunday, October 1, 2023 5:24 PM

You can **share data 3 ways**.

1. Direct Share
A Direct Share allows you to share specific database objects (a share) with another account in your region. You don't need to copy or move data shared with a Direct Share.
To create a Direct Share, follow these steps:

1. Sign in to Snowsight.
2. Select **Data** > **Private Sharing**.
3. Select **Share** > **Create a Direct Share**.
4. In the **Share Data** dialog, select the database objects you want to share and specify the account you want to share them with.
5. Click **Create**.

2. Listing
A Listing allows you to offer a share and additional metadata as a data product to one or more accounts. Buyers of a Listing can then subscribe to the share and access the data.
To create a Listing, follow these steps:

1. Sign in to Snowsight.
2. Select **Data** > **Private Sharing**.
3. Select **Share** > **Create a Listing**.
4. In the **Share Data** dialog, select the database objects you want to share and specify the metadata for the data product.
5. Click **Create**.

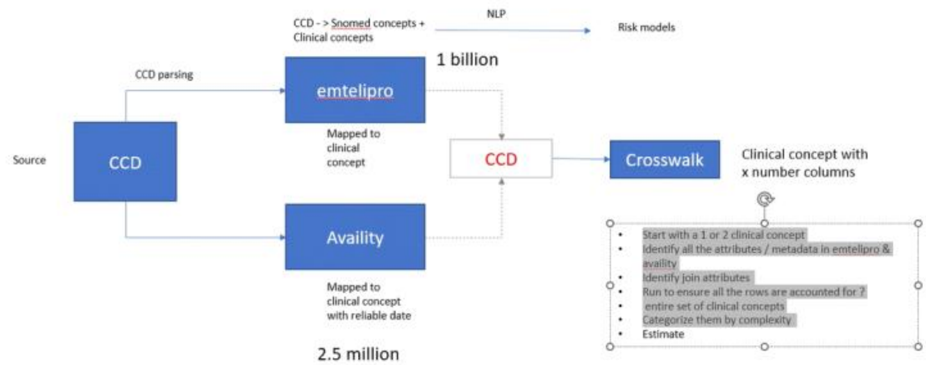
3. Data Exchange
A Data Exchange allows you to set up and manage a group of accounts and offer a share to that group. Data consumers in the group can then access the share.
To create a Data Exchange, you must request that it be provisioned and configured for your account. Once it is provisioned, you can invite members to the exchange and specify whether they can consume data, provide data, or both.
To share data with a Data Exchange, follow these steps:

1. Sign in to Snowsight.
2. Select **Data** > **Private Sharing**.
3. Select **Share** > **Share with a Data Exchange**.
4. Select the Data Exchange you want to share with and specify the database objects you want to share.
5. Click **Share**.

Once you have shared your data, data consumers can access it by querying the share.
Who pays for data sharing in Snowflake?
The data provider pays for the data storage costs, and the data consumer pays for the compute resources consumed by their queries.
Conclusion
Snowflake provides a variety of ways to share data, including Direct Shares, Listings, and Data Exchanges. The best way to share your data will depend on your specific needs and requirements.

Miscellaneous

Saturday, October 7, 2023 10:53 PM



++

Ask

Anantha

- Start with a 1 or 2 clinical concepts
- Identify all attributes/metadata in EmtelliPro and Availity
- Identify crosswalk attributes
- Run to ensure all rows are accounted for
- Run Entire set of clinical concepts
- Categorize them by complexity
- Estimate

Dave

OAS approach will adopt a multi-prong approach

Run multiple tests with multiple options

Compare results

At the end, recommend optimum solution that is best suited for size, complexity, runtime and cost challenges

Get access to CCD, EmtelliPro and Availity data sources from all relevant areas

Receive KT from CCD SME

Learn about the data layouts, complexity and nuances

Look for similarities between EmtelliPro and Availity data outputs

Identify crosswalks

Document findings and review with CCD team

Try to get an isolated environment, if possible

Run tests on new Emtelligent engine, compare dates. **Suggestion**: *start small, even with one composition. Proceed further, if results are promising.*

Look for ways to improve performance

Create cluster keys

Create materialized views

Run with Parallel clause on tables

Test results with and without Parallel clause

Subdivide the process into multiple jobs

Factors influencing: query complexity, dataset size, and current loads on the Snowflake system

Some optimization suggestions:

- Use appropriate data types for your columns.
- Materialized views and cluster keys
- Use Parallel clause to parallelize queries
- Break down complex queries into smaller, more manageable queries
- Use the `EXPLAIN` to identify bottlenecks

++

Materialized views and Parallel

Even though you can't use Parallel on materialized views, however, there are a few ways to achieve parallel processing on materialized views:

- **Use a distributed table:** A distributed table is a table that is partitioned across multiple Snowflake compute clusters. This allows you to query the table in parallel by distributing the query across the different clusters.
- **Use a materialized view with a cluster refresh:** A cluster refresh is a process that automatically updates a materialized view on a regular basis. You can configure a cluster refresh to run in parallel, which can improve the performance of the refresh process.
- **Use a Snowpipe:** A Snowpipe is a streaming data ingestion service that can be used to load data into Snowflake in parallel. You can use a Snowpipe to load data into a materialized view, which will then be available for querying.

Chosen approach will be based on your use case

1. Distributed table to query a materialized view in parallel:

SQL

```
CREATE TABLE distributed_table (
  id INT,
  name VARCHAR(255)
)
DISTRIBUTED BY(id);

CREATE MATERIALIZED VIEW mv_distributed_table
AS SELECT id, name
FROM distributed_table;
```

```
-- Query the materialized view in parallel
SELECT * FROM mv_distributed_table
DISTRIBUTED BY(id);
```

2. Using cluster refresh to update a materialized view in parallel:

```
CREATE MATERIALIZED VIEW mv_cluster_refresh
AS SELECT id, name
FROM table
CLUSTER REFRESH INTERVAL 1 DAY PARALLEL 4;
```

1. Using Snowpipe to load data into a materialized view in parallel:

```
CREATE SNOWPIPE pipe
INTO materialized_view
FROM stage
FORMAT =JSON;
```

++

Ramakrishnan, Anantha <anantha.ramakrishnan@optum.com>; Ellis, William H <william.ellis@optum.com>; Nystrom, Brian <brian.nystrom@optum.com>; Salvi, Nitin <nitin.salvi@optum.com>; Palla, Sandeep <sandeep.palla@optum.com>

++

Schedule:

Week 1

Get database access
Receive current state KT
Setup isolated environment
Populate it with Test data

Week 2

Analyze current state data
Explore commonalities between Availity EmtelliPro out comes
Create sample crosswalk
Analyze EmtelliPro (latest version) engine's output
Document findings

Week 3

Document various options
Review options with client and get their approval
Run various options and document results
Compare results from size, complexity, timing and cost perspectives
Review results with client

Week 4

Document all findings
Make recommendations
Conduct internal review of recommendations
Make necessary updates
Review findings and recommendations with client SMEs and receive their approval
Make recommendations to the CCD Crosswalk leadership and secure formal approval.

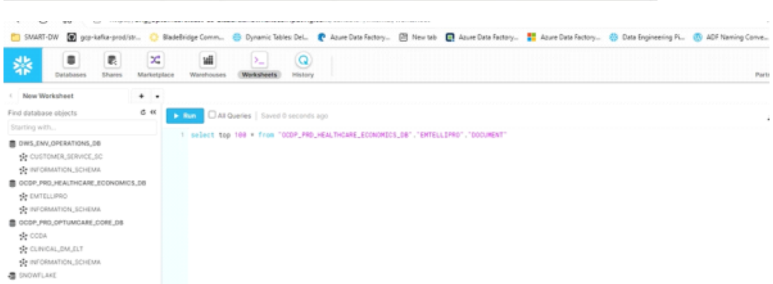
++

Rajesh,

Is Direct Data sharing allowed in our DWaaS instance? We have a situation where the outcomes of data from EmtelliPro Engine and Availity reside in an existing schema . But, we want to create a separate schema/instance where we can access this data and modify it, reshape it, and experiment with it as necessary. Please advise. If you're not the right person, do you know who might be; could you please point me to that resource? Thank you.

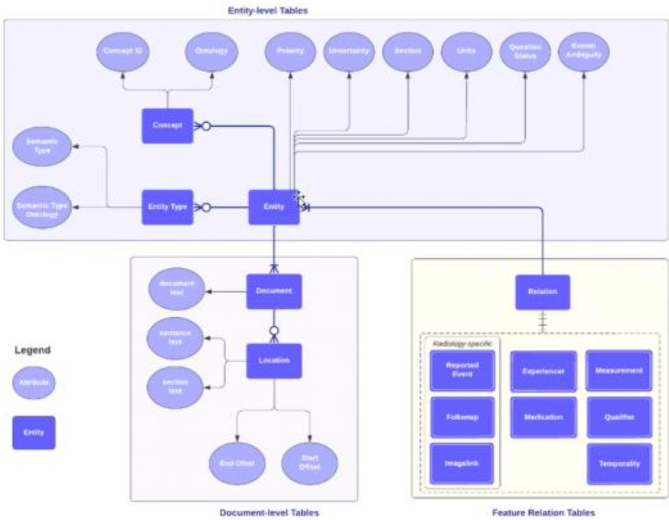
++

```
FROM
    "OCDP_PRD_OPTUMCARE_CORE_DB"."ECDH_L1"."COMPOSITION"            c
JOIN
    "OCDP_PRD_OPTUMCARE_CORE_DB"."ECDH_L1"."COMPOSITION_NOTES_DH"        cdh
ON
     c.COMPOSITION_ID = cdh.COMPOSITION_ID
JOIN
    "OCDP_PRD_OPTUMCARE_CORE_DB"."ECDH_L1"."COMPOSITION_CODE" cc
ON
    c.COMPOSITION_ID = cc.COMPOSITION_ID AND cdh.SOURCE_ID_SECTION = cc.CODING_CODE
JOIN
    "OCDP_PRD_HEALTHCARE_ECONOMICS_DB"."EMTELLIPRO_ST_RD"."X_WALK_AVAILITY_EMTELLIGENT_TEST" XWLK
ON
    cdh.SRC_DOCUMENT_ROOT = xwlk.XML_DOCUMENT_DOCUMENT_ROOT AND
    c.IDENTIFIER_SYSTEM   = xwlk.ID_ROOT                    AND
    cc.CODING_SYSTEM      = xwlk.CODE_SYSTEM                AND
    cc.CODING_CODE        = xwlk.CODE
JOIN
    -- NOTE: EMTELLIPRO schema with 5.5 million Documents and 5.2 BILLION rows in DOCUMENTSTRUCTURE
    "OCDP_PRD_HEALTHCARE_ECONOMICS_DB"."EMTELLIPRO"."DOCUMENTSTRUCTUREDMETADATA" DSM
ON
    XWLK.EMTELLIGENT_DOCUMENT_ID = DSM.DOCUMENT_ID
```

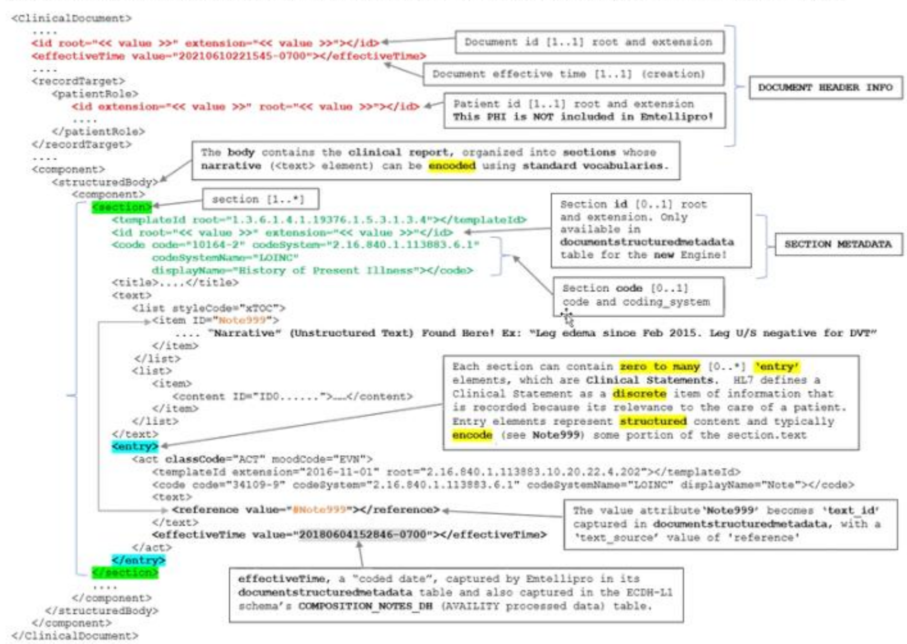


++

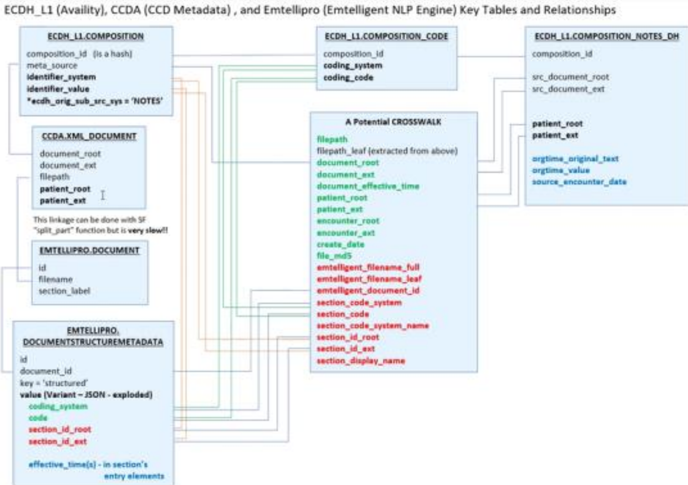
ENV (Snowflake)	DB_NAME	SCHEMA	Read Access Provided
PROD	OCDP_PRD_OPTUMCARE_CORE_DB	CCDA	Yes
PROD	OCDP_PRD_OPTUMCARE_CORE_DB	ECDH_L1	No
PROD	OCDP_PRD_HEALTHCARE_ECONOMICS_DB	EMTELLIPRO_ST_FINAL	No
	OCDP_PRD_HEALTHCARE_ECONOMICS_DB		



Example of the Main Components of a CCD XML Document from PCP EMRs. Each CCD XML is processed by both Emtellipro and Availity



ECDH_L1 (Availity), CCDA (CCD Metadata) , and Emtellipro (Emtelligent NLP Engine) Key Tables and Relationships



```
EXPLAIN
SELECT *
FROM "OCDP_PRD_OPTUMCARE_CORE_DB"."CCDA"."XML_DOCUMENT"
WHERE DOCUMENT_ROOT = '1.2.840.114350.1.13.175.2.7.8.688883.237892677'

-- **** 'Exploding' extrapolation of DOCUMENTSTRUCTUREDMETADATA *** average number of sections? or components within the sections?????
SELECT
        value,
        dm.id,
        dm.document_id,
        PARSE_JSON(dm.value):class_code::string                       as class_code,
        PARSE_JSON(dm.value):code_system::string                      as coding_system,
        PARSE_JSON(dm.value):code_system_name::string                 as coding_system_name,
        PARSE_JSON(dm.value):code::string                             as code,
        PARSE_JSON(dm.value):display_name::string                     as display_name,
        PARSE_JSON(dm.value):for_elt_tag::string                      as for_elt_tag,
        PARSE_JSON(dm.value):section::string                          as section,
        PARSE_JSON(dm.value):label::string                            as label,
        PARSE_JSON(dm.value):text_id::string                          as text_id,
        PARSE_JSON(dm.value):text_source::string                      as text_source,
        PARSE_JSON(dm.value):text::string                             as text,
        PARSE_JSON(dm.value):effective_time[0].value::string          as eff_time_value,
        PARSE_JSON(dm.value):effective_time[0].low.value::string      as eff_time_low,
        PARSE_JSON(dm.value):effective_time[0].high.value::string     as eff_time_high
FROM    "OCDP_PRD_HEALTHCARE_ECONOMICS_DB"."EMTELLIPRO_FINAL"."DOCUMENTSTRUCTUREDMETADATA" dm
JOIN    "OCDP_PRD_HEALTHCARE_ECONOMICS_DB"."EMTELLIPRO_FINAL"."DOCUMENT" d
ON      dm.document_id=d.id
WHERE
        -- Either one of the below will give 1,902 rows
        --d.filename='/tmp/2.16.840.1.113883.3.1966.100.0.0.255.65.1_122717889_06-11-2007.xml'
        d.id = '2b36cbab-ed69-4798-b714-89d7833be2c3'


select * from "OCDP_PRD_HEALTHCARE_ECONOMICS_DB"."EMTELLIPRO_ST_RD"."STRUCTURED_METADATA_AS_TABLE"
```

Column	Source	Req per CDA	Guaranteed Unique
id (Emtellipro document table)	Emtellipro document table	NA	YES
document id root	XML_DOCUMENT	YES	NO
document id extension	XML_DOCUMENT	YES	NO
patient root	XML_DOCUMENT	YES	?
patient extension	XML_DOCUMENT	YES	?
section code code	XML_DOCUMENT (NEW)	NO	
section code codeSystem	XML_DOCUMENT (NEW)	NO	
section id root	XML_DOCUMENT (NEW)	NO	
section id extension	XML_DOCUMENT (NEW)	NO	

document id (Emtellipro)

document id root
document id extension

patient root
patient extension

section code code
section code codeSystem

section id root
section id extension

++

The documentstructuredmetadata Table

The structured XML data present in CCD is extracted and stored in the **documentstructuredmetadata** table.

Each row in this table represents an instance of a **coded** term.

The table has 4 columns:

- id - unique identifier for the entry
- document_id - a foreign key that references document (id)
- key - a static string value set to structured_data
- value - JSON object representing one coded entry

+++

Sample Document root: 1.2.840.114350.1.13.175.2.7.8.688883.237892677

+++

emtelligent_document_id
section_code_system
section_code
section_code_system_name
section_id_root
section_id_ext
section_display_name

+++

Duplicate rows in XML_DOCUMENT is vital
Dean will be the person

CCDA xml_Document is the bridge between Availity and EmtelliPro output. Which data element in the EMtelligent Pro output joins the document ID/Document_root in XML_Document in CCDA?
Duplicate data in CCDA XML_DOCUMENT - multiple instances of the same DOCUMENT_ROOT id

Where are the CCD sections?
    section_id_root
    section_id_ext
    section_display_name
Any reason there is no document_root linkage between CCDA.XML_DOCUMENT and EMTELLIPRO.DOCUMENT? Is there anyway we can get it?

- Understand IT's concerns about the high compute usage and recommendations -
- Materialized view vs. dynamic table and their recommendations - dynamic table
- Multi-cluster warehouses, maximized vs. Auto-scale and their lessons learned - do use it
- Which Data element in the Emtelligent Pro output maps to Document_root in XML_Document in CCDA?
- Recommendations to process such large volume of data within given time constraints (Query is timing out after 60 minutes)? Any recommendations?
- Duplicate data in CCDA XML_DOCUMENT - multiple instances of the same DOCUMENT_ROOT id. This is a bridge between Availity and EmtelliPro engine data. Which version do we use?

Materialized views do not work well when to many changes occur. It can go unlimited. Dynamic tables is a better option.
Do streams on XML documents to only get changes
Use create date to chunk workload
Filenames and paths change over time
We may not get the raw XML_DOCUMENT file, just a link. Since that is necessity for us, they might have to provide us the raw XML via some other method.
Use Select AS query to insert data into another table
Cluster - analyze your use case before using it. However, it make sense for our use case
Create Dynamic table in the EMTELLIPRO_ST_RD schema

++

Questions for Brian:
    Where are:
        section_id_root
        section_id_ext
        section_display_name
            - Should be available on 10/10/2023 in the morning
    What do we do when there are duplicate effective datetime?
        - Question for Leanne
    what do we do when effective date is missing?
        - can't do much

What is the purpose of the crosswalk and what are really getting from the Availity?
It is more just effective datetime

++

Bill, et al,

We wanted to create a cluster key on some of the base tables. When I executed the following alter statement, I got the errorshown below.

ALTER TABLE OCDP_PRD_OPTUMCARE_CORE_DB.CCDA.XML_DOCUMENT
CLUSTER BY(CREATE_DATE, DOCUMENT_ROOT, DOCUMENT_EXT, DOCUMENT_EFFECTIVE_TIME);

SQL access control error:

```
Insufficient privileges to operate on table 'XML_DOCUMENT'
```

Also, still no permissions to create dynamic table.

SQL access control error:

```
Insufficient privileges to operate on schema 'EMTELLIPRO_ST_RD'
```

++

CREATE_DATE	COUNT(*)
2022-04-30 06:02:20	3,098,478
2022-05-01 21:19:34	43,707,368
2022-11-29 14:31:02	1,000
2022-11-29 14:31:03	1,000
2022-11-29 14:31:04	1,000
2022-11-29 14:31:22	1,000

MIN(CREATE_DATE)	...	MAX(CREATE_DATE)
2022-04-30 06:02:20.499		2022-12-14 08:59:45.424

Select count(*) from OCDP_PRD_HEALTHCARE_ECONOMICS_DB.EMTELLIPRO.DOCUMENT;

COUNT(*)
5,255,177

Select count(*) from OCDP_PRD_HEALTHCARE_ECONOMICS_DB.EMTELLIPRO.DOCUMENTSTRUCTUREDMETADATA;

COUNT(*)
5,225,950,636

Greg, would you happen to know where the query timeout 60 minutes limit come from, because the Snowflake timeout limit is two days? For example, I'm working in the OCDP_PRD_HEALTHCARE_ECONOMICS_DB / EMTELLIPRO_ST_FINAL, ECDH_L1 and CCDA schema. Here is an example of a query timeout: Query - 01af9efa-0c05-2576-010d-9201b99a47e7. Any help you could provide will be greatly appreciated. Thanks in advance.

++

a.filepath, a.filepath_LEAF, a.DOCUMENT_ROOT, a.DOCUMENT_EXT
, a.DOCUMENT_EFFECTIVE_TIME, a.PATIENT_ROOT, a.PATIENT_EXT
, a.ENCOUNTER_ROOT, a.ENCOUNTER_EXT, a.CREATE_DATE, a.FILE_MD5
, a.filename, a.filename_leaf, a.ID, a.DOCUMENT_ID

, section_code_system, section_code, section_code_system_name
, section_id_root, section_id_ext, section_display_name

section_code_system section_code section_code_system_name section_id_root section_id_ext section_display_name	**COMPOSITION_ID** VARCHAR(16777216), SRC_COMPOSITION_ID VARCHAR(16777216), PATIENT_ROOT VARCHAR(16777216), PATIENT_EXT VARCHAR(16777216), SRC_DOCUMENT_ROOT VARCHAR(16777216), SRC_DOCUMENT_EXT VARCHAR(16777216), SRC_DOCUMENT_DTM VARCHAR(16777216), MIN_HASH_KEY VARCHAR(16777216), MAX_HASH_KEY VARCHAR(16777216), SOURCE_TYPE VARCHAR(16777216), CODING_ID VARCHAR(16777216), CODING_SYSTEM VARCHAR(16777216), CODING_SYSTEM_NAME VARCHAR(16777216), CODING_VERSION VARCHAR(16777216), CODING_CODE VARCHAR(16777216), CODING_DISPLAY VARCHAR(16777216), CODING_USER_SELECTED VARCHAR(16777216), TEXT VARCHAR(16777216), LOOKUP VARCHAR(16777216), QUALIFIER VARCHAR(16777216), REFERENCE VARCHAR(16777216), TYPE VARCHAR(16777216), ECDH_ROWKEY VARCHAR(16777216), ECDH_INGEST_DTM VARCHAR(16777216), ECDH_ETL_LOAD_DTM VARCHAR(16777216), ECDH_LOAD_DTM VARCHAR(16777216), ECDH_UPDATE_DTM VARCHAR(16777216), ECDH_DATA_QUALITY_CD VARCHAR(16777216), ECDH_DATA_SECURE_RULE_LIST VARCHAR(16777216), ECDH_SUBMITTED_FILE_TYPE VARCHAR(16777216)	**A Potential CROSSWALK** filepath filepath_leaf (extracted from above) document_root document_ext document_effective_time patient_root patient_ext encounter_root encounter_ext create_date file_md5 emtelligent_filename_full emtelligent_filename_leaf emtelligent_document_id section_code_system section_code section_code_system_name section_id_root section_id_ext section_display_name

Adfasdfsd

Results of Query from composition tables to Emtelligent tables

Row	ID	DOCUMENT_ID	CODING_SYSTE	CODE	DISPLAY_NAME	FOR_ELT_TAG	SECTION	LABEL	TEXT_ID	SENTENCE	ETIME_VALUE	ETIME_LOW	ETIME_HIGH
1	90f1b969-4...	2b36cbab-e...	LOINC	10164-2	History of Pr...	section	NULL	13380d83-a...	NULL	HISTORY OF PRESENT ILLNESS...	NULL	NULL	NULL
2	03805ed2-b...	2b36cbab-e...	LOINC	51848-0	Assessments	section	NULL	0b83a5ff-78...	NULL	ASSESSMENT SECTION • Diag...	NULL	NULL	NULL

SRC_DOCUMEN	ORGTIME_ORIGINAL_TEXT	ORGTIME_VALUE	SOURCE_ENCOUNTER... ↓	CODING_SYSTE	CODING_CODE	CODING_DISPL	XWALK_XML_DC	XWALK_EMTEL
1.2.840.1143...	06/04/2018 3:28 PM PDT	2018-06-04T22:28:00.000Z	2018-06-04T22:02:43.000Z	2.16.840.1.11...	10164-2	Progress No...	1.2.840.1143...	2b36cbab-e...
1.2.840.1143...	NULL	NULL	2018-06-04T22:02:43.000Z	2.16.840.1.11...	51848-0	Visit Diagno...	1.2.840.1143...	2b36cbab-e...

documentstructuredmeta value column made into table. Rows 936 – 938 relate to example:

Row	ID	DOCUMENT_ID	CODING_SYSTE	CODE	DISPLAY_NAME	FOR_ELT_TAG	SECTION	LABEL	TEXT_ID	SENTENCE	ETIME_VALUE	ETIME_LOW	ETIME_HIGH
935	0545a756-75...	2b36cbab-ed...	LOINC	39156-5	NULL	observation	50c2d60a-0b...	5d84c10e-b1...	bmi_5599059...	• Body Mass I...	20180604221600+0000	NULL	NULL
936	90f1b969-42...	2b36cbab-ed...	LOINC	10164-2	History of Pre...	section	NULL	13380d83-a3...	NULL	HISTORY OF ...	NULL	NULL	NULL
937	bd4aba62-25...	2b36cbab-ed...	LOINC	34109-9	Note	act	13380d83-a3...	b5987e2d-2b...	Note107	Wang, Feng, ...	20180604152846-0700	NULL	NULL
938	b8ba783f-bc...	2b36cbab-ed...	LOINC	11506-3	Progress note	act	13380d83-a3...	c8db204c-34...	Note107	Wang, Feng, ...	20180604152846-0700	NULL	NULL

XML_DOCUMENT

Row	FILEPATH	DOCUMENT	CREATE_DATE	DOCUMENT_ROOT	DOCUMENT_EXT	DOCUMENT_EFFE	PATIENT_ROOT	PATIENT_EXT	ENCOUNTER_ROO	ENCOUNTER_EXT	FILE_MD5	CONTAINER
1	2022-04-27_164853_CST/38/1/2...	<ClinicalDocu...	2022-12-02 15:...	1.2.840.114350...	NULL	202106102215...	1.2.840.114350...	TEC9013011	1.2.840.114350...	66039845	d4cb31374c01...	
2	2022-04-28_160628_CST/108/12...	<ClinicalDocu...	2022-12-03 11:...	1.2.840.114350...	NULL	202106102215...	1.2.840.114350...	TEC9013011	1.2.840.114350...	66039845	0cf47b0e6899...	
3	2022-04-28_160628_CST/108/12...	<ClinicalDocu...	2022-05-01 21:...	1.2.840.114350...	NULL	202106102215...	1.2.840.114350...	TEC9013011	1.2.840.114350...	66039845	0cf47b0e6899...	sdx-src-ccd-hi...
4	2022-04-27_164853_CST/38/1/2...	<ClinicalDocu...	2022-05-01 21:...	1.2.840.114350...	NULL	202106102215...	1.2.840.114350...	TEC9013011	1.2.840.114350...	66039845	d4cb31374c01...	sdx-src-ccd-hi...

```sql
CREATE TRANSIENT TABLE SPALLA2_XWALK_TEST_3 AS

WITH CCDA_XML_DOCUMENT AS (
select * from "SPALLA2_XML_DOCUMENT_2"
  --WHERE DOCUMENT_ROOT = '2.16.840.1.113883.3.8792.100.2.200.5'
QUALIFY ROW_NUMBER() OVER (PARTITION BY FILE_MD5 ORDER BY CREATE_DATE DESC) = 1
)
,
E1 AS
(
select  FILEPATH, B.DOCUMENT_ROOT, B.DOCUMENT_EXT, B.PATIENT_ROOT, B.PATIENT_EXT, A.ID AS EMTELLIPRO_ID from "SPALLA2_DOCUMENT_2" A JOIN CCDA_XML_DOCUMENT B
ON SPLIT_PART(A.FILENAME,'/',-1) = SPLIT_PART(B.FILEPATH,'/',-1)

QUALIFY ROW_NUMBER() OVER (PARTITION BY A.FILENAME ORDER BY 1 ) = 1
)
```

++

```javascript
const snowflake = require('snowflake-sdk');

// Connection details
const connectionOptions = {
  account: 'your_account_name',
  username: 'your_user',
  password: 'your_password',
  warehouse: 'your_warehouse',
  database: 'your_database',
  schema: 'your_schema',
};

// Create a connection object
const connection = snowflake.createConnection(connectionOptions);

// Open the connection
connection.connect(function (err, conn) {
  if (err) {
    console.error('Error connecting to Snowflake: ' + err.message);
  } else {
    console.log('Successfully connected to Snowflake!');
    // Use the connection for executing queries or other operations
  }
});
```

++

```sql
Create temporary table CCDA_XML_DOCUMENT AS (
select FILEPATH, DOCUMENT_ROOT, DOCUMENT_EXT, PATIENT_ROOT, PATIENT_EXT, FILE_MD5 from "OCDP_PRD_OPTUMCARE_CORE_DB"."CCDA"."XML_DOCUMENT"
QUALIFY ROW_NUMBER() OVER (PARTITION BY FILE_MD5 ORDER BY CREATE_DATE DESC) = 1
)
,
select * from CCDA_XML_DOCUMENT limit 5;
Create temporary table E1 AS
(
select  B.FILEPATH, B.DOCUMENT_ROOT, B.DOCUMENT_EXT, B.PATIENT_ROOT, B.PATIENT_EXT, A.ID  AS EM_DOC_ID, FILE_MD5 from "OCDP_PRD_HEALTHCARE_ECONOMICS_DB"."EMTELLIPRO_ST_FINAL"."DOCUMENT" A JOIN CCDA_XML_DOCUMENT B
ON SPLIT_PART(A.FILENAME,'/',-1) = SPLIT_PART(B.FILEPATH,'/',-1)

QUALIFY ROW_NUMBER() OVER (PARTITION BY A.FILENAME ORDER BY 1 ) = 1
)
,
Create temporary table E2 AS
(
Select FILE_MD5, A.ID AS EM_DSM_ID, E1.EM_DOC_ID,value,
E1.FILEPATH,
E1.DOCUMENT_ROOT, E1.DOCUMENT_EXT, E1.PATIENT_ROOT, E1.PATIENT_EXT,
parse_json(a.value):display_name::string  as display_name,
parse_json(a.value):text::text  as text,
parse_json(a.value):label::string  as label,
parse_json(a.value):code::string  as code,
parse_json(a.value):section::string  as section,
parse_json(a.value):id[0].root::string  as id,
parse_json(a.value):id[0].extension::string  as extension,
parse_json(a.value):for_elt_tag::string  as for_elt_tag,
parse_json(a.value):code_elt_tag::string  as code_elt_tag,
```

```sql
A1.FILEPATH,
A1.COMPOSITION_ID,
A1.SRC_COMPOSITION_ID,
A1.FILE_ID,
A1.ORGTIME_ORIGINAL_TEXT,
A1.EM_DSM_ID,
A1.EM_DOC_ID,
A1.DOCUMENT_ROOT,
A1.DOCUMENT_EXT,
A1.PATIENT_ROOT,
A1.PATIENT_EXT,
A1.DISPLAY_NAME,
A1.SECTION_CODE,
A1.SECTION_LABEL,
A1.SECTION_ID,
A1.SECTION_EXT,
A1.SECTION_TEXT,
E4.SECTION,
E4.ENTRY_CODE,
E4.ENTRY_ID,
E4.ENTRY_EXT,
E4.ENTRY_TEXT,
E4.EFF_TIME_VALUE,
E4.EFF_TIME_LOW,
E4.EFF_TIME_HIGH
from A1 LEFT OUTER JOIN E4 ON A1.COMPOSITION_ID = E4.COMPOSITION_ID AND A1.SRC_COMPOSITION_ID = E4.SRC_COMPOSITION_ID AND A1.SECTION_CODE = E4.SECTION_CODE
```

```
E1.FILEPATH,
E1.DOCUMENT_ROOT, E1.DOCUMENT_EXT, E1.PATIENT_ROOT, E1.PATIENT_EXT,
parse_json(a.value):display_name::string as display_name,
parse_json(a.value):text::text as text,
parse_json(a.value):label::string as label,
parse_json(a.value):code::string as code,
parse_json(a.value):section::string as section,
parse_json(a.value):id[0].root::string as id,
parse_json(a.value):id[0].extension::string as extension,
parse_json(a.value):for_elt_tag::string as for_elt_tag,
parse_json(a.value):code_elt_tag::string as code_elt_tag,
parse_json(a.value):effective_time[0].value::string as eff_time_value,
parse_json(a.value):effective_time[0].low.value::string as eff_time_low,
parse_json(a.value):effective_time[0].high.value::string as eff_time_high
from "OCDP_PRD_HEALTHCARE_ECONOMICS_DB"."EMTELLIPRO_ST_FINAL"."DOCUMENTSTRUCTUREDMETADATA" a JOIN E1 ON E1.EM_DOC_ID = A.DOCUMENT_ID
)
,
Create temporary table E3 AS
(
select FILEPATH, EM_DSM_ID, EM_DOC_ID, DOCUMENT_ROOT, DOCUMENT_EXT, PATIENT_ROOT, PATIENT_EXT, CODE AS SECTION_CODE, LABEL AS SECTION_LABEL,
  ID AS SECTION_ID, extension AS SECTION_EXT, display_name, text AS SECTION_TEXT

  from E2 where for_elt_tag = 'section'
)
,
Create temporary table A1 as (
select N.COMPOSITION_ID, N.SRC_COMPOSITION_ID, N.FILE_ID, N.ORGTIME_ORIGINAL_TEXT, E3.*
FROM "OCDP_PRD_OPTUMCARE_CORE_DB"."ECDH_L1"."COMPOSITION_NOTES_DH" N
JOIN E3 ON
```

++

	OA
Determine and define operational model and division of responsibilities	
Identify the owner technincal management and maintenance of production scripts	
How much automation / scripting is required	dave
Additional Logs / notification necessary	Aditya
Documentation of steps / Run book for managing / reprocess/ Troubleshooting	Adita
-Crosswalk building steps	
-Convert interactive SQLs into stored procedures	Harish
-Implement workload chunking framework	
-Create Crosswalk building driver and engine	
-Automation job to run automatically and on-demand	
-Create POC to run mini workloads in parallel	
-Determine optimum size and number of VW for optimum results	
-Test CCD Crosswalk build process	
Review results with CCD team	

+++

Select count(*) from XML_DOCUMENT_COMBINED -- 74,938,038 total rows
Select count(*) from XML_DOCUMENT_COMBINED -- 73,520,090 unique rows

Select count(*) from XML_DOCUMENT -- 132,376,227 total rows
Select count(*) from XML_DOCUMENT -- 68,385,680 unique rows

Select count(*) from XML_DOCUMENT and XML_DOCUMENT_COMBINED combined together - 141,900,413 unique rows

Select count(*) from E1 (all unique XML_DOCUMENT and XML_DOCUMENT_COMBINED data combined (joined) with EmtelliPro.DOCUMENT data
-- 46, 444 rows (exactly the same number as without XML_DOCUMENT_COMBINED.

Previous shared stats:

```
Select count(*) from CCDA_XML_DOCUMENT - 68,385,680 unique rows
select count(*) from "OCDP_PRD_HEALTHCARE_ECONOMICS_DB"."EMTELLIPRO_ST_FINAL"."DOCUMENT" - 77, 641 - total rows
select count(1) from E1 --46,444 - when document is joined with CCDA xml-document
select count(*) from E2 -- - when document is joined with DOCUMENTSTRUCTUREDMETADATA on document.id =
DOCUMENTSTRUCTUREDMETADATA.document_id - 61076862
```

+++

```
create or replace TABLE OCDP_PRD_OPTUMCARE_CORE_DB.CCDA.XML_DOCUMENT_COMBINED (
	FILEPATH VARCHAR(500),
	DOCUMENT VARIANT,
	CREATE_DATE TIMESTAMP_NTZ(9),
	DOCUMENT_ROOT VARCHAR(255),
	DOCUMENT_EXT VARCHAR(255),
	DOCUMENT_EFFECTIVE_TIME VARCHAR(25),
	PATIENT_ROOT VARCHAR(255),
	PATIENT_EXT VARCHAR(255),
	ENCOUNTER_ROOT VARCHAR(255),
	ENCOUNTER_EXT VARCHAR(255),
	FILE_MD5 VARCHAR(16777216),
	CONTAINER VARCHAR(16777216)
);
create or replace TABLE OCDP_PRD_OPTUMCARE_CORE_DB.CCDA.XML_DOCUMENT (
	FILEPATH VARCHAR(500),
	DOCUMENT VARIANT,
	CREATE_DATE TIMESTAMP_NTZ(9),
	DOCUMENT_ROOT VARCHAR(255),
	DOCUMENT_EXT VARCHAR(255),
	DOCUMENT_EFFECTIVE_TIME VARCHAR(25),
	PATIENT_ROOT VARCHAR(255),
	PATIENT_EXT VARCHAR(255),
	ENCOUNTER_ROOT VARCHAR(255),
	ENCOUNTER_EXT VARCHAR(255),
	FILE_MD5 VARCHAR(16777216),
	CONTAINER VARCHAR(16777216)
);
```

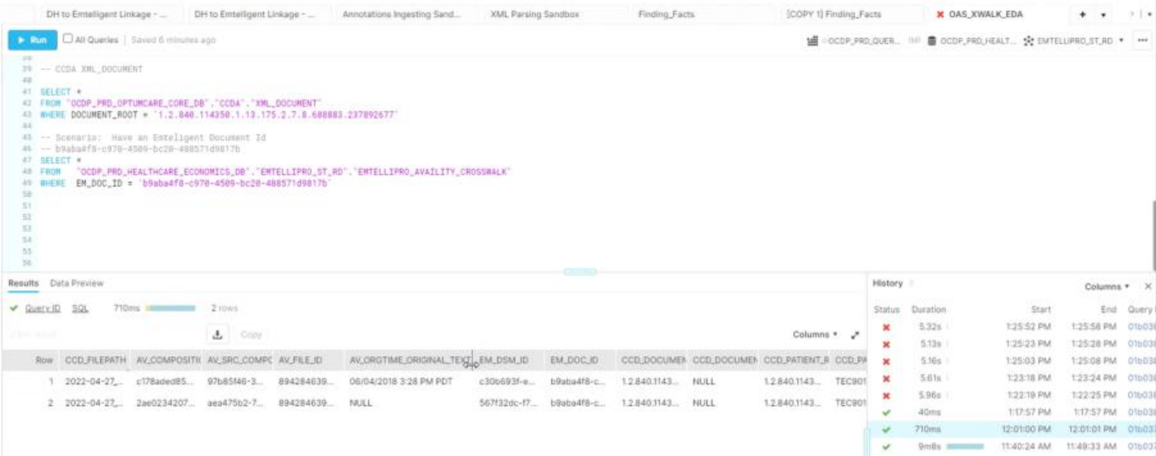
++

Harish,

The following are my comments related to SP_EmtelliPro_Availity_Crosswalk stored procedure:

- batch_size = 100000 -- Make that as a parameter, with default value = 100000
- What if we need to re-process, how do we adjust the checkpoint and re-run process?
- We should build ccda_xml_documents before the while loop, so that we won't have to rebuild that joined dataset on every loop
- SAMPLE BLOCK (1) - Educate me how this enables parallelism? Please point me to some reference information that points to it implementing parall elism
- Now we know that we should only have to worry about the presence of data in EmtelliPro Document and CCDA XML Document. Becaus e, Availity run every 3-4 hours, so it is considered to be there all the time.
- We should figure out how to filter out unprocessed CCDs from the EmtelliPro.DOCUMENT table. It will be the basis for the rest of the filtering process. Thank you.

++



Design processing steps to establish crosswalk - 10/15 - The initial design has been reviewed and accepted by CCD team with the following caveats:

1. CCD team may ask for additional attributes in the Crosswalk table (no change to the design though)
2. If Dean agrees to add 4 additional attributes (document_root, document_ext patient_root, patient_ext) to the EmtelliPro DOCUMENT table, it will eliminate the dependence on the XML_Document and XML_document_combined tables.

Define draft table structures for tables to optimize query performance - 10/23 - They have been defined and reviewed with the CCD team. The reason I was holding off on the status update was due to CCD team was going to request additional attributes.

Governance meeting to review estimates and future work based in preliminary findings 10/25 - completed. We have received the go-ahead to move forward

Define automation approach to execute the process - 10/25 - The automation job will severely depend on the logistic decisions made with Ed's team. For example, when the EmtelliPro job is completed, somebody sends a message to somebody that the job is completed, then there is not much of an automation. However, if there is a defined notification process in place, then we certainly can create an automation job.

++

You will be same as last year so use the same goals from last year's common review.

I would like to get some input from each of you on a couple of those goals. Please let me know what you believe you accomplished for the following:

1. Practice Development: this can be anything related to refining/defining service offerings and also winning new work with clients
   - refining/defining service offerings:
     - New vendor / tools identification and vetting
     - Presenting on work that was done in forums like CDLT CoP
     - Helping define / refine what we can do as part of a service (like Cloud Data Migration)
     - Etc
   - winning new work with clients
     - Identifying a new opportunity with a client
     - Working on proposals and pricing proforma
     - Helping w/ staffing (i.e. reviewing resumes and interviews)
2. Role Competency: Description of your competency in current GL as defined by the OAS Career Model (attached) and your perspective on achieving client satisfaction based on your personal interactions on projects.

1. Practice Development: this can be anything related to refining/defining service offerings and also winning new work with clients
   - refining/defining service offerings:
     - New vendor / tools identification and vetting
     - Presenting on work that was done in forums like CDLT CoP
     - Helping define / refine what we can do as part of a service (like Cloud Data Migration)
     - Etc

1. Researched, documented and presented to the leadership my findings on Data Mesh practice
2. Analyzed and documented ideal use cases for Data Mesh
3. Researched, documented and presented to the leadership my findings on Member 360. Analyzed and identified ideal clients who get can get the most out of Member 360 projects
4. Advised OAS team members on the capabilities, strengths and weaknesses of Azure Data Factory and Databricks
5. Worked on several sales support initiatives, e.g., MRIS Data Lead Transformation, MRIS Data Quality, and MRIS Data Lineage
6. Identified, Interviewed and filtered out candidates for various projects
7. Worked on MRIS DQ, MRIS Metadata Lineage, and CCD Crosswalk build projects
8. Presented detailed overview of MRIS DQ project to the CDLT CoP
9. Worked on Stepwise project performing data analytics on Databricks platform
10. All patterns, architecture collateral I developed while working on various projects, I shared with the OAS group with the expectation that they could be further refined/extended to make them into repeatable service
11. Always provide guidance or feedback to peers whenever I' am asked
12. Participate in OAS initiatives whenever an opportunity presents itself

++

map() method **supports concurrency** – **doesn't accept multiple arguments and blocks the main program** until all the processes are complete. It also **maintains the order** of **the result** (although the computation order could differ!).

apply_async - A callback function in apply_async() can be used to return the value immediately after its execution is complete. This method **maintains the order of the result** and **supports concurrency**

Unlike map(), map_async() is **non-blocking (and maintains the order of the result)**

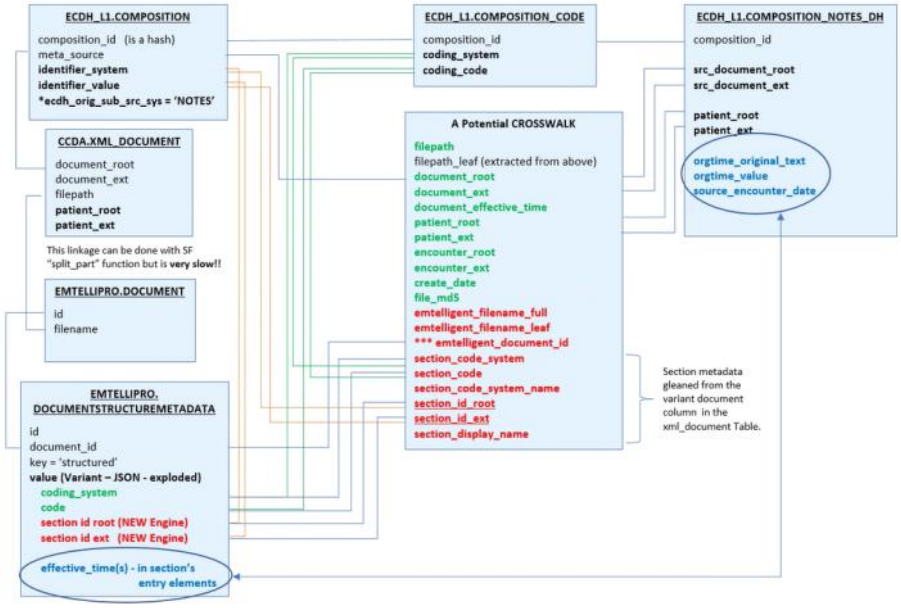
++

Just Answer: If you have further questions, you may reach a customer service agent at 800-821-2910

From <https://my-secure.justanswer.com/impress/thank-you?paymentId=67cf3117daf049b684caaad2dbca7dd9>

++

ECDH_L1 (Availity), CCDA (CCD Metadata) , and Emtellipro (Emtelligent NLP Engine) Key Tables and Relationships



What we're proposing here is that we would do about a week's worth of work (via an SOW) to update the emtelliPro client so that if you invoke the client, and pass it a JSON file with a pointer to a CCD file, then at the time of processing, the client will populate the `documentmetadata` table.

The JSON file that you'd pass to the client would look something like (all data is just made up):

```
[
  {
    "id": "1",
    "source_document_id": "1000011",
    "category": "Clinical",
    "subcategory": "CCD",
    "original_category": "Clinical",
    "original_subcategory": "CCD File",
    "description": "Continuity of Care Document",
    "filepath": "./CCDs/input_ccd_file_20230401_1000011.xml",
    "subject_id": "11110011",
    "institution": "Seal Beach Family Urgent Care",
    "hadm_id": 110001121,
    "chartdate": "2022-02-16 20:48:09",
    "author_name": "Russell Sano",
    "subject_name": "Chioma Okonkwo",
    "subject_dob": "1967-01-05 03:24:04",
    "subject_gender": "female",
    "requestor": "Eun-Young Park"
  }
]
```

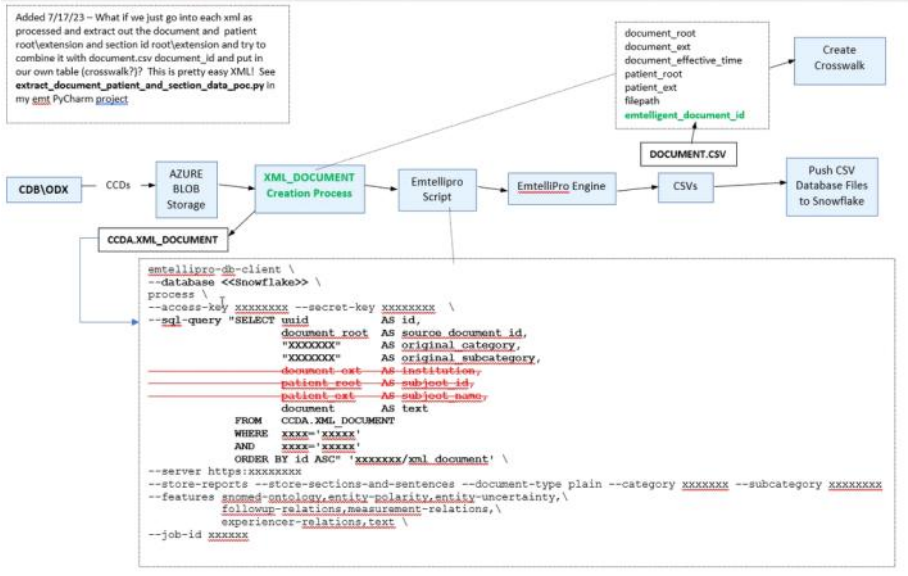
DOCUMENT ROOT

PATIENT ROOT

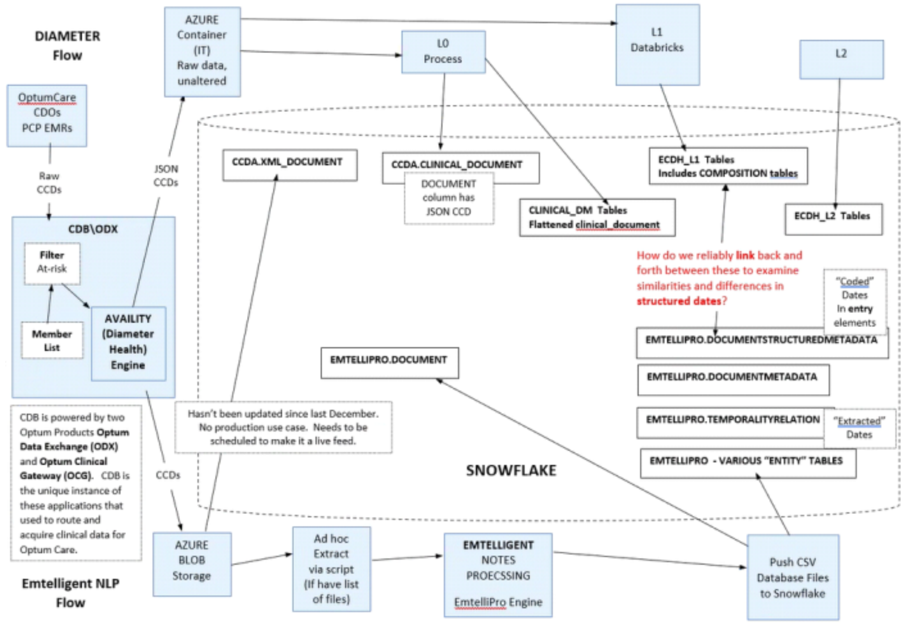
DOCUMENT EXT

PATIENT EXT

Then what about other potential document metadata like **encompassingEncounter effectiveDate** where would that now go? chartdate?



High Level Process Flow (Current) of a CCD XML Document through both Availity (old Diameter Health) and Emtelligent NLP Engine




```sql
SELECT *
FROM "OCDP_PRD_OPTUMCARE_CORE_DB"."CCDA"."XML_DOCUMENT"
WHERE DOCUMENT_ROOT = '1.2.840.114350.1.13.175.2.7.8.688883.237892677'

-- Scenario:  Have an Emteligent Document Id
-- b9aba4f8-c970-4509-bc20-488571d9817b
SELECT *
FROM   "OCDP_PRD_HEALTHCARE_ECONOMICS_DB"."EMTELLIPRO_ST_RD"."EMTELLIPRO_AVAILITY_CROSSWALK"
WHERE  EM_DOC_ID = 'b9aba4f8-c970-4509-bc20-488571d9817b'

-- GET AVAILITY JSON
SELECT *
FROM "OCDP_PRD_OPTUMCARE_CORE_DB"."CCDA"."CLINICAL_DOCUMENT"
WHERE SRC_DOCUMENT_ROOT = '1.2.840.114350.1.13.175.2.7.8.688883.237892677'


SELECT *
FROM "OCDP_PRD_OPTUMCARE_CORE_DB"."CCDA"."XML_DOCUMENT"
WHERE PATIENT_ROOT = '1.2.840.114350.1.13.175.2.7.3.688884.100'
AND   PATIENT_EXT = 'TEC9013011'




select * from "OCDP_PRD_HEALTHCARE_ECONOMICS_DB"."EMTELLIPRO_ST_FINAL"."PROCESSINGDETAILS"
where id = 'af8c52b7-1d25-492d-b1f1-fff0c6b1c128' -- text is a processing_details_id value from document table
```

CCD Document Meta Data

Column Name	Location in CCD XML	Req per CDA	
id	N/A Primary Key	N/A	
emtelligent_document_id	N/A Foreign Key to Emtelligent document table id column	N/A	
document_id_root	ClinicalDocument.id@root	Y	
document_id_ext	ClinicalDocument.id@extension	Y	
document_eff_time	ClinicalDocument. effectiveTime@value		
patient_id_root	ClinicalDocument.recordTarget.patientRole.id@root	Y	
patient_id_ext	ClinicalDocument.recordTarget.patientRole.id@extension	Y	
service_event_eff_time_low	ClinicalDocument. documentationOf.serviceEvent. effectiveTime.low@value		
service_event_eff_time_high	ClinicalDocument. documentationOf.serviceEvent. effectiveTime.high@value		
encompassing_encounter_id_root	ClinicalDocument. componentOf. encompassingEncounter.id@root		
encompassing_encounter_id_ext	ClinicalDocument. componentOf. encompassingEncounter.id@extension		
encompassing_encounter_eff_time_low	ClinicalDocument. componentOf. encompassingEncounter. effectiveTime.low@value		
encompassing_encounter_eff_time_high	ClinicalDocument. componentOf. encompassingEncounter. effectiveTime.high@value		

OTHERS?

For ClinicalDocument

```
<code code="34133-9" codeSystem="2.16.840.1.113883.6.1" codeSystemName="LOINC" displayName="Summarization of Episode Note"></code>
```

For EncompassingEncounter

```
<code code="99214" codeSystem="2.16.840.1.113883.6.12" displayName="PATIENT OFFICE VISIT,DETAILED">
```

++

To check non-serializable objects:

```python
import json
def is_jsonable(x):
   try:
      json.dumps(x)
      return True
   except:
      return False

for key,value in arg_dict.items():
   if not is_jsonable(value):
      keys_to_delete.append(key)
```

++

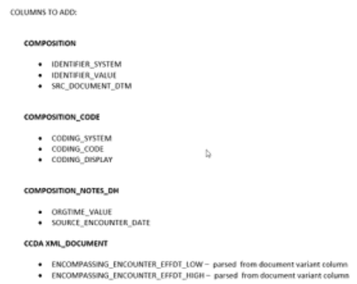
You're encountering an issue with pickling in the multiprocessing module. The error message indicates that the execute_query function or one of its arguments contains an object of type _thread.lock, which cannot be pickled. To resolve this issue, you can try the following:

Avoid Using Global Variables:

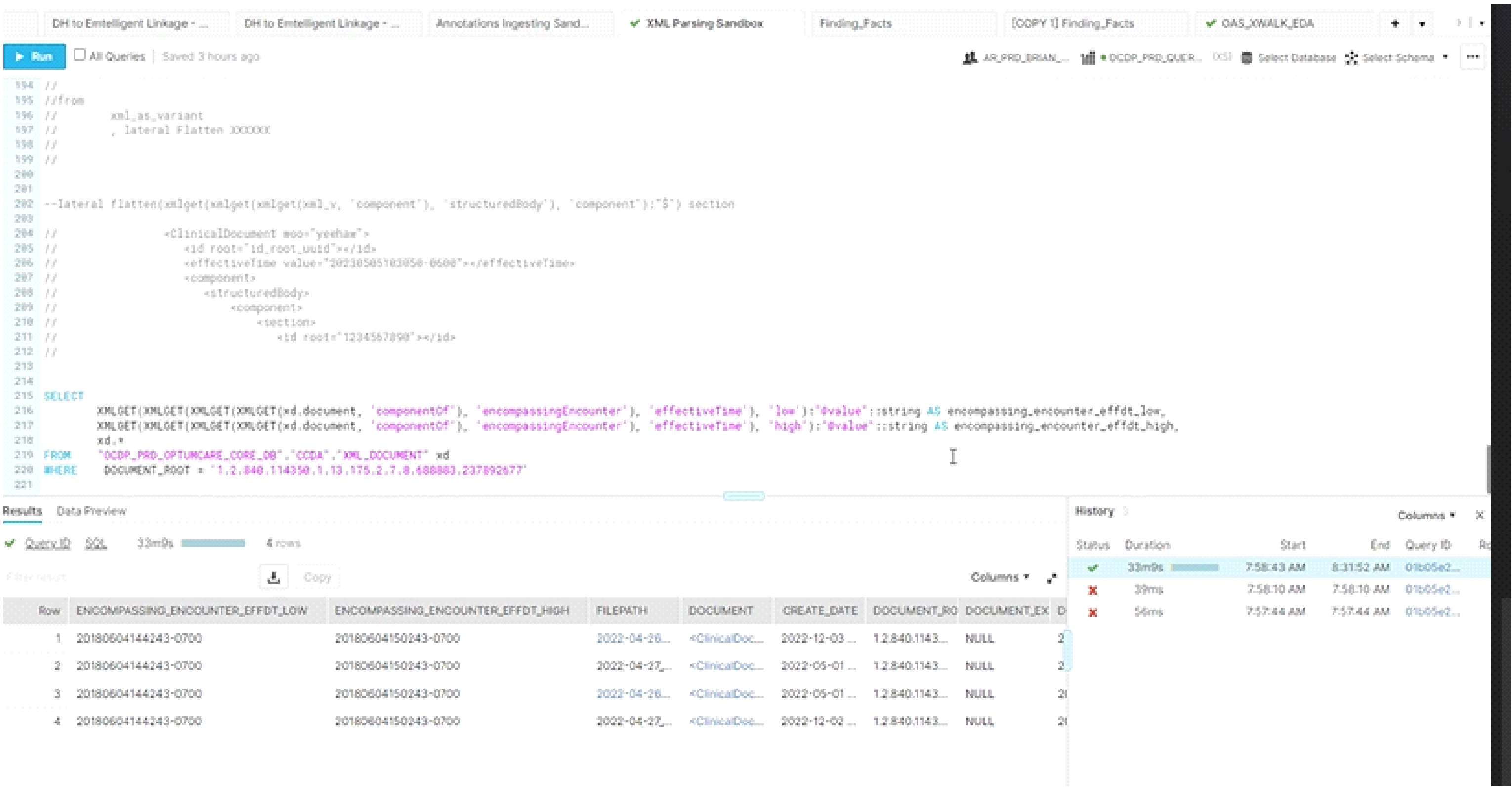
```python
from multiprocessing.pool import ThreadPool
pool = ThreadPool()

result = pool.starmap_async(execute_query, queries, callback=collect_result)
```

```
result.wait()
result.get(timeout=10)
results.append(result)
pool.close()
pool.join()
```

Check for Unpicklable Objects: Review the code within the execute_query function and its dependencies. Ensure that all objects used or referenced within the function are serializable. Objects like database connections, file handles, or thread locks might cause issues with pickling.
+++
+++++



+++



++

Python CCD Parse Simple Example

```
from lxml import etree
import json


namespaces = {
   "hl7": "urn:hl7-org:v3",
   "fhir": "http://hl7.org/fhir",
   "xsi": "http://www.w3.org/2001/XMLSchema-instance",
}

def expand_tag(tag, prefix="hl7"):
   """
   Expand a tag to use the given namespace prefix.
   """
   return "{%s}%s" % (namespaces[prefix], tag)


# -- Elements/attributes to extract
doc_eff_time = None
doc_eff_time_value = None
doc_id = None
doc_id_root = None
doc_id_ext = None
doc_patient_id = None
doc_patient_id_root = None
doc_patient_id_ext = None
```

```
# -- Read the CCD XML file and get its root element - ClinicalDocument
clinical_document_root = etree.parse('test_ccd.xml')

# -- Ensure root element name is 'ClinicalDocument'
if clinical_document_root.docinfo.root_name == 'ClinicalDocument':

   # -- See if ClinicalDocument root has a child 'effectiveTime' element
   doc_eff_time = clinical_document_root.find(expand_tag('effectiveTime'))
   # -- If so, check if effectiveTime element has value attributes=
   if doc_eff_time is not None:
      doc_eff_time_value = doc_eff_time.attrib.get('value', None)

   # -- See if ClinicalDocument root has a child 'id' element
   doc_id = clinical_document_root.find(expand_tag('id'))

   # -- If so, check if id element has root and extension attributes
   if doc_id is not None:
      doc_id_root = doc_id.attrib.get('root', None)
      doc_id_ext = doc_id.attrib.get('extension', None)

   # -- See if ClinicalDocument root has a "recordTarget/patientRole/id" descendant element
   doc_patient_id_find_path = \
      f"{expand_tag('recordTarget')}/" \
      f"{expand_tag('patientRole')}/" \
      f"{expand_tag('id')}"

   doc_patient_id = clinical_document_root.find(doc_patient_id_find_path)

   # -- If so, check if id element has root and extension attributes
   if doc_patient_id is not None:
      doc_patient_id_root = doc_patient_id.attrib.get('root', None)
      doc_patient_id_ext = doc_patient_id.attrib.get('extension', None)
else:
   print(f"Root of document is {clinical_document_root.docinfo.root_name}, not 'ClinicalDocument'\n")

# -- Results to stdout for checking
print(f"doc_id_root: {doc_id_root}")
print(f"doc_id_ext: {doc_id_ext}")
print()
print(f"doc_eff_time_value: {doc_eff_time_value}")
print()
print(f"doc_patient_id_root: {doc_patient_id_root}")
print(f"doc_patient_id_ext: {doc_patient_id_ext}")


# -- Write out extracted elements as a JSON file
results_dict = \
   {
      "document_id_root": doc_id_root,
      "document_id_ext": doc_id_ext,
      "document_eff_time": doc_eff_time_value,
      "patient_id_root": doc_patient_id_root,
      "patient_id_ext": doc_patient_id_ext
   }

results_as_json = json.dumps(results_dict, indent=3)
with open("simple_json_parse_output.json", 'w') as f:
    f.write(results_as_json)
```

Here is one very simple example of parsing CCD XML to find data of interest and dumping to JSON. It uses the lxml XML module, but you could also use ElementTree perhaps as well, but one ***important*** thing to consider is that these do not protect well against malicious xml.   I have used the above script against my own test data only! A package such as defusedxml would be a better option for parsing production xml from external sources in order to protect against xml vulnerabilities!!
+++

[12:20 PM] Nystrom, Brian

so I checked with IT and learned that diameter health considers XML to JASON mapping proprietary and they have not shared it with us. What we can do is to reverse engineer and get it confirmed from availaity on they weekly call. Dean said Availity folks can confirm on the call if we are tracing it back to the right field or not.
+++
++++++++++++++++++++++++


Tables

OCDP_PRD_HEALTHCARE_ECONOMICS_DB.EMTELLIPRO_ST_RD.EMTELLIPRO_AVAILITY_CROSSWALK

Purpose : < what is need and use of this table >

Fields / attributes : - List the attributes , call out anything that is key for stored proc / do not delete

Rules / consideration for modification, add, delete update

OCDP_PRD_HEALTHCARE_ECONOMICS_DB.EMTELLIPRO_ST_RD.FILTERED_CCD_DOCUMENTS

OCDP_PRD_HEALTHCARE_ECONOMICS_DB.EMTELLIPRO_ST_RD.CCD_BUILD_HISTORY

Stored procedures
sp_Save_CCD_Crosswalk_Run_Stats

Purpose:

Section and its use

sp_Get_CCD_Build_CCD_Build_SQL_Statements
sp_Build_CCD_Crosswalk
sp_Save_CCD_Crosswalk_Run_Stats

Python code
CCDRuntimeParameters.py
DBAccessUtilities.py
CCDBuildParallelProcess
Test Cases & Test Scripts
++
Purpose : < what is need and use of this table >
Fields / attributes : - List the attributes , call out anything that is key for stored proc / do not delete

Rules / consideration for modification, add, delete update

For storage consideration
CICD guidance
++

- Transient - FILTERED_CCD_DOCUMENTS
- CCD_BUILD_HISTORY
- CCD_CROSSWALK_PROCESSED
- EMTELLIPRO_AVAILITY_CROSSWALK
- ERROR_LOG_TEST

- CCDA.XML_DOCUMENT_COMBINED
- CCDA.XML_DOCUMENT
- EMTELLIPRO_ST_FINAL.DOCUMENT
- EMTELLIPRO_ST_FINAL. PROCESSINGDETAILS
- EMTELLIPRO_ST_FINAL.DOCUMENTSTRUCTUREDMETADATA
- ECDH_L1.COMPOSITION_NOTES_DH
- ECDH_L1.COMPOSITION_CODE
- ECDH_L1.COMPOSITION

13. OCDP_PRD_HEALTHCARE_ECONOMICS_DB.EMTELLIPRO_ST_FINAL. PROCESSINGDETAILS
14. OCDP_PRD_OPTUMCARE_CORE_DB.CCDA.XML_DOCUMENT_COMBINED
15. OCDP_PRD_OPTUMCARE_CORE_DB.CCDA.XML_DOCUMENT
16. OCDP_PRD_HEALTHCARE_ECONOMICS_DB. EMTELLIPRO_ST_FINAL.DOCUMENT
17. OCDP_PRD_HEALTHCARE_ECONOMICS_DB.EMTELLIPRO_ST_FINAL.DOCUMENTSTRUCTUREDMETADATA
18. OCDP_PRD_OPTUMCARE_CORE_DB.ECDH_L1.COMPOSITION_NOTES_DH
19. OCDP_PRD_OPTUMCARE_CORE_DB.ECDH_L1.COMPOSITION_CODE
20. OCDP_PRD_OPTUMCARE_CORE_DB.ECDH_L1.COMPOSITION

++

Run stats (6-7 minutes):

```
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:\Users\dcheema\Documents\OAS\Projects\CCD Crosswalk\Work Items\CCDC
rosswalkBuildComponents\CCDBuildParallelProcess.py
42227
The CCD Crosswalk build process started at 2023-11-25 08:49:33.989184, processed
 42227 CCDs and it took 396 seconds.

= RESTART: C:\Users\dcheema\Documents\OAS\Projects\CCD Crosswalk\Work Items\CCDC
rosswalkBuildComponents\CCDBuildParallelProcess.py
42227
The CCD Crosswalk build process started at 2023-11-25 09:15:28.119879, processed
 42227 CCDs and it took 343 seconds.

= RESTART: C:\Users\dcheema\Documents\OAS\Projects\CCD Crosswalk\Work Items\CCDC
rosswalkBuildComponents\CCDBuildParallelProcess.py
42227
The CCD Crosswalk build process started at 2023-11-27 09:47:06.862786, processed
 42227 CCDs and it took 312 seconds.
```

++

- Completed and sign-off of design for the end-to-end data processing steps needed to establish the crosswalk of data across Availity and Emtelligent tables in Snowflake.
  - Identify any additional Snowflake capabilities (i.e. larger VW) needed to execute the crosswalk process in a performant and scalable manner
  - Define structured data formats (tables) for pre-processing and organizing vast amounts of data tied up in variant columns for more responsive query access for analysis.
  - Example: Emtellipro's documentstructuredmetadata table stores data in variant columns which could be pre-parsed out into separate structured data tables so parsing doesn't have to happen as part of the data query.
  - Determine the approach to enhancement of "xml_document" table to include section level data. Known options below:
  1. Done in the existing pipeline (before calling the Emtellipro Engine) when the "xml_document" table entries are created
  2. After the CCD XMLs have been run through the pipeline and the crosswalk is updated
  - Define the automation approach to execute the process at predefined intervals.
- Deploy fully signed-off crosswalk processing for automated execution
  - Coordinate any dependencies with IT team that owns the processing and load of Availity & Emtelligent data into Snowflake
  - Develop & unit test data processes to build crosswalk connecting Availity and Emtelligent data
  - Coordinate Snowflake capacity needs for on-going execution of crosswalk process
  - Perform integrated testing of end-to-end processing as needed, including IT team as needed
- Provide 2-weeks warranty period
  - Ensure productionized process is running as expected and address any potential bugs or issues as needed.

CCD IT to provide Snowflake capacity provisioning, Infrastructure (Virtual Warehouse) sizing and cost estimates - meeting scheduled for tomorrow
Data structured - completed
Pre-processing - XML_Document Union - Greg Holt & Joe Dale working on it
Approach to enhancement of "xml_document" table to include section level data - completed using queries and stored procedures
Before calling the Emtellipro Engine) when the "xml_document" table entries are created - Does not give the best results, as a result, **not adopted and agreed upon by the CCD team**
After the CCD XMLs have been run through the pipeline - **most appropriate and efficient** approach. Use Document table of EmtelliPro and use that as a baseline data to build the Crosswalk

- Deploy fully signed-off crosswalk processing for automated execution
  - Coordinate any dependencies with IT team that owns the loading and processing - Working with CCD IT team - had initial discussions and sent relevant artifacts for their review and analysis, initial artifacts review meeting is scheduled for tomorrow
  - Develop & unit test data processes to build crosswalk connecting Availity and Emtelligent data - **completed**
  - Perform integrated testing of end-to-end processing as needed, including IT team as needed - **waiting for volume data** to conduct integration tests
  -
- Warranty period (two weeks)
  - Provide support as necessary

CCD IT to provide Snowflake capacity provisioning, Infrastructure (Virtual Warehouse) sizing and cost estimates - meeting scheduled for tomorrow
Pre-processing - XML_Document Union - Greg Holt & Joe Dale working on it

- Deploy fully signed-off crosswalk processing for automated execution
  - Coordinate any dependencies with IT team that owns the loading and processing - Working with CCD IT team - had initial discussions and sent relevant artifacts for their review and analysis, initial artifacts review meeting is scheduled for tomorrow
  - Develop & unit test data processes to build crosswalk connecting Availity and Emtelligent data - **completed**
  - Perform integrated testing of end-to-end processing as needed, including IT team as needed - **waiting for volume data** to conduct integration tests

Coordinating with CCD IT team to ensure smooth transition of CCD Crosswalk from development to deployment and operations - had initial meetings, sent relevant artifacts and review is scheduled for 11/30/2023

Coordinate any dependencies with IT team that owns the loading and processing - Working with CCD IT team - had initial discussions and sent relevant artifacts for their review and analysis, initial artifacts review meeting is scheduled for tomorrow

To improve efficiency and performance of operations, working with CCD IT team to do as much pre-processing as practical

Dependencies on CCD IT team:

- Snowflake capacity provisioning
- Infrastructure (VW) sizing and cost estimates
- Volume (1+ M CCDs) test data
- Environment provisioning for UAT and prod environments
- Automation script (scheduled/manual)
- CICD process guidance
- Runtime configurations, e.g., predecessors (EmtelliPro, Availity, XML Document pre-processing completion
- Database credentials (stored procedures execution and Python service account credentials to run python script to execute stored procedures)
- GitHub cloud credentials
- Default starting date to start the CCD Crosswalk process

Please feel free to add anything that I may have left out. Thank you.

++

Cold start, null parameters:

= RESTART: C:\Users\dcheema\Documents\OAS\Projects\CCD Crosswalk\Work Items\CCDCrosswalkBuildComponents\CCDBuildParallelProcess.py
The CCD Crosswalk build process started at 2023-11-29 19:24:00.977589, processed 46442 CCDs and it took 384 seconds.
CCD Build Process ended.

= RESTART: C:\Users\dcheema\Documents\OAS\Projects\CCD Crosswalk\Work Items\CCDCrosswalkBuildComponents\CCDBuildParallelProcess.py
The CCD Crosswalk build process started at 2023-11-29 19:32:23.998079, processed 46442 CCDs and it took 372 seconds.
CCD Build Process ended.

With parameters:

= RESTART: C:\Users\dcheema\Documents\OAS\Projects\CCD Crosswalk\Work Items\CCDCrosswalkBuildComponents\CCDBuildParallelProcess.py
The CCD Crosswalk build process started at 2023-11-29 20:06:24.240698, processed 46442 CCDs and it took 352 seconds.
CCD Build Process ended.

++

	CCD	Time to execute	RAM	Compute	GPU			
23 *	46500	7 m	64 GB	8 cores		200000	2 mont	1.4
	46500	23 M	124 GB	16 cores		400000	2	3
	~ 25000					400000	3	2

+++

When parameter is passed:"

startdate	9/1/2023	CCD_BUILD_HISTORY	
end_date	9/30/2023	last_timestamp	9/30/2023
Next run		CCD_BUILD_HISTORY	
startdate	30-Nov	max(last_timestamp)	we will get max(last_timestamp) as startdate from ccd_build history
end_date	12/1/2023		

On normal runs, we don't provide any parameters.
For example, last night we processed 11/30/2023 CCDs based on the latest (maximum)
ccd_build_run_timestamp from CCD_BUILD_HISTORY as the start date, and the latest (maximum) submit_timestamp of the ProcessingDetails table as the end date

We got request to process historical CCDs (from 09/01/2023 - 09/30/2023); parameters: start_date => 09/01/2023 and end_date => 09/30/2023
Dates are used as they arebeing passed, no calculations are done
At the end of the historical CCDs processing, we'll have a row with last processing date as 09/30/2023 in the CCD Build History table

On the next day, on a regular run, no parameters will be passed.
The system will determine the start date and the end date:

- The start date will be the maximum ccd_build_run_timestamp from the CCD Build History table
- The end date will be the maximum submit_timestamp for the ProcessingDetails, which will be 11/30/2023, not 09/30/2023

As a result, no CCDs will be reprocessed, unless it is intentional

++

Storage
4 TB x $23/TB/month x 12 months
= $1,104/year

Daily Loading Task
2.5 hours/day x 31 days/month x 12 months/year x
2 credits x $2 x 0.95 discount
= $3,534/year

Daily Work
10 hours/day x 20 days/month x 12 months/year x
4 credits x $2 x 0.95 discount
= $18,240

Agenda

- Cost Depends On?
- Types of Costs
- Storage Types
- Compute Cost
- Snowflake Credit
- Serverless features

Cost Depends On?

Snowflake cost is dependent on

1. Snowflake Edition
   - Standard - $2.7/Credit
   - Enterprise - $4/Credit
   - Business Critical - $5.4/Credit
   - VPS – Depends on Org
2. Region where Snowflake account created
3. Cloud platform where Snowflake account hosted
4. Virtual Warehouse Size

Types of Costs

1. Storage Cost
2. Compute Cost
3. Cloud Services Cost

Storage Cost

- The charge for Storage is per terabyte after compressed, per month.
- 2 Storage Plans available in Snowflake.

1. On Demand Storage:
   - Most flexible and easiest model
   - Pay-as you Use / Pay-as you go
   - Customer are charged FIXED rate of $40/TB Per month (compressed)
2. Capacity Storage (or) Fixed Storage:
   - Option to Pre-purchase capacity
   - A Capacity purchase is a specific dollar commitment to snowflake
   - Snowflake charges $23 per month/TB (compressed)

Compute Cost

- Compute cost is calculated based on the Virtual Warehouse usage per month
- Compute cost is calculated in Snowflake Credits
- Billed per seconds with 1 minute minimum
- The different sizes of virtual warehouses consume snowflake credits at the following rates.
- If we choose Large size and if we used it for 30 minutes then it will be billed as 4 Credits

Snowflake Warehouse Sizes and Credit Usage per Hour										
Size	X-Small	Small	Medium	Large	X-Large	2X-Large	3X-Large	4X-Large	5X-Large (In preview)	6X-Large (In preview)
Credit Usage per Hour	1	2	4	8	16	32	64	128	256	512

Cap	100GB	0.1*23	$2.3	10GB		2 clusters	M	M	4*2*2	16
					1st	$4				
OnD	10GB	0.01*40	$0.4		2nd	$4	$4			
							$12			
Cap	800GB	0.8*23	18.4	800GB						
OnD	800GB	0.8*40	32							

What is a Snowflake Credit

- A snowflake credit is a unit of measure
- Snowflake credits are used to pay for the consumption of the resources on Snowflake
- It is consumed only when a customer is using resources, such as when using virtual warehouses.
- Users receives $400 worth of free usage upon creation of Snowflake free trial account

Serverless Features

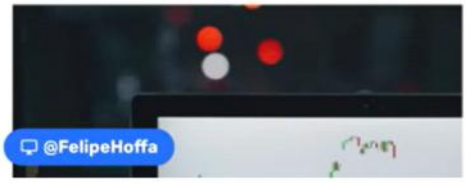
- Serverless features use snowflake-managed compute resources and consume snowflake credits when they are used.
- We can see the credit usage on the bill.
- Below are the serverless features.

Functional Area	Feature	Usage
Tables	Automatic Clustering	Automated background maintenance of each clustered table, including initial clustering and reclustering as needed
	Search Optimization Service	Automated background maintenance of the search access paths used by the search optimization service
External tables	Automatically Refreshing External Table Metadata	Automated refreshing of the external table metadata with the latest set of associated files in the external stage and path
Views	Materialized views maintenance	Automated background synchronization of each materialized view with changes in the base table for the view
Data loading	Snowpipe	Automated processing of file loading requests for each pipe object
Database replication	Database Replication and Failover/Failback	Automated copying of data between accounts, including initial data replication and maintenance as needed
Tasks	Executing SQL Statements on a Schedule Using Tasks	Snowflake-managed compute resources provided to execute SQL code, rather than a user-managed virtual warehouse.
Warehouses	Using the Query Acceleration Service	Snowflake-managed compute resources provided to execute portions of eligible queries. [1]

5 practical things you can do today to optimize cost in your Snowflake deployment

Gilberto Hernandez · Following
Published in Snowflake
6 min read · 6 days ago

Listen Share ••• More



BEST PRACTICES

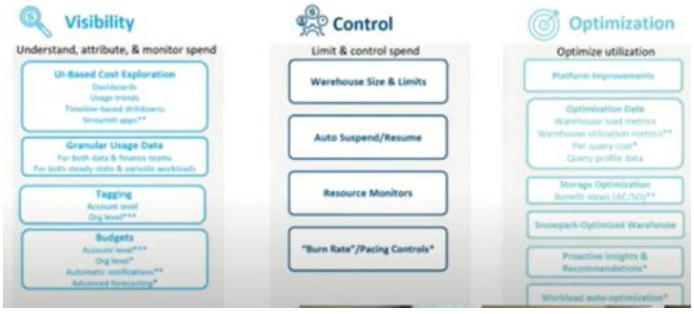
Obtain Visibility Into Cost & Implement Chargeback
Track usage at granularities such as specific workloads, warehouses, tasks, etc. Attribute and aggregate spend to specific teams, customers, or cost centers (via tagging, cost attribution features) towards chargeback/showback

Proactive Budgeting & Alerting
Budget at the level of account and/or custom groupings of resources. Alert on spending limits and guards (via resource monitors) against overspending

ROI-Based Workload Segmentation
Understand the cost and ROI of workloads to make informed decisions on re-architecture, decrease/increase of usage, or even retire certain workloads with the goal of ensuring the most return on spend

Throttle/Control Resource Allocation & Scaling Permissions
A central team manages policies around credit consuming operations such as warehouse creation, sizing, and scaling limits to control costs and reduce indeterminacy

CURRENT FUNCTIONALITY AND ROADMAP



METRICS GUIDED WAREHOUSE OPTIMIZATION

If your workload has adequate throughput and/or latency performance AND (queued query load is low OR total query load <1 for prolonged periods) AND utilization is low (say 50%):

consider downsizing the warehouse or reducing # of clusters

If your workload is running slowly (based on throughput and/or latency measurements detailed) AND running query load is low AND utilization is high (e.g., >75%):

consider upsizing the warehouse or adding clusters

If there are recurring usage spikes (based on warehouse utilization history):

consider moving queries that represent the spikes to a new warehouse or adding clusters

++

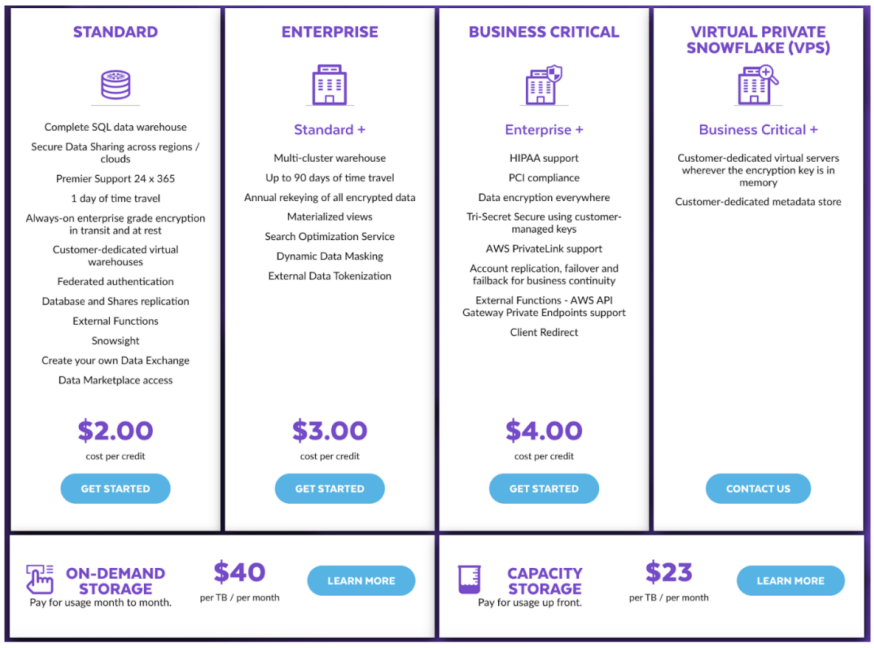
A Breakdown of Snowflake Costs

You can estimate your Snowflake costs in 4 steps:

1. Estimate the number of warehouses by size that are required.
2. Estimate the amount of compute hours that each warehouse will use by size.
3. Estimate the storage size of your data.
4. Determine the features required to determine the Snowflake account level required.

Snowflake Warehouse Sizes and Credits Per Hour										
Size	X-Small	Small	Medium	Large	X-Large	2X-Large	3X-Large	4X-Large	5X-Large*	6X-Large*
Credits/Hour	1	2	4	8	16	32	64	128	256	512

*Available for general use in most AWS regions and as a preview feature in Azure and US Government regions



*Note that these costs per credit are based on an account in AWS US-East-1 and may differ depending on the cloud provider/region the Snowflake account is created in.
Once all of the above are determined, it is pretty straightforward to provide an estimate based on cost per credit and number of credits that will be consumed.

But that's the baseline. There are other questions to consider:

- Are you going to be using replication?
  - Storage: Amount of table data in the primary database, or databases in a replication/failover group, that changes as a result of data loading or DML operations.
  - Compute resources: Frequency of secondary database, or replication/failover group, refreshes from the primary database or rep lication/failover group.
- Are you going to be using Materialized Views?
  - Storage: Each materialized view stores query results, which adds to the monthly storage usage for your account.
  - Compute resources: In order to prevent materialized views from becoming out-of-date, Snowflake performs automatic background maintenance of materialized views. When a base table changes, all materialized views defined on the table are updated by a background service that uses compute resources provided by Snowflake.
- Are you going to be using Snowpipe?
  - Given the number of factors that can differentiate Snowpipe loads, it is very difficult for Snowflake to provide sample costs . File formats and sizes, and the complexity of COPY statements (including SELECT statement used for transformations), all impact the resource consumption and file overhead charged for a Snowpipe load.
- Are you going to be using Search Optimization Service?
  - Storage resources: The search optimization service creates a search access path data structure that requires space for each t able on which search optimization is enabled.
  - Compute resources: Adding search optimization to a table consumes resources during the initial build phase. Maintaining the s earch optimization service also requires resources. Resource consumption is higher when there is high churn (i.e. when large volumes of data in the table change). The se costs are roughly proportional to the amount of data ingested (added or changed). Deletes also have some costs.

Why do each of the above matter?

While tracking SQL query usage and credit consumption is relatively straightforward via query history, the functions listed above are not straightforward and often difficult to track. As a new customer, you might not know what to expect. As an experienced customer, it can quickly become a black box.
Trying to determine how best to transform and serve your data? Check out our blog on transforming data with tasks and views.

How do I Ensure Snowflake Costs Are Efficient?

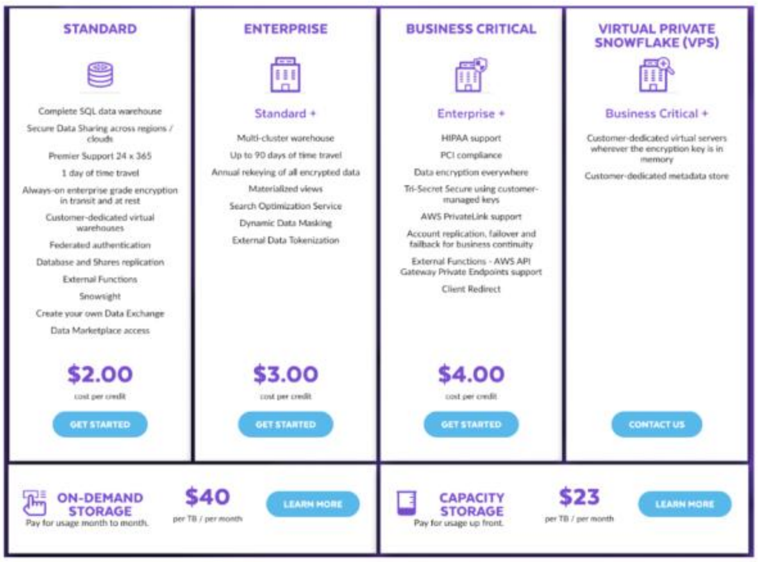
When you blow out your budget in 3 months (or are on track to), how can you pinpoint the areas that need to improve?

Can you point to bad (long-running) SQL? Is it tied to your ingest process? Is there a business unit that is consuming far more credits than necessary? Are developers letting queries run over the weekend?

It can be difficult for an organization to get clear answers to these questions when they don't have the necessary controls to reign in the spending — especially when the promise is on demand data for anyone and everyone across the organization.

VIRTUAL WAREHOUSES CREDITS PER HOUR

	XS	S	M	L	XL	2XL	3XL	4XL	5XL*	6XL*
Standard	1	2	4	8	16	32	64	128	256	512
Snowpark-Optimized	N/A	N/A	6	12	24	48	96	192	384	786



*Note that these costs per credit are based on an account in AWS US-East-1 and may differ depending on the cloud provider/region the Snowflake account is created in.

A Breakdown of Snowflake Costs

- Estimate the number of warehouses by size that are required.
- Estimate the amount of compute hours that each warehouse will use by size.
- Estimate the storage size of your data.
- Determine the features required to determine the Snowflake account level required

Snowflake Warehouse Sizes and Credits Per Hour										
Size	X-Small	Small	Medium	Large	X-Large	2X-Large	3X-Large	4X-Large	5X-Large*	6X-Large*
Credits/Hour	1	2	4	8	16	32	64	128	256	512

*Available for general use in most AWS regions and as a preview feature in Azure and US Government regions

Total CCDs: 46442
CCDs per minute: 46442/7 = 6634.5714 (CCDs/minute)
CCDs processed/per hour: 6634.5714 * 60 = 398074.284/hour
Total CCDs: 51000000 CCDs to be processed
Total hours to process: 51000000 / 398074.284 = 128.1168 hours
Total compute credits to process all CCDs = 128.1168 * 4 = 512.4672 Credits
Cost of total CCDs to process: 512.4672 * 4 = 2,049.8688

Based on current VW configuration - 1 VW, size: Medium, per unit credits used per hour: 4, each credit cost:
++
[1:13 PM] Harlicker, Ryan S

- Hey Folks, in theory it might make sense for this to live in the **Pop Health repo**, but a couple of code review comments ahead of our meeting
  - Work with David M. and Greg H.

[1:15 PM] Harlicker, Ryan S
DBAccessUtilities.py

- One note, I see that we are running this process as a user credential. We should ensure that all snowflake artifacts (tables/stored procedures) are owned by a **non-user account** to prevent losing access to them in case of end users leaving the company
  - Agree. Done, using existing non-user id
- In theory data ETL processes **should use a DATA_LOAD_* warehouse** instead of a QUERY warehouse.
  - Agree. Using it
- What is the **expected refresh cycle of running these workflows**. We have had a recent scenario where a team did a 10 minute refresh with an Large warehouse. That turns out to be around 42k/month to keep a large warehouse alive 24/7. Around 22k for the 64 (medium) warehouse you have mentioned here. We want to make sure that the refresh cadence is appropriate for this sized warehouse
  - It looks like **daily**, but they envision it to run on demand/ event based triggers

[1:21 PM] Harlicker, Ryan S

- for this scenario you will need to **create a non-user account** as a best practice and **change ownership to it based on the team that is going to maintain** this going forward.
  - Agree. Done;
- I'll capture a screenshot for the Data_load warehouses from secure
  - Done
- you will need to request a non-user account to request the access
  - Done

[1:29 PM] Harlicker, Ryan S

- The other option would be to extend the current non-user account that we use to provision the EmtelliPro schema but I'm not sure if it has the cross schema rights as of yet
  - Adopted. David M. helped
- Note, depending upon the appropriateness we could in theory add the addition requirements to this non-user account. Have to consider if that is appropriate but…
  - done
- **snowsql -a uhg_optumcare.east-us-2.azure -u ocemtel_ocdpprd@optum.com --private-key-path ../files/rsa_key.p8 --rolename AR_PRD_OCEMTEL_OCDPPRD_OPTUM_ROLE --dbname OCDP_PRD_HEALTHCARE_ECONOMICS_DB --schemaname EMTELLIPRO --warehouse OCDP_PRD_STREAM_WH**
- I believe we have rights to the 64 data load warehouse for this user, but probably not the ECDH schemas
  - Done. It works.
- Add David Mahajan to this conversation for this instance only
  - Done.
- Estimate the cost of initial workload of processing ~51,000,000 CCDs and ongoing cost (daily ~130K CCDs). Do you know **how we can get it**? If you don't, do you know who might be able help us with that? BTW, we do know how to cost estimate, we just need the rates for the available warehouses
  - **OptumTech rate: $7.90/credit**
- We have two versions of this process - a stored procedure version and a Python version, doing the same thing. Which one is a preferred approach?
  - Python is preferred
- Snowflake capacity provisioning - need to understand the procedure
  - Leverage existing capability
- Repo: **Pop Health repo** - need access
  - Done. David M. set it up. Now OAS team to make Secure requests to get access to it.

- Default starting date to start the CCD Crosswalk process - After CCD_BUILD_HISTORY table is created (Default values: '2018-01-01 00:00:00', 2000, 300, current_timestamp()). - dependency on table definition completion. **OAS** team responsible
- Volume (1+ M CCDs) test data
  - In-progress

>>>

- CICD process need guidance
  - All code in Databricks, that is integrated with Enterprise cloud repo - Greg Holt
    - David M./Greg H. responsible; Ryan & Sharon to be consulted
- Do you have a sample script to trigger this job?
  - Built-in scheduler in Databricks; **Greg H./David M. responsible; Ryan & Sharon to be consulted**
- Environment provisioning for UAT and prod environments - Need to work with David M.
  - David M. responsible; Ryan & Sharon to be consulted
- Automation script (scheduled/manual) (most likely Databricks)
  - David M./Greg H. will do it; Ryan & Sharon to be consulted
- Runtime configurations, e.g., predecessors (EmtelliPro, Availity, XML Document pre-processing completion
  - Data is stored in a table and then manually pinged; David M. & Greg H. responsible; Ryan & Sharon to be consulted
- To run this job successfully, the following tables must have relevant data (jobs that populate these tables must be completed):

Schema	Table name
CCDA	XML_DOCUMENT_COMBINED
CCDA	XML_DOCUMENT (Note: XML_DOCUMENT_COMBINED & XML_DOCUMENT tables are being joined to create a one unique CCDs set)
EMTELLIPRO_ST_FINAL	DOCUMENT
EMTELLIPRO_ST_FINAL	PROCESSINGDETAILS
EMTELLIPRO_ST_FINAL	DOCUMENTSTRUCTUREDMETADATA
ECDH_L1	COMPOSITION_NOTES_DH
ECDH_L1	COMPOSITION_CODE
ECDH_L1	COMPOSITION

  - David M. responsible; Ryan & Sharon to be consulted

++

145,990,660 - 145985303 = 5357

https://github.com/optum-care
https://github.com/optum-care/pop-health-ccd-crosswalk
github_users => https://platform.pages.ocdp.optum.com/team/onboarding/github/#getting-access-to-caredata-platform-ghes-organizations

++

++

Healthcare Platform
A product of UNITEDHEALTH GROUP

SDRP
Strategic Data Repository Platform

SDRP (Strategic Data Repository Platform) – Snowflake Ratecard Update for 2024

ALERT TYPE:	COMMUNICATION	NEW	UPDATE	REMINDER	RESOLVED/COMPLETED

ANNOUNCEMENT

We are Happy to announce that Optum Financial has significantly reduced the rates for 2024. For more details please visit our SDRP Docs **Costs - 2024**.

TENANT ACTIONS

- Please submit your **2024 Forecast** by **Nov 15th, 2023**

If you have any questions or concerns, please contact us in Teams

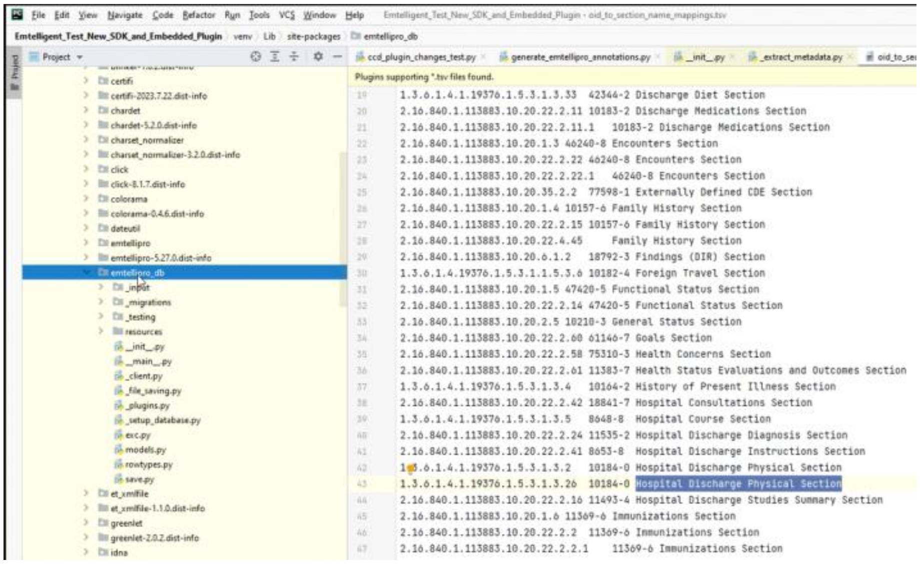
Healthcare Platform

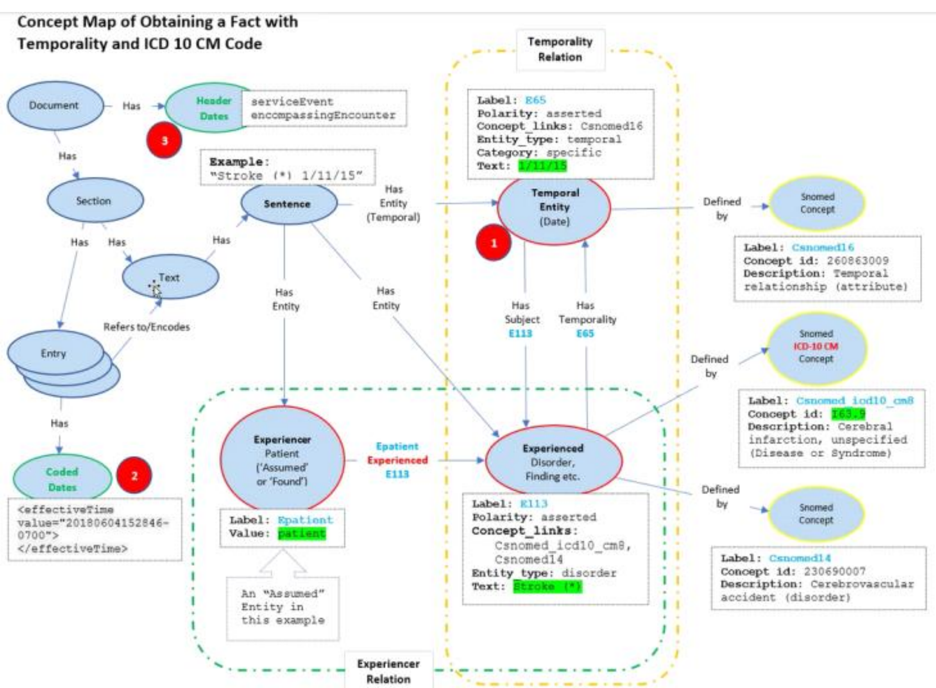

```
= RESTART: C:\Users\dcheema\Documents\OAS\Projects\CCD Crosswalk\Work Items\CCDCrosswalkBuildComponents\CCDBuildParallelProcess.py
The CCD Crosswalk build process started at 2023-12-08 12:14:21.624391, processed 46442 CCDs and it took 294 seconds.
CCD Build Process ended.

= RESTART: C:\Users\dcheema\Documents\OAS\Projects\CCD Crosswalk\Work Items\CCDCrosswalkBuildComponents\CCDBuildParallelProcess.py
Traceback (most recent call last):
  File "C:\Users\dcheema\Documents\OAS\Projects\CCD Crosswalk\Work Items\CCDCrosswalkBuildComponents\CCDBuildParallelProcess.py", line 44, in main
    last_processingDetails_ts = result[result.find(" - ") + len(" - "):][:-1]
                                       ^^^^^^^^^^^
```

AttributeError: 'NoneType' object has no attribute 'find'
CCD Build Process ended.

= RESTART: C:\Users\dcheema\Documents\OAS\Projects\CCD Crosswalk\Work Items\CCDCrosswalkBuildComponents\CCDBuildParallelProcess.py
The CCD Crosswalk build process started at 2023-12-08 12:21:49.633670, processed 46442 CCDs and it took 267 seconds.
CCD Build Process ended.

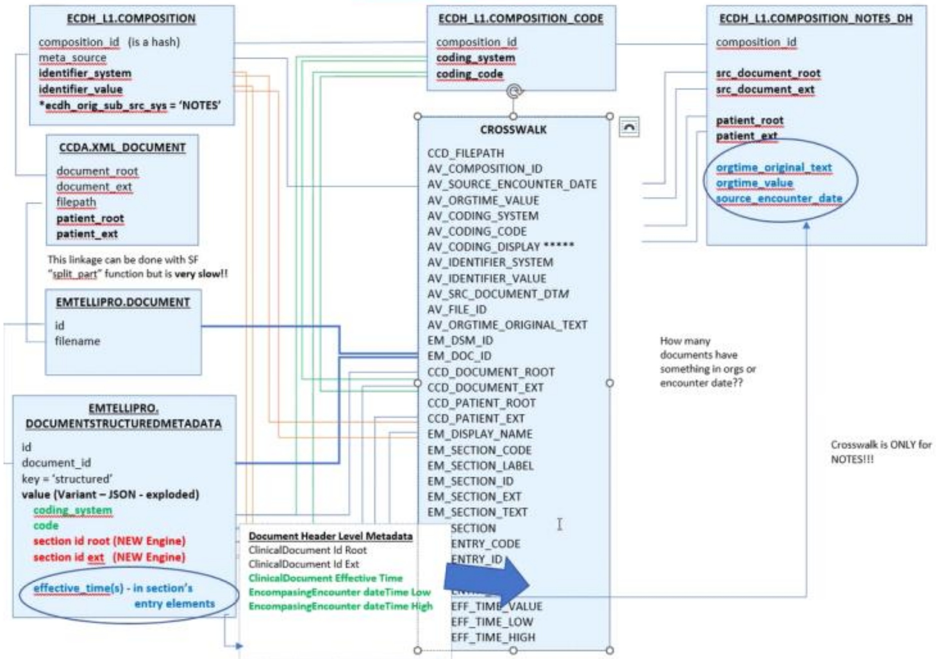
= RESTART: C:\Users\dcheema\Documents\OAS\Projects\CCD Crosswalk\Work Items\CCDCrosswalkBuildComponents\CCDBuildParallelProcess.py
The CCD Crosswalk build process started at 2023-12-08 12:28:01.080879, processed 46442 CCDs and it took 219 seconds.
CCD Build Process ended.
++



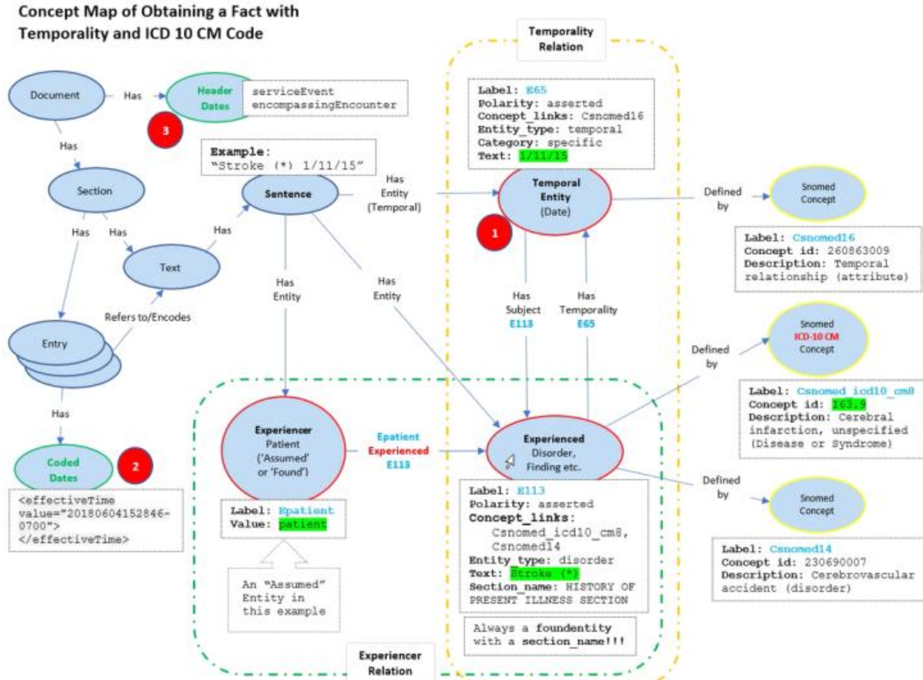


+++

ECDH_L1 (Availity), CCDA (CCD Metadata) , and Emtellipro (Emtelligent NLP Engine) Key Tables and Relationships



Concept Map of Obtaining a Fact with Temporality and ICD 10 CM Code



Composition (Availity)			DocumentStructuredMetadata		
Compostion Table	Composition column		FOR_ELT_TAG	CODE_ELT_TAG	Column
COMPOSITION_NOTES_DH	src_document_root	↔	ClinicalDocument	code	ID
COMPOSITION_NOTES_DH	src_document_ext	↔	ClinicalDocument	code	EXTENSION
COMPOSITION_CODE	coding_system	↔	section	code	coding_system
COMPOSITION_CODE	coding_code	↔	section	code	code
COMPOSITION	identifier_system	↔	section	code	ID
COMPOSITION	identifier_value	↔	section	code	EXTENSION

```
-- #2
SELECT
    dm.value as x_value
FROM  "OCDP_PRD_HEALTHCARE_ECONOMICS_DB"."EMTELLIPRO_ST_FINAL"."DOCUMENTSTRUCTUREDMETADATA" dm
    , LATERAL FLATTEN( INPUT => x_value:dm.template_id)
WHERE
    -- 10/17/2023 - new doc id for canonical document is now b9aba4f8-c970-4509-bc20-488571d9817b
    DOCUMENT_ID = 'b9aba4f8-c970-4509-bc20-488571d9817b' --order by for_elt_tag
and  for_elt_tag = 'section'
and  code_elt_tag = 'code'

select value from "OCDP_PRD_HEALTHCARE_ECONOMICS_DB"."EMTELLIPRO_ST_FINAL"."DOCUMENTSTRUCTUREDMETADATA"
   where PARSE_JSON(value):template_id in (select template_id from OCDP_PRD_HEALTHCARE_ECONOMICS_DB.EMTELLIPRO_ST_RO.SECTION_NAMES_NORMALIZED);
```

```
(
SELECT
        dm.document_id                                          as document_id,
        dm.value                                                as dsm_json,    -- RENAME VALUE COLUMN!!!
        PARSE_JSON(dm.value):code_system_name::string           as coding_system_name,
        PARSE_JSON(dm.value):code::string                       as code_element_code_attribute,
        PARSE_JSON(dm.value):display_name::string               as display_name,
        PARSE_JSON(dm.value):for_elt_tag::string                as for_elt_tag,
        PARSE_JSON(dm.value):code_elt_tag::string               as code_elt_tag,
        PARSE_JSON(dm.value):template_id                        as template_id     -- but can be multiple template ids used!!!
        --PARSE_JSON(dm.value):template_id[0].root::string        as template_id_root,      -- but can be multiple template ids used!!!
        --PARSE_JSON(dm.value):template_id[0].extension::string   as template_id_extension,-- but can be multiple template ids used!!!
FROM    "OCDP_PRD_HEALTHCARE_ECONOMICS_DB"."EMTELLIPRO_ST_FINAL"."DOCUMENTSTRUCTUREDMETADATA" dm

WHERE

,
select for_elt_tag,
        display_name,
        code_element_code_attribute,
        value:root::string as template_id_root,
        value:extension::string as template_id_extension
from    flatten_example
        , LATERAL FLATTEN( INPUT => template_id )
,

WHERE
        -- 10/17/2023 - new doc id for canonical document is now b9aba4f8-c970-4509-bc20-488571d9817b
        DOCUMENT_ID = 'b9aba4f8-c970-4509-bc20-488571d9817b' --order by for_elt_tag
and     for_elt_tag = 'section'
and     code_elt_tag = 'code'
),
section_template_ids as
(
select for_elt_tag,
        display_name,
        code_element_code_attribute,
        value:root::string as template_id_root,
        value:extension::string as template_id_extension
from    flatten_example
        , LATERAL FLATTEN( INPUT => template_id )
)
select
        tmp_id.*,
        nrm.*
from
        section_template_ids tmp_id
join
         "OCDP_PRD_HEALTHCARE_ECONOMICS_DB"."EMTELLIPRO_ST_RD"."SECTION_NAMES_NORMALIZED" nrm
on
        tmp_id.template_id_root = nrm.template_id
```

+++

The Query:

```
SELECT
     dm.id,
     dm.document_id,
     PARSE_JSON(dm.value):code_system_name::string            as coding_system_name,
     PARSE_JSON(dm.value):code::string                        as code,
     PARSE_JSON(dm.value):display_name::string                as display_name,
     PARSE_JSON(dm.value):for_elt_tag::string                 as for_elt_tag,
     PARSE_JSON(dm.value):code_elt_tag::string                as code_elt_tag,
     PARSE_JSON(dm.value):template_id                         as template_id,     -- but can be multiple template ids used!!!
     --PARSE_JSON(dm.value):template_id[0].root::string         as template_id_root,     -- but can be multiple template ids used!!!
     --PARSE_JSON(dm.value):template_id[0].extension::string    as template_id_extension,-- but can be multiple template ids used!!!
     PARSE_JSON(dm.value):section::string                     as section,
     parse_json(dm.value):id[0].root::string                  as id,
     parse_json(dm.value):id[0].extension::string             as extension,
     PARSE_JSON(dm.value):label::string                       as label,
     PARSE_JSON(dm.value):text_id::string                     as text_id,
     PARSE_JSON(dm.value):effective_time[0].value::string     as etime_value,
     PARSE_JSON(dm.value):effective_time[0].low.value::string  as etime_low,
     PARSE_JSON(dm.value):effective_time[0].high.value::string as etime_high
FROM   "OCDP_PRD_HEALTHCARE_ECONOMICS_DB"."EMTELLIPRO_ST_FINAL"."DOCUMENTSTRUCTUREDMETADATA" dm
WHERE
     -- 10/17/2023 - new doc id for canonical document is now b9aba4f8-c970-4509-bc20-488571d9817b
     DOCUMENT_ID = 'b9aba4f8-c970-4509-bc20-488571d9817b' --order by for_elt_tag
and   (for_elt_tag = 'ClinicalDocument' or for_elt_tag = 'encompassingEncounter' or for_elt_tag = 'section' or for_elt_tag = 'serviceEvent') -- or for_elt_tag = 'act')
and   code_elt_tag = 'code
```

+++
++++++++

Agenda Items:

The agenda items and notes:

1. Duplicates
   A. Remove AV_CODING_DISPLAY column from crosswalk
   B. Distinct\Group By\Windowing on the remaining columns to dedupe? left w 68000

2. Normalization of Section Names - Need new Column and Lookup table

   A. Create a new column EM_DISPLAY_NAME_NORMALIZED, and keep EM_DISPLAY_NAME
   B. VaLue comes from Emtelligent oid_to_section_name_mappings.tsv file mapping. This should be made into a Snowflake table.
   C. Lookup done by section Template Id -- EX: <templateId root="1.3.6.1.4.1.19376.1.5.3.1.3.4"> or section code 10164-2

   INCLUDE TSV

3. CCD Document Header Level Metadata
   A. ClinicalDocument (CCD root element)
      1. id root and id extension available in documentstructuredmetadata
      2. effectiveTime in available in documentstructuredmetadata
   B. encompassingEncounter elemnt and children available in documentstructuredmetadata
      1. effectiveTime.low value attribute
      2. effectiveTime.high value attribute

4. Composition query in Artifacts
   A. "source_" columns usage. LEAVE AS IS
   B. Need composition and composition code table joins? YES, LEAVE AS IS

5. CCDA XML_DOCUMENT Usage
   A. Needed in light of point 3. above?
6. Eliminate text variant columns

+++

```
SELECT
    dm.id,
    dm.document_id,
    PARSE_JSON(dm.value):code_system_name::string              as coding_system_name,
    PARSE_JSON(dm.value):code::string                          as code,
    PARSE_JSON(dm.value):display_name::string                  as display_name,
    PARSE_JSON(dm.value):for_elt_tag::string                   as for_elt_tag,
    PARSE_JSON(dm.value):code_elt_tag::string                  as code_elt_tag,
    PARSE_JSON(dm.value):template_id                           as template_id,      -- but can be multiple template ids used!!!
    --PARSE_JSON(dm.value):template_id[0].root::string          as template_id_root,      -- but can be multiple template ids used!!!
    --PARSE_JSON(dm.value):template_id[0].extension::string     as template_id_extension,-- but can be multiple template ids used!!!
    PARSE_JSON(dm.value):section::string                       as section,
    parse_json(dm.value):id[0].root::string                    as id_root,
    parse_json(dm.value):id[0].extension::string               as id_extension,
    PARSE_JSON(dm.value):label::string                         as label,
    PARSE_JSON(dm.value):text_id::string                       as text_id,
    PARSE_JSON(dm.value):effective_time[0].value::string       as etime_value,
    PARSE_JSON(dm.value):effective_time[0].low.value::string   as etime_low,
    PARSE_JSON(dm.value):effective_time[0].high.value::string  as etime_high
FROM  "OCDP_PRD_HEALTHCARE_ECONOMICS_DB"."EMTELLIPRO_ST_FINAL"."DOCUMENTSTRUCTUREDMETADATA" dm
WHERE
```

++

```
SELECT
--    dm.id,
--    dm.document_id,
    PARSE_JSON(dm.value):code_system_name::string              as coding_system_name,
    PARSE_JSON(dm.value):code::string                          as code_element_code_attribute,
    PARSE_JSON(dm.value):display_name::string                  as display_name,
    PARSE_JSON(dm.value):for_elt_tag::string                   as for_elt_tag,
    PARSE_JSON(dm.value):code_elt_tag::string                  as code_elt_tag,
    PARSE_JSON(dm.value):template_id                           as template_id,      -- but can be multiple template ids used!!!
    --PARSE_JSON(dm.value):template_id[0].root::string          as template_id_root,      -- but can be multiple template ids used!!!
    --PARSE_JSON(dm.value):template_id[0].extension::string     as template_id_extension,-- but can be multiple template ids used!!!

    NVL(parse_json(dm.value):id[0].root::string, '*')          as id_root,
    NVL(parse_json(dm.value):id[0].extension::string, '*')     as id_extension,

    PARSE_JSON(dm.value):effective_time[0].value::string       as etime_value,
    PARSE_JSON(dm.value):effective_time[0].low.value::string   as etime_low,
    PARSE_JSON(dm.value):effective_time[0].high.value::string  as etime_high,

    PARSE_JSON(dm.value):section::string                       as section,
    PARSE_JSON(dm.value):label::string                         as label,
    PARSE_JSON(dm.value):text_id::string                       as text_id
FROM  "OCDP_PRD_HEALTHCARE_ECONOMICS_DB"."EMTELLIPRO_ST_FINAL"."DOCUMENTSTRUCTUREDMETADATA" dm
WHERE
    -- 10/17/2023 - new doc id for canonical document is now b9aba4f8-c970-4509-bc20-488571d9817b
    DOCUMENT_ID = 'b9aba4f8-c970-4509-bc20-488571d9817b' --order by for_elt_tag
and for_elt_tag in ( 'ClinicalDocument', 'encompassingEncounter', 'section', 'serviceEvent' ) --  ,'act')
and code_elt_tag = 'code'
--and  section = '4d87e445-fbf8-46bc-a793-8374b9e5ee51'  -- this is 10164-2's label value.  Search for 'section' column value of the same value
```

Crosswalk Column Updates and Order

NB Nystrom, Brian
To Cheema, Dave; Boinpally, Harish
Cc Ellis, William H
Retention Policy UHGInbox (90 days) Expires 3/13/2024
You replied to this message on 12/14/2023 1:05 PM.

Dave, Harish,

Here are the updates and new order for the crosswalk columns:

```
A1.FILEPATH                  as CCD_FILEPATH,
A1.src_document_root         as AV_DOCUMENT_ROOT,
A1.src_document_ext          as AV_DOCUMENT_EXT,
A1.patient_root              as AV_PATIENT_ROOT,
A1.patient_ext               as AV_PATIENT_EXT,
A1.COMPOSITION_ID            as AV_COMPOSITION_ID,
A1.coding_system             as AV_CODING_SYSTEM,
A1.coding_code               as AV_CODING_CODE,
A1.identifier_system         as AV_IDENTIFIER_SYSTEM,
A1.identifier_value          as AV_IDENTIFIER_VALUE,
A1.src_document_dtm          as AV_SRC_DOCUMENT_DTM,
A1.OBSTIME_VALUE             as AV_OBSTIME_VALUE,                     *** NEW
A1.ORGTIME_ORIGINAL_TEXT     as AV_ORGTIME_ORIGINAL_TEXT,
A1.AV_ORGTIME_VALUE          as AV_ORGTIME_VALUE,
A1.SOURCE_ENCOUNTER_DATE     as AV_SOURCE_ENCOUNTER_DATE,
A1.FILE_ID                   as AV_FILE_ID,                           *** REMOVE
A1.EM_DSM_ID                 as EM_DSM_ID,
A1.EM_DOC_ID                 as EM_DOC_ID,
xxxxx                        as EM_CLINICAL_DOCUMENT_ID_ROOT,         *** NEW
xxxxx                        as EM_CLINICAL_DOCUMENT_ID_EXT,          *** NEW
xxxxx                        as EM_CLINICAL_DOCUMENT_EFF_TIME,        *** NEW
xxxxx                        as EM_ENCOMPASING_ENCOUNTER_EFF_TIME_LOW,  *** NEW
xxxxx                        as EM_ENCOMPASING_ENCOUNTER_EFF_TIME_HIGH, *** NEW
A1.DISPLAY_NAME              as EM_DISPLAY_NAME,
A1.section_name_normalized   as EM_SECTION_NAME_NORMALIZED,
A1.SECTION_CODE              as EM_SECTION_CODE,
A1.SECTION_LABEL             as EM_SECTION_LABEL,
A1.SECTION_ID                as EM_SECTION_ID,
A1.SECTION_EXT               as EM_SECTION_EXT,
A1.text_id                   as EM_TEXT_ID,
E4.SECTION                   as EM_SECTION,
E4.ENTRY_CODE                as EM_ENTRY_CODE,
E4.ENTRY_ID                  as EM_ENTRY_ID,
E4.ENTRY_EXT                 as EM_ENTRY_EXT,
E4.ENTRY_TEXT_ID             as EM_ENTRY_TEXT_ID,
E4.EFF_TIME_VALUE            as EM_ENTRY_EFF_TIME_VALUE,              *** MODIFIED (added ENTRY)
E4.EFF_TIME_LOW              as EM_ENTRY_EFF_TIME_LOW,                *** MODIFIED (added ENTRY)
E4.EFF_TIME_HIGH             as EM_ENTRY_EFF_TIME_HIGH                *** MODIFIED (added ENTRY)
```

Please let me know if you have any questions.

Regards,

Brian Nystrom
Sr Healthcare Econ Cnslt, HCE-Population Health Analytics | Optum Health

'ClinicalDocument', 'encompassingEncounter', 'section', 'serviceEvent'

```
EM_CLINICAL_DOCUMENT_ID_ROOT
EM_CLINICAL_DOCUMENT_ID_EXT
EM_CLINICAL_DOCUMENT_EFF_TIME
EM_ENCOMPASING_ENCOUNTER_EFF_TIME_LOW
EM_ENCOMPASING_ENCOUNTER_EFF_TIME_HIGH
```

++

Composition (Availity)			DocumentStructuredMetadata			
Compostion Table	Composition column		FOR_ELT_TAG	CODE_ELT_TAG	Column****	
COMPOSITION_NOTES_DH	src_document_root	↔	ClinicalDocument	code	ID_ROOT	doc level only
COMPOSITION_NOTES_DH	src_document_ext	↔	ClinicalDocument	code	ID_EXTENSION	doc level only
COMPOSITION_CODE	coding_system	↔	~~section~~	code	coding_system	
COMPOSITION_CODE	coding_code	↔	~~section~~	code	code	
COMPOSITION	identifier_system	↔	section	code	ID_ROOT	
COMPOSITION	identifier_value	↔	section	code	ID_EXTENSION	

++

++

++
+++
+++

How did we complete the CCD Crosswalk build projects:

1. **Problem**
    1. Needed CCD information with reliable facts and dates for machine learning models
    2. **No single system** available to provide complete set of information
    3. A Crosswalk bridge is required to provide relevant information from both systems
    4. Requires, at minimum, EmtelliPro data and Availity data
    5. The query timeout limit is set at 60 minutes
    6. Current process is **so slow that it was timing out** before query completion
    7. Large data volume (multi-hundred millions rows) of data in each system
2. **Domain understanding**
    1. OAS received KT from CCD team to understand the data models and relationships and data types
    2. OAS and CCD decided to adopt a multi-pronged approach to ensure the optimum solution and expected outcome
3. **Approach**
    1. Worked very closely with the CCD team SMEs
    2. OAS started with **7-8 different POCs**
    3. Built **strong partnership with CCD team** SMEs to ensure proper guidance and course correction, when needed
    4. Finally we settled on two solutions - a). Python/Snowflake stored procedures and b). Stored procedures only.
    5. It was determined that **Python/Snowflake stored procedures** solution will be **implemented**
    6. Main **features** of the selected solution are:
        a. **Chunking framework** to divide large workloads into smaller chunks - process large workloads without query timeouts
        b. **No dependency on raw** XML_DOCUMENT and XML_DOCUMENT_COMBINED - 30% improvement in processing time
        c. Latest results show that the current process can **process ~1M CCDs in ~4 minutes**
        d. **No redundant processing** of the same data -
            i. skipped already processed data by keeping track of processing dates
            ii. ability to re-process any range of historical data
        e. Flexibility and improved change response - **modular and configuration based approach** made system less error-prone and quicker response to change
        f. The CCD Crosswalk build process could be scheduled, event-based triggered or run on demand
    7. Created a **run book** for easier migration, configuration, operations and maintenance

    8. **Migrated solution** code and related artifacts to a **GitHub repository** to ensure version control and proper audits
    9. **Working closely with the CCD IT team** to ensure smooth production deployment, operations, and beyond

+++

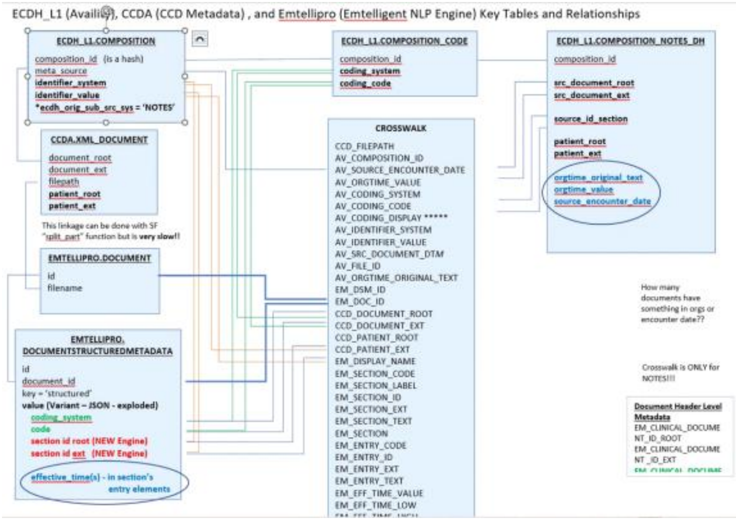
Composition (Availity)			Emtelligent DocumentStructuredMetadata Table			
Compostion Table	Composition column		FOR_ELT_TAG	CODE_ELT_TAG	value column (variant - JSON)	Comment
COMPOSITION_NOTES_DH	src_document_root	↔	ClinicalDocument	code	PARSE_JSON(dm.value):id[0].root::string	CCD **document** level only
COMPOSITION_NOTES_DH	src_document_ext	↔	ClinicalDocument	code	PARSE_JSON(dm.value):id[0].extension::string	CCD **document** level only
COMPOSITION_CODE	coding_system	↔	section	code	PARSE_JSON(dm.value):code_system::string	
COMPOSITION_CODE	coding_code	↔	section	code	PARSE_JSON(dm.value):code::string	
COMPOSITION	identifier_system	↔	section	code	PARSE_JSON(dm.value):id[0].root::string	
COMPOSITION	identifier_value	↔	section	code	PARSE_JSON(dm.value):id[0].extension::string	

++

- Implement a crosswalk build process that runs successfully
- Develop a chunking framework to process large data volumes
- Develop parallel processing framework to process large workloads efficiently and timely
- Perform as much pre-processing as practical to minimize build duration
- Achieve balance between cost and crosswalk build duration
- Allow re-process any range of historical data
- Adopt configurable and modular approach to enable quicker response to change

```
    -- *** UPDATED ***
      (Select EM_DOC_ID, ROOT_ID as DOCUMENT_ROOT, ROOT_EXT  AS DOCUMENT_EXT,
         E1.eff_time_value as EM_CLINICAL_DOCUMENT_EFF_TIME
      From E1
      WHERE for_elt_tag = 'ClinicalDocument'
      AND   code_elt_tag = 'code'
      ),--select * from E2
    -- *** ORGINAL ***
    (Select DISTINCT EM_DOC_ID, ROOT_ID as DOCUMENT_ROOT, ROOT_EXT  AS DOCUMENT_EXT,
       E1.eff_time_value as EM_CLINICAL_DOCUMENT_EFF_TIME
    From E1
    WHERE for_elt_tag = 'ClinicalDocument'
    ),
  E21 AS
    /* Get Encompassing Encounter related info. */
  (Select DISTINCT EM_DOC_ID, E1.ROOT_ID as EM_ENCOMPASSING_ENCOUNTER_ID_ROOT,
    E1.ROOT_EXT as EM_ENCOMPASSING_ENCOUNTER_ID_EXT,
    E1.eff_time_low as EM_ENCOMPASSING_ENCOUNTER_EFF_TIME_LOW,
    E1.eff_time_high as EM_ENCOMPASSING_ENCOUNTER_EFF_TIME_HIGH
  From E1
  Where E1.for_elt_tag = 'encompassingEncounter'
  ),
E3 as
  /* Segregates Section level info. from E1. */
  (Select DISTINCT EM_DSM_ID, EM_DOC_ID, CODE AS SECTION_CODE,
    E1.CODING_SYSTEM, LABEL AS SECTION_LABEL,
    ROOT_ID AS SECTION_ID, ROOT_EXT AS SECTION_EXT,
    E1.TEMPLATE_ID_ROOT, snn.SECTION_NAME_NORMALIZED
  From E1
  LEFT JOIN OCDP_PRD_HEALTHCARE_ECONOMICS_DB.EMTELLIPRO_ST_RD.SECTION_NAMES_NORMALIZED snn
```

```
    ON A11.COMPOSITION_ID       = E32.COMPOSITION_ID
    AND A11.SRC_COMPOSITION_ID  = E32.SRC_COMPOSITION_ID
  -- *** OK TO THIS POINT, PICKS UP THE TWO ROWS ***
  LEFT OUTER JOIN E21
    /* Join to get Encompassing Encounter info., if available. */
   ON E32.EM_DOC_ID    = E21.EM_DOC_ID
   -- *** OK TO THIS POINT, PICKS UP THE TWO ROWS ***
   -- ****************** BELOW SEEMS TO BE THE ISSUE.  WORKS FOR BOTH CASES WHEN COMMENTED OUT. CHECK INTO WHY AND IF THIS IS CORRECT THING TO DO ******************
   --WHERE E32.CODE_ELT_TAG = 'code'
  )
  ---
  /* Final outcome with intuitive aliases and appropriate sequence. */
  Select SRC_DOCUMENT_ROOT                as AV_DOCUMENT_ROOT,
     SOURCE_ID_EXT                        as AV_DOCUMENT_EXT,
     patient_root                         as AV_PATIENT_ROOT,
     patient_ext                          as AV_PATIENT_EXT,
     COMPOSITION_ID                       as AV_COMPOSITION_ID,
     SRC_COMPOSITION_ID                   as AV_SRC_COMPOSITION_ID,
     CODING_SYSTEM                        as AV_CODING_SYSTEM,
     coding_code                          as AV_CODING_CODE,
     identifier_system                    as AV_IDENTIFIER_SYSTEM,
     identifier_value                     as AV_IDENTIFIER_VALUE,
     src_document_dtm                     as AV_SRC_DOCUMENT_DTM,
     OBSTIME_VALUE                        as AV_OBSTIME_VALUE,
     ORGTIME_ORIGINAL_TEXT                as AV_ORGTIME_ORIGINAL_TEXT,
     ORGTIME_VALUE                        as AV_ORGTIME_VALUE,
     SOURCE_ENCOUNTER_DATE                as AV_SOURCE_ENCOUNTER_DATE,
     EM_DSM_ID                            as EM_DSM_ID,
     EM_DOC_ID                            as EM_DOC_ID,
     DOCUMENT_ROOT                        as EM_CLINICAL_DOCUMENT_ID_ROOT,
```



++

SP_BUILD_CCD_CROSSWALK

Row numbers between 1 and 77624 successfully processed.

Query Details ...

Query duration 2m 2s

Rows 1

Query ID 01b17f3a-0a05-7bd7-0...

SP_BUILD_CCD_CROSSWALK A

100% filled

++

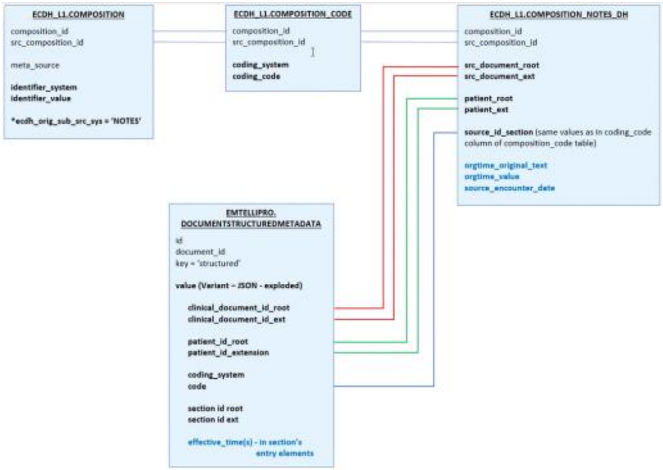
```
with p1 as
(
            Select
                dm.ID                                                    AS EM_DSM_ID,
                dm.document_id,
                --E0.EM_DOC_ID,
                --'b9aba4f8-c970-4509-bc20-488571d9817b' as EM_DOC_ID,
//                PARSE_JSON(dm.value):code_system_name::string    as coding_system_name,
//                PARSE_JSON(dm.value):code_system::string         as coding_system,
//                PARSE_JSON(dm.value):code::string                as code,
//                PARSE_JSON(dm.value):display_name::string        as display_name,
//                PARSE_JSON(dm.value):template_id[0].root::string as template_id_root,
                PARSE_JSON(dm.value):for_elt_tag::string         as for_elt_tag,
                PARSE_JSON(dm.value):code_elt_tag::string        as code_elt_tag,
//                PARSE_JSON(dm.value):section::string             as section,
                PARSE_JSON(dm.value):id[0].root::string          as root_id,
                PARSE_JSON(dm.value):id[0].extension::string     as root_ext
//                PARSE_JSON(dm.value):label::string               as label,
//                PARSE_JSON(dm.value):text_id::string             as text_id,
//                PARSE_JSON(dm.value):effective_time[0].value::string as eff_time_value,
//                PARSE_JSON(dm.value):effective_time[0].low.value::string as eff_time_low,
//                PARSE_JSON(dm.value):effective_time[0].high.value::string as eff_time_high
            FROM
             -- ***** 1 Million Documents Schema *****
             "OCDP_PRD_HEALTHCARE_ECONOMICS_DB"."EMTELLIPRO"."DOCUMENTSTRUCTUREDMETADATA" dm
            WHERE for_elt_tag = 'patientRole' and code_elt_tag = 'patient'
)
select count(root_id) from p1 where root_id is not null


select count(id) from "OCDP_PRD_HEALTHCARE_ECONOMICS_DB"."EMTELLIPRO"."DOCUMENT"
```

++

```
with p1 as
(
            Select
                dm.ID                                                    AS EM_DSM_ID,
                dm.document_id,
                --E0.EM_DOC_ID,
//                PARSE_JSON(dm.value):code_system_name::string    as coding_system_name,
//                PARSE_JSON(dm.value):code_system::string         as coding_system,
//                PARSE_JSON(dm.value):code::string                as code,
//                PARSE_JSON(dm.value):display_name::string        as display_name,
//                PARSE_JSON(dm.value):template_id[0].root::string as template_id_root,
                PARSE_JSON(dm.value):for_elt_tag::string         as for_elt_tag,
                PARSE_JSON(dm.value):code_elt_tag::string        as code_elt_tag,
//                PARSE_JSON(dm.value):section::string             as section,
                PARSE_JSON(dm.value):id[0].root::string          as root_id,
                PARSE_JSON(dm.value):id[0].extension::string     as root_ext
//                PARSE_JSON(dm.value):label::string               as label,
//                PARSE_JSON(dm.value):text_id::string             as text_id,
//                PARSE_JSON(dm.value):effective_time[0].value::string as eff_time_value,
//                PARSE_JSON(dm.value):effective_time[0].low.value::string as eff_time_low,
//                PARSE_JSON(dm.value):effective_time[0].high.value::string as eff_time_high
            FROM
             -- ***** 1 Million Documents Schema *****
             "OCDP_PRD_HEALTHCARE_ECONOMICS_DB"."EMTELLIPRO"."DOCUMENTSTRUCTUREDMETADATA" dm
            WHERE for_elt_tag = 'patientRole' and code_elt_tag = 'patient'
)
select count(root_id) from p1 where root_id is not null
```

```
A11 as  /* Availity Composition data */
    (
    Select DISTINCT c.COMPOSITION_ID, c.SRC_COMPOSITION_ID,
       COALESCE(c.identifier_system, '*') as identifier_system,
       COALESCE(c.identifier_value, '*') as identifier_value,
       c.SRC_DOCUMENT_DTM
    FROM OCDP_PRD_OPTUMCARE_CORE_DB.ECDH_L1.COMPOSITION c
    WHERE c.ECDH_ORIG_SUB_SRC_SYS = 'NOTES'
    )
```

++



++

```sql
        ),
    E21 AS
        /* Get Encompassing Encounter related info. */
    (Select DISTINCT EM_DOC_ID, E1.ROOT_ID as EM_ENCOMPASSING_ENCOUNTER_ID_ROOT,
        E1.ROOT_EXT as EM_ENCOMPASSING_ENCOUNTER_ID_EXT,
        E1.eff_time_low as EM_ENCOMPASSING_ENCOUNTER_EFF_TIME_LOW,
        E1.eff_time_high as EM_ENCOMPASSING_ENCOUNTER_EFF_TIME_HIGH
    From E1
    Where E1.for_elt_tag = 'encompassingEncounter'
      AND E1.code_elt_tag = 'code'
    ),
    -- *******************************************************************
    -- ******** I ADDED THIS FOR PATIENT ID ROOT AND EXTENSION!!! ********
    -- *******************************************************************
    E22 AS
        /* Get Patient id root and ext  info. */
    (SELECT
            DISTINCT
            EM_DOC_ID,
            E1.ROOT_ID  as EM_PATIENT_ROOT,
            E1.ROOT_EXT as EM_PATIENT_EXT
    From    E1
    WHERE   for_elt_tag = 'patientRole'
    AND     code_elt_tag = 'patient'
    ),
    --select * from E22
  E3 as
    /* Segregates Section level info. from E1. */
    (Select DISTINCT EM_DSM_ID, EM_DOC_ID, CODE AS SECTION_CODE,
        E1.CODING_SYSTEM, LABEL AS SECTION_LABEL,
        COALESCE(ROOT_ID, '*') AS SECTION_ID, COALESCE(ROOT_EXT, '*') AS SECTION_EXT,
        E1.TEMPLATE_ID_ROOT, snn.SECTION_NAME_NORMALIZED
    From E1
    LEFT JOIN OCDP_PRD_HEALTHCARE_ECONOMICS_DB.EMTELLIPRO_ST_RD.SECTION_NAMES_NORMALIZED snn
        /* Get standardized template name, if available. */
        ON E1.TEMPLATE_ID_ROOT = snn.TEMPLATE_ID
    WHERE  E1.for_elt_tag = 'section'
```

```sql
        E32.ORGTIME_VALUE,
        E32.ENTRY_CODE,
        E32.DISPLAY_NAME,
        E32.ENTRY_ID,
        E32.ENTRY_EXT,
        E32.TEXT_ID,
        E32.eff_time_value,
        E32.eff_time_low,
        E32.eff_time_high,
        E2.DOCUMENT_ROOT,
        E2.DOCUMENT_EXT,
        E2.EM_CLINICAL_DOCUMENT_EFF_TIME,
        E21.EM_ENCOMPASSING_ENCOUNTER_ID_ROOT,
        E21.EM_ENCOMPASSING_ENCOUNTER_ID_EXT,
        E21.EM_ENCOMPASSING_ENCOUNTER_EFF_TIME_LOW,
        E21.EM_ENCOMPASSING_ENCOUNTER_EFF_TIME_HIGH
    FROM E32
    Join E2
        /* Join to get ClinicalDocument info. */
      ON  E32.SRC_DOCUMENT_ROOT   = E2.DOCUMENT_ROOT
      And E32.SOURCE_ID_EXT       = E2.DOCUMENT_EXT

    -- *************** ADDED FOR PATIENT INFO *****************
    JOIN E22
      on  E32.EM_DOC_ID = E22.EM_DOC_ID
    -- ********************************************************
select * from E4
     LEFT OUTER JOIN A11
       ON A11.COMPOSITION_ID      = E32.COMPOSITION_ID
       AND A11.SRC_COMPOSITION_ID = E32.SRC_COMPOSITION_ID
    LEFT OUTER JOIN E21
        /* Join to get Encompassing Encounter info., if available. */
      ON E32.EM_DOC_ID = E21.EM_DOC_ID
     )
     --select * from E4 -- TODO: PRODUCES NO RESULTS
    /* Final outcome with intuitive aliases and appropriate sequence. */
    Select SRC_DOCUMENT_ROOT                          as AV_DOCUMENT_ROOT,
        SOURCE_ID_EXT                                 as AV_DOCUMENT_EXT,
        patient_root                                  as AV_PATIENT_ROOT,
        --em_patient_root,
        patient_ext                                   as AV_PATIENT_EXT,
        --em_patient_ext,
        COMPOSITION_ID                                as AV_COMPOSITION_ID,
        SRC_COMPOSITION_ID                            as AV_SRC_COMPOSITION_ID,
        CODING_SYSTEM                                 as AV_CODING_SYSTEM,
        coding_code                                   as AV_CODING_CODE,
        identifier_system                             as AV_IDENTIFIER_SYSTEM,
```

```sql
    -- *************** ADDED FOR PATIENT INFO *****************
    JOIN E22
       on  E32.EM_DOC_ID = E22.EM_DOC_ID
    -- *******************************************************
-) select * from E4
     LEFT OUTER JOIN A11
        ON A11.COMPOSITION_ID      = E32.COMPOSITION_ID
        AND A11.SRC_COMPOSITION_ID = E32.SRC_COMPOSITION_ID
    LEFT OUTER JOIN E21
        /* Join to get Encompassing Encounter info., if available. */
       ON E32.EM_DOC_ID = E21.EM_DOC_ID
     )
     --select * from E4 -- TODO: PRODUCES NO RESULTS
    /* Final outcome with intuitive aliases and appropriate sequence. */
    Select SRC_DOCUMENT_ROOT                          as AV_DOCUMENT_ROOT,
        SOURCE_ID_EXT                                 as AV_DOCUMENT_EXT,
        patient_root                                  as AV_PATIENT_ROOT,
        em_patient_root,
        patient_ext                                   as AV_PATIENT_EXT,
        em_patient_ext,
        COMPOSITION_ID                                as AV_COMPOSITION_ID,
        SRC_COMPOSITION_ID                            as AV_SRC_COMPOSITION_ID,
        CODING_SYSTEM                                 as AV_CODING_SYSTEM,
        coding_code                                   as AV_CODING_CODE,
        identifier_system                             as AV_IDENTIFIER_SYSTEM,
        identifier_value                              as AV_IDENTIFIER_VALUE,
        src_document_dtm                              as AV_SRC_DOCUMENT_DTM,
        OBSTIME_VALUE                                 as AV_OBSTIME_VALUE,
        ORGTIME_ORIGINAL_TEXT                         as AV_ORGTIME_ORIGINAL_TEXT,
        ORGTIME_VALUE                                 as AV_ORGTIME_VALUE,
        SOURCE_ENCOUNTER_DATE                         as AV_SOURCE_ENCOUNTER_DATE,
        EM_DSM_ID                                     as EM_DSM_ID,
        EM_DOC_ID                                     as EM_DOC_ID,
        DOCUMENT_ROOT                                 as EM_CLINICAL_DOCUMENT_ID_ROOT,
        DOCUMENT_EXT                                  as EM_CLINICAL_DOCUMENT_ID_EXT,
        EM_CLINICAL_DOCUMENT_EFF_TIME                 as EM_CLINICAL_DOCUMENT_EFF_TIME,
        EM_ENCOMPASSING_ENCOUNTER_ID_ROOT              as EM_ENCOMPASSING_ENCOUNTER_ID_ROOT,
        EM_ENCOMPASSING_ENCOUNTER_ID_EXT               as EM_ENCOMPASSING_ENCOUNTER_ID_EXT,
        EM_ENCOMPASSING_ENCOUNTER_EFF_TIME_LOW         as EM_ENCOMPASSING_ENCOUNTER_EFF_TIME_LOW,
        EM_ENCOMPASSING_ENCOUNTER_EFF_TIME_HIGH        as EM_ENCOMPASSING_ENCOUNTER_EFF_TIME_HIGH,
        DISPLAY_NAME                                  as EM_DISPLAY_NAME,
        SECTION_NAME_NORMALIZED                       as EM_SECTION_NAME_NORMALIZED,
        SECTION_CODE                                  as EM_SECTION_CODE,
        SECTION_LABEL                                 as EM_SECTION_LABEL,
        SECTION_ID                                    as EM_SECTION_ID,
        SECTION_EXT                                   as EM_SECTION_EXT,
        TEXT_ID                                       as EM_TEXT_ID,
```

+++

To convert your Python code into a Databricks notebook, follow these steps:

1. Open your Python code in a text editor or IDE.

2. Create a new Databricks notebook in your Databricks workspace.

3. In the first cell of the Databricks notebook, add the necessary imports and configurations. For example:

```python
# Databricks notebook source
from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()
```

4. Copy and paste the remaining Python code into separate cells in the Databricks notebook. Each cell should contain a logical unit of code.

5. If your code uses any external libraries or dependencies, make sure to include the necessary installation commands in a separate cell. For example:

```python
# Databricks notebook source
!pip install pandas
```

6. Review and adjust the code as needed to ensure it will run correctly in the Databricks environment. Keep in mind that Databricks notebooks use Spark, so you may need to modify your code to take advantage of distributed computing capabilities.

7. Save the Databricks notebook.

8. Execute the code by running each cell individually or by running all cells at once.

By following these steps, you can convert your Python code into a Databricks notebook and take advantage of the Databricks environment's distributed computing capabilities.
+++
Dinesh, we wanted to provide you the latest update on the CCD Crosswalk build project. We were able to complete the development and any modifications CCD team wanted. The latest run stats of the CCD Crosswalk build process are: 78K NYHA CCDs take ~3 minutes and 1M+ CCDs take 9-10 minutes. The client seems to be happy with the overall results.

The next stage would be to transition this project over to the operations, maintenance and management team(s). It is taking longer than expected. But we are working with Sangeetha Anandkumar and her team to prepare them to take the ownership. They are reviewing the code and the rest of the artifacts and we're there to support them, when needed. We need to get commitment from them of the transition completion, which we hope to get in the next meeting. We will provide all necessary help and guidance for the successful transition. Thank you.

Meetings

Monday, October 9, 2023 9:02 AM

Comments on two questions listed for the CCD, CCDA and EmtelliPro Meeting

NB Nystrom, Brian
Reply | Reply All | Forward
To Cheema, Dave; Palla, Sandeep; Bangale, Gaurav; Ramakrishnan, Anantha; Ellis, William H
Mon 10/9/2023 8:34
Retention Policy UHGInbox (90 days) Expires 1/7/2024

All,

Below are comments on a couple of questions listed in the agenda for today's **CCD, CCDA and EmtelliPro** Meeting:

QUESTION: Data element in the Emtelligent Pro output that represents the document ID/Document_root in XML_Document in CCDA?

- Emtellipro output does **not** contain **document_root** metadata from inbound CCDs
- The Emtellipro **document** table's **id** column is **not** represented in the **ccda.xml_document** table
- The two facts above made it necessary to link the **ccda.xml_document** table's **filepath** to the **emtellipro.document** table's **filename** by leaf nodes to tie the two tables together

QUESTION: Duplicate data in CCDA XML_DOCUMENT - multiple instances of the same DOCUMENT_ROOT id. This is a bridge between Availity and EmtelliPro engine data

- There can be multiple documents in the **ccda.xml_document** table with the same **document_root**. The same CCD document may be processed multiple times.

Regards,

Brian Nystrom
Sr Healthcare Econ Cnslt, HCE-Population Health Analytics | Optum Health

++

Meeting 11/02/2023 with CCD team:

- xml_document is not source of truth
- Availity never sends the full data set
- xml_document_combined has 2023 data
- Will never match 100%
- Getting ready to migrate, ODX will be sunset
- XML_DOCUMENT_METADATA is not up-to-date
- Get data from both tables XML_DOCUMENT and XML_DOCUMENT_COMBINED
- To process historical data, it will depend on resources available and cost factor --> Caroline and Leanne will decide

++

XML Combined schedule

HG Holt, Gregory
Reply | Reply All
To Ellis, William H; Nystrom, Brian; Cheema, Dave
Retention Policy UHGInbox (90 days) Expires 2/1/2024
Start your reply all with: Working on it now. | Thank you! | Completed. | Feedback

Good afternoon, Team! Just wanted to follow up to send the current loading schedules for the XML_DOCUMENT_COMBINED tables:

We have (2) processes that are needed to be run before we can say the load is complete.

(Process 1 – Discovery of the new files – Schedule, 4:30 AM CST):

Schedule

At 09:30 AM (UTC+00:00 — UTC)

Edit schedule | Pause | Delete

(Process 2 – Actual loading of files into snowflake from the container, 1PM CST):

Schedule

At 06:00 PM (UTC+00:00 — UTC)

Edit schedule | Pause | Delete

++

Meeting 12/11/2023 with Brian and Bill

1. Duplicates
   A. Remove AV_CODING_DISPLAY column from crosswalk
   B. Distinct\Group By\Windowing on the remaining columns to dedupe? left w 68000
2. Normalization of Section Names - Need new Column and Lookup table
   A. Create a new column EM_DISPLAY_NAME_NORMALIZED, and keep EM_DISPLAY_NAME
   B. VaLue comes from Emtelligent oid_to_section_name_mappings.tsv file mapping. This should be made into a Snowflake table.
   C. Lookup done by section Template Id -- EX: <templateId root="1.3.6.1.4.1.19376.1.5.3.1.3.4"> or section code 10164-2

   INCLUDE TSV

3. CCD Document Header Level Metadata
    A. ClinicalDocument (CCD root element)
        1. id root and id extension available in documentstructuredmetadata
        2. effectiveTime in available in documentstructuredmetadata
    B. encompassingEncounter elemnt and children available in documentstructuredmetadata
        1. effectiveTime.low value attribute
        2. effectiveTime.high value attribute
4. Composition query in Artifacts
    A. "source_" columns usage. LEAVE AS IS
    B. Need composition and composition code table joins? YES, LEAVE AS IS
5. CCDA XML_DOCUMENT Usage
    A. Needed in light of point 3. above?
6. Eliminate text variant columns

++

```
SELECT
    dm.id,
    dm.document_id,
    PARSE_JSON(dm.value):code_system_name::string          as coding_system_name,
    PARSE_JSON(dm.value):code::string                      as code,
    PARSE_JSON(dm.value):display_name::string              as display_name,
    PARSE_JSON(dm.value):for_elt_tag::string               as for_elt_tag,
    PARSE_JSON(dm.value):code_elt_tag::string              as code_elt_tag,
    PARSE_JSON(dm.value):template_id                       as template_id,    -- but can be multiple template ids used!!!
    --PARSE_JSON(dm.value):template_id[0].root::string       as template_id_root,    -- but can be multiple template ids used!!!
    --PARSE_JSON(dm.value):template_id[0].extension::string    as template_id_extension,-- but can be multiple template ids used!!!
    PARSE_JSON(dm.value):section::string                   as section,
    parse_json(dm.value):id[0].root::string                as id,
    parse_json(dm.value):id[0].extension::string           as extension,
    PARSE_JSON(dm.value):label::string                     as label,
    PARSE_JSON(dm.value):text_id::string                   as text_id,
    PARSE_JSON(dm.value):effective_time[0].value::string     as etime_value,
    PARSE_JSON(dm.value):effective_time[0].low.value::string  as etime_low,
    PARSE_JSON(dm.value):effective_time[0].high.value::string as etime_high
FROM   "OCDP_PRD_HEALTHCARE_ECONOMICS_DB"."EMTELLIPRO_ST_FINAL"."DOCUMENTSTRUCTUREDMETADATA" dm
WHERE
    -- 10/17/2023 - new doc id for canonical document is now b9aba4f8-c970-4509-bc20-488571d9817b
    DOCUMENT_ID = 'b9aba4f8-c970-4509-bc20-488571d9817b' --order by for_elt_tag
and  (for_elt_tag = 'ClinicalDocument' or for_elt_tag = 'encompassingEncounter' or for_elt_tag = 'section' or for_elt_tag = 'serviceEvent') -- or for_elt_tag = 'act')
and  code_elt_tag = 'code'
```

++++++++++++++++++

section_nor
malizatio...

Section standardized names file

++

Composition (Availity)			DocumentStructuredMetadata		
Compostion Table	**Composition column**		**FOR_ELT_TAG**	**CODE_ELT_TAG**	**Column**
COMPOSITION_NOTES_DH	src_document_root	↔	ClinicalDocument	code	ID
COMPOSITION_NOTES_DH	src_document_ext	↔	ClinicalDocument	code	EXTENSION
COMPOSITION_CODE	coding_system	↔	section	code	coding_system
COMPOSITION_CODE	coding_code	↔	section	code	code
COMPOSITION	identifier_system	↔	section	code	ID
COMPOSITION	identifier_value	↔	section	code	EXTENSION

Solution Approach

Thursday, October 26, 2023 1:11 PM

- The Approach for Crosswalk
- Solution to store / aggregate table & a sample data model
- Estimate for completing the remaining over and timeframe
- Assumptions & open items -
    - 78K out of 53 million?
    - SLA / batched over predetermined Frequency than stream / continuous .
    - Only subset are loaded, CCD are processed partially crosswalk qulaity
- Deliverables at end of 12 weeks
    - Robust crosswalk -
    - Guidelines for using SF – Pro /con as solution technology / alternates if any
    - Guidelines for on-going maintenance of x-walk & Data model

Suggested operating model - < IT team might not own the responsibility?>

++

Activities to build CCD Crosswalk Process and data model

Review and clarify requirements, especially:
- Which particular concept types to filter CCD documents on
- Volume and frequency of data
- Volume variability among CCDA data, EmtelliPro data and Availity data

Review POC queries and make them production-ready code
Apply best practices
Create data model of process control and Crosswalk entities
Review data model with CCD team and incorporate their feedback
Create Crosswalk building steps
Create crosswalk of specified entities
Convert interactive queries into stored procedure(s)
Implement a framework to break the whole workload into multiple manageable mini workload
Build a driver and a Crosswalk engine process to divide up workload in small manageable workloads
Test data models, production-ready code, stored procedures, framework to divide up workload into manageable mini workloads, and driver and Crosswalk engine
Document results
Review with the CCD SMEs and incorporate their feedback
Create an automation job to run the CCD Crosswalk build process
Enable CCD Crosswalk build process to run on demand as well as in a scheduled mode
Test automation job to run the CCD Crosswalk build process
Experiment and determine the optimum VW size and number for the optimum process time
Make all components production-ready, e.g., package all components, pointing to the right environment, proper governance model, a smoke test in the production environment
Communicate to CCD team of its readiness and ensure smoke test works as expected
Create a POC to run mini workloads in parallel
Create runbook of CCD Crosswalk process

SSH Keys

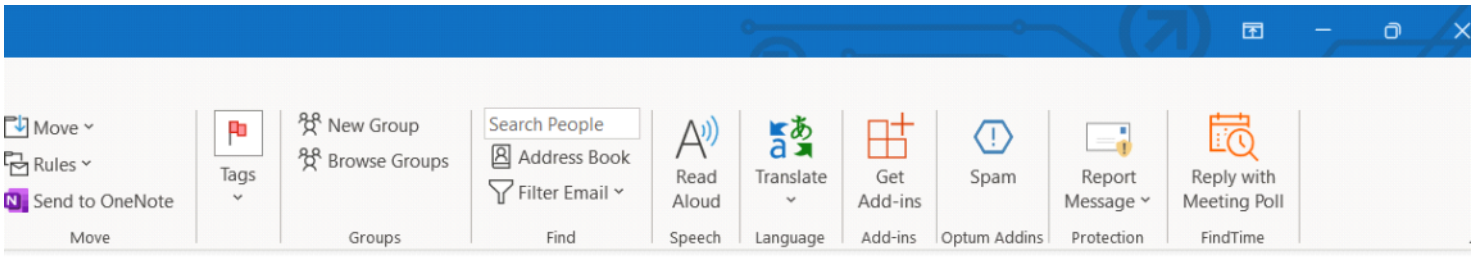
Sunday, October 29, 2023 2:42 PM

rsa_key

rsa_key

Emails

Thursday, November 2, 2023 9:35 AM



RE: CCD Crosswalk - Operational and division of responsibility concerns

Reply | Reply All | Forward

Ramakrishnan, Anantha
To Cheema, Dave; Mandal, Aditya; Boinpally, Harish
Cc Palla, Sandeep; Bangale, Gaurav; Shin, John S; Nand, Durga
Thu 11/2/2023 9:05 AM

Retention Policy UHGInbox (90 days) Expires 1/31/2024

++ Durga

Regards
Ananth

Anantha Ramakrishnan
Sr. Director, Digital Transformations | Optum

M 1-551-358-4609
anantha.ramakrishnan@optum.com

Optum

From: Ramakrishnan, Anantha
Sent: Thursday, November 2, 2023 10:04 AM
To: Cheema, Dave <dave.cheema@optum.com>; Mandal, Aditya <aditya.mandal@optum.com>; Boinpally, Harish <harish_boinpally@optum.com>
Cc: Palla, Sandeep <sandeep.palla@optum.com>; Bangale, Gaurav <gaurav.bangale@optum.com>; Shin, John S <john.shin@optum.com>
Subject: RE: CCD Crosswalk - Operational and division of responsibility concerns

Dave & team

Thank you for analytical thinking.

I would suggest we position this differently, instead asking the questions, would take stake in the ground for each of the questions

- Who is going to deploy it? –
  a. The solution would be designed with scripts, that would allow separation of duties to deploy through 'xxx' tools. We would like to confirm whether this would satisfy the deployment ? – we can also get guidance from Durga on what tools/ controls to be in place.
- Who would be responsible for the deployment? If there are multiple teams involved, e.g., infrastructure team, database administration team and executable deployment team, who would coordinate it? –
  a. We have xxx need for database, script and to be deployed in this environment – who do we need to coordinate ?
- What would the deployment team(s) would require from us?
  a. What documentations are required if our solution entails ,a,b,c
  b. What lead time would be required?
- Who would maintain/extend it?
  a. The script will
     i. require configuration change, of types a,b,c,
     ii. require performance tuning by varying thread / parallel number etc.
     iii. require these config / files to be renamed for rerunning in case of error/ inadvertent issues,
     iv. require review of logs.
     v. require a, b,c skills for doing this)- what & who will maintain

iii. require these config / files to be renamed for rerunning in case of error/ inadvertent issues,
iv. require review of logs.
v. require a, b,c skills for doing this)- what & who will maintain

Modify the questions below along the samelines

- Who will be responsible for the operations of it?
- How would the running and rerunning be facilitated?
- How and how often the notifications be generated when the same data is ready to be run in all three systems (EmtelliPro, Availity, and CCDA)?
- How will the build process completion be notified to its stakeholders?
- What would be the expected volume and frequency of data?
- We need to experiment with the numbers and sizes of VW clusters to be able to achieve the acceptable results, within acceptable timeframe, at the reasonable price point. How will that work – will we have permissions to create different combinations or somebody else will create them for us? If somebody else, what will be the turnaround time for it?

Hope this makes sense. Will demonstrate product engineering depth.

Yes I agree with starting with Bill and Bryan. But we have to architect the solution rather than asking these questions.

Regards
Ananth

Anantha Ramakrishnan
Sr. Director, Digital Transformations | Optum

M 1-551-358-4609
anantha.ramakrishnan@optum.com

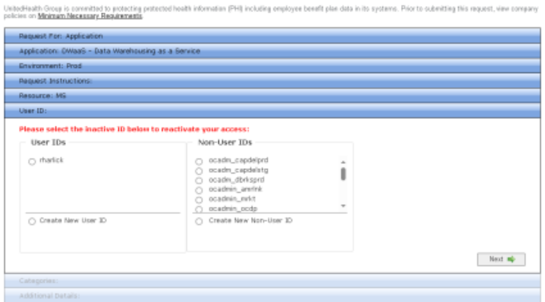
Optum

Items for CCD Crosswalk Build to Prod

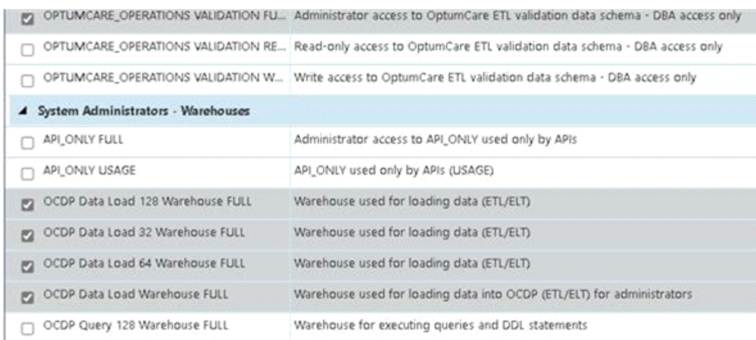
Monday, December 4, 2023 7:05 PM

Need access to:

1. Non-user credentials - done
2. Be able to login to Snowflake using non-user credentials - done



3. Data_Load_* warehouses - done
4. Infrastructure (VW) sizing and cost estimates (Need Admin access - done)



☑	OPTUMCARE_OPERATIONS VALIDATION FU...	Administrator access to OptumCare ETL validation data schema - DBA access only
☐	OPTUMCARE_OPERATIONS VALIDATION RE...	Read-only access to OptumCare ETL validation data schema - DBA access only
☐	OPTUMCARE_OPERATIONS VALIDATION W...	Write access to OptumCare ETL validation data schema - DBA access only
◢	System Administrators - Warehouses	
☐	API_ONLY FULL	Administrator access to API_ONLY used only by APIs
☐	API_ONLY USAGE	API_ONLY used only by APIs (USAGE)
☑	OCDP Data Load 128 Warehouse FULL	Warehouse used for loading data (ETL/ELT)
☑	OCDP Data Load 32 Warehouse FULL	Warehouse used for loading data (ETL/ELT)
☑	OCDP Data Load 64 Warehouse FULL	Warehouse used for loading data (ETL/ELT)
☑	OCDP Data Load Warehouse FULL	Warehouse used for loading data into OCDP (ETL/ELT) for administrators
☐	OCDP Query 128 Warehouse FULL	Warehouse for executing queries and DDL statements

Repo: **Pop Health repo** - done
Non-user account (use existing: **ocemtel_ocdpprd@optum.com**) - done
Get non-user SSH credentials - done
Change ownership of DB objects to non-user account - credentials - done; to migrate objects over and change DB connection to non-user credentials
Refresh cycle: once/day (for now)
Schema we need access to: done

1. EMTELLIPRO_ST_RD - Admin
2. EMTELLIPRO_ST_FINAL - Read
3. CCDA - Read
4. ECDH_L1 - Read

- Snowflake capacity provisioning - need to understand the procedure
- Volume (1+ M CCDs) test data -done
  - Key vault credentials
  - Databricks instance
- Environment provisioning for UAT and prod environments - Need to work with David M.
- Automation script (scheduled/manual) (most likely use Databricks scheduling capabilities)
- CICD process need guidance
- Runtime configurations, e.g., predecessors (EmtelliPro, Availity, XML Document pre-processing completion
  Work with David M. to determine that
- Default starting date to start the CCD Crosswalk process - After CCD_BUILD_HISTORY table is created (Default values: '2018-01-01 00:00:00', 2000, 300, current_timestamp()). - dependency on table definition completion. OAS - part of deployment script
  - To run this job successfully, the following tables must have relevant data (jobs that populate these tables must be completed):

Schema	Table name
CCDA	XML_DOCUMENT_COMBINED
CCDA	XML_DOCUMENT (Note: XML_DOCUMENT_COMBINED & XML_DOCUMENT tables are being joined to create a one unique CCDs set)
EMTELLIPRO_ST_FINAL	DOCUMENT
EMTELLIPRO_ST_FINAL	PROCESSINGDETAILS
EMTELLIPRO_ST_FINAL	DOCUMENTSTRUCTUREDMETADATA
ECDH_L1	COMPOSITION_NOTES_DH
ECDH_L1	COMPOSITION_CODE
ECDH_L1	COMPOSITION

++

- GitHub (**pop-health-ccd-crosswalk)** Repo - https://github.com/optum-care/pop-health-ccd-crosswalk/tree/main/
- Non-user account and credentials - use existing account: ocemtel_ocdpprd@optum.com
- Snowflake databases and schemas - identified
  - CCD Crosswalk build process has dependency of the following schemas and tables:

Schema	Table name

EMTELLIPRO_ST_FINAL	DOCUMENT
EMTELLIPRO_ST_FINAL	PROCESSINGDETAILS
EMTELLIPRO_ST_FINAL	DOCUMENTSTRUCTUREDMETADATA
ECDH_L1	COMPOSITION_NOTES_DH
ECDH_L1	COMPOSITION_CODE
ECDH_L1	COMPOSITION
EMTELLIPRO_ST_RD	**EMTELLIPRO_AVAILITY_CROSSWALK**
EMTELLIPRO_ST_RD	**CCD_BUILD_HISTORY**

Note:

a. Permanent and transient tables are created in the EMTELLIPRO_ST_RD schema only
b. You may have to switch EMTELLIPRO_ST_FINAL schema to EMTELLIPRO

- All deployable objects in the above-mentioned **pop-health-ccd-crosswalk** repo
- Infrastructure recommendations -
  - For regular workload: Medium VW and 50K CCD chunk size
  - For large workloads: Large VW and 100K CCD size on loads (over 1 million CCDs)
- Roles and warehouses:

  Role: ARD_PRD_OCEMTEL_OCDPPRD_OPTUM_ROLE

  Warehouse: OCDP_PRD_DATA_LOAD_64_WH/OCDP_PRD_DATA_LOAD_128_WH
- Configuration changes should be made in the CCDRuntimeParameters.py module
- There should be three (dev, integration/UAT, and production) environments
- CCD Crosswalk build process should be able to be scheduled, event-based triggered and manually kicked off. Start with manual trigger and the rest of the capability can be deferred, should that be necessary
  - **Note**: Initially, this job is expected to run once/per day, but later on it may change to be scheduled or event-based
- Automation script (scheduled/manual) (most likely use Databricks scheduling capabilities)