**ETL (Extract, Transform, and Load) Guide**

*Making Medicaid Data More Accessible Through Common Data Models and FHIR APIs*

For

Office of the Assistant Secretary for Planning and Evaluation and

The US National Library of Medicine

(HP-21-002\_NIH\_NLM)

August 15, 2022

Version 1.0

Contents

[ETL Implementation Guide 5](#_Toc115457525)

[Background 5](#_Toc115457526)

[The Environment 5](#_Toc115457527)

[Who this guide is written for 6](#_Toc115457528)

[What you should get out of this guide 6](#_Toc115457529)

[Why we chose Databricks 6](#_Toc115457530)

[Why we chose OMOP CDM 7](#_Toc115457531)

[Our development environment 7](#_Toc115457532)

[Data Use Agreement and User IDs 7](#_Toc115457533)

[How to run the code 7](#_Toc115457534)

[Notebook cell by notebook cell 7](#_Toc115457535)

[Notebook by notebook 7](#_Toc115457536)

[Launch 8](#_Toc115457537)

[JOB utility 8](#_Toc115457538)

[Starting the cluster 8](#_Toc115457539)

[How is the code structured? 9](#_Toc115457540)

[01\_DDL\_CDM 9](#_Toc115457541)

[01\_DDL\_E\_0000 9](#_Toc115457542)

[01\_DDL\_E\_YYYY 9](#_Toc115457543)

[02\_optimize 9](#_Toc115457544)

[A Class 9](#_Toc115457545)

[B Class 10](#_Toc115457546)

[Table 1. Notebooks and their outcomes 10](#_Toc115457547)

[Prerequisites 10](#_Toc115457548)

[User ID and DUA 10](#_Toc115457549)

[00\_Support [Third Party Data] 11](#_Toc115457550)

[Athena 11](#_Toc115457551)

[National Plan and Provider Enumeration System (NPPES) 11](#_Toc115457552)

[PX Handshake 12](#_Toc115457553)

[How do you start the deployment? 13](#_Toc115457554)

[Working with the CDM 15](#_Toc115457555)

[Origin Tables (TAF-RIF) 15](#_Toc115457556)

[Table 1. Original Tables 15](#_Toc115457557)

[Table 2. Destination Tables (OMOP) Supported by the ETL 15](#_Toc115457558)

[Unsupported or ‘Curious’ CDM features 15](#_Toc115457559)

[Visit\_Occurrence\_ID 15](#_Toc115457560)

[Finance 15](#_Toc115457561)

[Person\_ID 16](#_Toc115457562)

[Dates 16](#_Toc115457563)

[Claim to event relationship 16](#_Toc115457564)

[National Drug Codes (NDC) 16](#_Toc115457565)

[How to query the CDM 16](#_Toc115457566)

[Ideal Workflow 17](#_Toc115457567)

[1. Establish exposure of interest 17](#_Toc115457568)

[2. Reduce the CDM 17](#_Toc115457569)

[3. Evaluate the presence of a given feature within the population of interest. 17](#_Toc115457570)

[4. Alter and refine the exposure of interest to better reflect the peculiarities of the CDM. 17](#_Toc115457571)

[5. Draw a defensible conclusion from the study data. 17](#_Toc115457572)

[Tutorial Notebook 18](#_Toc115457573)

[Real World Example 18](#_Toc115457574)

[Update the CDM when new TAF-RIF years become available 18](#_Toc115457575)

[Reconciliation with SAS TAF-RIF files 18](#_Toc115457576)

[Using Databricks in VRDC/CCW 19](#_Toc115457577)

[What is Databricks? 19](#_Toc115457578)

[How is it different from SAS? 19](#_Toc115457579)

[Cost 19](#_Toc115457580)

[Complexity 20](#_Toc115457581)

[How is my Databricks instance set up? 20](#_Toc115457582)

[Databricks updates 20](#_Toc115457583)

[Resources, Tips and Troubleshooting 20](#_Toc115457584)

[1. SQL vs. Spark-SQL 20](#_Toc115457585)

[2. Spark-R is still under development 20](#_Toc115457586)

[3. Scala is currently not supported for security reasons 20](#_Toc115457587)

[4. Structured data storage 20](#_Toc115457588)

[5. Meta tables and proc datasets 21](#_Toc115457589)

[6. Storage is more complex than it seems 21](#_Toc115457590)

[7. Spark can take a ‘long’ time to write 21](#_Toc115457591)

[8. Spark clusters must be configured intentionally on an iterative basis 21](#_Toc115457592)

[9. User AWS buckets 21](#_Toc115457593)

[10. AWS Hive Garbage Collector 21](#_Toc115457594)

[11. Multiple table operations are different in Spark 22](#_Toc115457595)

# ETL Implementation Guide

## Background

Medicaid clinical documents are curated as Research Identifiable File (RIF) Transformed-MSIS(Medicaid Statistical Information System) Analytic extract Files (TAF-RIF) records. These records describe clinical information known to Medicaid via claims and payments for clinical services. The Medicaid ‘population’ is low income, high acuity (meaningfully ill), and a nationally distributed population. Using Medicaid records in research studies can ensure the study includes the nation’s most vulnerable people. Accessing and using Medicaid records is complicated with several barriers to success. Namely, the records themselves are designed to support Medicaid operations and financial professionals, not researchers.

Common Data Models (CDMs) are a means of integrating data across data sets for clinical studies. Under this award we created code that will transform TAF-RIF records into a CDM which research analysts may be more familiar with. By transforming TAF-RIF from a billing data model to a research data model we lower the barrier to entry and time to outcome for researchers using TAF-RIF records in their studies. For this effort, we selected the Observational Medical Outcomes Partnership’s (OMOP) Common Data Model which allows for the systematic analysis of disparate observational databases. You can read more about the OMOP CDM here: <https://www.ohdsi.org/data-standardization/the-common-data-model/> .

## The Environment

The Extract, Transform and Load (ETL) code is Spark-SQL code designed to be run on Databricks within the Chronic Conditions Warehouse VMware horizon client on TAF-RIF records. There is a lot of terminology to unpack to understand the environment this code is designed to operate within. Descriptively, the Centers for Medicare and Medicaid Services (CMS) supplies data to the Virtual Resource Data Center (VRDC). The Chronic Conditions Warehouse (CCW) curates VRDC data to third parties with CMS and VRDC approval. The CCW interface is a virtual desktop via a VMware horizon client. The client contains several programs that allow its users to interact with CMS data, in this case TAF-RIF records. Traditionally users of CCW use SAS Enterprise Guide to interact with VRDC data stored as SAS library 7bdat files. SAS Grid is used to implement SAS code, allowing large scale, multi-computer (cluster) resources to aid processing times. You can read more about SAS Grid here: <https://support.sas.com/rnd/scalability/grid/>.

CCW has transitioned to the cloud, allowing for cloud computing services to support CCW users implantation of VRDC data use cases. Cloud options include SAS’s cloud client and Databricks; though enterprise guide and SAS Grid are still available. Databricks represents the most distinct departure from SAS Enterprise Guide in the CCW environment. While SAS conceals a lot of read and write processes from the user and has dedicated file formats Databricks does not. The CCW Databricks implementation uses Amazon Web Services (AWS) Hive managed tables for storage, and Microsoft Azure Cloud for processing (analysis) though advanced users could alter this. Note Hive uses distributed storage of tables, so tables will be stored in small pieces, rather than one object. You can read more about Hive here: <https://aws.amazon.com/big-data/what-is-hive/>.

Towards analysis, because Databricks uses Spark, some familiarity with how Spark computes would be beneficial. Spark uses distributed processing within clusters and divides analysis among workers, to whom cluster hardware resources are allocated dynamically and intentionally. How workers understand your instructions is of particular importance. Spark graphs the user’s software language of choice to its implementation language, and then implements what it thinks you asked for. This translation moment is key, as Spark may not maintain a 1:1 translation and somethings may get lost. There is a great deal of material about how to ‘tune’ your Spark cluster and code to make sure that the analysis is distributed among workers fairly and understood by workers thoroughly. Code that translates as a labor burden to one worker rather than all workers means Spark has resources but is not using them (idle worker) efficiently. If Spark misunderstands your intentions, you will have curious output for your request. You can read more about Databricks and tuning a cluster here: <https://www.databricks.com/glossary/spark-tuning>. In the later sections of the document, there is in depth help for using Databricks on CCW.

## Who this guide is written for

This guide is written for the individual who will be implementing the deployment of the ETL code. We assume this individual is familiar with the complexities and specificities of using the CCW. We do not assume a high level of familiarity with Databricks or Spark-SQL, the ETL implementation language. Spark-SQL was chosen for this ETL due to the predominant familiarity of SQL among OMOP users. You can read more about Spark-SQL here: <https://spark.apache.org/sql/>.

## What you should get out of this guide

The ETL code will generate an OMOP CDM to support your research efforts. This guide will support you in deploying the ETL code developed under this award. The guide will also help you troubleshoot your build, give you an understanding of when to start over and when to continue building.

## Why we chose Databricks

This (see [Databricks section](#_Using_Databricks_in) for more information) ETL was developed for Databricks. Databricks is Python and R friendly and can also carryout operations which SAS may find difficult. Many studies are conducted in R, and perhaps less so in Python. The ability to carry out an analysis outside of SAS means more analysts and research studies could find value in Medicaid records. The workforce is pivoting towards R, and analysts may be more comfortable or productive using open-source languages. Further there are cloud-advantages and cloud-disadvantages to using a cloud-based system; Databricks presents several advantages over using CCW-SAS which may be of interest to specific studies. The ability to use machine learning methods in Databricks may be its highest value proposition, followed by workforce accessibility.

## Why we chose OMOP CDM

The ETL uses the OMOP common data model. OMOP is commonly used in observational research, which TAF-RIF records are apt to support given their retrospective and non-interventional nature. OMOP is a popular choice among researchers and it is regularly used in multi-center studies.

# Our development environment

## Data Use Agreement and User IDs

To use the ETL users must update it to reflect the Read-Write Permissions Bucket and the Read Permissions Bucket **before** uploading into Databricks. Users might consider using ‘edit find replace’ within the notebook GUI but with 21 notebooks that could take a while. Be careful you do not damage the underlying data in the read only bucket. CCW will have to re-image it for you if you do. The Data Use Agreement (DUA) is encoded in the bucket name as is the User ID. **These function like SAS libraries and must be declared correctly, or Spark will not find the table you reference.** If you do not know your DUA or User ID, ask CCW:

ccwhelp@ccwdata.org

Example

Read and Write Permissions Bucket format: DUA\_<DUA-Number> \_<User\_ID>

Read Permissions Bucket format: DUA\_<DUA\_Number>

User ID File Path: User/<User\_ID>

## How to run the code

There are four methods for doing this, each with its own advantages and disadvantages.

### Notebook cell by notebook cell

This method would be ‘interactive’ and time intensive, but it is the best way to understand the ETL; especially if changes were desired. Make sure to run notebooks in their ‘super order’ of 00 through B denominated notebooks. Do note that there are several instances of porting terms through the ETL. This is unimportant with job-ID which merely populates the ETL log. Year is perhaps the most important of piped terms.

### Notebook by notebook

This is recommended; as the notebooks are designed to be interactive if they encounter a memory fault or to run entirely through without issue. There are container notebooks in the ETL. For example transform\_load\_YYYY would run all transformation and load commands which are Year ‘YYYY’ (perhaps 2014) specific. This may be helpful in both updating the ETL as well as managing to replace a single table.

### Launch

A chained or launch.py notebook is supplied. It can be run from start to finish but this would require avoidance of any out of memory issues in processing. Clusters are configured to be ‘cheap’ rather than ‘greedy’ so if any other user out bids you for resources (RAM, CPU) the workers will fall below the minimum and the launch will crash (spot termination). As runs which fail part way through happen, you will need to pivot to notebook level execution to overwrite the last half-populated table. Another option is to use the job method below. Launch is the second-best option. But launch only works if the cluster is configured to restrict Scala output and avoid spot termination errors. CCW must do this for you with the following command:

`Spark.databricks.driver.disableScalaOutput`.

This is because the volume of the ‘successful run’ statements the log creates exceeds the 20 MB limit of the log. For example, ‘Error: RPC Response Too Large RPC response of :20,987,896 bytes exceeds limit of 20,971,520 bytes.’

### JOB utility

Spark has a job utility which allows users to submit notebooks in order as a scheduled job. The ETL is designed to be run once, not repeatedly. Further the local ETL log will confound some of the Job level log features and may produce erroneous log entries(which is perhaps minor). Configuring the job service is not difficult but users must point the job to a preexisting cluster as CCW users cannot originate or alter their clusters. Users should configure a job to run the notebooks in the order they appear in Launch. Note Support does not appear in launch as it requires a lot of hand holding to port data through the datastore; it should be run cell by cell, once, before a Launch is attempted.

## Starting the cluster

It can take up to half an hour to find, collect and launch a cluster with the requested resources. From time to time, the type of cluster CCW assigns you may be retired from the specific AWS server farm it is being called from. When this happens CCW will need to reassign a cluster type for you; request this via an email to ccwhelp@ccwdata.org. Note that Autoscaling should be enabled as well; this is essential. Coordinate with ccwhelp to make the cluster configuration match ours:

Policy: Unrestricted

Cluster Mode : High Concurrency

DRV 9.1 LTS ML

Enable autoscaling: yes

Max workers: 4

CPU: 20

Ram: 160G

Restricted Scala Output

Within the notebooks be sure to enable caching and partition shuffle above 6000; this is declared in the ETL as necessary. If the cluster restarts or if workers are dropped and added again this may need to be respecified from time to time. To avoid ‘spot termination’, consider asking CCW to change cluster assignments. It will take several tries to see an improvement but the issue is resolvable.

## How is the code structured?

Towards the remaining ETL code, the files are named intentionally where when sorted alphabetically the first item is a number, followed by A and B. The 00 file is Support and the 01 files are Data Definition Language and Extraction notebooks. 02 optimizes the extraction tables. A class notebooks are ‘run one time’ Transform and Load programs while B programs are annualized Transform and Load programs. The remaining components of the file names either describe the specific year for kinds of code or the destination table the notebooks populate. For example, A6 consumes data from the demog\_elig\_base TAF-RIF table learned during extraction file 01\_DDL\_E\_0000 and populates the CDM death table. B files must be run once per 01\_DDL\_E\_YYYY file. The Transform\_Load\_YY Python programs run the necessary B level files once per data year. There is a ‘clean up’ notebooks which are not necessary but genuinely improve the quality of the ETL product.

### 01\_DDL\_CDM

This file creates the CDM destination tables. Note technically the Athena tables are CDM tables but they are defined and populated under 00\_Support. The ‘support’ program is described below.

### 01\_DDL\_E\_0000

This file creates DDL and extracts demog\_elig\_base and demog\_elig\_dates.

### 01\_DDL\_E\_YYYY

Here data is extracted to populate the B CDM tables. These files are clones of each other with the ‘year’ updated. To add another data year (such as 2021) simply clone this file as well as a Transform\_Load\_YY Python file and update the year.

### 02\_optimize

This file is called at the beginning of the Transform\_Load\_YY files to optimize the source (extracted) tables for use. If you want to change the optimization method do so within this notebook.

### A Class

The A class notebooks are run once and only once. They consume extraction files from non-annualized extraction notebooks such as 01\_DDL\_E\_0000 and 00\_Support. They do not require monthly records.

### B Class

The B class notebooks are run once per data year. They are the Transformation and Load programs which populate the destination DDL CDM tables made in 01\_DDL\_CDM. Launch notebook will port a data year to class B notebooks which are called as desired, one year per cell.

### Table 1. Notebooks and their outcomes

|  |  |  |
| --- | --- | --- |
| Run Order | Notebook Name | Description |
| 1 | 00\_Support | Creates supporting tables from support data. |
| 2 | 01\_DDL\_CDM.sql | Creates empty CDM tables. |
| 3 | 01\_DDL\_E\_2014.sql | Extracts 2014 denominated records from TAF-RIF. |
| 4 | 01\_DDL\_E\_2015.sql | Extracts 2015 denominated records from TAF-RIF. |
| 5 | 01\_DDL\_E\_2016.sql | Extracts 2016 denominated records from TAF-RIF. |
| 6 | 01\_DDL\_E\_2017.sql | Extracts 2017 denominated records from TAF-RIF. |
| 7 | 01\_DDL\_E\_2018.sql | Extracts 2018 denominated records from TAF-RIF. |
| 8 | 02\_Demo\_all\_years.sql | Extracts demo and enrollment date records from TAF-RIF |
| 9 | 02\_optimize.sql | Optimizes read times from extracted TAF-RIF data. |
| 10 | A1\_location.sql | Creates CDM Location table from support data. |
| 11 | A2\_care\_site.sql | Creates CDM Care Site table support data. |
| 12 | A4\_Person.sql | Creates CDM Person table from TAF-RIF DEMO |
| 12 | A5\_observation\_period.sql | Creates CDM Observation Period table from TAF-RIF ELIG |
| 14 | A6\_death.sql | Creates CDM Death table from TAF-RIF DEMO |
| 15 | B0\_provider.sql | Creates CDM Provider table from support and TAF-RIF. |
| 16 | B1\_visit\_occurrence.sql | Creates CDM Visit Occurrence table from TAF-RIF. |
| 17 | B2\_condition\_occurrence.sql | Creates CDM Condition Occurrence table from TAF-RIF. |
| 18 | B3\_procedure\_occurrence.sql | Creates CDM Procedure Occurrence table from TAF-RIF. |
| 19 | B4\_drug\_exposure.sql | Creates CDM Drug Exposure table from TAF-RIF. |
| 20 | B5\_observation.sql | Creates CDM Observation table from TAF-RIF. |
| 21 | B6\_measurement.sql | Creates CDM Measurement table from TAF-RIF. |
| 22 | 09\_provider\_clean\_up | Makes Provider, Care Site and Location tables distinct. |

**Please note that the run order is not optional and prior notebooks in the run order offer required terms and tables which post run order notebooks require.**

## Prerequisites

### User ID and DUA

Several steps must be taken before the code is run. After the User ID and DUA is replaced, and the cluster correctly configured be sure to set up third party tables before beginning the implementation. The implementation starts with 00\_Support, which requires several tables be made in third party applications. The tables are described below.

### 00\_Support [Third Party Data]

Setup requires three external data sets (some are multiple tables) to be loaded into CCW and then into Databricks and then into Spark. Once complete, Spark can write them to AWS and the ETL can read them. This process is confusing to many users as once the data is ‘loaded’ it must be loaded twice more. First users must use the CCW upload feature to import the three extra tables, described below, into the CCW environment. Users can see the CCW upload instructions once you log in to the CCW website. Users can access the file transfer utility the same way files are downloaded from CCW; just use the upload button rather than download. The next step is finding the specified data within the uploads folder in the VRDC environment. Next open Databricks and import the files into the ‘file store’.

For more on the Databricks file store see: https://docs.databricks.com/data/filestore.html.

Once the tables are in the file store Spark cannot see it, AWS cannot see it, Hive cannot see it and users cannot interact with it. Users must import the data set into a Spark like object and then save that Spark like object to AWS-Hive as a formal table. Until this final step is done you cannot declare operations on the tables you have imported. Remember this is a frustrating process for first time users and it has three import steps that must be completed sequentially. Note the Setup notebook assumes stable table names and file locations. It will handle the last two steps (writing from the file store to Spark and then to the AWS bucket) but you must update the user directory IDs.

The tables you need can be found here:

### Athena

1. Download Athena from: [https://Athena.ohdsi.org/](https://athena.ohdsi.org/) Note the ‘DOWNLOAD’ button on the upper right-hand side of the website. Users do not need all Athena vocabularies for this project; following vocabularies are required at a minimum: APC, ICD9CM, HCPCS, ICD10CM, CPT4, ICD10, Visit Type, ICD10PCS, Observation Type, CMS Place of Service, Medicare Specialty, DRG, NDC, ICD9Proc, RxNorm, and RxNorm Extension.

All Athena tables are required: metadata, CDM source, concept, vocabulary, domain, concept class, concept relationship, relationship, concept synonym, concept ancestor, source to concept map, drug strength, cohort definition, and attribute definition (14 tables).

2. Obtain a UMLS API key (https://www.nlm.nih.gov/research/umls/index.html)

3. Add CPT to Athena extract (https://www.youtube.com/watch?v=2WdwBASZYLk)

You are now ready to upload Athena tables to CCW, then to Databricks and on to AWS.

### National Plan and Provider Enumeration System (NPPES)

CMS curates provider detail as National Provider ID (NPI) tokenized records.

<https://download.cms.gov/nppes/NPI_Files.html>

Users must download and reduce the record width using a statistical programing language such that the NPI file for individuals and places is reduced to two gigabytes or less. The width reduction is accomplished by sub-setting the file which reduces it to variables which are used in the ETL and splitting the result into two files described below. Databricks file store will limit uploaded file size to 2G. The NPPES file is a data dump from a website with a rolling monthly update so the ‘freshness’ of the ETL CDM could require regular updates if desired.

The following NPPES tables and variables were used:

NPI\_DATA\_PFILE.

The file is too large for the datastore, so it was split into provider and place. The specifications for each file is below as npidata\_pfile.csv (provider) and npi\_place.csv (place). We use R for this; tidyverse\dplyr, Base or datatable would work as would Python pandas, SQL-lite or any other such solution.

1. npidata\_pfile.csv

Retained Variables: NPI, Entity Type Code, Replacement NPI Employer Identification Number (EIN), Provider Organization, Name (Legal Business Name), Provider Last Name (Legal Name), Provider First Name, Provider Middle Name, Provider Name Prefix Text, Provider Name Suffix Text, Provider Credential Text, Healthcare Provider Taxonomy Group\_1.

Note NPI is the primary key.

2. npi\_place.csv

Retained Variables: NPI, Entity Type Code, Provider First Line Business Mailing Address, Provider Second Line, Business Mailing Address, Provider Business Mailing Address City Name, Provider Business Mailing Address State Name, Provider Business Mailing Address Postal Code, Provider First Line Business Practice Location, Address Provider Second Line Business Practice Location, Address Provider Business Practice Location, Address City Name, Provider Business Practice Location Address State Name, Provider Business Practice Location Address Postal Code.

Note NPI is the primary key.

Reducing to the required variables is ideal as NPPES is too large to be uploaded into Databricks file store natively. Without these tables you will not be able to learn provider details.

You are now ready to upload NPPES subset tables to CCW, then to Databricks and on to AWS.

### PX Handshake

The PX (procedure codes) handshake finds procedures and DX (diagnostic codes) codes that are identical across vocabularies despite having different meanings and reconciles what they are by harmonizing vocabulary ID for CMS with Athena. This will improve the ground truth of the ETL. The file is a four-line, two column CSV printed below:

PX Handshake (as an implied four record CSV with header)

|  |  |
| --- | --- |
| vocabulary\_id\_a | vocabulary\_id |
| 01 | CPT4 |
| 02 | ICD9Proc |
| 06 | HCPCS |
| 07 | ICD10PCS |

You could hand enter this, cast it in SQL to a table or upload it if you like. Either way, you are now ready to upload PX\_ Handshake to CCW, then to Databricks and on to AWS.

# How do you start the deployment?

1. Acquire the code, which is available here:

<https://forums.ohdsi.org/t/medicaid-etl-taf/17347?u=nick_williams>

1. Replace DUA numbers <DUA>
2. Replace User IDs <user\_id>
3. Replace write bucket <write\_bucket>
4. Complete 00\_Setup prerequisites
5. Decide on your run method (See ‘how to run the code’)
6. Logic and Range Checks

Table 2. Logic and Range Check

|  |  |  |  |
| --- | --- | --- | --- |
|  | Primary Keys | Row Count | Distinct Event |
| Observation Period | 117,539,100 | 378,510,705 |  |
| Person | 119,048,562 | 119,048,562 |  |
| Death | 3,159,046 | 3,159,046 |  |
| Provider | 1,795,870 | 8,645,062 |  |
| Location | 18,394,222 | 18,394,222 |  |
| Care Site | 1,469,652 | 1,469,652 |  |
| Visit Occurrence | 17,241,120,140 | 17,241,120,140 |  |
| Condition Occurrence | 98,181,509 | 8,255,639,272 | 90,954 |
| Drug Exposure | 84,425,269 | 3,298,566,455 | 238,927 |
| Measurement | 19,573,638 | 50,883,856 | 209 |
| Observation | 85,185,932 | 1,465,552,586 | 6,706 |
| Procedure Occurrence | 101,029,214 | 11,736,185,952 | 145,085 |
| \*Provider, clean up | 1,795,870 | 3,413,731 |  |
| \*Care Site, clean up | 352,891 | 352,891 |  |
| \*Location, clean up | 403,226 | 403,226 |  |

# Working with the CDM

## Origin Tables (TAF-RIF)

TAF-RIF files are separated into Annual and Monthly files. Demographic and Eligibility Base File (Demog\_Elig\_Base) and Demographic and Eligibility Dates (Demog\_Elig\_Dates) files are annual files. Monthly files include Inpatient Header file (OT\_Header), Inpatient Line file (OT\_Line), Other Services Header file (OT\_Header), Other Services Line file (OT\_Line), Long Term Header file (LT\_Header), Long Term Line file (LT\_Line), Medication Header file (RX\_Header) and Medication Line File (RX\_Line).

## Table 1. Original Tables

|  |  |  |
| --- | --- | --- |
| Annual Files | Monthly Header Files | Monthly Line Files |
| Demog\_Elig\_Base | OT\_Header | OT\_Line |
| Demog\_Elig\_Dates | LT\_Header | LT\_Line |
|  | RX\_Header | RX\_Line |
|  | IP\_Header | IP\_Header |

Not all OMOP tables are supported by the ETL. Health economics tables in particular are not developed in this version of the ETL. Tables which are essential for our observational clinical research are supported.

## Table 2. Destination Tables (OMOP) Supported by the ETL

|  |  |  |  |
| --- | --- | --- | --- |
| Person | Provider | Condition | Observation |
| Enrollment | Visit Occurrence | Procedure | Measurement |
| Death | Location | Drug Exposure |  |

## 

## Unsupported or ‘Curious’ CDM features

### 

### Visit\_Occurrence\_ID

The visit occurrence ID is not a random number. Instead, it is a complex foreign key that expresses terms of uniqueness of the related header claim. You cannot use it to find or order clinical visits as this claims data model is not really visit based, but claims based. Visit occurrence should perhaps be renamed to claims occurrence table. Further the data you need to audit and reconcile anything observed in the CDM with TAF-SAS is contained within this ID. If you want additional information that is not included in the CDM but is in TAF-SAS consider using the Claim\_ID, month, year and file type (Visit\_Occurrence\_ID) to find what you are looking for.

### Finance

Finance was deemed out of scope for this project. If interested in making ‘money’ CDM tables do use the Visit\_Occurrence\_ID as is. The Claim\_ID can help the extraction of financial values at claim level while assigning them to the diagnostic, procedure, and medication values from the same source claims.

### Person\_ID

The CDM Person\_ID is the CCW Beneficiary ID; this is intentional as it supports reconciliation. Spark sometimes coerces numbers to scientific notation so you may notice Person\_ID has an E in it and Bene\_ID does not. This should not confound joins within Spark. For example, the Bene\_ID ‘1234567890’ can appear in the CDM as ‘1.234567890E9’. This is scientific notation and is easily reconciled in post processing; the SQL ‘Cast’ function can support coercion if desired. Any join of the ETL to SAS Bene\_IDs should first cast Bene\_ID to string via Spark-SQL if necessary. We did not evaluate how SAS handles scientific notation to integer joins.

### Dates

Almost all dates are ‘service through’ dates. There are exceptions such as date of admission or date of discharge. In this way dates are ‘fuzzy’ and are not a true ‘EHR’ event onset dates as OMOP requires. Further the ‘first’ date for a DX in the CDM is not the first time a patient presented with a DX, simply the first time Medicaid detected it within the observation window (2014-2018).

### Claim to event relationship

Users should reduce any study event set to a count of distinct events per distinct individual within a logical study range. For example, it is possible that a clinical event exists as 12 different claims or 12 claims describe 12 clinical events. Within the study definition a maximum range of events should be established, and the study event set should be reduced to such a framework. For example, if cases have one Acute Myocardial Infarction (AMI) claim on an inpatient file record or more than one AMI after a logical period of time; the Principal Investigator should create terms to evaluate what repeat claims for AMI means within a claims context. Does it mean the patient had a second AMI, or a complex billing history? On some level the CDM can not tell you what your study definitions of a case-event are.

### National Drug Codes (NDC)

Not all National Drug Codes (NDC) are in Athena, and in turn not all NDCs have Concept\_IDs. If you have negative controls, be sure to reconcile un-identified NDCs as they may bias the data reuse study. Further TAF-RIF contains device NDC and CDC vaccine NDCs which must be handled with care. Many states use zero injection without hyphenation; this can make reconciliation of NDCs difficult. Be sure to rule out alternate candidates within the study event index of observed NDC codes.

## How to query the CDM

There are several primary keys that can be used to understand complex information stored in the CDM and not all primary keys are present in all tables. Primary keys all have a ‘subject’ table which contains one primary key denominated record per row and other ‘event’ tables which contain many events attributed to the primary key.

To retrieve information about an individual or series of individuals use ‘Person\_ID’, which has subject details in the person table and event details in Condition, Procedure, Drug Exposure, Observation, Measurement, Death, Enrollment, and Visit Occurrence. To retrieve information about an individual clinical provider, use ‘Provider\_ID’ which has subject details in the provider table and event details in visit occurrence, condition, procedure, drug exposure as well as observation and measurement. Note the provider table does not store ‘providers’ in its subject table, but instead stores provider-specialty-location pairs, which need not be distinct. To retrieve information about a location, use ‘Location\_ID’ which has subject details in the location table and event details in Visit Occurrence, Condition, Procedure, Observation, and Measurement tables. Visit level data has its own primary key, ‘Visit\_ID’ which is described at subject level within Visit Occurrence table and at event level within Condition, Procedure, Drug Exposure, Observation and Measurement tables.

## Ideal Workflow

### 1. Establish exposure of interest

This could be an outcome or a medication as well as a clinical condition declared in the condition occurrence table. A complex cohort definition could be used though users should refrain from being too specific until they are familiar with the contents of TAF-RIF.

### 2. Reduce the CDM

The CDM is very large and when working with it, users should reduce to a table series where the tables only contain the population of interest (that had the condition). Reduction will significantly improve processing time, as it is unlikely that the entire TAF-RIF population shares an exposure of interest.

### 3. Evaluate the presence of a given feature within the population of interest.

This could be time to event, presence of event (mortality, live birth) or a discrimination analysis (therapeutic inferiority) of some kind.

### 4. Alter and refine the exposure of interest to better reflect the peculiarities of the CDM.

You may find the exposed population is too small, or that the outcome of interest is too rare to support the study. We really do not know in advance what studies the CDM can support.

### 5. Draw a defensible conclusion from the study data.

You might consider a ‘smell test’ where the prevalence or incidence from non-CDM data is used to qualify the recall of your case index volume. For example, if there are 1.2 million Americans with HIV alive in the USA in any given year, and you can only find 10 HIV Medicare cases, something is most likely wrong with your code. Further if you find all 1.2 million cases that would be strange as Medicaid is not a universal health care service.

You should also ask if the outcomes observed have a ‘theory in practice’. For example, in toxicology there are known chemical relationships between medications and human metabolisms. If performing a pharmacology study, you could ask if the observed outcome is known to toxicology/pharmacology/chemistry. If not, the conclusion may not be defensible.

Consider if the conclusion is known to clinical knowledge to help decide if your observations are defensible. Novel discoveries are possible from observational data, but they require very strong evidence, study design and reproduction with additional CDMs.

## Tutorial Notebook

### Real World Example

The ETL code repository contains a tutorial for conducting large aspects of an observational study using the CDM. The tutorial follows the traditional method of cohort definition to conclusion. The effort follows a fairly easy cohort model using quickly completable code. Starting from Athena tables, a medication and condition cohort are generated, and an intersection (co-membership) is considered. How many cases who were exposed to x had condition y can be followed by `time to event analysis` and `case control` efforts to qualify observed effects. The basic steps of querying the CDM to extract conditions and medications of interest, and creating intersections of these case-event tables is provided. More ‘complex’ analysis is possible with the CDM. The tutorial discovers kinds of patients who had gout, spontaneous abortion and took alopurinol. The tutorial also demonstrates what to do when you don’t have exposure records for your study idea (think about what exposures are available, especially metoclopramide). When you are ready for more complex analysis consider: <https://ohdsi.github.io/TheBookOfOhdsi/.>

## Update the CDM when new TAF-RIF years become available

Updating the CDM is a straightforward process. You need only create an extract notebook which mirrors the extraction notebook for prior TAF-RIF years. Search and replace for the next year will achieve this. Further you must create a ‘run’ notebook to add a year of records to the CDM from the extraction tables. The annualized nature of extraction and transform-load notebooks should make updating the CDM easy. Note this method will not update the locations and provider names found in the Support\_XX notebook. If desired also consider running A1\_location and A2\_care\_site along with the 09\_provider\_cleanup. This will result in a CDM which contains the new TAF-RIF years and updated provider details from CMS.

## Reconciliation with SAS TAF-RIF files

Note that because of the combinatoric Visit\_ID, it is possible to connect the TAF-RIF ETL CDM tables with their originals in Databricks as well as in TAF-SAS. To do such a thing you would need the Claim\_ID or and the Person\_ID. The file type is also helpful to limit the scope of the search, as is the claim through date. While disambiguation between line and header is not possible it should be easily discerned by file type and linked CDM table. For example, procedure occurrence ICD10PCS code would be Inpatient header while Healthcare Common Procedure Coding System (HCPCS) would be Line where the file type is inpatient concept code. TAF-RIF file type is encoded in Visit\_ID.

# Using Databricks in VRDC/CCW

## What is Databricks?

Databricks is not a traditional single application environment. Instead, it is a platform which allows for multiple environments and features to be accessed through a single interface. The interoperability, documentation, and utility of Databricks varies by the creativity of the user and the task at hand.

Databricks does not own its workflow but harmonizes several workflows across cloud services. Databricks uses Google, Microsoft Azure, and Amazon Web Services through a single interface. Most operations are ‘masked’ to the user and the user may not be aware which cloud service provider and which cloud service is being used at any given time. This unraveling is key to troubleshooting Databricks, which can be quite hard as the causal problem may be due to disparately documented cloud service features, `not integrated into`, but `accessed through` Databricks.

Databricks then is an interface between Spark, a multi-language, multi-API service for distributed data analysis and cloud services for storage, processing and custom features. CCW Databricks uses AWS for data storage. the CCW Databricks User Directory and Data Directory are AWS EC3 data storage buckets/objects. CCW users can only access their buckets through Databricks. If users need something placed in the bucket which Databricks cannot port due to its size, contact CCW by emailing CCW [ccwhelp@ccwdata.org](mailto:ccwhelp@ccwdata.org) and request they move the file. Storage is important in Databricks, and permissions, configuration and processing structure must be configured correctly. For example, AWS keeps its data Health Insurance Portability and Accountability Act (HIPAA) compliant / encrypted during processing, storage and transit, but the size of the processing space (managed by AWS) is relatively small for this ETL job. CCW had to expand the encryption processing space within the storage environment to allow for large amounts of data to be encrypted in processing. Users may encounter similar ‘out of memory’ errors which have less to do with the available memory on the cluster and the specific cloud level task the memory is working on.

CCW Databricks uses Hive managed tables. This means that each table is two classes of information, the actual data and an index entry which describes the data. This index (or meta data table) is used to deploy table operations quickly by saving instructions for how to process a table instead of altering it and then deploying edits when a result is called.

## How is it different from SAS?

### Cost

CCW will charge users ‘tokens’ for each Databricks job run. The cost is relative to the size and complexity of the job. There is no CCW cost for storage currently, unlike CCW SAS. Non-successful runs still incur charges. The ETL considered 2014-2018 data years, it may be easier to acquire the ETL files than re-create them.

### Complexity

Databricks is no more complex than SAS Grid or SAS HASH, but users can use SAS to great effect without having a deep understanding of these advanced features. This is not true of Databricks; users really need to have a deep grounding in one of the ancillary programing languages (Spark-SQL, R, Python) and the primary language (Spark). Further a knowledge of cloud services, and cloud troubleshooting is also helpful.

### How is my Databricks instance set up?

The CCW Databricks instance works like this:

Users are authenticated using Google Cloud Services

Users store data in Hive managed tables in Amazon Web Services Buckets

Users write code in Databricks notebooks and run it on Microsoft Azure servers.

The result is either written to the console or saved in AWS buckets.

The above list may be perfunctory but note that the documentation users need to troubleshoot each aspect of the service is dependent on what is running the service. Where the error is happening is a big clue towards which documentation to use.

### Databricks updates

Databricks, Spark and all API/cloud services underlying them are in a state of active development and update. This means that sometimes when CCW updates Databricks prior code will fail and require updates.

## Resources, Tips and Troubleshooting

### 1. SQL vs. Spark-SQL

Spark-SQL is very different from traditional SQL. There are several complex departures that are worth understanding in depth. There are several documentation options to better understand best practices in Spark-SQL. Take the time to review them.

### 2. Spark-R is still under development

Spark-R is very different from traditional R. Because it is still ‘officially’ under development be careful that your R operations are not too complex to exceed the current level of support.

### 3. Scala is currently not supported for security reasons

Scala is not allowed in CCW as of this writing. This may change in the future. If Scala is a ‘must have’ consider asking CCW for permissions.

### 4. Structured data storage

There is no ‘library’, database or structured storage. Any operations or code which depends on this will fail. Data management at the table level, and recursively altering a table based on a list is not recommended. Alter Table (SQL) proc datasets (SAS) and any other ‘meta’ like operation will fail.

### 5. Meta tables and proc datasets

There is little to no meta table work; access to the Hive meta table is restricted to prevent breaches and meta table disconnect errors at this time.

### 6. Storage is more complex than it seems

The storage object is highly intentional in Databricks. Not all functions and operations can run on every object-file. Not all languages can interact with each object-file type. Not all object-tables can be coerced into all object-tables either. Learning what languages can interact with which table-objects is key to success. Consider the following pages:

Spark

[https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-Spark-apis-rdds-dataframes-and-datasets.html](https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html)

R (this link also contains a high value notebook)

[https://databricks.com/blog/2016/12/28/10-things-i-wish-i-knew-before-using-apache-Sparkr.html](https://databricks.com/blog/2016/12/28/10-things-i-wish-i-knew-before-using-apache-sparkr.html)

SQL

<https://Spark.apache.org/docs/latest/sql-data-sources-parquet.html>

### 7. Spark can take a ‘long’ time to write

Spark uses write once, read repeatedly models for its underlying storage model (WORM (write once, read many)). This supports large scale, enterprise data operations but ad-hoc and analytical tasks.

### 8. Spark clusters must be configured intentionally on an iterative basis

Spark is meant to be configured intentionally by a high skills administrator who can enable cashing, partitioning of tables, and ‘shuffle partitions’ (for the ETL the right answer is above 6000). When developing applications consider starting with a sub-set of the data and tuning the Spark configurations to evaluate run times.

### 9. User AWS buckets

Users will have read only and read-write permissions to two AWS buckets. These are sometimes called user directory but in truth they are standalone AWS buckets.

Read and Write Permissions Bucket format: DUA\_<DUA number> \_<ccw id>

Read Permissions Bucket format: DUA\_<DUA number>

### 10. AWS Hive Garbage Collector

CCW Databricks uses Hive managed tables. Occasionally the index in Hive is out of sync with the tables themselves. This causes jobs to fail, tables operations to crash and the log to report that either the table does not exist (in the index) or cannot be created because it already exists in the index, but not the AWS bucket that stores the tables. Eventually the ‘garbage collector’ will purge tables that lack an index or index entries which lack a table. A simple solution is to add the number ‘1’ to the end of every string which calls the phantom table in the ETL; both the create table and from table statements. Note provider tables were named provider1 within the ETL for several weeks of development for this very reason.

### 11. Multiple table operations are different in Spark

Spark does not have an analogous SAS Macro facility to support complex, iterative or point in time operations. In turn Spark code may look redundant. Sister solutions (mapping, views) are not that well supported as views are materialized when called as tables and mapping is a very complex topic in Spark. If you are using an iterative approach, consider being intentional and prototyping your effort on a small subset. This may save time. Spark is more like traditional SAS workflows where users would normalize their data rather than working across tables.