***FINAL REPORT***

for

***DESIGN AND IMPLEMENTATION OF HEALTH INSURANCE TREND ANALYTICS WITH PL/SQL***

Category: Relational Application Development

***INFO 606***

***TERM: WINTER 2022***

ZACH CARLSON, KATY MATULAY, JACOB STANK, JACOB WILLIAMSON

Table of Contents

[I. Overview 3](#_Toc98441174)

[II. Problem Statement 3](#_Toc98441175)

[**a.** **Problem Description: context, goal, scope** 3](#_Toc98441176)

[**b.** **Assumptions** 3](#_Toc98441177)

[**c.** **Scope** 3](#_Toc98441178)

[**d.** **Functional Requirements** 4](#_Toc98441179)

[**e.** **Data Requirements** 5](#_Toc98441180)

[**f.** **Hardware/Software Requirements** 5](#_Toc98441181)

[**g.** **Methodology** 5](#_Toc98441182)

[III. Conceptual Design 6](#_Toc98441183)

[**a.** **ERD** 6](#_Toc98441184)

[**b.** **ERD Explanation** 6](#_Toc98441185)

[IV. Logical Design 7](#_Toc98441186)

[**a.** **Relational Schema** 7](#_Toc98441187)

[**b.** **Data dictionary** 7](#_Toc98441188)

[V. Implementation 7](#_Toc98441189)

[**a.** **MySQL:** 7](#_Toc98441190)

[**b.** **Oracle:** 8](#_Toc98441191)

[VI. PL/SQL Implementation and Testing: 8](#_Toc98441192)

[a. Zach Carlson PL/SQL units: 8](#_Toc98441193)

[b. Katy Matulay PL/SQL units: 10](#_Toc98441194)

[c. Jacob Stank PL/SQL units: 22](#_Toc98441195)

[d. Jacob Williamson PL/SQL units: 31](#_Toc98441196)

[VII. Visualizations and Analysis 34](#_Toc98441197)

[VIII. Conclusion 37](#_Toc98441198)

[IX. References 38](#_Toc98441199)

[X. Appendix 38](#_Toc98441200)

[**a.** **Data Dictionary (Oracle)** 38](#_Toc98441201)

# **Overview**

The project goals were to develop a health insurance database and a) connect it to a Jupyter Notebook using python to conduct data analytics and visualizations, and b) implement PL/SQL to accomplish business goals related to clinical, transactional, and reporting use cases. We accomplished these goals by developing both MySQL and Oracle databases. We implemented PL/SQL in the Oracle database and connected the Heroku/ClearDB MySQL database to python using SQLAlchemy in a Jupyter Notebook to conduct time series analysis.

# **Problem Statement**

1. **Problem Description: context, goal, scope**

Health Insurance Companies require the ability to analyze claim costs and utilization patterns over time. This is required to inform pricing decisions, create financial forecasts, inform product design, and enact cost saving programs. Within this database, health plan operational data will be enhanced to allow for more refined analysis to better target the areas driving per member per month claim costs. The adjustments to the data will enable more targeted actions to be taken and ensure effective use of staffing resources in producing solutions to drivers of trend and implementation of more sophisticated predictive analytics and business intelligence applications.

At a high level the enhancement to this data will be:

* Bucketing of clinical data elements to standardize reporting categories.
* Logic to assign a unique visit a member had to a provider within the defined reporting category.
* Integration of chronic conditions of members to provide a clinical view of trend.

1. **Assumptions**

* Users will run the analytical Python code within a Jupyter Notebook environment (e.g., Google Colab, Anaconda/Jupyter Lab).
* Users will have installed all required Python packages before running Notebook.
* Users will have created their own local MySQL database (or Heroku/ClearDB MySQL database) and entered their own connection string URL into the database\_info.py file (Explained in further detail in [Github](https://github.com/zachcarlson/InsuranceDatabase/blob/main/insurance_database.ipynb)).
* The company has an existing backup solution.

1. **Scope**

**IN-SCOPE:**

The scope of the project will be limited to trend analytics centering on members, providers, and claims using Python and PL/SQL.

**Record Keeping**

1. Manage (insert/delete/edit) the member and member data
2. Manage the provider and provider data
3. Manage member policy data
4. Manage member claim data
5. Manage the clinical conditions of members
6. Manage claim categories and utilization
7. Manage date mapping into trend periods

**Reporting**

1. Create rolling average time series visualizations that highlight any seasonality or long-term trends.
2. Visualize demographics of members.
3. Identify members who are high dollar claimants and assess whether these outliers have an impact on overall trends.
4. Separately group data by Claims, Member Condition, and Member Location
   1. Analyze grouped data by utilization and cost over time.
5. Identify high-risk members and aggregate preventative care claims to assess risk score.

**Search Functionality**

Search and retrieve Member records using the fully qualified criteria:

1. Member ID
2. Member last name
3. Member Condition
4. Member Location

**Processing Functionality**

1. Application of business rules to data for reporting and analytics
2. Ability to ingest new incremental data monthly
3. Audit tracking of which user/process inserted, edited, or made changes
4. Clinical view of trend
5. Claims transaction view of trend
6. Aggregation of data into easy-to-use summary views/tables to lessen the need for joins and simplify the experience for end-users and BI developers.

**OUT-SCOPE:**

This project will not include payment processing activities and business management activities such as: payroll, benefits, premium, pharmacy rebates, pharmacy claims, and office supply purchasing. All data will be de-identified and contain only the minimum necessary information needed to perform trend analytics defined in the reporting requirements.

1. **Functional Requirements**

The database developed needs to allow for simple joins for the analysts to perform the task which they are assigned to resolve. Additionally, the data needs to contain information that will enable clinical interventions or outreach to a member. Since the use cases for this database are to complete trend analysis and projections the database needs to be able to store records that allow for a longitudinal view of a member and their associated claims and conditions.

1. **Data Requirements**

For this project, a sample of claims, eligibility, provider, and condition data were pulled from Highmark Health operational systems. The data was approved for this use case after all steps were taken to deidentify the data so that it could not be linked back to any client, group, member, or provider. As such, all values associated with member and provider names, address, or day of birth have been tokenized. However, the claims, provider, eligibility, and condition data are unchanged to allow for meaningful analysis to take place. The data is a snapshot of information that spans from Jan 1, 2015, through December 31, 2021 to allow for trend analysis and forecasting opportunities.

1. **Hardware/Software Requirements**

The hardware/software solution will work in both Windows 10 and mac OS if Google Colab is used to access the Jupyter Notebook and the database is either created within MySQL ClearDB/Heroku (or any other valid MySQL database set up) or Oracle. It is up to the user’s discretion whether to use a local MySQL server or create a Heroku account to connect the database to the Jupyter notebook, but the notebook is set up currently to work with a Heroku connection string URL.

The PL/SQL is designed to run only in Oracle but could be adapted by users to run in a local MySQL environment. The functions, triggers, and procedures have been tested and work in Drexel Oracle Server, SQL Workbench, and SQLDeveloper with Oracle 19c. ClearDB did not allow for the creation of PL/SQL procedures, functions, or triggers due to the shared cluster privileges, so we created two separate databases to implement PL/SQL in Oracle. A dedicated cluster would need to be purchased to enable this feature with Heroku, which was not feasible for this project. Additional plugin, driver, package, and GUI installation as specified in the Jupyter notebook is assumed to be handled by the user dependent on their machine and OS.

1. **Methodology**

The database starts with inserts of data which include some of the snowflake dimensions, and then by using procedures and triggers, builds additional summary tables to lessen the need for users to join multiple tables as well as builds tables specifically designed for BI tools to leverage. More information on database setup and connecting SQL to Python can be found on our Github in the **Database Setup** section [here](https://github.com/zachcarlson/InsuranceDatabase).

# **Conceptual Design**

* + - * 1. **ERD**

Diagram, engineering drawing

Description automatically generated

* 1. **ERD Explanation**

The ERD diagram above is driven by our fact table which stores our claims detail information for our enrolled members. The dimensions join to this table in either a 1:1 or 1:N relationship, depending on the relationship. For instance, the Member table joins to Claim Fact in a 1:N relationship, as members can have 1 or more claims, but each claim can only have 1 member. The database contains two dimensions for our membership and membership conditions as those change over time. While this is represented as a STAR schema with some snowflake dimensions it does not follow a true STAR schema as natural keys are left in place of surrogate key values in our fact table. This is done because some analysis can take place directly leveraging the claim fact table and we did not want to force our users to join to a dimension if they did not require any of the additional detail in the dimension to complete their analysis.

# **Logical Design**

1. **Relational Schema**

A screenshot of a computer

Description automatically generated with low confidence

1. **Data dictionary**

The Data Dictionary for the Oracle database can be found in Appendix 1 and as the attachment DataDictionary.xlsx. Note that there is also a tab for the MySQL database data dictionary in the excel file.

# **Implementation**

All Python code, SQL DML/DDL files, and PL/SQL files can be found on our Github [here](https://github.com/zachcarlson/InsuranceDatabase). Read the **File Manifest** section to get more information on each file. We’ll also briefly describe them below, however due to the size of the files we will only be pasting the PL/SQL contents as files range from 100-100,000 lines.

The files are formatted such that the first word outlines the database used (e.g., mysql\_), followed by the step number, and a brief explanation on its contents. The initial\_ddl files create tables, the insert\_data\_claim\_fact files populate only the claim facts table, and the insert\_data\_all\_others files populate all remaining tables.

1. **MySQL:**

* The following SQL script files were used to create and populate the database in MySQL Workbench:

1. mysql\_step01\_initial\_ddl.sql
2. mysql\_step02\_insert\_data\_claim\_fact.sql
3. mysql\_step03\_insert\_data\_all\_others.sql
4. **Oracle:**

* The following SQL scripts were used to create and populate the database in a local Oracle environment:

1. oracle\_step01\_initial\_ddl.sql
2. oracle\_step02\_insert\_data\_claim\_fact.sql
3. oracle\_step03\_insert\_data\_all\_others.sql
4. oracle\_step04\_plsql.sql

The oracle\_step04\_plsql.sql file compiles all PL/SQL functions, procedures, triggers, and sequences. While all files can be run in their entirety, we recommend going section by section for the PL/SQL file, as running several execute statements can throw errors. Additionally, some views are created throughout the PL/SQL to demonstrate functions and thus will need to be run segment by segment.

In the next section we’ll show each team member’s individual PL/SQL units.

# **PL/SQL Implementation and Testing:**

Team member’s PL/SQL text files were compiled into the oracle\_step04\_plsql.sql file. Running these raw text files in Oracle will throw errors.

* 1. **Zach Carlson PL/SQL units:**

**Function.** Return the name of the provider based on the provider ID. Considers whether the provider is an organization (name saved in column ORGNAME) or whether the provider is a doctor (name saved in column FIRST NAME and LAST NAME). Used in reporting purposes later.

**CREATE** **OR** **REPLACE** **FUNCTION** GET\_PROVIDER\_NAME **(**provider\_id **IN** number**)**

**RETURN** varchar2 **IS**

v\_provider\_type dim\_provider**.**providertype**%type;**

v\_provider\_name varchar2**(**100**);**

**BEGIN**

/\*Get provider type, specifies whether doctor or organization\*/

**SELECT** PROVIDERTYPE

**INTO** v\_provider\_type

**FROM** DIM\_PROVIDER

**WHERE** PROVIDERID **=** provider\_id**;**

/\*if doctor\*/

**IF** v\_provider\_type **=** 1 **THEN**

**SELECT** ORGNAME **INTO** v\_provider\_name **FROM** DIM\_PROVIDER **WHERE** PROVIDERID **=** provider\_id**;**

**END** **IF;**

**IF** v\_provider\_type **=** 2 **THEN**

**SELECT** FIRSTNAME **||** ' ' **||** LASTNAME **AS** FULL\_NAME **INTO** v\_provider\_name **FROM** DIM\_PROVIDER **WHERE** PROVIDERID **=** provider\_id**;**

**END** **IF;**

**RETURN** **(**v\_provider\_name**);**

**END** GET\_PROVIDER\_NAME**;**

**Function.** Returns the specialty of the provider based on provider ID. Used for reporting purposes later and potential analysis.

**CREATE** **OR** **REPLACE** **FUNCTION** GET\_PROVIDER\_SPECIALTY **(**provider\_id **IN** number**)**

**RETURN** varchar2 **IS**

v\_provider\_specialtyid number**;**

v\_provider\_specialty varchar2**(**100**);**

**BEGIN**

/\*Get provider type, specifies whether doctor or organization\*/

**SELECT** SPECIALTYID

**INTO** v\_provider\_specialtyid

**FROM** DIM\_PROVIDER

**WHERE** PROVIDERID **=** provider\_id**;**

/\*Get specialty name\*/

**SELECT** SPECIALTY\_DESC

**INTO** v\_provider\_specialty

**FROM** DIM\_PROVIDER\_SPECIALTY

**WHERE** SPECIALTYID **=** v\_provider\_specialtyid**;**

**RETURN** **(**v\_provider\_specialty**);**

**END** GET\_PROVIDER\_SPECIALTY**;**

**Procedure.** Procedure used to populate a table of all providers including their name, specialty, total number of claims and the total sum of allowed charges.

--Step 1: Create table to populate

**DROP** **TABLE** PROVIDER\_CLAIM\_REPORT**;**

**CREATE** **TABLE** PROVIDER\_CLAIM\_REPORT

**(**

PROVIDERID INT **NOT** **NULL,**

PROVIDERNAME VARCHAR2**(**100**),**

PROVIDERSPECIALTY VARCHAR2**(**100**),**

NUM\_CLAIMS NUMBER**(**14**,**2**),**

SUM\_ALLOWEDCHARGES NUMBER**(**14**,**2**)**

**);**

--Step 2: Create procedure

**CREATE** **OR** **REPLACE** **PROCEDURE** populate\_provider\_claim\_report **AS**

**BEGIN**

**INSERT** **INTO** provider\_claim\_report **(**PROVIDERID**,** PROVIDERNAME**,** PROVIDERSPECIALTY**,** NUM\_CLAIMS**,** SUM\_ALLOWEDCHARGES**)**

**SELECT** PROVIDERID**,** GET\_PROVIDER\_NAME**(**PROVIDERID**)** **AS** PROVIDER\_NAME**,** GET\_PROVIDER\_SPECIALTY**(**PROVIDERID**)** **AS** PROVIDER\_SPECIALTY**,** **COUNT(DISTINCT** CLAIMID**)** **AS** NUM\_CLAIMS**,** **SUM(**ALLOWEDCHARGES**)** **AS** SUM\_ALLOWEDCHARGES

**FROM** CLAIM\_FACT

**GROUP** **BY** PROVIDERID**;**

**COMMIT;**

**END** populate\_provider\_claim\_report**;**

**execute** populate\_provider\_claim\_report**;**

**Trigger.** Trigger to update provider\_claim\_report whenever a claim is inserted, deleted, or updated in claim\_fact

**CREATE** **OR** **REPLACE** **TRIGGER** provider\_claim\_report\_AIUDS

**AFTER** **INSERT** **OR** **DELETE** **OR** **UPDATE** **ON** claim\_fact

**DECLARE**

CURSOR c\_stat **IS**

**SELECT** PROVIDERID**,** GET\_PROVIDER\_NAME**(**PROVIDERID**)** **AS** PROVIDER\_NAME**,** GET\_PROVIDER\_SPECIALTY**(**PROVIDERID**),** **COUNT(DISTINCT** CLAIMID**)** **AS** NUM\_CLAIMS**,** **SUM(**ALLOWEDCHARGES**)** **AS** SUM\_ALLOWEDCHARGES

**FROM** CLAIM\_FACT

**GROUP** **BY** PROVIDERID**;**

**BEGIN**

**FOR** v\_stat **in** c\_stat **LOOP**

**UPDATE** provider\_claim\_report **SET** NUM\_CLAIMS **=** v\_stat**.**NUM\_CLAIMS

**WHERE** PROVIDERID **=** v\_stat**.**PROVIDERID**;**

**UPDATE** provider\_claim\_report **SET** SUM\_ALLOWEDCHARGES **=** v\_stat**.**SUM\_ALLOWEDCHARGES

**WHERE** PROVIDERID **=** v\_stat**.**PROVIDERID**;**

**END** **LOOP;**

**END** provider\_claim\_report\_AIUDS**;**

* 1. **Katy Matulay PL/SQL units:**

The PLSQL in this section is geared towards analytic applications for high-risk patients and thus focuses on calculating instances of various chronic conditions, preventative care services, and then presents this as a historical snapshot view for managerial use. There are 9 functions, 1 procedure, and 1 trigger.

1. **Function CALC\_Age**

/\* Katy Matulay

Function returns age to be used in age demographic analysis \*/

CREATE OR REPLACE FUNCTION CALC\_AGE(p\_dob DATE)

RETURN NUMBER IS

v\_Age NUMBER(3);

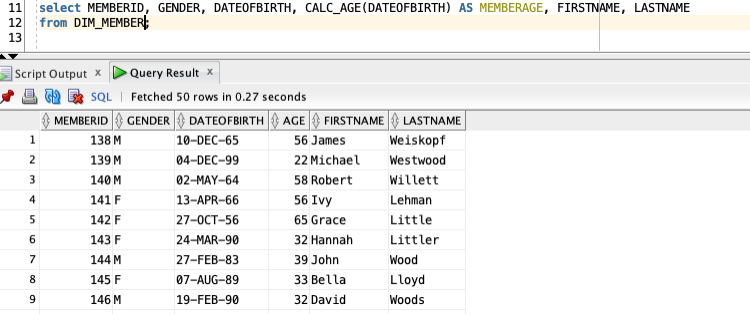
BEGIN

v\_Age := round((sysdate-p\_dob)/365);

RETURN (v\_AGE);

END CALC\_AGE;

/



1. **Function CALC\_Comorbid**

/\*Katy Matulay

CALC\_Comorbid Function calculates a cumulative comorbidity score for members, to be used in the high\_risk\_patient views\*/

CREATE OR REPLACE FUNCTION CALC\_Comorbid (p\_memberID IN INT)

RETURN INT IS

v\_ComorbidityScore INT(2);

BEGIN

SELECT SUM(MAX(DIABETES) + MAX(COPD) + MAX(CAD) + MAX(CHF) + MAX(HYPERTENSION))

INTO v\_ComorbidityScore

FROM DIM\_MEMBER\_CONDITION

WHERE MEMBERID = p\_memberID

GROUP BY MEMBERID;

RETURN(v\_ComorbidityScore);

END CALC\_Comorbid;

/

\*Note- This function provides a cumulative max count of comorbidities, however if a user wanted to isolate for a specific enrollment period it would need to be adjusted on an individual basis using “startdate” and “enddate” in the dim\_member\_condition table. Because enrollment periods are user-specific, this is beyond the scope of this project as it requires more sophisticated data domain knowledge. However, the next function aims to isolate for a singular most recent enrollment period, per individual member to get current enrollment calculations.

1. **Function CALC\_Comorbid\_Current**

/\*Katy Matulay

The following function calculates a comorbidity score for members in their most recent enrollment period (based on max enddate), to be used in the high\_risk\_patient views and HR\_Mem\_Outreach table/procedure \*/

CREATE OR REPLACE FUNCTION CALC\_Comorbid\_Current (p\_memberID IN INT)

RETURN INT IS

v\_ComorbidityScore INT(3);

BEGIN

SELECT SUM(DIABETES + COPD + CAD + CHF + HYPERTENSION)

INTO v\_ComorbidityScore

FROM DIM\_MEMBER\_CONDITION

WHERE MEMBERID = p\_memberID

AND enddate =

(select max(enddate)

from dim\_member\_condition

where memberID = p\_memberID);

RETURN(v\_ComorbidityScore);

END CALC\_Comorbid\_Current;

/

1. **Function GET\_VaxCovid**

/\*Katy Matulay

GET\_VACXCOVID function determines if members have received a covid vaccination

and represents it as a numeric count (i.e 1st dose, 2nd, 3rd, etc.),

to be used in the high\_risk\_patient views

The covid vaccine code values were determined using

https://www.cms.gov/medicare/medicare-part-b-drug-average-sales-price/covid-19-vaccines-and-monoclonal-antibodies \*/

CREATE OR REPLACE FUNCTION get\_VaxCovid (p\_memberID IN INT)

RETURN INT IS

v\_covidVax INT(2);

BEGIN

SELECT count(code\_type)

INTO v\_covidVax

FROM CLAIM\_FACT

WHERE code\_value IN ('0001A','0002A','0003A','0004A','0011A','0012A','0013A','0064A','0071A', '0072A','91300','91301','91302','91303','91304','91305','91306','0021A',

'0022A','0031A','0034A','0051A','0052A','0053A','0054A')

AND memberID = p\_memberID

AND CLAIMSTATUSCD like 'A';

RETURN v\_covidVax;

END get\_VaxCovid;

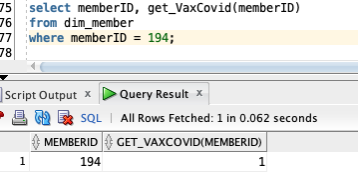
/

--Demonstrate Function

select memberID, get\_VaxCovid(memberID)

from dim\_member

where memberID = 194;

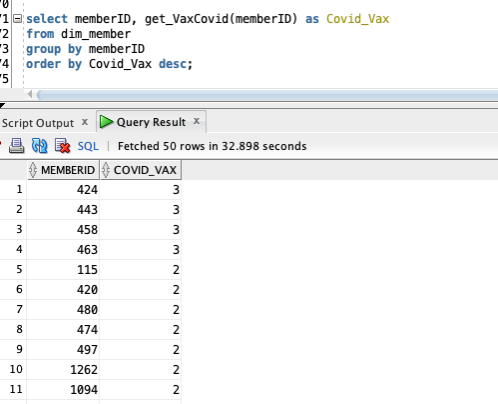


select memberID, get\_VaxCovid(memberID) as COVID\_VAX

from dim\_member

group by memberID

order by COVID\_VAX desc;



1. **Function GET\_VaxCovid\_Current**

/\*Katy Matulay

GET\_VAXCOVID\_CURRENT function determines if members have received a covid vaccination in the most recent enrollment period (based on max enddate) and represents it as a numeric count (i.e 1st dose, 2nd, 3rd, etc.),

to be used in the high\_risk\_patient views and HR\_Mem\_Outreach table

The covid vaccine code values were determined using https://www.cms.gov/medicare/medicare-part-b-drug-average-sales-price/covid-19-vaccines-and-monoclonal-antibodies \*/

CREATE OR REPLACE FUNCTION get\_VaxCovid\_Current (p\_memberID IN INT)

RETURN INT IS

v\_covidVax INT(2);

BEGIN

SELECT count(CF.code\_type)

INTO v\_covidVax

FROM CLAIM\_FACT CF

JOIN DIM\_MEMBER M

ON (M.memberID = CF.memberID)

WHERE CF.code\_value IN ('0001A','0002A','0003A','0004A','0011A','0012A','0013A','0064A','0071A', '0072A','91300','91301','91302','91303','91304','91305','91306','0021A','0022A','0031A','0034A','0051A','0052A','0053A','0054A')

AND CF.memberID = p\_memberID

AND CLAIMSTATUSCD like 'A'

AND M.enddate =

(select max(enddate)

from dim\_member

where memberID = p\_memberID);

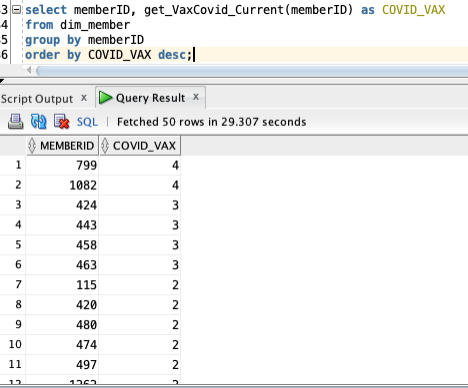
RETURN v\_covidVax;

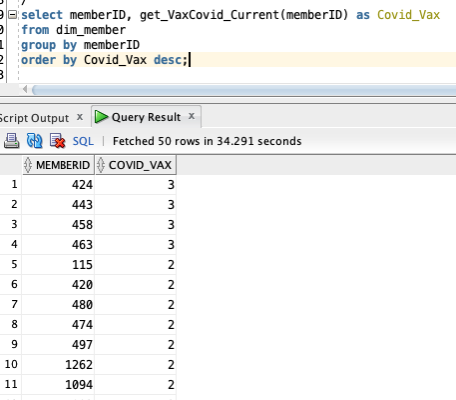
END get\_VaxCovid\_Current;

/

--TEST FUNCTION

\*The first image is before the addition of the claimstatuscd filter, second is after adjusting for only approved claims

.



1. **Function GET\_VaxFlu**

/\*Katy Matulay

GET\_VAXFLU function determines a members historical flu vaccine status as an int value, to be used in the high\_risk\_patient views

The flu vaccine code values were determined using https://www.cms.gov/medicare/preventive-services/flu-shot-coding \*/

CREATE OR REPLACE FUNCTION get\_VaxFlu (p\_memberID IN INT)

RETURN INT IS

v\_fluVax INT(4);

BEGIN

SELECT count(code\_type)

INTO v\_fluVax

FROM CLAIM\_FACT

WHERE code\_value IN ('90630','G0008','90653','90654','90655','90656','90657','90658','90660','90662',

'90672','90673','90674','90682','90685','90686','90687','90688','90689','90694','90756','Q2034','Q2035','Q2036','Q2037','Q2038','Q2039')

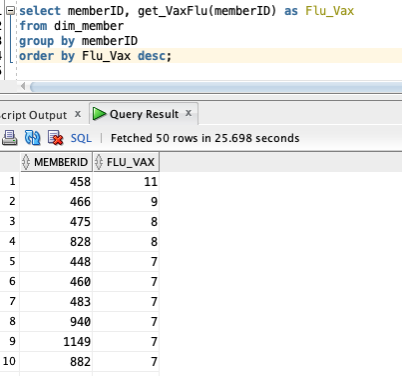
AND CLAIMSTATUSCD like 'A'

And memberID = p\_memberID;

RETURN v\_fluVax;

END get\_VaxFlu;

/



\*Note- based on the numbers returned, further analysis and functional knowledge is necessary. For further analysis, see notes after the function get\_VaxFlu\_Current.

1. **Function GET\_VaxFlu\_Current**

/\*Katy Matulay

GET\_VAXFLU\_CURRENT function determines a members current enrollment period flu vaccine status as an int value, to be used in the high\_risk\_patient views and HR\_Mem\_Outreach table

The flu vaccine code values were determined using https://www.cms.gov/medicare/preventive-services/flu-shot-coding \*/

CREATE OR REPLACE FUNCTION get\_VaxFlu\_Current (p\_memberID IN INT)

RETURN INT IS

v\_fluVax INT(2);

BEGIN

SELECT count(CF.code\_type)

INTO v\_fluVax

FROM CLAIM\_FACT CF

JOIN DIM\_MEMBER M

ON (M.memberID = CF.memberID)

WHERE CF.code\_value IN ('90630','G0008','90653','90654','90655','90656','90657','90658','90660','90662', '90672','90673','90674','90682','90685','90686','90687','90688','90689','90694','90756','Q2034','Q2035','Q2036','Q2037','Q2038','Q2039')

AND CF.CLAIMSTATUSCD like 'A'

And CF.memberID = p\_memberID

AND M.enddate =

(select max(enddate)

from dim\_member

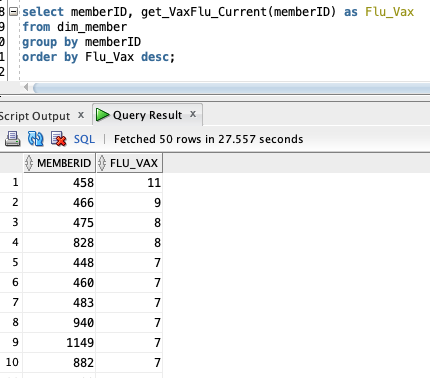
where memberID = p\_memberID);

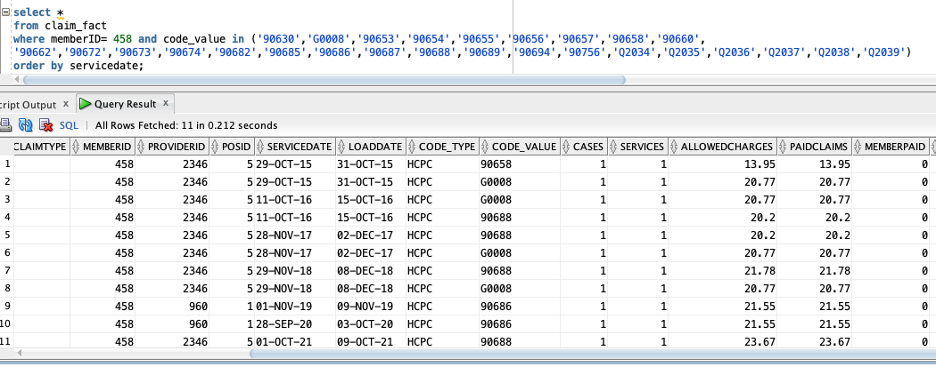
RETURN v\_fluVax;

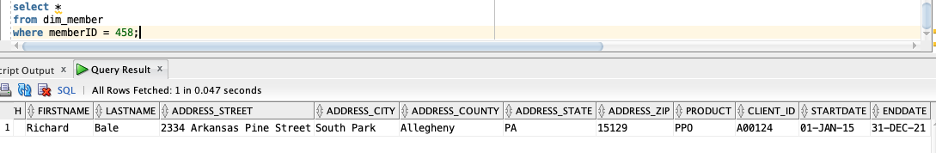
END get\_VaxFlu\_Current;

/

\*Note- based on the numbers below, it is likely that two issues are happening: 1) some of the code\_values represent various components for flu vaccine administration (i.e. vaccine component + administration overhead), resulting in duplicate counts; 2) enrollment periods for continuously enrolled members can span multiple years as evidenced below. Depending on analysis goals, a different measure of time and refinement of code\_values beyond the scope of this project may be necessary to capture current year flu vaccine status. However, the function still provides useful information to determine if a member consistently gets their flu shots or if additional incentives are necessary for high risk patients.







1. **Function GET\_PC**

/\*Katy Matulay

GET\_PC function determines the number of preventative care visits as an int, to be used in the high\_risk\_patient views

Preventative care visits are denoted by the PROF\_CD value of P43\*/

CREATE OR REPLACE FUNCTION get\_PC (p\_memberID IN INT)

RETURN NUMBER IS

v\_PCvisits INT(3);

BEGIN

SELECT count(A.code\_type)

INTO v\_PCvisits

FROM CLAIM\_FACT A

JOIN DIM\_PROCEDURE B

ON (A.CODE\_TYPE = B.CODE\_TYPE

AND A.CODE\_VALUE = B.CODE\_VALUE)

WHERE B.PROF\_CD like 'P43'

AND A.memberID = p\_memberID

AND A.claimstatuscd like 'A';

RETURN v\_PCvisits;

END get\_PC;

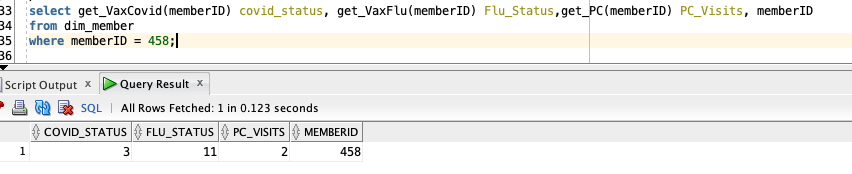
/

-Test Functions

select get\_VaxCovid(memberID) covid\_status, get\_VaxFlu(memberID) Flu\_Status,get\_PC(memberID) PC\_Visits, memberID

from dim\_member

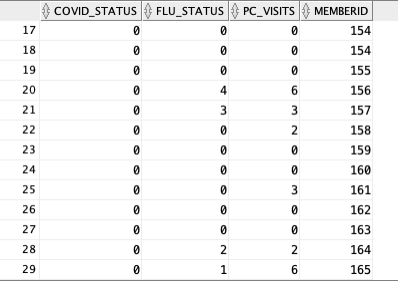
where memberID = 458;



select get\_VaxCovid(memberID) covid\_status, get\_VaxFlu(memberID) Flu\_Status,get\_PC(memberID) PC\_Visits, memberID

from dim\_member

where memberID is not null;



1. **Function GET\_PC\_Current**

/\*Katy Matulay

GET\_PC\_CURRENT function determines the number of preventative care visits as an int per current enrollment period, to be used in the high\_risk\_patient views and HR\_Mem\_Outreach table

Preventative care visits are denoted by the PROF\_CD value of P43\*/

CREATE OR REPLACE FUNCTION get\_PC\_Current (p\_memberID IN INT)

RETURN NUMBER IS

v\_PCvisits INT(3);

BEGIN

SELECT count(A.code\_type)

INTO v\_PCvisits

FROM CLAIM\_FACT A

JOIN DIM\_PROCEDURE B

ON (A.CODE\_TYPE = B.CODE\_TYPE

AND A.CODE\_VALUE = B.CODE\_VALUE)

JOIN DIM\_MEMBER M

ON (M.memberID = A.memberID)

WHERE B.PROF\_CD like 'P43'

AND A.memberID = p\_memberID

AND A.claimstatuscd like 'A'

AND M.enddate =

(select max(enddate)

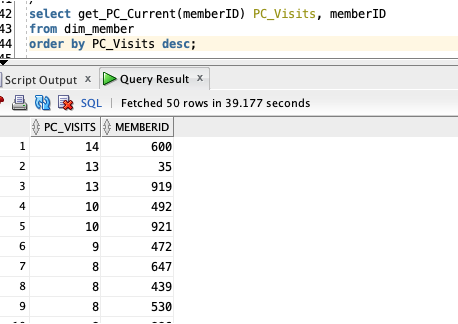
from dim\_member

where memberID = p\_memberID);

RETURN v\_PCvisits;

END get\_PC\_Current;

/



1. **Create View HIST\_HIGH\_RISK\_PATIENT**

The final resulting analytic view aims to provide a summary snapshot of all high-risk patients and their historical preventative care records. This view is then used for the remaining procedures and triggers.

/\* Katy Matulay

Step1: The following view represents a snapshot of patients’ historical tabulations regarding high risk conditions and preventative care. It utilizes the functions: calc\_age, get\_VaxCovid, get\_VaxFlu, and get\_PC to determine cumulative historical counts\*/

DROP VIEW v\_HIST\_HIGH\_RISK\_PATIENTS;

CREATE OR REPLACE VIEW v\_HIST\_HIGH\_RISK\_PATIENTS as(

select distinct(memberID), firstname, lastname, gender, dateofbirth, MemberAge, Covid\_Vaccine, Flu\_Vaccine, PC\_visits, Comorbidity\_Count

from (

select M.memberID, M.firstname, M.lastname, M.gender, M.dateofbirth,

calc\_age(M.dateofbirth) as MemberAge,

get\_VaxCovid(M.memberID) as Covid\_Vaccine,

get\_VaxFlu(M.memberID) as Flu\_Vaccine,

get\_PC(M.memberID) as PC\_visits,

CALC\_Comorbid\_Current(M.memberID) as Comorbidity\_Count

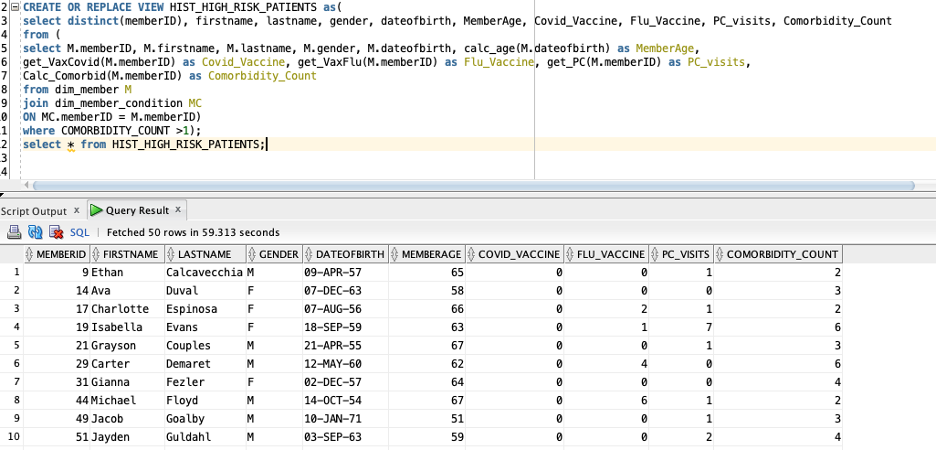
from dim\_member M)

where Comorbidity\_Count >1

);

/\*view the results\*/

select \* from HIST\_HIGH\_RISK\_PATIENTS;



1. **Create Table HR\_MEM\_Outreach**

/\*Step 2: Create DDL for High\_Risk\_Member\_Outreach table as HR\_Mem\_Outreach

The following DDL creates the table HR\_Mem\_Outreach is to be utilized in the procedure populate\_member\_outreach and triggers Claim\_Fact\_HighDollar\_AIUR and Claim\_Fact\_HighRiskProcedure (Jake Williamson's), and will store

member contact details, condition information, demographics, and current high risk condition calcs from earlier functions\*/

DROP TABLE HR\_MEM\_Outreach;

CREATE TABLE HR\_MEM\_Outreach

(IDNO INT NOT NULL,

MemberID INT NOT NULL,

Gender CHAR(1)NOT NULL,

DateOfBirth DATE NOT NULL,

Age INT NOT NULL,

Firstname VARCHAR2(100) NOT NULL,

lastname VARCHAR2(100) NOT NULL,

address\_street VARCHAR2(255),

address\_city VARCHAR2(100),

address\_County VARCHAR2(50),

address\_state CHAR(2),

address\_zip CHAR(5),

Comorbid\_Curr INT NOT NULL,

Covid\_Vax\_Curr INT NOT NULL,

Flu\_Vax\_Curr INT NOT NULL,

PC\_Visit\_Curr INT NOT NULL,

High\_Dollar CHAR(1) NOT NULL,

High\_Risk\_Codes INT NOT NULL,

PRIMARY KEY (IDNO));

/\*Step 3: Create sequence for primary key\*/

create sequence seq\_hr\_mem\_outreach start with 1 increment by 1;

1. **Procedure Populate\_member\_outreach**

/\*Step 4: Create Procedure

The following procedure populate\_member\_outreach creates a HIGH RISK MEMBER OUTREACH (HR\_MEM\_OUTREACH) table which can be utilized to input values from Triggers for members that have high dollar claims and high risk conditions\*/

CREATE OR REPLACE PROCEDURE populate\_member\_outreach AS

BEGIN

DELETE FROM HR\_Mem\_Outreach;

INSERT INTO HR\_Mem\_Outreach(IDNO, MemberID, Gender, DateOfBirth, Age, Firstname, Lastname, address\_street, address\_city, address\_county, address\_state, address\_zip, Comorbid\_Curr, Covid\_Vax\_Curr, Flu\_Vax\_Curr, PC\_Visit\_Curr, High\_Dollar, High\_Risk\_Codes)

SELECT

seq\_hr\_mem\_outreach.NEXTVAL,

HR.MemberID,

HR.Gender,

HR.DateOfBirth,

Calc\_Age(HR.DateOfBirth),

HR.Firstname,

HR.Lastname,

M.address\_street,

M.address\_city,

M.address\_county,

M.address\_state,

M.address\_zip,

CALC\_Comorbid\_Current(HR.memberID),

get\_VaxCovid\_Current(HR.memberID),

get\_VaxFlu\_Current(HR.memberID),

get\_PC\_Current(HR.memberID),

'N' as High\_Dollar,

0 as High\_Risk\_Codes

FROM v\_HIST\_HIGH\_RISK\_PATIENTS HR

JOIN DIM\_MEMBER M

ON (HR.MemberID = M.MemberID);

COMMIT;

End populate\_member\_outreach;

/\*Step 5: Execute the procedure\*/

exec populate\_member\_outreach;

1. **Trigger Claim\_Fact\_HighDollar\_AIUR**

/\*Step 6: Create trigger

The following trigger Claim\_Fact\_HighDollar\_AIUR triggers when a claim on CLAIM\_FACT contains an allowed charge > $25,000 and updates the field High\_Dollar with 'Y' in the table HR\_MEM\_OUTREACH

NOTE: data would need to be reloaded/inserted after creating this to see the results \*/

CREATE OR REPLACE TRIGGER Claim\_Fact\_HighDollar\_AIUR

AFTER

INSERT OR

UPDATE OF allowedcharges

ON Claim\_Fact

FOR EACH ROW

BEGIN

IF :NEW.allowedcharges > 25000 THEN

UPDATE HR\_MEM\_Outreach

SET High\_Dollar = 'Y'

WHERE MemberID = :NEW.MEMBERID;

END IF;

END;

/

/\*Step 7: Optional- drop and reload all data using the insert.txt file to see the results of the trigger\*/

/\*Step 8: View the results\*/

select \* from HR\_Mem\_Outreach;

* 1. **Jacob Stank PL/SQL units:**

This section contains two procedures, one function, and one trigger. The procedures are created to build summary tables for both claims and enrollment which will allow for quickly looking into month over month analysis and enable the creation of exhibits to complete per member per month (PMPM) calculations to normalize the claims by membership. The function is developed to quickly capture claims over 25K in allowed charges which will help for a clinical services analyst to research the reason behind the high charges to see if additional intervention and follow-up is needed. The triggers developed create two use case specific summary tables which will enable for BI tools such as Power BI, Cognos, or Tableau to reference for visualizations.

**Procedure 01:**

/\*\*\* Developed by Jacob Stank\*\*\*/

/\*\*\* This section of code is intended to create a Member Summary table which will store data at the Member - Month level for quick summarization of information \*\*\*/

/\*\*\* Step 01: Create DDL for Member Summary Table \*\*\*/

DROP TABLE SUM\_MEMBER\_CT;

CREATE TABLE SUM\_MEMBER\_CT

(IDNO INT NOT NULL

,MemberID INT NOT NULL

,PRODUCT CHAR(3) NOT NULL

,CLIENT\_ID CHAR(6) NOT NULL

,Gender CHAR(1) NOT NULL

,AgeRange VARCHAR2(20) NOT NULL

,MemberRegion VARCHAR2(20) NOT NULL

,DIABETES INT NOT NULL

,CAD INT NOT NULL

,CHF INT NOT NULL

,HYPERTENSION INT NOT NULL

,COPD INT NOT NULL

,IncMonth DATE NOT NULL

,Member\_Count NUMBER(10,0) NOT NULL

,CoMorbity\_Count NUMBER(10,0) NOT NULL

,PRIMARY KEY (IDNO)

);

/\*\*\* Step 02: Create sequence number for primary key \*\*\*/

create sequence seq\_mbrsum start with 1 increment by 1;

/\*\*\* Step 03: Define the Procedure to populate member summary. This table will be a complete replace each time run. Future changes would be to look to make it incremental load.\*\*\*/

CREATE OR REPLACE PROCEDURE populate\_member\_summary AS

BEGIN

DELETE FROM SUM\_Member\_Ct;

INSERT INTO SUM\_Member\_Ct (IDNO, MemberID, PRODUCT, CLIENT\_ID, Gender, AgeRange, MemberRegion, DIABETES, CAD ,CHF, HYPERTENSION , COPD, IncMonth , Member\_Count , CoMorbity\_Count)

SELECT

seq\_mbrsum.NEXTVAL

,A1.MemberID

,A1.PRODUCT

,A1.CLIENT\_ID

,A1.Gender

,CASE WHEN ROUND((sysdate - A1.DateofBirth)/365.25) BETWEEN 0 AND 17 THEN 'Cat01\_00\_17'

WHEN ROUND((sysdate - A1.DateofBirth)/365.25) BETWEEN 18 AND 30 THEN 'Cat02\_18\_30'

WHEN ROUND((sysdate - A1.DateofBirth)/365.25) BETWEEN 31 AND 40 THEN 'Cat03\_31\_40'

WHEN ROUND((sysdate - A1.DateofBirth)/365.25) BETWEEN 41 AND 50 THEN 'Cat04\_41\_50'

WHEN ROUND((sysdate - A1.DateofBirth)/365.25) BETWEEN 51 AND 57 THEN 'Cat05\_51\_57'

WHEN ROUND((sysdate - A1.DateofBirth)/365.25) BETWEEN 58 AND 64 THEN 'Cat06\_56\_64'

ELSE 'Cat07\_65+'

END as AgeRange

,CASE WHEN A1.address\_County NOT IN ('Allegheny') Then A1.address\_County

when substr(A1.address\_city,1,1) = 'P' THEN 'Allegheny - South'

when substr(A1.address\_city,1,1) IN ('M','B','W','S') Then 'Allegheny - North'

else 'Allegheny - Other'

END AS MemberRegion

,CASE WHEN B1.DIABETES IS NULL THEN 0 ELSE B1.DIABETES END AS DIABETES

,CASE WHEN B1.CAD IS NULL THEN 0 ELSE B1.CAD END AS CAD

,CASE WHEN B1.CHF IS NULL THEN 0 ELSE B1.CHF END AS CHF

,CASE WHEN B1.HYPERTENSION IS NULL THEN 0 ELSE B1.HYPERTENSION END AS HYPERTENSION

,CASE WHEN B1.COPD IS NULL THEN 0 ELSE B1.COPD END AS COPD

,A1.IncMonth

,1 AS Member\_Count

,CASE WHEN B1.CoMorbity\_Count IS NULL THEN 0 ELSE B1.CoMorbity\_Count END AS CoMorbity\_Count

FROM (SELECT A.\*

,B.IncMonth

FROM DIM\_MEMBER A

INNER JOIN DIM\_DATE B

ON (B.IncMonth BETWEEN A.StartDate and A.EndDate)

) A1

LEFT JOIN

(SELECT C.\*

,(C.DIABETES + C.CAD + C.CHF + C.HYPERTENSION + C.COPD) AS CoMorbity\_Count

,D.IncMonth

FROM DIM\_MEMBER\_CONDITION C

INNER JOIN DIM\_DATE D

ON (D.IncMonth BETWEEN C.StartDate and C.EndDate)

) B1

ON (A1.MemberID = B1.MemberID

AND A1.IncMonth = B1.IncMonth)

;

COMMIT;

End populate\_member\_summary;

/

/\*\*\* Step 04 - Execute inserting data into the member summary \*\*\*/

execute populate\_member\_summary;

**Procedure 02:**

/\*\*\* Developed by Jacob Stank \*\*\*\*/

/\*\*\* This next section of code will build a claims summary table to allow for quick summarization of claims detail for adhoc queries. It is designed currently to be a full table

replace and future iterations would look to make it an incremental load. \*\*\*/

/\*\*\* Step 01: Create DDL for Claims Summary \*\*\*/

DROP TABLE SUM\_CLAIMS\_CT;

CREATE TABLE SUM\_CLAIMS\_CT

(IDNO INT NOT NULL

,MemberID INT NOT NULL

,PRODUCT CHAR(3) NOT NULL

,CLIENT\_ID CHAR(6) NOT NULL

,Gender CHAR(1) NOT NULL

,AgeRange VARCHAR2(20) NOT NULL

,MemberRegion VARCHAR2(20) NOT NULL

,DIABETES INT NOT NULL

,CAD INT NOT NULL

,CHF INT NOT NULL

,HYPERTENSION INT NOT NULL

,COPD INT NOT NULL

,ProviderID INT NOT NULL

,ProviderRegion VARCHAR2(20) NOT NULL

,SpecRup\_DESC VARCHAR2(100) NOT NULL

,RPT\_CD VARCHAR2(10) NOT NULL

,LVL\_01 VARCHAR2(10) NOT NULL

,LVL\_02 VARCHAR2(100) NOT NULL

,LVL\_03 VARCHAR2(100) NOT NULL

,pos\_Desc VARCHAR2(100) NOT NULL

,IncMonth DATE NOT NULL

,CoMorbity\_Count NUMBER(10,0) NOT NULL

,Cases NUMBER(12,0) NOT NULL

,Services NUMBER(12,0) NOT NULL

,ClaimCount NUMBER(12,0) NOT NULL

,AllowedCharges NUMBER(14,2) NOT NULL

,PaidClaims NUMBER(14,2) NOT NULL

,MemberPaid NUMBER(14,2) NOT NULL

,OPL NUMBER(14,2) NOT NULL

,PRIMARY KEY (IDNO)

);

/\*\*\* Step 02: Create sequence number for primary key \*\*\*/

create sequence seq\_clmsum start with 1 increment by 1;

/\*\*\* Step 03: Define the Procedure to populate Claims summary \*\*\*/

CREATE OR REPLACE PROCEDURE populate\_claims\_summary AS

BEGIN

DELETE FROM SUM\_CLAIMS\_CT;

INSERT INTO SUM\_CLAIMS\_CT (IDNO,MemberID,PRODUCT,CLIENT\_ID,Gender,AgeRange,MemberRegion,DIABETES,CAD,CHF,HYPERTENSION,COPD,ProviderID,ProviderRegion,SpecRup\_DESC,RPT\_CD,LVL\_01,LVL\_02,LVL\_03,pos\_Desc,IncMonth,CoMorbity\_Count,Cases,Services,ClaimCount,AllowedCharges,PaidClaims,MemberPaid,OPL)

SELECT

seq\_clmsum.NEXTVAL

,S01.MemberID

,COALESCE(S02.PRODUCT,'NA') AS PRODUCT

,COALESCE(S02.CLIENT\_ID,'NA') AS CLIENT\_ID

,COALESCE(S02.Gender,'X') AS Gender

,COALESCE(S02.AgeRange,'NA') AS AgeRange

,COALESCE(S02.MemberRegion,'NA') AS MemberRegion

,COALESCE(S02.DIABETES,0) AS DIABETES

,COALESCE(S02.CAD,0) AS CAD

,COALESCE(S02.CHF,0) AS CHF

,COALESCE(S02.HYPERTENSION,0) AS HYPERTENSION

,COALESCE(S02.COPD,0) AS COPD

,S01.ProviderID

,S01.ProviderRegion

,S01.SpecRup\_DESC

,S01.RPT\_CD

,S01.LVL\_01

,S01.LVL\_02

,S01.LVL\_03

,S01.POS\_DESC

,S01.IncMonth

,COALESCE(S02.CoMorbity\_Count,0) AS CoMorbity\_Count

,S01.Cases

,S01.Services

,S01.ClaimCount

,S01.AllowedCharges

,S01.PaidClaims

,S01.MemberPaid

,S01.OPL

FROM (

SELECT

d01.MemberID

,d01.ProviderID

,d01.ProviderRegion

,d01.SpecRup\_DESC

,d01.RPT\_CD

,d01.LVL\_01

,d01.LVL\_02

,d01.LVL\_03

,d01.POS\_DESC

,d01.IncMonth

,SUM(d01.Cases) AS Cases

,SUM(d01.Services) AS Services

,COUNT(DISTINCT d01.ClaimID) AS ClaimCount

,SUM(d01.AllowedCharges) AS AllowedCharges

,SUM(d01.PaidClaims) as PaidClaims

,SUM(d01.MemberPaid) as MemberPaid

,SUM(d01.OPL) as OPL

FROM (

SELECT

A.\*

,CASE WHEN D.address\_County NOT IN ('Allegheny') Then D.address\_County

when substr(D.address\_city,1,1) = 'P' THEN 'Allegheny - South'

when substr(D.address\_city,1,1) IN ('M','B','W','S') Then 'Allegheny - North'

else 'Allegheny - Other'

END AS ProviderRegion

,E.SpecRup\_DESC

,CASE WHEN A.ClaimType IN ('I','O') THEN COALESCE(F1.RPT\_CD,'OTH')

ELSE COALESCE(F2.RPT\_CD,'OTH')

END AS RPT\_CD

,CASE WHEN A.ClaimType IN ('I','O') THEN COALESCE(F1.LVL\_01,'OTH')

ELSE COALESCE(F2.LVL\_01,'OTH')

END AS LVL\_01

,CASE WHEN A.ClaimType IN ('I','O') THEN COALESCE(F1.LVL\_02,'OTH')

ELSE COALESCE(F2.LVL\_02,'OTH')

END AS LVL\_02

,CASE WHEN A.ClaimType IN ('I','O') THEN COALESCE(F1.LVL\_03,'OTH')

ELSE COALESCE(F2.LVL\_03,'OTH')

END AS LVL\_03

,B.POS\_DESC

,I.IncMonth

FROM CLAIM\_FACT A

INNER JOIN DIM\_DATE I

ON (A.ServiceDate BETWEEN I.StartDate AND I.EndDate)

LEFT JOIN DIM\_PLACE\_OF\_SERVICE B

ON (A.posID = B.posID)

LEFT JOIN DIM\_PROCEDURE C

ON (A.Code\_Type = C.Code\_Type

AND A.Code\_Value = C.Code\_Value )

LEFT JOIN DIM\_PROVIDER D

ON (A.ProviderID = D.ProviderID)

LEFT JOIN DIM\_PROVIDER\_SPECIALTY E

ON (D.SpecialtyID = E.SpecialtyID)

LEFT JOIN DIM\_REPORTING\_CATEGORY F1

ON (C.FACILITY\_CD = F1.RPT\_CD)

LEFT JOIN DIM\_REPORTING\_CATEGORY F2

ON (C.PROF\_CD = F2.RPT\_CD)

) d01

GROUP BY

d01.MemberID

,d01.ProviderID

,d01.ProviderRegion

,d01.SpecRup\_DESC

,d01.RPT\_CD

,d01.LVL\_01

,d01.LVL\_02

,d01.LVL\_03

,d01.POS\_DESC

,d01.IncMonth

) S01

LEFT JOIN SUM\_Member\_Ct S02

ON (S01.MemberID = S02.MemberID

AND S01.IncMonth = S02.IncMonth);

COMMIT;

END populate\_claims\_summary;

/

/\*\*\* Step 04: Execute inserting data into the member summary \*\*\*/

execute populate\_claims\_summary;

**Function 01:**

/\*\*\* Developed by Jacob Stank \*\*\*/

/\*\*\* The goal of this next piece is to build a check to see if Claim is over 25K in Allowed Charges. This will allow for investigation and tracking of these

claims by the clinical teams as well as enable forecast analysis to back out of claim outliers to smooth the curve for projections. \*\*\*/

CREATE OR REPLACE FUNCTION Calc\_HD\_Claim (p\_claimID IN INT)

RETURN INT IS

v\_HighDollarClaim INT;

BEGIN

SELECT D.ClaimID INTO v\_HighDollarClaim

FROM

(SELECT ClaimID, SUM(AllowedCharges) AS AMT

FROM CLAIM\_FACT

WHERE ClaimID = p\_ClaimID

GROUP BY ClaimID

) D

WHERE D.AMT>2500.00;

RETURN(v\_HighDollarClaim);

END Calc\_HD\_Claim;

/

**Trigger 01:**

/\*\*\* Developed by Jacob Stank \*\*\*/

/\*\*\*

This next section of code is developed to populate two reporting tables that will allow for a BI tool to leverage to create Trend Reports that capture Utilization per 1k,

Cost per Utilization, and PMPM at the level of detail needed for the specified report.

Summary Report 01 (SUMMARY\_RPT\_01) will compile the data Needed to produce a Trend Report analyzing differences at the Cobmorbity Levels

Summary Report 02 (SUMMARY\_RPT\_02) will compile the data needed to produce a Trend Report at RPT Category 01 & 02 Levels to determine the categories to drill into detail

for further analysis.

\*\*\*/

/\*\*\* Step 01: Create the DDL for the Report Tables. \*\*\*/

DROP TABLE SUMMARY\_RPT\_01;

CREATE TABLE SUMMARY\_RPT\_01

(IDNO INT NOT NULL

,CoMorbity\_Count NUMBER(10,0) NOT NULL

,IncMonth DATE NOT NULL

,Cases NUMBER(12,0) NOT NULL

,Services NUMBER(12,0) NOT NULL

,ClaimCount NUMBER(12,0) NOT NULL

,AllowedCharges NUMBER(14,2) NOT NULL

,PaidClaims NUMBER(14,2) NOT NULL

,MemberPaid NUMBER(14,2) NOT NULL

,OPL NUMBER(14,2) NOT NULL

,Member\_Count NUMBER(10,0) NOT NULL

,PRIMARY KEY (IDNO)

);

DROP TABLE SUMMARY\_RPT\_02;

CREATE TABLE SUMMARY\_RPT\_02

(IDNO INT NOT NULL

,LVL\_01 VARCHAR2(10) NOT NULL

,LVL\_02 VARCHAR2(100) NOT NULL

,IncMonth DATE NOT NULL

,Cases NUMBER(12,0) NOT NULL

,Services NUMBER(12,0) NOT NULL

,ClaimCount NUMBER(12,0) NOT NULL

,AllowedCharges NUMBER(14,2) NOT NULL

,PaidClaims NUMBER(14,2) NOT NULL

,MemberPaid NUMBER(14,2) NOT NULL

,OPL NUMBER(14,2) NOT NULL

,Member\_Count NUMBER(10,0) NOT NULL

,PRIMARY KEY (IDNO)

);

/\*\*\*Step 02: Create sequence number for primary key purposes \*\*\*/

create sequence seq\_rpt01 start with 1 increment by 1;

create sequence seq\_rpt02 start with 1 increment by 1;

/\*\*\*Step 03: Define the Trigger which will populate these Report Tables after SUM\_CLAIM\_FACT is created \*\*\*/

CREATE OR REPLACE TRIGGER report\_insert

AFTER INSERT ON SUM\_CLAIMS\_CT

BEGIN

DELETE FROM SUMMARY\_RPT\_01;

INSERT INTO SUMMARY\_RPT\_01 (IDNO,CoMorbity\_Count,IncMonth,Cases,Services,ClaimCount,AllowedCharges,PaidClaims,MemberPaid,OPL,Member\_Count)

SELECT seq\_rpt01.NEXTVAL AS IDNO

,D2.\*

FROM (

SELECT

D1.CoMorbity\_Count

,D1.IncMonth

,SUM(D1.Cases) AS Cases

,SUM(D1.Services) AS Services

,SUM(D1.ClaimCount) AS ClaimCount

,Sum(D1.AllowedCharges) AS AllowedCharges

,Sum(D1.PaidClaims) AS PaidClaims

,SUM(D1.MemberPaid) AS MemberPaid

,SUM(D1.OPL) AS OPL

,SUM(D1.Member\_Count) AS Member\_Count

FROM (

SELECT

A.CoMorbity\_Count

,B.IncMonth

,SUM(A.Cases) AS Cases

,SUM(A.Services) AS Services

,SUM(A.ClaimCount) AS ClaimCount

,Sum(A.AllowedCharges) AS AllowedCharges

,Sum(A.PaidClaims) AS PaidClaims

,SUM(A.MemberPaid) AS MemberPaid

,SUM(A.OPL) AS OPL

,SUM(0) AS Member\_Count

FROM SUM\_CLAIMS\_CT A

INNER JOIN DIM\_DATE B

ON(A.IncMonth = B.IncMonth

AND B.tValue0 BETWEEN 2 AND 37)

GROUP BY A.CoMorbity\_Count, B.IncMonth

UNION

SELECT

A1.CoMorbity\_Count

,B1.IncMonth

,SUM(0) AS Cases

,SUM(0) AS Services

,SUM(0) AS ClaimCount

,SUM(0) AS AllowedCharges

,SUM(0) AS PaidClaims

,SUM(0) AS MemberPaid

,SUM(0) AS OPL

,SUM(A1.Member\_Count) AS Member\_Count

FROM SUM\_MEMBER\_CT A1

INNER JOIN DIM\_DATE B1

ON(A1.IncMonth = B1.IncMonth

AND B1.tValue0 BETWEEN 2 AND 37)

GROUP BY A1.CoMorbity\_Count, B1.IncMonth

) D1

GROUP BY D1.CoMorbity\_Count, D1.IncMonth

) D2;

DELETE FROM SUMMARY\_RPT\_02;

INSERT INTO SUMMARY\_RPT\_02 (IDNO, LVL\_01,LVL\_02,IncMonth,Cases,Services,ClaimCount,AllowedCharges,PaidClaims,MemberPaid,OPL,Member\_Count)

SELECT seq\_rpt02.NEXTVAL AS IDNO

,D2.\*

FROM (

SELECT

D1.LVL\_01

,D1.LVL\_02

,D1.IncMonth

,SUM(D1.Cases) AS Cases

,SUM(D1.Services) AS Services

,SUM(D1.ClaimCount) AS ClaimCount

,Sum(D1.AllowedCharges) AS AllowedCharges

,Sum(D1.PaidClaims) AS PaidClaims

,SUM(D1.MemberPaid) AS MemberPaid

,SUM(D1.OPL) AS OPL

,SUM(D1.Member\_Count) AS Member\_Count

FROM (

SELECT

A.LVL\_01

,A.LVL\_02

,B.IncMonth

,SUM(A.Cases) AS Cases

,SUM(A.Services) AS Services

,SUM(A.ClaimCount) AS ClaimCount

,Sum(A.AllowedCharges) AS AllowedCharges

,Sum(A.PaidClaims) AS PaidClaims

,SUM(A.MemberPaid) AS MemberPaid

,SUM(A.OPL) AS OPL

,SUM(0) AS Member\_Count

FROM SUM\_CLAIMS\_CT A

INNER JOIN DIM\_DATE B

ON(A.IncMonth = B.IncMonth

AND B.tValue0 BETWEEN 2 AND 37)

GROUP BY A.LVL\_01,A.LVL\_02, B.IncMonth

UNION

SELECT

'ENR' AS LVL\_01

,'ENR' AS LVL\_02

,B1.IncMonth

,SUM(0) AS Cases

,SUM(0) AS Services

,SUM(0) AS ClaimCount

,SUM(0) AS AllowedCharges

,SUM(0) AS PaidClaims

,SUM(0) AS MemberPaid

,SUM(0) AS OPL

,SUM(A1.Member\_Count) AS Member\_Count

FROM SUM\_MEMBER\_CT A1

INNER JOIN DIM\_DATE B1

ON(A1.IncMonth = B1.IncMonth

AND B1.tValue0 BETWEEN 2 AND 37)

GROUP BY B1.IncMonth

) D1

GROUP BY D1.LVL\_01, D1.LVL\_02, D1.IncMonth

) D2;

END;

/

* 1. **Jacob Williamson PL/SQL units:**

Function: HasDiabetes

Input: Member ID

Output: 1 if the member’s latest enrollment has diabetes, 0 otherwise.

/\* Jacob Williamson

Returns 1 if the member currently has diabetes, 0 otherwise

\*/

CREATE OR REPLACE FUNCTION HASDIABETES (

mem\_id IN number

)

RETURN number IS

has\_diabetes number;

BEGIN

SELECT Diabetes

INTO has\_diabetes

FROM DIM\_MEMBER\_CONDITION

WHERE MemberID = mem\_id

ORDER BY enddate DESC

FETCH FIRST 1 ROW ONLY; -- Only works on 12c and later

RETURN has\_diabetes;

END HASDIABETES;

/

**Trigger: Claim\_Fact\_HighRiskProcedure**

When: A claim is entered that has a procedure value of a high risk procedure

Do: Update the count of high risk procedures in the High Risk Member Outreach table

/\* Jacob Williamson

Triggers when a claim for a high-risk procedure is entered

Update the High Risk Member Outreach table and adds to the member's count

of high-risk procedures

Future: Create table for high-risk codes to prevent updating the list in multiple places in the trigger

\*/

CREATE OR REPLACE TRIGGER Claim\_Fact\_HighRiskProcedure

AFTER

INSERT OR

UPDATE OF code\_value

ON Claim\_Fact

FOR EACH ROW

BEGIN

IF INSERTING THEN

IF :NEW.code\_value IN ('023','236','246','247','266','270') THEN

UPDATE HR\_MEM\_Outreach

SET High\_Risk\_Codes = High\_Risk\_Codes + 1

WHERE MemberID = :NEW.MEMBERID;

END IF;

ELSE -- Updating

IF :NEW.code\_value IN ('023','236','246','247','266','270') AND :OLD.code\_value NOT IN ('023','236','246','247','266','270') THEN

UPDATE HR\_MEM\_Outreach

SET High\_Risk\_Codes = High\_Risk\_Codes + 1

WHERE MemberID = :NEW.MEMBERID;

ELSIF :NEW.code\_value NOT IN ('023','236','246','247','266','270') AND :OLD.code\_value IN ('023','236','246','247','266','270') THEN

UPDATE HR\_MEM\_Outreach

SET High\_Risk\_Codes = High\_Risk\_Codes - 1

WHERE MemberID = :NEW.MEMBERID;

END IF;

END IF;

END;

/

**Procedure: Populate Rebate Mailer Table**

Description: The Rebate Mailer table is meant to be consumed by an external service that generates and mails out rebates to members who meet certain criteria. This specific procedure adds members who have diabetes and have received a COVID vaccine.

/\* Jacob Williamson

Creates a table containing information for rebate mailers.

To be consumed by external service.

Null values on MAILED\_ON indicates that the rebate has not been mailed out.

\*/

DROP TABLE REBATE\_MAILER;

CREATE TABLE REBATE\_MAILER

(

IDNO INT NOT NULL,

MEMBERID INT NOT NULL,

FIRSTNAME VARCHAR2(100) NOT NULL,

LASTNAME VARCHAR2(100) NOT NULL,

ADDRESS\_STREET VARCHAR2(255) NOT NULL,

ADDRESS\_CITY VARCHAR2(100) NOT NULL,

ADDRESS\_STATE VARCHAR2(2) NOT NULL,

ADDRESS\_ZIP VARCHAR2(5) NOT NULL,

REASON VARCHAR2(255) NOT NULL,

MAILED\_ON DATE,

PRIMARY KEY (IDNO)

);

/\* Jacob Williamson

Creates sequence for Rebate Mailer table

\*/

CREATE SEQUENCE seq\_rebatemailer

MINVALUE 1

START WITH 1

INCREMENT BY 1;

/\* Jacob Williamson

Populates Rebate Mailer table based on if members meet the following condition:

Member has diabetes AND received a COVID vaccine

\*/

CREATE OR REPLACE PROCEDURE populate\_rebate\_mailer AS

BEGIN

INSERT INTO rebate\_mailer (IDNO, MEMBERID, FIRSTNAME, LASTNAME, ADDRESS\_STREET, ADDRESS\_CITY, ADDRESS\_STATE, ADDRESS\_ZIP, REASON)

SELECT seq\_rebatemailer.NEXTVAL, MEMBERID, FIRSTNAME, LASTNAME, ADDRESS\_STREET, ADDRESS\_CITY, ADDRESS\_STATE, ADDRESS\_ZIP, 'Diabetes and Covid Vaccine'

FROM DIM\_MEMBER

WHERE HASDIABETES(MEMBERID) = 1

AND get\_VaxCovid(MEMBERID) > 0

AND MEMBERID NOT IN (

SELECT MEMBERID

FROM REBATE\_MAILER

WHERE REASON LIKE 'Diabetes and Covid Vaccine%'

);

COMMIT;

END populate\_rebate\_mailer;

**Trigger 2:** Add New Entries to Rebate Mailer table as they come in

When: New claims for COVID vaccines are added

Do: If the member has diabetes and hasn’t already received a rebate mailer for their COVID vaccine, then add them to the mailer table.

/\* Jacob Williamson

Trigger adds new entries to the rebate mailer table as new members meet the criteria of:

Member has diabetes AND received a COVID vaccine

\*/

CREATE OR REPLACE TRIGGER rebate\_mailer\_diabetes\_and\_covidvaccine

AFTER

INSERT OR

UPDATE OF code\_value

ON Claim\_Fact

FOR EACH ROW

BEGIN

IF :NEW.code\_value IN ('0001A','0002A','0003A','0004A','0011A','0012A','0013A','0064A','0071A', '0072A','91300','91301','91302','91303','91304','91305','91306','0021A','0022A','0031A','0034A','0051A','0052A','0053A','0054A') AND HASDIABETES(:NEW.memberid) = 1 THEN

INSERT INTO rebate\_mailer (IDNO, MEMBERID, FIRSTNAME, LASTNAME, ADDRESS\_STREET, ADDRESS\_CITY, ADDRESS\_STATE, ADDRESS\_ZIP, REASON)

SELECT seq\_rebatemailer.NEXTVAL, MEMBERID, FIRSTNAME, LASTNAME, ADDRESS\_STREET, ADDRESS\_CITY, ADDRESS\_STATE, ADDRESS\_ZIP, 'Diabetes and Covid Vaccine'

FROM DIM\_MEMBER

WHERE MEMBERID = :NEW.memberid

AND :NEW.memberid NOT IN (

SELECT MEMBERID

FROM REBATE\_MAILER

WHERE REASON LIKE 'Diabetes and Covid Vaccine%'

);

END IF;

END;

/

1. **Visualizations and Analysis**

To save space in this final report, we’ll omit most analysis. To view all analysis, view the most up-to-date Jupyter Notebook saved on our Github here: <https://github.com/zachcarlson/InsuranceDatabase/blob/main/insurance_database.ipynb>

As mentioned previously, we were unable to create PL/SQL units within our MySQL database that was created with Heroku/ClearDB due to security issues on having our database on a shared cluster. Purchasing a dedicated cluster was not feasible for this project, thus the PL/SQL units were implemented in a local Oracle environment. However, the data in both MySQL and Oracle are the same and both configurations can be connected in Python. We wanted to ensure analysis was conducted on a shared cluster, so the MySQL database configuration was used.

All SQLAlchemy requires for connection is a connection string URL containing the username, password, localhost name, and database name. Our Github is publicly available, so to prevent expose our login credentials, we saved the information in a Python file called database\_info.py. This file contained one variable of type string, connection\_string formatted as such:

#database\_info.py file is to store user specific connection string, replace the <> with your details

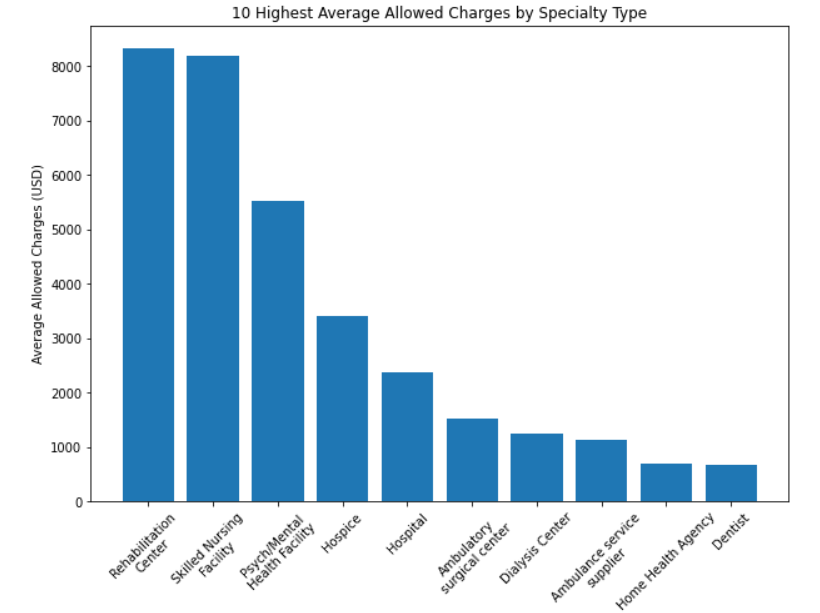
connection\_string **=** 'mysql+mysqldb://<username>:<password>@<localhost>/<database name>'

More information on setting up this connection safely is available on our Github.

The analytical goal was to conduct time series analysis on allowed charges for each claim, grouped by month. We hoped to use data from January 1st, 2015 to December 31st, 2020 to predict the values for January 1st, 2021 to December 31st, 2021.

The Jupyter Notebook contains several exploratory data analysis figures. We’ll showcase two: the final time plot and the claims predictions. Please view the Github to see additional figures, observations, and model setup.

To take full advantage of the star/snowflake schema, we combined the providers, provider\_specialty, and claims\_fact tables within Python to find which providers and specialties had the highest average allowed charges. This was possible using the .merge() method which acts similarly to a SQL join, where PK/FK columns are specified.



**Final Time Plot:**

This plot splits the claims into professional claims and outpatient claims. We used a 60-day rolling average to smooth out the curves as much as possible:

Chart, line chart, histogram

Description automatically generated

**Prediction:**

This plot uses total monthly allowed charges, rather than 60-day rolling average, to predict the values for 2021. The red line is the prediction whereas the blue line is the actual values for 2021.

Chart, line chart

Description automatically generated

1. **Conclusion**

We were able to successfully implement a MySQL database with Heroku/ClearDB to store healthcare insurance information as well as import that data into a Python working environment to do time series analysis. We used ARIMA to make monthly predictions for the month of 2021 with a somewhat decent accuracy. The model could be further improved with additional time. However, the data is inherently erratic and the seasonality might be quite weak compared to more standardized data, like sales or stocks.

Unfortunately, Heroku/ClearDB does not allow for the creation of PL/SQL units on its shared clusters, thus we implemented the PL/SQL units in Oracle. A business might be inclined to purchase a dedicated cluster on Heroku/ClearDB to implement these units to facilitate analysis in Python.

1. **References**

1.) “Covid-19 Vaccines and Monoclonal Antibodies.” *CMS*, <https://www.cms.gov/medicare/medicare-part-b-drug-average-sales-price/covid-19-vaccines-and-monoclonal-antibodies>.

2.) “Flu Shot Coding.” *CMS*, <https://www.cms.gov/medicare/preventive-services/flu-shot-coding>.

1. **Appendix**
   1. **Data Dictionary (Oracle)**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **TableName** | **ColunmName** | **DataType** | **Domain** | **Nullable** | **PK** | **FK** | **ColumnDescription** |
| claim\_fact | IDNO | INT | All | No | Yes | No | Surrogate Key of claimID, ClaimlineNum, claimStatusCd, & Servicedate |
| claim\_fact | ClaimID | INT | All | No | No | No | ID of claim submitted for payment |
| claim\_fact | ClaimLineNum | INT | All | No | No | No | Service line number of claim |
| claim\_fact | ClaimStatusCd | CHAR(1) | A, J, V | No | No | No | Claim Status Code, Approved, Voided, or Adjusted |
| claim\_fact | ClaimType | CHAR(1) | I, O, P | No | No | No | Type of Claim, Inpatient, Outpatient, or Professional |
| claim\_fact | MemberID | INT | All | No | No | Yes | Member ID who received Service |
| claim\_fact | ProviderID | INT | All | No | No | Yes | Provider ID who performed Service |
| claim\_fact | posID | INT | All | No | No | Yes | Place of Service ID where service performed |
| claim\_fact | ServiceDate | DATE | >= '01-JAN-2015' | No | No | No | Date Service happened |
| claim\_fact | LoadDate | DATE | >= '01-JAN-2015' | No | No | No | Date the claim record was loaded |
| claim\_fact | Code\_type | VARCHAR2(10) | DRG, RVCD, HCPC | No | No | No | Code Type for service performed |
| claim\_fact | Code\_value | VARCHAR2(10) | All | No | No | No | Code Value for service perfomed |
| claim\_fact | Cases | NUMBER(12,0) | All | No | No | No | Number of cases performed |
| claim\_fact | Services | NUMBER(12,0) | All | No | No | No | Number of Services performed |
| claim\_fact | AllowedCharges | NUMBER(14,2) | All | No | No | No | Amount Provider Charged for Service |
| claim\_fact | PaidClaims | NUMBER(14,2) | All | No | No | No | Amount Insurance Company Paid for Service |
| claim\_fact | MemberPaid | NUMBER(14,2) | All | No | No | No | Amount Member Paid for Service |
| claim\_fact | OPL | NUMBER(14,2) | All | No | No | No | Amount Covered by Other Insurance Carrier |
| dim\_provider | ProviderID | INT | All | No | Yes | No | Unique Provider ID |
| dim\_provider | SpecialtyID | INT | All | No | No | Yes | Provider Specialty ID |
| dim\_provider | ProviderType | INT | 1,2 | No | No | No | Provider Type, 1 - Facility, Ambulance, or Supplier, 2 - Individual Clinician |
| dim\_provider | OrgName | VARCHAR2(100) | All | No | No | No | Provider Organization Name |
| dim\_provider | FirstName | VARCHAR2(100) | All | No | No | No | Provider First Name |
| dim\_provider | LastName | VARCHAR2(100) | All | No | No | No | Provider Last Name |
| dim\_provider | address\_street | VARCHAR2(255) | All | Yes | No | No | Primary Provider Street Address |
| dim\_provider | address\_city | VARCHAR2(100) | All | Yes | No | No | Primary Provider City |
| dim\_provider | address\_county | VARCHAR2(50) | All | Yes | No | No | Primary Provider County |
| dim\_provider | address\_state | CHAR(2) | All | Yes | No | No | Primary Provider State Abbreviation |
| dim\_provider | address\_zip | CHAR(5) | Between 00001 and 99999 | Yes | No | No | Primary Provider Zip Code |
| dim\_member | IDNO | INT | All | No | Yes | No | Surrogate Key of MemberID, Product, StartDate, & EndDate |
| dim\_member | memberID | INT | All | No | No | No | Unique Member Identifier |
| dim\_member | Gender | CHAR(1) | M or F | No | No | No | Member Gender |
| dim\_member | dateOfBirth | DATE | >= '01-JAN-1900' | No | No | No | Member Date of Birth |
| dim\_member | firstname | VARCHAR2(100) | All | No | No | No | Member First Name |
| dim\_member | lastname | VARCHAR2(100) | All | No | No | No | Member Last Name |
| dim\_member | address\_street | VARCHAR2(255) | All | Yes | No | No | Primary Member Street Address |
| dim\_member | address\_city | VARCHAR2(100) | All | Yes | No | No | Primary Member City |
| dim\_member | address\_county | VARCHAR2(50) | All | Yes | No | No | Primary Member County |
| dim\_member | address\_state | CHAR(2) | All | Yes | No | No | Primary Member State Abbreviation |
| dim\_member | address\_zip | CHAR(5) | Between 00001 and 99999 | Yes | No | No | Primary Member Zip Code |
| dim\_member | Product | CHAR(3) | HMO, PPO | No | No | No | Product Type Member is enrolled |
| dim\_member | Client\_ID | CHAR(6) | A00123, A00124 | No | No | No | Client the member has a policy through |
| dim\_member | StartDate | DATE | >= '01-JAN-2015' | No | No | No | Member Product Start Date |
| dim\_member | EndDate | DATE | >= '01-JAN-2015' | No | No | No | Member Product Cancel Date |
| dim\_member\_condition | IDNO | INT | All | No | Yes | No | Surrogate Key of MemberID, All Conditions, StartDate & EndDate |
| dim\_member\_condition | memberID | INT | All | No | No | Yes | Unique Member Identifier |
| dim\_member\_condition | Diabetes | INT | 0,1 | No | No | No | 1 if Members has Diabetes, 0 else |
| dim\_member\_condition | CAD | INT | 0,1 | No | No | No | 1 if Members has Coronary Artery Disease, 0 else |
| dim\_member\_condition | CHF | INT | 0,1 | No | No | No | 1 if Members has Congestive Heart Failure, 0 else |
| dim\_member\_condition | Hypertension | INT | 0,1 | No | No | No | 1 if Members has Hypertension, 0 else |
| dim\_member\_condition | COPD | INT | 0,1 | No | No | No | 1 if Members has Chronic Obstructive Pulmonary Disease, 0 else |
| dim\_member\_condition | StartDate | DATE | >= '01-JAN-2015' | No | No | No | Member Condition Cohort Start Date |
| dim\_member\_condition | EndDate | DATE | >= '01-JAN-2015' | No | No | No | Member Condition Cohort End Date |
| dim\_date | IncMonth | DATE | >= '01-JAN-2015' | No | Yes | No | Incurred Date in date format, first day set to 1st |
| dim\_date | IncMonthNum | INT | ALL | No | No | No | Incurred Date in YYYYMM format |
| dim\_date | StartDate | DATE | >= '01-JAN-2015' | No | No | No | Incurred Date Start of month value |
| dim\_date | EndDate | DATE | >= '01-JAN-2015' | No | No | No | Incurred Date End of Month Value |
| dim\_date | tValue0 | INT | >= 0 | No | No | No | Integer value associated with Incurred Month used for Looping |
| dim\_place\_of\_service | posID | INT | All | No | Yes | No | Unique Place of Service ID |
| dim\_place\_of\_service | pos\_Desc | VARCHAR2(100) | All | No | No | No | Place of Service Description |
| dim\_place\_of\_service | pos\_category | VARCHAR2(10) | All | No | No | No | Place of Service Category |
| dim\_provider\_specialty | specialtyID | INT | All | No | Yes | No | Unique Provider Specialty ID |
| dim\_provider\_specialty | Specialty\_Desc | VARCHAR2(10) | All | No | No | No | Provider Specialty Description |
| dim\_provider\_specialty | SpecRup\_Cd | VARCHAR2(10) | All | No | No | No | Provider Specialty Rollup Code |
| dim\_provider\_specialty | SpecRup\_DESC | VARCHAR2(100) | All | No | No | No | Provider Specialty Rollup Description |
| dim\_provider\_specialty | SpecType | VARCHAR2(10) | All | No | No | No | Provider Specialty Type |
| dim\_procedure | IDNO | INT | All | No | Yes | No | Surrogate Key for Code\_Type and Code\_Value |
| dim\_procedure | Code\_Type | VARCHAR2(10) | All | No | No | No | Code Type for service performed |
| dim\_procedure | Code\_Value | VARCHAR2(10) | All | No | No | No | Code Value for service perfomed |
| dim\_procedure | procedure\_desc | VARCHAR2(255) | All | No | No | No | Code Description |
| dim\_procedure | facility\_cd | VARCHAR2(10) | All | No | No | Yes | Code Mapping if claim type Inpatient or Outpatient |
| dim\_procedure | prof\_cd | VARCHAR2(10) | All | No | No | Yes | Code Mapping if claim type is Professional |
| dim\_reporting\_category | rpt\_cd | VARCHAR2(10) | All | No | Yes | No | Reporting Category Code |
| dim\_reporting\_category | lvl\_01 | VARCHAR2(10) | All | No | No | No | Reporting Category level 01 Description |
| dim\_reporting\_category | lvl\_02 | VARCHAR2(100) | All | No | No | No | Reporting Category level 02 Description |
| dim\_reporting\_category | lvl\_03 | VARCHAR2(100) | All | No | No | No | Reporting Category level 03 Description |
| sum\_member\_ct | IDNO | INT | All | No | Yes | No | Surrogate Key to maintain uniqueness on summary table |
| sum\_member\_ct | MemberID | INT | All | No | No | Yes | Member ID who received Service |
| sum\_member\_ct | PRODUCT | CHAR(3) | HMO, PPO | No | No | No | Product Type Member is enrolled |
| sum\_member\_ct | CLIENT\_ID | CHAR(6) | A00123, A00124 | No | No | No | Client the member has a policy through |
| sum\_member\_ct | Gender | CHAR(1) | M or F | No | No | No | Member Gender |
| sum\_member\_ct | AgeRange | VARCHAR2(20) | All | No | No | No | Age Bucket for Member |
| sum\_member\_ct | MemberRegion | VARCHAR2(20) | All | No | No | No | Reporting Region for Member Location |
| sum\_member\_ct | DIABETES | INT | 0,1 | No | No | No | 1 if Members has Diabetes, 0 else |
| sum\_member\_ct | CAD | INT | 0,1 | No | No | No | 1 if Members has Coronary Artery Disease, 0 else |
| sum\_member\_ct | CHF | INT | 0,1 | No | No | No | 1 if Members has Congestive Heart Failure, 0 else |
| sum\_member\_ct | HYPERTENSION | INT | 0,1 | No | No | No | 1 if Members has Hypertension, 0 else |
| sum\_member\_ct | COPD | INT | 0,1 | No | No | No | 1 if Members has Chronic Obstructive Pulmonary Disease, 0 else |
| sum\_member\_ct | IncMonth | DATE | >= '01-JAN-2015' | No | No | Yes | Incurred Month of Service |
| sum\_member\_ct | Member\_Count | NUMBER(10,0) | 0,1 | No | No | No | 1 if member has coverage, 0 if not |
| sum\_member\_ct | CoMorbity\_Count | NUMBER(10,0) | >=0 | No | No | No | Count of Member Conditions |
| sum\_claims\_ct | IDNO | INT | All | No | Yes | No | Surrogate Key to maintain uniqueness on summary table |
| sum\_claims\_ct | MemberID | INT | All | No | No | Yes | Member ID who received Service |
| sum\_claims\_ct | PRODUCT | CHAR(3) | HMO, PPO | No | No | No | Product Type Member is enrolled |
| sum\_claims\_ct | CLIENT\_ID | CHAR(6) | A00123, A00124 | No | No | No | Client the member has a policy through |
| sum\_claims\_ct | Gender | CHAR(1) | M or F | No | No | No | Member Gender |
| sum\_claims\_ct | AgeRange | VARCHAR2(20) | All | No | No | No | Age Bucket for Member |
| sum\_claims\_ct | MemberRegion | VARCHAR2(20) | All | No | No | No | Reporting Region for Member Location |
| sum\_claims\_ct | DIABETES | INT | 0,1 | No | No | No | 1 if Members has Diabetes, 0 else |
| sum\_claims\_ct | CAD | INT | 0,1 | No | No | No | 1 if Members has Coronary Artery Disease, 0 else |
| sum\_claims\_ct | CHF | INT | 0,1 | No | No | No | 1 if Members has Congestive Heart Failure, 0 else |
| sum\_claims\_ct | HYPERTENSION | INT | 0,1 | No | No | No | 1 if Members has Hypertension, 0 else |
| sum\_claims\_ct | COPD | INT | 0,1 | No | No | No | 1 if Members has Chronic Obstructive Pulmonary Disease, 0 else |
| sum\_claims\_ct | ProviderID | INT | All | No | No | Yes | Provider ID who performed Service |
| sum\_claims\_ct | ProviderRegion | VARCHAR2(20) | All | No | No | No | Reporting Region for Provider Location |
| sum\_claims\_ct | SpecRup\_DESC | VARCHAR2(100) | All | No | No | No | Provider Specialty Rollup Description |
| sum\_claims\_ct | RPT\_CD | VARCHAR2(10) | All | No | No | Yes | Reporting Category Code |
| sum\_claims\_ct | LVL\_01 | VARCHAR2(10) | All | No | No | No | Reporting Category level 01 Description |
| sum\_claims\_ct | LVL\_02 | VARCHAR2(100) | All | No | No | No | Reporting Category level 02 Description |
| sum\_claims\_ct | LVL\_03 | VARCHAR2(100) | All | No | No | No | Reporting Category level 03 Description |
| sum\_claims\_ct | pos\_Desc | VARCHAR2(100) | All | No | No | No | Place of Service Description |
| sum\_claims\_ct | IncMonth | DATE | >= '01-JAN-2015' | No | No | Yes | Incurred Month of Service |
| sum\_claims\_ct | CoMorbity\_Count | NUMBER(10,0) | >=0 | No | No | No | Count of Member Conditions |
| sum\_claims\_ct | Cases | NUMBER(12,0) | All | No | No | No | Number of cases performed |
| sum\_claims\_ct | Services | NUMBER(12,0) | All | No | No | No | Number of Services performed |
| sum\_claims\_ct | ClaimCount | NUMBER(12,0) | All | No | No | No | Number of Claims |
| sum\_claims\_ct | AllowedCharges | NUMBER(14,2) | All | No | No | No | Amount Provider Charged for Service |
| sum\_claims\_ct | PaidClaims | NUMBER(14,2) | All | No | No | No | Amount Insurance Company Paid for Service |
| sum\_claims\_ct | MemberPaid | NUMBER(14,2) | All | No | No | No | Amount Member Paid for Service |
| sum\_claims\_ct | OPL | NUMBER(14,2) | All | No | No | No | Amount Covered by Other Insurance Carrier |
| summary\_rpt\_01 | IDNO | INT | >=0 | No | Yes | No | Surrogate key of comorbity & IncMonth |
| summary\_rpt\_01 | CoMorbity\_Count | NUMBER(10,0) | >=0 | No | No | No | Count of Member Conditions |
| summary\_rpt\_01 | IncMonth | DATE | >= '01-JAN-2015' | No | No | Yes | Incurred Month of Service |
| summary\_rpt\_01 | Cases | NUMBER(12,0) | All | No | No | No | Number of cases performed |
| summary\_rpt\_01 | Services | NUMBER(12,0) | All | No | No | No | Number of Services performed |
| summary\_rpt\_01 | ClaimCount | NUMBER(12,0) | All | No | No | No | Number of Claims |
| summary\_rpt\_01 | AllowedCharges | NUMBER(14,2) | All | No | No | No | Amount Provider Charged for Service |
| summary\_rpt\_01 | PaidClaims | NUMBER(14,2) | All | No | No | No | Amount Insurance Company Paid for Service |
| summary\_rpt\_01 | MemberPaid | NUMBER(14,2) | All | No | No | No | Amount Member Paid for Service |
| summary\_rpt\_01 | OPL | NUMBER(14,2) | All | No | No | No | Amount Covered by Other Insurance Carrier |
| summary\_rpt\_01 | Member\_Count | NUMBER(10,0) | All | No | No | No | Count of Members with coverage |
| summary\_rpt\_02 | IDNO | INT | >=0 | No | Yes | No | Surrogate key of LVL\_01, LVL\_02, & IncMonth |
| summary\_rpt\_02 | LVL\_01 | VARCHAR2(10) | All | No | No | No | Reporting Category level 01 Description |
| summary\_rpt\_02 | LVL\_02 | VARCHAR2(100) | All | No | No | No | Reporting Category level 02 Description |
| summary\_rpt\_02 | IncMonth | DATE | >= '01-JAN-2015' | No | No | Yes | Incurred Month of Service |
| summary\_rpt\_02 | Cases | NUMBER(12,0) | All | No | No | No | Number of cases performed |
| summary\_rpt\_02 | Services | NUMBER(12,0) | All | No | No | No | Number of Services performed |
| summary\_rpt\_02 | ClaimCount | NUMBER(12,0) | All | No | No | No | Number of Claims |
| summary\_rpt\_02 | AllowedCharges | NUMBER(14,2) | All | No | No | No | Amount Provider Charged for Service |
| summary\_rpt\_02 | PaidClaims | NUMBER(14,2) | All | No | No | No | Amount Insurance Company Paid for Service |
| summary\_rpt\_02 | MemberPaid | NUMBER(14,2) | All | No | No | No | Amount Member Paid for Service |
| summary\_rpt\_02 | OPL | NUMBER(14,2) | All | No | No | No | Amount Covered by Other Insurance Carrier |
| summary\_rpt\_02 | Member\_Count | NUMBER(10,0) | All | No | No | No | Count of Members with coverage |
| HR\_Mem\_Outreach | IDNO | INT | All | No | Yes | No | Surrogate key to maintain uniqueness on Member Outreach table |
| HR\_Mem\_Outreach | MemberID | INT | All | No | No | Yes | Member ID who received Service |
| HR\_Mem\_Outreach | Gender | CHAR(1) | M or F | No | No | No | Member Gender |
| HR\_Mem\_Outreach | DateOfBirth | DATE | >= '01-JAN-1900' | No | No | No | Membe Date of Birth |
| HR\_Mem\_Outreach | Age | INT | >= 0 | No | No | No | Age calculated using CALC\_Age Function |
| HR\_Mem\_Outreach | FirstName | VARCHAR2(100) | All | No | No | No | Member First Name |
| HR\_Mem\_Outreach | LastName | VARCHAR2(100) | All | No | No | No | Member Last Name |
| HR\_Mem\_Outreach | address\_street | VARCHAR2(255) | All | No | No | No | Primary Member Street Address |
| HR\_Mem\_Outreach | address\_city | VARCHAR2(100) | All | No | No | No | Primary Member City |
| HR\_Mem\_Outreach | address\_county | VARCHAR2(50) | All | No | No | No | Primary Member County |
| HR\_Mem\_Outreach | address\_state | CHAR(2) | All | No | No | No | Primary Member State Abbreviation |
| HR\_Mem\_Outreach | address\_zip | CHAR(5) | Between 00001 and 99999 | No | No | No | Primary Member Zip Code |
| HR\_Mem\_Outreach | Comorbid\_Curr | INT | >=0 | No | No | No | # Comorbidites calculated using CALC\_Comorbid\_Current function |
| HR\_Mem\_Outreach | Covid\_Vax\_Curr | INT | >=0 | No | No | No | # Covid Vaccines calculated using get\_VaxCovid\_Current function |
| HR\_Mem\_Outreach | Flu\_Vax\_Curr | INT | >=0 | No | No | No | # Flu Vaccines calculated using get\_VaxFlu\_Current function |
| HR\_Mem\_Outreach | PC\_Visit\_Curr | INT | >=0 | No | No | No | # Primary Care Preventative health visits, calculated using get\_PC\_Current function |
| HR\_Mem\_Outreach | High\_Dollar | CHAR(1) | Y or N | No | No | No | Flag that signifies if a member has had a claim over 25,000, based on a trigger |
| HR\_Mem\_Outreach | High\_Risk\_Codes | INT | >=0 | No | No | No | # High risk codes that the member has had claims for based on codes definied in trigger |
| PROVIDER\_CLAIM\_REPORT | PROVIDERID | NUMBER(38,0) | >= 1 | No | Yes | Yes | PK and FK for ProviderID, connects back to the dim\_provider table |
| PROVIDER\_CLAIM\_REPORT | PROVIDERNAME | VARCHAR2(100) | All | Yes | No | No | Name of the provider |
| PROVIDER\_CLAIM\_REPORT | PROVIDERSPECIALTY | VARCHAR2(100) | All | Yes | No | No | Specialty description of provider |
| PROVIDER\_CLAIM\_REPORT | NUM\_CLAIMS | NUMBER(14, 2) | >= 0 | Yes | No | No | Total number of distinct claims for each provider |
| PROVIDER\_CLAIM\_REPORT | SUM\_ALLOWEDCHARGED | NUBMER(14, 2) | >= 0 | Yes | No | No | Total allowed charges across all claims for each provider |
| REBATE\_MAILER | IDNO | INT | All | No | Yes | No | Surrogate key to maintain uniquesness |
| REBATE\_MAILER | MEMBERID | INT | All | No | No | Yes | Member ID who received Service |
| REBATE\_MAILER | FirstName | VARCHAR2(100) | All | No | No | No | Member First Name |
| REBATE\_MAILER | LastName | VARCHAR2(100) | All | No | No | No | Member Last Name |
| REBATE\_MAILER | LastName | VARCHAR2(100) | All | No | No | No | Member Last Name |
| REBATE\_MAILER | Address\_Street | VARCHAR2(255) | All | No | No | No | Primary Member Street Address |
| REBATE\_MAILER | address\_city | VARCHAR2(100) | All | No | No | No | Primary Member City |
| REBATE\_MAILER | address\_state | CHAR(2) | All | No | No | No | Primary Member State Abbreviation |
| REBATE\_MAILER | address\_zip | CHAR(5) | Between 00001 and 99999 | No | No | No | Primary Member Zip Code |
| REBATE\_MAILER | reason | VARCHAR(255) | All | No | No | No | Describes the reason for the rebate |
| REBATE\_MAILER | mailed\_on | DATE | All | Yes | No | No | Date the rebate was mailed |