Healthcare Transparency

Group 5

Rodrigo Chen (016371706), Michelle Fang (014019252), Natalie Leung (014790932)

Computer Engineering Department

San José State University

May 1, 2024

Table of Contents

Introduction

The dataset used in our project is “cms_synthetic_patient_data_omop.” Our application in using this dataset is to help patients and regular people understand a particular procedure in the medical field, which is often not understandable to those without any education. Thus, this application can be used to be transparent when understanding different medical procedures or their own. This application is interesting because it helps bridge regular people to the medical profession. The application will not replace your doctor but give you a general understanding before entering your appointment. The application's primary audiences are the patients, as it provides a sense of their procedures. The patients can use our application to search for their procedure; for example, they can search for lung cancer, and there will be a data table with information like the cost paid, what provider is needed, etc. This application would have to consider thousands of data and simplify medical terms for the rest to understand. With thousands of data to accommodate, there is a need for latency and response to our application. Thus, understanding how to optimize our queries is highly recommended for a fast response time and to inform the audience. The algorithms we choose would mostly be joining algorithms as we combine multiple tables to show a result. Due to various joins, the run time may be slow; thus using algorithms would be helpful.

Related Work

Our application has common goals with other work/applications: patient empowerment, accessibility, and continuous learning and improvement. Our app aims to empower patients with transparent and understandable information about medical procedures. Our application and others in the field seek to make complex medical information more accessible to a broader audience. By optimizing our medical application, stay updated on medical research, technology,

and user feedback. This will ensure your app provides users with the most relevant and helpful information.

WebMD is a highly trusted and reliable online platform that provides a wealth of resources for individuals seeking information on medical conditions, treatments, medications, and procedures. The platform offers many comprehensive articles written and reviewed by healthcare professionals, covering various topics from common ailments to complex medical issues. Users can find information about symptoms, diagnosis, treatment options, and preventive measures for different health concerns. Overall, WebMD is an excellent resource for anyone looking to stay informed about their health and take control of their well-being. Finding and understanding different medical terms and diagnoses is similar to our objective and application.

Methods Description

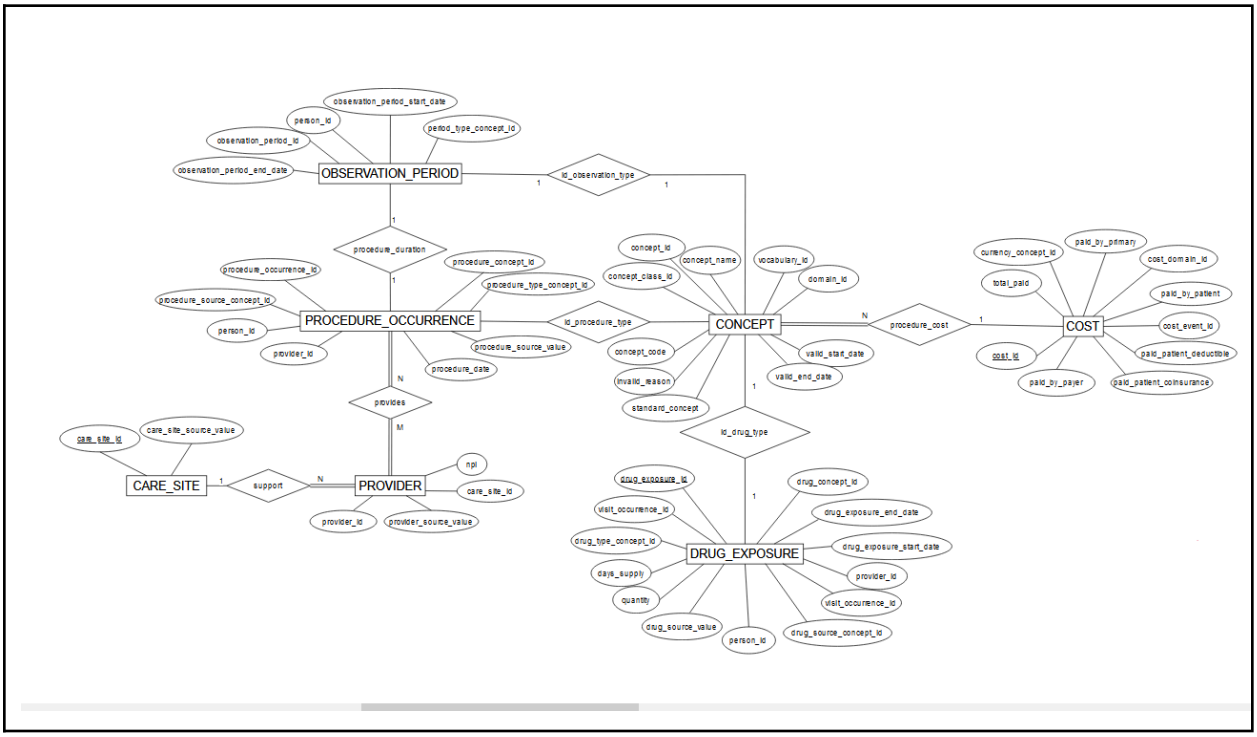

Fig 1: ER Diagram

Once you select a specific procedure, you will be provided with a plethora of information based on that procedure alone. Our application will display different and valuable information about each procedure. There will be no need to worry about privacy issues as there is no need to sign up for anything. Our application will be an information hub that holds information in one space rather than searching other websites and applications. Our queries will be displayed in tables with a small description of each table for a more accessible interface. We want to focus on providing healthcare transparency. So, we want to let the user know procedure-specific information about the care sites and providers that support them, costs, drugs used, how long one will use a procedure drug, and how long the observation time will take. To narrow down the queries we have to make, we have focused on the scenario where our application looks up one procedure, chemotherapy, and reports data based on that procedure.

The first SQL query involved reporting the five top and bottom care sites that offer chemotherapy based on the number of patients. This will give insight into popular care sites a user may visit. The following SQL query is the top 20 providers that provide chemotherapy. This is important when looking into getting a provider to cover chemotherapy costs. Following that, we have a query on the top 10 chemotherapy procedures to allow users to understand possible procedures they may go under when getting chemotherapy treatment. Adding on to this, we have a query that reports the maximum and minimum percentage of how much the patient and provider cover the procedure. This will give users a better idea of how much they will have to pay out of pocket and how much will be covered by providers, which can be used to compare the providers' offers they are getting. We also have the top 10 drugs used in chemotherapy queries so users can know what drugs will be used and be alerted if they are allergic to any commonly used drugs. We note that the 1st most popular drug received from this query was not named. This is

because the dataset we used has a lot of null data. Therefore, we did our last query's 2nd most popular chemotherapy drug exposure and observation time. This information will be helpful for the user to understand better how long they will be using the drug and how long the procedure will take.

We created a github repository that holds the queries we made as well as the query results. The link is listed below.

https://github.com/Rodarkhen/cmpe138_project.git

Analysis of Algorithms

The most critical queries in our application are the top 10 chemotherapy procedure treatments, max and min percentage costs covered by the payer and provider, and 2nd most popular drug's use time and observation time.

As our focus is on letting the user know what to expect from having to do a particular procedure, we wanted our queries to relay information that targets the common concerns regarding health treatments. We narrowed down that people would often have questions about types of procedure treatments, costs, drug type and dependence time, and observation time. Therefore, our most important queries were made based on these factors, Specifically the query on the top 10 chemotherapy procedure treatments would provide a little description/label on popular chemotherapy procedure treatment to better inform what a person they may undergo. The next query provides information about the maximum and minimum percentage cost covered by the patient and provider. This will be for the feature relaying information about procedure costs and setting expectations for the user to understand how much they will have to pay. This

will in return allow them to prepare their financial situation and also give a comparison value when searching for providers. The last important query is about the 2nd most popular drug's use time and observation time based on the drug. We used the 2nd most popular drug for reference because the most popular drug was unnamed. This will take care of the feature for reporting how long a patient will use a drug and how long their observation time is based on the drug.

In our queries, we are using multiple inputs to get the desired outputs. We decided on referencing mainly seven tables of the dataset because most of the tables had null values. These null values hindered which tables we could use because of the lack of table relationships and data it could provide.

The queries we have are listed in the appendix. Appendix 1 has the inputs of: provider_id from the procedure_occurence and provider table, procedure_concept_id from the procedure_occurence table. The outputs are: care_site_id from the provider table. Appendix 2 has the inputs of: provider_id from the procedure_occurence and provider table, procedure_concept_id from the procedure_occurence table. The outputs are: npi from provider table and count of person_id from procedure_occurrence table. Appendix 3 has the inputs of: procedure_concept_id from the procedure_occurence table, concept_id from concept table, and concept_name from concept table. The outputs are: concept_id from concept table, concept_name from concept table, vocabulary_id from concept table, and count of person_id from concept table. Appendix 4 has the inputs of: visit_occurrence_id from procedure_occurrence table, visit_occurrence_id from drug_exposure table, drug_concept_id from drug_exposure table, concept_id from concept table and procedure_concpet_id from procedure_ocurrence_id from procedure_occurrence table. The outputs are: cocnept_name from concept_table and count of drug_exposure_id from drug_exposure table. Appendix 5 has the

inputs of: paid_by_patient, paid_by_payer, and total_paid that are all from the cost table. The outputs are the percentages as a result of the division of the values paid_by_patient/total_paid and paid_by_payer/total_paid. Appendix 6 has the inputs of: concept_id from concept table, drug_concept_id from drug_concept table, concept_name from concept table, and person_id from observation_period and drug_exposure table. The outputs are: the round average of the data difference between the drug_exposure_end_date and drug_exposure_start_date form the drug_exposure table and the round average of the data difference between the observation_period_end_date and observation_period_start_date form the observation_period table.

We can optimize all the queries by including nested queries, doing a union join rather than nested queries, and changing variable names. We did not use indexes and instead focused on these methods mentioned in class. Our query processing system uses Joining and Query Optimization algorithms, allowing faster computations and better performance. Joining algorithms are chosen for their nested loops, block nested loops, sort-merge, and hash joins capabilities, while Query Optimization Algorithms are used for Cost-Based Optimization. We continuously optimize our algorithms to provide the most efficient query-processing system possible.

IO COST		
Queries	Non Optimized ($)	Optimize ($)
Appendix 1	2.36E-08	3.63E-08
Appendix 2	3.63E-08	3.63E-08
Appendix 3	2.59E-08	2.59E-08
Appendix 4	3.73E-08	3.73E-08
Appendix 5	9.26E-08	9.26E-08
Appendix 6	2.89E-08	2.36E-08

Table 1: IO cost of our queries

We can compute these IO costs using the equations found online. For example, one calculation for the IO cost of the first query listed in the appendix was done below:

To find our total data processed, we ran each query and found the total bytes processed. In the non-optimize, we got 4.06 GB processed, and for optimized, it is 6.24 GB. Another piece of information is that Big Query bills a rate of $6.25/TB.

I/O Cost = Total Data Processed (TB) × Cost per TB of Data Processed

Convert GB to TB:

$$\text{Total Data Processed}_{\text{non-opt}} = \frac{4.06}{(1024^3)} = 3.78 \times 10^{-9} \text{ TB}$$

$$\text{Total Data Processed}_{\text{opt}} = \frac{6.24}{(1024^3)} = 5.81 \times 10^{-9} \text{ TB}$$

$$\text{I/O Cost}_{\text{non-opt}} = (3.78 \times 10^{-9}) \times 6.25 = \$2.36 \times 10^{-8}$$

$$\text{I/O Cost}_{\text{opt}} = (5.81 \times 10^{-9}) \times 6.25 = \$3.63 \times 10^{-8}$$

The results are all in cents because we are processing at GB instead of the usual TB.

Results and Findings

We evaluated our solution based on the outputs from the queries we came up with, and we analyzed them to determine if they appeared to be appropriate and fictiously correct from what we initially expected to have as output. To evaluate the performance and efficiency of our queries, we referenced the execution details from the BigQuery. It shows the processing details, such as elapsed time, slot time consumed, and the computation time of various stages, including

the input, joins, and aggregate functions, depending on how the queries are written. This information helps us determine if the query is acceptable and if the optimized equivalent queries are better and more efficient. We included the charts showing all the processing details for the queries in the figures below.

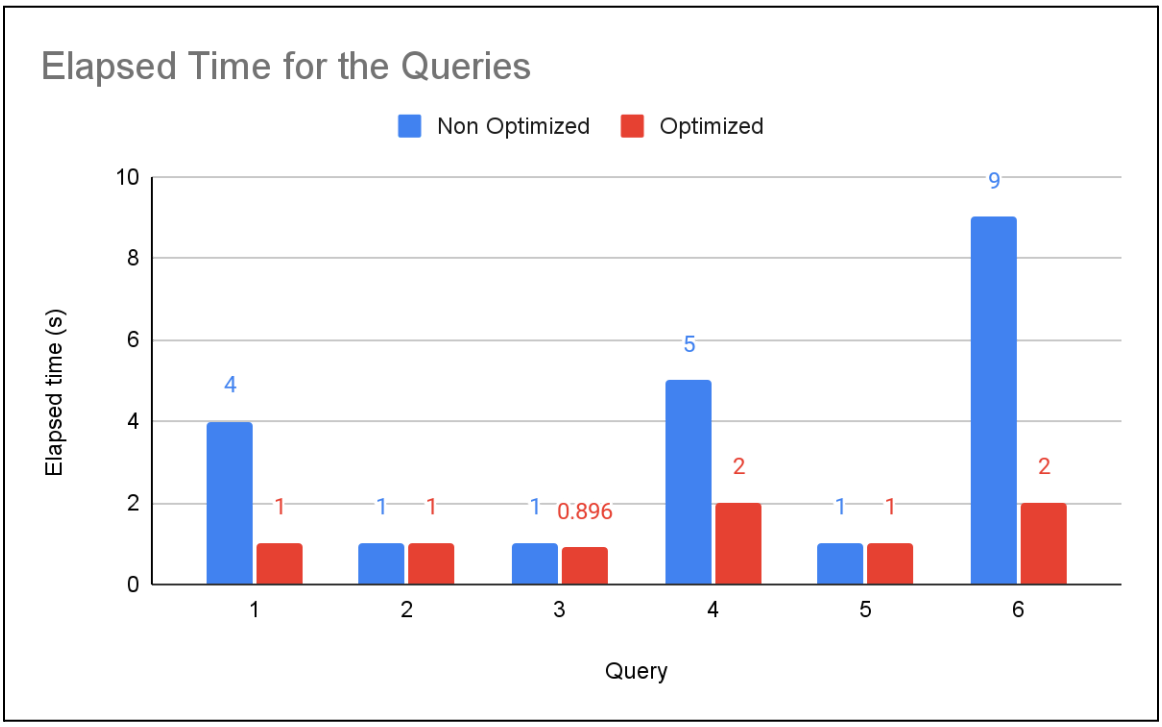

Fig. 2. Elapsed Time Before and After Optimization

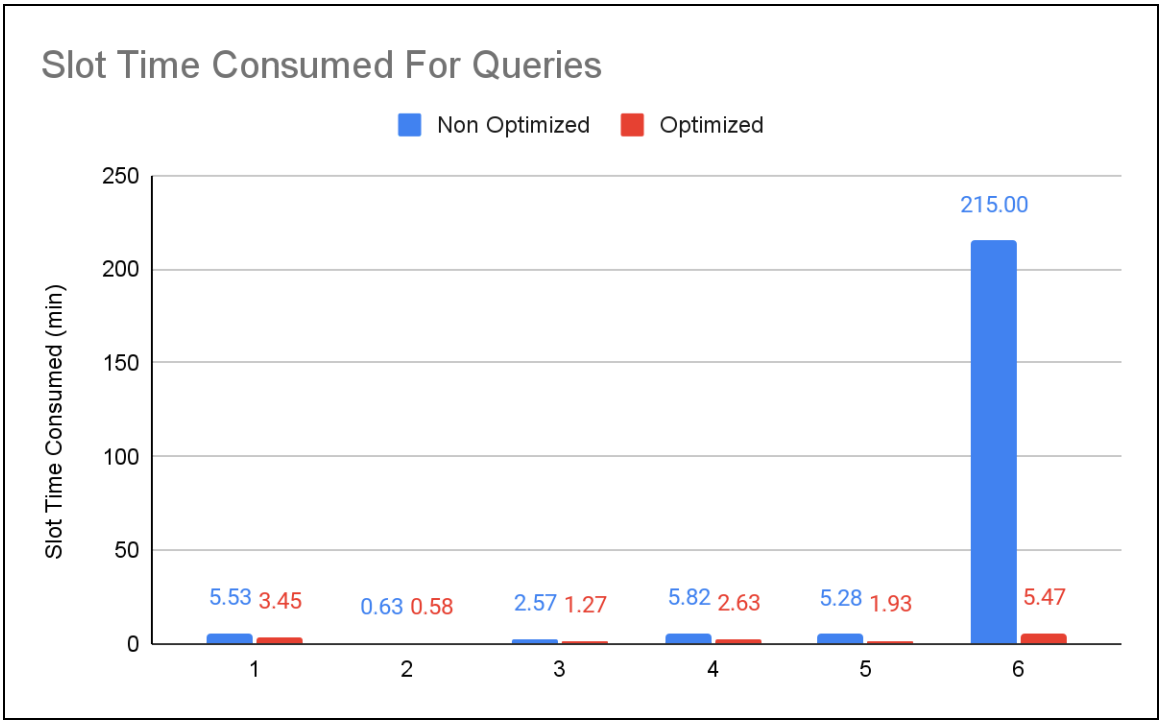

Fig. 3. Slot Time Consumption Before and After Optimization

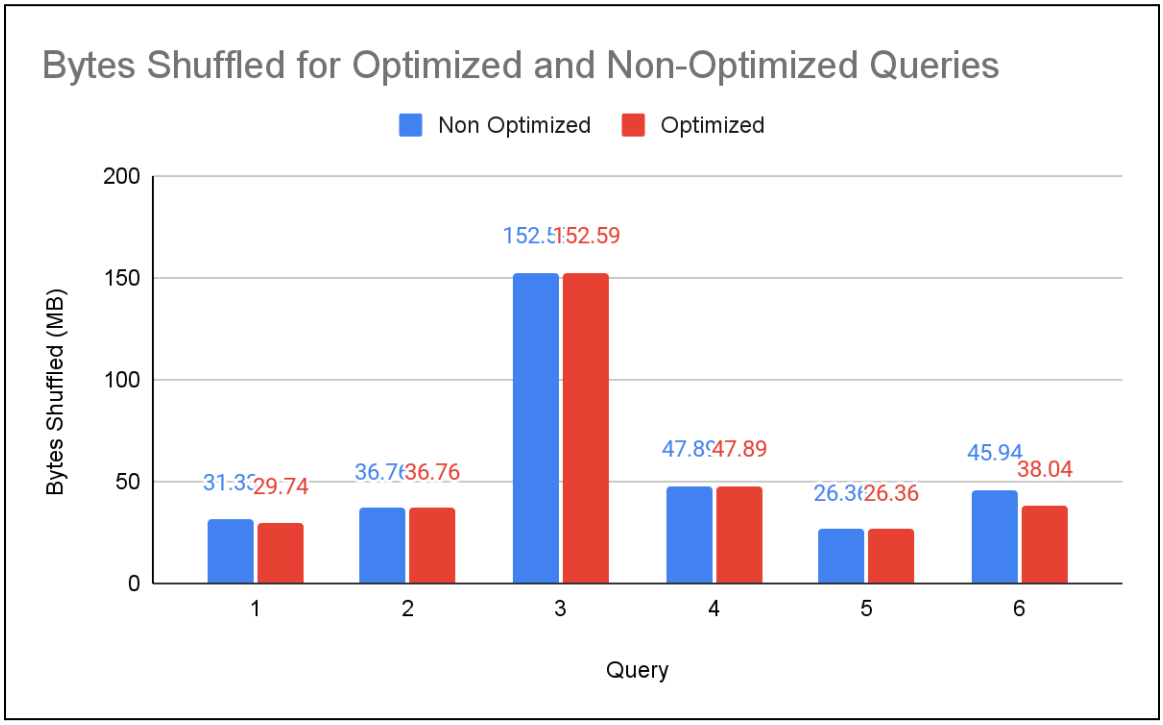

Fig. 4. Bytes Shuffled Before and After Optimization

The charts in Fig. 2 and 3 show that all queries we implemented had a general improvement with the optimized versions in terms of the elapsed time and the slot time consumed. Noticeably, query 6 had the most significant improvement, changing the elapsed time from 9 seconds to 2 seconds or the slot time consumed from 215 minutes to 5 minutes. This represents a decrease of approximately 97%. Similarly, other queries had improvements as well. For example, query 1 had a percentage decrease of 5%. Interestingly, this timing amelioration also came with less byte shuffled, which indicates that we had better efficient data movement and cost savings. The primary cause of the extensive time was the use of JOINs, so one of the suggestions to address it is to evaluate and optimize the JOIN operations carefully within the queries. We can employ some potential improving strategies for the queries, such as choosing the right type of JOIN based on the data and the application requirements, changing the execution order of JOIN operations, or using filtering clauses before performing the JOIN operation. It might result in a more efficient join sequence and reduce the number of rows processed during a join, consequently improving the query performance.

Conclusions and Lessons Learned

We have created 6 queries from our vast dataset, BigQuery, which focuses on the medical field. However, since our data set is synthetic, which means it has "fake" values, some of our results may not make sense due to these discrepancies. Despite this significant challenge, we have learned some valuable lessons throughout the process. For example, we have discovered new ways to optimize our queries and better understood SQL functions such as ROUND(), UNION ALL, and LOWER(). In addition to using the functions we learned in class, we have also invested time outside of class to know and understand how to use different functions in our queries. Overall, our project has been a challenging but rewarding experience. We have better

understood the medical field, data analysis, and query optimization. We look forward to utilizing our newfound knowledge and skills in future projects.

Reference

WebMD. (2023). WebMD - Better information. Better health. WebMD. https://www.webmd.com/

Appendixes

Appendix 1: 5 top and bottom care sites that offer chemotherapy based on the number of patients

```
WITH chemo_sites AS (
    SELECT
        care_site_id,
        COUNT(DISTINCT p.person_id) AS chemo_num_patients
    FROM `bigquery-public-data.cms_synthetic_patient_data_omop.procedure_occurrence` p
    JOIN `bigquery-public-data.cms_synthetic_patient_data_omop.provider` pro
        ON p.provider_id = pro.provider_id
    WHERE p.procedure_concept_id = 4289151
    GROUP BY care_site_id
)
SELECT care_site_id, chemo_num_patients FROM (
    SELECT * FROM chemo_sites
    ORDER BY chemo_num_patients DESC
    LIMIT 5
) AS top_5
UNION ALL
SELECT care_site_id, chemo_num_patients FROM (
    SELECT * FROM chemo_sites
    ORDER BY chemo_num_patients
    LIMIT 5
) AS bottom_5;
```

Appendix 2: Providers that provide chemotherapy

```
SELECT npi,
       COUNT(DISTINCT p.person_id) chemo_cnt_patients
FROM `bigquery-public-data.cms_synthetic_patient_data_omop.procedure_occurrence` p
JOIN `bigquery-public-data.cms_synthetic_patient_data_omop.provider` pro
    ON p.provider_id = pro.provider_id
WHERE p.procedure_concept_id = 4289151
GROUP BY 1
ORDER BY 2 DESC
LIMIT 20
```

Appendix 3: Top 10 chemotherapy procedures

```
SELECT
  concept_id,
  concept_name,
  vocabulary_id,
  COUNT(Distinct person_id) num_patients
FROM `bigquery-public-data.cms_synthetic_patient_data_omop.procedure_occurrence` p
JOIN `bigquery-public-data.cms_synthetic_patient_data_omop.concept` c
  ON p.procedure_concept_id = c.concept_id
WHERE LOWER(concept_name) LIKE '%chemotherapy%'
GROUP BY 1,2,3
ORDER BY 4 DESC
LIMIT 10
```

Appendix 4: Top 10 drugs used in chemotherapy

```
SELECT
    concept_name AS drug,
    COUNT(DISTINCT drug_exposure_id) chemo_num_visits
FROM `bigquery-public-data.cms_synthetic_patient_data_omop.procedure_occurrence` p
JOIN `bigquery-public-data.cms_synthetic_patient_data_omop.drug_exposure` d
    ON p.visit_occurrence_id = d.visit_occurrence_id
JOIN `bigquery-public-data.cms_synthetic_patient_data_omop.concept` c
    ON c.concept_id = d.drug_concept_id
WHERE p.procedure_concept_id = 4289151
GROUP BY 1
ORDER BY 2 DESC
LIMIT 10
```

Appendix 5: Max and min percentage of how much patient and provider covers

```
SELECT
  MAX(percent_patient_covers) AS max_percentage_patient_covers,
  MIN(percent_patient_covers) AS min_percentage_patient_covers,
  MAX(percent_provider_covers) AS max_percentage_provider_covers,
  MIN(percent_provider_covers) AS min_percentage_provider_covers

FROM (
  SELECT
    paid_by_patient/total_paid AS percent_patient_covers,
    paid_by_payer/total_paid AS percent_provider_covers
```

```
  FROM
    `bigquery-public-data.cms_synthetic_patient_data_omop.cost`
  WHERE total_paid != 0.00
)
```

Appendix 6: 2nd most popular chemotherapy drug exposure time and observation time

```
WITH get_drug_id AS (
    SELECT
        concept_name AS drug,
        drug_concept_id as dd
    FROM `bigquery-public-data.cms_synthetic_patient_data_omop.drug_exposure` d
    JOIN `bigquery-public-data.cms_synthetic_patient_data_omop.concept` c
        ON c.concept_id = d.drug_concept_id
    WHERE c.concept_name LIKE '%Sodium Chloride Injectable Solution%'
    LIMIT 1
),

calc_date_diff AS (
    SELECT
    DATE_DIFF(d.drug_exposure_end_date, d.drug_exposure_start_date, day)AS
drug_time,
    DATE_DIFF(o.observation_period_end_date, o.observation_period_start_date, day)AS
observation_time
    FROM `bigquery-public-data.cms_synthetic_patient_data_omop.drug_exposure`d
    JOIN get_drug_id g
        ON d.drug_concept_id = g.dd
    JOIN `bigquery-public-data.cms_synthetic_patient_data_omop.observation_period` o
        ON o.person_id = d.person_id
)

SELECT
    ROUND(AVG(drug_time),2) avg_drug_exposure_time_in_days,
    ROUND(AVG(observation_time),2) avg_observation_time_in_days
FROM calc_date_diff
```