NEW YORK UNIVERSITY

CSCI-GA.2433-001 DATABASE SYSTEMS FINAL PROJECT

Furkan Ozyurt

Introduction

Today, it is not unusual for a company to have

- very large amount of data that becomes hard to manage,
- flawed data sources
- inaccurate data that needs to be fixed/corrected
- messy data that needs to be cleaned/processed properly.

That's why companies prefer to follow a framework named Enterprise Data Architecture (EDA) to deal with these issues.

Enterprise Data Architecture is a framework that is used to manage, integrate, and govern the data within an organization. Collection, storage, management and usage of data is handled with this framework. The goal with the EDA is to ensure that the data is available, accurate, and secure.

There are some key components in EDA.

1) **Enterprise Data Model:** This is a model that is used to represent how data is organized and related across the entire organization.
    a. **Conceptual Data Model:** This can be seen as a very high-level view of the data within the enterprise. It shows the main data entities and relationships. Entity relationship (ER) diagram and enhanced entity relationship (EER) diagram can be given as examples of conceptual data model. There is also an UML diagram, but UML diagram differs from ER diagram/EER diagram. UML is more suitable to use for object-oriented systems. Unlike ER/EER diagram, it can show methods associated with classes.

       Things that are offered by UML diagram and not by ER/EER diagram:
       Things that are offered by ER/EER diagram and not by UML diagram:

    b. **Logical Data Model:** This can be seen as more detailed representation of the data within the enterprise compared to conceptual data model. The examples of the logical data model can be:
        i. **Relational Model:** It represents the data as tables, primary keys, attributes, foreign keys, relationships between different tables, etc.
        ii. **Object Oriented Data Model:** It represents the data as objects. These objects have attributes and methods.
        iii. **Dimensional Model:** This is another way to represent the data. In this model, two types of relations (tables) are used: fact tables and dimension tables. Fact table contains the quantitative and main data. And it contains foreign keys to dimension tables. In the dimension tables, descriptive attributes that provide additional information about the information stored in the fact table are stored. Through the foreign keys that are stored in fact table, we can access these descriptive attributes in the dimension tables. There are two different ways to organize the fact table and dimension table. For instance, one fact table can be surrounded by many dimension tables. This is called star schema. If we divide the dimension tables into multiple separate tables, this is called snowflake schema. Dimensional model provides faster query performance but it needs to

store larger amount of data in exchange for that.

   c. **Physical Data Model:** Technical implementation of the logical model.

2) **Data Integration:** As can be understood from its name, data integration is basically integrating data from different sources. The goal is to provide a single and unified view of the organization's data and improve the accessibility, quality, and usability of the data by creating data warehouses/data lakes/data marts, merging different databases, and combining data across different departments.
   a. **ETL (Extract – Transform – Load Process):** Extracting the data, transforming/cleaning it, and moving it between systems.
   b. **Data Pipelines:** Automated and scheduled workflows of data processing. The difference between ETL and data pipeline is …
   c. **APIs:** Interfaces that are used to exchange data between different applications/programs and systems.

3) **Data Storage:** The goal is to store the data in a suitable way.
   a. **Databases:** The database can be relational or non-relational depending on the data we want to store.
      i. Relational Database
      ii. NoSQL Databases
      iii. Object Oriented Database
      iv. Hierarchical Database
   b. **Transactional Data Store:** It is a type of database designed to gather and manage real-time transactions. Each transaction is recorded as it occurs, and it is used for day-to-day operations.
   c. **Operational Data Store:** It is a type of database designed to integrate data from multiple TDS and other sources. This kind of database is used for generating reports and analytics to support daily operations. The difference between transactional data store and operational data store is that transactional data store focuses on capturing transactions as they occur, and it contains the most up-to-date data while operational data store provides a combined view of data that are integrated from different sources for reporting and analytics.
   d. **Data Lakes**: When we gather all structured and unstructured raw data without preprocessing/cleaning in one repository, this is called a data lake. The goal in here is to store a large amount of raw and unprocessed data in its original format so that they can be transformed from here.
   e. **Data Warehouses**: When we gather structured and clean data from various different sources and create a large and centralized repository, we call this repository data warehouse. The goal in here is to preprocess/clean and integrate data from multiple sources to obtain a structured and unified data.
   f. **Data Marts**: And a subset of data warehouse can be defined as data mart. For instance, when we gather all data that is related to auto insurance from the data warehouse, this specialized subset of clean and transformed data can be called data mart. The goal in here is to provide quick access to relevant data for specific departments in the company.

4) **Data Governance**: The goal is to ensure that regulatory requirements and internal policies are met, and the data is accurate, available, secure, and consistent. Responsibilities are assigned to different people to ensure data quality and integrity.

5) **Data Retention Standards:** The value of the data typically changes. And the goal in here is manage this data properly by
    a. consistently moving data between storage tiers as it ages **(uniform migration)**
    b. deleting or archiving data that is no longer needed **(data removal)**
    c. handling older data different than handling new data **(data aging)**
    d. deciding about data retention together with stakeholders **(consensus-driven)**
    e. ensuring that data retention policies align with IT strategies and capabilities **(involvement of IT officers)**

6) **Data Access and Data Security:** The goal is to define who can access the data and who cannot, protect the data (encryption), and to track the data access and modifications/updates that are made to the data to ensure that the data stays consistent, accurate, secure and compliant with regulatory requirements and internal policies in the enterprise.

7) **Data Quality:** The goal is to ensure that the data is accurate, and consistent, clean the data, and continuously monitor the quality of the data.

When we follow the EDA, there are some key points that we should consider. For instance, in an organization, different teams tend to create incompatible systems. To bring all of them into the same table and to ensure consistency between different projects, some kind of template is needed. This template is called reference architecture.

Through reference architecture, best practices in the industry can be incorporated and teams become aware of/follow industry best practices. Reference architecture can include recommended technologies for the projects, and this simplifies the decision-making process. Reference architectures also ensure that all systems meet regulatory requirements and internal policies. In addition, they can be designed in such a way that specific business goals and strategies are met. Also, in the framework reference architectures provide, scalability and adaptability to the future needs is considered and this makes it easier to use in the long-term.

Entity Relationship Diagram

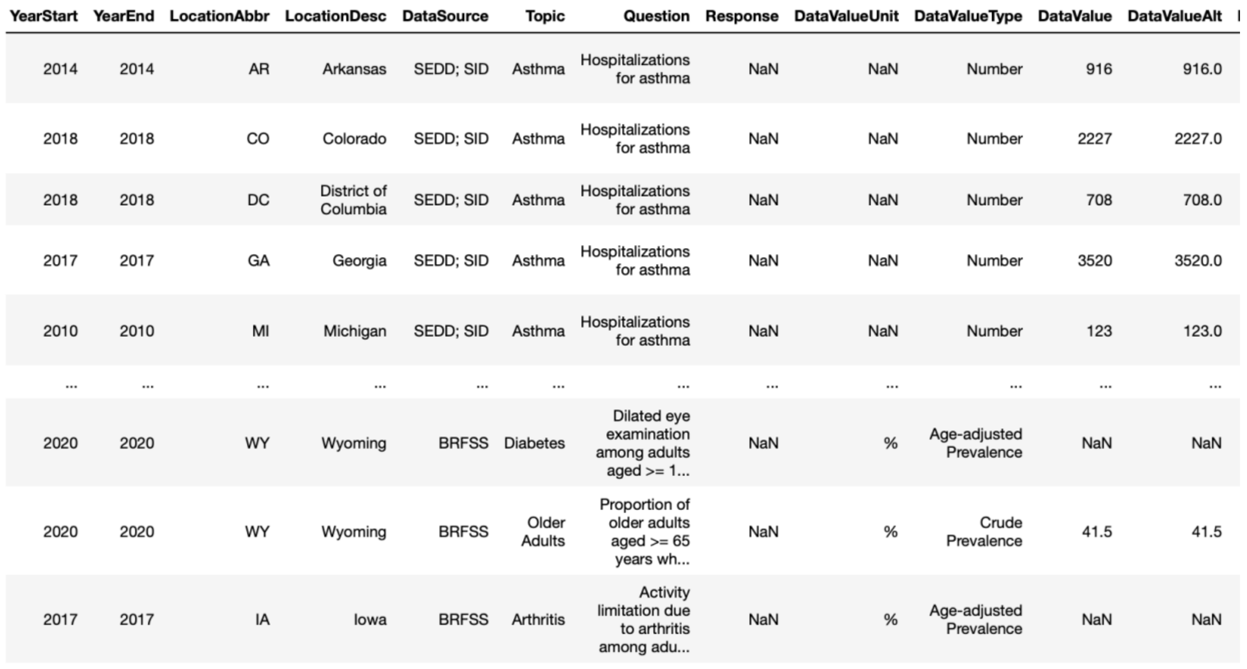
Organizations develop their EDA as a subset of their reference architecture which is broader than EDA. And one of the first steps of developing EDA is data modeling.

Therefore, in the first part of the project, will create an example of conceptual data model for an insurance company.

The entity relationship diagram that is created after following all of these steps is attached as PDF in this folder.

After modeling the insurance company's data with entity relationship diagram, the next step is to integrate different types of data that can help the company to forecast the situation of chronic diseases (e.g., cardiovascular disease, diabetes, tumors, hyperlipemia, obesity, chronic respiratory disease) in the US into the company's database system because this data do not exist within company and collecting them is important to forecast chronic diseases and take necessary actions earlier.

In this part of the project, the data that provides statistics various statistics about the chronic disease across the United States. These statistics are categorized under different sex, race, ethnicity, and age groups. The data is taken from **Chronic Disease Indicators.**



	YearStart	YearEnd	LocationAbbr	LocationDesc	DataSource	Topic	Question	Response	DataValueUnit	DataValueType	DataValue	DataValueAlt
	2014	2014	AR	Arkansas	SEDD; SID	Asthma	Hospitalizations for asthma	NaN	NaN	Number	916	916.0
	2018	2018	CO	Colorado	SEDD; SID	Asthma	Hospitalizations for asthma	NaN	NaN	Number	2227	2227.0
	2018	2018	DC	District of Columbia	SEDD; SID	Asthma	Hospitalizations for asthma	NaN	NaN	Number	708	708.0
	2017	2017	GA	Georgia	SEDD; SID	Asthma	Hospitalizations for asthma	NaN	NaN	Number	3520	3520.0
	2010	2010	MI	Michigan	SEDD; SID	Asthma	Hospitalizations for asthma	NaN	NaN	Number	123	123.0
	...	...	...	...	...	...	...	...	...	...	...	...
	2020	2020	WY	Wyoming	BRFSS	Diabetes	Dilated eye examination among adults aged >= 1...	NaN	%	Age-adjusted Prevalence	NaN	NaN
	2020	2020	WY	Wyoming	BRFSS	Older Adults	Proportion of older adults aged >= 65 years wh...	NaN	%	Crude Prevalence	41.5	41.5
	2017	2017	IA	Iowa	BRFSS	Arthritis	Activity limitation due to arthritis among adu...	NaN	%	Age-adjusted Prevalence	NaN	NaN

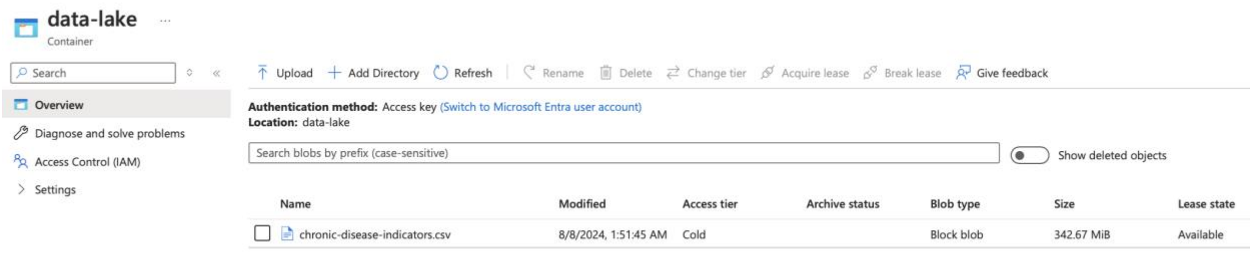
Figure 2.1: Raw Data

Note: *It was difficult to find the most up-to-date data for these categories. The latest date in this data is 2021. That's why, I will assume that we are in 2021 in the whole project.*

Creating Data Lake in Microsoft Azure

In the second part of the project, a data lake is created in Microsoft Azure by following the instructions in this tutorial. As we mentioned in the first part of the project, when we gather all structured and/or unstructured raw data without preprocessing/cleaning in one repository, this is called a data lake. The goal in here is to store a large amount of raw and unprocessed data in its original format so that they can be transformed from here.

Then I put the data I collected into the data lake as can be seen from below.



Authentication method: Access key (Switch to Microsoft Entra user account)
Location: data-lake

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
chronic-disease-indicators.csv	8/8/2024, 1:51:45 AM	Cold		Block blob	342.67 MiB	Available

Figure 2.2: Data Lake

Logical Schema and Normalization

After putting the data into the data lake, the next step was to create a logical schema using the entity relationship diagram that is created in the first part of the project. There are different ways to generate the logical schema. Because of the type of data I use until this point, I will use relational schema.

When creating the logical schema, the external data that I mentioned will be integrated into the schema and then the schema will be optimized and normalized.

In here optimization means improving the performance and efficiency of database schema. Normalization, on the other hand, means organizing the data in such a way that the overall redundancy and dependency are reduced. During normalization, a database is broken down into related tables to minimize duplicated data and to improve the data integrity.

During the normalization process, all the relations in the logical schema are updated so that they can all meet at least the conditions below:

- Each relation will have a primary key.
- All attributes in all relations will be atomic/indivisible.
- There will be no multi-valued attributes.
- There will be no non-key attribute that is dependent on only part of the primary key in any of the relation.
- There will be no non-key attribute that is dependent on to another non-key attribute in any of the relation.

By ensuring that all these conditions are met in the entire database design, we will minimize redundant data and improve the data integrity.

The logical schema that is created after following all these steps is attached as PDF in this folder.

Building PostgreSQL Database

After creating the logical schema, the next step is building a database, and then creating empty tables in that database based on the logical schema that is created in the second part of the project.

In my first attempt, I have created a database in Microsoft Azure. But because my 1-month free trial period ended, I decided to install the database in my local computer using PostgreSQL to avoid costs.

To create the PostgreSQL database, I followed this tutorial. After following the steps in this tutorial, the database and database server are created.

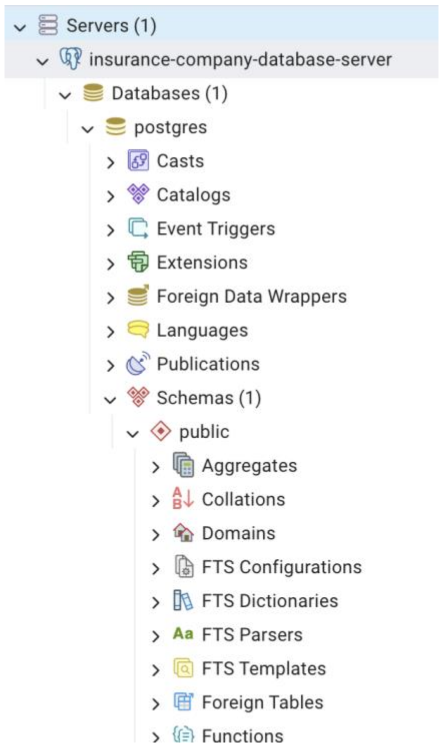


Figure 3.1: PostgreSQL Database

Now, the next step is to translate the logical schema that is created in the second part of the project into physical structure. This is basically called physical design.

During physical design, we can take advantage of some methods to improve query performance, speed up data retrieval and the joining process, make data more manageable, and reduce computation time for frequently executed queries.

Optimizing Physical Database Design

1) Indexing

One of the methods that can be used to improve query performance and speed up data retrieval is indexing. Indexing is like preparing a table of contents for a book with hundreds of pages.

A table of contents prevents us from searching every single page until we find the information we want, and indexing serves the same purpose.

During indexing, a new data structure called an index is created based on the values in a column. It is much smaller compared to the original data set and is organized for fast searching, just like a table of contents. Just as a table of contents helps us jump to the start page of specific sections, an index helps us jump to the exact locations of full data by storing pointers (addresses).

So, to which columns can we apply indexing ?

- **Primary Key**: An index is typically created on the primary key.
- **Foreign Keys**: Foreign keys are used in JOIN operations. Without an index on the columns we want to join, the database will fully scan both of the relations we want to join. This is highly ineffective, which is why indexing can be applied to foreign keys.
- **Columns that are used frequently in WHERE, ORDER BY, GROUP BY, and RANGE operations**:
  - For instance, if we have a query *SELECT * FROM employees WHERE department_id = 5*, applying indexing on department_id allows us to quickly retrieve all records where department_id equals 5 instead of scanning every record one by one.

  - If we have a query *SELECT * FROM employees ORDER BY last_name* and there is an index on last_name, the database can retrieve data in the already sorted order of the index. This avoids the sorting operation, which is expensive.

  - In GROUP BY operations, the data should be sorted as well. Applying an index to columns that are frequently used in GROUP BY operations avoids the sorting operation.

  - A RANGE operation (e.g., BETWEEN, >, <) retrieves a subset of records/rows based on a specified range of values in a column. Columns that are frequently used in RANGE operations benefit from indexing as well because of the similar reasons that are explained previously.

- **Columns with many distinct values**: Columns that have a lot of distinct values benefit from indexing more than columns with very few distinct values because indexes are more beneficial and effective when they can narrow down the search space. When a column has many unique values, they can do this and isolate the specific rows that match the condition specified in the query very quickly. Indexes on columns that has many distinct values provide better and more effective filtering. When a column has small number of unique values and the values are repeated very frequently, the performance over scanning the full table is minimal.
- **Composite columns**: If we frequently query using a specific combination of columns, a composite index on both columns can be beneficial because …

And to which columns can we avoid applying indexing?

- **Columns with very few unique values**: These columns might not benefit from indexing. For instance, if there is a gender column, we expect to see only two distinct values in this column. So, applying indexing to this column does not make sense because it won't be able to narrow down the search space in each step efficiently. That's why, applying indexing on this column would not be very beneficial.
- **Columns in small tables**: Indexing might not be necessary for small tables as the performance gain is minimal.
- **Columns that change frequently**: Each update to a column means an additional update to the index, which can be expensive.
- **Columns with very long values**: Indexing these can be inefficient because the index entries themselves can become large.

One final note is that although indexes are helpful and they speed up data retrieval, they take up additional storage because we have to store the index on the disk as well. Therefore, they can slow down inserting, updating, and deleting operations because when we apply these operations, we have to update the index too.

Too many indexes can cause the query optimizer to make suboptimal choices because it must evaluate the cost of using each index. If there are too many indexes, it becomes harder for the query optimizer to determine which one to use for a given query. This can lead to suboptimal decisions.

Finally, if write operations on the table are expected to be more frequent than read operations, avoiding indexing might be beneficial because indexing can slow down the writing process.

2) Partitioning

Another method we can utilize during physical database design is partitioning. Partitioning is an important technique used to improve database performance, manageability, and availability, especially when the database becomes large. Partitioning is generally considered when tables contain millions or billions of rows of data. It involves dividing a large database into smaller and more manageable parts. These parts are named partitions, and each partition is a subset of the original database but with the same schema of the data from which we create partitions.

There are different ways to partition the data. For instance, we can partition based on a range of values in the partition key. Partitioning customer data by date ranges is an example of this.

We can also partition based on a list of discrete values in the partition key. Partitioning customer data by the city they live in is an example of this.

Additionally, we can use a hash function on the partition key and distribute data across partitions based on this hash function, which is called hash partitioning.

Partitioning is useful for executing queries in parallel. It allows the database to process different partitions simultaneously by using multiple CPU cores or distributed systems. Moreover, when we create a partitioned table, the database maintains metadata about each partition. When a query is made, the database compares the conditions in the query with this metadata. This way, it can determine which partitions may contain relevant data and which partitions can be ignored. This reduces the amount of data scanned and improves query performance.

Partitioning can be applied when:

- We deal with very large data.
- New data is continuously and frequently inserted into the database (especially if this is time-series data).
- The database supports parallel query execution across partitions.
- We need to manage different parts of the data independently without affecting processes in other parts.

And partitioning may be avoided when:

- We have a small data (less than a million rows for instance)
- We will use queries that join a partitioned table with other tables because
  - The database may need to access multiple partitions and combine these partitions with the other tables. This causes increased number of IO operations. Also, more data is transferred across the network during this process.
- We don’t have the system that supports parallel query execution.
- We will use queries that access data across multiple partitions because partitioning works best when queries are specific to specific partitions. If this is not the case, we partition elimination is not made and we lose the benefit of partitioning.

- The distribution of our data changes frequently because if the data is moved frequently between different partitions, this means increased IO and CPU usage and this can reduce the benefit of partitioning.

3) Clustering

Aside from indexing and partitioning, we can also use clustering during the physical database design. Clustering is the process of grouping records on the disk physically. The records that are frequently accessed together or records that have similar values in specific columns are stored close to each other in the disk. This is achieved by organizing the data based on a key. This key is usually an index or a set of columns that determine the physical order of the data. There are two types of clustering: index clustering and table clustering.

In index clustering, the data in the table is physically ordered based on the specified index. For instance, if we have a customer table, clustering this table based on customer ID groups all orders for each customer together.

In table clustering, multiple tables that are related are stored in the same physical space together. This is useful for tables to which JOIN operations are applied frequently. For instance, if we have Customer and Invoice tables and if we expect to join these two frequently, we can store them together.

Clustering reduces the number of IO operations for queries that access data based on the clustering key. When related tables are clustered together, joining them becomes more efficient. Also, the performance of queries that retrieve records that are related can be increased with clustering. It is the most effective for columns or tables frequently used in queries, especially in WHERE or JOIN operations. It can impact insert and update performance, as maintaining the clustered order may require data reorganization.

While clustering improves read performance for specific query patterns, it may slow down write operations (inserts, updates, deletes) due to the need to maintain the physical order. It may not be beneficial for tables with frequent, random inserts or updates.

In summary, clustering is most beneficial for:

- Tables with many rows.
- Queries that return a range of rows.
- Columns used frequently in WHERE clauses and JOIN conditions.

It's less useful for tables with frequent random inserts or updates, as maintaining the clustered order can be costly in these scenarios.

4) Selective Materialization

Selective materialization is an optimization method of creating and storing pre-computed results for certain queries or parts of queries. The goal is to speed up query execution by avoiding to compute the same results from the raw data repeatedly. It is especially useful for complex queries that run frequently and/or that involve time consuming operations such as JOINS.

In selective materialization, all possible views are not materialized. Which views to be materialized is selected based on different factors such as query frequency, computation cost, available storage, etc.

Although it speeds up the retrieving the results of a query, the disadvantage of this method is that this method requires extra storage to store the materialized view of the queries.

Also, materialized views can be updated automatically when the underlying data changes. And it is different from regular view which is basically just a saved query. Materialized query, on the other hand, stores the results of the query as well. It is like creating a new table and storing the results of the query in that table.

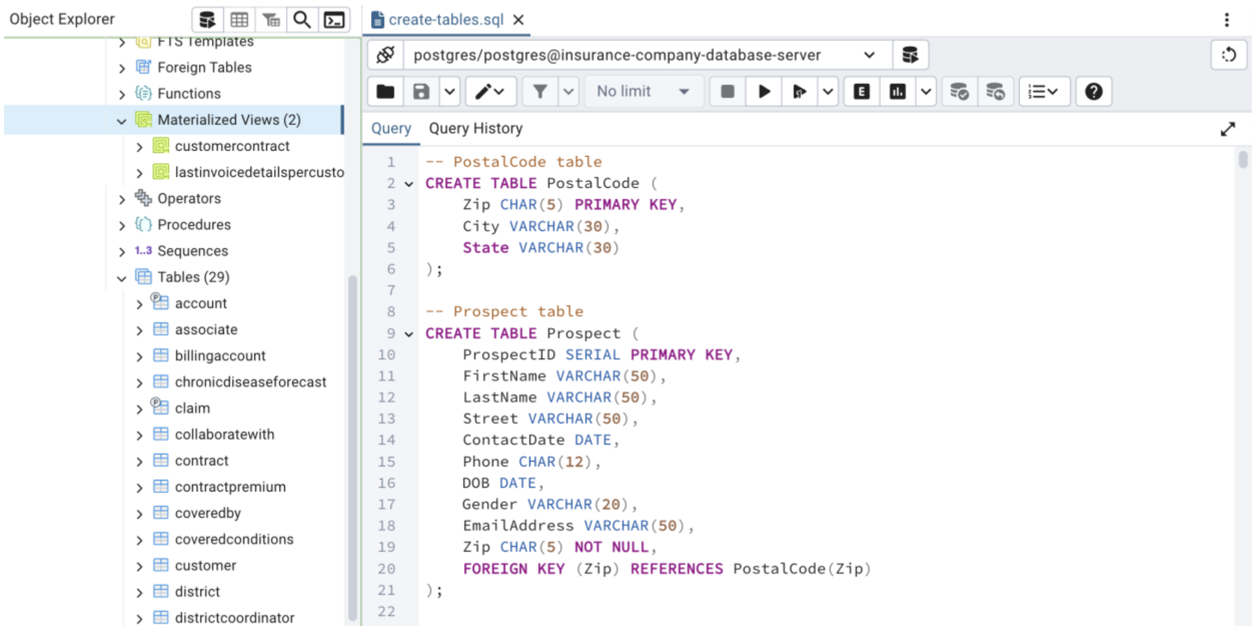
Physical Design

If we return back to our case, to optimize the tables, I applied partitioning to the tables that are expected to be large and continuously inserted with new data, such as **Customer**, **ParticipantClaimant**, **Account**, and **InvoiceDetail**. If the column that I used to partition a table was not part of the primary key, I integrated that column into primary key, implemented index on that column.

Indexes are automatically created for primary keys, so I didn’t need to manually create indexes on primary keys. However, I applied indexes to the foreign keys to speed up the joining process, as well as to columns that I expect to be frequently used with WHERE operations.

The SQL commands I used to create and physically optimize the tables can be found in the **create-tables.sql** file in this folder.

After running the commands in **create-tables.sql**, the tables are initialized and they can be seen in Figure 3.2.



```
-- PostalCode table
CREATE TABLE PostalCode (
    Zip CHAR(5) PRIMARY KEY,
    City VARCHAR(30),
    State VARCHAR(30)
);

-- Prospect table
CREATE TABLE Prospect (
    ProspectID SERIAL PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Street VARCHAR(50),
    ContactDate DATE,
    Phone CHAR(12),
    DOB DATE,
    Gender VARCHAR(20),
    EmailAddress VARCHAR(50),
    Zip CHAR(5) NOT NULL,
    FOREIGN KEY (Zip) REFERENCES PostalCode(Zip)
);
```

Figure 3.2: Initialized Tables in the Database

After creating and optimizing the tables, the next step is to insert all the data into these empty tables in the database.

Data Creation

Because there was no data, I used Mockaroo and Python scripts to generate realistic-looking synthetic data that is suitable for the format of the tables in the database.

By using the codes in **create-data.py** file, I created synthetic data for **Policy, CoveredConditions, InNetworkProviders** tables. The **create-data.py** file can be seen in this folder.

And by using the Mockaroo, I created synthetic data for **Account, BillingAccount, Prospect, CoveredBy, Invoice, InvoiceDetail, Operation, ParticipantClaimant, PostalCode, Contract, ContractPremium, Customer** tables.

I didn't create data for **Employee, Associate, DistrictCoordinator, RegionalCoordinator, StateCoordinator, District, Region, State** because I am not planning to use these data in the user-interface.

At this point, the only data that is needed for the user-interface and that is missing is the forecasting results that will be used to update the premium amounts for chronic disease policies.

Therefore, the next step will be creating forecasts of mortality and hospitalizations caused by chronic diseases. Once this is completed, we will have all the necessary data and will be able to populate the tables we created in the database.

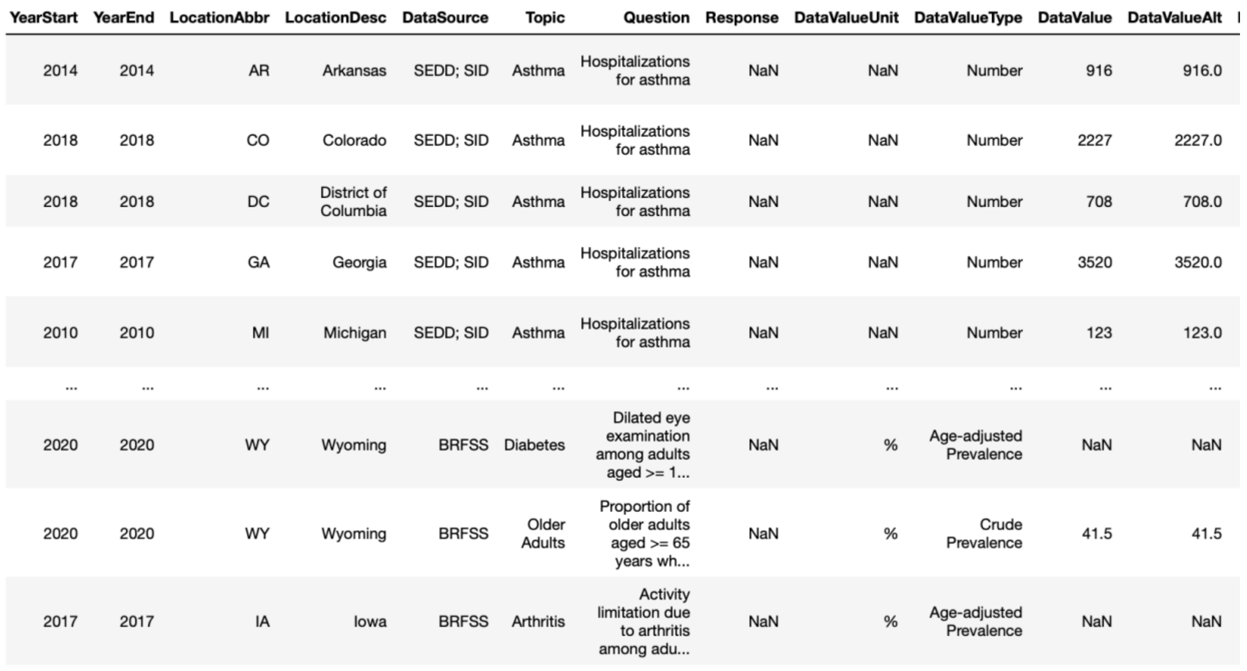
Time Series Forecasting

Before starting to forecast the chronic diseases in the US, the first step is to clean and process the Chronic Disease Indicators data taken from the Centers for Disease Control and Prevention (CDC) and put into the data lake in Microsoft Azure and generate forecasts for the number of hospitalizations and deaths caused by chronic diseases. All of these processes are explained below and the implementation of these steps can be seen in **chronic-disease-prediction.py** file in this folder.

These forecasts will then be used to update the premium amounts of the chronic disease insurance policy for the following year.

Data Processing/Cleaning

The original/raw data taken from the CDC and uploaded to Microsoft Azure can be seen in Figure 3.3.



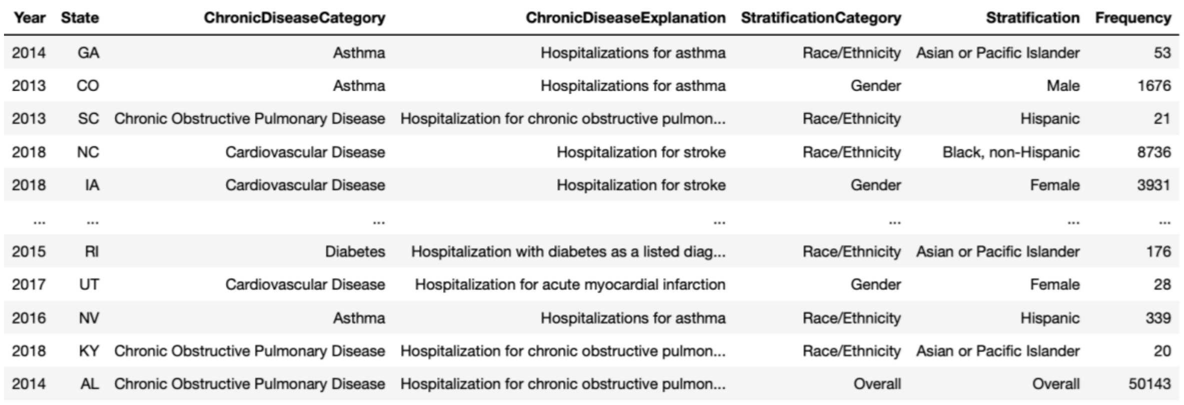
YearStart	YearEnd	LocationAbbr	LocationDesc	DataSource	Topic	Question	Response	DataValueUnit	DataValueType	DataValue	DataValueAlt
2014	2014	AR	Arkansas	SEDD; SID	Asthma	Hospitalizations for asthma	NaN	NaN	Number	916	916.0
2018	2018	CO	Colorado	SEDD; SID	Asthma	Hospitalizations for asthma	NaN	NaN	Number	2227	2227.0
2018	2018	DC	District of Columbia	SEDD; SID	Asthma	Hospitalizations for asthma	NaN	NaN	Number	708	708.0
2017	2017	GA	Georgia	SEDD; SID	Asthma	Hospitalizations for asthma	NaN	NaN	Number	3520	3520.0
2010	2010	MI	Michigan	SEDD; SID	Asthma	Hospitalizations for asthma	NaN	NaN	Number	123	123.0
...	...	...	...	...	...	...	...	...	...	...	...
2020	2020	WY	Wyoming	BRFSS	Diabetes	Dilated eye examination among adults aged >= 1...	NaN	%	Age-adjusted Prevalence	NaN	NaN
2020	2020	WY	Wyoming	BRFSS	Older Adults	Proportion of older adults aged >= 65 years wh...	NaN	%	Crude Prevalence	41.5	41.5
2017	2017	IA	Iowa	BRFSS	Arthritis	Activity limitation due to arthritis among adu...	NaN	%	Age-adjusted Prevalence	NaN	NaN

Figure 3.3: Raw/Original Data

In this data, we see various statistics about chronic health diseases. In the first step, I filtered the values that contain the keywords "hospitalization" or "mortality" in the Question column. This helped me focus on hospitalizations and mortalities caused by different chronic diseases in the US.

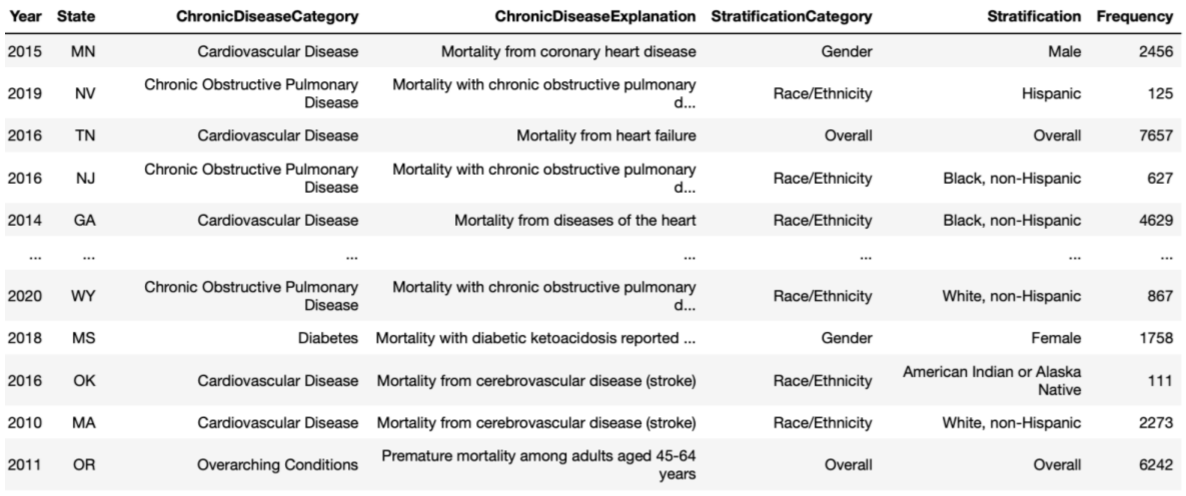
The raw data also includes various forms of statistics such as frequency, rate/percentage, number of cases per 10,000, and number of cases per 1,000,000. I only used the frequency data by filtering out the "Number" value from the DataValueType column because the goal is to forecast the number (frequency) of hospitalizations and mortalities caused by chronic diseases in the US.

In the final step, I divided the data into two tables: one for hospitalization and the other for mortality so that I can put them into two separate time series forecasting models and obtain the forecasts separately. After dividing the data into two tables, I removed the columns that would not be useful for the time series forecasting process. The final versions of the hospitalization and mortality data can be seen in Figure 3.4 and Figure 3.5.



Year	State	ChronicDiseaseCategory	ChronicDiseaseExplanation	StratificationCategory	Stratification	Frequency
2014	GA	Asthma	Hospitalizations for asthma	Race/Ethnicity	Asian or Pacific Islander	53
2013	CO	Asthma	Hospitalizations for asthma	Gender	Male	1676
2013	SC	Chronic Obstructive Pulmonary Disease	Hospitalization for chronic obstructive pulmon...	Race/Ethnicity	Hispanic	21
2018	NC	Cardiovascular Disease	Hospitalization for stroke	Race/Ethnicity	Black, non-Hispanic	8736
2018	IA	Cardiovascular Disease	Hospitalization for stroke	Gender	Female	3931
...	...	...	...	...	...	...
2015	RI	Diabetes	Hospitalization with diabetes as a listed diag...	Race/Ethnicity	Asian or Pacific Islander	176
2017	UT	Cardiovascular Disease	Hospitalization for acute myocardial infarction	Gender	Female	28
2016	NV	Asthma	Hospitalizations for asthma	Race/Ethnicity	Hispanic	339
2018	KY	Chronic Obstructive Pulmonary Disease	Hospitalization for chronic obstructive pulmon...	Race/Ethnicity	Asian or Pacific Islander	20
2014	AL	Chronic Obstructive Pulmonary Disease	Hospitalization for chronic obstructive pulmon...	Overall	Overall	50143

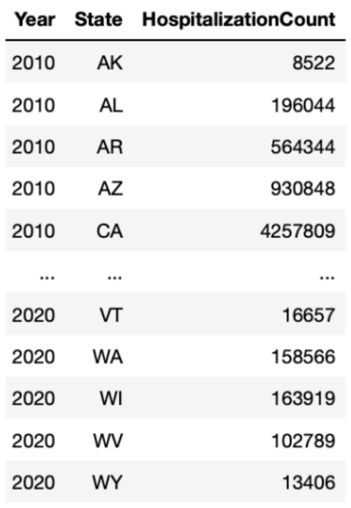
Figure 3.4: Hospitalization Data



Year	State	ChronicDiseaseCategory	ChronicDiseaseExplanation	StratificationCategory	Stratification	Frequency
2015	MN	Cardiovascular Disease	Mortality from coronary heart disease	Gender	Male	2456
2019	NV	Chronic Obstructive Pulmonary Disease	Mortality with chronic obstructive pulmonary d...	Race/Ethnicity	Hispanic	125
2016	TN	Cardiovascular Disease	Mortality from heart failure	Overall	Overall	7657
2016	NJ	Chronic Obstructive Pulmonary Disease	Mortality with chronic obstructive pulmonary d...	Race/Ethnicity	Black, non-Hispanic	627
2014	GA	Cardiovascular Disease	Mortality from diseases of the heart	Race/Ethnicity	Black, non-Hispanic	4629
...	...	...	...	...	...	...
2020	WY	Chronic Obstructive Pulmonary Disease	Mortality with chronic obstructive pulmonary d...	Race/Ethnicity	White, non-Hispanic	867
2018	MS	Diabetes	Mortality with diabetic ketoacidosis reported ...	Gender	Female	1758
2016	OK	Cardiovascular Disease	Mortality from cerebrovascular disease (stroke)	Race/Ethnicity	American Indian or Alaska Native	111
2010	MA	Cardiovascular Disease	Mortality from cerebrovascular disease (stroke)	Race/Ethnicity	White, non-Hispanic	2273
2011	OR	Overarching Conditions	Premature mortality among adults aged 45-64 years	Overall	Overall	6242

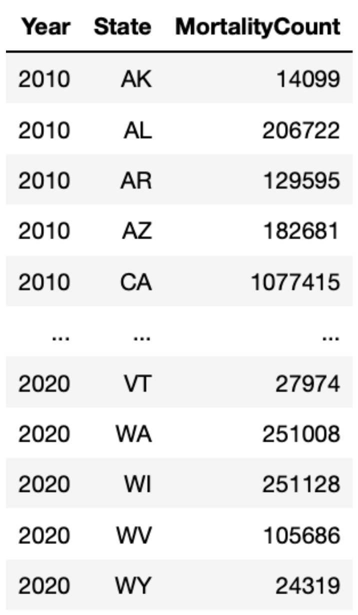
Figure 3.5: Mortality Data

Lastly, I filtered the 'Overall' values in the Stratification column, grouped the data by year and state, and calculated the total number of hospitalizations and mortalities caused by all chronic diseases in each state for each year. The final data is shown in Figure 3.6 and Figure 3.7.



Year	State	HospitalizationCount
2010	AK	8522
2010	AL	196044
2010	AR	564344
2010	AZ	930848
2010	CA	4257809
...	...	...
2020	VT	16657
2020	WA	158566
2020	WI	163919
2020	WV	102789
2020	WY	13406

Figure 3.6: Total Number of Hospitalizations Caused by Chronic Diseases



Year	State	MortalityCount
2010	AK	14099
2010	AL	206722
2010	AR	129595
2010	AZ	182681
2010	CA	1077415
...	...	...
2020	VT	27974
2020	WA	251008
2020	WI	251128
2020	WV	105686
2020	WY	24319

Figure 3.7: Total Number of Mortalities Caused by Chronic Diseases

Time Series Model

These datasets are then input into ARIMA time series models to forecast the total hospitalizations and mortalities for the following year. An example of the forecast for the total number of hospitalizations and mortalities in Indianapolis for the upcoming year can be seen in Figure 3.8 and Figure 3.9.

Hospitalization Forecast

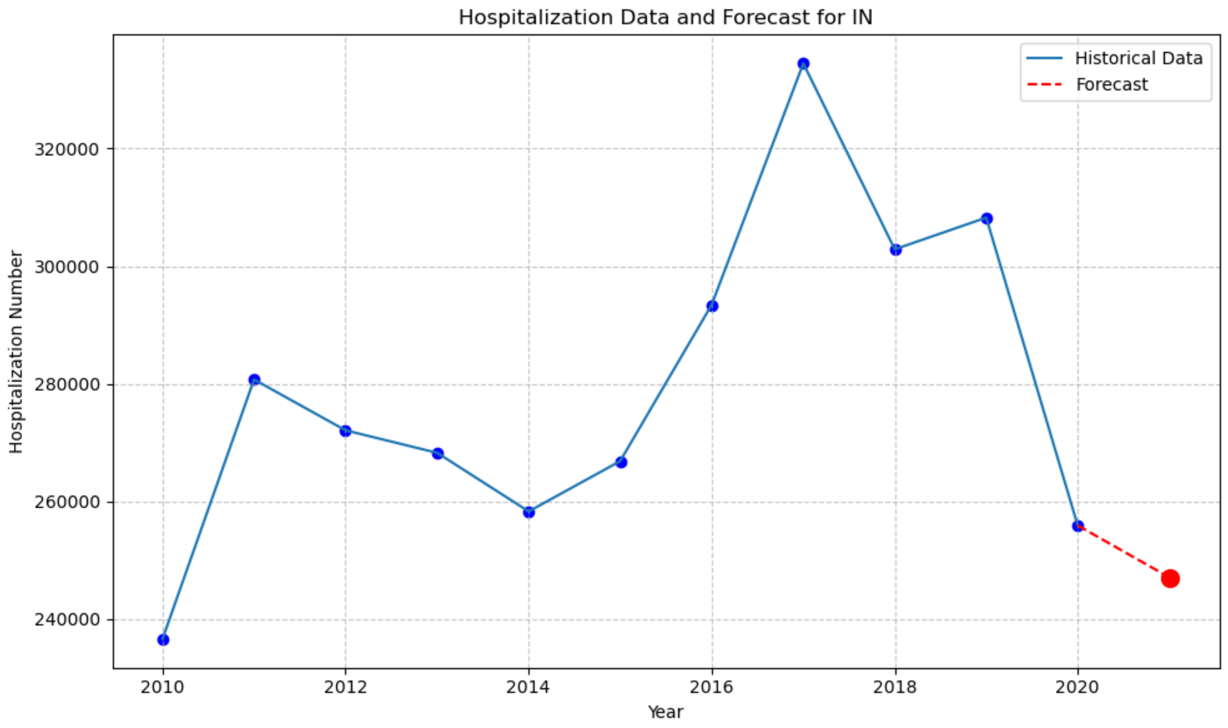


Figure 3.8: Forecast of the Total Number of Hospitalizations in Indianapolis Due to Chronic Diseases in 2021

Mortality Forecast

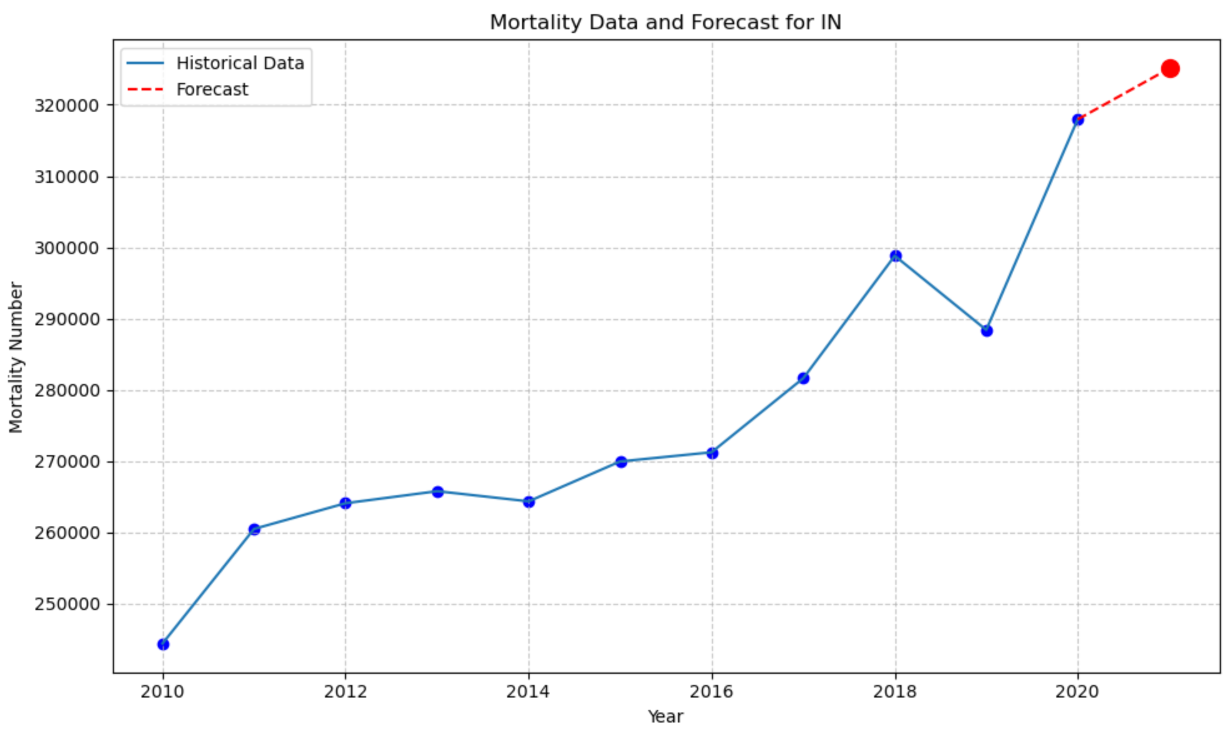


Figure 3.9: Forecast of the Total Number of Mortalities in Indianapolis Due to Chronic Diseases in 2021

The forecast data that is obtained for each state for the following year can be seen in the **combined_forecasts_with_changes.csv** file in this folder. It is also put into data the lake.

Forecast Effect on Premium Amount

Assuming that the insurance company want to decide the next year's premium amount (the amount of money that should be paid by the customers in exchange for the chronic disease insurance coverage) from the previous year, having forecasting of the next year's chronic diseases would be quite helpful for the insurance company.

After obtaining the forecasts of the hospitalities and mortalities across the United States for the next year, the premium amount for the next year can be calculated with the formulas below.

1) **Mortality Change (MC)** = (Mortality Count Next Year Forecast – Mortality Count Current Year) / (Mortality Count Current Year)

2) **Hospitalization Change (HC)** = (Hospitalization Count Next Year Forecast – Hospitalization Count Current Year) / (Hospitalization Count Current Year)

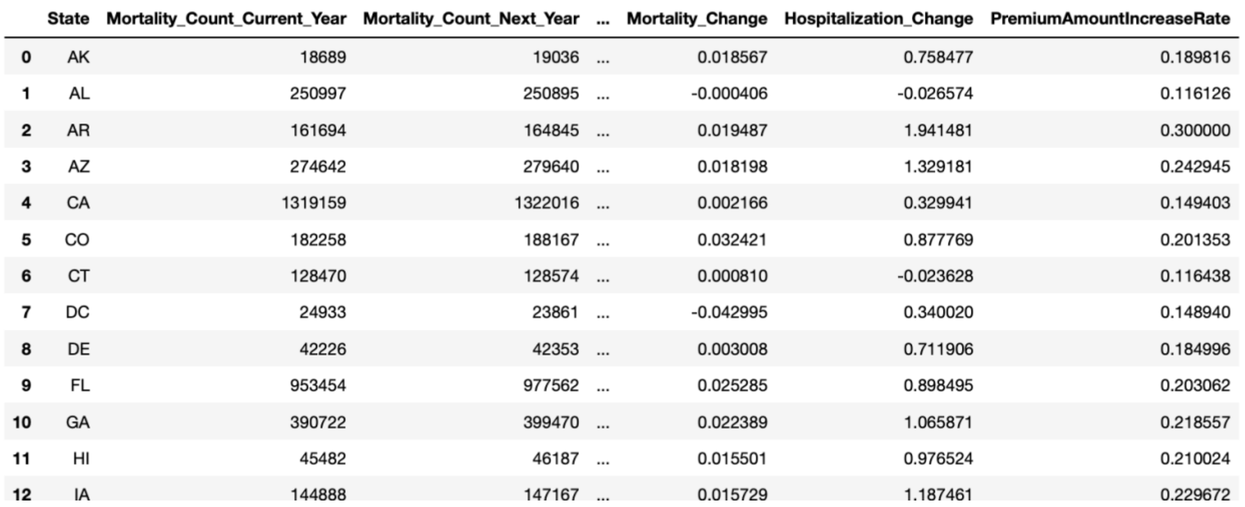
3) **Change** = 0.75 * HC + 0.25 * MC

4) **Premium Amount Increase Rate** = Min-Max Scaling(Change)*

5) **Premium Amount Update** = Premium Amount Current Year x Premium Amount Increase Rate

6) **Premium Amount (Next Year)** = Premium Amount Current Year + Premium Amount Update

Note: *With Min-Max Scaling(.), we scale the change rate in the premium amount (percentage of increase) to somewhere between (0.05 and 0.30). This ensures that the minimum increase in the premium amount will be 5% and maximum increase will be 30%.*



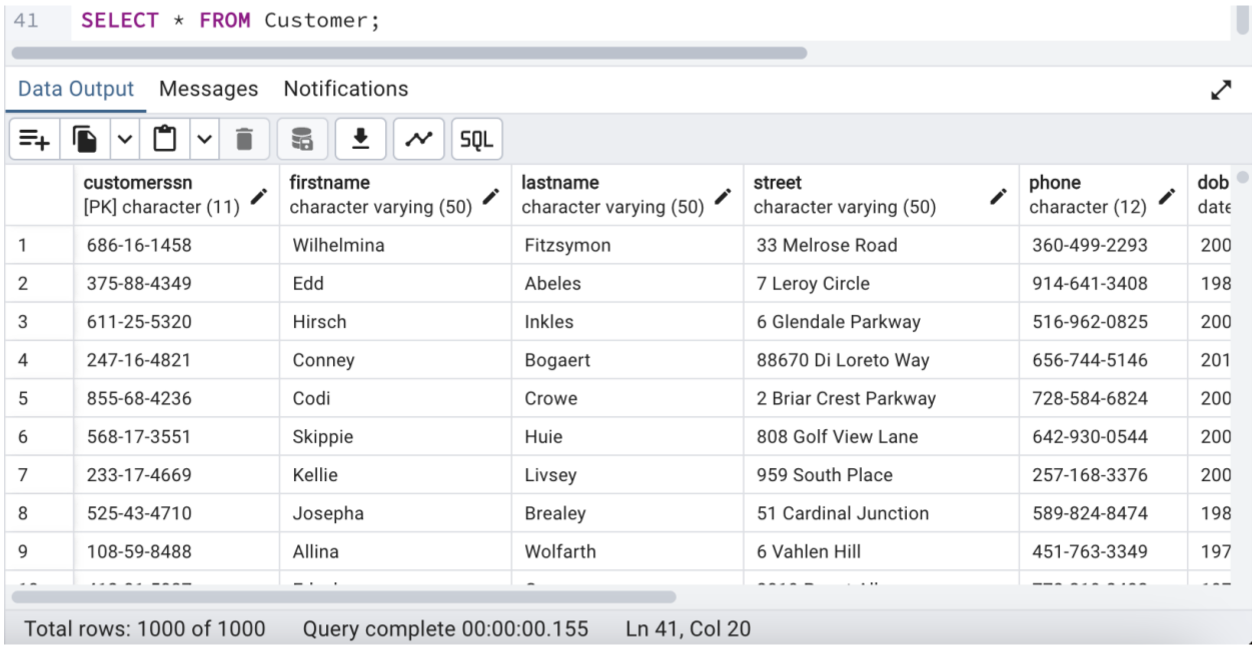
	State	Mortality_Count_Current_Year	Mortality_Count_Next_Year	...	Mortality_Change	Hospitalization_Change	PremiumAmountIncreaseRate
0	AK	18689	19036	...	0.018567	0.758477	0.189816
1	AL	250997	250895	...	-0.000406	-0.026574	0.116126
2	AR	161694	164845	...	0.019487	1.941481	0.300000
3	AZ	274642	279640	...	0.018198	1.329181	0.242945
4	CA	1319159	1322016	...	0.002166	0.329941	0.149403
5	CO	182258	188167	...	0.032421	0.877769	0.201353
6	CT	128470	128574	...	0.000810	-0.023628	0.116438
7	DC	24933	23861	...	-0.042995	0.340020	0.148940
8	DE	42226	42353	...	0.003008	0.711906	0.184996
9	FL	953454	977562	...	0.025285	0.898495	0.203062
10	GA	390722	399470	...	0.022389	1.065871	0.218557
11	HI	45482	46187	...	0.015501	0.976524	0.210024
12	IA	144888	147167	...	0.015729	1.187461	0.229672

Figure 3.10: Forecast of the Total Number of Mortalities in Indianapolis Due to Chronic Diseases in 2021

Inserting Data into Database

Now, we have all the necessary data to populate the tables in the database. Now it is time to put these data into the tables in the database. I used **fill-table.py** to do this and it can be seen in this folder.

After running the **fill-table.py** and filling the tables, the contents of the tables can be seen in the figures below.

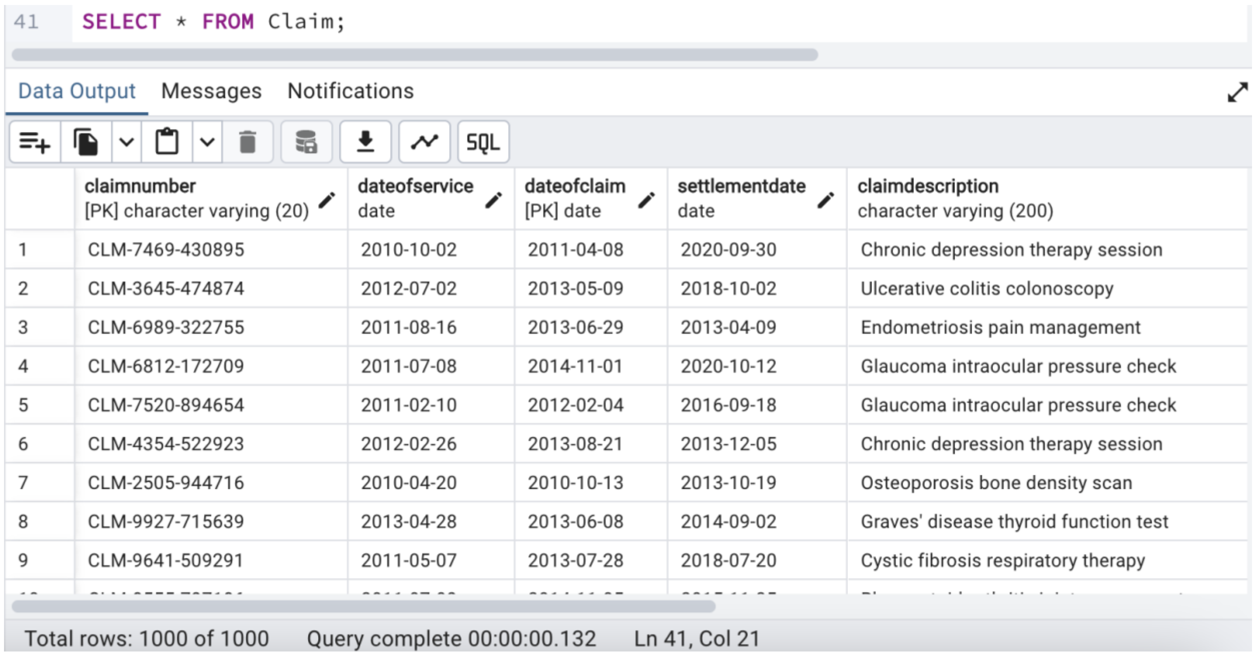


```sql
SELECT * FROM Customer;
```

	customerssn [PK] character (11)	firstname character varying (50)	lastname character varying (50)	street character varying (50)	phone character (12)	dob date
1	686-16-1458	Wilhelmina	Fitzsymon	33 Melrose Road	360-499-2293	200
2	375-88-4349	Edd	Abeles	7 Leroy Circle	914-641-3408	198
3	611-25-5320	Hirsch	Inkles	6 Glendale Parkway	516-962-0825	200
4	247-16-4821	Conney	Bogaert	88670 Di Loreto Way	656-744-5146	201
5	855-68-4236	Codi	Crowe	2 Briar Crest Parkway	728-584-6824	200
6	568-17-3551	Skippie	Huie	808 Golf View Lane	642-930-0544	200
7	233-17-4669	Kellie	Livsey	959 South Place	257-168-3376	200
8	525-43-4710	Josepha	Brealey	51 Cardinal Junction	589-824-8474	198
9	108-59-8488	Allina	Wolfarth	6 Vahlen Hill	451-763-3349	197

Total rows: 1000 of 1000 Query complete 00:00:00.155 Ln 41, Col 20

Figure 3.11: Customer Table in the Database

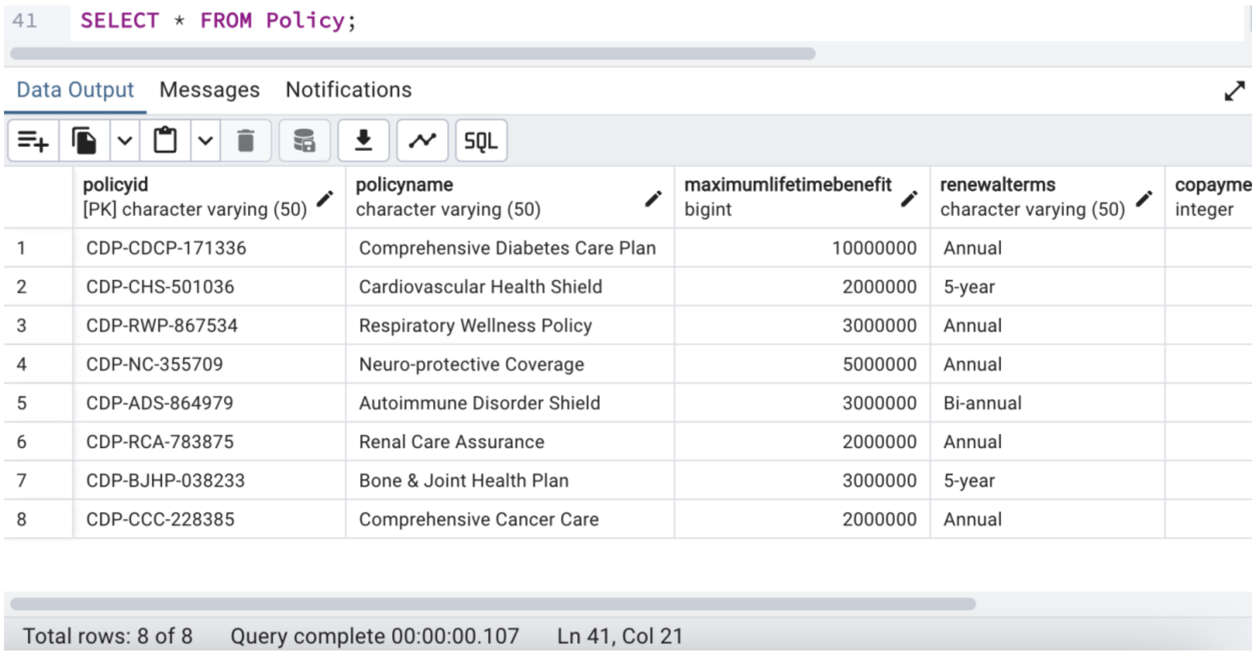


```sql
SELECT * FROM Claim;
```

	claimnumber [PK] character varying (20)	dateofservice date	dateofclaim [PK] date	settlementdate date	claimdescription character varying (200)
1	CLM-7469-430895	2010-10-02	2011-04-08	2020-09-30	Chronic depression therapy session
2	CLM-3645-474874	2012-07-02	2013-05-09	2018-10-02	Ulcerative colitis colonoscopy
3	CLM-6989-322755	2011-08-16	2013-06-29	2013-04-09	Endometriosis pain management
4	CLM-6812-172709	2011-07-08	2014-11-01	2020-10-12	Glaucoma intraocular pressure check
5	CLM-7520-894654	2011-02-10	2012-02-04	2016-09-18	Glaucoma intraocular pressure check
6	CLM-4354-522923	2012-02-26	2013-08-21	2013-12-05	Chronic depression therapy session
7	CLM-2505-944716	2010-04-20	2010-10-13	2013-10-19	Osteoporosis bone density scan
8	CLM-9927-715639	2013-04-28	2013-06-08	2014-09-02	Graves' disease thyroid function test
9	CLM-9641-509291	2011-05-07	2013-07-28	2018-07-20	Cystic fibrosis respiratory therapy

Total rows: 1000 of 1000 Query complete 00:00:00.132 Ln 41, Col 21

Figure 3.12: Claim Table in the Database



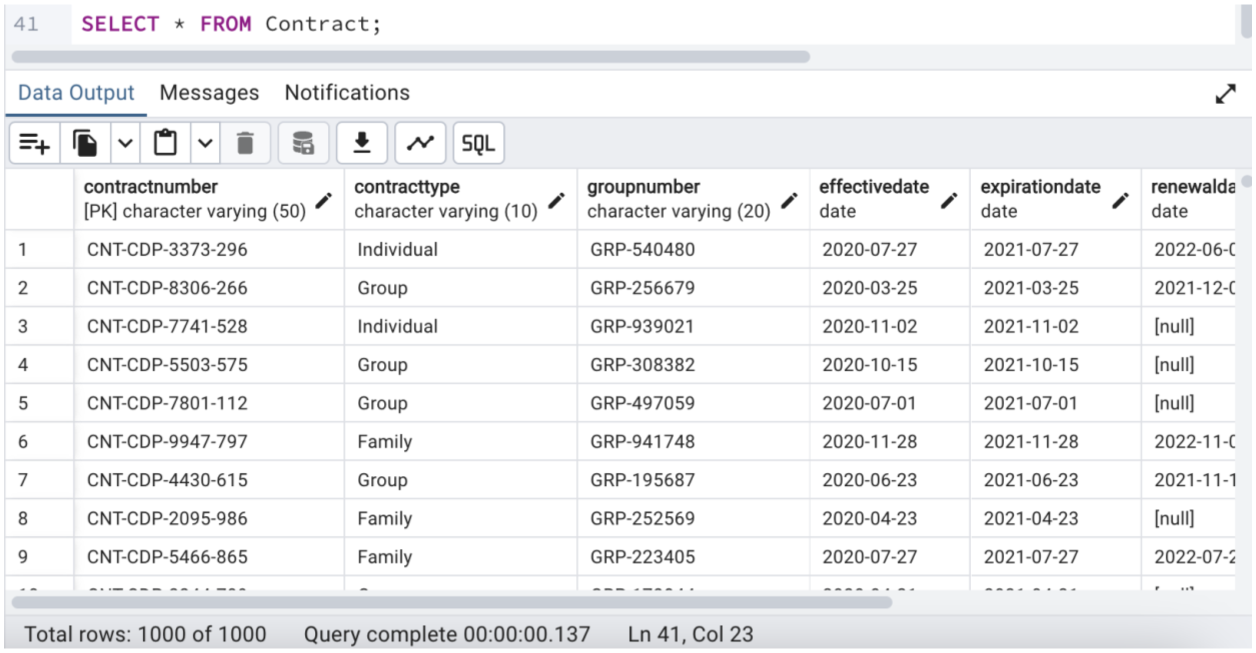
```
SELECT * FROM Policy;
```

Data Output Messages Notifications

	policyid [PK] character varying (50)	policyname character varying (50)	maximumlifetimebenefit bigint	renewalterms character varying (50)	copayme integer
1	CDP-CDCP-171336	Comprehensive Diabetes Care Plan	10000000	Annual	
2	CDP-CHS-501036	Cardiovascular Health Shield	2000000	5-year	
3	CDP-RWP-867534	Respiratory Wellness Policy	3000000	Annual	
4	CDP-NC-355709	Neuro-protective Coverage	5000000	Annual	
5	CDP-ADS-864979	Autoimmune Disorder Shield	3000000	Bi-annual	
6	CDP-RCA-783875	Renal Care Assurance	2000000	Annual	
7	CDP-BJHP-038233	Bone & Joint Health Plan	3000000	5-year	
8	CDP-CCC-228385	Comprehensive Cancer Care	2000000	Annual	

Total rows: 8 of 8 Query complete 00:00:00.107 Ln 41, Col 21

Figure 3.13: Policy Table in the Database



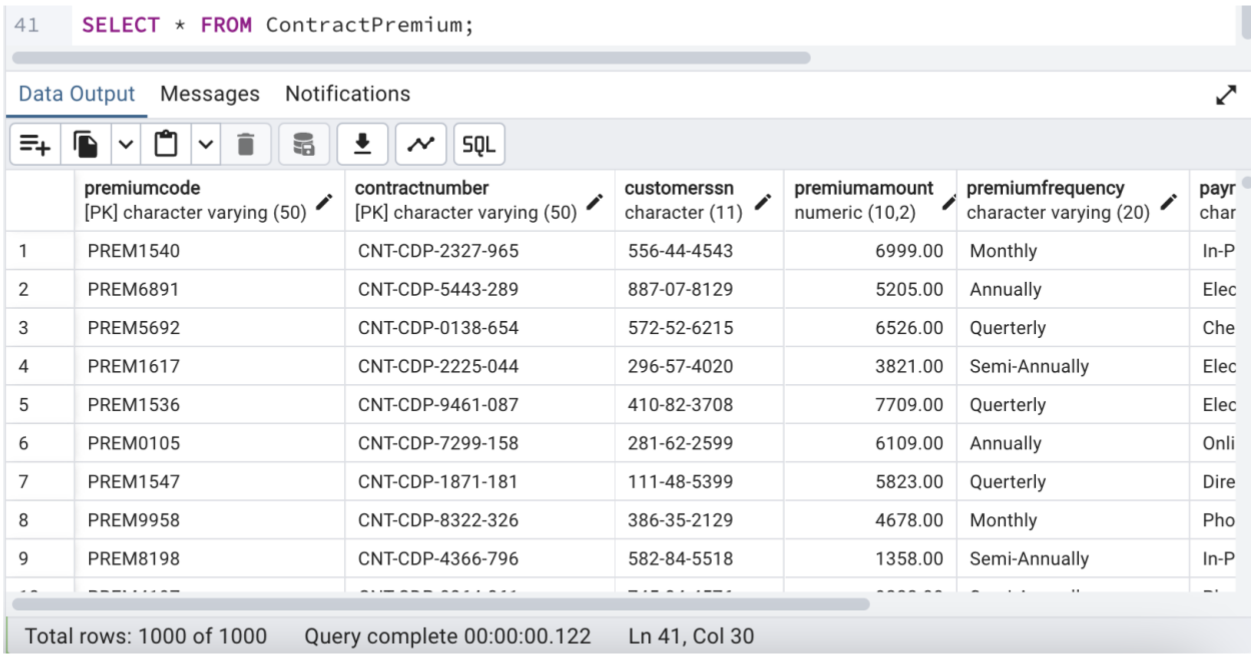
```
SELECT * FROM Contract;
```

Data Output Messages Notifications

	contractnumber [PK] character varying (50)	contracttype character varying (10)	groupnumber character varying (20)	effectivedate date	expirationdate date	renewalda date
1	CNT-CDP-3373-296	Individual	GRP-540480	2020-07-27	2021-07-27	2022-06-
2	CNT-CDP-8306-266	Group	GRP-256679	2020-03-25	2021-03-25	2021-12-
3	CNT-CDP-7741-528	Individual	GRP-939021	2020-11-02	2021-11-02	[null]
4	CNT-CDP-5503-575	Group	GRP-308382	2020-10-15	2021-10-15	[null]
5	CNT-CDP-7801-112	Group	GRP-497059	2020-07-01	2021-07-01	[null]
6	CNT-CDP-9947-797	Family	GRP-941748	2020-11-28	2021-11-28	2022-11-
7	CNT-CDP-4430-615	Group	GRP-195687	2020-06-23	2021-06-23	2021-11-1
8	CNT-CDP-2095-986	Family	GRP-252569	2020-04-23	2021-04-23	[null]
9	CNT-CDP-5466-865	Family	GRP-223405	2020-07-27	2021-07-27	2022-07-
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Total rows: 1000 of 1000 Query complete 00:00:00.137 Ln 41, Col 23

Figure 3.14: Contract Table in the Database



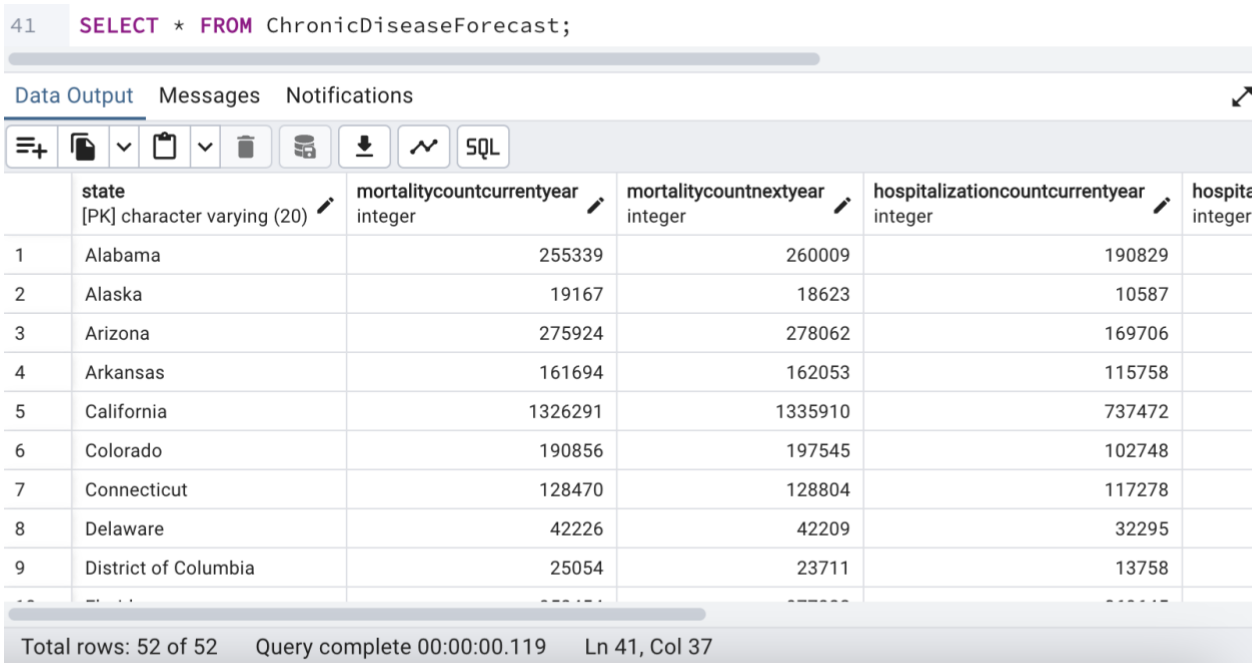
```
SELECT * FROM ContractPremium;
```

Data Output | Messages | Notifications

	premiumcode [PK] character varying (50)	contractnumber [PK] character varying (50)	customerssn character (11)	premiumamount numeric (10,2)	premiumfrequency character varying (20)	payr char
1	PREM1540	CNT-CDP-2327-965	556-44-4543	6999.00	Monthly	In-P
2	PREM6891	CNT-CDP-5443-289	887-07-8129	5205.00	Annually	Elec
3	PREM5692	CNT-CDP-0138-654	572-52-6215	6526.00	Querterly	Che
4	PREM1617	CNT-CDP-2225-044	296-57-4020	3821.00	Semi-Annually	Elec
5	PREM1536	CNT-CDP-9461-087	410-82-3708	7709.00	Querterly	Elec
6	PREM0105	CNT-CDP-7299-158	281-62-2599	6109.00	Annually	Onli
7	PREM1547	CNT-CDP-1871-181	111-48-5399	5823.00	Querterly	Dire
8	PREM9958	CNT-CDP-8322-326	386-35-2129	4678.00	Monthly	Pho
9	PREM8198	CNT-CDP-4366-796	582-84-5518	1358.00	Semi-Annually	In-P

Total rows: 1000 of 1000   Query complete 00:00:00.122   Ln 41, Col 30

Figure 3.15: Contract Premium Table in the Database



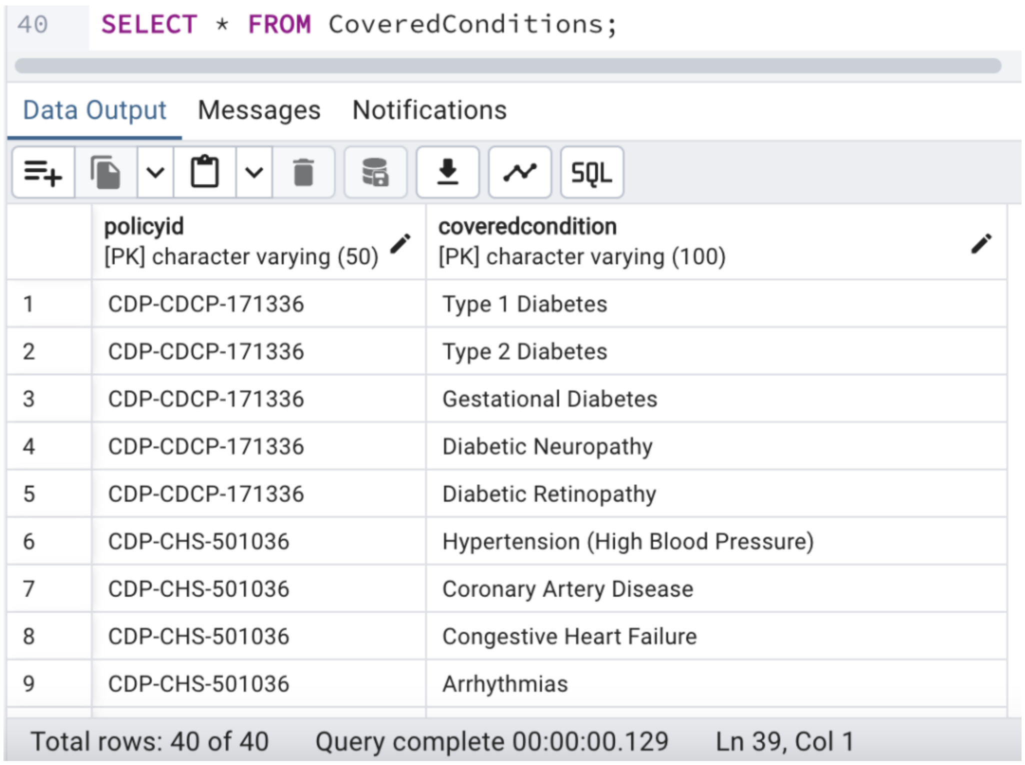
```
SELECT * FROM ChronicDiseaseForecast;
```

Data Output | Messages | Notifications

	state [PK] character varying (20)	mortalitycountcurrentyear integer	mortalitycountnextyear integer	hospitalizationcountcurrentyear integer	hospita integer
1	Alabama	255339	260009	190829	
2	Alaska	19167	18623	10587	
3	Arizona	275924	278062	169706	
4	Arkansas	161694	162053	115758	
5	California	1326291	1335910	737472	
6	Colorado	190856	197545	102748	
7	Connecticut	128470	128804	117278	
8	Delaware	42226	42209	32295	
9	District of Columbia	25054	23711	13758	

Total rows: 52 of 52   Query complete 00:00:00.119   Ln 41, Col 37

Figure 3.16: Chronic Disease Forecast Table in the Database



```sql
SELECT * FROM CoveredConditions;
```

Data Output Messages Notifications

	policyid [PK] character varying (50)	coveredcondition [PK] character varying (100)
1	CDP-CDCP-171336	Type 1 Diabetes
2	CDP-CDCP-171336	Type 2 Diabetes
3	CDP-CDCP-171336	Gestational Diabetes
4	CDP-CDCP-171336	Diabetic Neuropathy
5	CDP-CDCP-171336	Diabetic Retinopathy
6	CDP-CHS-501036	Hypertension (High Blood Pressure)
7	CDP-CHS-501036	Coronary Artery Disease
8	CDP-CHS-501036	Congestive Heart Failure
9	CDP-CHS-501036	Arrhythmias

Total rows: 40 of 40 Query complete 00:00:00.129 Ln 39, Col 1

Figure 3.17: Covered Conditions Table in the Database

Once the tables are filled with the data, I created materialized views that I am planning to use for the user-interface. The codes that I used to create these views can be seen in **create-materialized-views.sql** file in this folder.

Business Use Case Analysis and Workflow Modeling for Insurance Quote Application

Lastly, the workflows of prospects, customers, and participants who go to hospital to get service for his chronic disease can be seen in Figure 3.18, 3.19, and 3.20.

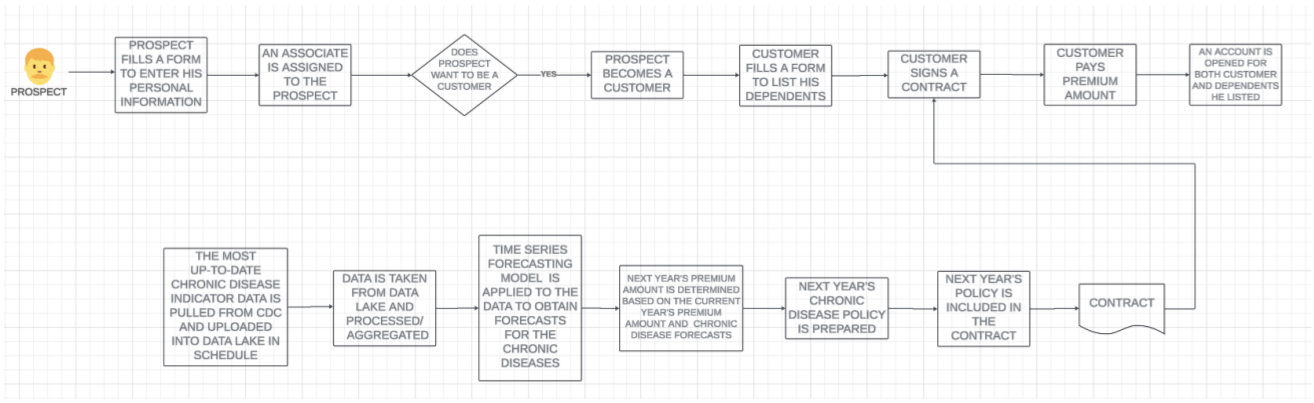

PROSPECT
PROSPECT FILLS A FORM TO ENTER HIS PERSONAL INFORMATION
AN ASSOCIATE IS ASSIGNED TO THE PROSPECT
DOES PROSPECT WANT TO BE A CUSTOMER
YES
PROSPECT BECOMES A CUSTOMER
CUSTOMER FILLS A FORM TO LIST HIS DEPENDENTS
CUSTOMER SIGNS A CONTRACT
CUSTOMER PAYS PREMIUM AMOUNT
AN ACCOUNT IS OPENED FOR BOTH CUSTOMER AND DEPENDENTS HE LISTED
THE MOST UP-TO-DATE CHRONIC DISEASE INDICATOR DATA IS PULLED FROM CDC AND UPLOADED INTO DATA LAKE IN SCHEDULE
DATA IS TAKEN FROM DATA LAKE AND PROCESSED/ AGGREGATED
TIME SERIES FORECASTING MODEL IS APPLIED TO THE DATA TO OBTAIN FORECASTS FOR THE CHRONIC DISEASES
NEXT YEAR'S PREMIUM AMOUNT IS DETERMINED BASED ON THE CURRENT YEAR'S PREMIUM AMOUNT AND CHRONIC DISEASE FORECASTS
NEXT YEAR'S CHRONIC DISEASE POLICY IS PREPARED
NEXT YEAR'S POLICY IS INCLUDED IN THE CONTRACT
CONTRACT

Figure 3.18: Prospect Workflow

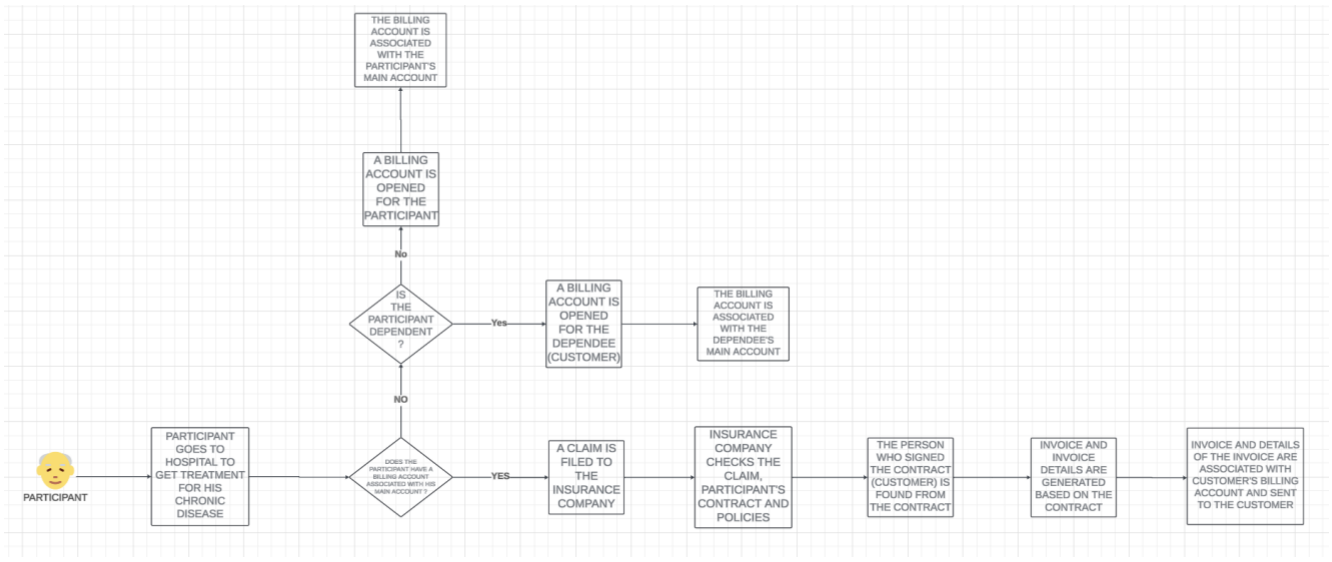

PARTICIPANT
PARTICIPANT GOES TO HOSPITAL TO GET TREATMENT FOR HIS CHRONIC DISEASE
DOES THE PARTICIPANT HAVE A BILLING ACCOUNT ASSOCIATED WITH HIS MAIN ACCOUNT ?
NO
IS THE PARTICIPANT DEPENDENT ?
No
A BILLING ACCOUNT IS OPENED FOR THE PARTICIPANT
THE BILLING ACCOUNT IS ASSOCIATED WITH THE PARTICIPANT'S MAIN ACCOUNT
Yes
A BILLING ACCOUNT IS OPENED FOR THE DEPENDEE (CUSTOMER)
THE BILLING ACCOUNT IS ASSOCIATED WITH THE DEPENDEE'S MAIN ACCOUNT
YES
A CLAIM IS FILED TO THE INSURANCE COMPANY
INSURANCE COMPANY CHECKS THE CLAIM, PARTICIPANT'S CONTRACT AND POLICIES
THE PERSON WHO SIGNED THE CONTRACT (CUSTOMER) IS FOUND FROM THE CONTRACT
INVOICE AND INVOICE DETAILS ARE GENERATED BASED ON THE CONTRACT
INVOICE AND DETAILS OF THE INVOICE ARE ASSOCIATED WITH CUSTOMER'S BILLING ACCOUNT AND SENT TO THE CUSTOMER

Figure 3.19: Participant Workflow

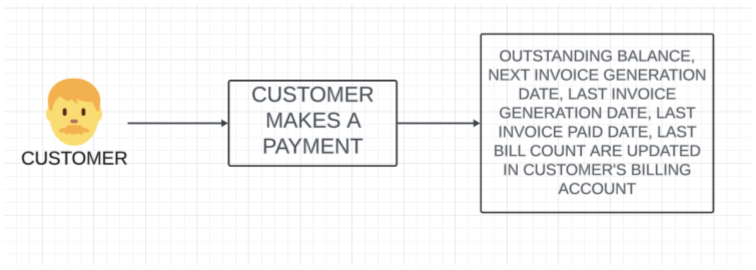

CUSTOMER
CUSTOMER MAKES A PAYMENT
OUTSTANDING BALANCE, NEXT INVOICE GENERATION DATE, LAST INVOICE GENERATION DATE, LAST INVOICE PAID DATE, LAST BILL COUNT ARE UPDATED IN CUSTOMER'S BILLING ACCOUNT

Figure 3.20: Customer Workflow

Workflows

At this point, we have created the entity-relationship diagram, converted it into the logical schema, built a database using PostgreSQL based on this schema, generated synthetic data using Mockaroo and Python, developed a time series forecasting model, obtained forecasts for the number of hospitalizations and deaths due to chronic health diseases for each state in the US, and populated the database tables with all this data.

The next step is to automate these processes and develop a pipeline that

1. downloads the Chronic Disease Indicators data from CDC,
2. preprocesses/cleans this data,
3. uses the cleaned data as an input to the pre-trained and optimized time series forecasting model that is obtained in the 3rd part of the project,
4. obtains forecasts for the number of hospitalizations and deaths due to chronic health diseases for the next year in each state of the US,
5. calculates the premium amount increase rate based on these forecasts,
6. updates the ChronicDiseaseForecast table in the database,
7. refreshes the materialized view of the joined table, which is created by joining the ChronicDiseaseForecast, Contract, ContractPremium, and Customer tables

in the first day of each month.

With this system, the entire operation will run seamlessly, and this will allow customers to learn their new premium amounts with the most up-to-date forecasting results without human intervention. The whole process is summarized in **Figure 4.1** (It can also be seen in the workflow1.pdf file in this folder).

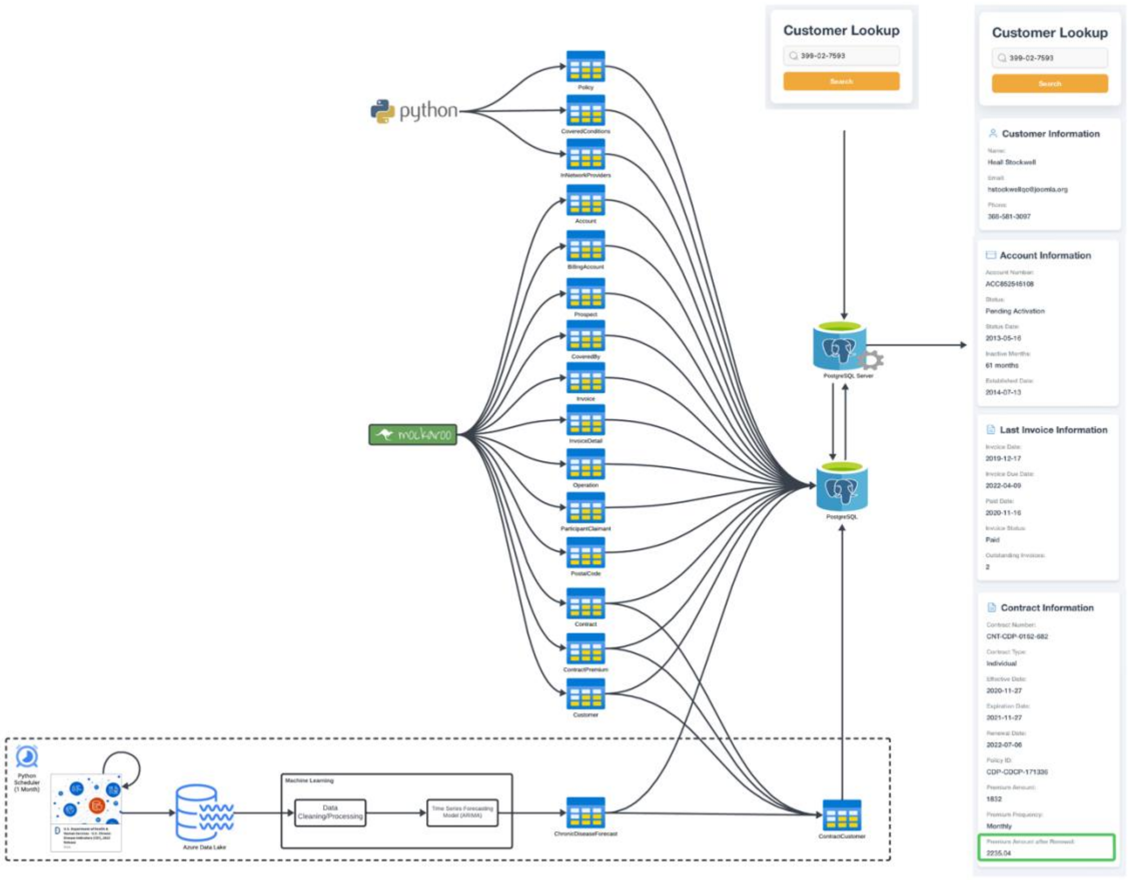


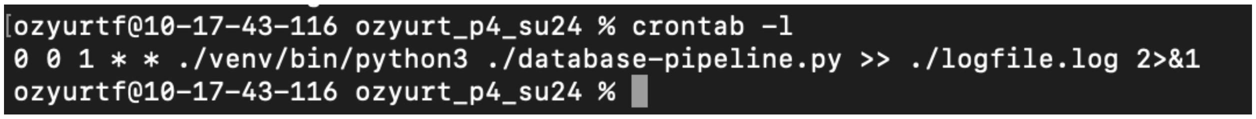
Figure 4.1: Pipeline

The code that is used to automate all these processes can be found in the **database-pipeline.py** file within this folder.

And to make sure that this pipeline runs each month, crontab is used with the instructions below.

chmod +x database-design.py
crontab -e
0 0 1 * * .venv/bin/python3 ./monthly_update.py >> ./logfile.log 2>&1

After running these instructions, we can see the scheduled cron job in Figure 4.2.



```
[ozyurtf@10-17-43-116 ozyurt_p4_su24 % crontab -l
 0 0 1 * * ./venv/bin/python3 ./database-pipeline.py >> ./logfile.log 2>&1
 ozyurtf@10-17-43-116 ozyurt_p4_su24 % 
```

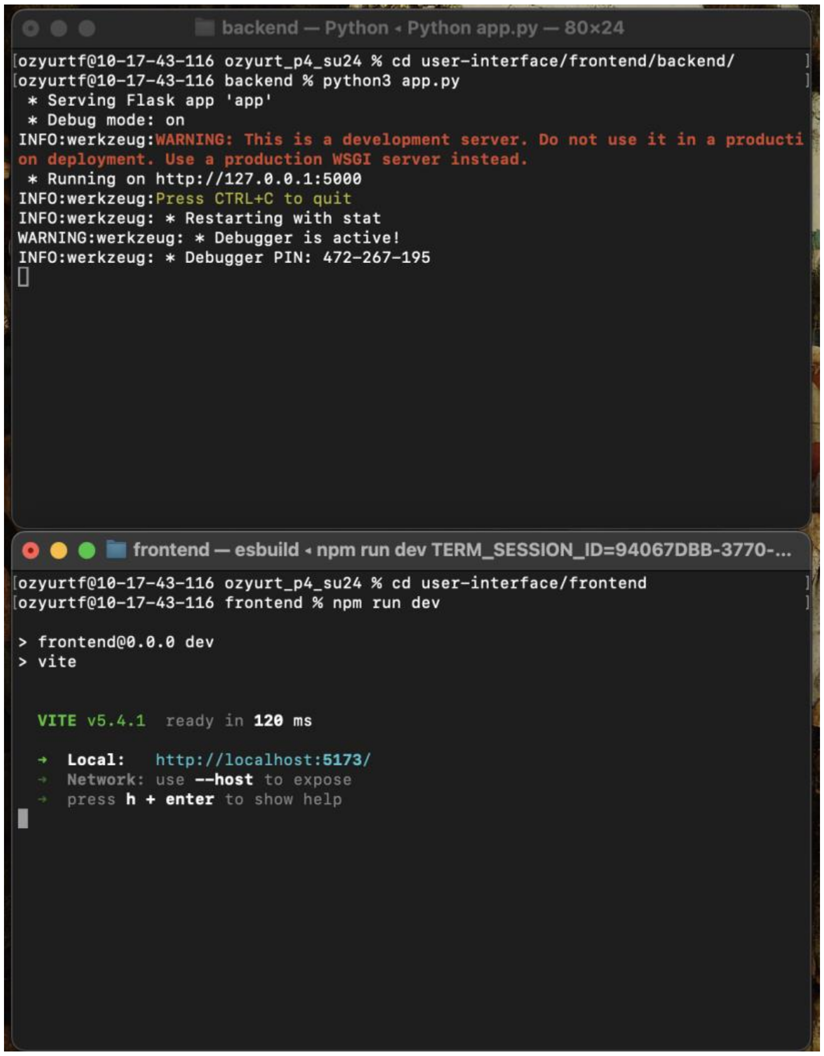
Figure 4.2: Scheduled Cron Job

With this cron job,**.venv/bin/python3 ./database-pipeline.py >> logfile.log 2>&1** will be run in the terminal in the first day of each month and the database will be updated automatically on a monthly basis.

Now, the final step is to build a user interface through which users can interact with the database. The files that are used to create the user interface are located in the **user-interface** subfolder in this folder.

The user interface consists of two main components: the backend and the frontend. The backend component is the **app.py** file located in the **user-interface/frontend/backend/** subfolder. The frontend component is the **CustomerLookup.jsx** file located in the **user-interface/frontend/src/customers/** subfolder.

To start the user interface, we need to run these two files simultaneously in separate terminals. Instructions on how to do this can be found in Figure 4.3.



```
[ozyurtf@10-17-43-116 ozyurt_p4_su24 % cd user-interface/frontend/backend/
[ozyurtf@10-17-43-116 backend % python3 app.py
 * Serving Flask app 'app'
 * Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in a producti
on deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with stat
WARNING:werkzeug: * Debugger is active!
INFO:werkzeug: * Debugger PIN: 472-267-195
```

frontend — esbuild ◂ npm run dev TERM_SESSION_ID=94067DBB-3770-...

```
[ozyurtf@10-17-43-116 ozyurt_p4_su24 % cd user-interface/frontend
[ozyurtf@10-17-43-116 frontend % npm run dev

> frontend@0.0.0 dev
> vite


  VITE v5.4.1  ready in 120 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

Figure 4.3: Running User-Interface

After following these instructions, we can visit http://localhost:5173/ to interact with the user interface that can be seen in Figure 4.4.

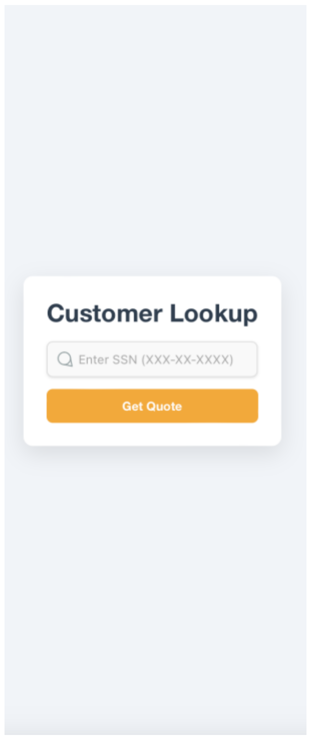
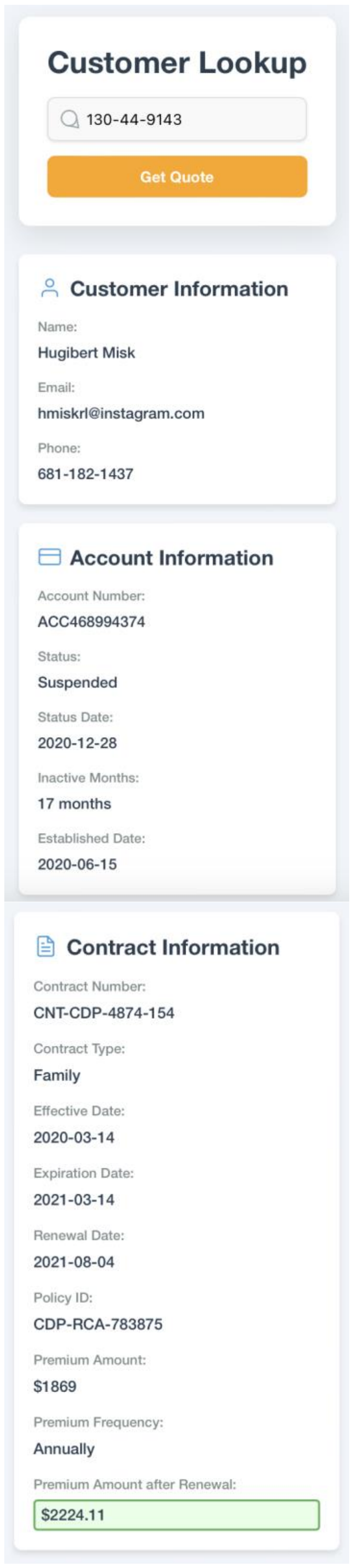


Figure 4.4: User-Interface

Once we enter the SSN and clicking the “Get Quote” button, the results are retrieved from the database and displayed in the user interface, as illustrated in Figure 4.5.



Customer Information

Name:
Hugibert Misk

Email:
hmiskrl@instagram.com

Phone:
681-182-1437

Account Information

Account Number:
ACC468994374

Status:
Suspended

Status Date:
2020-12-28

Inactive Months:
17 months

Established Date:
2020-06-15

Contract Information

Contract Number:
CNT-CDP-4874-154

Contract Type:
Family

Effective Date:
2020-03-14

Expiration Date:
2021-03-14

Renewal Date:
2021-08-04

Policy ID:
CDP-RCA-783875

Premium Amount:
$1869

Premium Frequency:
Annually

Premium Amount after Renewal:
$2224.11

Figure 4.5: User-Interface

The instructions for running the scripts in the user-interface subfolder and interacting with the user interface can also be found in the **user-interface.mov** file in this folder.

As shown in Figure 4.5, once the SSN of the customer is entered, the new premium amount is calculated based on the customer's state and forecasts of hospitalizations and deaths due to chronic disease in that state. This amount is then displayed to the user, as indicated by the green box at the bottom of Figure 4.5.

Note: *If the Renewal Date appears as None in the user interface, it means the customer does not want to renew his contract. In this case, the new premium amount is highlighted with a red box.*

Reference Architecture

As mentioned in the 1st part of the project, when we follow the EDA (Enterprise Data Architecture), there are some key points that we should consider. For instance, in an organization, different teams tend to create incompatible systems. To bring all of them into the same table and to ensure consistency between different projects, some kind of template is needed. This template is called reference architecture. And the reference architecture that is followed throughout this project can be seen in the Figure 4.6.

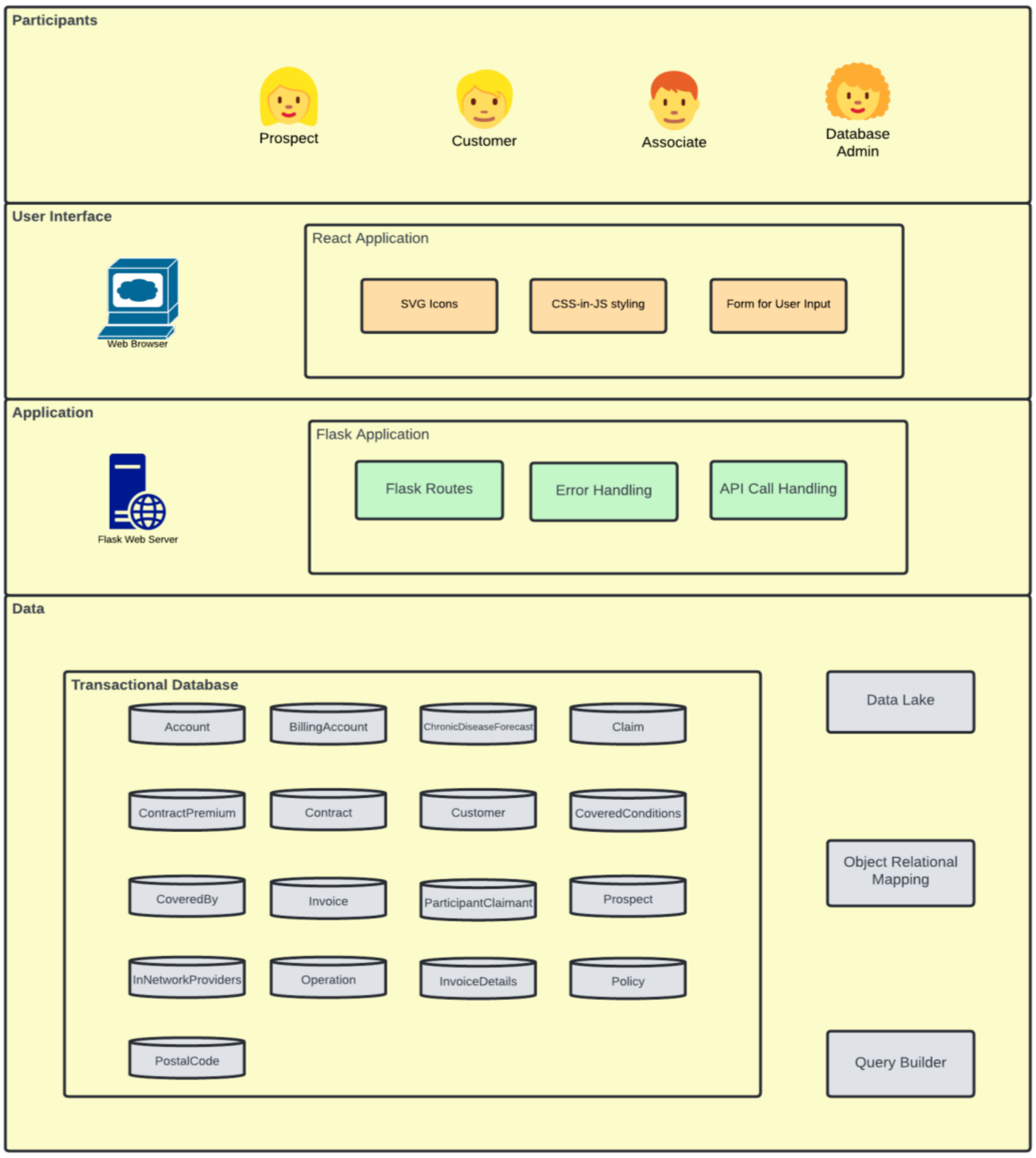


Figure 4.6: Reference Architecture

Data Governance, Data Quality, and Database Schema Constraints

As we mentioned in the 1st part of the project, one of our goals during the database design is to ensure that regulatory requirements and internal policies are met, and the data is accurate, available, secure, and consistent. Responsibilities are assigned to different people to ensure data quality and integrity.

To ensure data accuracy and consistency, various constraints were applied during the database design. For instance, when creating the tables, the expected data type for each column is specified. In this way, if someone attempts to insert an integer into a column that expects a date value, for instance, the system will prevent it. In addition, the data that is used for time series forecasting is cleaned/preprocessed properly so that high-quality, clean, and accurate data can be used to train the model. During the cleaning process:

- only the records that are related to either hospitalization or mortality are filtered,
- the columns that won't be useful for the forecasting process are removed,
- only the records with numbers are used and the records that are associated with rates, averages, are ignored
- the format of the data is changed from long format to wide format by aggregating data.

In addition to these, the tables in the database are designed in such a way that if a foreign key in one table references a column in another table, the foreign key must exist in the referenced column to maintain data integrity.

Additionally, the unique and non-null values are enforced in the primary key columns to prevent duplication and increase accessibility because duplicated values in a primary key column can violate data integrity and cause inaccuracies and null values in the primary key column prevents us to access/use the data.

Potential Issues with Data Used for Time Series Forecasting

In the current pipeline, only Cardiovascular Disease, Alcohol, Chronic Kidney Disease, Chronic Obstructive Pulmonary Disease, and Diabetes related hospitalizations and deaths are used because the data taken from the https://www.cdc.gov includes only these.

Basically, the number of hospitalizations and deaths due to these diseases are summed for each state to obtain the total number of hospitalizations and deaths due to these diseases. The hospitalizations and deaths are forecasted based on this data and premium amounts of the contracts are updated according to this.

However, this is not the most logical approach because the company in this project has 8 policies in the database and each policy covers different diseases. So, applying the same increase rate to all premium amounts that are associated with different policies may cause some unfair increases to the premium amounts.

Future Developments

In the future, various encryption methods (e.g., pgcrypto extension, SSL/TSL, custom encryption functions, data masking) and access controls (e.g., role-based access control, row-level security, network-level access control) can be implemented to protect sensitive insurance data and to ensure that only specific groups within the company have access to the data in this database.

Additionally, each policy in the insurance company covers a list of specific conditions in the current database. And in the current pipeline, the number of hospitalizations and deaths due to various chronic diseases is summed for each state, **forecasts of the total number of hospitalizations and deaths** are generated, and premium amounts of all contracts that are associated with different policies are updated based on these forecasts.

However, we can compute the forecasts of hospitalizations and deaths for each chronic disease type **separately** in the future. The contract premium amount can then be updated **based on the forecasts for the diseases covered by the policy** that is associated with the contract.

For instance, if Policy 1 covers Type 1 Diabetes, Type 2 Diabetes, Gestational Diabetes, Diabetic Neuropathy, and Diabetic Retionopathy, the premium amount of the contract that is associated with Policy 1 can be updated based on the forecasts of the hospitalizations and deaths due to diseases covered by Policy 1. In the current system, the premium amount is updated based on the forecasts of the total number of hospitalizations and deaths due to all kinds of chronic diseases.

Aside from these, a more up-to-date data that is obtained in a smaller time frames (e.g., daily, weekly, monthly) can be used. Through this way, we can run the pipeline and the data in the database reflects the real world more frequently.

Lastly, a customer can have multiple accounts, multiple contracts and multiple policies. In the user-interface, only one of these is shown. In the future, the user-interface can be updated accordingly and all accounts, contracts, and policies associated with the customer can be shown.