**Database Project Part 4**

**Member 1 Name**: Singh, Siddhartha (N11434921)

**Member 2 Name**: Bhattacharya, Aakash (N13936743)

**Date**: 22nd December 2022

**Course Section Numbe**r: CSCI-GA.2433-001

**Total in points (100 points total):** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Professor’s Comments:**

We, hereby certify by submitting this project that all the efforts put into this assignment are our own; any resources referred to have been marked in the References section of the homework.

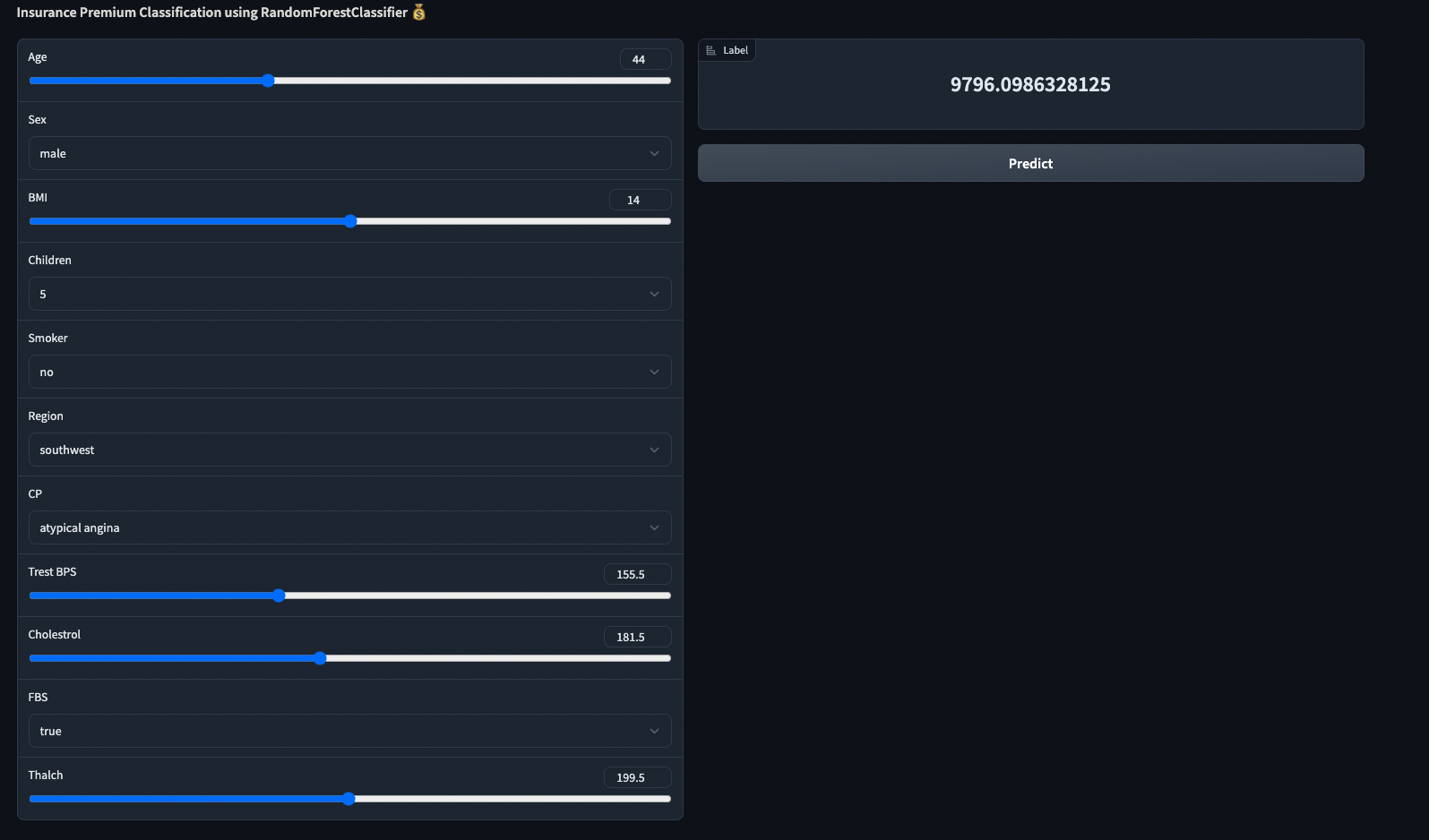
**Affirmation of our Independent Efforts:** Siddhartha Singh, [Aakash Bhattacharya](mailto:ab9541@nyu.edu)

**User Interface for the application**

Note: We have also utilized SQLAlchemy ORM to create our tables. More information on this later in the document.

Github link: <https://github.com/Black-Caesaris/DB-Project-FL22>

We have utilized Gradio to build our User Interface for the Application. We have predicted insurance prices for a customer depending on his/her/their health parameters.

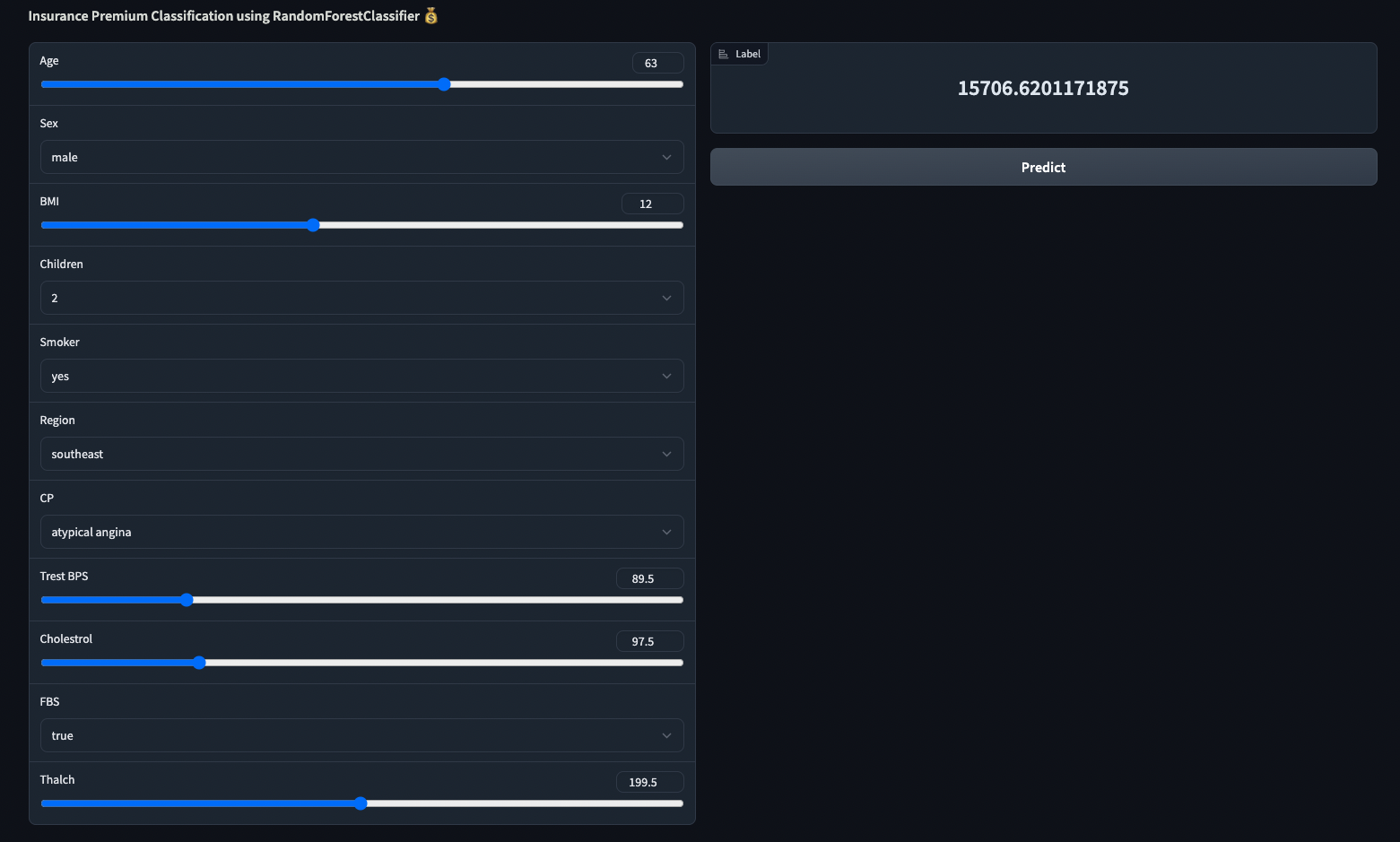


On the basis of the above mentioned parameters we predict the insurance premium price.

The predicted premium price changes as the information changes. The higher the age, children, and smoking nature the premium price increases as expected. We also include a penalty of the patient having a higher probability of having a heart attack to the premium price. The prediction that a person may have a heart attack is based on parameters such as cholesterol, thalch, trest bps, FBS etc. We have trained two ML models

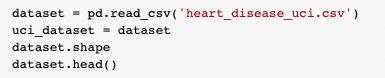
1. Heart disease prediction which predicts if a person will have a heart disease or not in the future
2. Insurance expenses prediction using person's vitals

We club output of both the predictions to give out a final premium price. The vitals being added by the user can be added back to the DB to keep it in records so that they can be used by the company for better insights in the future.



**ML Model Training Process**

1. Loaded heart disease dataset

****

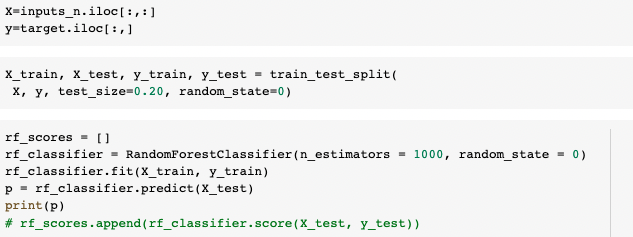
1. Took care of missing values using mean/median



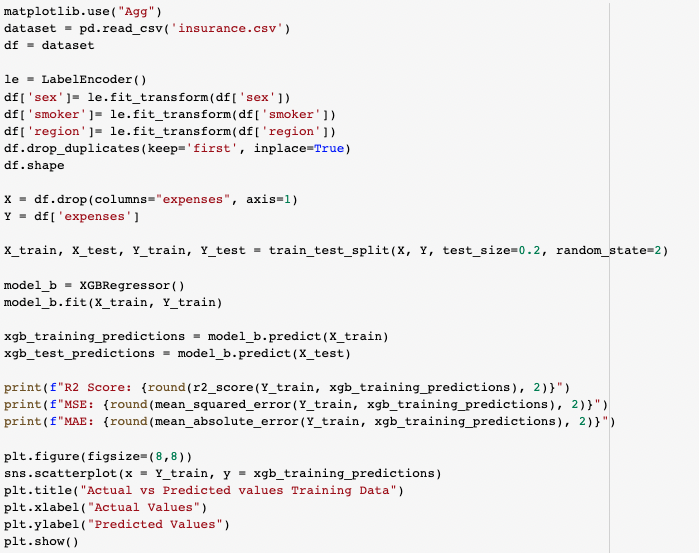
1. Converted categorical values to numerical



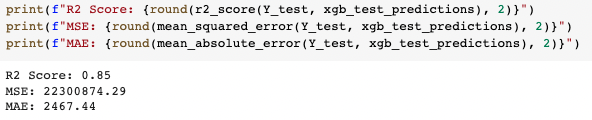
1. Ran Random Forest Classifier

****

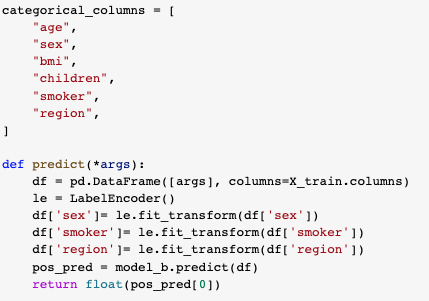
1. Loaded insurance dataset and used XGBoost for regression to predict premium prices

****

1. Accuracy of prediction



1. Prediction function for the UI to use on button click

****

1. Sample part of UI code using Gradio

****

**Database Table Creation using ORM**

ORM stands for object-relational mapping. It is a technique that allows a software application to interact with a database in an object-oriented way, rather than a more traditional, procedural way.

The purpose of an ORM is to abstract away the complexities of working with a database and allow developers to work with objects in their code. This can make it easier to write and maintain code, as well as improve the performance of the application.

Some examples of ORM tools include Hibernate (Java), ActiveRecord (Ruby on Rails), and Doctrine (PHP). These tools provide a way to map objects in the application to tables in the database, and can also handle tasks such as querying the database, managing transactions, and maintaining object relationships.

SQLAlchemy is an open-source Python library that provides a nice API for interacting with databases. It includes an ORM (object-relational mapper) that allows you to work with databases in a more object-oriented way.

With SQLAlchemy, you can define classes that map to tables in a database, and instance of those classes map to rows in the table. You can then use the SQLAlchemy ORM to query the database, create, read, update, and delete records, and manage transactions.

One of the benefits of using SQLAlchemy is that it provides a high-level API for interacting with databases, which can make it easier to write and maintain code. It also supports a wide range of databases, including popular ones such as PostgreSQL, MySQL, and SQLite.

We have attempted to make use of SQLAlchemy ORM to create our relations.

In the following pages we discuss each ORM, and its characteristics in detail.

**Customer Relationship creation using ORM**

|  | The customer relationship outlines the details of the customers. This class holds all the details about a customer and it is also crucial to the healthcare datasets that have been used to generate insights.    This data is already in 1NF form as each cell contains a single value and each record is unique.  This data is already in 2NF form as the data is already in 1NF and the single column primary key is not functionally dependent on any subset of candidate key relation.  This class interacts with the policy transactions, health attributes, claim transactions, claim progression classes.  This data can be further normalized to 3NF form as there’s a transitive functional dependency in the data. Essentially, changing a non-key column causes other non-key columns to change. For example, Name causes SSN to change. |
| --- | --- |

**Policy Transaction ORM**

|  | The policy transactions class essentially holds all the transactions that flow through the insurance company.  This data is already in 1NF form as each cell contains a single value and each record is unique.  This data is already in 2NF form as the data is already in 1NF and the single column primary key is not functionally dependent on any subset of candidate key relation.  This class interacts with the policy, invoice, customer, data dimension, employee classes. |
| --- | --- |

**Invoice ORM**

|  | Invoice class holds the details about a generated invoice.  This data is already in 1NF form as each cell contains a single value and each record is unique.  This data is already in 2NF form as the data is already in 1NF and the single column primary key is not functionally dependent on any subset of candidate key relation.  This class interacts with the policy transactions. |
| --- | --- |

**Policy ORM**

|  | Policy class is the crux class which holds the details about a policy.  This data is already in 1NF form as each cell contains a single value and each record is unique.  This data is already in 2NF form as the data is already in 1NF and the single column primary key is not functionally dependent on any subset of candidate key relation.  This class interacts with the policy transactions, claim progression, plan, claim transaction. |
| --- | --- |

**Plan ORM**

|  | Plan class basically holds the details about the insurance plan a customer is opting for.  This data is already in 1NF form as each cell contains a single value and each record is unique.  This data is already in 2NF form as the data is already in 1NF and the single column primary key is not functionally dependent on any subset of candidate key relation.  This class interacts with the profit loss snapshot, revenue snapshot, claim progression and policy classes. |
| --- | --- |

**Revenue Snapshot**

|  | The revenue snapshot class is used for data analytics use cases in the whole model to get revenue insights about the insurance business.  This data is already in 1NF form as each cell contains a single value and each record is unique.  This data is already in 2NF form as the data is already in 1NF and the single column primary key is not functionally dependent on any subset of candidate key relation.  This class interacts with the plan and data dimension classes. |
| --- | --- |

**Employee**

|  | The employee class basically depicts the two types of employees in the insurance business. The two types are: agent and a price actuator.  This data is already in 1NF form as each cell contains a single value and each record is unique.  This data is already in 2NF form as the data is already in 1NF and the single column primary key is not functionally dependent on any subset of candidate key relation.  This class interacts with the policy transactions, claim progression, claim transaction and address classes. |
| --- | --- |

**Address**

|  | Address essentially holds the address fields in the overall design.  This data is already in 1NF form as each cell contains a single value and each record is unique.  This data is already in 2NF form as the data is already in 1NF and the single column primary key is not functionally dependent on any subset of candidate key relation.  This class interacts with the employee class. |
| --- | --- |

**Date Dimension**

|  | The data is a separate dimension in the overall design. This is definitely useful for data analytics use cases.  This data is already in 1NF form as each cell contains a single value and each record is unique.  This data is already in 2NF form as the data is already in 1NF and the single column primary key is not functionally dependent on any subset of candidate key relation.  This class interacts with the policy transactions, revenue snapshot and the profit loss snapshot classes. |
| --- | --- |

**Claim Progression**

|  | Claim progression class essentially manages how a claim progresses for a customer. It manages the lifecycle of a claim.  This data is already in 1NF form as each cell contains a single value and each record is unique.  This data is already in 2NF form as the data is already in 1NF and the single column primary key is not functionally dependent on any subset of candidate key relation.  This class interacts with the policy, employee, plan and customer classes. |
| --- | --- |

**Profit Loss Snapshot**

|  | This snapshot class is again responsible for deriving data insights about the insurance company for any profit and losses.  This data is already in 1NF form as each cell contains a single value and each record is unique.  This data is already in 2NF form as the data is already in 1NF and the single column primary key is not functionally dependent on any subset of candidate key relation.  This class interacts with the plan and date dimension classes. |
| --- | --- |

**Added Relations and Attributes from Data Sets for Insights Generation**

**Health Essential Attributes**

|  | This class is responsible for fetching the data from the data lake and initializing specific fields from the dataset to the class.  The ultimate motive of this healthcare data analytics is to predict the likeliness of a disease given some of the attributes using an ML model.  This data can then be used by the insurance company to decide the premium price of the insurance and whether it will be profitable to give out insurance to an unhealthy person in the long run using analytical techniques. |
| --- | --- |

**Indexes**

In our previous part we had used postgresql and had experimented with creating indexes to make querying more efficient. Although we haven’t created indexes using the ORM; since we create indexes is something that we feel would come from experience with our product. Also, we have talked about indexes and have set them up in our earlier project parts.

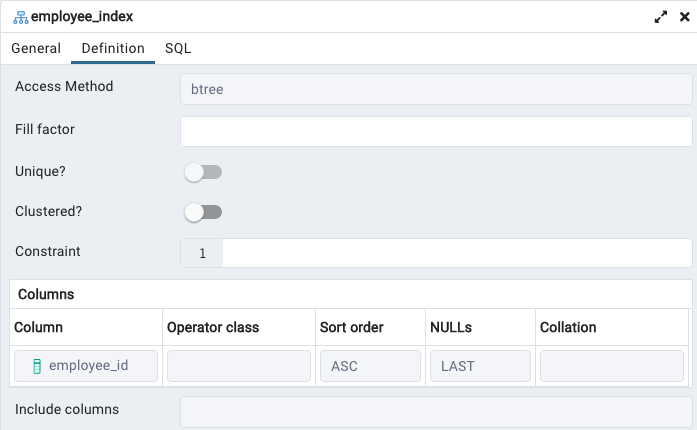
Indexing is a way to order an unordered table that maximizes the query’s efficiency during lookup and searching. When a table is created, the order in which the data is stored is not known to us and it is not discernable. In order to query a table effectively without searching overheads, indexes can be created on tables which will be queried repeatedly. This will lead to optimized query execution leading to lesser time taken for searching specific data.

There are multiple types of indexes that exist in Postgres. All the index types have a different storage structure and algorithm to cater to different types of queries. Each of the index types have been explained subsequently:

1. B-tree Indexes - B-tree is a self balancing tree that maintains sorted data and allows insertion, searching, deletions in logarithmic time. A B-tree index can improve performance when using equality and range queries on data that can be sorted into some ordering.
2. GiST Index - A hash index may improve performance when managing 2 dimensional geometric data types and nearest neighbor searches.
3. Hash Index - A hash index can improve performance when managing equality comparisons.
4. GIN Index - A GIN index may improve performance when managing values with more than one key.
5. Space-Partitioned GiST Index - SP-GiST may improve when managing non-balnced data structure.
6. BRIN Index - A BRIN index may improve performance when managing minimum and maximum values and ranges.

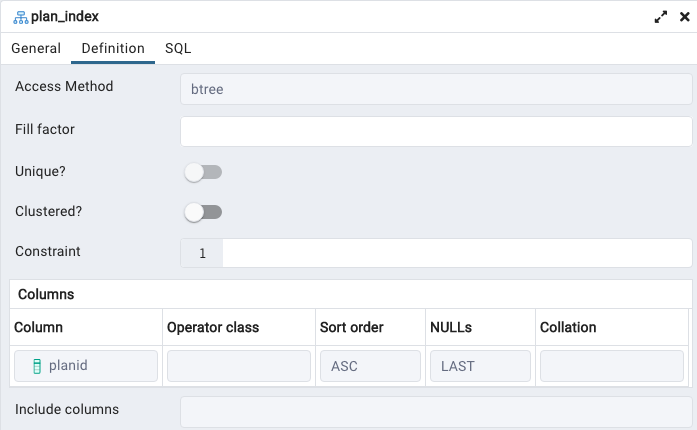
For our use-case a simple B-tree index has been used which is the default type of indexing in Postgres. It has been used because it optimizes the range queries apart from equality queries. Four different indexes have been created which have been explained subsequently.

**Employee Index**

****Figure: The Employee Table Index

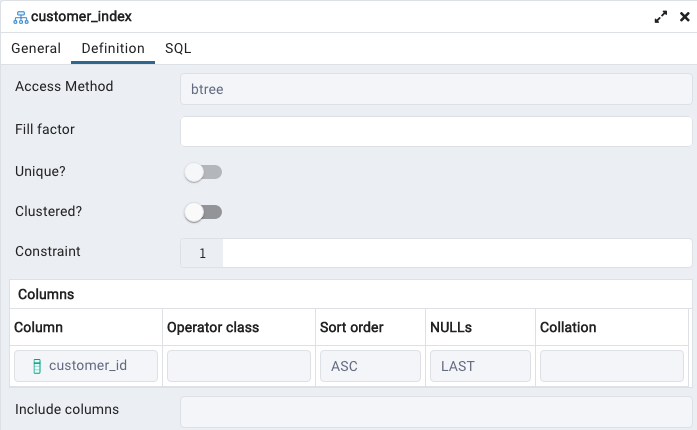
Here, an index has been created based on the employee\_id column. Employee\_id is the surrogate primary key of the Employee table which has been configured to start from 1 and auto-increment the value by 1 everytime. This index has been created because the table is used by address, claim transaction, claim progression and policy transaction. All the transaction and progression table will query the employee table based on the employee\_id which is the primary key of the table and used as a foreign key by the transaction and progression tables listed above. While querying the Employee table, if the id is sorted in ascending order, the lookup will be faster and more effective.

**Plan Index**

****Figure: The Plan Table Index

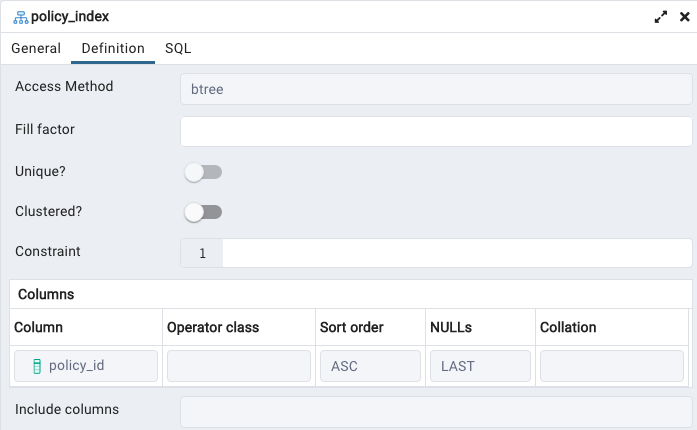
Here, an index has been created based on the plan\_id column. plan\_id is the surrogate primary key of the Plan table which has been configured to start from 1 and auto-increment the value by 1 every time. This index has been created because the table is used by Policy, profit loss snapshot, revenue snapshot and claim progression. All the snapshot and progression table will query the plan table based on the plan\_id which is the primary key of the table and used as a foreign key by the snapshot and progression tables listed above. While querying the Plan table, if the id is sorted in ascending order, the lookup will be faster and more effective.

**Customer Index**

****Figure: The Customer Table Index

Here, an index has been created based on the customer\_id column. Customer\_id is the surrogate primary key of the Customer table which has been configured to start from 1 and auto-increment the value by 1 every time. This index has been created because the table is used by claim transaction, claim progression and policy transaction. All the transaction and progression table will query the Customer table based on the customer\_id which is the primary key of the table and used as a foreign key by the transaction and progression tables listed above. While querying the Customer table, if the id is sorted in ascending order, the lookup will be faster and more effective.

**Policy Index**

****Figure: The Policy Table Index

Here, an index has been created based on the policy\_id column. policy\_id is the surrogate primary key of the Policy table which has been configured to start from 1 and auto-increment the value by 1 every time. This index has been created because the table is used by plan, claim transaction, claim progression and policy transaction. All the transaction and progression tables will query the Policy table based on the policy\_id which is the primary key of the table and used as a foreign key by the transaction and progression tables listed above. While querying the Customer table, if the id is sorted in ascending order, the lookup will be faster and more effective.

**Conclusion**

Overall this project turned out to be one of the most fruitful projects that we have done during our time at NYU. We didn’t have any experience with ML, and UI development but we were successfully able to do so. Through this project we realized the importance of having a good DB, Database Lake systems setup for enterprises. We can successfully say that we can now design an end to end application for an enterprise, at least partially from the experience that we gained through this project.

**REFERENCES**

1. Course Slides and Lectures. [Brightspace - Fall 2022 Database Systems, Section 001]
2. Postgres Cluster Guide. [<https://hevodata.com/learn/postgresql-cluster/#4>]
3. <https://www.kaggle.com/code/mysha1rysh/polynomial-regression-for-beginners>
4. <https://towardsdatascience.com/predicting-presence-of-heart-diseases-using-machine-learning-36f00f3edb2c>
5. <https://gradio.app/>
6. <https://www.sqlalchemy.org/>
7. https://github.com/sqlalchemy/sqlalchemy