**Solution Design Document**

**1. Solution**

The proposed solution involves the creation of data pipelines and analysis mechanisms using various technologies to enhance the revenue generation and customer understanding of the Health Care insurance company. The solution will follow these steps:

**1. Data Ingestion:**

* Competitor company data is collected from various sources and stored in an AWS S3 bucket named "input-data".
* Data can be collected through web scraping, third-party APIs, or other data sources relevant to the insurance industry.
* AWS S3 provides a scalable and cost-effective storage solution for raw data before further processing.

**2. Data Cleaning and Preprocessing**:

* Data cleaning modules are developed to handle data quality issues and inconsistencies in the collected data.
* Null values are identified and handled appropriately. For example, missing values can be imputed using mean, median, or other relevant methods.
* Duplicate records are identified and removed to avoid skewing analysis results.
* Data types are standardized, ensuring consistency across columns.
* In the case of the "Patients" dataset, the age column might be checked for outliers (e.g., negative values) and corrected.
* The cleaned data for each dataset is then uploaded into corresponding Redshift tables.

**3. Data Analysis and Result Generation:**

* Data analysis modules are implemented using Pyspark and Databricks to perform various analyses on the datasets.
* For example, to identify the disease with the maximum claims:

Join the "Patients" and "Claims" tables on patient ID.

Group the data by disease and calculate the count of claims for each disease.

Identify the disease with the maximum claim count.

* Similarly, other analyses are performed using appropriate Pyspark transformations, aggregations, and filtering operations.
* Results of each analysis are stored in separate Redshift tables within the "Project-Output" schema.

**4. Data Deployment:**

* The code will be deployed on AWS EMR with the help of GitHub.

**5. Use Case Customization:**

* The analyses and results are customized to address each use case defined in the project requirements.
* For instance, to find subscribers below 30 years subscribing to any subgroup:

Filter the "Subscriber" table to include only records where age is less than 30.

Identify subscribers who have subscribed to any subgroup using appropriate joins with "Group and subgroup" table.

Present the results listing these subscribers and their chosen subgroups.

* Each use case involves specific queries and data manipulations to provide insights tailored to the insurance company's goals.

**6. Business Strategies:**

* The insights generated from the analyses provide valuable information for the insurance company's business strategies.
* For example, identifying the most profitable group can help the company allocate resources and marketing efforts more effectively.
* Determining preferred policy types (government vs. private) guides the company's policy offerings.
* Calculating royalties for loyal customers can enhance customer engagement and revenue generation.

7. **Connecting to AWS S3:**

To connect to AWS S3, we will need AWS credentials and the details of the S3 bucket where our data is stored. Here's an outline of the process:

**AWS Credentials:**

* We will need an AWS Access Key ID and Secret Access Key with sufficient permissions to access the S3 bucket. These credentials are used for authentication.
* We will then create an IAM (Identity and Access Management) user with the necessary permissions and generate the access keys.

**S3 Bucket Details:**

* We will need to know the name of the S3 bucket where the data is stored.
* We will also specify the folder path within the bucket as our data is organized in subfolders.

**Connecting in Code:**

* In your code (Python), we will use AWS SDKs (such as boto3 for Python) to establish a connection to S3.
* We will then use the provided credentials to authenticate the connection and access the desired bucket and objects.

**8. Connecting to Redshift:**

To connect to AWS Redshift, we will need connection details and credentials. Then we will need the following process:

**Redshift Cluster Details:**

* We need to know the endpoint (URL) of your Redshift cluster and the port number used for communication.

**Database Credentials:**

* We will then need the database username and password that we'll use to connect to the Redshift cluster.

Connecting in Code:

* Use a database connection library to establish a connection to the Redshift cluster.
* Provide the cluster endpoint, port, database name, username, and password.

**9. Connecting to Databricks:**

Databricks provides a unified analytics platform that integrates with various data sources, including AWS services. Here's how we will connect:

Databricks Workspace:

* Access the Databricks workspace where we'll develop our data processing and analysis pipelines.

Cluster Setup:

* Create a Databricks cluster with appropriate configurations, including the necessary libraries and dependencies.

Notebook Development:

* Develop our data processing and analysis logic in Databricks notebooks.
* Databricks notebooks will allow us to write and execute code interactively using languages like Python, and SQL.

Connecting to AWS Services:

* Databricks can directly access data from AWS services like S3. We'll need to configure access permissions in Databricks to read data the data stored in our S3 bucket.

Redshift Integration:

* We can also connect Databricks to Redshift for analysis or data loading purposes.
* We will use the appropriate JDBC drivers along with Redshift cluster details and credentials to establish a connection.

**10. Connecting to AWS EMR:**

Amazon EMR (Elastic MapReduce) is used for processing large datasets using frameworks like Hadoop and Spark. We need to make sure of following process before we can process our data.

EMR Cluster Setup:

* We will set up an EMR cluster with the appropriate configuration for our data processing needs.
* We can choose the Spark framework for processing as we will be using Pyspark.

Connecting to EMR in Code:

* Since we are using Python, we can use pyspark to connect to the EMR cluster and execute Spark jobs.

**11. Connecting and Pushing/Pulling in GitHub:**

Here's how to connect to GitHub and manage code in the master branch:

Connecting to GitHub:

* We will create a GitHub account if you don't have one.
* We will then install Git on our local machine.
* We can then clone the repository using the repository's URL. Run: git clone <repository\_url>

Pushing Changes:

* After making changes to our code, use git add <filename> to stage the changes.
* We can run git commit -m "Your commit message" to commit our changes locally.
* Finally, we can use git push origin master to push our changes to the master branch on GitHub.

Pulling Changes:

* Before making any changes, it's a good practice to pull the latest changes from the remote repository. Run: git pull origin master
* This ensures we're working with the most up-to-date code.

**2. Use Cases**

The solution will be applicable to the following use cases:

* Identify disease with maximum claims.
* Find subscribers below 30 years subscribing to any subgroup.
* Determine the group with the most subgroups.
* Identify the hospital serving the most patients.
* Determine the subgroups with the highest subscription frequency.
* Count total number of rejected claims.
* Identify the city with the highest claim count.
* Determine whether subscribers prefer government or private policies.
* Calculate average monthly premium paid by subscribers.
* Identify the most profitable group.
* List patients under 18 admitted for cancer.
* List patients with cashless insurance and charges ≥ Rs. 50,000.
* List female patients over 40 who had knee surgery in the past year.

**3. Database Design**

**Tables Metadata Info:**

* **Patients:** Patient information including patient ID (PK), patient\_name, patient\_gender, patient\_birth\_date, patient\_phone, disease\_name, city, hospital\_id (FK).
* **Disease:** Details of diseases with Subgrp\_ID (FK), ???, and disease\_name.
* **Claims:** Claims information including claim ID (PK), patient ID (FK), disease\_name, SUB\_ID (FK), claim\_Or\_Rejected, claim\_type, claim\_amount and claim\_date.
* **Group**: Information about policy groups and subgroups with Country, ???, ??? group\_ID(PK), policy\_name, policy\_type, City, policy\_date.
* **Hospitals:** Hospital information including Hospital\_id (PK), Hospital\_name, city, state, country.
* **Subgroup:** Information regarding patients including Subgrp\_ID (PK), disease???, ???, group\_ID(FK)
* **Subscriber**: Subscriber information including sub\_ID (PK), first\_name, last\_name, Street, Birth\_date, Gender, Phone, Country, City, Zip Code, Subgrp\_id (FK), Elig\_ind, eff\_date, term\_date

**ER Diagram: (Optional)**

A black screen with white text

Description automatically generated

**4. Technologies and Platforms**

The solution will utilize the following technologies and platforms:

AWS Services: S3 for data storage, Redshift for data warehousing, EMR for data processing.

Databricks: Unified analytics platform for data processing, analysis and data visualization.

Pyspark: Python-based framework for data manipulation and analysis.

Jira: Project management and issue tracking.

GitHub: Version control and code repository.

Gmail: To communicate to team members and managers.

Microsoft Teams : To have meetings about the project.

Draw.io : To create the ER Class Diagram