**Solution Design Document**

**1. Solution**

The solution involves creating an end-to-end data pipeline to analyze competitor data for the Health Care insurance company. The following steps outline the solution approach:

**Step-by-Step Solution**

1. **Data Ingestion**:
   * Gather competitor data from various sources, including web scraping and third-party providers.
   * Upload the raw data files to AWS S3 in a folder named "input-data" for secure storage and easy access by other AWS services.
2. **Data Cleaning and Transformation**:
   * Load data from AWS S3 into Databricks, where PySpark scripts will handle data cleaning and transformation.
   * Perform data cleaning steps:
     + Identify and handle null values by replacing them with “NA” where applicable.
     + Detect and remove duplicate records to ensure data accuracy.
     + Standardize data formats across datasets to ensure consistency.
   * Save the cleaned data into Redshift tables, with one table per dataset (Patients, Subscriber, Claims, Group\_subgroup).
3. **Schema Design and Data Storage in Redshift**:
   * Design a Redshift schema to store the cleaned datasets with primary and foreign key relationships, ensuring efficient querying and data integrity.
   * Upload the cleaned data from Databricks to the appropriate tables in Redshift.
4. **Result Generation in Redshift**:
   * Implement queries in Redshift to produce the required insights:
     + E.g., identify disease with maximum claims, hospitals with the highest patient volume, etc.
   * Create individual Redshift tables to store the results for each use case. All tables are part of the "Project-Output" schema for easy accessibility and data retrieval.
5. **Data Analysis and Visualization**:
   * Use Databricks for visual analysis of Redshift results, creating data visualizations and summaries of key insights.
   * Generate visual snapshots to share insights with stakeholders, enhancing decision-making for business strategies.
6. **Deployment and Code Management**:
   * Deploy PySpark scripts and Redshift queries on AWS EMR for scalable, production-level processing.
   * Use GitHub for version control and collaboration, with JIRA tracking for sprint management.
   * The solution can be executed in Databricks for testing and migrated to EMR for production.
7. **Testing and Documentation**:
   * Develop test cases for each use case to validate data accuracy and pipeline functionality.
   * Store documentation, including solution design and test cases, in GitHub.

**2. Use Cases**

The solution addresses the following use cases for the Health Care insurance company:

1. **Identify the Most Profitable Policy Group**:
   * Analyze historical data to determine which policy groups generate the highest revenue relative to claims costs. This insight can guide the company in promoting and expanding profitable policy offerings and adjusting premiums or benefits for underperforming groups.
2. **Analyze Disease Patterns by Claims Volume**:
   * Identify the diseases with the highest number of claims to understand common health challenges among policyholders. This data can help the company design targeted health programs, develop disease-specific policies, or negotiate better rates with healthcare providers for high-demand treatments.
3. **Assess Subscriber Demographics for Customized Marketing**:
   * Identify specific demographic groups (e.g., subscribers under age 30 who belong to certain subgroups) to tailor marketing campaigns, offering them relevant policies and benefits. This customization helps the company attract and retain younger customers with more personalized product offerings.
4. **Determine Most Frequent Cities for Claims**:
   * Analyze which cities generate the highest volume of claims to identify geographic trends. This data can be used for regional marketing strategies, creating city-specific health initiatives, or collaborating with local healthcare providers for better services in high-claim areas.
5. **Track Rejected Claims for Process Improvement**:
   * Examine the total number of rejected claims, identifying patterns or common reasons for rejections. This analysis can improve customer satisfaction by streamlining the claims process, reducing rejection rates, and addressing specific customer service or documentation gaps.

**3. Database Design**

All datasets will be stored in Redshift tables, structured to maintain data integrity and efficient query performance.

**Tables and Metadata Info with PK/FK Relationships**

The primary tables for this solution in Redshift include:

1. **Patients**:
   * **Columns**: patient\_id (PK), age, gender, disease, admission\_date, hospital\_id, insurance\_type, total\_charges
   * **Relationships**: Links to Claims table (patient\_id)
2. **Subscribers**:
   * **Columns**: subscriber\_id (PK), age, gender, group\_id, subscription\_date
   * **Relationships**: Links to Claims table (subscriber\_id), Group\_subgroup table (group\_id)
3. **Claims**:
   * **Columns**: claim\_id (PK), patient\_id (FK), subscriber\_id (FK), claim\_date, claim\_status, disease, city, amount
   * **Relationships**: Links to Patients and Subscribers tables
4. **Group\_subgroup**:
   * **Columns**: group\_id (PK), subgroup\_id (PK), subscription\_count
   * **Relationships**: Links to Subscribers table (group\_id)
5. **Hospitals**:
   * **Columns**: hospital\_id (PK), name, location, patient\_count
   * **Relationships**: Links to Patients table (hospital\_id)
6. **Output Tables (Project-Output schema)**:
   * For each use case, a table will be created to store specific results (e.g., most profitable group, most subscribed subgroups, etc.)

**4. Technologies and Platforms**

The following technologies and platforms will be utilized for this solution:

1. **Data Storage**:
   * **AWS S3**: For storing raw input data files and staging for Redshift.
   * **AWS Redshift**: For structured data storage and query processing, optimized for analytical workloads.
2. **Data Processing**:
   * **Databricks**: For data cleaning, transformation, and testing the data pipeline.
   * **AWS EMR**: For scalable, production-grade data processing using PySpark.
3. **Data Transformation and Analysis**:
   * **PySpark**: For writing scripts to clean and process data and implement ETL logic.
4. **Project Management and Collaboration**:
   * **JIRA**: For sprint planning, tracking tasks, and managing user stories.
   * **GitHub**: For source code management, documentation, and version control.
5. **Data Visualization**:
   * **Databricks**: For visualizing and presenting analytical results to stakeholders.