Project Title : Improving Data Integration Quality For Multi-Source Analytics

Team Members:

1. **Name : Mrudula.R**
 ID:CAN_33718403
2. **Name: M Spurthi**
 ID:CAN_33717842
3. **Name: N ArunKumar**
 ID:CAN_33717217
4. **Name: A H Bhushieth**
 ID:CAN_33718570

Institution Name :Vemana Institute Of Technology

Phase 2 -Solution Architecture

1. Overview of Solution Design and System Architecture

Objective: The goal of Phase 2 is to define the architecture and implement the solution for multi-source data integration, ensuring high-quality, scalable, and real-time data processing. This phase will focus on system design, the creation of ETL pipelines, and ensuring data quality through robust transformation techniques.

- **Key Actions:**
  - Finalize the data integration approach using a unified schema.
  - Develop real-time and batch processing solutions.
  - Establish data quality monitoring tools.

2. System Architecture and Tools Selection

Overview of Architecture: The solution architecture will integrate various data sources into a unified system, leveraging cloud-native tools for scalability and efficiency. The architecture will include the following components:

- **ETL Pipelines:** Using tools like Apache Spark or AWS Glue for data ingestion, transformation, and loading.
- **Real-Time Integration:** Kafka or Flink for handling streaming data sources.
- **Metadata Management:** Implementing tools like Apache Atlas for tracking data lineage.
- **Data Quality Monitoring:** Employing tools like Great Expectations to validate data integrity and completeness.

3. Data Integration and Transformation Design

Data Ingestion and Transformation:

- **Data Sources:** Integrate data from multiple sources such as databases, APIs, and flat files.
- **Schema Unification:** Develop a standardized schema to ensure data consistency across systems.
- **Data Transformation:** Use Python and Spark to clean, remove duplicates, and standardize data formats.

Python Code for Data Transformation:

The following Python function demonstrates how to integrate data from multiple sources while ensuring high-quality transformation and consistency:

```
import pandas as pd

import numpy as np
```

```
def improve_data_integration_quality(dataframes, primary_key_columns):
    """
    Improves data integration quality for multi-source analytics.

    Args:
        dataframes: A list of pandas DataFrames to integrate.
        primary_key_columns: A list of column names to use as primary keys for joins.

    Returns:
        A pandas DataFrame representing the integrated dataset, or None if an error occurs.
    """

    # Check if inputs are valid
    if not isinstance(dataframes, list) or not all(isinstance(df, pd.DataFrame) for df in dataframes):
        print("Error: Invalid input. 'dataframes' must be a list of pandas DataFrames.")
        return None

    if not isinstance(primary_key_columns, list) or not all(isinstance(col, str) for col in
primary_key_columns):
        print("Error: Invalid input. 'primary_key_columns' must be a list of strings.")
        return None

    # Handling Missing Values
    for df in dataframes:
        for col in primary_key_columns:
```

```
            if col in df.columns:
                df[col] = df[col].astype(str).fillna('missing_key')

    # Standardizing Data Types
    for df in dataframes:
        for col in df.columns:
            if df[col].dtype == 'object':
                df[col] = df[col].astype(str).str.strip().str.lower()

    # Deduplication
    for df in dataframes:
        df.drop_duplicates(subset=primary_key_columns, inplace=True, keep='first')

    # Joining DataFrames
    merged_df = dataframes[0]
    for i in range(1, len(dataframes)):
        try:
            merged_df = pd.merge(merged_df, dataframes[i], on=primary_key_columns, how='outer')
        except KeyError as e:
            print(f"Warning: Could not merge based on all specified keys due to missing columns: {e}")
            missing_columns = set(primary_key_columns) - set(dataframes[i].columns)
            print("Missing Columns", missing_columns)
            continue # Skip merging if error occurred
```

```
    # Handling Inconsistent Values
    if "country" in merged_df.columns:
        country_mapping = {"united states": "USA", "us": "USA", "america": "USA"}
        merged_df['country'] = merged_df['country'].replace(country_mapping)

    # Removing duplicates after merges
    merged_df.drop_duplicates(subset=primary_key_columns, inplace=True, keep="first")

    return merged_df

# Example Usage:
data1 = {'id': [1, 2, 3], 'value1': ['A', 'B', 'C'], 'country': ['us', 'united states', 'canada']}
data2 = {'id': [2, 3, 4], 'value2': [10, 20, 30], 'country': ['United States', 'Canada', 'Mexico']}
df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)

integrated_df = improve_data_integration_quality([df1, df2], ['id'])
print(integrated_df)
```

ETL Pipeline Implementation:

- Design and implement the ETL pipeline for batch processing.
- Implement real-time data integration pipelines for continuous ingestion.

4. Data Quality Assurance and Testing

Quality Metrics:

- **Accuracy:** Ensuring data is correctly ingested and processed.
- **Timeliness:** Verifying real-time data integration and batch processing performance.
- **Completeness:** Using automated checks to ensure no data is missed during transformation.

Automated Data Quality Checks:

- Implement Python-based scripts for routine checks on data quality, using tools like Pandas or Great Expectations for automated testing.
- Develop validation reports for continuous monitoring.

5. Scalability and Performance Testing

Stress Testing:

- Test the system under heavy data loads to ensure the ETL pipelines can scale.
- Verify performance of data ingestion and transformation processes under different scenarios.

6. Solution Implementation and Deployment

Cloud Infrastructure:

- Use cloud-based platforms like AWS, Azure, or GCP for hosting and managing data pipelines.
- Ensure the system is scalable to accommodate future growth in data volume.

Deployment Strategy:

- Develop a deployment framework for moving from development to production environments.
- Ensure seamless integration with business systems and analytics platforms.

7. Feasibility Assessment and Evaluation

Initial Testing:

- Validate the implemented solution with sample datasets to confirm data quality.
- Assess system performance in live environments.

Metrics for Future Evaluation:

- Precision and recall to assess data accuracy.
- Evaluate the system using business KPIs like operational efficiency and the time-to-insight.

8. Conclusion

Phase 2 will establish the core infrastructure for multi-source data integration, ensuring the quality and scalability required for future analytical applications.