Project Name: ETL pipeline Designing with Talend

Guided By

Sai Prakash

Team Members

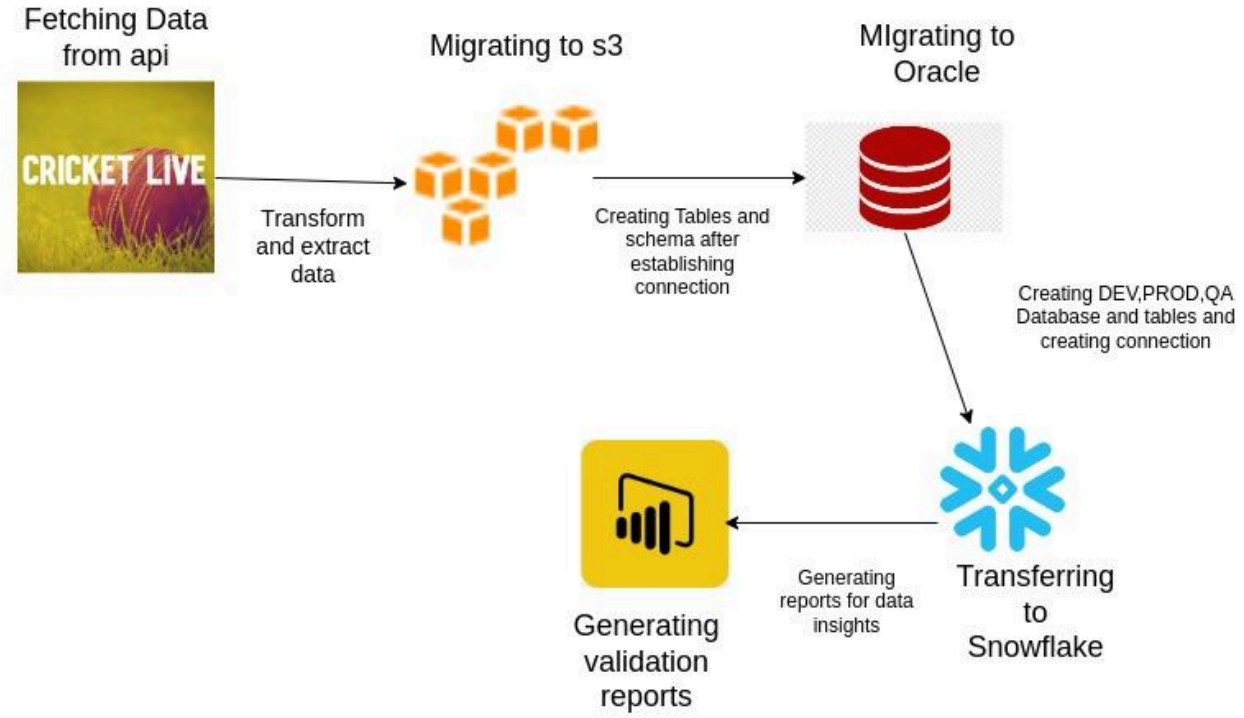
Jagruti Airao
Jatan Sahu
Jigar Shekhat
Priyansh Agarwal
Purvang Maheria
Vipasha Vaghela
Vivek Soni

Table of Contents

Abstract:

This project aims to streamline the data management process through the implementation of various stages encompassing data modeling, acquisition, integration, migration, and validation. Initially, a data model is constructed utilizing the Start/Snowflake Schema, providing a foundation for structured data organization. Subsequently, cricket data is obtained from external sources via RAPID API and stored in Amazon S3 utilizing Python, with scheduled data acquisition ensuring timely updates. The acquired data is then transferred to Oracle utilizing Talend for further processing and analysis. Following this, a migration process is employed to transition data from Oracle to Snowflake, leveraging Talend's capabilities for seamless data migration. Finally, to ensure data integrity and reliability, a comprehensive data validation process is established, complemented by the generation of sample reports using Power BI, facilitating insightful data analysis and decision-making. Through the integration of these components, this project endeavors to optimize data management practices, enabling organizations to harness the full potential of their data assets.

Project Architecture:



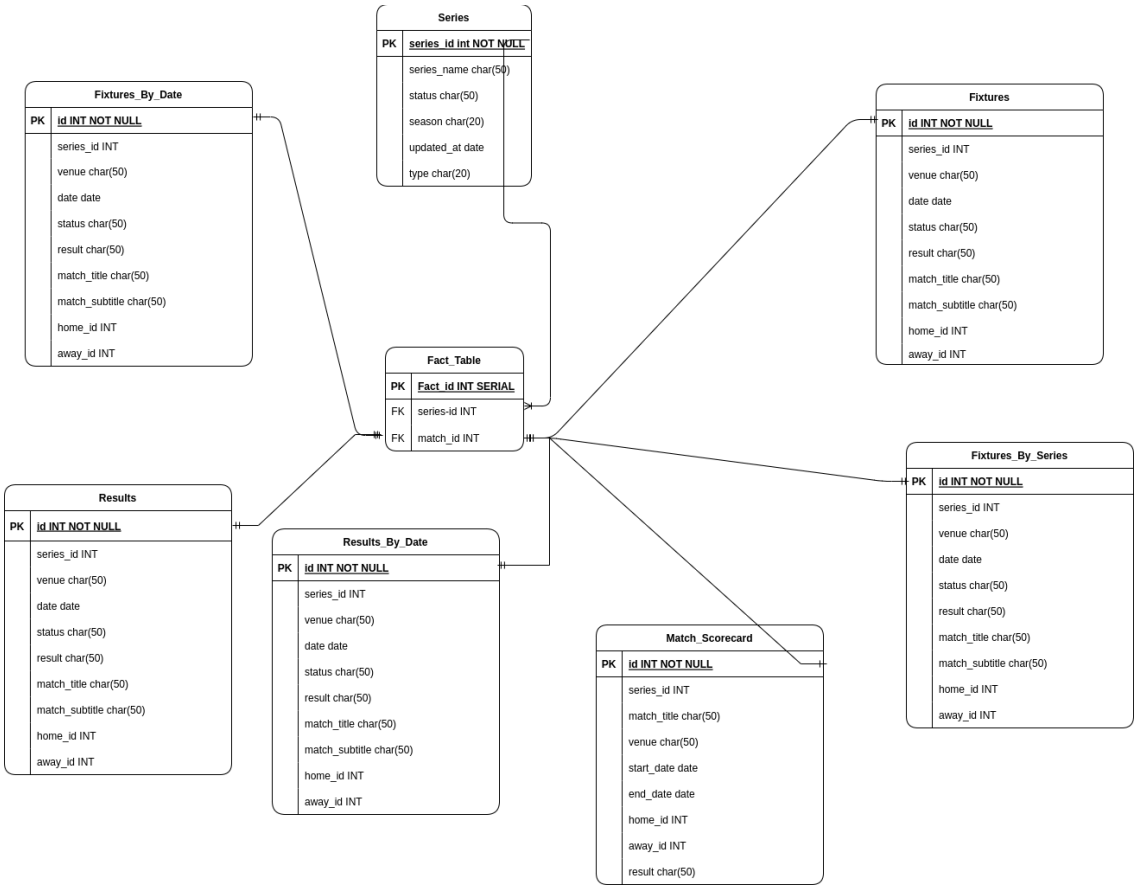
Methodology:

1. Prepare Data model (Star/ Snowflake Schema) with ER Diagram - Jatan and Priyansh

Star and snowflake schemas are both widely used in data warehousing to organize and structure relational databases for efficient querying and analysis. They have some similarities but differ in their structure and complexity.

Star schemas feature a denormalized structure where dimension tables are fully denormalized, resulting in simpler schema designs with fewer tables and straightforward queries. In contrast, snowflake schemas follow a normalized approach where dimension tables are partially or fully normalized into multiple related tables, leading to a more complex schema with additional joins required for querying. Consequently, while star schemas offer simplicity and faster query performance, snowflake schemas prioritize data integrity and storage efficiency at the cost of increased query complexity.

Star Schema :

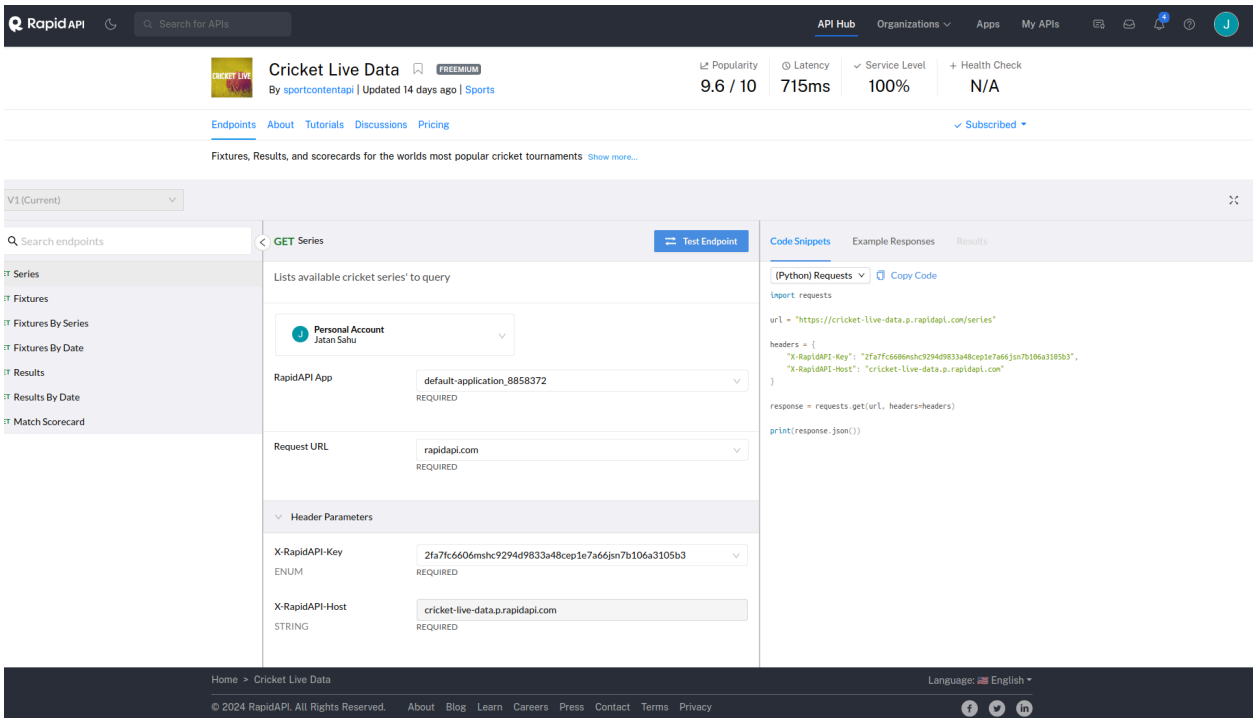


2. Get Data from 3rd Party with API and send to s3 with Python and schedule it - Jatan and Priyansh

Getting data from API

For this we have used RAPID API as a third party source to gather the data using python.

[Click to view data source]



As we are using free version of RAPIDAPI we have very limited access(250 requests only) so we have requested limited number of calls and ingested less amount of data.

For sending this data to S3 bucket we have written python code. As we have 7 tables and one which is venue table which is a nested file which contains venue id and the team name and team code. So we have created total 7 python files and 8 csv file, send all these structured (.csv) data into s3 bucket.

The data is in JSON format we have also done required preprocessing task for all the tables. **Python Code** Snippets [Click to view code]

1.Series

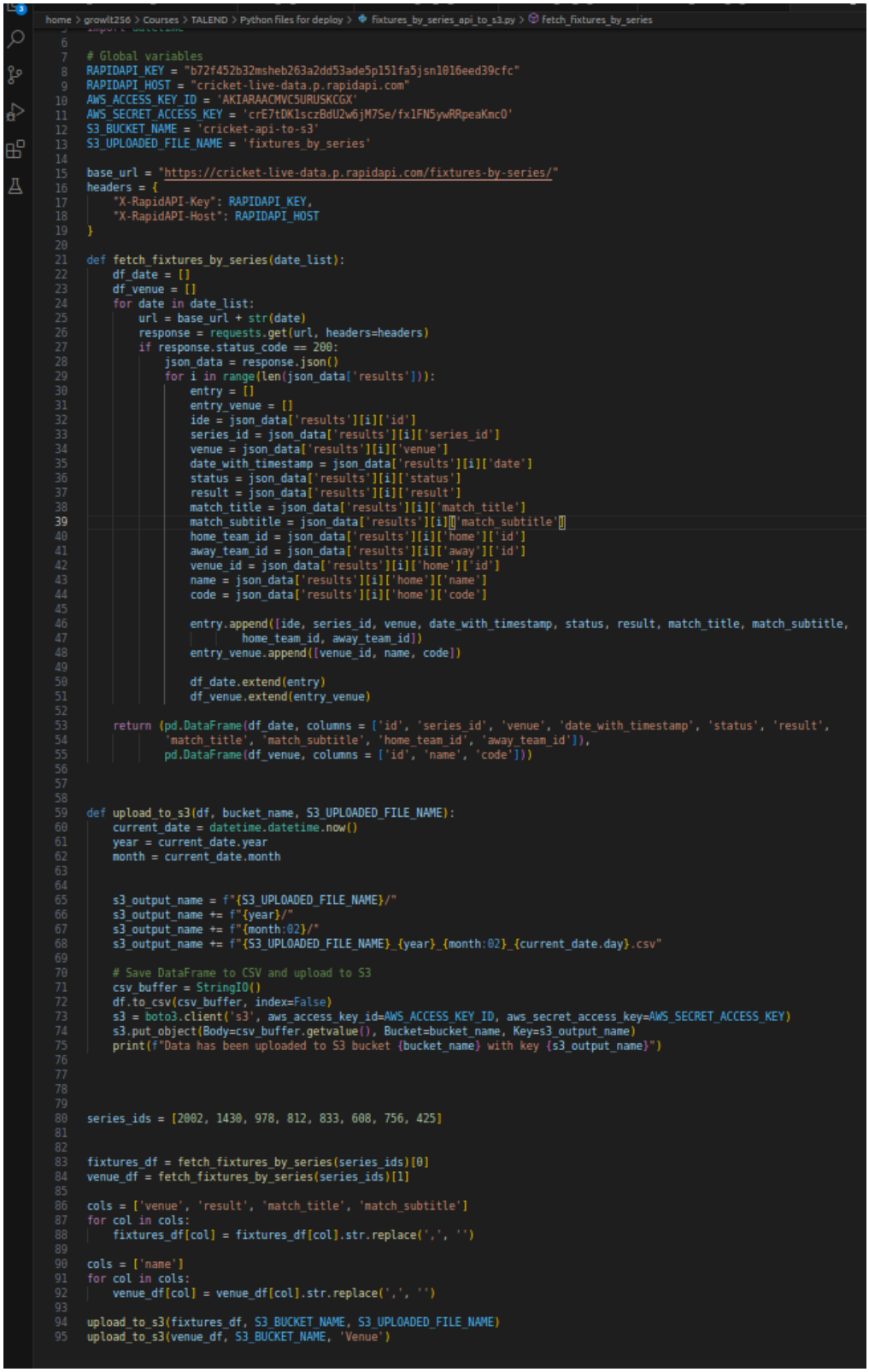

home > growlt256 > Courses > TALEND > Python files for deploy > series_upload_to_s3.py > upload_to_s3
import pandas as pd
import requests
import boto3
from io import StringIO
import datetime
# Global variables
RAPIDAPI_KEY = "b72f452b32msheb263a2dd53ade5p151fa5jsn1016eed39cfc"
RAPIDAPI_HOST = "cricket-live-data.p.rapidapi.com"
AWS_ACCESS_KEY_ID = 'AKIARAACMVC5URUSKCGX'
AWS_SECRET_ACCESS_KEY = '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'
S3_BUCKET_NAME = 'cricket-api-to-s3'
S3_UPLOADED_FILE_NAME = 'series'
# API endpoint function
def fetch_data_from_api(url, headers):
response = requests.get(url, headers=headers)
if response.status_code == 200:
return response.json()
else:
print("Failed to fetch data:", response.status_code)
return None
# Function to process data
def process_data(json_data):
all_series_data = []
for result in json_data['results']:
series_data = result['series']
series_type = result['type']
# Add 'type' to each series data
for data in series_data:
data['type'] = series_type
all_series_data.extend(series_data)
return pd.DataFrame(all_series_data)
# # Function to save DataFrame to CSV and upload to S3
# def upload_to_s3(df, bucket_name, s3_output_name):
# csv_buffer = StringIO()
# df.to_csv(csv_buffer, index=False)
# s3 = boto3.client('s3', aws_access_key_id=AWS_ACCESS_KEY_ID, aws_secret_access_key=AWS_SECRET_ACCESS_KEY)
# s3.put_object(Body=csv_buffer.getvalue(), Bucket=bucket_name, Key=s3_output_name)
# print(f"Data has been uploaded to S3 bucket {bucket_name} with key {s3_output_name}")
def upload_to_s3(df, bucket_name, folder):
# Get current year and month
current_date = datetime.datetime.now()
year = current_date.year
month = current_date.month
# Construct S3 key
s3_output_name = f"{folder}/"
s3_output_name += f"{year}/"
s3_output_name += f"{month:02}/"
s3_output_name += f"{folder}_{year}_{month:02}_{current_date.day}.csv"
# Save DataFrame to CSV and upload to S3
csv_buffer = StringIO()
df.to_csv(csv_buffer, index=False)
s3 = boto3.client('s3', aws_access_key_id=AWS_ACCESS_KEY_ID, aws_secret_access_key=AWS_SECRET_ACCESS_KEY)
s3.put_object(Body=csv_buffer.getvalue(), Bucket=bucket_name, Key=s3_output_name)
print(f"Data has been uploaded to S3 bucket {bucket_name} with key {s3_output_name}")
# Main function
def main():
url = "https://cricket-live-data.p.rapidapi.com/series"
headers = {
"X-RapidAPI-Key": RAPIDAPI_KEY,
"X-RapidAPI-Host": RAPIDAPI_HOST
}
json_data = fetch_data_from_api(url, headers)
if json_data:
df_series = process_data(json_data)
cols = ['series_name', 'status']
for col in cols:
df_series[col] = df_series[col].str.replace(',', '')
upload_to_s3(df_series, S3_BUCKET_NAME, S3_UPLOADED_FILE_NAME)
else:
print("No data fetched from the API.")
if __name__ == "__main__":
main()

2. Fixtures

home > growlt256 > Courses > TALEND > Python files for deploy > fixtures_api_to_s3.py > ...

```python
import pandas as pd
import requests
import boto3
from io import StringIO
import datetime

# Global variables
RAPIDAPI_KEY = "f381eaaa79msh51537dd6e014035p180695jsn67cf7d549118"
RAPIDAPI_HOST = "cricket-live-data.p.rapidapi.com"
AWS_ACCESS_KEY_ID = 'AKIARAACMVC5URUSKCGX'
AWS_SECRET_ACCESS_KEY = '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'
S3_BUCKET_NAME = 'cricket-api-to-s3'
S3_UPLOADED_FILE_NAME = 'fixtures'

def fetch_data_from_api(url):
    headers = {
        "X-RapidAPI-Key": RAPIDAPI_KEY,
        "X-RapidAPI-Host": RAPIDAPI_HOST
    }
    response = requests.get(url, headers=headers)
    if response.status_code == 200:
        json_data = response.json()
        df_date = []
        for i in range(len(json_data['results'])):
            entry = []
            ide = json_data['results'][i]['id']
            series_id = json_data['results'][i]['series_id']
            venue = json_data['results'][i]['venue']
            date_with_timestamp = json_data['results'][i]['date']
            status = json_data['results'][i]['status']
            result = json_data['results'][i]['result']
            match_title = json_data['results'][i]['match_title']
            match_subtitle = json_data['results'][i]['match_subtitle']
            home_team_id = json_data['results'][i]['home']['id']
            away_team_id = json_data['results'][i]['away']['id']
            entry.append([ide, series_id, venue, date_with_timestamp, status, result, match_title, match_subtitle,
                    home_team_id, away_team_id])
            df_date.extend(entry)
        return pd.DataFrame(df_date, columns = ['id', 'series_id', 'venue', 'date_with_timestamp', 'status', 'result',
                                                'match_title', 'match_subtitle', 'home_team_id', 'away_team_id'])
    else:
        print("Failed to fetch data from:", url)
        return None

# Function to save DataFrame to CSV and upload to S3
def upload_to_s3(df, bucket_name, folder):
    current_date = datetime.datetime.now()
    year = current_date.year
    month = current_date.month

    s3_output_name = f"{folder}/"
    s3_output_name += f"{year}/"
    s3_output_name += f"{month:02}/"
    s3_output_name += f"{folder}_{year}_{month:02}_{current_date.day}.csv"


    csv_buffer = StringIO()
    df.to_csv(csv_buffer, index=False)
    s3 = boto3.client('s3', aws_access_key_id=AWS_ACCESS_KEY_ID, aws_secret_access_key=AWS_SECRET_ACCESS_KEY)
    s3.put_object(Body=csv_buffer.getvalue(), Bucket=bucket_name, Key=s3_output_name)
    print(f"Data has been uploaded to S3 bucket {bucket_name} with key {s3_output_name}")

fixtures_df = fetch_data_from_api("https://cricket-live-data.p.rapidapi.com/fixtures")
cols = ['venue', 'result', 'match_title', 'match_subtitle']
for col in cols:
    fixtures_df[col] = fixtures_df[col].str.replace(',', '')
if fixtures_df is not None:
    upload_to_s3(fixtures_df, S3_BUCKET_NAME, S3_UPLOADED_FILE_NAME)
```

3.Fixtures_by_series:



```
home > growlt256 > Courses > TALEND > Python files for deploy > fixtures_by_series_api_to_s3.py > fetch_fixtures_by_series

# Global variables
RAPIDAPI_KEY = "b72f452b32msheb263a2dd53ade5p151fa5jsn1016eed39cfc"
RAPIDAPI_HOST = "cricket-live-data.p.rapidapi.com"
AWS_ACCESS_KEY_ID = 'AKIARAACMVC5URUSKCGX'
AWS_SECRET_ACCESS_KEY = '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'
S3_BUCKET_NAME = 'cricket-api-to-s3'
S3_UPLOADED_FILE_NAME = 'fixtures_by_series'

base_url = "https://cricket-live-data.p.rapidapi.com/fixtures-by-series/"
headers = {
    "X-RapidAPI-Key": RAPIDAPI_KEY,
    "X-RapidAPI-Host": RAPIDAPI_HOST
}

def fetch_fixtures_by_series(date_list):
    df_date = []
    df_venue = []
    for date in date_list:
        url = base_url + str(date)
        response = requests.get(url, headers=headers)
        if response.status_code == 200:
            json_data = response.json()
            for i in range(len(json_data['results'])):
                entry = []
                entry_venue = []
                ide = json_data['results'][i]['id']
                series_id = json_data['results'][i]['series_id']
                venue = json_data['results'][i]['venue']
                date_with_timestamp = json_data['results'][i]['date']
                status = json_data['results'][i]['status']
                result = json_data['results'][i]['result']
                match_title = json_data['results'][i]['match_title']
                match_subtitle = json_data['results'][i]['match_subtitle']
                home_team_id = json_data['results'][i]['home']['id']
                away_team_id = json_data['results'][i]['away']['id']
                venue_id = json_data['results'][i]['home']['id']
                name = json_data['results'][i]['home']['name']
                code = json_data['results'][i]['home']['code']

                entry.append([ide, series_id, venue, date_with_timestamp, status, result, match_title, match_subtitle,
                        home_team_id, away_team_id])
                entry_venue.append([venue_id, name, code])

                df_date.extend(entry)
                df_venue.extend(entry_venue)

    return (pd.DataFrame(df_date, columns = ['id', 'series_id', 'venue', 'date_with_timestamp', 'status', 'result',
            'match_title', 'match_subtitle', 'home_team_id', 'away_team_id']),
            pd.DataFrame(df_venue, columns = ['id', 'name', 'code']))



def upload_to_s3(df, bucket_name, S3_UPLOADED_FILE_NAME):
    current_date = datetime.datetime.now()
    year = current_date.year
    month = current_date.month


    s3_output_name = f"{S3_UPLOADED_FILE_NAME}/"
    s3_output_name += f"{year}/"
    s3_output_name += f"{month:02}/"
    s3_output_name += f"{S3_UPLOADED_FILE_NAME}_{year}_{month:02}_{current_date.day}.csv"

    # Save DataFrame to CSV and upload to S3
    csv_buffer = StringIO()
    df.to_csv(csv_buffer, index=False)
    s3 = boto3.client('s3', aws_access_key_id=AWS_ACCESS_KEY_ID, aws_secret_access_key=AWS_SECRET_ACCESS_KEY)
    s3.put_object(Body=csv_buffer.getvalue(), Bucket=bucket_name, Key=s3_output_name)
    print(f"Data has been uploaded to S3 bucket {bucket_name} with key {s3_output_name}")




series_ids = [2002, 1430, 978, 812, 833, 608, 756, 425]


fixtures_df = fetch_fixtures_by_series(series_ids)[0]
venue_df = fetch_fixtures_by_series(series_ids)[1]

cols = ['venue', 'result', 'match_title', 'match_subtitle']
for col in cols:
    fixtures_df[col] = fixtures_df[col].str.replace(',', '')

cols = ['name']
for col in cols:
    venue_df[col] = venue_df[col].str.replace(',', '')

upload_to_s3(fixtures_df, S3_BUCKET_NAME, S3_UPLOADED_FILE_NAME)
upload_to_s3(venue_df, S3_BUCKET_NAME, 'Venue')
```

4.Fixtures_by_date


home > growlt256 > Courses > TALEND > Python files for deploy > Fixtures_by_date_api_to_s3.py > ...
import pandas as pd
import requests
import boto3
from io import StringIO
import datetime
# Global variables
RAPIDAPI_KEY = "b72f452b32msheb263a2dd53ade5p151fa5jsn1016eed39cfc"
RAPIDAPI_HOST = "cricket-live-data.p.rapidapi.com"
AWS_ACCESS_KEY_ID = 'AKIARAACMVC5URUSKCGX'
AWS_SECRET_ACCESS_KEY = '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'
S3_BUCKET_NAME = 'cricket-api-to-s3'
S3_FOLDER_NAME = 'fixtures_by_date'
# API endpoint base URL and headers
base_url = "https://cricket-live-data.p.rapidapi.com/fixtures-by-date/"
headers = {
"X-RapidAPI-Key": RAPIDAPI_KEY,
"X-RapidAPI-Host": RAPIDAPI_HOST
}
arr_date = ['2023-04-02', '2023-04-03', '2023-04-04', '2023-04-05', '2024-03-22', '2024-03-23', '2024-03-23',
'2024-03-24', '2024-03-24', '2024-03-25', '2024-03-26', '2024-03-27', '2024-03-28', '2024-03-29',
'2024-03-30', '2024-03-31', '2024-03-31', '2024-04-01', '2024-04-02', '2024-04-03']
def fetch_fixtures_by_date(arr_date):
df_date = []
for date in arr_date:
url = base_url + date
response = requests.get(url, headers=headers)
if response.status_code == 200:
json_data = response.json()
for i in range(len(json_data['results'])):
entry = []
ide = json_data['results'][i]['id']
series_id = json_data['results'][i]['series_id']
venue = json_data['results'][i]['venue']
date_with_timestamp = json_data['results'][i]['date']
status = json_data['results'][i]['status']
result = json_data['results'][i]['result']
match_title = json_data['results'][i]['match_title']
match_subtitle = json_data['results'][i]['match_subtitle']
home_team_id = json_data['results'][i]['home']['id']
away_team_id = json_data['results'][i]['away']['id']
entry.append([ide, series_id, venue, date_with_timestamp, status, result, match_title, match_subtitle,
home_team_id, away_team_id])
df_date.extend(entry)
return pd.DataFrame(df_date, columns = ['id', 'series_id', 'venue', 'date_with_timestamp', 'status', 'result',
'match_title', 'match_subtitle', 'home_team_id', 'away_team_id'])
# Function to save DataFrame to CSV and upload to S3
def upload_to_s3(df, bucket_name, folder, file_name):
current_date = datetime.datetime.now()
year = current_date.year
month = current_date.month
s3_output_name = f"{folder}/{year}/{month:02}/{file_name}_{year}_{month:02}_{current_date.day}.csv"
csv_buffer = StringIO()
df.to_csv(csv_buffer, index=False)
s3 = boto3.client('s3', aws_access_key_id=AWS_ACCESS_KEY_ID, aws_secret_access_key=AWS_SECRET_ACCESS_KEY)
s3.put_object(Body=csv_buffer.getvalue(), Bucket=bucket_name, Key=s3_output_name)
print(f"Data has been uploaded to S3 bucket {bucket_name} with key {s3_output_name}")
df_fixtures_by_date = fetch_fixtures_by_date(arr_date)
# upload_to_s3(df_fixtures_by_date, S3_BUCKET_NAME, S3_FOLDER_NAME, S3_FOLDER_NAME)
cols = ['venue', 'status', 'result', 'match_title', 'match_subtitle']
for col in cols:
df_fixtures_by_date[col] = df_fixtures_by_date[col].str.replace(',', '')
upload_to_s3(df_fixtures_by_date, S3_BUCKET_NAME, S3_FOLDER_NAME, S3_FOLDER_NAME)

5.Result:

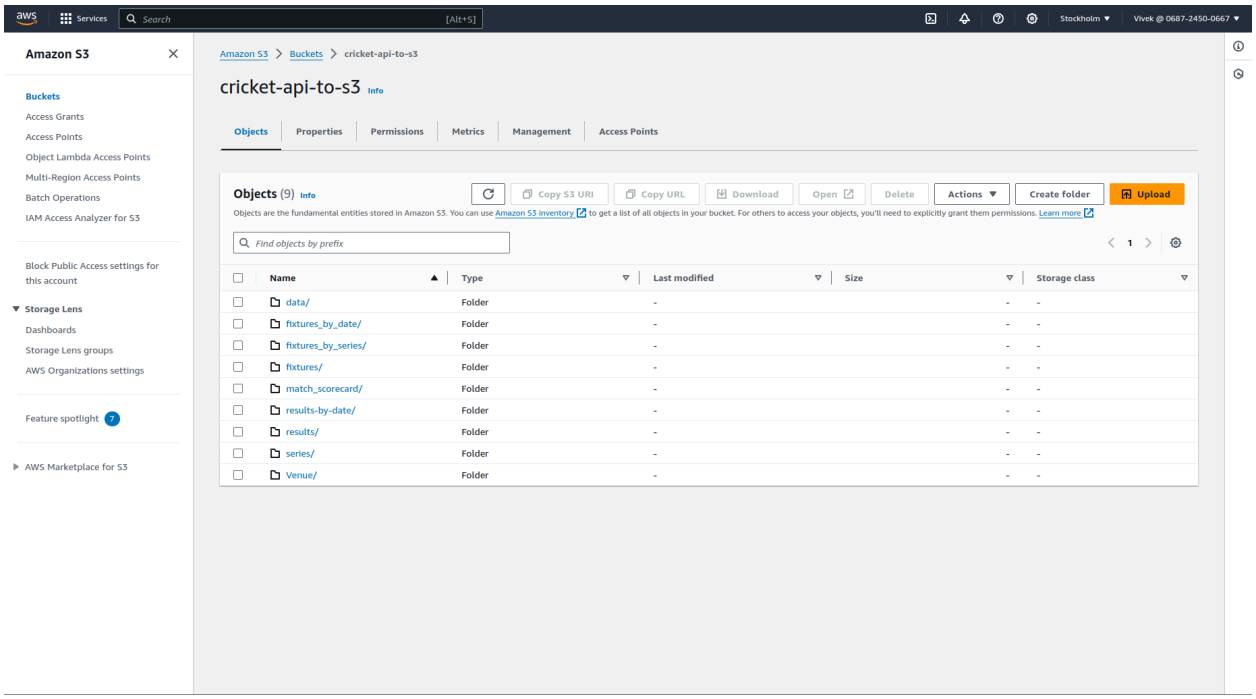
```python
import pandas as pd
import requests
import boto3
from io import StringIO
import datetime

# Global variables
RAPIDAPI_KEY = "f381eaaa79msh51537dd6e014035p180695jsn67cf7d549118"
RAPIDAPI_HOST = "cricket-live-data.p.rapidapi.com"
AWS_ACCESS_KEY_ID = 'AKIARAACMVC5URUSKCGX'
AWS_SECRET_ACCESS_KEY = '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'
S3_BUCKET_NAME = 'cricket-api-to-s3'
S3_FOLDER_NAME = 'results'


def fetch_data_from_api(url):
    headers = {
        "X-RapidAPI-Key": RAPIDAPI_KEY,
        "X-RapidAPI-Host": RAPIDAPI_HOST
    }
    response = requests.get(url, headers=headers)
    if response.status_code == 200:
        json_data = response.json()
        df_date = []
        for i in range(len(json_data['results'])):
            entry = []
            ide = json_data['results'][i]['id']
            series_id = json_data['results'][i]['series_id']
            venue = json_data['results'][i]['venue']
            date_with_timestamp = json_data['results'][i]['date']
            status = json_data['results'][i]['status']
            result = json_data['results'][i]['result']
            match_title = json_data['results'][i]['match_title']
            match_subtitle = json_data['results'][i]['match_subtitle']
            home_team_id = json_data['results'][i]['home']['id']
            away_team_id = json_data['results'][i]['away']['id']
            entry.append([ide, series_id, venue, date_with_timestamp, status, result, match_title, match_subtitle,
                    home_team_id, away_team_id])
            df_date.extend(entry)
        return pd.DataFrame(df_date, columns = ['id', 'series_id', 'venue', 'date_with_timestamp', 'status', 'result',
                                                'match_title', 'match_subtitle', 'home_team_id', 'away_team_id'])
    else:
        print("Failed to fetch data from:", url)
        return None

# Function to save DataFrame to CSV and upload to S3
def upload_to_s3(df, bucket_name, folder, file_name):

    current_date = datetime.datetime.now()
    year = current_date.year
    month = current_date.month

    s3_output_name = f"{folder}/{year}/{month:02}/{file_name}_{year}_{month:02}_{current_date.day}.csv"


    csv_buffer = StringIO()
    df.to_csv(csv_buffer, index=False)

    s3 = boto3.client('s3', aws_access_key_id=AWS_ACCESS_KEY_ID, aws_secret_access_key=AWS_SECRET_ACCESS_KEY)
    s3.put_object(Body=csv_buffer.getvalue(), Bucket=bucket_name, Key=s3_output_name)

    print(f"Data has been uploaded to S3 bucket {bucket_name} with key {s3_output_name}")


results_df = fetch_data_from_api("https://cricket-live-data.p.rapidapi.com/results")

cols = ['venue', 'result', 'match_title', 'match_subtitle']
for col in cols:
    results_df[col] = results_df[col].str.replace(',', '')


# Upload to S3
if results_df is not None:
    upload_to_s3(results_df, S3_BUCKET_NAME, S3_FOLDER_NAME, S3_FOLDER_NAME)
```

6.Result_by_date

```python
import pandas as pd
import requests
import boto3
from io import StringIO
import datetime

# Global variables
RAPIDAPI_KEY = "f381eaaa79msh51537dd6e014035p180695jsn67cf7d549118"
RAPIDAPI_HOST = "cricket-live-data.p.rapidapi.com"
AWS_ACCESS_KEY_ID = 'AKIARAACMVC5URUSKCGX'
AWS_SECRET_ACCESS_KEY = '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'
S3_BUCKET_NAME = 'cricket-api-to-s3'
S3_UPLOADED_FILE_NAME = 'results-by-date'

# API endpoint base URL and headers
base_url = "https://cricket-live-data.p.rapidapi.com/results-by-date/"
headers = {
    "X-RapidAPI-Key": RAPIDAPI_KEY,
    "X-RapidAPI-Host": RAPIDAPI_HOST
}
def fetch_result_by_date(date_list):
    df_date = []
    for date in date_list:
        url = base_url + date
        response = requests.get(url, headers=headers)
        if response.status_code == 200:
            json_data = response.json()
            for i in range(len(json_data['results'])):
                entry = []
                ide = json_data['results'][i]['id']
                series_id = json_data['results'][i]['series_id']
                venue = json_data['results'][i]['venue']
                date_with_timestamp = json_data['results'][i]['date']
                status = json_data['results'][i]['status']
                result = json_data['results'][i]['result']
                match_title = json_data['results'][i]['match_title']
                match_subtitle = json_data['results'][i]['match_subtitle']
                home_team_id = json_data['results'][i]['home']['id']
                away_team_id = json_data['results'][i]['away']['id']
                entry.append([ide, series_id, venue, date_with_timestamp, status, result, match_title, match_subtitle,
                        home_team_id, away_team_id])
                df_date.extend(entry)
    return pd.DataFrame(df_date, columns = ['id', 'series_id', 'venue', 'date_with_timestamp', 'status', 'result',
                                            'match_title', 'match_subtitle', 'home_team_id', 'away_team_id'])


# Function to save DataFrame to CSV and upload to S3
def upload_to_s3(df, bucket_name, folder):
    # Get current year and month
    current_date = datetime.datetime.now()
    year = current_date.year
    month = current_date.month

    # Construct S3 key
    s3_output_name = f"{folder}/"
    s3_output_name += f"{year}/"
    s3_output_name += f"{month:02}/"
    s3_output_name += f"{folder}_{year}_{month:02}_{current_date.day}.csv"

    # Save DataFrame to CSV and upload to S3
    csv_buffer = StringIO()
    df.to_csv(csv_buffer, index=False)
    s3 = boto3.client('s3', aws_access_key_id=AWS_ACCESS_KEY_ID, aws_secret_access_key=AWS_SECRET_ACCESS_KEY)
    s3.put_object(Body=csv_buffer.getvalue(), Bucket=bucket_name, Key=s3_output_name)
    print(f"Data has been uploaded to S3 bucket {bucket_name} with key {s3_output_name}")

# List of dates
arr_datess = ['2024-03-19', '2024-03-20', '2024-03-21','2024-03-22', '2024-03-23',
              '2024-03-23', '2024-03-24', '2024-03-24', '2024-03-25', '2024-03-26',
              '2024-03-27', '2024-03-28', '2024-03-29', '2024-03-30', '2024-03-31',
              '2024-03-31', '2024-04-01']

# Fetch data by date
df_result_by_date = fetch_result_by_date(arr_datess)

cols = ['venue', 'result', 'match_title', 'match_subtitle']
for col in cols:
    df_result_by_date[col] = df_result_by_date[col].str.replace(',', '')

# Upload to S3
upload_to_s3(df_result_by_date, S3_BUCKET_NAME, S3_UPLOADED_FILE_NAME)
```

7.Match_scorecard

S3 Bucket interface



Problems we have faced:

1. Most of the time we faced api errors because of our free version.
2. While debugging code according to the requirement of upstream.
3. We have faced a problem while dealing with nested JSON files.

3. Send S3 data to Oracle with Talend - Vivek

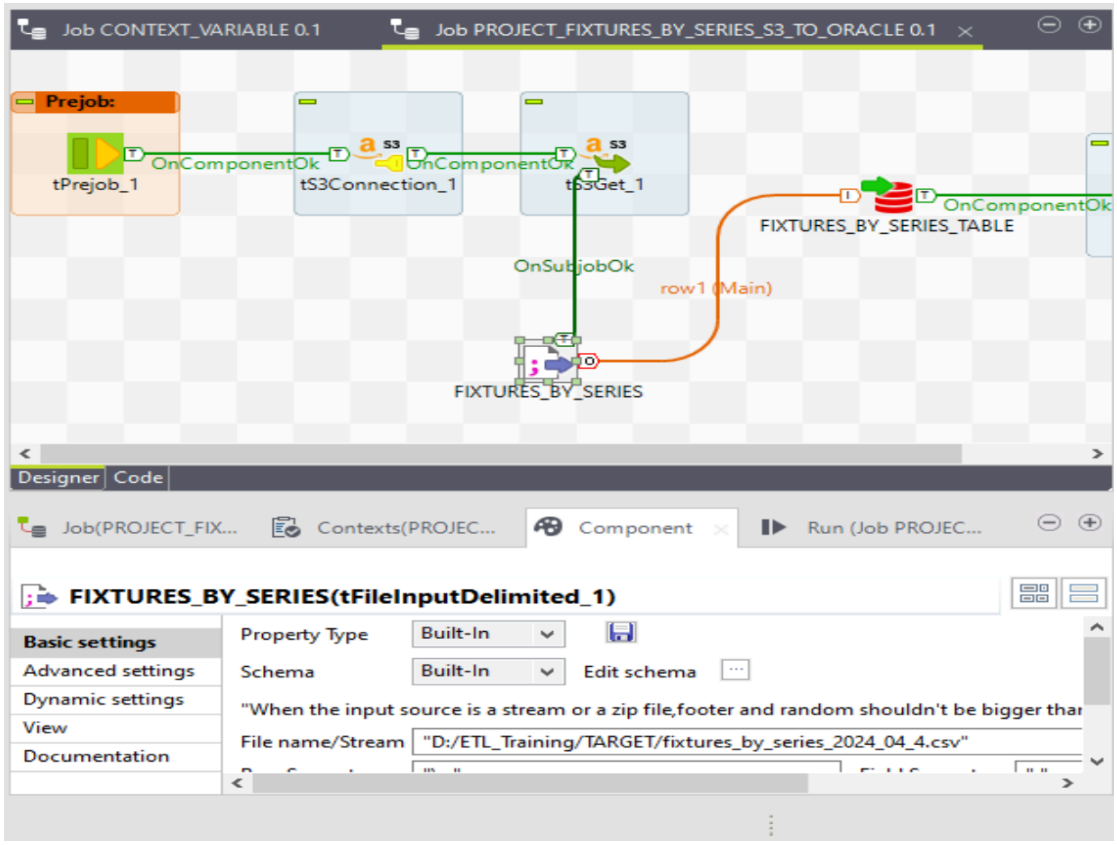
Here we have to prepare 7 individual jobs as we have 7 csv files in S3. So to process each file we have to create individual job for each file. Configuration in these jobs are same. So attaching only one jobs configuration screen shots. We have created metadata of all the csv files in metadata section to give schema information in our component.

1. tS3Connection: Start by connecting to the TeamSpeak 3 server using ts3connection, then using ts3get to retrieve information or perform actions like querying user data.

2. After processing data from the TeamSpeak server, using tfileinputdelimited to read CSV data, transform it as needed, and then using tdboutput to write the transformed data into a database table.
3. Once the data processing is complete and the connection is no longer needed, using ts3close to gracefully close the connection to the TeamSpeak 3 server.

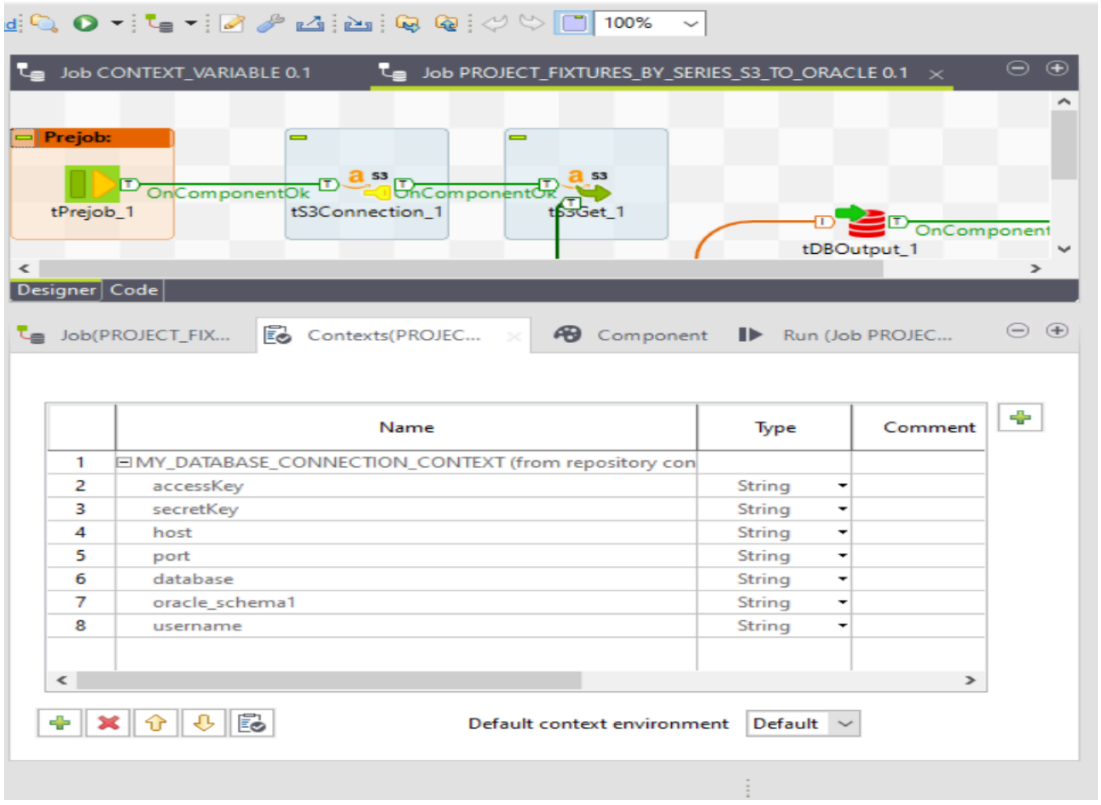
Here are the screenshots for fixtures by series data sets. All the connection and context parameter screenshots are attached.

1.

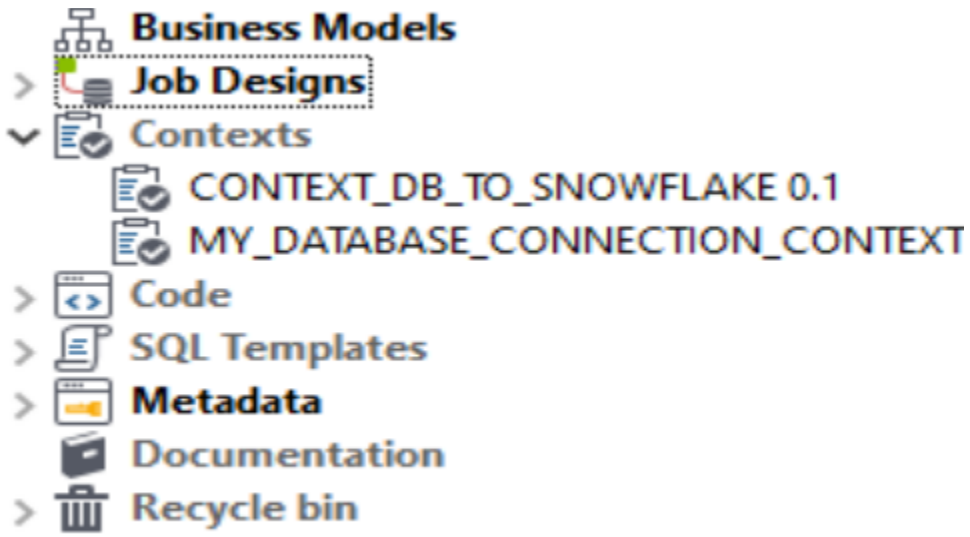


Setting up the context parameter for a particular job. Here database connection details are the same for all the jobs. But if in any scenario we want to change the database connection details then we can change it from the .csv file that we have created as a context parameter.

2.

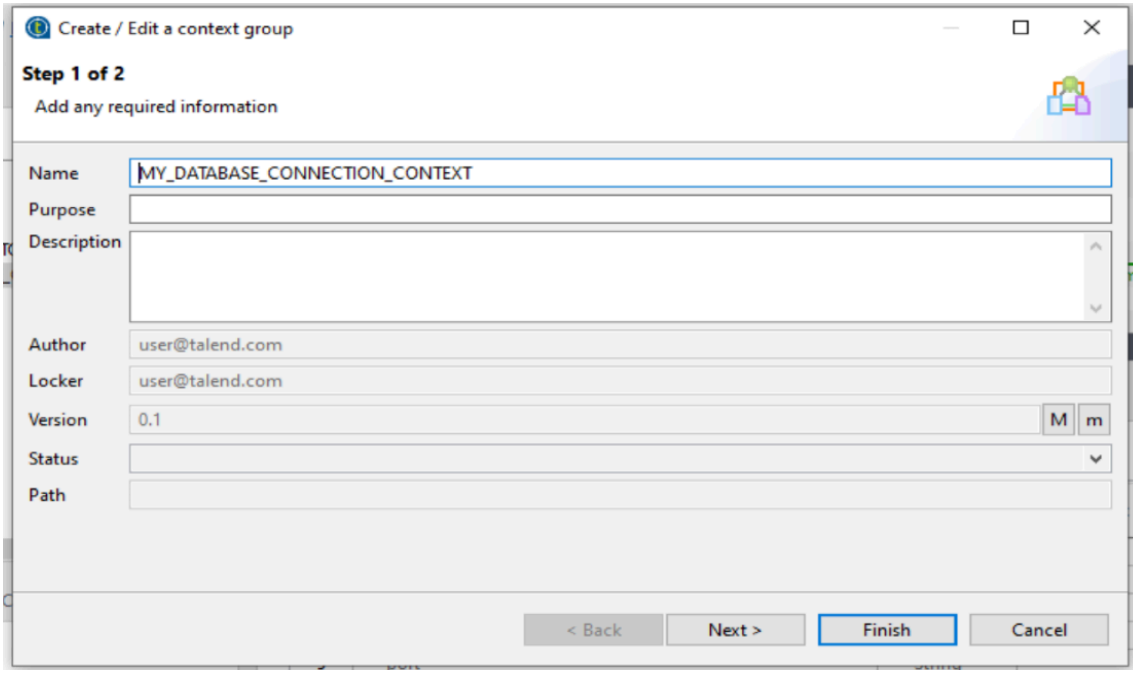

Job CONTEXT_VARIABLE 0.1
Job PROJECT_FIXTURES_BY_SERIES_S3_TO_ORACLE 0.1
Prejob:
tPrejob_1
OnComponentOk
tS3Connection_1
OnComponentOk
tS3Get_1
tDBOutput_1
Designer
Code
Job(PROJECT_FIX...
Contexts(PROJEC...
Component
Run (Job PROJEC...
Name
Type
Comment
1 MY_DATABASE_CONNECTION_CONTEXT (from repository con
2 accessKey String
3 secretKey String
4 host String
5 port String
6 database String
7 oracle_schema1 String
8 username String
Default context environment
Default

3.

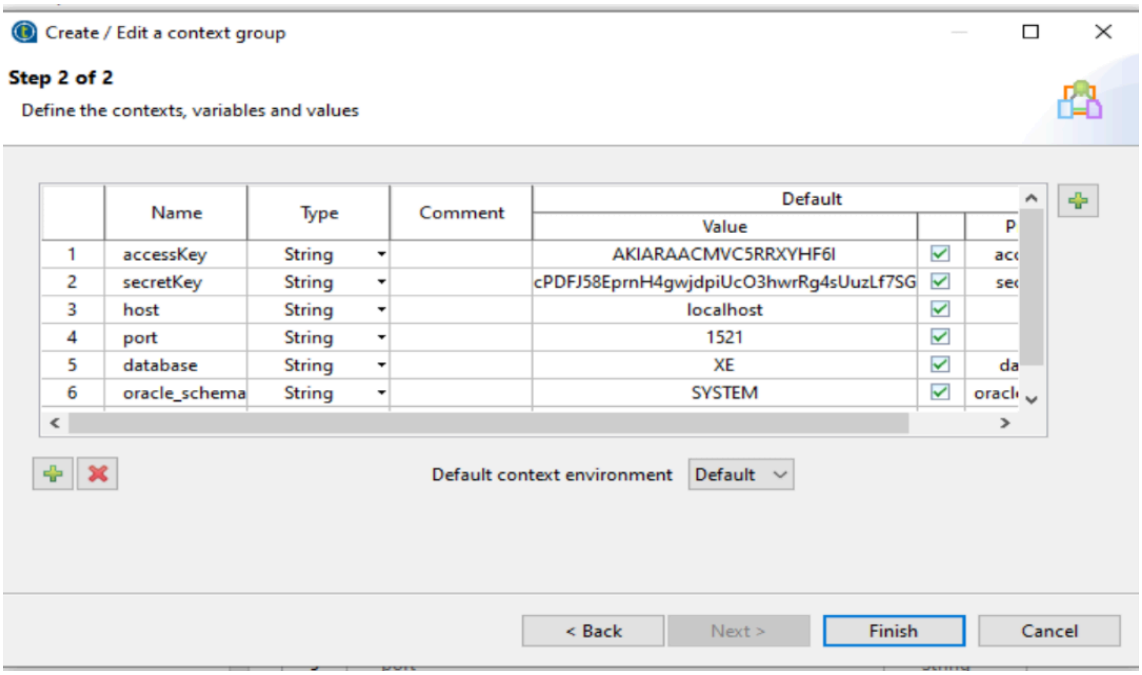

Business Models
Job Designs
Contexts
CONTEXT_DB_TO_SNOWFLAKE 0.1
MY_DATABASE_CONNECTION_CONTEXT
Code
SQL Templates
Metadata
Documentation
Recycle bin

Parameter name in context file and in context group should be the same, otherwise it will give an error.

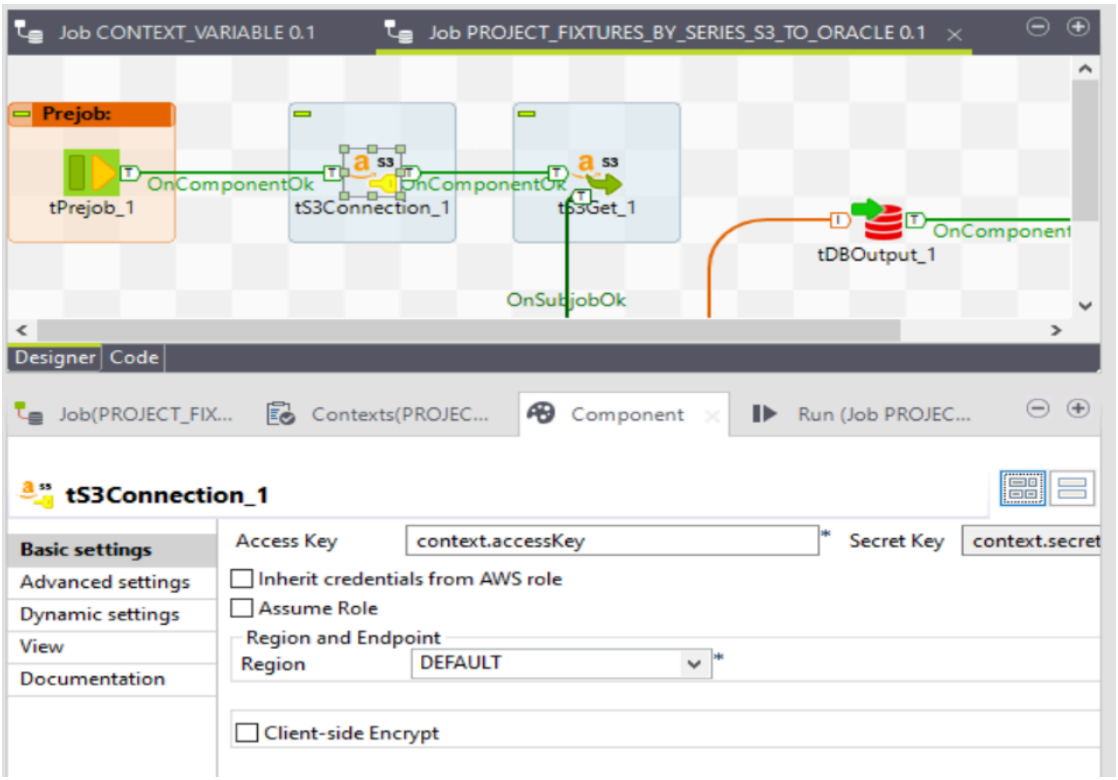
4.



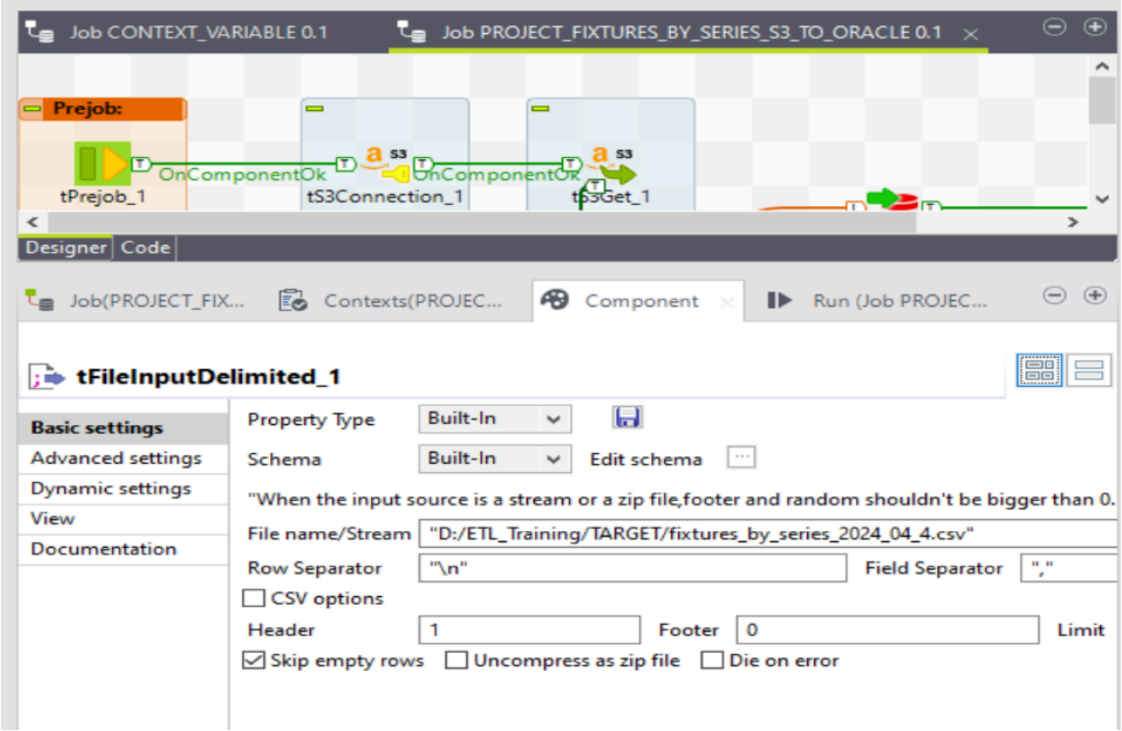
5.



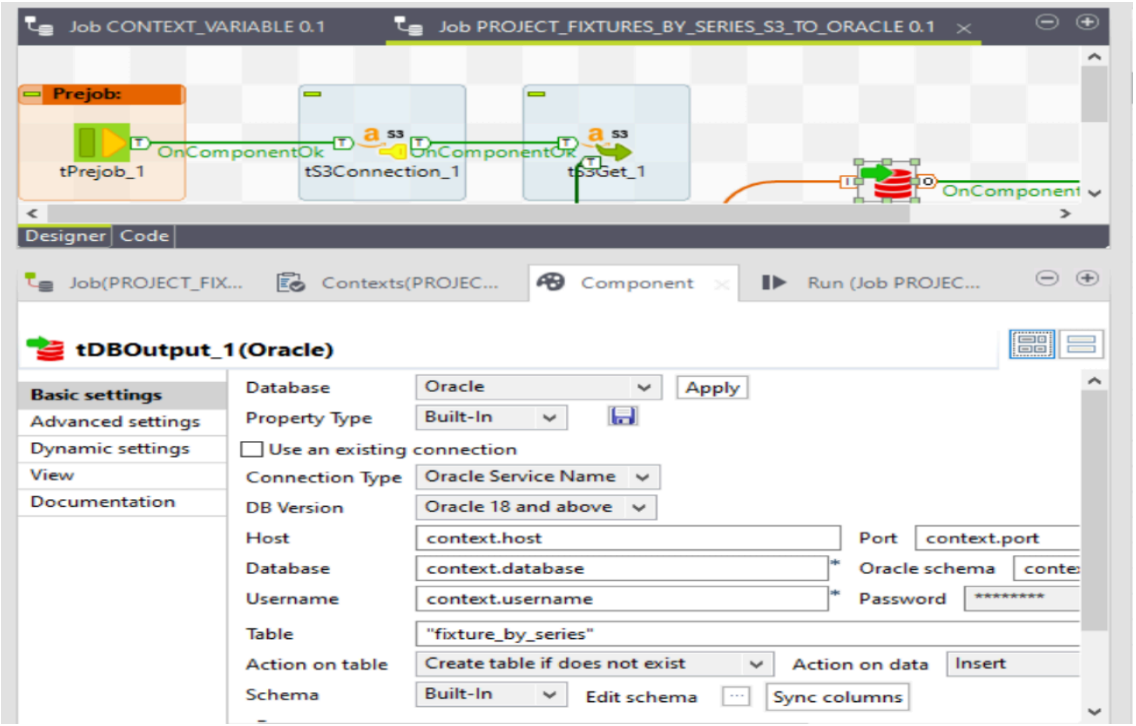
6.


Job CONTEXT_VARIABLE 0.1
Job PROJECT_FIXTURES_BY_SERIES_S3_TO_ORACLE 0.1
Prejob:
tPrejob_1
OnComponentOk
tS3Connection_1
OnComponentOk
tS3Get_1
tDBOutput_1
OnComponent
OnSubjobOk
Designer
Code
Job(PROJECT_FIX...
Contexts(PROJEC...
Component
Run (Job PROJEC...
tS3Connection_1
Basic settings
Advanced settings
Dynamic settings
View
Documentation
Access Key
context.accessKey
Secret Key
context.secret
Inherit credentials from AWS role
Assume Role
Region and Endpoint
Region
DEFAULT
Client-side Encrypt

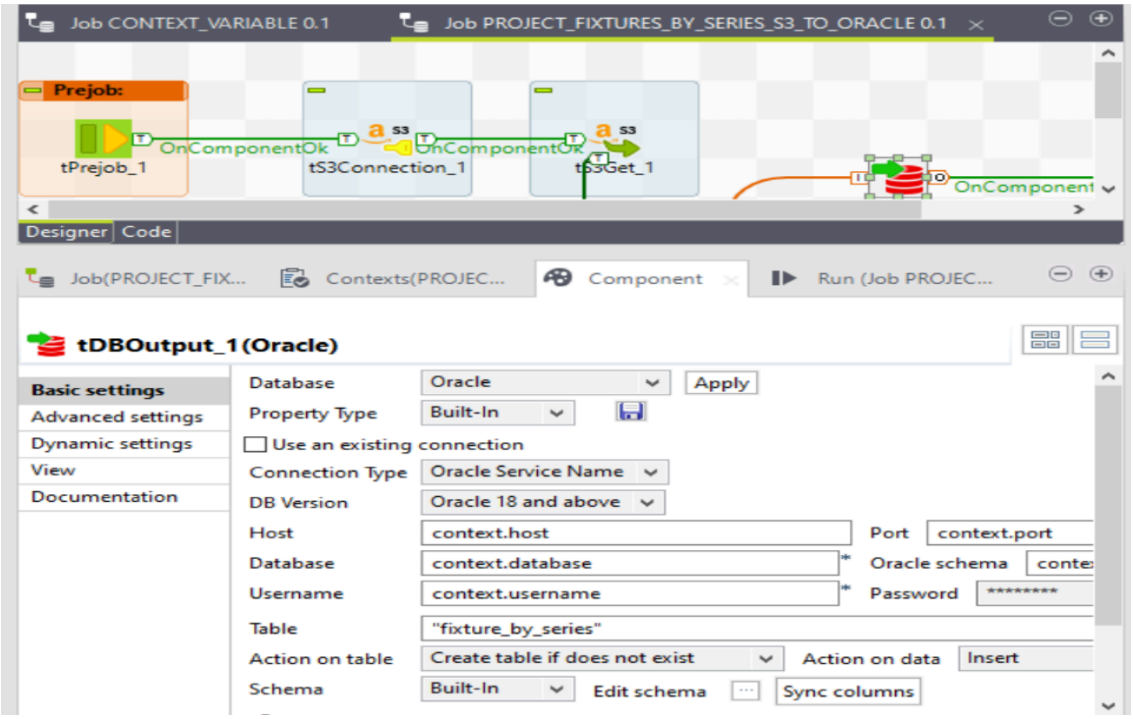
7.


Job CONTEXT_VARIABLE 0.1
Job PROJECT_FIXTURES_BY_SERIES_S3_TO_ORACLE 0.1
Prejob:
tPrejob_1
OnComponentOk
tS3Connection_1
tS3Get_1
Designer
Code
Job(PROJECT_FIX...
Contexts(PROJEC...
Component
Run (Job PROJEC...
tFileInputDelimited_1
Basic settings
Advanced settings
Dynamic settings
View
Documentation
Property Type
Built-In
Schema
Edit schema
"When the input source is a stream or a zip file,footer and random shouldn't be bigger than 0.
File name/Stream
"D:/ETL_Training/TARGET/fixtures_by_series_2024_04_4.csv"
Row Separator
"\n"
Field Separator
","
CSV options
Header
1
Footer
0
Limit
Skip empty rows
Uncompress as zip file
Die on error

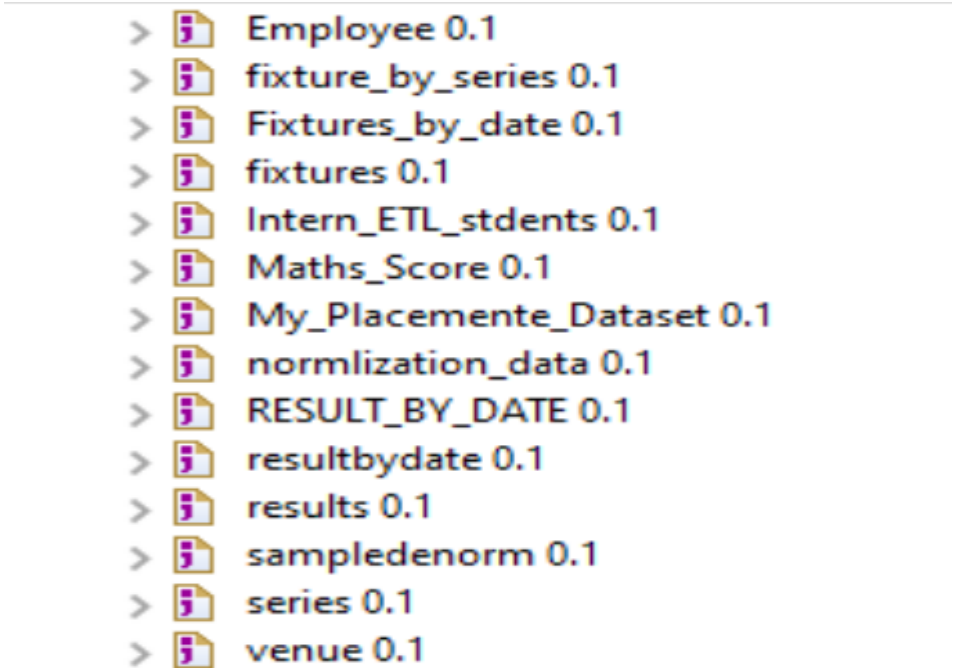
8.


Job CONTEXT_VARIABLE 0.1
Job PROJECT_FIXTURES_BY_SERIES_S3_TO_ORACLE 0.1
Prejob:
tPrejob_1
OnComponentOk
tS3Connection_1
tS3Get_1
Designer
Code
Job(PROJECT_FIX...
Contexts(PROJEC...
Component
Run (Job PROJEC...
tDBOutput_1 (Oracle)
Basic settings
Advanced settings
Dynamic settings
View
Documentation
Database
Oracle
Apply
Property Type
Built-In
Use an existing connection
Connection Type
Oracle Service Name
DB Version
Oracle 18 and above
Host
context.host
Port
context.port
Database
context.database
Oracle schema
Username
context.username
Password
Table
"fixture_by_series"
Action on table
Create table if does not exist
Action on data
Insert
Schema
Built-In
Edit schema
Sync columns

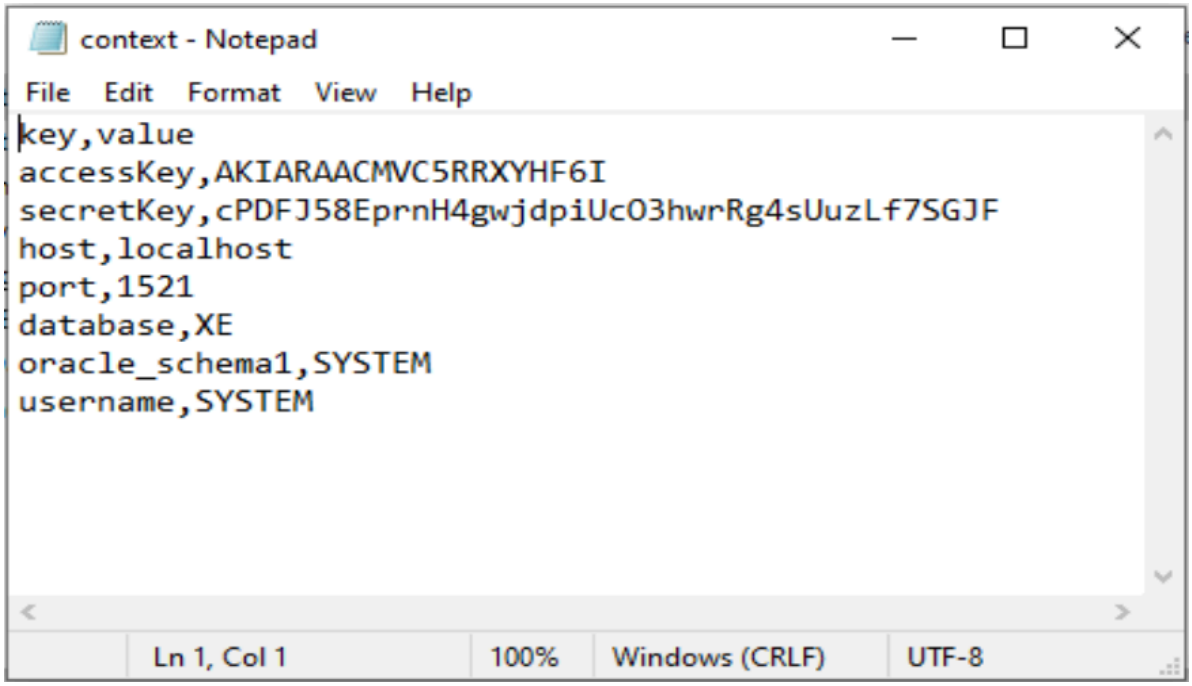
9.


Job CONTEXT_VARIABLE 0.1
Job PROJECT_FIXTURES_BY_SERIES_S3_TO_ORACLE 0.1
Prejob:
tPrejob_1
OnComponentOk
tS3Connection_1
OnComponentOk
tS3Get_1
OnComponent
Designer
Code
Job(PROJECT_FIX...
Contexts(PROJEC...
Component
Run (Job PROJEC...
tDBOutput_1 (Oracle)
Basic settings
Advanced settings
Dynamic settings
View
Documentation
Database
Oracle
Apply
Property Type
Built-In
Use an existing connection
Connection Type
Oracle Service Name
DB Version
Oracle 18 and above
Host
context.host
Port
context.port
Database
context.database
Oracle schema
Username
context.username
Password
Table
"fixture_by_series"
Action on table
Create table if does not exist
Action on data
Insert
Schema
Built-In
Edit schema
Sync columns

10.



11.



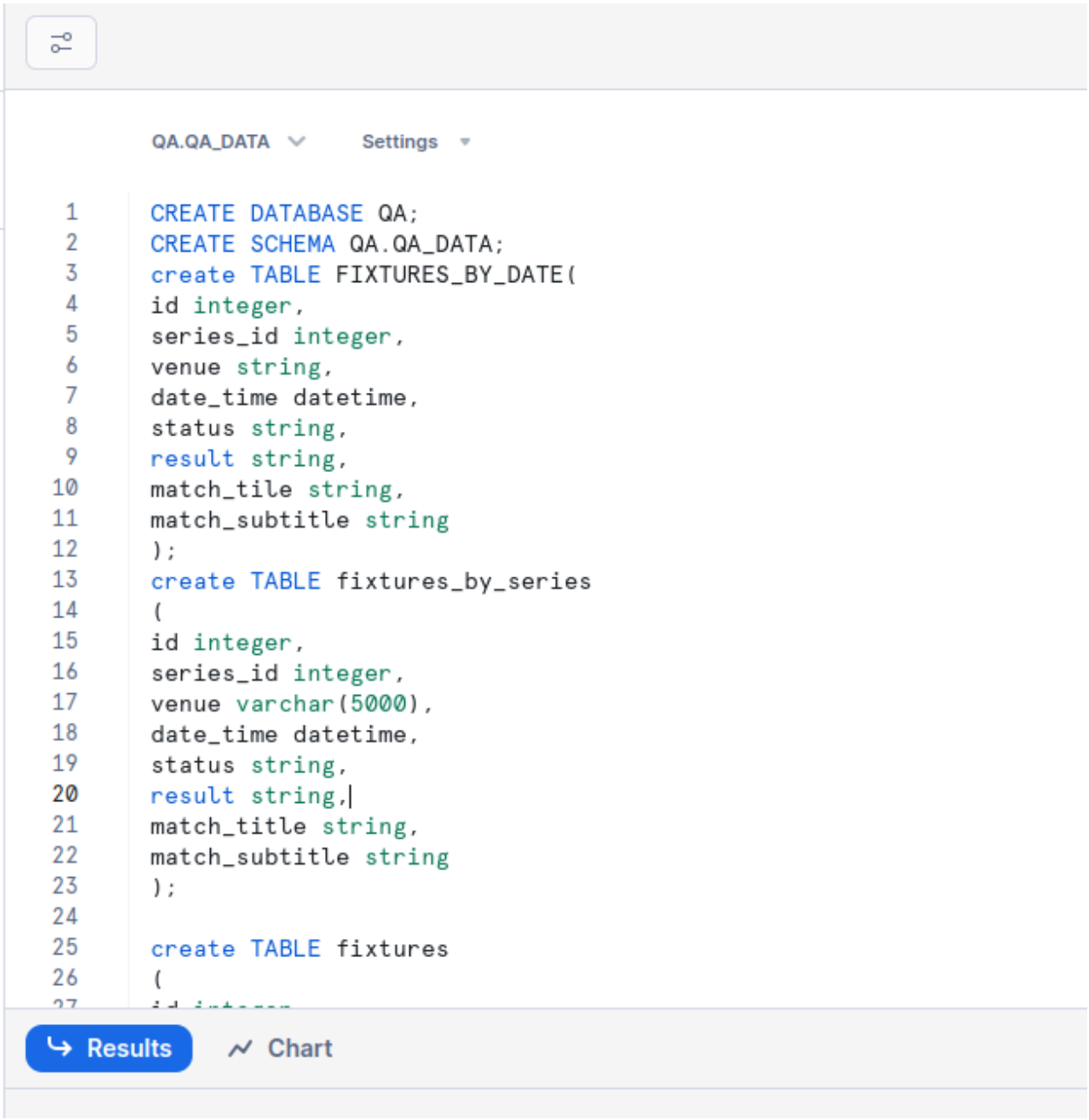
This is the .csv file for setting up the connection details.

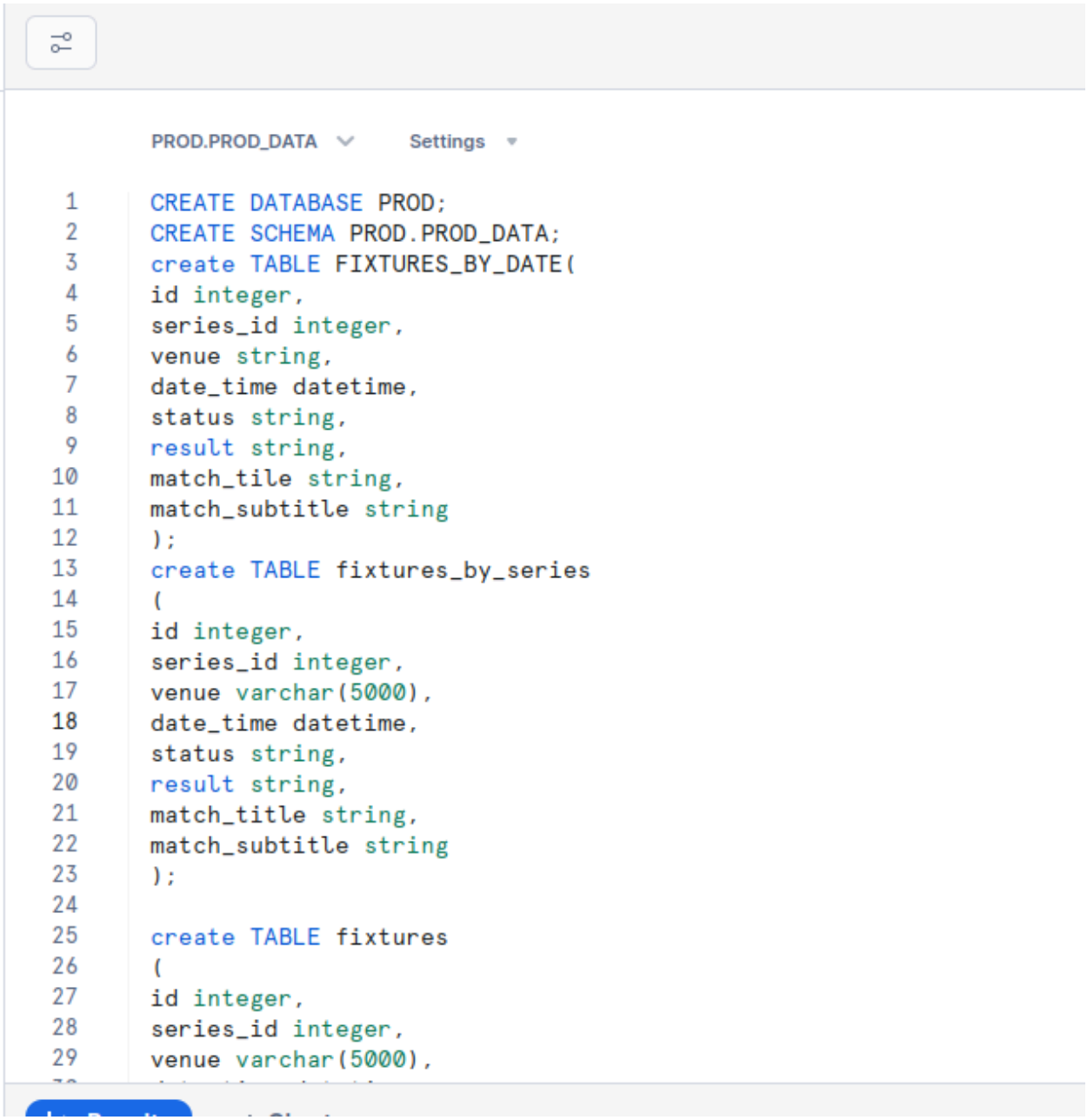
4. Migrate Oracle data to Snowflake with Talend - [Purvang,Jagruti]

Creating Schema and Database in snowflake:
DEV, PROD and QA:

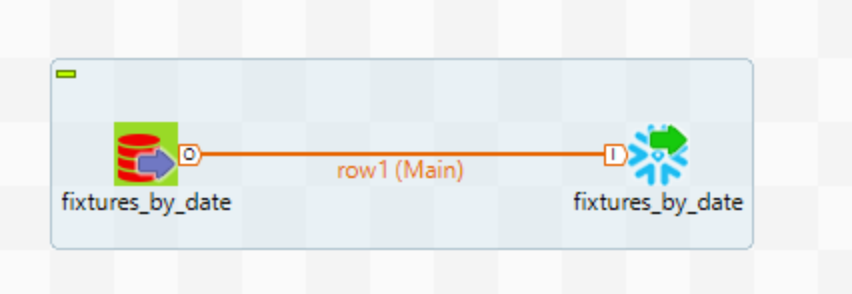
DEV.DEV_DATA Settings

```
CREATE DATABASE DEV;
CREATE SCHEMA DEV.DEV_DATA;
create TABLE FIXTURES_BY_DATE(
id integer,
series_id integer,
venue string,
date_time datetime,
status string,
result string,
match_tile string,
match_subtitle string,
HOME_TEAM_ID NUMBER(38,0),
AWAY_TEAM_ID NUMBER(38,0)
);
create TABLE fixtures_by_series
(
id integer,
series_id integer,
venue varchar(5000),
date_time string,
status string,
result string,
match_title string,
match_subtitle string,
HOME_TEAM_ID NUMBER(38,0),
AWAY_TEAM_ID NUMBER(38,0)

);

create TABLE fixtures
```

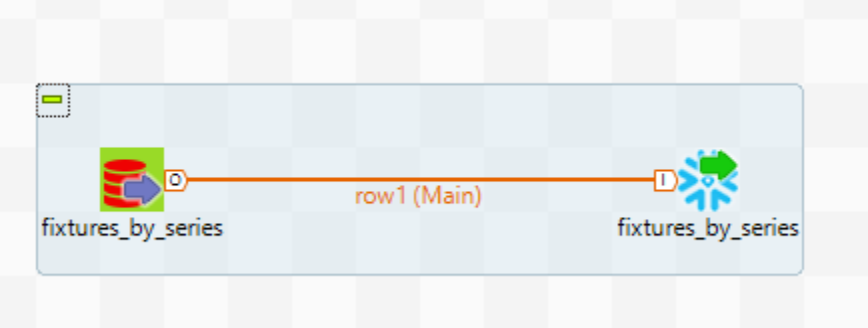

QA.QA_DATA
Settings
1 CREATE DATABASE QA;
2 CREATE SCHEMA QA.QA_DATA;
3 create TABLE FIXTURES_BY_DATE(
4 id integer,
5 series_id integer,
6 venue string,
7 date_time datetime,
8 status string,
9 result string,
10 match_tile string,
11 match_subtitle string
12);
13 create TABLE fixtures_by_series
14 (
15 id integer,
16 series_id integer,
17 venue varchar(5000),
18 date_time datetime,
19 status string,
20 result string,
21 match_title string,
22 match_subtitle string
23);
24
25 create TABLE fixtures
26 (
Results
Chart



1. Fixtures_by_date:



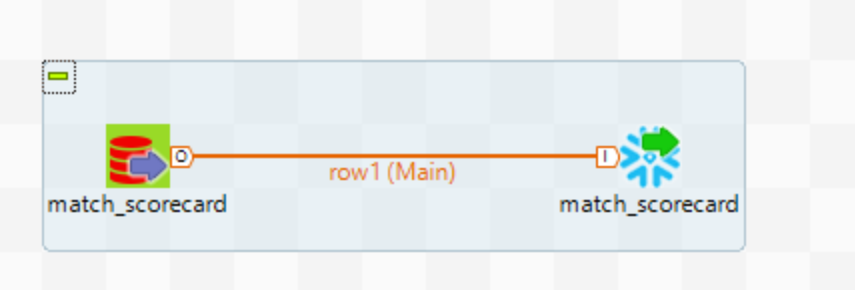
2. Fixtures_by_series:



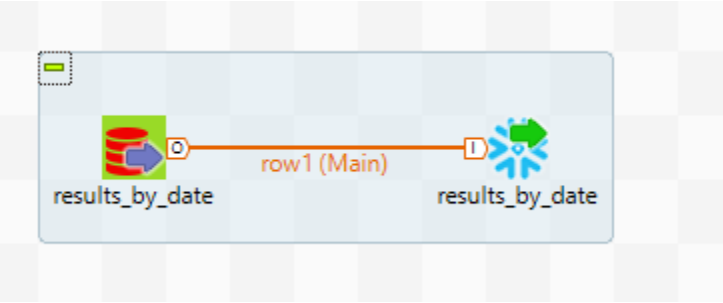
.

3. Results:

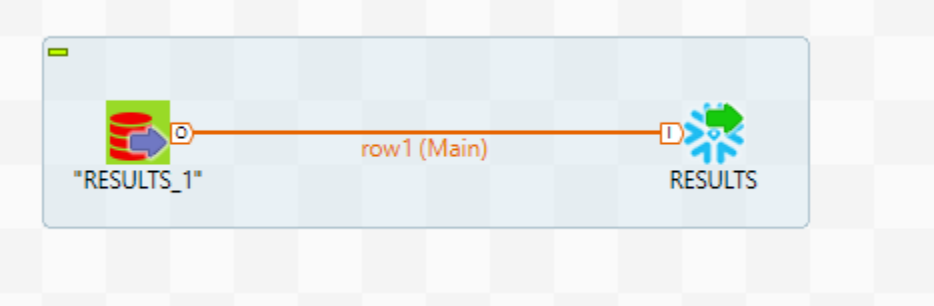
4. Match_scorecard:



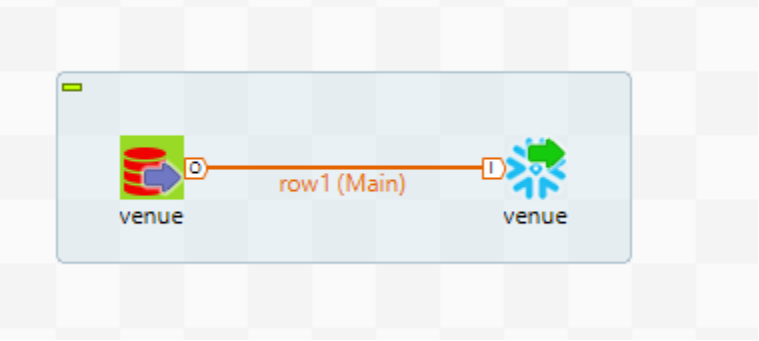
5. Results_by_date:



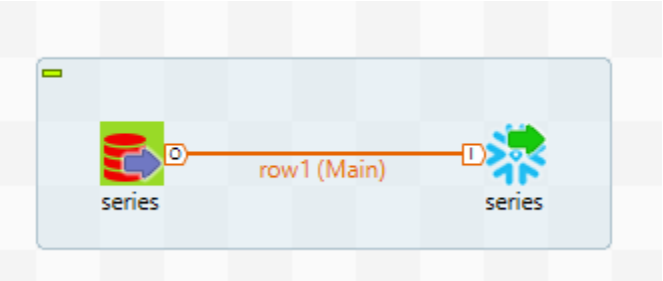
6. Results:



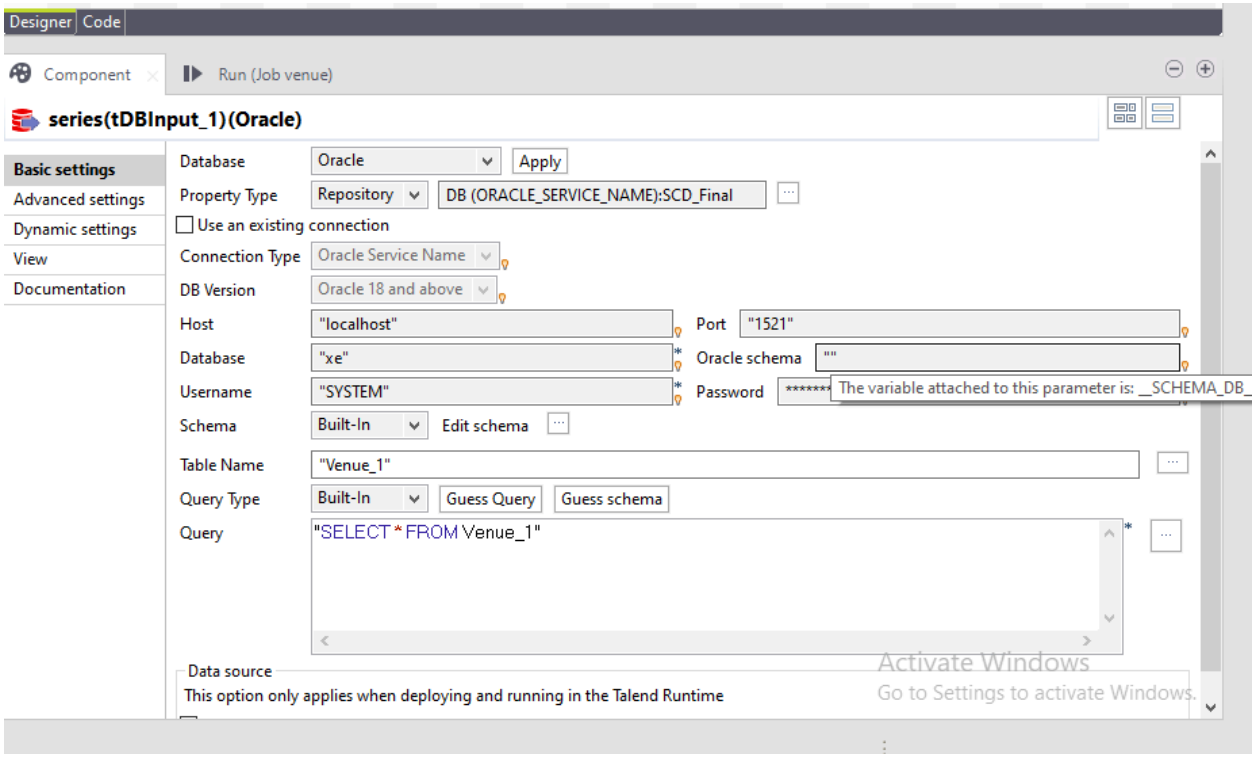
7. Venue:



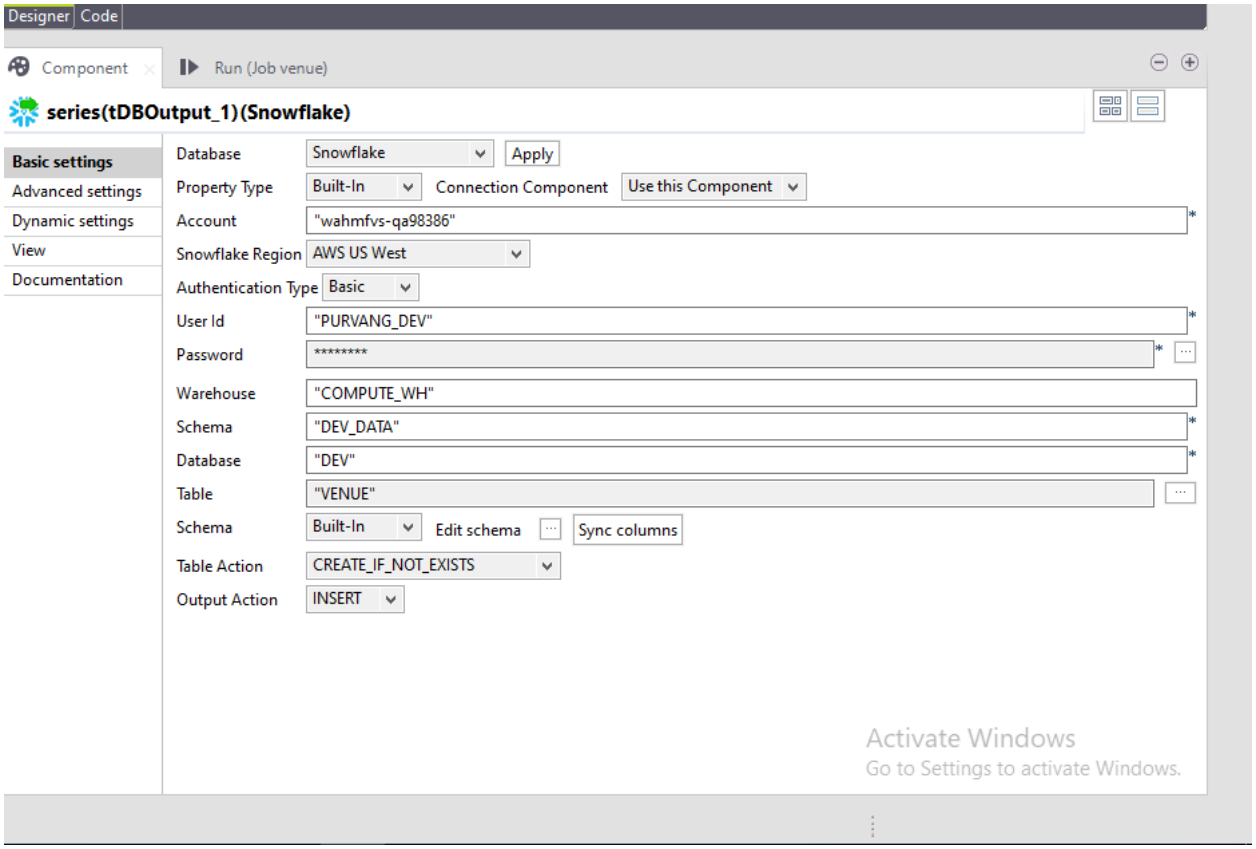
8. Series:



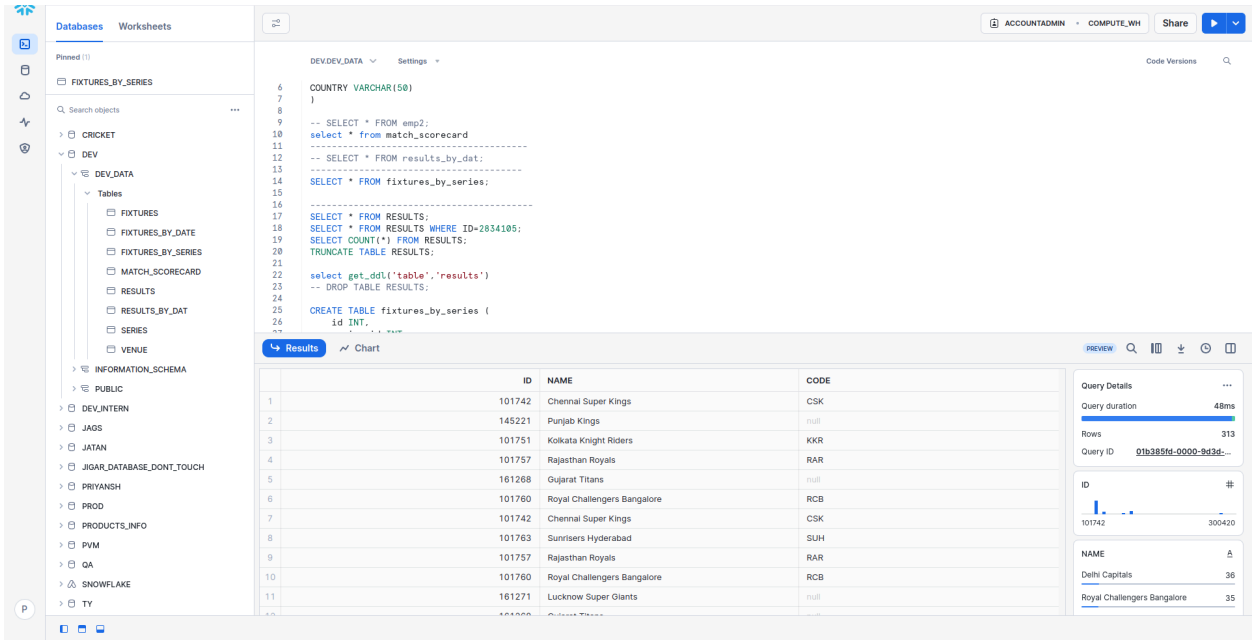
Oracle Configuration:



Snowflake Configuration:



Snowflake:



5. Prepare Data validation process and sample reports in Power BI for data prepared in Snowflake [Vipasha,Jagruti and Jigar]

The ETL process serves as the backbone of any data-driven decision-making process. It's the means through which raw data is transformed into meaningful insights. This makes data validation an integral part of ETL.

- Maintain Data Quality: Data validation helps in maintaining high data quality by identifying and rectifying errors, inconsistencies, and discrepancies in the data during the ETL process.
- Ensure Accurate Reporting: By validating data during the ETL process, you ensure that the final data used for reporting and analysis is accurate. This leads to reliable insights and informed decision-making.
- Minimize Errors: Through data validation, you can spot and rectify errors in the early stages of the ETL process, saving time and effort in handling them later.
- Protect Data Integrity: By ensuring that the transformation rules are correctly applied, and that the right data is loaded into the target system, data validation protects the integrity of your data.

Stages of Data Validation in ETL

Data validation in ETL isn't a one-time operation but rather a continuous process that spans across all stages of ETL. Let's take a closer look at how data validation is integrated at each step.

Extraction Stage: Validating the Source Data

The extraction phase is where data is collected from various source systems. It's the starting point of the ETL process and the first opportunity for data validation.

- Data Completeness Check: At this stage, you want to ensure that all required data has been extracted. This can be done by comparing record counts or using checksums from the source and the extracted data.
- Data Accuracy Check: Data pulled from source systems should match exactly what's in those systems. Basic field-level checks can be performed to verify accuracy.

- Initial Data Quality Check: If the quality of the source data is suspect, it may be helpful to perform some preliminary data quality checks to identify potential issues early.

Transformation Stage: Validating Transformation Rules

The transformation phase, where data is cleaned and transformed into a format suitable for the target system, is another crucial point for data validation.

- Validation of Transformation Rules: As data gets transformed, validating that the rules and logic applied are producing the expected results is crucial.
- Data Consistency Check: Ensuring that data remains consistent after transformation is key. For instance, the same type of data should not have different formats.
- Null Check: It's essential to check that mandatory fields are not left null after transformation.

Load Stage: Validating the Target Data

The final stage of ETL is loading the transformed data into the target system, usually a data warehouse. Here, validation ensures that the load operation was successful and accurate.

- Data Completeness Check: Similar to the extraction stage, you want to ensure that all data has been loaded into the target system.
- Data Integrity Check: Validating that relationships between data elements (like foreign key relationships) have been maintained during the load process is crucial.
- Reconciliation Check: This involves comparing the data in the source and target systems to ensure they match, confirming the ETL process was successful.

Techniques for Data Validation in ETL

Data validation in the ETL process encompasses a range of techniques designed to ensure data integrity, accuracy, and consistency. Here are some commonly utilized validation techniques:

Data Type Checks

Data type checks involve verifying that each data element is of the correct data type. This could include checking whether numeric fields contain numeric data, date fields contain valid dates, and so forth.

data quality issues even in the source system.

(i) Non-numerical type: Under this classification, we verify the accuracy of the non-numerical content. Examples are Emails, Pin codes, Phone in a valid format.

(ii) Domain analysis: In this type of test, we pick domains of data and validate for errors. There are three groupings for this:

- Based on Value: Here we create a list of values that can occur for a field (column in the table). Then validate if the column values are a subset of our list.
    - Example: Verify if the Gender column contains either M or F.
- Based on Range: Here we set minimum and maximum range for valid data values for a column, based on logical or business reasoning. We then validate if the column values fall within this range.
    - Example: 0 to 120 for Age.
- Reference File: Here the system uses an external validity file.
    - Example: Are Country codes valid, do they pick the right value from the reference file, are Country codes the same between QA and the Production environment? If the reference file had a country code updated, is it rightly updated in DB?

- Data Type Check: Example: Will Total Sales work correctly with Decimal (8, 16, or 20 bytes) or Double type?
- Data Length Check: Example: Will the data length for the Address field be sufficient with 500 chars? It might be a case where data migration is done as new geography is getting added to the company. The addresses of the new geography might have an exceedingly long format and sticking to the original length might error a use case.

- Index check: Example: Is there indexing done for the OrderId column in the target system? What if a merger of companies happened, requiring data migration and the Orders table grows 100 times in size in the target system?

Range Checks

Range checks validate that data values fall within acceptable ranges. For instance, if a data field is supposed to contain a person's age, a range check would confirm that the values fall within a plausible range, such as 0 to 120.

Constraint Checks

Constraint checks involve verifying that data adheres to predefined constraints. These could be unique constraints (e.g., every customer ID should be unique), primary key constraints (e.g., every row should have a unique identifier), or foreign key constraints (e.g., a reference to another table that must exist).

Consistency Checks

Consistency checks are used to ensure data values are consistent across datasets. For example, if you have two tables containing customer data, the customer names and IDs should be consistent across both.

Uniqueness Checks

Uniqueness checks are similar to constraint checks but specifically focus on ensuring that values in a certain field are unique where required. A typical example would be checking that each customer has a unique customer ID.

Null Data

In this type of test, we focus on the validity of null data and verification that the important column cannot be null.

- Example: Filter out all null data and validate if null is allowed.

- If there are important columns for business decisions, make sure nulls are not present.

Range Check

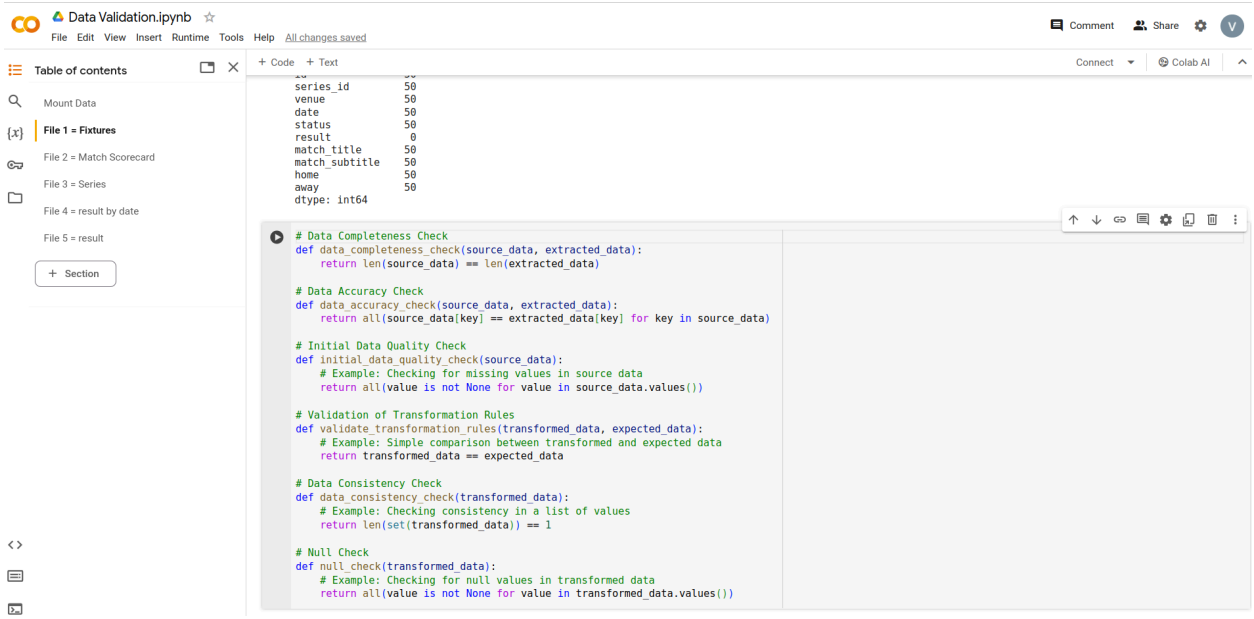
Data entity where ranges make business sense should be tested.

- Example: Order quantity per invoice cannot be more than 5K in the software category.
- Age should not be more than 120.

Referential Integrity Checks

Referential integrity checks involve validating that relationships between tables remain intact. This is particularly important in the loading stage of ETL, where maintaining relationships between data elements (like foreign keys) is crucial.

Using these techniques, you can help ensure that your data is clean, consistent, and reliable, regardless of its source. Remember, a single validation technique might not be enough. It's often a combination of these techniques that ensures thorough validation and helps maintain the quality and integrity of your data.



I have validated and preprocessed the given 5 data tables: with fixing file format,flattening and other text preprocessing and transformed the data into clean CSV format from raw formats.

Jagruti has also performed on 2 files

Jagruti - Tried to extract winning team name from result. Also, since venue name was available in match_title. I removed that and created 2 more columns for teams playing.

Data validation on fixture by date and series files

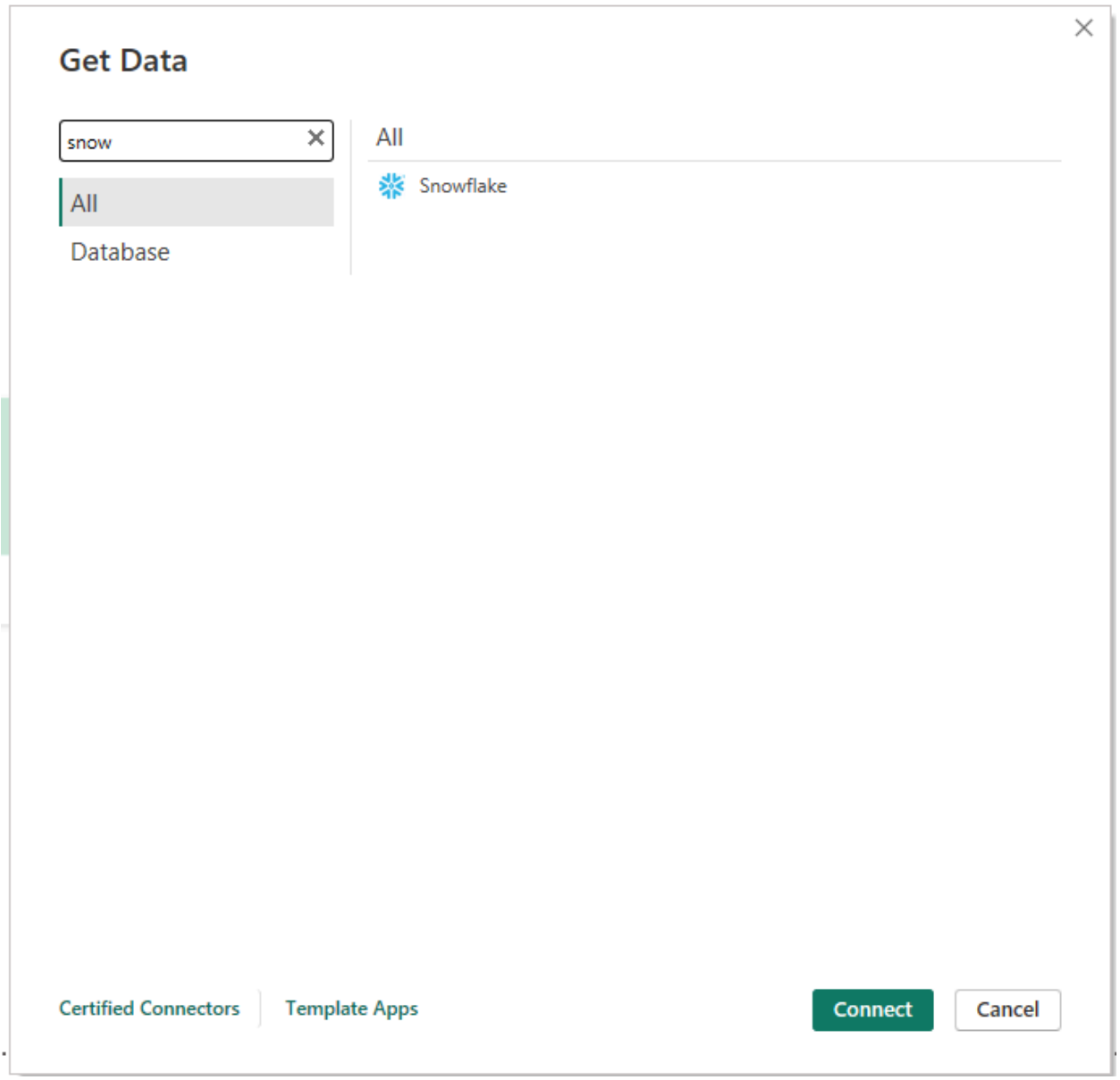
Data Validation Code Link

SnowFlake to PowerBI (Jigar Shekhat)

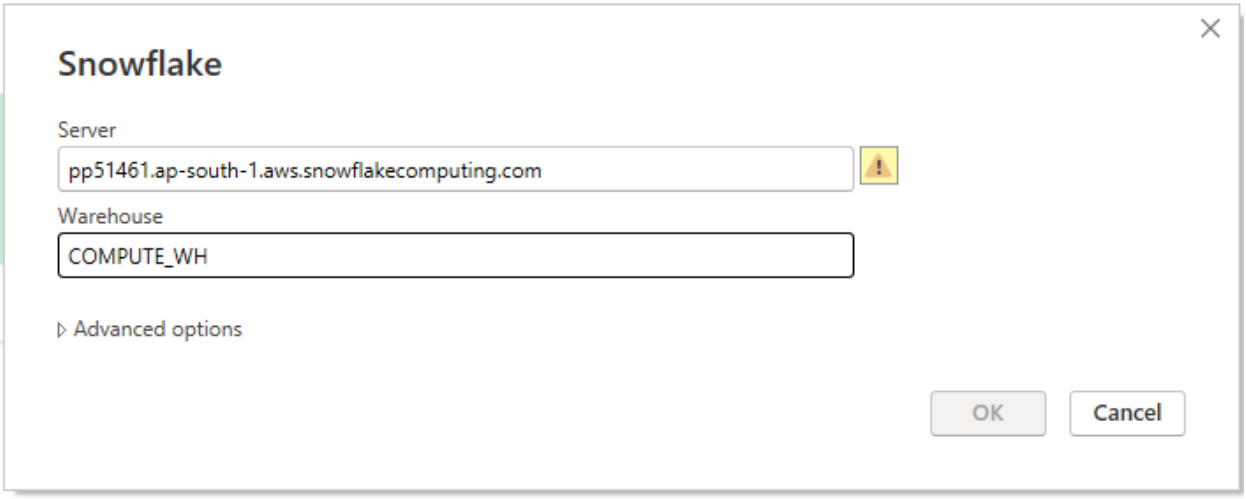
For analytic insight in powerBI, I am load data from DEV database of snowflake.

After loading Data, I prepared some sample report about that cricket data.
Connection with Snowflake :

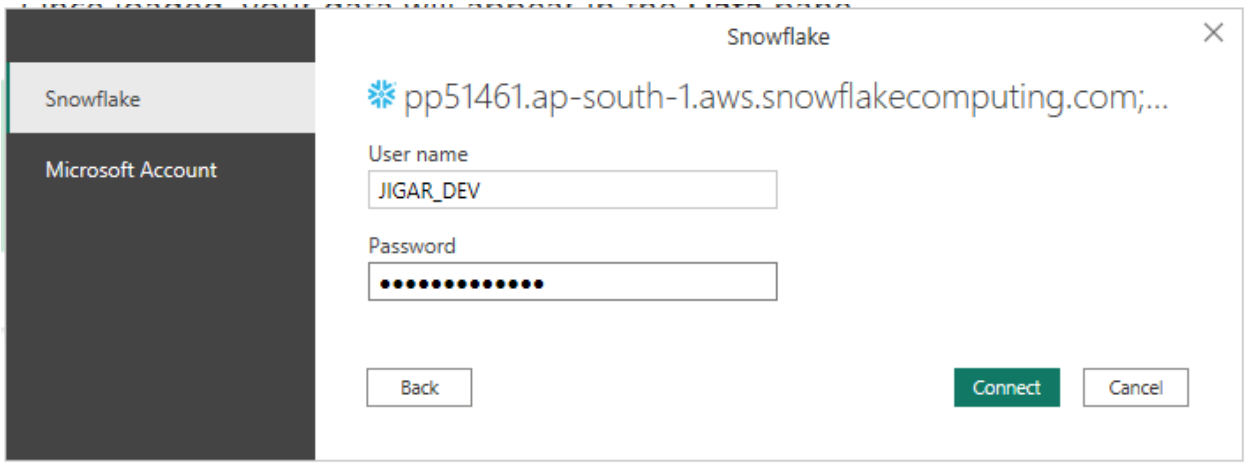
1. **Select snowflake in get data tab.**



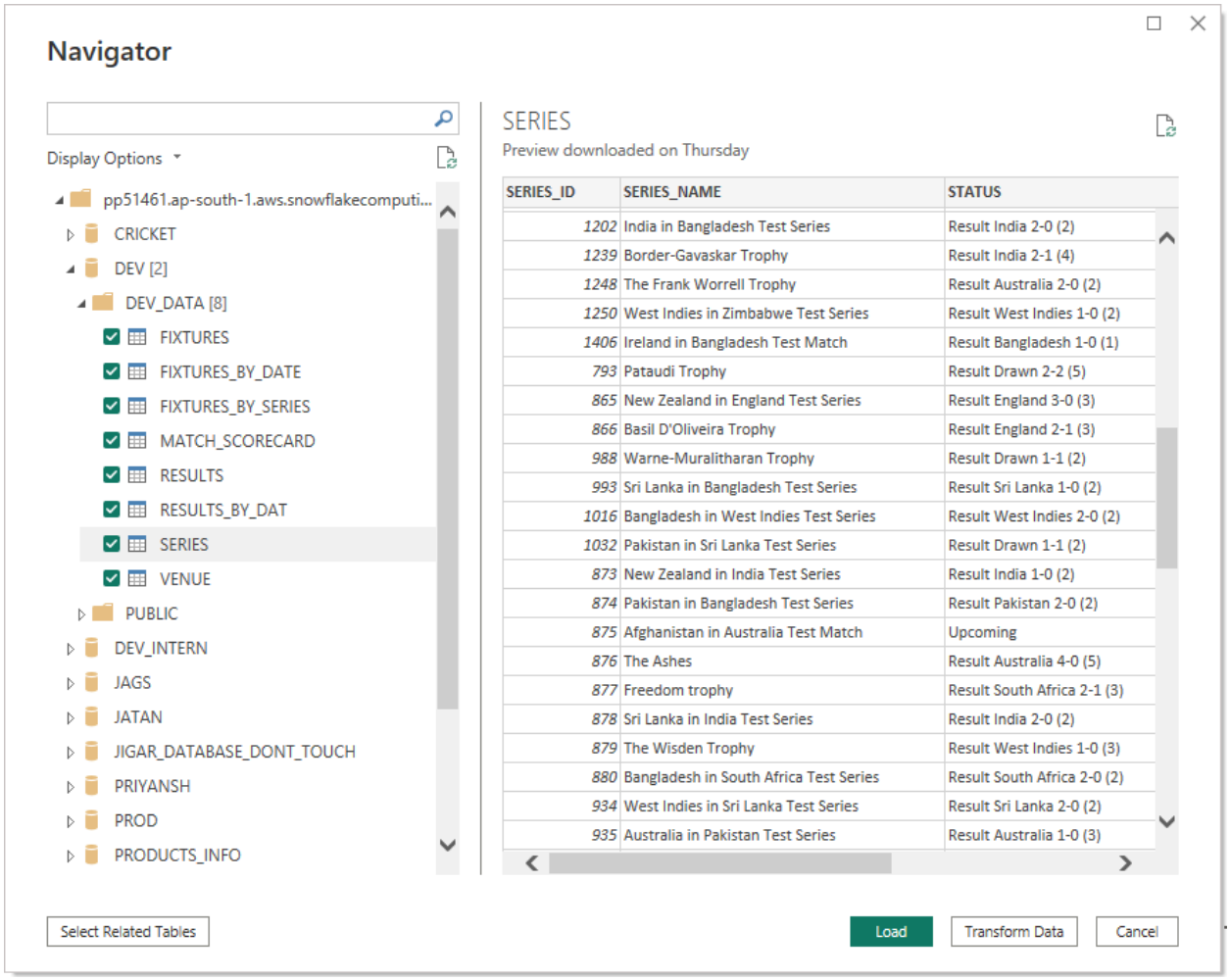
2. **Enter Server and warehouse detail**



3. **After that, add your credentials of snowflake account**

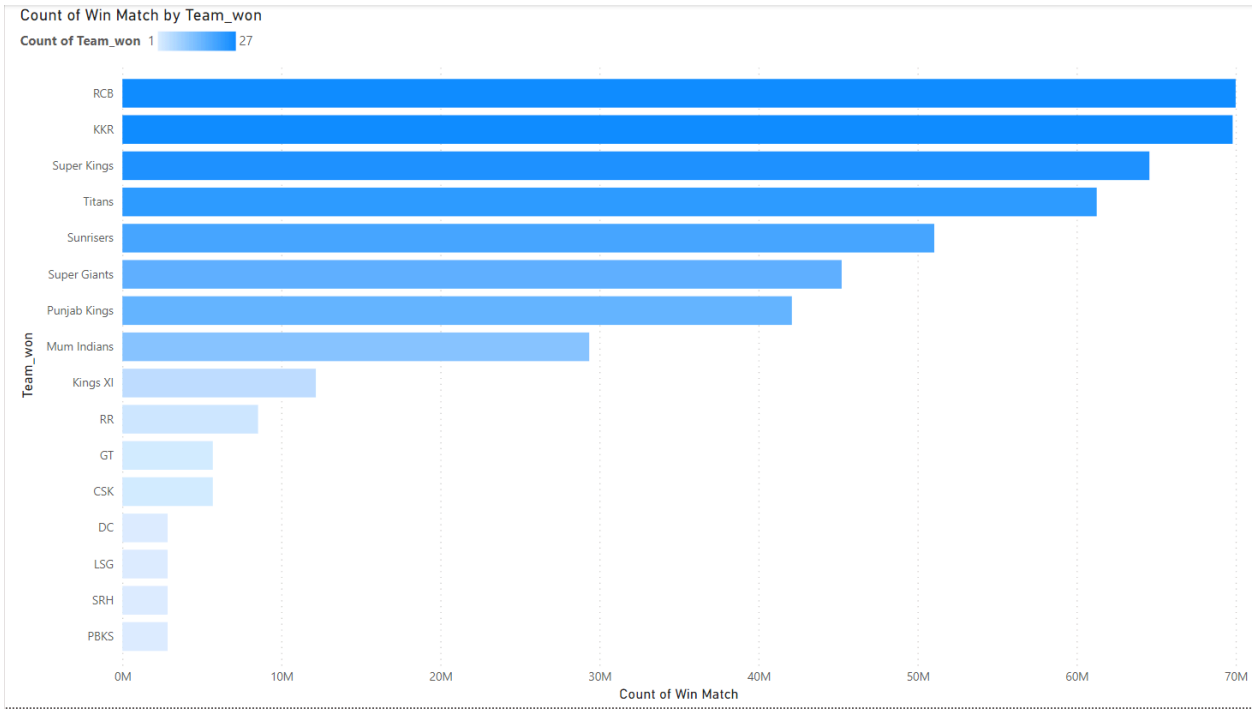


4. **In this navigator,we can select tables which you have to load into our powerBI.**



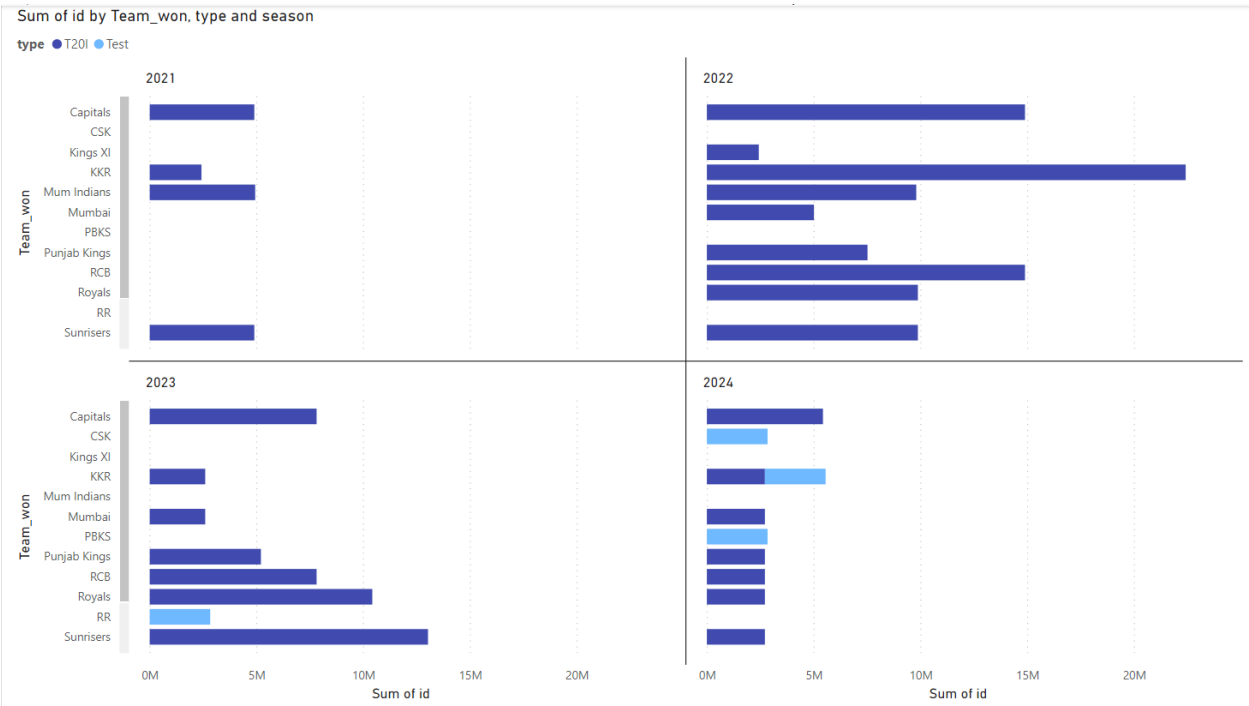
Report Screenshots :

1.



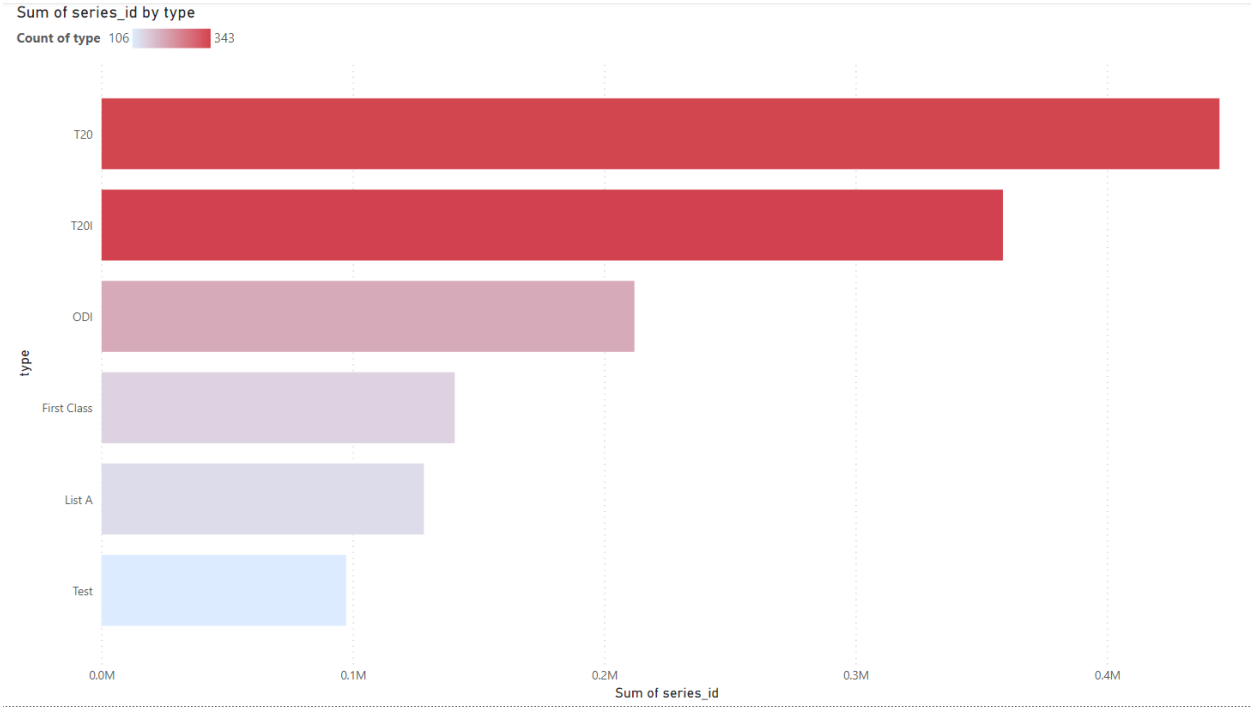
By this graph, we can represent which team has more wins than other teams.

2.



By this chart, we can show the winning team of each match by years (2021,2022,2023 and 2024).

3.



We can see that there is more data about T20, T20I and ODI series than other series in cricket api by this graph.

Conclusion:

This project emphasizes the significance of efficient data management and integration in today's competitive landscape. It entails developing a robust data model utilizing the Start/Snowflake Schema, acquiring Cricket data from third-party sources through RAPID APIs, storing it in Amazon S3 with Python, transferring it to Oracle, and migrating it to Snowflake using Talend. Additionally, the project involves implementing data validation and reporting in Power BI.

Efficient data management and integration are pivotal for organizational success. This project demonstrates the establishment of a strong data foundation and seamless processes, spanning acquisition, migration, and validation. By leveraging modern technologies such as Talend and Power BI, organizations can fully exploit their data assets, gaining valuable insights and maintaining a competitive edge in the data-driven landscape of today.