2017-2021 Spotify Chart Analytics

Created by Marcus Pestana

Introduction

The purpose of this project is to gather data primarily from Spotify Ranking Charts and from other data sources to enrich information about artists, resulting in an analytics star schema database for insights generation.

Data source #1 - Spotify Charts

This is a complete dataset of all the "Top 200" and "Viral 50" charts published globally by Spotify. Spotify publishes a new chart every 2-3 days. This is its entire collection since January 1, 2017.
LINK: https://www.kaggle.com/datasets/dhruvildave/spotify-charts

Data source #2 - MusicBrainz Artists popularity

The dataset consists of over 1.4 Million musical artists present in MusicBrainz database -- their names, tags, and popularity (listeners/scrobbles), based on data scraped from last.fm.
LINK: https://www.kaggle.com/datasets/pieca111/music-artists-popularity

Data source #3 - Countries and Continents

Contains basic JSON data for countries, such as ISO2, ISO3 codes, continents, capitals, phone prefixes and currency codes.
LINK: https://www.kaggle.com/datasets/pieca111/music-artists-popularity

Tools and Technologies

- Python
  - Its built-in analytics tools can also easily penetrate patterns, correlate information in extensive sets, and provide better insights, in addition to other critical matrices in evaluating performance.
  - Pandas
    - a powerful data analysis and manipulation tool, great for file ingesting, in this projectÇ the best choice for JSON
  - psycopg2
    - database adapter for PostgreSQL
- PostgreSQL
  - great for analytical queries with its window functions and easy to port to Redshift in case there is need for bigger data handling.

About the Data

Scope

Data source #1 provides information about Spotify's ranking charts, with song's titles, artists, chart ("Top 200" and "Viral 50") and its region ("Country"), but there is no further information from the artist itself. So with help from **data source #2** which holds information about artists and its popularity, it is possible to gather "country" and "tags" data from it, creating the possibility to generate insights and answering questions such as "How many times an artist from Brazil got in the top 10 rank charts?" or "Are there any jazz-related artists in the top 200?". **Data source #3** complements country information with "continent" data, resulting in more large-scale analytics.

Exploring and Cleaning

The data was loaded to staging tables in PostgreSQL and some queries where made for identifying errors

Charts Dataset

First of all, for creating dimensional table **songs**, the song's URL theoretically could be its primary key, since there's no way you would click it and result in 2 different songs. So this query was ran to ensure it:

```
SELECT url, count(DISTINCT title) as titles, count(DISTINCT artist) as
artists
      FROM public.staging_charts
    GROUP BY url
    ORDER BY count(DISTINCT artist) DESC, count(DISTINCT title) DESC;
```

Results:

url	titles	artists
"https://open.spotify.com/track/3K5JiDtBPEIni1IrjLIGOG"	2	3
"https://open.spotify.com/track/132wAeR05o10R9iSjA1XAE"	1	3
"https://open.spotify.com/track/6w8yBI2vthyN9UnwO4UBWb"	1	3
"https://open.spotify.com/track/7I8L3vYCLThw2FDrE6LuzE"	1	3
"https://open.spotify.com/track/2fzqpmKuYcNQS7clW1XxAS"	1	3

What happens is that some songs may have been updated, their artists credits, or even its name. So as there is no NULL URL as well, my solution was to get the most recent update using OVER and PARTITION in PostgreSQL, and storing its unique id substring.

songs_table_insert in **sql_queries.py:**

```
INSERT INTO dim_songs (
        song_id,
        name,
        artist_name
    )   SELECT DISTINCT SUBSTRING(url, 32), FIRST_VALUE(title)
OVER(PARTITION BY url ORDER BY date DESC),
                        FIRST_VALUE(artist) OVER(PARTITION BY url ORDER
BY date DESC)
                from public.staging_charts
    WHERE url IS NOT NULL;
```

As for the rest of its dimensions, there were no NULL values, so it was possible to create charts (with region) and artists dimensional tables. Only the column “streams” had NULL values, but as the creator stated: “The value of streams is NULL when the chart column is ‘viral50’.”, there is no such information for this type of chart.

Artists Dataset

As described by the author on the dataset page, there are a lot of duplicated artists, and even if we are only **LEFT** joining charts table (artists from charts table not matched will have country and other artist dimensions as NULL), we need to get the best row option for the join. So firstly, I focused only on MusicBrainz dimensions (with mb suffix), as country_lastfm has some rows with country and **city** name appended and would not join with **data source #3.**

```
SELECT mbid, artist_mb, artist_lastfm, country_mb, country_lastfm, tags_mb,
tags_lastfm, listeners_lastfm, scrobbles_lastfm, ambiguous_artist
                        FROM public.staging_artists
                        where country_mb is not null and
                        tags_mb is not null
```

Using this filtered select and "**DISTINCT ON**", I inserted only the first matched artists into the dimensional table:

artists_table_insert in **sql_queries.py:**

```
INSERT INTO dim_artists (
        name,
        country,
        continent,
        tags,
        listeners_lastfm,
        scrobbles_lastfm)

SELECT DISTINCT ON (c.artist) c.artist, a.country_mb, cou.continent,
a.tags_mb, a.listeners_lastfm, a.scrobbles_lastfm
            FROM public.staging_charts as c
            LEFT JOIN (SELECT mbid, artist_mb, artist_lastfm, country_mb,
country_lastfm, LOWER(tags_mb) as tags_mb, tags_lastfm, listeners_lastfm,
scrobbles_lastfm, ambiguous_artist
                        FROM public.staging_artists
                        where country_mb is not null and
                        tags_mb is not null) as a
            ON c.artist = a.artist_mb
            LEFT JOIN staging_countries as cou
            ON a.country_mb = cou.country_name
            WHERE c.artist IS NOT NULL
            ORDER BY c.artist;
```

Also used LOWER() on **tags** for easier querying.

Countries Dataset

This dataset contains more data than "countries and continents", but I only focused on those. When reading into Pandas dataframe, continent data was turned into columns, so I had to use melt function to turn into rows and join it:

load_staging_json in **etl.py:**

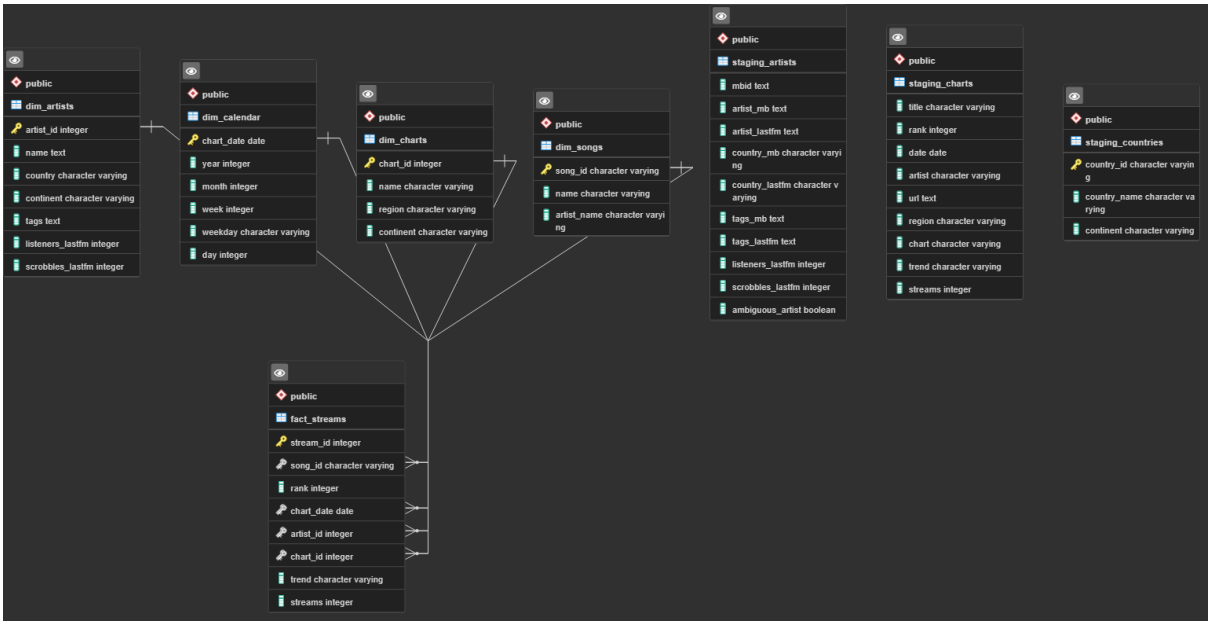
```
continent_df = pd.read_json(continent, lines=True)
    continent_df = pd.melt(continent_df, var_name='country_id',
value_name="continent")

    country_df = pd.read_json(country, lines=True)
    country_df = pd.melt(country_df, var_name='country_id',
value_name="country_name")

    join_df = pd.merge(country_df, continent_df, on='country_id',
how='left')
```

Data Model

I chose a staging tables approach for faster insert queries when cleaning and inserting data into the multidimensional schema, using copy statements in most data sources. The multidimensional schema tables are using a star schema design which is one of the best approaches for analytical databases, resulting in easy and fast queries. **Dimensional** tables (dim_artists, dim_songs, dim_calendar, dim_charts) hold data for filtering as the **Fact** table (fact_streams) holds the filtered data.



Data Dictionary

column name	description	table
artist_id	generated unique integer	dim_artists
name	artist's name	dim_artists
country	artist's country	dim_artists
continent	artist's continent	dim_artists
tags	artist's genres of music	dim_artists
listeners_lastfm	artist's listeners on LastFM	dim_artists
scrobbles_lastfm	Qty. of streams on 2017 on LastFM	dim_artists
chart_date	date that chart was released	dim_calendar
year	year that chart was released	dim_calendar

month	month that chart was released	dim calendar
weekday	day of week that chart was released	dim_calendar
day	day that chart was released	dim_calendar
chart_id	generated unique integer	dim_charts
name	chart's name (top200 or viral50)	dim_charts
region	country that the chart featured in	dim_charts
continent	continent that the chart featured in	dim_charts
song_id	song_id extracted from URL	dim_songs
name	song's name	dim_songs
artist_name	song's artist name	dim_songs
rank	rank that the song got in said chart and date	fact_streams
trend	string specifying if the song went up or down in the chart	fact_streams
streams	Number of times that the song was streamed	fact_streams

Pipelines

After creating each table, **data source #1** and **#2** data is copied in its entirety to staging tables **charts** and **artists** using copy queries:

load_staging_tables in etl.py:

```
def load_staging_tables(cur, conn):
    for query in copy_table_queries:
        print(query)
        cur.execute(query)
        conn.commit()
        print("### DONE ###")
```

sql_queries.py:

```
copy_artists_table_staging = (
    """COPY staging_artists
        FROM {0}
        DELIMITER ','
        CSV
        HEADER;
""").format(ARTIST_DATA)

copy_charts_table_staging = (
    """COPY staging_charts
        FROM {0}
        DELIMITER ','
        CSV
        HEADER;
""").format(CHART_DATA)
```

Data source #3 is loaded into **pandas,** transformed and inserted into staging_countries:

load_staging_json in etl.py:

```
continent_df = pd.read_json(continent, lines=True)
continent_df = pd.melt(continent_df, var_name='country_id',
value_name="continent")

country_df = pd.read_json(country, lines=True)
country_df = pd.melt(country_df, var_name='country_id',
value_name="country_name")

join_df = pd.merge(country_df, continent_df, on='country_id', how='left')
    for index, row in join_df.iterrows():
        cur.execute(insert_countries_table_staging, (row['country_id'],
row['country_name'], row['continent']))
conn.commit()
```

sql_queries.py:

```
insert_countries_table_staging = (
    """INSERT INTO staging_countries(country_id, country_name, continent)
        values(%s, %s, %s);
""")
```

After staging, all data is inserted into the star schema tables with **insert** statements:

sql_queries.py:

```
artists_table_insert = (
    """
    INSERT INTO dim_artists (
        name,
        country,
        continent,
        tags,
        listeners_lastfm,
        scrobbles_lastfm)

    SELECT DISTINCT ON (c.artist) c.artist, a.country_mb, cou.continent,
a.tags_mb, a.listeners_lastfm, a.scrobbles_lastfm
            FROM public.staging_charts as c
            LEFT JOIN (SELECT mbid, artist_mb, artist_lastfm,
country_mb, country_lastfm, LOWER(tags_mb) as tags_mb, tags_lastfm,
listeners_lastfm, scrobbles_lastfm, ambiguous_artist
                        FROM public.staging_artists
                        where country_mb is not null and
                        tags_mb is not null) as a
            ON c.artist = a.artist_mb
            LEFT JOIN staging_countries as cou
            ON a.country_mb = cou.country_name
            WHERE c.artist IS NOT NULL
            ORDER BY c.artist;
    """
)

songs_table_insert = (
    """
    INSERT INTO dim_songs (
        song_id,
        name,
        artist_name
    )   SELECT DISTINCT SUBSTRING(url, 32), FIRST_VALUE(title)
OVER(PARTITION BY url ORDER BY date DESC),
                        FIRST_VALUE(artist) OVER(PARTITION BY url ORDER
BY date DESC)
```

```
                from public.staging_charts
    WHERE url IS NOT NULL;
    """
)

calendar_table_insert = (
    """
    INSERT INTO dim_calendar (
        chart_date,
        year,
        month,
        week,
        weekday,
        day
    )   SELECT DISTINCT
                date,
                EXTRACT(year from date),
                EXTRACT(month from date),
                EXTRACT(week from date),
                EXTRACT(isodow from date),
                EXTRACT(day from date)
        FROM staging_charts;
    """
)

charts_table_insert = (
    """
    INSERT INTO dim_charts (
        name,
        region,
        continent
    )   SELECT DISTINCT c.chart, c.region, cou.continent
      FROM public.staging_charts as c
    LEFT JOIN staging_countries as cou
    ON c.region = cou.country_name
    WHERE chart IS NOT NULL;
    """
)

streams_table_insert = (
    """
    INSERT INTO fact_streams (
        song_id,
        rank,
        chart_date,
        artist_id,
```

```
        chart_id,
        trend,
        streams
    )
    SELECT DISTINCT SUBSTRING(stg.url, 32), stg.rank, stg.date,
                    a.artist_id, c.chart_id, stg.trend, sum(stg.streams)
      FROM public.staging_charts as stg
    LEFT JOIN dim_artists as a
    ON stg.artist = a.name
    LEFT JOIN dim_charts as c
    ON stg.chart = c.name AND
       stg.region = c.region
    WHERE stg.artist IS NOT NULL
    AND stg.chart IS NOT NULL
    GROUP BY SUBSTRING(stg.url, 32), stg.rank, stg.date,
                    a.artist_id, c.chart_id, stg.trend;
    """
)
```

After all data is loaded, quality checks are made:

```
artist_quality_check = """SELECT COUNT(artist_id)
                          FROM dim_artists
                          WHERE artist_id IS NULL ;"""
song_quality_check = """SELECT COUNT(song_id)
                          FROM dim_songs
                          WHERE song_id IS NULL ;"""
chart_quality_check = """SELECT COUNT(chart_id)
                          FROM dim_charts
                          WHERE chart_id IS NULL ;"""
calendar_quality_check = """SELECT COUNT(chart_date)
                          FROM dim_calendar
                          WHERE chart_date IS NULL ;"""
stream_quality_check = """SELECT COUNT(stream_id)
                          FROM fact_streams
                          WHERE stream_id IS NULL;"""
```

```
data_quality_checks = [{'sql':artist_quality_check  , 'expected_result':0},
                       {'sql':song_quality_check  , 'expected_result':0},
                       {'sql':chart_quality_check  , 'expected_result':0},
                       {'sql':calendar_quality_check  , 'expected_result':0},
                       {'sql':stream_quality_check  , 'expected_result':0}]
```

How to Run

First, configure db_connection.cfg with PostgreSQL credentials and data sources file path, install Python dependencies with pip, run create_tables.py and etl.py

```
pip install -r requirements.txt
python create_tables.py
python etl.py
```

All pipelines would be done and you should receive an OK from all Data Quality checks.

Results

After all work is done, it is possible to run any analytical query within artists, songs, charts and calendar (date when the chart was released). Example:

"Are there any jazz-related artists in the top 200?"

```
SELECT COUNT(DISTINCT f.artist_id) as artists
      FROM public.fact_streams as f
    LEFT JOIN dim_artists as a ON f.artist_id = a.artist_id
    LEFT JOIN dim_charts as c ON f.chart_id = c.chart_id
    WHERE a.tags LIKE '%jazz%' AND c.name = 'top200';
```

RESULT: 140 artists

"How many times an artist from Brazil got in the top 10 rank charts?"

```
SELECT COUNT(DISTINCT f.stream_id) as times
      FROM public.fact_streams as f
    LEFT JOIN dim_artists as a ON f.artist_id = a.artist_id
    WHERE a.country LIKE '%Brazil%' AND f.rank <= 10;
```

RESULT: 3.367 times

“Top 10 songs from north american rock artists featured in Top200 chart in the year 2020”

```
SELECT s.name as song, a.name as artist,
        COUNT(DISTINCT f.stream_id) as hits, sum(f.streams) as streams
      FROM public.fact_streams as f
    LEFT JOIN dim_artists as a ON f.artist_id = a.artist_id
    LEFT JOIN dim_charts as c ON f.chart_id = c.chart_id
    LEFT JOIN dim_songs as s ON f.song_id = s.song_id
    LEFT JOIN dim_calendar as dt ON f.chart_date = dt.chart_date
    WHERE a.tags LIKE '%rock%' AND c.name = 'top200' AND
          a.continent LIKE 'NA' AND dt.year = 2020
    GROUP BY s.name, a.name
    ORDER BY sum(f.streams) DESC
    LIMIT 10;
```

RESULT:

song	artist	hit times	streams
"Memories"	"Maroon 5"	17.060	1.161.623.512
"Rain On Me (with Ariana Grande)"	"Lady Gaga"	10.009	875.662.668
"Believer"	"Imagine Dragons"	8.473	460.031.448
"Take You Dancing"	"Jason Derulo"	6.048	411.601.521
"Midnight Sky"	"Miley Cyrus"	5.394	407.459.725
"Lose You To Love Me"	"Selena Gomez"	6.525	399.875.498
"Play Date"	"Melanie Martinez"	5.376	379.826.434
"Wonder"	"Shawn Mendes"	4.147	313.290.976
"cardigan"	"Taylor Swift"	2.672	274.327.253
"Rockin' Around The Christmas Tree"	"Brenda Lee"	1.647	255.306.312

Future

Regarding information quality, MusicBrainz's dataset is not as accurate as it could be, as some artists may have wrong location and musical genre data. So finding a more accurate data source would be a great addition.

Updates

As Spotify's chart releases occur in 2-3 days, updating its data every 3 days would be sufficient, as for MusicBrainz's dataset daily updates would be better, to get recent information as soon as possible. Countries dataset as a territorial data itself, it can last a long time with no updates.

Other scenarios

- If the data was increased by 100x.
    - Spark would need to be integrated, such amount of data would need to be handled by a larger quantity of nodes
- If the pipelines were run on a daily basis by 7am.
    - Airflow would need to be integrated for better control of the pipelines and time management
- If the database needed to be accessed by 100+ people.
    - A cloud data warehouse would be a great solution, for roles and permission management as well for better availability