University of Tours

B.D.M.A

Big Data Management Analytics

Report Phase 1 Decisional Project

Author:
Akaichi, Ines
Cissé, Ismaila
de Saint Ceran, Louis
Nascimento Filho, Jessé F.

November 19, 2018

Contents

1 Introduction

Not long ago, there was a strong belief that the internet was killing the music industry. For years now, the industry has failed to keep up with the rapid pace of technological advancement and barely had any understanding of their audience, who was buying their CDs, or cassettes.[9]

With streaming services taking over, these companies found ways to track the listening habits of users and have access to detailed information such as when, how, where, and who is listening to what.

The aim of the industry, now, is to use these customer behavior insights together with knowledge of the music itself, which is made possible only with Big Data. The raw music that is produced is essentially like unstructured data. In the digital era, this raw music can be easily digitized and analyzed.

As part of our master degree 's program , we choose to work on the design and implementation of a business intelligence system[10] supporting the analysis of songs. Our objective is to analyze the current music streaming industry and more precisely the songs and the artists' popularity. We decided to bring the subject closer to us by focusing on French artists.

2 Management Methodology

Our team is a small multicultural and autonomous group, for this reason we found a common equation to process with ours tasks project life cycle. The result from this equation become our management methodology that is based in a lean solid and agile solution called Scrum. For this purpose we decided to follow the best practices of Scrum combined with a Kanban approach.

2.1 Scrum

The Scrum arise as a process framework to manage complex projects since the early 1990s. The essence of Scrum is to provide effective iterations and incremental knowledge transfer to the success of a project [11].

Our Project will be composed of 4 sprints during this semester:

- Sprint 0 — Study : The list of requirements and project planning;

- Sprint 1 — Model : Conception & modeling of data warehouse;
- Sprint 2 — ETL : Data Extraction, Transformation and Loading;
- Sprint 3 — DEMO : B.I system demonstration of an initial version;
- Sprint 4 — DEFENSE : Oral project presentation;

2.1.1 Schedule

[1]

2.1.2 Sprint 0

The name SPRINT 0 has been learned to describe the preparation phase which precedes the launching of the project. The term SPRINT 0 is being simpler to use than the preparation or inception phase, it is increasingly used in SCRUM projects. Sprint 0 does not diminish the flexibility of our project. On the contrary, it will allow us to anticipate certain actions and have an overview that will facilitate the management of changes that will emerge at the following sprints. [7]

In this Sprint we will be able to :

1. Share a clear vision of the project;
2. Identify users need;
3. Identify the preliminary workload resulting of the users need;
4. Prepare the project management plan;

2.1.3 Sprint 1

The preliminary specification of the workload in sprint 0 will help us in this sprint in modeling our data warehouse,thus the formalization of the entire workload. In addition, in this phase it is essential to maintain an active technological watch to choose our Essential BI tools used in next sprints.

2.1.4 Sprint 2

In this sprint we will be able to define our data warehouse 's architecture, assess the data quality and implement the designed ETL system.

2.1.5 Sprint 3

In this sprint we will be able to visualize our data using the BI restitution tools .

2.1.6 Sprint 4

In this final sprint , We will be able to prepare an oral presentation where we summarize all the steps that we have gone through when developing our project 's data warehouse and present our work to our professors.

2.2 Kanban

In addition of Scrum methodology we choose to use Kanban approach, that means "visual card" in Japanese, to help us simplify the sprints workload. We going to make a visual work-flow using Trello for create and manage all cards with micro tasks, it will result in each sprints deliveries milestones.

2.2.1 Trello

Trello[6] is a project management software that utilizes the concept of boards to represent projects and within boards, cards to represent tasks. Trello supports Team Collaboration enabling members to discuss a project in real-time. It keeps everybody informed through task assignments, activity log, and e-mail notifications.[2]

3 Preliminary Workload

3.1 Data Sources

The main services of stream nowadays are responsible for creating an efficient way to capture data and generate real information about the customers with it, but to obtain the success in this new world where data is like gold, they need to provide a service that is capable to get users loyalty. For this reason companies and communities of independent artists are building continuously distinct forms of technologies to present not only songs, but music with value add on it, such as, variety, quality and shareability. As a result of this frequent process innovation, today It's possible for any person to have access to huge an open-sources databases about music subjects. Hence,to make decisional projects become more simple with the follow assets MusicBrainz encyclopedia[3] and Spotify Web API. We choose to use into our project these datasets and, if necessary, others data sources could be attached as a asset during this project.

MusicBrainz Database[3] includes information about artists, release groups, releases, recordings, works, and labels, as well as the many relationships between them [table 1]. It is a community-maintained open source encyclopedia of music information. This means that anyone can help contribute to the project by adding information about your favorite artists and their related works.The entire dataset is 1.8 GB.

Attributes	type	granularity
released songs	text	by title
location	text	by region, country
genre	text	by binary
date	timestamp	by month,year
artists	text	by name, alias
origins	text	by country
cover	text	by name

Table 1: A brief description of MusicBrainz Database 's attributes.

Spotify Web API [4] Based on simple REST principles, the Web API endpoints return metadata about music artists, albums, and tracks[table 10], directly from the Spotify Data Catalogue.

Attributes	type	granularity
popularity	number	0-100
energy	float	0.0 to 1.0
valence	float	0.0 to 1.0
genre	text	by binary
danceability	float	0.0 to 1.0
duration	number	by by milliseconds
loudness	float	by decibels (dB)
mode	int	by major, minor
tempo	float	by beats per minute (BPM)

Table 2: A brief description of Spotify Web API 's attributes.

3.2 User Needs

Statistics about songs and artists:

1. Number and Average of released songs disaggregating by genre, year , location and artist .
2. The biggest/ lowest number of released songs by location , genre , year and artist .
3. Number of artists disaggregating by origins.
4. Number of artists appeared every year in the music industry.
5. Number of songs or artists that achieved a certain popularity.
6. The average popularity of the songs where artists participated to analyze artist's performance.
7. What is the less popular songs by country and find out why ?
8. What is the impact of cover art on success of an album? Number of recorded covers disaggregating by artist and song .
9. The most covered songs by artist and song .
10. Artists that are most engaged in the last years.
11. What makes a top performer based on songs 's technical features?

Statistics about song musical features:

- Average duration, average tempo by artist and/or location and/or year.

- What makes a tube based on culture, market, political time, features of the song or the category of the song.

4 Data Warehouse Modelling

4.1 Conceptual Design

Conceptual modeling is widely recognized to be the necessary foundation for building a database that is well-documented and fully satisfies the user requirements. In particular, from the designer point of view the availability of a conceptual model provides a higher level of abstraction in describing the warehousing process and its architecture in all its aspects.

For modelling our data warehouse schema we used The Dimensional Fact Model or DFM which is a graphical conceptual model, specifically devised for multidimensional design. In subsection 4.2.1 , we present our conceptual models corresponding to our proposed fact schemas .

4.1.1 Conceptual Schemas

In order to represent our DFM models , we used the requirement-based design approach. It is a bottom-up technique that allowed us to do the mapping between our analyzed requirements in subsection 3.2 onto our available data source so that we get to locate the conceptual objects at sources and make sure that the data verifying our user needs effectively exists.

Using this approach , We first start by defining the set of facts , measures and finally dimensions. Figure 1 shows the fact schema for the analysis of songs and artists musical features in which the fact consists of a released song with all its features in the month ,city of release ,the song 's genre and the group artist that released the song .

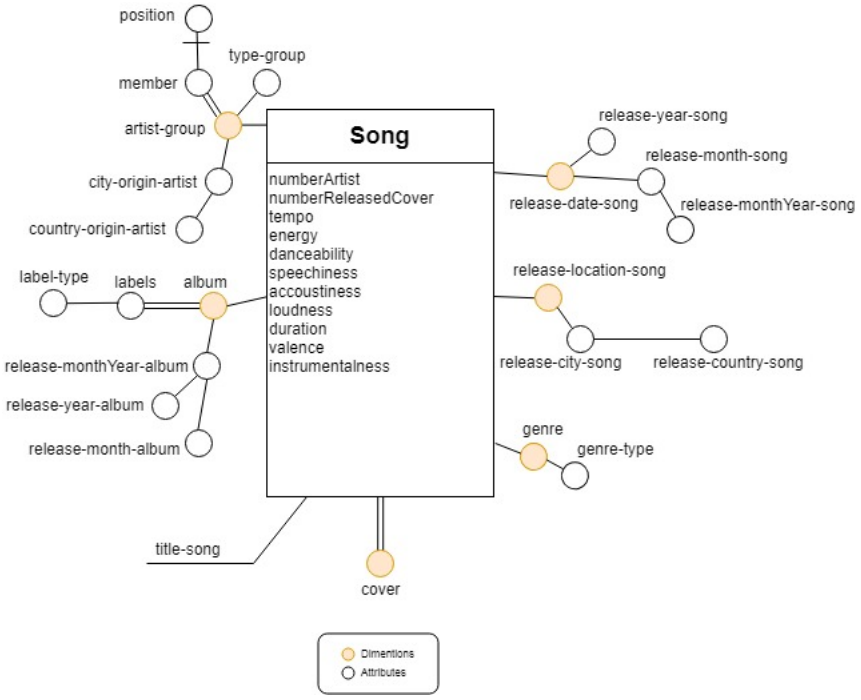

Figure 4.1.1-1: The Dimension Fact model for song.

Figure 2 and 3 shows the fact schema for the analysis of songs and artists popularity in the which the fact consists of song's popularity (figure 2) and the artist 's popularity (figure3) in every month .

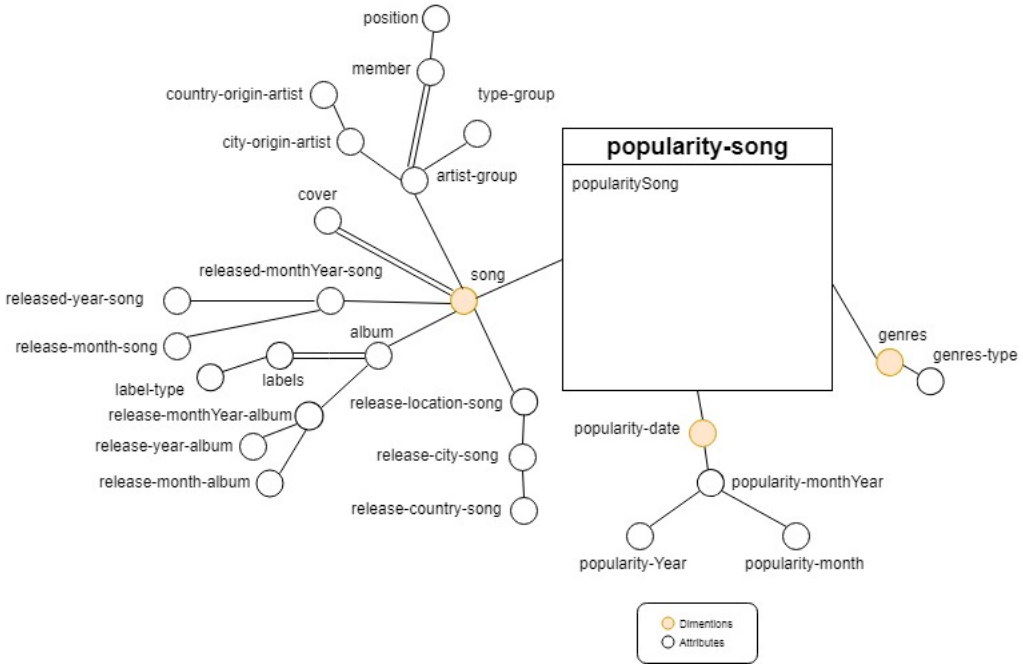

Figure 4.1.1-2: The Dimension Fact model for song popularity.

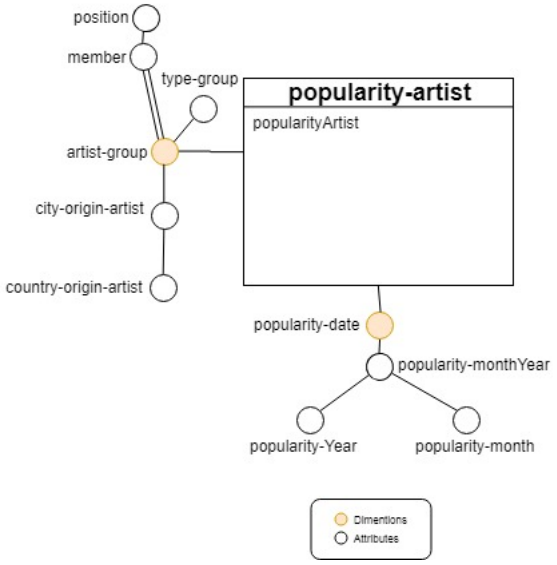

Figure 4.1.1-3: The Dimension Fact model for artist popularity.

4.1.2 Additivity Matrix

In the additivity matrix we identify all the aggregation functions applied on measures and grouped by dimensions .

Table 3 shows the equivalent additivity matrix for the fact schema in figure 1.

	Group-artist	Release-year-song	Release-city-song	genre
numberArtist		Avg	Sum Avg	Sum Avg
numberReleasedCover	Max Avg	Avg	Avg	
numberSong	Max Min	Avg Max Min	Avg Max Min	Avg Max Min
tempo	Avg	Avg	Avg	Avg
energy	Avg	Avg	Avg	Avg
danceability	Avg	Avg	Avg	Avg
speechiness	Avg	Avg	Avg	Avg
accoustiness	Avg	Avg	Avg	Avg
loudness	Avg	Avg	Avg	Avg
duration	Avg	Avg	Avg	Avg
valence	Avg	Avg	Avg	Avg
instrumentalness	Avg	Avg	Avg	Avg

Table 3: Additivity matrix for Fact song.

Table 4 shows the equivalent additivity matrix for the fact schema in figure 2.

	Song	Popularity-month	Group-artist	Popularity-year
popularitySong	Min Max Avg	Min Max Avg	Min Max Avg	Min Max Avg

Table 4: Additivity matrix for Fact Popularity-Song .

Table 5 shows the equivalent additivity matrix for the fact schema in figure 3.

	Group-artist	Popularity-month	popularity-Year
popularityArtist	Min Max Avg	Min Max Avg	Min Max Avg

Table 5: Additivity matrix for Fact Popularity-Artist .

4.1.3 Data Dictionary

An important part of any software project is to be able to provide information in a clear and accessible way. For this purpose, to have, previously, a list with all of data variable names and description allow an efficient approach to have access to the definition of each metadata. It makes the action of manage different terminologies, formats and contents less painful for the team and users.

For this reason we created a dictionary that contents all of our attributes names, the respective description and if it is a measure (M), dimension (D) or attribute (A). Some description was imported from origin sources, however we made some changes to make more clear each attributes denotation.

Name	Description	M	D	A
artist-group	it could be a solo singer (e.g. "Drake"), group (e.g."AC DC") or lineup (e.g. "Zedd, Maren Morris and Grey") that the release is primarily credited (string).	no	yes	no
accoustiness	it is a float confidence measure from 0.0 to 1.0 of whether the track is acoustic, e.g. 1.0 represents high confidence the track is acoustic.	yes	no	no
album	is a string with the album name e.g. "Samedi soir sur la Terre"	no	yes	no
city-origin	origin city of an artist or a band e.g. "Liverpool"(string).	no	no	yes

Table 6: Data dictionary for dimensions, attributes and measures.

Name	Description	M	D	A
country-origin	origin country of an artist or a band e.g. "United Kingdom" (string).	no	no	yes
cover	is artist, lineup or band that make a new performance or recording of a song that already exists, e.g. Andrei Zevakin, Ariadne and Martti Hallik.	no	yes	yes
danceability	is a measure that describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A track closer to 0.0 is least danceable than a music near 1.0 that is most danceable, e.g. Cut To The Feeling has a 0.696 of danceability.	yes	no	no
duration	the duration of the track in milliseconds(Float).	yes	no	no
energy	is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity, e.g. Cut To The Feeling has a energy 0.905 that means a high level of activity. Combined with others measure can suggest feelings.	yes	no	no
genre	is the dimension for song genre. The genre of the song usually inherited from the album 's genre.	no	yes	no
genre-type	is the most popular type of a song genre. e.g. "Rock", "Hip-hop", "Blues", Jazz", etc.	no	no	yes
labels	the label which issued the release. There may be more than one for every released song.	no	yes	no
label-type	the label which issued the release. There may be more than one for every released song , e.g. Original Production, Reissue Production, None and Publisher (string).	no	no	yes

Table 7: Data dictionary for dimensions, attributes and measures.

Name	Description	M	D	A
instrumentalness	is a float measure that predicts whether a track contains no vocals. “Ooh” and “aah” sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly “vocal”. The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.	yes	no	no
loudness	it is a float measure of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative noise into a track. It is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typical range between -60 and 0 db.	yes	no	no
member	it could be a band member e.g. the current members of AC DC are Angus Young, Chris Slade, Stevie Young and W. Axl Rose (string).	no	yes	yes
numberArtist	is a measure to known the number of artists in a group or band, even in a solo arrangement. If artist-group is composed of one artist e.g. ”Lady Gaga”, numberArtist takes 1 as a value.	yes	no	no
numberSong	an integer measure to know how many songs a group artist have released .	yes	no	no
numberReleasedCover	a measure to know how many times a song has covered by an artist or group (Integer).	yes	no	no
popularity-month	the number month that corresponds to the value of the popularity was extracted , e.g. ”11” (string).	no	no	yes
popularity-year	the year that corresponds to the value of the popularity was extracted, e.g. ”2017” (”YYYY” string).	no	no	yes

Table 8: Data dictionary for dimensions, attributes and measures.

Name	Description	M	D	A
popularitySong	is a measure that present popularity of the track. This value will be between 0 and 100, whether closer to 100 being is the most popular. Songs that are being played a lot now will have a higher popularity than songs that were played a lot in the past. Duplicate tracks (e.g. the same track from a single and an album) are rated independently.	yes	no	no
position	is a optional attribute that can have as a string value the role(s) of a member in a band, e.g. Singer, Drummer or if does not exist a position listed "unknown".	no	no	yes
release-date-song	is a dimension with the first date of released songs.	no	yes	no
release-monthYear-song	month and year number of the first song released. e.g. "082018" ("MMYYYY" - string).	no	no	yes
release-month-song	month number of the first song released. e.g. "08" ("MM" - string).	no	no	yes
release-year-song	year number (e.g. "2018") of the first song release ("YYYY" - timestamp).	no	no	yes
release-month-album	month of the first release of the album(e.g. "April", in string)	no	no	yes
release-year-album	year of the first release of the album(e.g. "YYYY", in string).	no	no	yes
release-location-song	is a dimension with a release song locations.	no	yes	no
release-city-song	release city of a song	no	no	yes
release-country-song	release country of a song .	no	no	yes
song	it's a dimension that contains song 's features.	no	yes	no

Table 9: Data dictionary for dimensions, attributes and measures.

Name	Description	M	D	A
speechiness	is a float measure that can detect the presence of spoken words in a track.The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.	yes	no	no
valence	is a float measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).	yes	no	no
type-group	it represent the type of a group of musicians (e.g. band, orchestra) or an artist (e.g. solo) (string).	no	no	yes
tempo	is a measure of speed or pace of a given piece and derives directly from the average beat per minute (BPM) duration e.g. Billie Jean have 92 BPM (integer).	yes	no	no
title-song	a string that contains a music name or a title of a release e.g. "La corrida" .	no	no	yes

Table 10: Data dictionary for dimensions, attributes ans measures.

4.2 Formalization of Requirements

In this section ,we tried to refine our workload by identifying queries that are expressed directly on the conceptual model in a formal language in order to validate our conceptual models.

- user need 1/2:
  - SONG[genre,release-year-song,release-city-song,artist-group]. numberSong
- user need 3:
  - SONG[city-origin-artist].numberArtist
- user need 4:
  - SONG[release-year-song;title-song IN [title-song,release-year-song,artist-group, Min(release-year-song)].title-song].numberArtist
- user need 5:
  - POPULARITY-ARTIST[artist-group;popularity > X].numberArtist
  - POPULARITY-SONG[title-song;popularity > X].numberSong
- user need 6:
  - POPULARITY-SONG[artist-group].popularitySong
- user need 7:
  - POPULARITY-SONG[Song,release-country-song; popularitySong=[POPULARITY-SONG[release-country-song].Min(popuaritySong)]
- user need 8:
  - POPULARITY-SONG[coverOrNot].popularitySong
  - SONG[artist-group].numberReleasedCover
- user need 9:
  - SONG[artist-group,title-song,coverOrNot=1].numberReleasedCover
- user need 10:
  - SONG[artist-group,title-song].Max(numberReleasedCover)
- user need 11:
  - SONG+POPULARITY-SONG[;popularity > X].tempo

  - SONG+POPULARITY-SONG[;popularity > X].energy

- user need 12:
  - SONG[artist-group;release-year-song >= 2016 and release-year-song <=2018].Max(numberSong)
- user need 13:
  - SONG[artist-group,release-city-song,release-year-song].Avg(Duration)
  - SONG[artist-group,release-city-song,release-year-song].Avg(Tempo)
- additional request 1:
  This query calculates the popularity of group artists in each month in 2016 and its members are six or less:
  - Popularity-artist[month, groupArtist; year='2016', artist-group in SONG[year, artist-group; year = '2016' and numberArtist <= 6].artist-group].popularityArtist

4.3 Prototypes of Charts

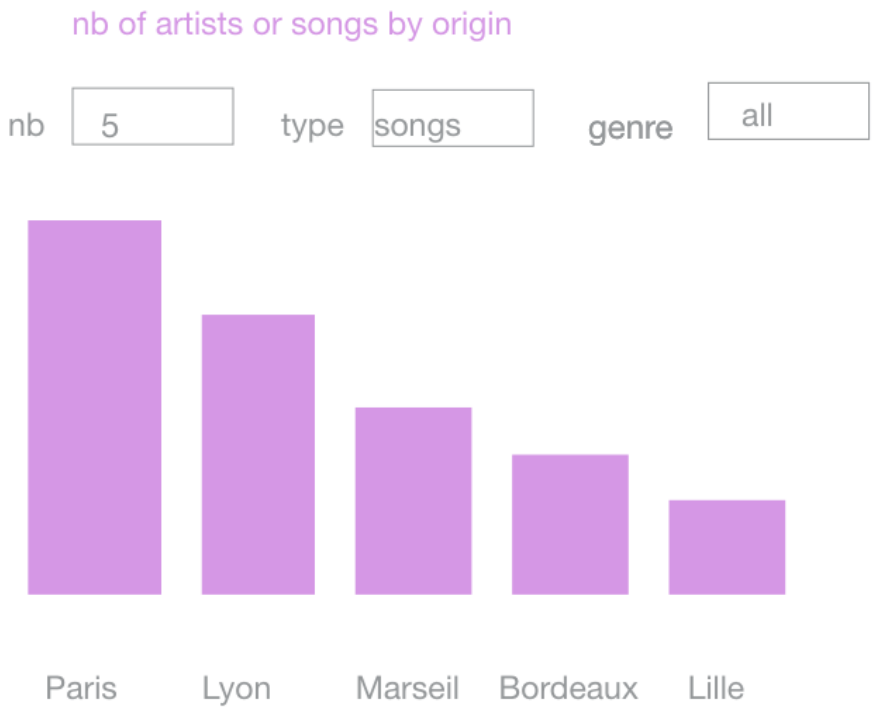

Figure 4.3.0-4: prototype of figure presenting the number of songs or artists by city

Figure 4.3.0-5: prototype of figure presenting the re-partition of artists and song function of popularity

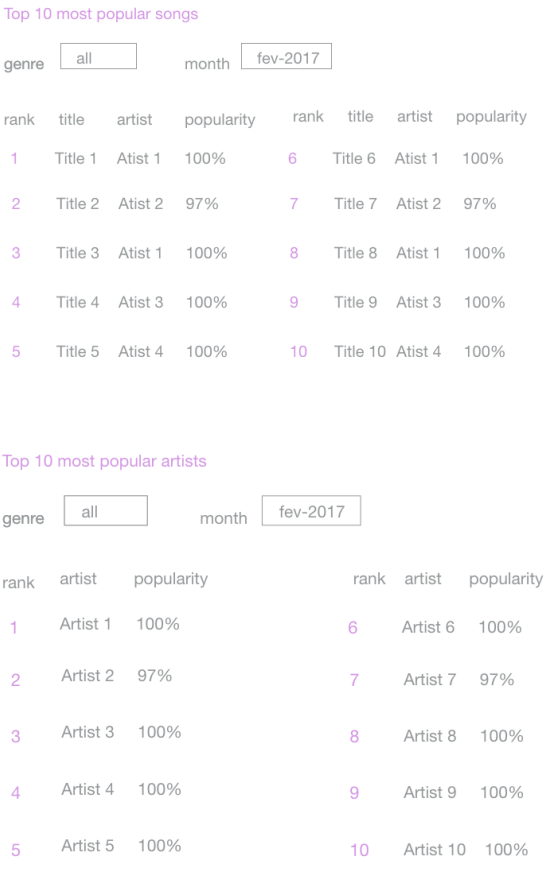

Figure 4.3.0-6: ranking of song and artist by popularity

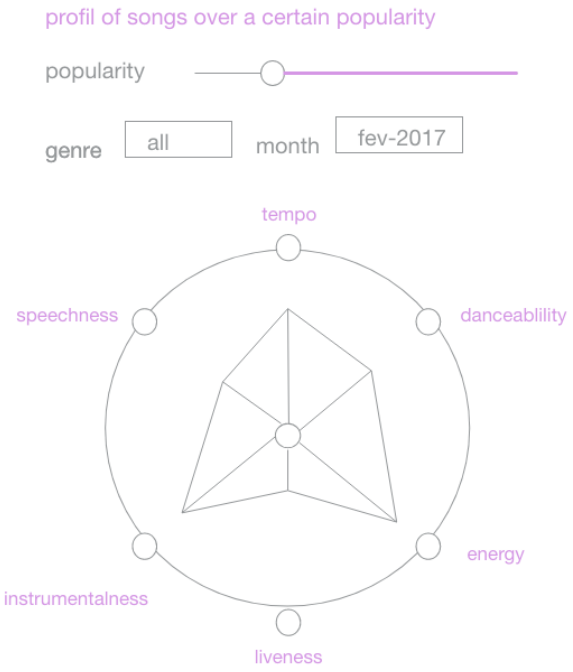

Figure 4.3.0-7: prototype of figure presenting the average technical measures for songs over a certain popularity

5 Data Warehouse Genesis

Data warehousing is a phenomenon that grew from the huge amount of data stored in recent years and from the urgent need to user that data to accomplish goals that go beyond the routine tasks linked to daily processing to quantify and evaluate performance, risk, cost, etc [8]. The analytic users, grace to technological progress, have access an advanced set of data and services, that once integrated can be a robust tool to help make a process decision.

For this purpose is necessary search data sources, clean it and find the best methodology to integrate it according to users need. Thus to reach our project goals a set of open-source technologies, see below, was considered needful:

1. Data warehouse Architecture;
2. Extraction process;
3. Transformation and process;

5.1 Data warehouse Architecture

Our data warehouse architecture includes the following layers (see figure 5.1) :

- Data source layer
- Data Staging layer
- Data Storage layer

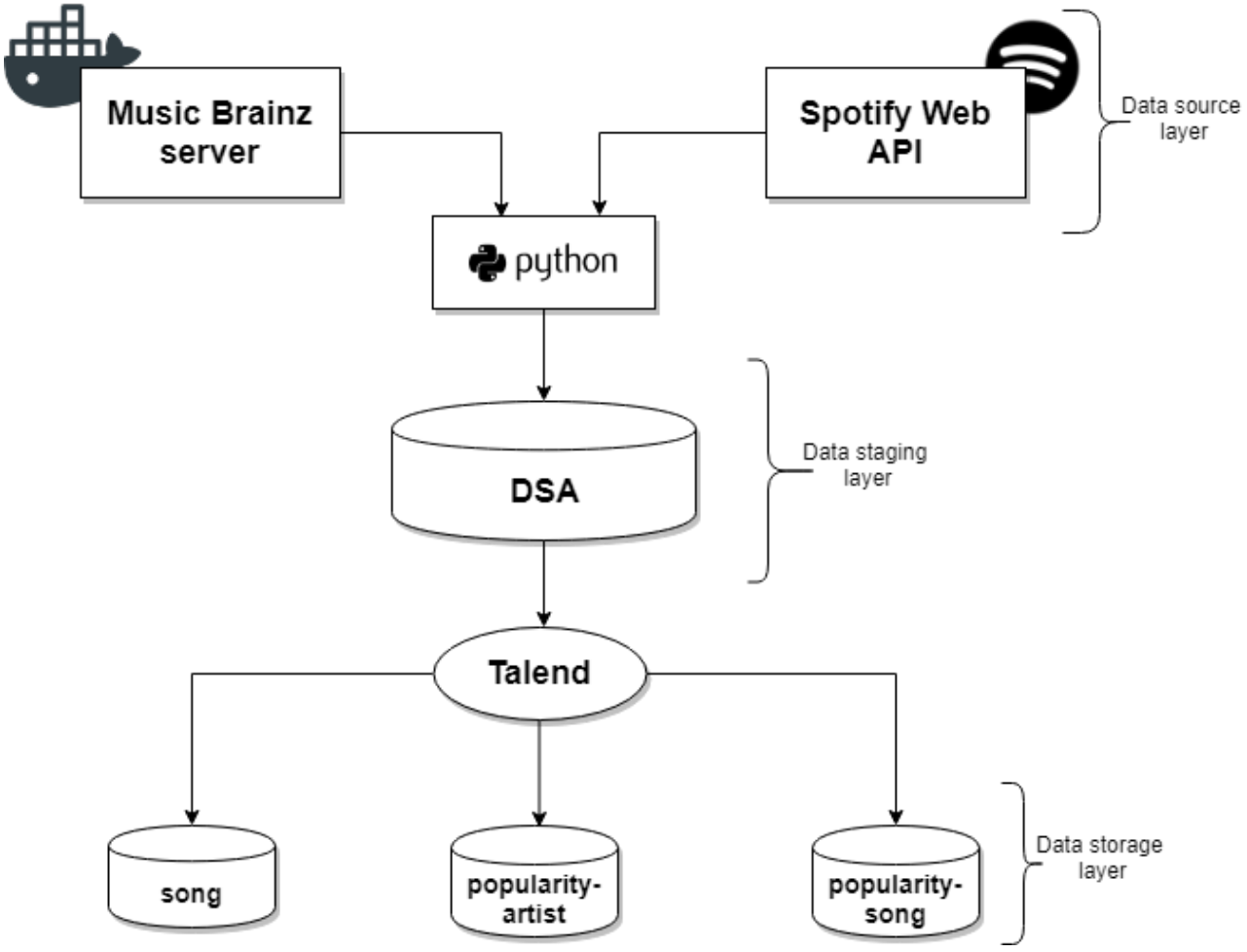

Figure 5.1.0-8: Data warehouse Architecture

5.1.1 Data source layer

The data in our data warehouse is collected from two different sources :

- The music brainz database that is deployed by us in a server running inside a docker container inside our linux virtual machine.
- The spotify web api Based on simple REST principles and that return JSON metadata about music artists, albums, and tracks, directly from the Spotify Data Catalogue.

The two sources are used to extract data related to artists, albums and songs using python scripts and a python library called Spotipy which gave us full access to all of the music data provided by the Spotify platform .

The result sets are directly stored in the staging database.

5.1.2 Data staging layer

We used the data staging area or DSA to store the extracted data in its raw format before transformations.The reason for using the dsa is that the data that comes from spotify is in json format so we need a landing stage where we can find all data from different sources in the same format.
Our DSA resides on a postgres server deployed on our virtual machine.

5.1.3 Data Storage layer

After transformations and cleansing using the Talend tool , data is stored in a ROLAP database .
We find in this layer , three datamarts : the song datamart,the popularity-artist datamart and popularity-song datamart. Our datamarts reside on a postgres server deployed on our virtual machine.

5.2 Extraction, Transformation and Load process

5.2.1 Extraction process

For this, we used three python scripts:
the first one extracts artists from music brainz database and search for every extracted artist in spotify returning their ids.

The second script, based on the ids of artists in spotify, we extracted their albums and ids.
Finally, the third script extracts the tracks and its features for every extracted album.

The reason behind using this strategy is that the spotify api methods need the ids of artists, albums and tracks to return the requested data.

5.2.2 Script that extracts artists

```
#!/usr/bin/env python
# coding: utf-8
#python -m pip install xlsxwriter
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
import pandas as pd
import time
from openpyxl import load_workbook
import os.path
import psycopg2
import xlsxwriter
#import pgdb

hostname = '10.195.25.10'
username = 'userbkp'
password = 'goma123'
database = 'musicbrainz'
port='54587'
cid ="a154f4842df643a99f9057fb741c86e0"
secret = "543080531fce4f30be3da2c36782ace1"

today_date=(time.strftime("%d_%m_%Y  %M_%S"))
date=(time.strftime("%d/%m/%Y"))
date_monthYear = (time.strftime("%m%Y"))
date_year = (time.strftime("%Y"))

filepath = 'F:\Documents\dproject_bdma2018\DATA'+today_date+'.xlsx'


checkifexist ="""SELECT id_msbz
```

```
                FROM public.stage_artist
                WHERE id_msbz ="""


query = """SELECT distinct artist_table.name_group,
            artist_table.id as id_msbz,
            artist_table.position as id_member_position,
            artist_table.type as id_type_group,
            artist_table.area as id_country_origin,
            area_table.countrycity_name as country_origin,
            artist_table.begin_area as id_city_origin ,
            (SELECT musicbrainz.area.name
            FROM musicbrainz.area
            WHERE musicbrainz.area.id = artist_table.begin_area
            AND musicbrainz.area.type = 3) as city_origin,
            gender_table.name as gender_sex

            FROM (SELECT musicbrainz.artist.name as name_group,*
              FROM musicbrainz.artist
              INNER JOIN musicbrainz.artist_credit_name
            ON musicbrainz.artist.id =
            musicbrainz.artist_credit_name.artist
            where musicbrainz.artist.type = 2) as artist_table
            LEFT JOIN musicbrainz.gender as gender_table
            ON artist_table.gender = gender_table.id
            INNER JOIN (select musicbrainz.area.id as id_area,
            musicbrainz.area.name as countrycity_name,
            musicbrainz.area_type.name as countrycity_type
                  FROM musicbrainz.area
                  INNER JOIN musicbrainz.area_type
                  on musicbrainz.area.type = musicbrainz.area_type.id
                  INNER JOIN (SELECT musicbrainz.area_alias.area as
                  id_area
                  FROM musicbrainz.area_alias
                  WHERE musicbrainz.area_alias.locale = 'en')
                  as area_alias_table
                  on musicbrainz.area.id = area_alias_table.id_area
                  WHERE musicbrainz.area_type.id = 1) as area_table
            ON artist_table.area = area_table.id_area
```

```
                WHERE artist_table.area is not null AND
                 artist_table.begin_area is not null
                AND artist_table.area != artist_table.begin_area"""

client_credentials_manager = SpotifyClientCredentials(client_id=cid,
                                client_secret=secret)
sp = spotipy.Spotify(client_credentials_manager=
client_credentials_manager)



def doQuery( conn ) :

    cur = conn.cursor()
    cur.execute(query)
    return cur.fetchall()

def existsOnStageQuery( check_id ) :
    connec = psycopg2.connect(host=hostname,
                              user=username,
                              password=password,
                              dbname='stages',
                              port=port )
    cursor = connec.cursor()
    cursor.execute(checkifexist+str(check_id))
    print (cursor.rowcount)
    if cursor.rowcount == 0:

      cursor.close()
      connec.close()
      return False
    else:
      cursor.close()
      connec.close()
      return True
```

```
artist_musicbrainz =[]

myConnection = psycopg2.connect( host=hostname,
                                 user=username,
                                 password=password,
                                 dbname=database, port=port )
artist_musicbrainz = doQuery( myConnection )
myConnection.close()
# print size list



#Verify if File in certain date already exists
filename=filepath
if os.path.exists(filename):
    print(filename+' '+'exists')
    print("Extraction is starts...")
else:
    # create empty lists where the results are going to be stored
    artist_name_group = []
    artist_id_msbz = []
    artist_id_member_position = []
    artist_id_type_group = []
    artist_id_country_origin =[]
    artist_country_origin =[]
    artist_id_city_origin = []
    artist_city_origin = []
    artist_gender_sex = []
    #spotify attributes
    artist_popularity = []
    artist_id_spotify = []
    artist_followers = []
    artist_type = []
    artist_genre = []
    artist_gender = []
```

```
for i in range(0,len(artist_musicbrainz)-1,1):
for name_group,
id_msbz,id_member_position,
id_type_group,id_country_origin,
country_origin,id_city_origin,
city_origin,gender_sex in artist_musicbrainz :
track_results =
sp.search(q=name_group,
type='artist',market='FR', limit=50)
#print (track_results)
for i, t in enumerate(track_results):
  if  not track_results[t]['items']:
      #print(track_results[t]['items'])
      print('artiste non trouv[U+FFFD]')
  elif (existsOnStageQuery(id_msbz)):
      print "$$$$$ --> I am already here. Rock n' Roll"
  else :
      try:
        print ('Into try')
        stageConnection = psycopg2.connect( host=hostname,
        user=username, password=password,
        dbname='stages',port=port )
        cur = stageConnection.cursor()

        print (track_results[t]['items'][0]['name']+",
        FROM ---->"+country_origin)

        cur.execute("""INSERT INTO
        public.stage_artist(id_msbz,
                  name_group,
                  id_member_position,
                  id_type_group,
                  id_country_origin,
                  country_origin,
                  id_city_origin,
                  city_origin,
                  gender_sex,
                  popularity,
                  id_spotify,
                  followers,
```

```
                    genre, date_full,
                    date_monthYear,
                    date_year)
                VALUES (%s,%s,%s,%s,%s,%s,%s,%s,
                %s,%s,%s,%s,%s,%s,%s,%s)""",
                (id_msbz,
                    track_results[t]['items'][0]['name'],
                    id_member_position,
                    id_type_group,
                    id_country_origin,
                    country_origin,
                    id_city_origin,
                    city_origin,
                    gender_sex,
                    track_results[t]['items'][0]['popularity'],
                    track_results[t]['items'][0]['id'],
                    track_results[t]['items'][0]['followers']['total'],
                    track_results[t]['items'][0]['genres'],
                    date,
                    date_monthYear,
                    date_year))

        cur.close()
        stageConnection.commit()
        stageConnection.close()

print ('finish -> connection')
except :
#    print e
  print ('##IGNORED##')
```

5.2.3 Script that extracts albums

```
#!/usr/bin/env python

import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
import pandas as pd
import psycopg2
import time
from sqlalchemy import create_engine
```

```
#musicBrainzCredentials
hostname = '10.195.25.10'
username = 'userbkp'
password = 'goma123'
database = 'stages'
port='54587'

#SpotifyCredentials
cid ="5f0b01a1813a41d1b360ed91e890a93f"
secret ="39877ef7e0a9467db85353f6090b0cbd"

#Declaration
today_date=(time.strftime("%d_%m_%Y"))
date=(time.strftime("%d_%m_%Y"))

client_credentials_manager = SpotifyClientCredentials(client_id=cid,
client_secret=secret)
sp = spotipy.Spotify(client_credentials_manager=
client_credentials_manager)

# Request MusicBrainz For artists using stage_artist
querySelect = """ SELECT id_msbz,id_spotify, name_group
FROM public.stage_artist ; """


def doReturnArtists( conn ) :

    cur = conn.cursor()
    cur.execute(querySelect)
    return cur.fetchall()

# Insert Albums in stage_Album
queryInsert= """INSERT INTO
public.stage_album (artist_name,
artist_id_msbz,
artist_id_spotify,album_id,
album_name,nb_tracks_album)
values(%s,%s,%s,%s,%s,%s)"""
```

```
def doInsertAlbums( artist_id_spotify ,artist_id_msbz ,
artist_name ,album_id ,album_name,nb_tracks_album) :
    myconnection =psycopg2.connect( host=hostname,
    user=username, password=password, dbname=database,
    port=port )
    cur = myconnection.cursor()
    cur.execute(queryInsert ,(artist_id_spotify ,
    artist_id_msbz ,artist_name ,album_id ,album_name,
    nb_tracks_album))
    cur.close()
    myconnection.commit()
    myConnection.close()

myConnection = psycopg2.connect( host=hostname,
user=username, password=password, dbname=database,port=port )
artist_musicbrainz = doReturnArtists( myConnection )
myConnection.close()
print ('here')

# create empty lists where the results are going to be stored
artist_namelist = []
artist_id_msbzlist= []
artist_id_spotifylist = []
album_idlist = []
nb_tracks_albumlist=[]
for i in range(0,len(artist_musicbrainz)-1,1):
    print ('*here')
    for id_msbz,id_spotify, name_group in artist_musicbrainz:
        print(id_spotify)
        albums_results=sp.artist_albums(id_spotify,
        album_type='album', limit=20, offset=0)
        for i, t in enumerate(albums_results['items']):
            print ('**here')
            print (t['id'])
            artist_namelist.append(name_group)
            artist_id_msbzlist.append(id_msbz)
            artist_id_spotifylist.append(id_spotify)
            album_idlist.append (t['id'])
            nb_tracks_albumlist.append(t['total_tracks'])
```

```
            print (t['total_tracks'])
            doInsertAlbums(name_group,id_msbz,id_spotify,
            t['id'],t['name'],t['total_tracks'])
            print('Inserted**')
```

5.2.4 Script that extracts tracks

```
#!/usr/bin/env python
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
import pandas as pd
import psycopg2
import time
from sqlalchemy import create_engine

#musicBrainzCredentials
hostname = '10.195.25.10'
username = 'userbkp'
password = 'goma123'
database = 'stages'
port='54587'

date_popularity=(time.strftime("%d/%m/%Y"))

cid ="a154f4842df643a99f9057fb741c86e0"
secret = "543080531fce4f30be3da2c36782ace1"

client_credentials_manager =
SpotifyClientCredentials(client_id=cid,
client_secret=secret)
sp = spotipy.Spotify(client_credentials_manager=
client_credentials_manager)

query = """ SELECT artist_name, artist_id_msbz,
artist_id_spotify, album_id, album_name FROM public.stage_album
; """

queryInsert=""" INSERT INTO
public.stage_track_features(
            artist_id_spotify, artist_id_msbz,
            artist_name, album_id, album_name,
```

```
            track_id, track_name, popularity,
            date_popularity, acousticness,
            danceability, duration_ms, energy,
            instrumentalness, key_f, liveness,
            loudness, mode_f, speechiness, tempo,
            time_signature, valence)
    VALUES (%s, %s, %s, %s, %s,
            %s, %s, %s, %s, %s,
            %s, %s, %s, %s, %s, %s, %s,
            %s, %s, %s,%s, %s);

"""

def doQuery( conn ) :

    cur = conn.cursor()
    cur.execute(query)
    return cur.fetchall()

def doInsertTracks(artist_id_spotify, artist_id_msbz,
artist_name, album_id, album_name,
            track_id, track_name, popularity,
            date_popularity, acousticness,
            danceability, duration_ms, energy,
            instrumentalness, key_f, liveness,
            loudness, mode_f, speechiness, tempo,
            time_signature, valence) :
    myconnection =psycopg2.connect( host=hostname,
    user=username, password=password, dbname=database,port=port )
    cur = myconnection.cursor()
    cur.execute(queryInsert,(artist_id_spotify,
    artist_id_msbz, artist_name, album_id, album_name,
            track_id, track_name, popularity,
            date_popularity, acousticness,
            danceability, duration_ms, energy,
            instrumentalness, key_f, liveness,
            loudness, mode_f, speechiness,
            tempo, time_signature, valence))
    cur.close()
    myconnection.commit()
```

```
    myConnection.close()

myConnection = psycopg2.connect( host=hostname,
user=username, password=password, dbname=database,port=port )
album_lists = doQuery( myConnection )
myConnection.close()
print ('here')


for i in range(0,len(album_lists)-1,1):
    print ('*here')
    print(i)
    for artist_name, artist_id_msbz, artist_id_spotify,
    album_id, album_name in album_lists :
        track_results=sp.album_tracks(album_id ,
        limit=20, offset=0)
        for i, t in enumerate(track_results['items']):
            print ('**here')
            track_name=t['name'] #track name
            print(t['name']) #track name
            print(artist_name) #artist name
            print(album_name)#album name
            track_id=t['id']#track id
            print(t['id']) #track id
            print('**here')
            track_features=sp.audio_features(t['id'])
            track_populartiy=sp.track(t['id'])
            popularity =track_populartiy['popularity']
            for r in track_features:
                try:
                    doInsertTracks(artist_id_spotify,
                    artist_id_msbz,artist_name, album_id,
                    album_name,track_id,track_name,
                    popularity,
                                date_popularity
                                ,r['acousticness'],
                                r['danceability'],
                                r['duration_ms'],
                                r['energy'],
                                r['instrumentalness'],
```

```
                                r['key'],r['liveness'],
                                r['loudness'],
                                r['mode'],
                                r['speechiness'],
                                r['tempo'],
                                r['time_signature'],
                                r['valence'])
            except:
                print("Ignore")
```

5.2.5 Transformation and Load process

As a result of our extraction now we have stages tables that can be easily used to load our data mart. To do this we used Talend studio [5] a French data integration tools that allow multiples forms of data manipulation. However, before take off to our transformation we need present our logical data mart model, through it we can create the data warehouse that I will receive all data.

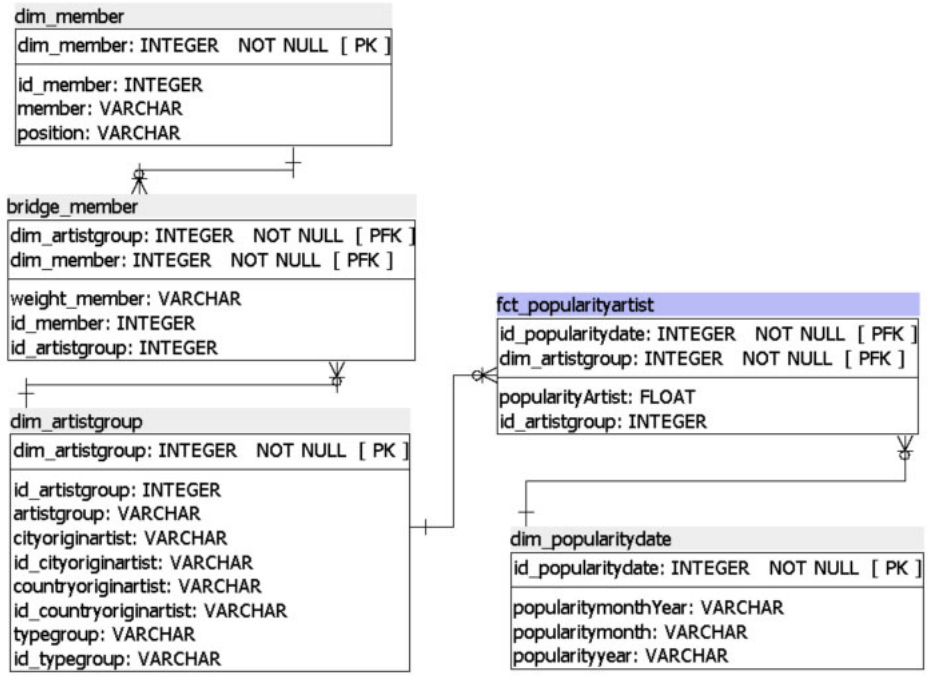

Figure 5.2.5-9: The logical model for artist popularity data mart.

We had chosen the data mart approach that will give us more flexibility and impetus during the development process. In fact, there is two reason for that, the first one is that a half of our group has not had positive attendance to get through the given tasks from phase two and three, consequently the another half is completely overcharged with the commitment of the project objectives and others master class projects. Finally, the second reason, it is the technical viability that not necessary require that the data marts will need exchanges relationships between them, because each data mart can answer the respective user needs directly.

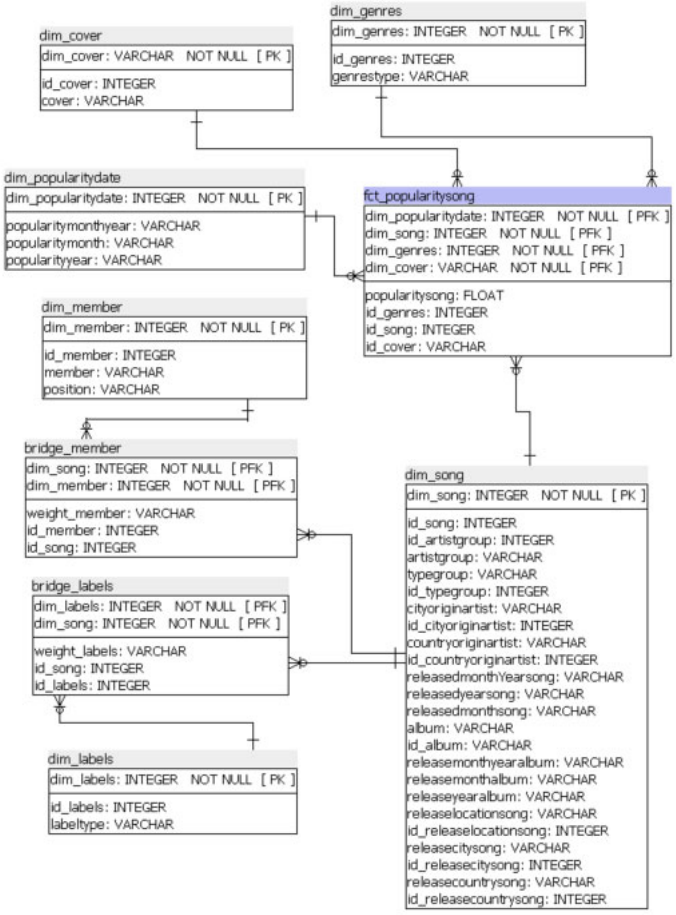

Figure 5.2.5-10: The logical model for song popularity data mart.

In addition to more details about the logical model, we had exported from our conception model the approach of multiple arcs, however it will not be used, because our front-end selected tool, Saiku, not have the appropriate support to this structure.

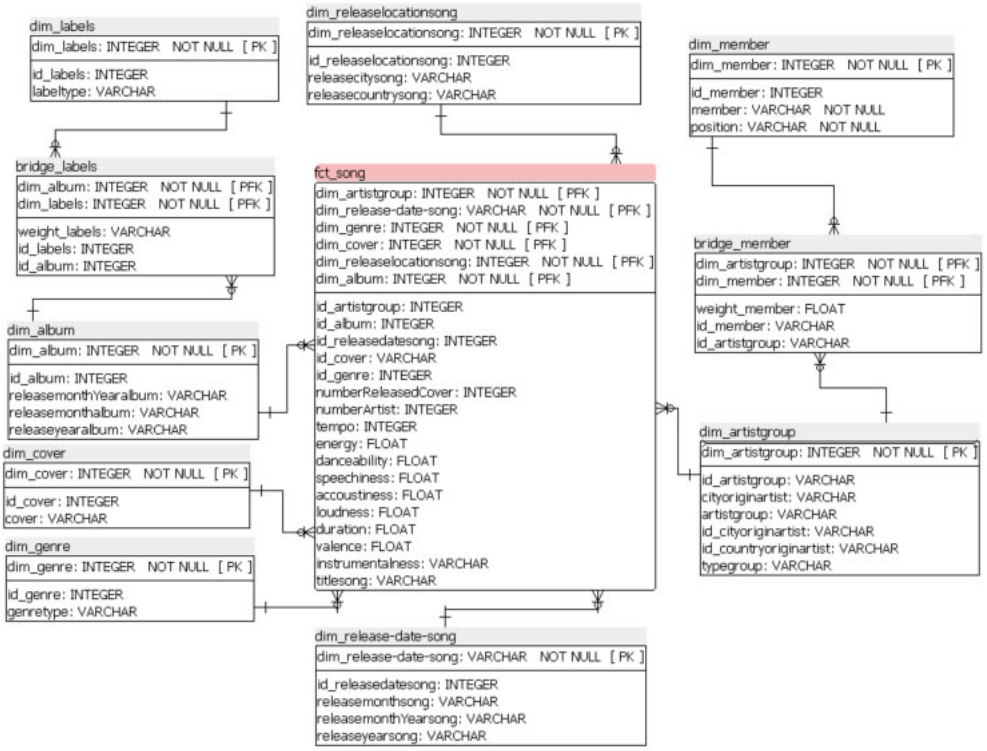

Figure 5.2.5-11: The logical model for song data mart.

For this reason, we have not sure if will be possible implement a parallel solution to it.

Since of the begin of our project, the group had hard difficulties to arrived in a common point about several features and characteristic that concerns our model, however above all our consideration, we not spectated a huge impact of a not assiduity of a part our group. That certainly had impaired the proper model improvement, what cause in several re-works over it. Because of that, during the loading charger a lot of errors appears, and they were fixed in flight time.

In addition what affect our work-flow of load, we choose to have a good practice near of loading our stage table, that was consisted in to create a table as close to our needs as possible. Then, with a simple "tMap" [5] component will are capable of make some adjustment, if necessary before any load.

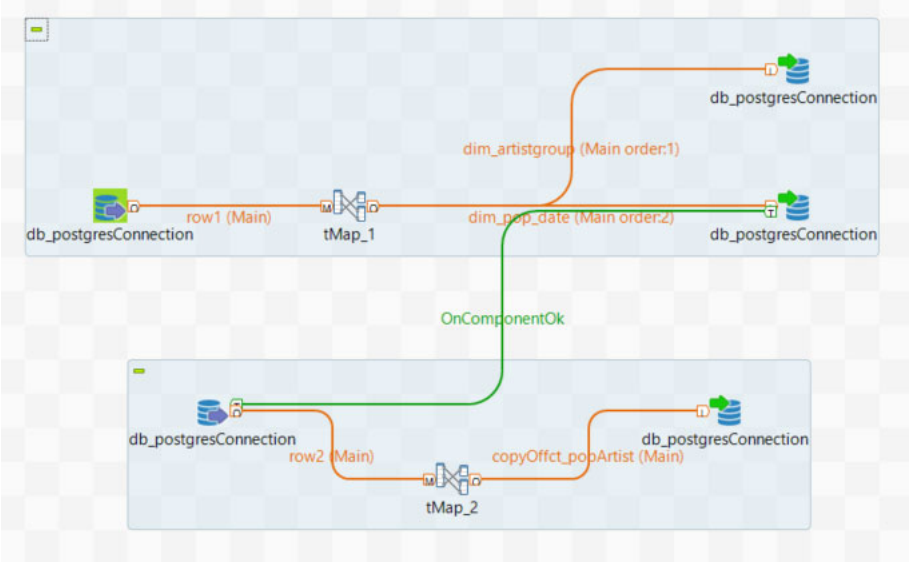

Figure 5.2.5-12: The work-flow for load Popularity Artist data mart.

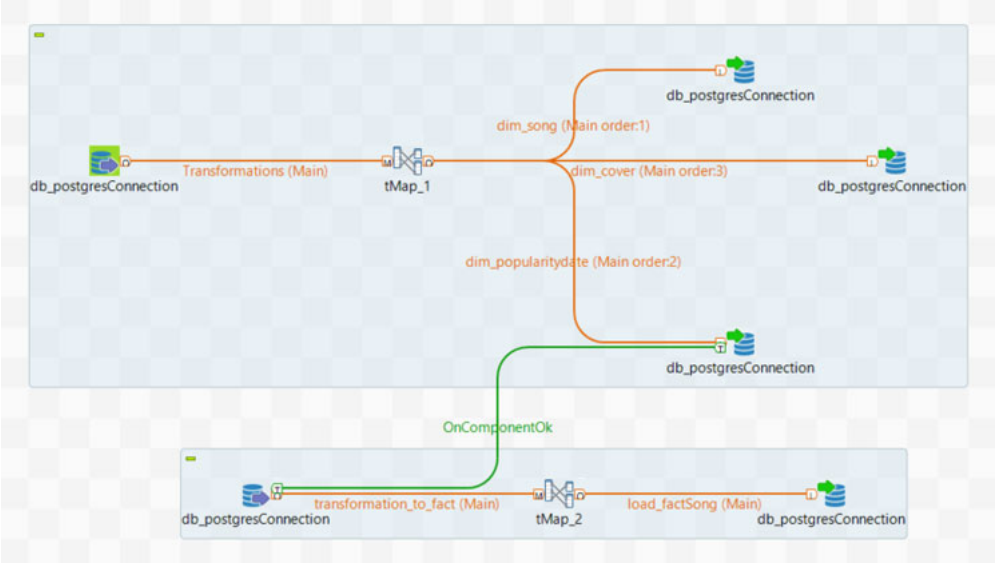

Figure 5.2.5-13: The work-flow for load Popularity Song data mart.

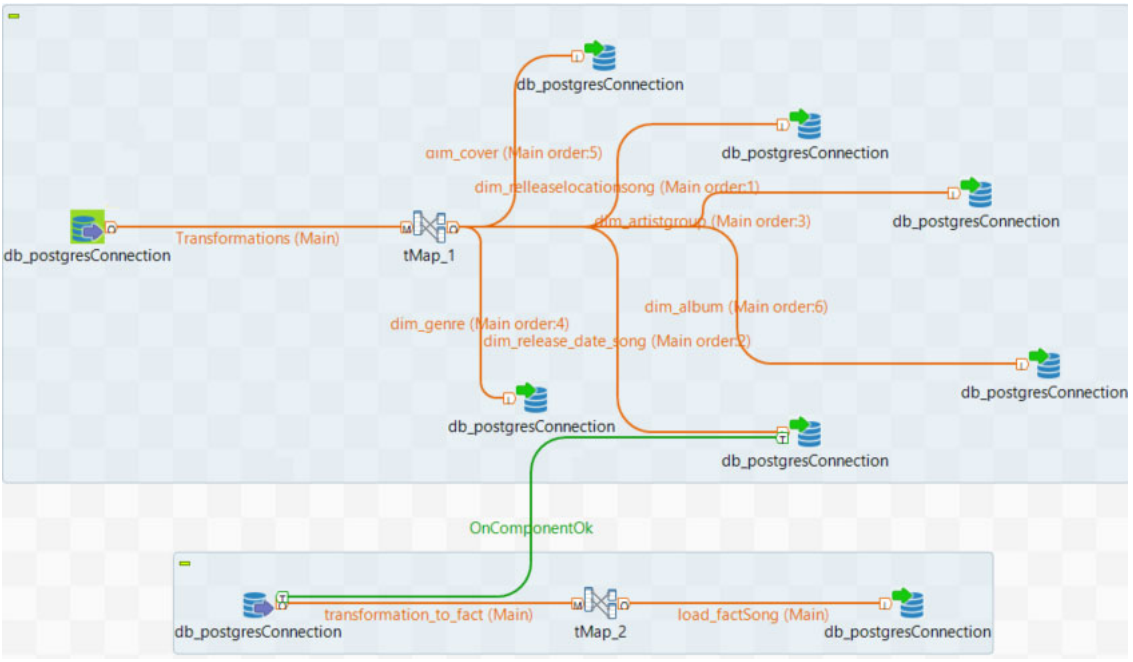

Figure 5.2.5-14: The work-flow for load song data mart.

References

[1] Coolors pattern. https://coolors.co/1a535c-20a39e-f7fff7-ff6b6b-ffe66d. accessed: 21.09.2018.

[2] Kanban. https://www.tutorialspoint.com/kanban/kanban_tutorial.pdf. accessed: 21.09.2018.

[3] Musicbrainz database and schema. https://musicbrainz.org/doc/MusicBrainz_Database/Schema. accessed: 21.09.2018.

[4] Spotify api. https://developer.spotify.com/documentation/web-api/. accessed: 21.09.2018.

[5] Talend studio. https://www.talend.com/products/talend-open-studio/. accessed: 19.11.2018.

[6] Trello. https://trello.com/. accessed: 21.09.2018.

[7] Christian DESTREMAU. Méthode scrum partie 2 : définition de la méthode. https://www.supinfo.com/articles/single/6054-methode-scrum-partie-2-definition-methode. accessed: 21.09.2018.

[8] Matteo Golfarelli. *Data Warehouse Design: Modern Principles and Methodologies*.

[9] Avantika Monnappa. Predicting the next big hit - big data and the music industry. https://www.simplilearn.com/big-data-science-in-music-industry-article. accessed: 21.09.2018.

[10] L.T. Moss and S. Atre. *Business Intelligence Roadmap: The Complete Project Lifecycle for Decision-Support Applications*. Addison-Wesley Information Technology Series. Pearson Education, 2003.

[11] Ken Schwaber and Jeff Sutherland. The scrum guide™ : The definitive guide to scrum: The rules of the game. https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf. accessed: 21.09.2018.