Introduction

Goal of this group project is to practically explore Knowledge and Data engineering methodologies facing the entire process that starts from row data and leads to a meaningful representation of the information in form of Knowledge Graphs, together with an application that makes use of SPARQL language to extract information from them.Furthermore, a particular attention is given to the process of designing an ontology, able to represent the type of information addressed by the original dataset and, exploited during the uplift process in order to reshape the original data but also to create new information that can be of interest in the stage of querying the application.

Background

A long standing and central challenge for musicians and artists and generally businesses online is to understand and model what users like or dislike about their art, and more particularly dive deep into the metadata of their production and extract interesting insights.We got inspired by the same problem-statement in the domain of Music. Our approach was to search for datasets online, that will help us create a knowledge graph for the Musical Domain and be able to answer questions and get insights.

We found MusicOSet, an open and enhanced musical dataset which offered rich features for us to do data engineering upon.

A brief description of the schema of the MusicOSet is provided below:

Artist Schema –

Followers Table: Information about the followers of the artist.
Popularity Table: The popularity of the artist. Values 0-100, 100 being most popular.
Artist Type: The type of the artists: one of "singer", "band", "duo" or "rapper".
Main Genre: The main genre in which the artist is associated with.
Genres: A list of the genres the artist is associated with.

Albums Schema –

Artists Table: The artists of the album.
Popularity Table: The popularity of the album. Values 0-100, 100 being most popular.
Tracks Table: The total number of tracks.
Album Type Table: The type of the album: one of "album", "single" or "compilation".

Songs Schema –

Artist Table: The artists who performed the track.
Popularity Table: The popularity of the track.
Explicit Table: Whether or not the track has explicit lyrics.
Song Type Table: The type of the song: "solo songs" or “collaborative songs”

Acoustic Features Table –

Pitch, Acousticness, Danceability, Energy, Intrumentalness, Liveness, Loudness etc are given as different properties of a song inside the table.

(To dig deeper into each acoustic feature and how its collected an introduction is provided here: https://marianaossilva.github.io/DSW2019/)

The total size of the dataset is 302MB, and contains 20,405 unique songs, 11,518 unique Artists and 26,225 unique albums spanning over 20 years!

We define our competency questions next, and define an ontology “music” which maps the uplifted dataset in a unique and meaningful way. The uplifted data is a subset of the total data.

Competency Questions

We totally have 10 competency questions which vastly explores the uplifted datasets and the ontology which has been created. All the answers for the question are derived by using various properties and values from the below mentioned three datasets:

- **Popularity Dataset**
- **Metadata Dataset**
- **Acoustic Features Dataset**

1. Which popular songs in Pop genre refers to "love" in their lyrics?

   This returns all the popular songs in Pop genre which has the word "**love**" in their lyrics. Below values are used from datasets in order to compute the desired output

Popularity Dataset	Metadata Dataset	Acoustic Features
- Song Popularity	**- Album Genre**	**- Lyrics**

2. What is the average duration of songs of type solo from the most popular artist of 2017?

   This returns the average duration (**in milliseconds**) of solo songs from the most popular artist of year 2017. Below values are used from datasets in order to compute the desired output

Popularity Dataset	Metadata Dataset	Acoustic Features
- Artist Popularity	**- Song Type**	**- Duration**

3. Which are the popular rap albums from an artist that have the higher average of "Speechiness" among its songs?

   This returns all the popular rap albums from various artists which have highest average value of speechiness (**presence of spoken words valued from 0 to 1**). Below values are used from datasets in order to compute the desired output

Popularity Dataset	Metadata Dataset	Acoustic Features
- Album Popularity	**- Artist Type**	**- Speechiness**

4. Which band has the greatest number of popular albums in the last 30 years?

   This returns the band details which had the greatest number of popular albums in last 30 years. Below values are used from datasets in order to compute the desired output

Popularity Dataset	Metadata Dataset
- Album Popularity	**- Artist Name** **- Artist Type**

5. Which is their most popular live song (liveness)?

   This is a continuation question from Question 4 which tells us the most popular live song (**liveness - Detects the presence of an audience in the recording**) from the band which has the greatest number of popular albums from last 30 years. Below values are used from datasets in order to compute the desired output

Popularity Dataset	Metadata Dataset	Acoustic Features
- Album Popularity **- Song Popularity**	**- Artist Type** **- Song Name**	**- Liveness**

6. Who is the artist of the album which has the highest valued happiness in 2015 (valance)?

   Returns the details of an artist and album which has the highest happiness factor (**Valence - A measure from 0.0 to 1.0 describing the musical positiveness conveyed**) in 2015. Below values are used from datasets in order to compute the desired output

Metadata Dataset	Acoustic Features
- Artist Name **- Album Name** **- Release Date**	**- Valence**

7. How many songs released in 2018 that are danceable became popular?

   Returns the details of the popular songs in 2018 which are danceable (**describes how suitable a track is for dancing based on a combination of musical elements ranging from 0 to 1**). Below values are used from datasets in order to compute the desired output

Popularity Dataset	Metadata Dataset	Acoustic Features
- Song Popularity	**- Song Name** **- Release Date**	**- Danceability**

8. What is the average value of tempo of the 20 most popular electro genre songs?

   Return the average value of tempo (**The overall estimated tempo of a track in beats per minute - BPM**) of 20 most popular electro genre songs. Below values are used from datasets in order to compute the desired output

Popularity Dataset	Metadata Dataset	Acoustic Features
- Song Popularity	**- Song Name** **- Genre**	**- Tempo**

9. Which are the 5 most popular albums with high value of energy?

   Returns details of top five popular albums which as high value of energy (**measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity**). Below values are used from datasets in order to compute the desired output

Popularity Dataset	Metadata Dataset	Acoustic Features
- Album Popularity	**- Album Name**	**- Energy**

10. Who is the most popular artist that produced the higher number of sad songs (valence)?

    Returns the details of the most popular artist having greater number of sad songs (**Valance - A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track**) Below values are used from datasets in order to compute the desired output

Popularity Dataset	Metadata Dataset	Acoustic Features
- Artist Popularity	**- Artist Name**	**- Valence**

Assumptions Made

The starting point in developing our applications are the original datasets, such datasets have been created by other parties probably with other scopes. For these reasons we can find few types of data that are not useful to us:

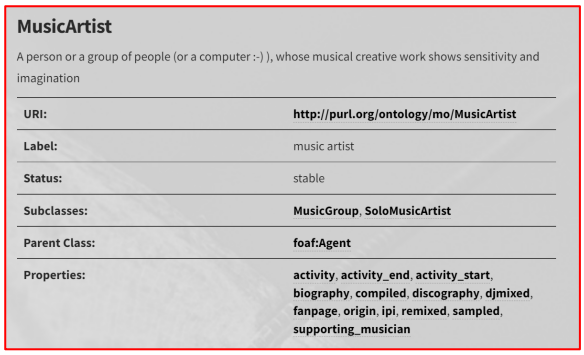
- IDs - information is stored in the original tables with an Identifier that addresses the rows, this type of information goes in conflict with one of the key aspects we must address: *Link different dataset in a meaningful and original way*.
  This is the reason why the design of the ontology has been made with the goal to discover alternatives paths to connect original tables together and none of the IDs has been used to simplify the work.
  Also, such type of information has been deleted during the uplift stage.
- Uninteresting tables - Ontology design is a stage that presents a high grade of liberty, the output ontology depends on many factors such as scope of the application and personal interpretation.
  In our case, has been decided to not force the hand trying to put all the original information in the process. Is the case of few tables that, indeed, are not used, because deemed unnecessaries: "*album_chart.csv", "artist_chart.csv", "song_chart.csv*".

None of the tables has been pre modified to simplify the work. The only exceptions are represented by the fact that we had to rewrite in upper case the name of the columns in the tables and to change the separator character from ";" to ":". These exceptions have become necessary to avoid compilation problems with the r2rml engine, not able to process the older values.

Ontology References

In this project, we did not re-use ontology from others. We are aiming to build a music ontology in this project, at the very beginning of the project, we thought we could use 'People' from FOAF, since artist would be a subclass of People, 'Year' from SIOC may also be used, since we have release_data property of a album/song. Moreover, there are existing music ontologies that may be referenced at the 1st glance.

http://musicontology.com/specification/#term-MusicArtist



MusicArtist

A person or a group of people (or a computer :-)), whose musical creative work shows sensitivity and imagination

URI:	http://purl.org/ontology/mo/MusicArtist
Label:	music artist
Status:	stable
Subclasses:	MusicGroup, SoloMusicArtist
Parent Class:	foaf:Agent
Properties:	activity, activity_end, activity_start, biography, compiled, discography, djmixed, fanpage, origin, ipi, remixed, sampled, supporting_musician

However, as deeper we dive into the project and the oncology, we found there're no need that we should re-use the vocabularies from others. First, we want to build the classes and relationships that are reasonable and original, references may suppress the creative ideas we would have for our own ontology. Second, we believe that xsd and rdf domain would be basic and general enough to build a new ontology. In the end we have references below:

```
@prefix : <http://www.semanticweb.org/ontologies/music.owl#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@base <http://www.semanticweb.org/ontologies/music.owl> .
```

The reused references are mainly used to define the data properties.

```
rdfs:range rdfs:Literal .
rdfs:range xsd:integer.
rdfs:range xsd:decimal .
rdfs:range xsd:boolean .
```

Data Mapping - Upliftment Process

Both ontology design and upliftment process have been developed with the aim to recycles fields from the tables to create connections between classes in a meaningful way without the use of IDs included in the original datasets. If ontology design focused on the definition of such connections, the uplift process provided a practical way to map the data into classes using those connections.Furthermore, upliftment process is the component responsible for the generation of new information derived from the original tables.

Uplift map script generates individuals for all the 8 classes defined in the ontology:

AcousticFeatures, Song, Album, Artist, Release, SongPopularity, ArtistPopularity, AlbumPopularity

And takes as input all the tables in the original datasets (apart from those that have been excluded from the project because considered uninteresting - we remind to Assumptions made):

- **Metadata Dataset**: ALBUMS, ARTISTS, RELEASES, SONGS, TRACKS
- **Popularity Dataset**: ALBUM_POP, ARTIST_POP, SONG_POOP
- **SongFeatures Dataset**: ACOUSTIC_FEATURES, LYRICS

To execute the goal have been defined 4 r2rml views (discussed below) to select data from the tables in a way useful to realize the mapping into the ontology defined classes:

- <#JoinTableReleasesAlbumsTracksSongsFeaturesLyrics>
- <#JoinTableArtistsReleasesAlbums>
- <#JoinPopularitySongsFeatures>
- <#JoinPopularityArtists>

<#JoinTableReleasesAlbumsTracksSongsFeaturesLyrics>

This is the most complex view but also the one that handles the greatest number of tables. This view selects all the fields from the tables involved and generates a new property that provides new information: HYSTORICAL_ORDER.

HYSTORICAL_ORDER is a 4 digit code that indicates the order by which the albums have been released (e.g. "0001" is the first album ever released in the dataset). It is part of the Album class in the ontology and it is concatenated together with the Track_number of the Song individuals to create a Track_reference. This is, indeed, the way the Album and the Song class are related.

Classes generated from this view:

- <#AcousticFeaturesMap>
- <#SongMap>
- <#AlbumMap>
- <#AlbumPopMap>

Tables involved:

- SONGS
- ACOUSTIC_FEATURES
- ALBUMS
- RELEASES
- LYRICS
- TRACKS
- ALBUM_POP

```
<#JoinTableReleasesAlbumsTracksSongsFeaturesLyrics> rr:sqlQuery """
SELECT S.SONG_ID,
     S.SONG_NAME,
     S.POPULARITY AS S_POPULARITY,
     S.EXPLICIT,
     S.SONG_TYPE,
     AF.DURATION_MS,
     AF.TEMPO,
     AF.TIME_SIGNATURE,
     AF.ACOUSTICNESS,
     AF.DANCEABILITY,
     AF.ENERGY,
     AF.INSTRUMENTALNESS,
     AF.LIVENESS,
     AF.LOUDNESS,
     AF.SPEECHINESS,
     AF.VALENCE,
     T.RELEASE_DATE AS T_RELEASE_DATE,
     T.RELEASE_DATE_PRECISION AS T_RELEASE_DATE_PRECISION,
     A.ALBUM_ID,
     A.IMAGE_URL,
     A.NAME,
     A.POPULARITY AS A_POPULARITY,
     A.TOTAL_TRACKS,
     A.ALBUM_TYPE,
     R.RELEASE_DATE AS R_RELEASE_DATE,
     R.RELEASE_DATE_PRECISION AS R_RELEASE_DATE_PRECISION,
     RIGHT(CONCAT('0000000000', (DENSE_RANK() OVER (ORDER BY  R.RELEASE_DATE,A.ALBUM_ID))), 4) AS HYSTORICAL_ORDER,
     RIGHT(CONCAT('0000000000', T.TRACK_NUMBER), 2) AS TRACK_REFERENCE,
     L.LYRICS,
     POP.YEAR_END_SCORE,
     POP.IS_POP,
     POP.YEAR
FROM RELEASES R INNER JOIN ALBUMS A
                  ON R.ALBUM_ID = A.ALBUM_ID
                INNER JOIN ALBUM_POP POP
                  ON A.ALBUM_ID = POP.ALBUM_ID
                INNER JOIN TRACKS T
                  ON A.ALBUM_ID = T.ALBUM_ID
                INNER JOIN SONGS S
                  ON T.SONG_ID = S.SONG_ID
                INNER JOIN ACOUSTIC_FEATURES AF
                  ON T.SONG_ID = AF.SONG_ID
                INNER JOIN LYRICS L
                  ON T.SONG_ID = L.SONG_ID
                ORDER BY R.RELEASE_DATE, A.ALBUM_ID ASC;
""".
```

The necessity to merge such a high number of tables comes from a series of waterfall constraints:

- The generated HYSTORICAL_ORDER field is calculated from ALBUMS that are ordered by RELEASES (2 tables).
- HYSTORICAL_ORDER is also part of Song and AlbumPopularity classes (2 more tables).
- AcousticFeatures class is generated having 2 fields in common with Song class (1 more table).
- The table LYRICS is translated to a property of AcousticFeatures class instead of becoming a separated Class(1 more table).

We had to respect all of this constraints in order to generate consistent information that is uplifted to individuals of all these classes.A practical example: if we generated HYSTORICAL_ORDER multiple times with smaller joins between classes we would end up with repeated values with a confused meaning since it is calculated from a DENSE_RANK(). Consequently, we needed it to be generated only one time and upon all the range of RELEASES dates.

<#JoinTableArtistsReleasesAlbums>

This view is used to generate Release individuals. Such class is linked to Artist by IMAGE_URL (Artist picture) and to Album by IMAGE_URL (Cover image).

Classes generated from this view:

- <#ReleaseMap>

Tables involved:

- ALBUMS
- RELEASES
- ARTISTS

```
<#JoinTableArtistsReleasesAlbums> rr:sqlQuery """
SELECT
      ALB.ALBUM_ID,
      ALB.IMAGE_URL AS COVER,
      REL.RELEASE_DATE,
      REL.RELEASE_DATE_PRECISION,
      ART.ARTIST_ID,
      ART.IMAGE_URL AS IMG
FROM ALBUMS ALB INNER JOIN RELEASES REL
                    ON ALB.ALBUM_ID = REL.ALBUM_ID
                INNER JOIN ARTISTS ART
                    ON REL.ARTIST_ID = ART.ARTIST_ID;
""".
```

<#JoinPopularitySongsFeatures>

This view is used to generate SongPopularity individuals. Such class is linked to Song by popularity, duration and tempo properties.

Classes generated from this view:

- <#SongPopMap>

Tables involved:

- ACOUSTIC_FEATURES
- SONGS
- SONG_POP

```
<#JoinPopularitySongsFeatures> rr:sqlQuery """
SELECT
    SNG.SONG_ID,
    POP.YEAR_END_SCORE,
    POP.IS_POP,
    POP.YEAR,
    SNG.POPULARITY,
    AF.DURATION_MS,
    AF.TEMPO
FROM SONGS SNG INNER JOIN ACOUSTIC_FEATURES AF
                    ON SNG.SONG_ID = AF.SONG_ID
               INNER JOIN SONG_POP POP
                    ON SNG.SONG_ID = POP.SONG_ID;
""".
```

<#JoinPopularityArtists>

This view is used to generate ArtistPopularity individuals. Such class is linked to Artist by popularity and main_genre properties.

Classes generated from this view:

- <#ArtistPopMap>

Tables involved:

- ACOUSTIC_FEATURES
- SONGS
- SONG_POP

```
<#JoinPopularityArtists> rr:sqlQuery """
SELECT
      POP.ARTIST_ID,
      POP.YEAR_END_SCORE,
      POP.IS_POP,
      POP.YEAR,
      ART.POPULARITY,
      ART.MAIN_GENRE
FROM ARTIST_POP POP INNER JOIN ARTISTS ART
                        ON POP.ARTIST_ID = ART.ARTIST_ID;
""".
```

Class relationship

The way the classes have been uplifted to reflect the relations among them defined in the ontology are shownbelow:

```
rr:predicateObjectMap [
    rr:predicate msc:areFeaturesOf;
    rr:objectMap [
        rr:parentTriplesMap <#SongMap>;
        rr:joinCondition [
            rr:child "DURATION_MS";
            rr:parent "DURATION_MS";
        ];
        rr:joinCondition [
            rr:child "TEMPO";
            rr:parent "TEMPO";
        ];
    ];
];
```

Class Relationship: AcousticFeature to Song using DURATION_MS & TEMPO (:areFeaturesOf)

```
    rr:predicateObjectMap [
        rr:predicate msc:hasFeatures;
        rr:objectMap [
            rr:parentTriplesMap <#AcousticFeaturesMap>;
            rr:joinCondition [
                rr:child "DURATION_MS";
                rr:parent "DURATION_MS";
            ];
            rr:joinCondition [
                rr:child "TEMPO";
                rr:parent "TEMPO";
            ];
        ];
    ];

    rr:predicateObjectMap [
        rr:predicate msc:isPartOf;
        rr:objectMap [
            rr:parentTriplesMap <#AlbumMap>;
            rr:joinCondition [
                rr:child "HYSTORICAL_ORDER";
                rr:parent "HYSTORICAL_ORDER";
            ];
        ];
    ];

    rr:predicateObjectMap [
        rr:predicate msc:hasSongPopularityProperties;
        rr:objectMap [
            rr:parentTriplesMap <#SongPopMap>;
            rr:joinCondition [
                rr:child "S_POPULARITY";
                rr:parent "POPULARITY";
            ];
            rr:joinCondition [
                rr:child "DURATION_MS";
                rr:parent "DURATION_MS";
            ];
            rr:joinCondition [
                rr:child "TEMPO";
                rr:parent "TEMPO";
            ];
        ];
    ];
```

Class Relationship: Song to AcousticFeatures using DURATION_MS & TEMPO (:hasFeatures)
Class Relationship: Song to Album using HYSTORICAL_ORDER (:isPartOf)
Class Relationship: Song to Popularity using POPULARITY & DURATION_MS & TEMPO (:hasSongPopularityProperties)

```
rr:predicateObjectMap [
    rr:predicate msc:isComposedBy;
    rr:objectMap [
        rr:parentTriplesMap <#SongMap>;
        rr:joinCondition [
            rr:child "HYSTORICAL_ORDER";
            rr:parent "HYSTORICAL_ORDER";
        ];
    ];
];

rr:predicateObjectMap [
    rr:predicate msc:containsAcousticFeatures;
    rr:objectMap [
        rr:parentTriplesMap <#AcousticFeaturesMap>;
        rr:joinCondition [
            rr:child "HYSTORICAL_ORDER";
            rr:parent "HYSTORICAL_ORDER";
        ];
    ];
];

rr:predicateObjectMap [
    rr:predicate msc:albumRelease;
    rr:objectMap [
        rr:parentTriplesMap <#ReleaseMap>;
        rr:joinCondition [
            rr:child "IMAGE_URL";
            rr:parent "COVER";
        ];
    ];
];

rr:predicateObjectMap [
    rr:predicate msc:hasAlbumPopularityProperties;
    rr:objectMap [
        rr:parentTriplesMap <#AlbumPopMap>;
        rr:joinCondition [
            rr:child "A_POPULARITY";
            rr:parent "A_POPULARITY";
        ];
        rr:joinCondition [
            rr:child "HYSTORICAL_ORDER";
            rr:parent "HYSTORICAL_ORDER";
        ];
    ];
];
```

Class Relationship: Album to Song using HYSTORICAL_ORDER (:isComposedBy)
Class Relationship: Album to Acoustic Features using HYSTORICAL_ORDER (:containsAcousticFeatures)(Transitive)
Class Relationship: Album to Release using COVER (:albumRelease)
Class Relationship: Album to Album Popularity using POPULARITY and HYSTORICAL ORDER (:hasAlbumPopularityProperties)

```
rr:predicateObjectMap [
    rr:predicate msc:artistRelease;
    rr:objectMap [
        rr:parentTriplesMap <#ReleaseMap>;
        rr:joinCondition [
            rr:child "IMAGE_URL";
            rr:parent "IMG";
        ];
    ];
];

rr:predicateObjectMap [
    rr:predicate msc:hasArtistPopularityProperties;
    rr:objectMap [
        rr:parentTriplesMap <#ArtistPopMap>;
        rr:joinCondition [
            rr:child "POPULARITY";
            rr:parent "POPULARITY";
        ];
        rr:joinCondition [
            rr:child "MAIN_GENRE";
            rr:parent "MAIN_GENRE";
        ];
    ];
];
```

Class Relationship: Artist to Release using IMG (:artistRelease)
Class Relationship: Artist and Artist Popularity using POPULARITY and MAIN GENRE (:hasArtistPopularityProperties)

```
rr:predicateObjectMap [
    rr:predicate msc:albumRelease;
    rr:objectMap [
        rr:parentTriplesMap <#AlbumMap>;
        rr:joinCondition [
            rr:child "COVER";
            rr:parent "IMAGE_URL";
        ];
    ];
];

rr:predicateObjectMap [
    rr:predicate msc:artistRelease;
    rr:objectMap [
        rr:parentTriplesMap <#ArtistMap>;
        rr:joinCondition [
            rr:child "IMG";
            rr:parent "IMAGE_URL";
        ];
    ];
];
```

Class Relationship: Release to Album using COVER (:albumRelease)
Class Relationship: Release to Artist using IMG (:artistRelease)

```
rr:predicateObjectMap [
    rr:predicate msc:hasSong;
    rr:objectMap [
        rr:parentTriplesMap <#SongMap>;
        rr:joinCondition [
            rr:child "POPULARITY";
            rr:parent "A_POPULARITY";
        ];
        rr:joinCondition [
            rr:child "DURATION_MS";
            rr:parent "DURATION_MS";
        ];
        rr:joinCondition [
            rr:child "TEMPO";
            rr:parent "TEMPO";
        ];
    ];
];
```

Class Relationship: SongPopularity to Song using POPULARITY, DURATION_MS and TEMPO (:hasSong)

```
rr:predicateObjectMap [
    rr:predicate msc:hasAlbum;
    rr:objectMap [
        rr:parentTriplesMap <#AlbumMap>;
        rr:joinCondition [
            rr:child "A_POPULARITY";
            rr:parent "A_POPULARITY";
        ];
        rr:joinCondition [
            rr:child "HYSTORICAL_ORDER";
            rr:parent "HYSTORICAL_ORDER";
        ];
    ];
];
```

Class Relationship: AlbumPopularity to Album using POPULARITY and HISTORTICAL ORDER (:hasAlbum)

.

```
rr:predicateObjectMap [
    rr:predicate msc:hasArtist;
    rr:objectMap [
        rr:parentTriplesMap <#ArtistMap>;
        rr:joinCondition [
            rr:child "POPULARITY";
            rr:parent "POPULARITY";
        ];
        rr:joinCondition [
            rr:child "MAIN_GENRE";
            rr:parent "MAIN_GENRE";
        ];
    ];
];
```

Class Relationship: ArtistPopularity to Artist using POPULARITY and MAIN_GENRE (:hasArtist)

Inverse, symmetric, transitive properties

In the process of designing the ontology has been decided to create double links between classes in order to make easier to extract information through the queries but also because it had logical sense, for this reason the majority of the classes connected are double linked with Inverse properties. Also, we decided to include two symmetrical properties: between Artist and Release and Album and Release.

This is because logically when an artist releases an album means also that the release representing this event is made by that artist.Furthermore, we knew that for many queries would have been necessary to extract information from the AcousticFeatures class of those songs part of an Album or realised by some Artist, for this reason we created a transitive property: containsAcousticFeatures (from Album to AcousticFeatures). This property is transitive with isComposedBy (from Album to Song) and hasFeatures (From Song to AcousticFeatures).

Query Interface

The query interface for the Application has been developed using Java Programming Language. We have used following two Java Frameworks which handles the back-end and the front-end capabilities of the application:

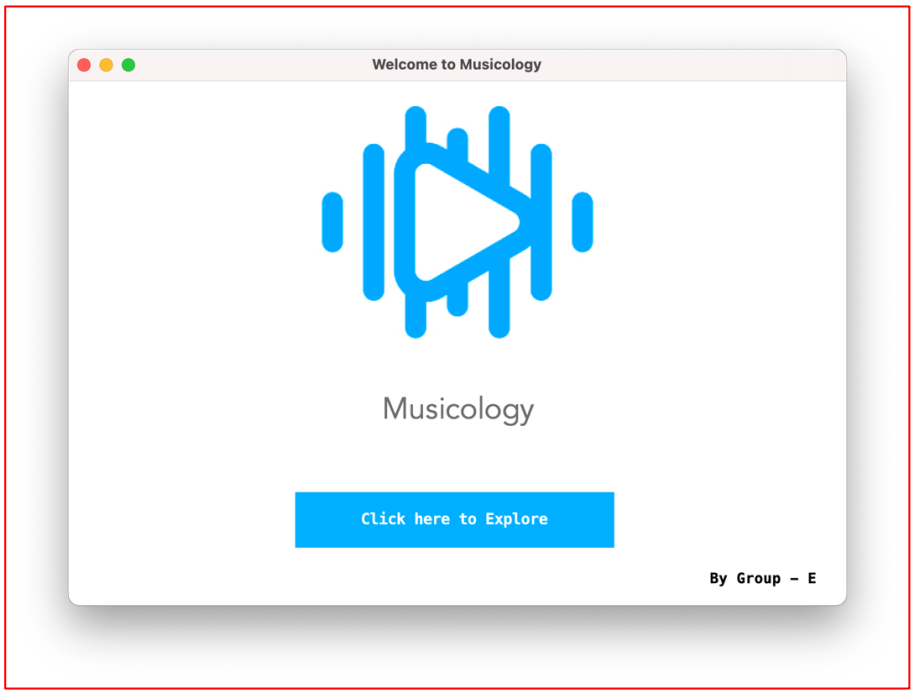
1. **Apache Jena** – It is an open-source java framework which is extensively used to work-on and build Semantic Web and Linked Data Applications. It helps in uplifting the raw data (CSV) and provides various tools to create a sustainable ontology as required. In our project, we are using Apache Jena mainly to parse and process the final .ttl (Turtle) file which includes the Uplifted data with our defined ontology. Apache Jena then provides us a platform in Java to extract the required information from the uplifted data by writing SPARQL queries. We can take advantage of wide range of features and libraries in Java to manipulate and present the data which is extracted through the queries.

   Below shown code is executed in order to process the .ttl file and queries to get the results for further manipulation.

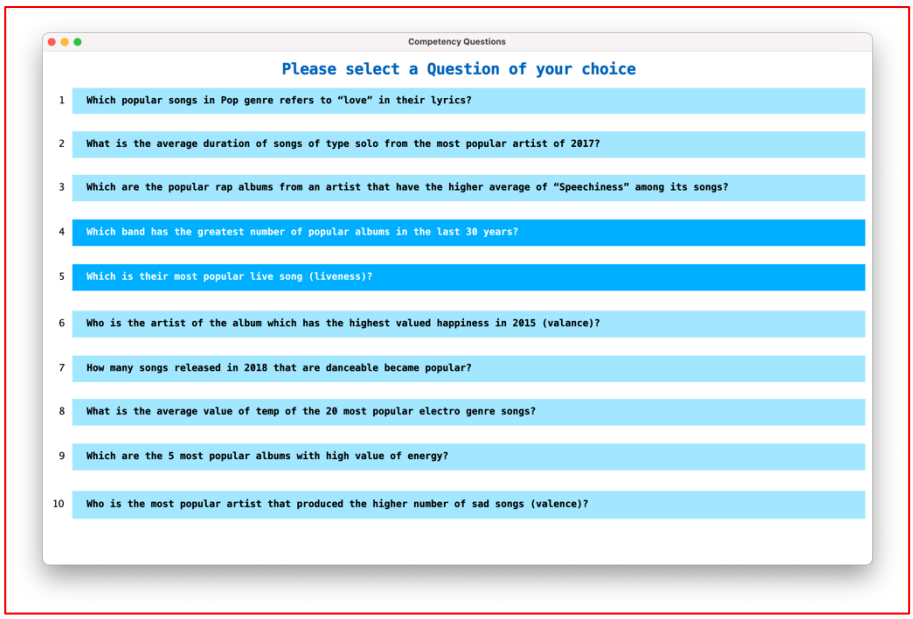
   ```
   String finalTtlFile =
   QueryProcessor.class.getResource(CommonConstants.OUTPUT_TTL_FILE).getFile();
   Model model = ModelFactory.createDefaultModel();
   model.read(finalTtlFile);
   Query query = QueryFactory.create(queryString);
   QueryExecution queryExecution = QueryExecutionFactory.create(query, model);
   ResultSet results = queryExecution.execSelect();
   ```

2. **Java Swing** – It is a lightweight toolkit in Java which is extensively used to create a GUI (Graphical User Interface) for Java applications. It is one among the most commonly used tools in Java since its known for its less memory consumption, stability and faster response. An interactive application is

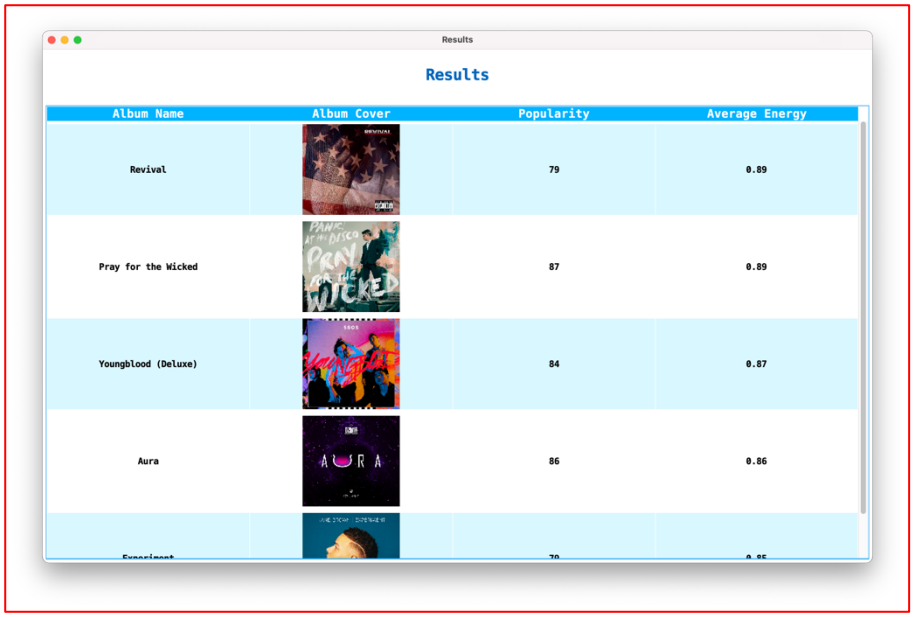
created in which users will be given an option to choose a Competency question which then displays a table which includes appropriate results (These results are obtained by Apache Jena Framework by processing the corresponding SPARQL Queries). In total, the Application user interface consists of 3 windows –



- **Welcome Window** - Which presents the user with a startup page for our application "Musicology". User can move ahead by click "Click here to Explore" button to get the list of Competency Questions.



- **Competency Questions Window** – This window displays the list of all the Competency questions which we tend to answer with our uplifted Data. The user can click on any question in order to see the results.



- **Results Window –** This window shows the results of the selected question in form of a table. We have also implemented to show album and artist images which makes it easier to understand for end-users.

Query Descriptions

Q1. Which popular songs in Pop genre refers to "love" in their lyrics
Lyrics column is an object of Acoustic Feature connected through predicate :lryics. We filter from song popularity with literal "True" and map all these to song. Inside song we query through :hasFeatures and find the lyrics that CONTAINS "love".

Q2. What is the average duration of songs of type solo from the most popular artist of 2017?
The inner query finds the most popular artist for a particular year using a subject ArtistPopularity and mapping it to a subject Artist. The outer query finds all the songs connected to this artist, as each song has an album which has a release done by the artist. We take avg of the song duration and div it by 60,000 as it is in ms to convert it to minutes.

Q3. Which are the popular rap albums from an artist that have the higher average of "Speechiness" among its songs?
Using the transitive property :containsAcousticFeatures, we query the top 100 Albums on the basis of their speechiness.

Q4. Which band has the greatest number of popular albums in the last 30 years?
We count the number of albums that have an artist with artist_type as "band" and filter the album popularity to show results for values above 80. This result is grouped by the Artist subject, and counts the number of albums for that artist satisfying the above criteria.

Q5. Which is their most popular live song (liveness)?
This query builds on top of the previous query in Q4. It selects the first result from the above query and from the artist, maps all their songs, and the song's popularity and liveness and orders the result by decreasing liveness.

Q6. Who is the artist of the album which has the highest valued happiness in 2015 (valance)?
We map all artists to their releases and all releases to their respective albums, this gives us the album name w.r.t the artist name, now for all songs that have released in the year 2015, we get the valence of these songs and group them by their album name. The highest average valence is displayed.

Q7. How many songs released in 2018 that are danceable became popular?

All songs released in 2018, are filtered based on their dance values and mapped to song popularity having the Boolean (in this case Literal) value "True".

Q8. What is the average value of tempo of the 20 most popular electro genre songs?
For every artist having a genre "Electro", we query the mapping release and album and finally song connected to that. The result is finally ordered by decreasing value of the popularity metric of the song.

Q9. Which are the 5 most popular albums with high value of energy?
Using :containsAcousticFeatures transitive property, we find the albums, that have songs with energy greater than 0.5 . All such albums are grouped first based on album name and then on their average energies.

Q10. Who is the most popular artist that produced the higher number of sad songs (valence)?
We query the songs that have valence < 0.5, as that is a property that ranges from 0-1 indicating happiness of cheerfulness in a song. We map this to the songs, and count the number of such songs grouped by artists.

Challenges Faced

1) During the process of uplifting, we faced major performance issues given that our dataset is huge. After tying and executing several multiple processes and changes of mapping techniques, we decided to reduce the size of our dataset by slicing and considering the major parts of it. This increased the performance and enabled us to uplift the data in faster rate.
2) Uploading OWL and Uplifted TTL as TTLs into GraphDB is not mentioned anywhere or intuitive and there's a gap in understanding how to merge the two.
3) Importing data in Protégé is difficult especially with multiple datasets, and defining individuals to explore properties is not helpful as it doesn't bind to the actual dataset later on.
4) To implement the front-end for the processed query result from Apache Jena, many logical customisations in Java was required in order to gather and display all the results in a form of a table.

Conclusion

The entire activity to explore the domain of Music by gathering datasets, defining competency questions, defining an initial draft of the schema and ontology, and refining that ontology in an iterative way by including properties at both data and object level to finally build SPARQL queries and answers the questions opened the world of knowledge engineering and it's applications.
Our take-aways from the activity were the following:

1) Choosing sparsely correlated dataset was important as connecting it using ontology was the basis of this project.
2) Competency questions were a means of exploring the dataset, and defining what we'd like to extract and present from it. They are the building blocks of ontological design process.
3) An initial connection and schema design, prepared us for upliftment process and assisted us in selecting the right properties for our ontology.
4) Designing ontology is an iterative process and we had to go back to review our schemas, apply joins on datasets, and rethink our class connections when we finally connected ontology with our SPARQL queries.
5) SPARQL query language helped us use the defined properties and mine knowledge on our Classes and their inter-relations.
6) Apache Jena Framework is neatly integrated with Java Swing which makes it easier for the end-users to use the Application and also to know more about various interesting facts of Music.

End.