Information Integration of four datasets

Gabriele Matini 1934803, Alessio Maiola 1933477

1 Introduction: the idea

We start with four csv datasets about music related data. The objective is to define an Information Integration System that can take the data from the four different datasets and reconcile them. Pentaho is an information integration software that we use in order to achieve this objective, and we will use the materialization approach. Moreover we will also try a virtualization example of one query, finding a perfect rewriting for the query and demonstrating that it is, in fact, a perfect rewriting. The datasets are: a spotify tracks csv with id, names, genre, artist and so on, a similar spotify and youtube tracks csv, an artist csv with name, country of artist and so on, a csv with the top 100 tracks on spotify from 2010 to 2019.

2 Source Schema

Column Name	**Column Description**
title	Song's Title
artist	Song's artist
genre	Genre of song
year released	Year the song was released
added	Day song was added to Spotify's Top Hits playlist
bpm	Beats Per Minute - The tempo of the song
nrgy	Energy - How energetic the song is
dnce	Danceability - How easy it is to dance to the song
dB	Decibel - How loud the song is
live	How likely the song is a live recording
val	How positive the mood of the song is
dur	Duration of the song
acous	How acoustic the song is
spch	The more the song is focused on spoken word
pop	Popularity of the song (not a ranking)
top year	Year the song was a top hit
artist type	Tells if artist is solo, duo, trio, or a band

Table 1: Top 100 tracks csv data

Column Name	**Column Description**
mbid	MusicBrainz ID
artist_mb	Artist name according to MusicBrainz
artist_lastfm	Artist name according to Last.fm
country_mb	Artist country according to MusicBrainz
country_lastfm	Artist country, based on Last.fm tags
tags_mb	Artist tags on MusicBrainz, separated by semicolon (;)

Continued on next page

Table 2 – *Continued from previous page*

Column Name	Column Description
tags_lastfm	Artist tags on Last.fm, separated by semicolon (;), sorted by frequency decreasing
listeners_lastfm	Number of listeners on Last.fm
scrobbles_lastfm	Number of scrobbles on Last.fm
ambiguous_artist	TRUE if more than one artist shares the same Last.fm page

Table 2: Description of Artists' csv data

Column Name	Column Description
id	Unique identifier for the track on Spotify.
name	Name of the track
genre	Genre of the song
artists	Names of the artists who performed the track, separated by commas if there are multiple artists
album	Name of the album the track belongs to
popularity	Popularity score of the track (0-100, where higher is more popular)
duration_ms	Duration of the track in milliseconds
explicit	Boolean indicating whether the track contains explicit content

Table 3: Spotify track data

Column Name	Column Description
Id	Id of track.
Track	Name of the song, as visible on the Spotify platform.
Artist	Name of the artist.
Url_spotify	The URL of the artist.
Album	The album in which the song is contained on Spotify.
Album_type	Indicates if the song is released on Spotify as a single or contained in an album.
Uri	A Spotify link used to find the song through the API.
Danceability	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 value of 0.0 is least danceable and 1.0 is most danceable.
Energy	Is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.
Key	The key the track is in. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C# / D#, 2 = D, and so on. If no key was detected, the value is -1.

Continued on next page

Table 4 – *Continued from previous page*

Column Name	**Column Description**
Loudness	The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typically range between -60 and 0 dB.
Speechiness	Detects 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.
Acousticness	A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.
Instrumentalness	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.
Liveness	Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.
Valence	A 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).
Tempo	The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.
Duration_ms	The duration of the track in milliseconds.
Stream	Number of streams of the song on Spotify.
Url_youtube	URL of the video linked to the song on YouTube, if it has any.
Title	Title of the videoclip on YouTube.
Channel	Name of the channel that published the video.
Views	Number of views.
Likes	Number of likes.
Comments	Number of comments.
Description	Description of the video on YouTube.
Licensed	Indicates whether the video represents licensed content, which means that the content was uploaded to a channel linked to a YouTube content partner and then claimed by that partner.
official_video	Boolean value that indicates if the video found is the official video of the song.

Table 4: Description of Columns for Spotify and YouTube Data CSV

2.1 Source schema formalization

We simply extract the information from all the source files, denormalizing the information about the artists in T3.
T1($\vec{a}$)
T2($\vec{b}$)
T3($c_1, c_2, c_3, c_5, c_6, c_7, c_8$)
T3$_{artists}$(c_2, c_4)
T4($\vec{d}$)

3 Global Schema

Album(id_album: string, name: string)
Artist(id_artist: string, name: string, type: string, country: string, num_listener: integer, num_crobbles: integer, is_ambiguous: boolean)
AlbumArtist(id_album: string, id_artist: string)
Song(id_song: int, title: string, genre: string, year: int, id_album: int)
SongSpotify(id_song: int, uri: string, bpm: float, energy: float, danceability: float, loudness: float, liveness: float, acousticness: float, speechiness: float, instrumentalness: float, valence: float, key:int, duration: int, popularity: int, explicit: boolean, added: date, top year:int, streams: integer)
SongYoutube(id_song: int, url: string, channel: string, views: int, likes: int, comments: int, description: string, licensed: boolean, official_video: boolean)
SongArtist(id_song: string, id_artist: string)

3.1 Global schema constraints

In addition to the primary key constraints, due to the usage of IDs, we need to specify the foreign key constraints.
$Song[id_{album}] \subseteq Album[id_{album}]$
$AlbumArtist[id_{album}] \subseteq Album[id_{album}]$
$AlbumArtist[id_{artist}] \subseteq Artist[id_{artist}]$
$SongArtist(id_{artist}) \subseteq Artist[id_{artist}]$
$SongArtist(id_{song}) \subseteq Song[id_{song}]$
$SongSpotify[id_{song}] \subseteq Song[id_{song}]$
$SongYoutube(id_{song}) \subseteq Song[id_{song}]$

Moreover, due to the duplicate elimination before adding the IDs, we have that all the fields in Song, Album and Artist are unique (excluding, of course, the IDs). For the same reason, we have that all the fields in SongYoutube and SongSpotify are primary keys as well (we omitted the underlining for readability reasons).

4 Mapping

In order to reconcile similar info on the same songs, we've resorted to a hierarchy of information: this is because some some datasets hold the same information (with regard to semantics) for the same songs. We've thus resorted to choose a hierarchy where information from T3 gets the priority over T4 and T4 gets the priority over T1. For the artists, it was sufficient to integrate the information from T2 with those found in T1, T3 and T4, which include only the names of the artists, and only T1 adds another information which is the type of the artist (single, band, duo, etc...). For the artists it was necessary in T3 to deserialize the field "artists" which contained a list of artists. Thus, T3_artists was created to contain all the artists in a tuple for each artist. As per the albums, it was sufficient to integrate the names found in T3 and in T4 (where the album type is "album"). IDs were added to each album, artist and song to identify them unequivocally. All in all, the global schema features three main semantic objects, which are Song, Artist and Album.

4.1 T3 Mapping

We will use all the information from this source, therefore, only one mapping assertion, where we spit the original information into the tables of the global schema.

$$\begin{aligned}
&\forall \vec{y}, a.(T3(\vec{y}) \wedge T3_{artists}(y_2, a)) \rightarrow \exists id_{album}, id_{song}, id_{artist}.(Album(id_{album}, y_5) \\
&\wedge \exists \vec{v}.(Artist(id_{artist}, a, \vec{v})) \\
&\wedge AlbumArtist(id_{album}, id_{artist}) \\
&\wedge \exists year.(Song(id_{song}, y_2, y_3, year, id_{album})) \\
&\wedge \exists uri, bpm, dan, liv, lou, ac, spe, ins, key, added, tyear, streams. \\
&\quad (SongSpotify(id_{song}, uri, bpm, dan, liv, lou, ac, spe, ins, key, y_7, y_6, y_8, added, tyear, streams)) \\
&\wedge SongArtist(id_{song}, id_{artist}))
\end{aligned} \tag{1}$$

4.2 T4 Mapping

4.2.1 Collect Album and Artist information

Here we specify how we generate the Album and AlbumArtist tables, increasing the information provided by T3. We also collect some information about the artist, but the way we dealt with merging additional information about artists will be described later on.

$$\begin{aligned}
&\forall \vec{z}.(T4(\vec{z}) \wedge z_6 = "album") \rightarrow \\
&\exists id_{album}, id_{song}, id_{artist}.(Album(id_{album}, z_5) \\
&\wedge \exists \vec{v}.(Artist(id_{artist}, z_3, \vec{v})) \\
&\wedge AlbumArtist(id_{album}, id_{artist}) \\
&\wedge \exists genre, year.(Song(id_{song}, z_2, genre, year, id_{album})))
\end{aligned} \tag{2}$$

4.2.2 Songs contained in T3 and T4

When the information about the same song, made by the same artist, is present in both sources, we guarantee the existence of mixed tuples (that satisfy also the previous and more generic mapping assertions). This pattern will be used for every kind of conflicting information in the source files.

$$\begin{aligned}
&\forall \vec{y}, \vec{z}, a.(T4(\vec{z}) \wedge T3(\vec{y}) \wedge T3_{artists}(y_2, a) \wedge y_2 = z_2 \wedge a = z_3) \rightarrow \\
&\exists id_{song}, id_{album}.(\exists year.(Song(id_{song}, z_2, y_3, year, id_{album})) \\
&\wedge \exists added, tyear.(SongSpotify(id_{song}, z_7, z_{17}, z_8, z_{11}, z_{15}, z_{13}, z_{12}, z_{14}, z_{10}, y_7, y_6, y_8, added, tyear, z_{19})) \\
&\wedge SongYoutube(id_{song}, z_{20}, z_{22}, z_{23}, z_{24}, z_{25}, z_{26}, z_{27}, z_{28}))
\end{aligned} \tag{3}$$

4.2.3 Songs contained in T4 and not in T3

Here we deal with the remaining tuples of T4 that don't contain songs already present in T3.

$$\begin{aligned}
&\forall \vec{z}.(T4(\vec{z}) \wedge \neg \exists \vec{y}, a.(T3(\vec{y}) \wedge T3_{artists}(y_2, a) \wedge y_2 = z_2 \wedge a = z_3)) \rightarrow \\
&\exists id_{song}, id_{artist}, id_{album}.(\exists \vec{v}.(Artist(id_{artist}, z_3, \vec{v})) \\
&\wedge \exists genre, year.(Song(id_{song}, z_2, genre, year, id_{album})) \\
&\wedge SongArtist(id_{song}, id_{artist}) \\
&\wedge \exists dur, pop, exp, added, tyear \\
&\wedge (SongSpotify(id_{song}, z_7, z_{17}, z_8, z_{11}, z_{15}, z_{13}, z_{12}, z_{14}, z_{10}, dur, pop, exp, added, tyear, z_{19})) \\
&\wedge SongYoutube(id_{song}, z_{20}, z_{22}, z_{23}, z_{24}, z_{25}, z_{26}, z_{27}, z_{28}))
\end{aligned} \tag{4}$$

4.3 T1 Mapping

4.3.1 Songs contained in T4, T3 and T1

Here we merge the information from all the sources, making the most complete tuples of our global schema. The additional information about the artists in T1 will dealt later, but the previous assertions already guarantee the existence of a tuple in SongArtist corresponding to the Artist of the song.

$$\begin{aligned}&\forall \vec{w}, \vec{y}, \vec{z}, a.(T1(\vec{w}) \wedge T4(\vec{z}) \wedge w_1 = z_2 \wedge w_2 = z_3 \wedge T3(\vec{y}) \wedge T3_{artists}(y_2, a) \wedge y_2 = w_1 \wedge a = w_2) \rightarrow \\ &\exists id_{song}, id_{album}.(Song(id_{song}, w_1, y_3, w_4, id_{album}) \\ &\wedge (SongSpotify(id_{song}, z_7, z_{17}, z_8, z_{11}, z_{15}, z_{13}, z_{12}, z_{14}, z_{10}, y_7, y_6, y_8, w_5, w_{16}, z_{19})))\end{aligned} \tag{5}$$

4.3.2 Songs contained in T1 and T4, but not in T3

$$\begin{aligned}&\forall \vec{w}, \vec{z}.(T1(\vec{w}) \wedge T4(\vec{z}) \wedge w_1 = z_2 \wedge w_2 = z_3 \wedge \neg\exists \vec{y}, a.(T3(\vec{y}) \wedge T3_{artists}(y_2, a) \wedge y_2 = w_1 \wedge a = w_2)) \rightarrow \\ &\exists id_{song}, id_{album}.(Song(id_{song}, w_1, w_3, w_4, id_{album}) \\ &\wedge \exists exp.(SongSpotify(id_{song}, z_7, z_{17}, z_8, z_{11}, z_{15}, z_{13}, z_{12}, z_{14}, z_{10}, w_{12}, w_{15}, exp, w_5, w_{16}, z_{19})))\end{aligned} \tag{6}$$

4.3.3 Songs contained in T1 and T3, but not in T4

$$\begin{aligned}&\forall \vec{w}, \vec{y}, a.(T1(\vec{w}) \wedge T3(\vec{y}) \wedge T3_{artists}(y_2, a) \wedge y_2 = w_1 \wedge a = w_2 \wedge \neg\exists \vec{z}.(T4(\vec{z}) \wedge z_2 = w_1 \wedge z_3 = w_2)) \rightarrow \\ &\exists id_{song}, id_{album}.(Song(id_{song}, w_1, y_3, w_4, id_{album}) \\ &\wedge \exists uri, lou, ins, key, streams. \\ &\quad (SongSpotify(id_{song}, uri, w_6, w_8, lou, w_{10}, w_{13}, w_{14}, ins, key, y_7, y_6, y_8, w_5, w_{16}, streams)))\end{aligned} \tag{7}$$

4.3.4 Songs contained only in T1

$$\begin{aligned}&\forall \vec{w}(T1(\vec{w}) \wedge \neg\exists \vec{y}, a.(T3(\vec{y}) \wedge T3_{artists}(y_2, a) \wedge y_2 = w_1 \wedge a = w_2) \wedge \neg\exists \vec{z}.(T4(\vec{z}) \wedge z_2 = w_1 \wedge z_3 = w_2)) \rightarrow \\ &\exists id_{song}, id_{album}, id_{song}.(Song(id_{song}, w_1, w_3, w_4, id_{album}) \\ &\wedge \exists \vec{v}.(Artist(id_{artist}, w_3, w_{17}, \vec{v})) \\ &\wedge SongArtist(id_{song}, id_{artist}) \\ &\wedge \exists uri, lou, ins, key, exp, streams. \\ &\quad (SongSpotify(id_{song}, uri, w_6, w_8, lou, w_{10}, w_{13}, w_{14}, ins, key, w_{12}, w_{15}, exp, w_5, w_{16}, streams)))\end{aligned} \tag{8}$$

4.4 Artist mapping

The information about the artists is contained in all sources. While T3 and T4 only provide artist names, T1 and T2 contain additional fields that we can merge with the already specified information. We exploit once again the fact that we had previously only guaranteed the existence of a tuple, making now more strict the specification of the information that any database should contain in order to satisfy the mapping.

4.4.1 Artists only in T2

$$\forall \vec{x}.(T2(\vec{x})) \rightarrow \exists id_{artist}, type.(Artist(id_{artist}, x_3, type, x_5, x_8, x_9, x_{10})) \tag{9}$$

4.4.2 Artists in T1 and in T2

$$\forall \vec{w}, \vec{x}.(T1(\vec{w}) \wedge T2(\vec{x}) \wedge w_2 = x_3) \rightarrow \exists id_{artist}.(Artist(id_{artist}, w_2, w_{17}, x_5, x_8, x_9, x_{10})) \tag{10}$$

4.4.3 Artists only in T1

$$\forall \vec{w}.(T1(\vec{w})) \rightarrow \exists id_{artist}, \vec{v}.(Artist(id_{artist}, w_2, w_{17}, \vec{v})) \tag{11}$$

4.4.4 Artists in T2 and T4

$$\forall \vec{x}, \vec{z}.(T2(\vec{x})) \wedge T4(\vec{z}) \wedge y_3 = z_3 \rightarrow \exists id_{artist}, type.(Artist(id_{artist}, y_3, type, x_5, x_8, x_9, x_{10})) \tag{12}$$

4.4.5 Artists in T1 and T4

$$\forall \vec{w}, \vec{z}.(T1(\vec{w})) \wedge T4(\vec{z}) \wedge z_3 = w_2 \rightarrow \exists id_{artist}, \vec{v}.Artist(id_{artist}, w_2, w_{17}, \vec{v}) \tag{13}$$

4.4.6 Artists in T1, T2 and T4

$$\forall \vec{w}, \vec{z}.(T1(\vec{w})) \wedge T4(\vec{z}) \wedge T2(\vec{x}) \wedge z_3 = w_2 \wedge w_2 = x_3 \rightarrow \exists id_{artist}.Artist(id_{artist}, w_2, w_{17}, x_5, x_8, x_9, x_{10}) \tag{14}$$

4.4.7 Artists in T1 and in T3

$$\begin{aligned} &\forall \vec{w}, songname, a.(T1(\vec{w}) \wedge T3_{artists}(songname, a) \wedge w_2 = a) \\ &\rightarrow \exists id_{artist}, \vec{v}.(Artist(id_{artist}, w_2, w_{17}, \vec{v})) \end{aligned} \tag{15}$$

4.4.8 Artists in T2 and in T3

$$\begin{aligned} &\forall \vec{x}, songname, a.(T2(\vec{x}) \wedge T3_{artists}(songname, a) \wedge x_3 = a) \\ &\rightarrow \exists id_{artist}, type.Artist(id_{artist}, x_3, type, x_5, x_8, x_9, x_{10})) \end{aligned} \tag{16}$$

4.4.9 Artists in T1, T2 and in T3

$$\begin{aligned} &\forall \vec{w}, \vec{x}, songname, a.(T1(\vec{w}) \wedge T2(\vec{x}) \wedge T3_{artists}(songname, a) \wedge w_2 = x_3 \wedge w_2 = a) \\ &\rightarrow \exists id_{artist}.(Artist(id_{artist}, w_2, w_{17}, x_5, x_8, x_9, x_{10})) \end{aligned} \tag{17}$$

5 Queries in FOL

We've selected a few queries to do on our materialization of the database described by the global schema, which is a non-universal solution, as we use serial ids instead of marked nulls and obviously usual SQL nulls for missing information.

5.1 Top 10 song names by popularity and their artists, with the associated song popularity

$$
\begin{aligned}
&\{(songname, artistname, pop)| \\
&\exists id_{song}, id_{artist}.(SongArtist(id_{song}, id_{artist}) \wedge Song(id_{song}, songname, ...) \wedge Artist(id_{artist}, artistname, ...) \\
&\quad \wedge SongSpotify(id_{song}, ..., pop, ...) \\
&\quad \wedge (\neg \exists id_1, id_2, id_3, id_4, id_5, id_6, id_7, id_8, id_9, id_{10}.SongSpotify(id_1, ..., pop_1, ...) \wedge \\
&\quad SongSpotify(id_2, ..., pop_2, ...) \wedge \\
&\quad SongSpotify(id_3, ..., pop_3, ...) \wedge \\
&\quad SongSpotify(id_4, ..., pop_4, ...) \wedge \\
&\quad SongSpotify(id_5, ..., pop_5, ...) \wedge \\
&\quad SongSpotify(id_6, ..., pop_6, ...) \wedge \\
&\quad SongSpotify(id_7, ..., pop_7, ...) \wedge \\
&\quad SongSpotify(id_8, ..., pop_8, ...) \wedge \\
&\quad SongSpotify(id_9, ..., pop_9, ...) \wedge \\
&\quad SongSpotify(id_{10}, ..., pop_{10}, ...) \wedge \\
&\quad id_1 \neq id_2 \wedge id_1 \neq id_3 \wedge id_1 \neq id_4 \wedge id_1 \neq id_5 \wedge id_1 \neq id_6 \wedge id_1 \neq id_7 \wedge id_1 \neq id_8 \wedge id_1 \neq id_9 \wedge id_1 \neq id_{10} \\
&\quad \wedge id_2 \neq id_3 \wedge id_2 \neq id_4 \wedge id_2 \neq id_5 \wedge id_2 \neq id_6 \wedge id_2 \neq id_7 \wedge id_2 \neq id_8 \wedge id_2 \neq id_9 \wedge id_2 \neq id_{10} \\
&\quad \wedge id_3 \neq id_4 \wedge id_3 \neq id_5 \wedge id_3 \neq id_6 \wedge id_3 \neq id_7 \wedge id_3 \neq id_8 \wedge id_3 \neq id_9 \wedge id_3 \neq id_{10} \\
&\quad \wedge id_4 \neq id_5 \wedge id_4 \neq id_6 \wedge id_4 \neq id_7 \wedge id_4 \neq id_8 \wedge id_4 \neq id_9 \wedge id_4 \neq id_{10} \\
&\quad \wedge id_5 \neq id_6 \wedge id_5 \neq id_7 \wedge id_5 \neq id_8 \wedge id_5 \neq id_9 \wedge id_5 \neq id_{10} \\
&\quad \wedge id_6 \neq id_7 \wedge id_6 \neq id_8 \wedge id_6 \neq id_9 \wedge id_6 \neq id_{10} \\
&\quad \wedge id_7 \neq id_8 \wedge id_7 \neq id_9 \wedge id_7 \neq id_{10} \\
&\quad \wedge id_8 \neq id_9 \wedge id_8 \neq id_{10} \\
&\quad \wedge id_9 \neq id_{10} \\
&\quad pop < pop_1 \wedge pop < pop_2 \wedge pop < pop_3 \wedge pop < pop_4 \wedge pop < pop_5 \wedge pop < pop_6 \wedge pop < pop_7 \\
&\quad \wedge pop < pop_8 \wedge pop < pop_9 \wedge pop < pop_{10}))\}
\end{aligned}
\tag{18}
$$

5.2 All artists with their albums

$$
\begin{aligned}
&\{(artistname, albumname)|\exists id_{artist}, id_{album}, \vec{v}.(AlbumArtist(id_{artist}, id_{album}) \\
&\wedge Artist(id_{artist}, artistname, \vec{v}) \wedge Album(id_{album}, albumname))\}
\end{aligned}
\tag{19}
$$

5.3 All artists who have created more than three songs

$$
\begin{aligned}
&\{(artistname)|\exists id_{artist}.(Artist(id_{artist}, artistname, ...) \wedge (\exists id_1, id_2, id_3, id_4.id_1 \neq id_2 \wedge id_1 \neq id_3 \\
&\wedge id_1 \neq id_4 \wedge id_2 \neq id_3 \wedge id_2 \neq id_4 \wedge id_3 \neq id_4 \wedge SongArtist(id_1, id_{artist}) \wedge SongArtist(id_2, id_{artist}) \wedge \\
&SongArtist(id_3, id_{artist}) \wedge SongArtist(id_4, id_{artist}))\}
\end{aligned}
\tag{20}
$$

5.4 All songs belonging to an album with their album

$$
\{(songname, albumname)|\exists id_{song}, id_{album}.(Song(id_{song}, songname, ..., id_{album}) \wedge Album(id_{album}, albumname))\}
\tag{21}
$$

5.5 All songs that have both a Spotify url and a Youtube url

$$\{(songname, uri, url) | \exists id_{song}.(Song(id_{song}, songname, ...) \\ \wedge SongSpotify(id_{song}, uri, ...) \wedge SongYoutube(id_{song}, url, ...))\} \tag{22}$$

6 Virtualization of a query

We will use the virtualization approach for another query: find all albums. The query on the global schema to accomplish such a task is simple enough:

$$\{(albumname) | \exists id_{album}.Album(id_{album}, albumname)\} \tag{23}$$

Let's focus on the two mapping assertions that populate the Album table:

$$\forall \vec{y}.(T3(\vec{y})) \rightarrow \exists id_{album}.(Album(id_{album}, y_5)) \tag{24}$$

$$\forall \vec{z}.(T4(\vec{z}) \wedge z_6 = "album") \rightarrow \exists id_{album}.(Album(id_{album}, z_5)) \tag{25}$$

These two mapping assertions lead to the following equivalent formulas:

$$\forall \vec{y}.\neg \exists id_{album}.(Album(id_{album}, y_5)) \rightarrow \neg T3(\vec{y}) \tag{26}$$

$$\forall \vec{z}.\neg \exists id_{album}.(Album(id_{album}, z_5)) \rightarrow \neg T4(\vec{z}) \vee z_6 \neq "album" \tag{27}$$

This reformulation highlights the dependence of the Album tuples from the source datasets T3 and T4. This means that in the certain answers of our query on the global schema we are going to have solely the tuples belonging to T3 and T4, implying that the perfect rewriting on the source schema should necessarily query T3 and T4. Thus, the certain answers that we get by querying T3 and T4 are the same we get by querying Album. Our chosen candidate perfect rewriting is:

$$\{(albumname) | \exists \vec{y}.(T3(y_1, y_2, y_3, y_4, albumname, ...)) \vee \exists \vec{z}.(T4(z_1, z_2, z_3, z_4, albumname, ...) \wedge z_6 = "album")\} \tag{28}$$

Let us call q_s the query-rewriting on the source schema and q_g the query on the global schema.

We will now prove that, every tuple belonging to the certain answers of q_s is also in the certain answers of q_g and vice-versa in our information integration system. First of all, let t be a tuple in the certain answers of q_s. Suppose t is not in the certain answers of q_g. By our mapping, this means that the album name in t would be neither in a tuple of T4, nor in a tuple of T3. But this cannot be the case, since we hypothesized that the returned album name is in the certain answers of q_s, which queries solely over T3 and T4. Secondly, let t' be a tuple in the certain answers of q_g. Suppose t' is not in the certain answers of the q_s. This means that for tuple t' it must hold : $\neg \exists \vec{y}.(T3(y_1, y_2, y_3, y_4, t', ...)) \wedge \neg \exists \vec{z}.(T4(z_1, z_2, z_3, z_4, t', ...) \wedge z_6 = "album")$. However, this means that our tuple t' is a tuple of the global schema that falsifies both hypotheses of the mapping assertions, so they are not in the certain answers of q_g as it was first assumed. □