Problem Definition:

Using Spotify's data, the objective is to enhance user engagement and content discovery by analyzing the dataset. The analysis aims to understand the user preferences, identify trends in the music industry and provide personalize recommendations based on the user behavior and other information.

Design of Data Warehouse:

Conceptual Model:

Tracks**:** Stores detailed information about tracks including their duration, popularity, explicitness, and position in albums.

- Id
- Name
- Duration
- disc_number

Albums: Holds attributes related to albums like name, type, release date, and popularity, enabling analysis based on album characteristics.

- Album_id
- Name
- Release_date
- Album_type
- Popularity

AudioFeatures: Stores audio features of tracks like acousticness, danceability, energy, etc., allowing for analysis based on the musical properties of tracks.

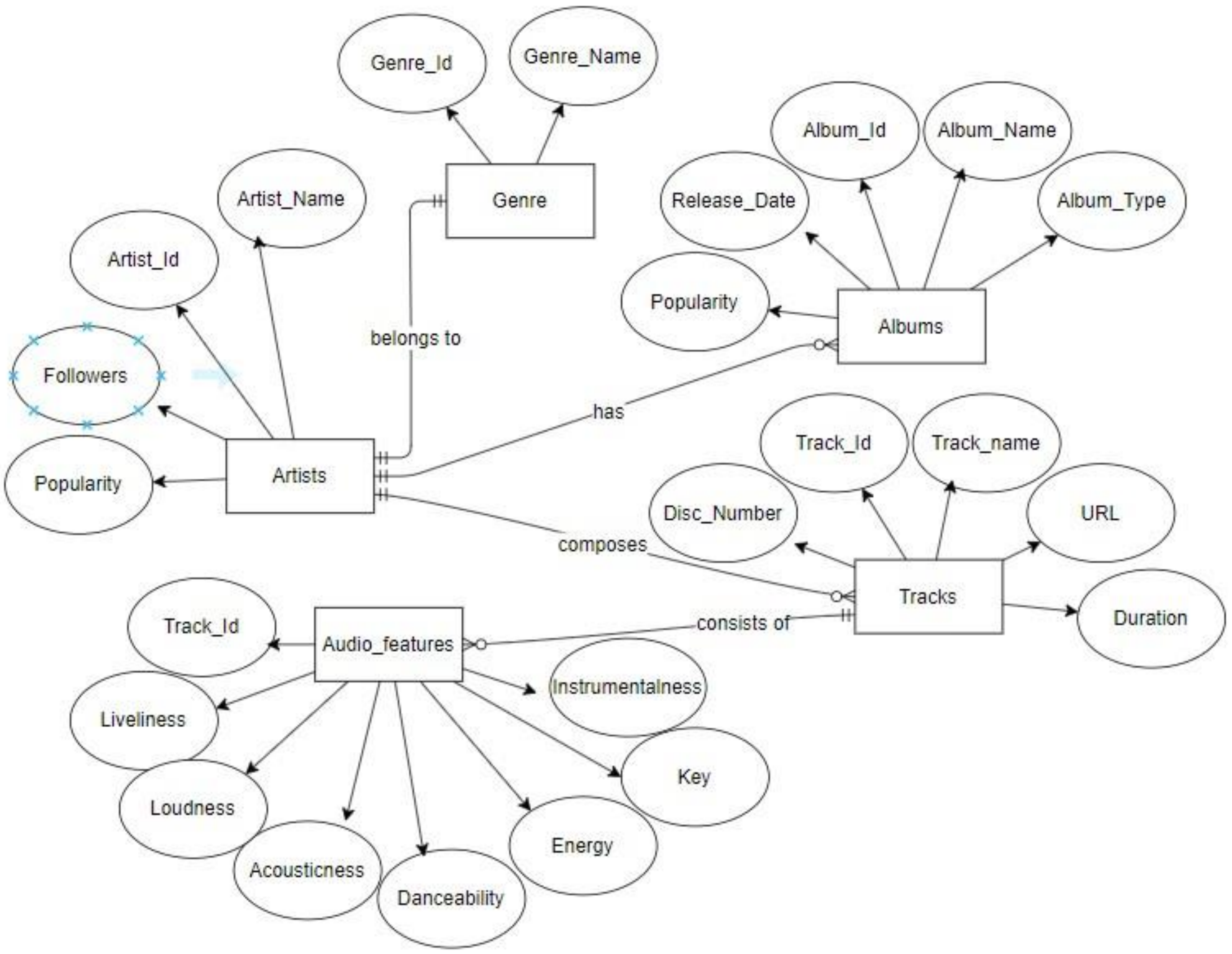
- Track_id
- Acousticness
- Danceability
- Energy
- Instrumentalness
- Key
- Liveness
- Loudness

Artist: Contains information about artists including their name, popularity, and followers, facilitating analysis based on artist-related factors.

- Artist_id
- Name
- Popularity
- Followers

Genre: Stores different genres of music, enabling analysis based on genre-specific trends and characteristics.

- Genre_id
- Genre_name



Logical Model:

Fact Table:

Tracks_info: Stores detailed information about tracks including their duration, popularity, explicitness, and position in albums.

Measures:

- track_id
- track_name
- track_total_duration
- track_popularity

- explicit
- track_number
- disc_number

Dimension Tables:

dimTime: This table will store all time related attributes.

- time_id (Primary Key)
- date
- day_of_week
- week_number
- month
- quarter
- year
- datetime

dimAlbums: This dimension table will store all the album related attributes.

- album_id (Primary Key)
- album_name
- album_type
- release_date_id (Foreign Key) ● popularity

dimAudioFeatures: This table will store all the track related audio features.

- audio_feature_id (Primary Key)
- acousticness
- danceability
- energy
- instrumentalness
- key
- liveness
- loudness
- mode
- speechiness
- tempo
- time_signature ● valence

dimArtist: This table will store all the artists related information.

- artists_id (Primary Key)
- artists_name

- popularity ● followers

Bridge Table:

Artists_Genre: Table to connect artists with Genres

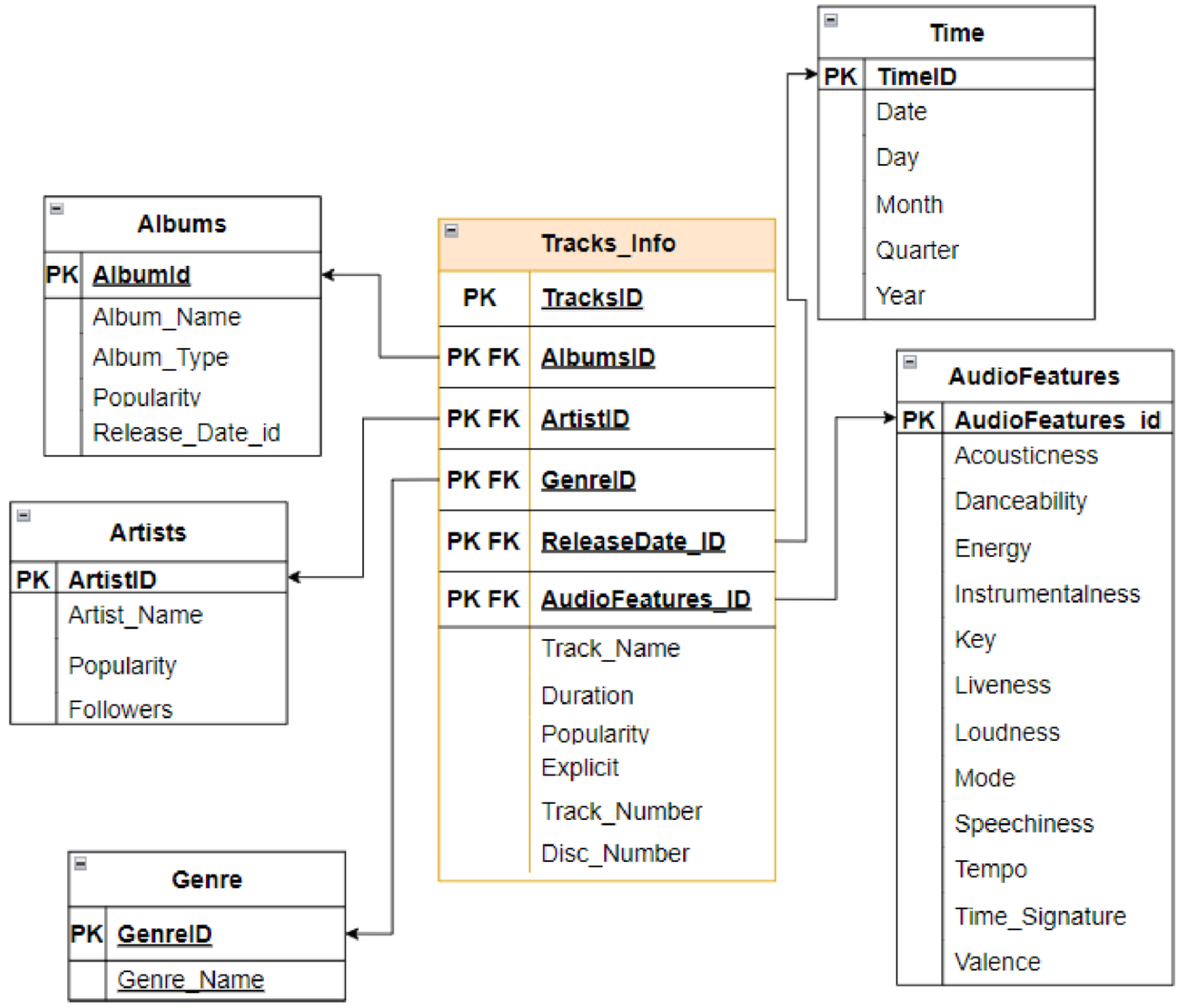
- ag_key
- artist_id
- genre_id

Hierarchies:

Time Hierarchy:

- Year > Quarter > Month > Day

OLAP Data Warehouse Creation script in Postgres:

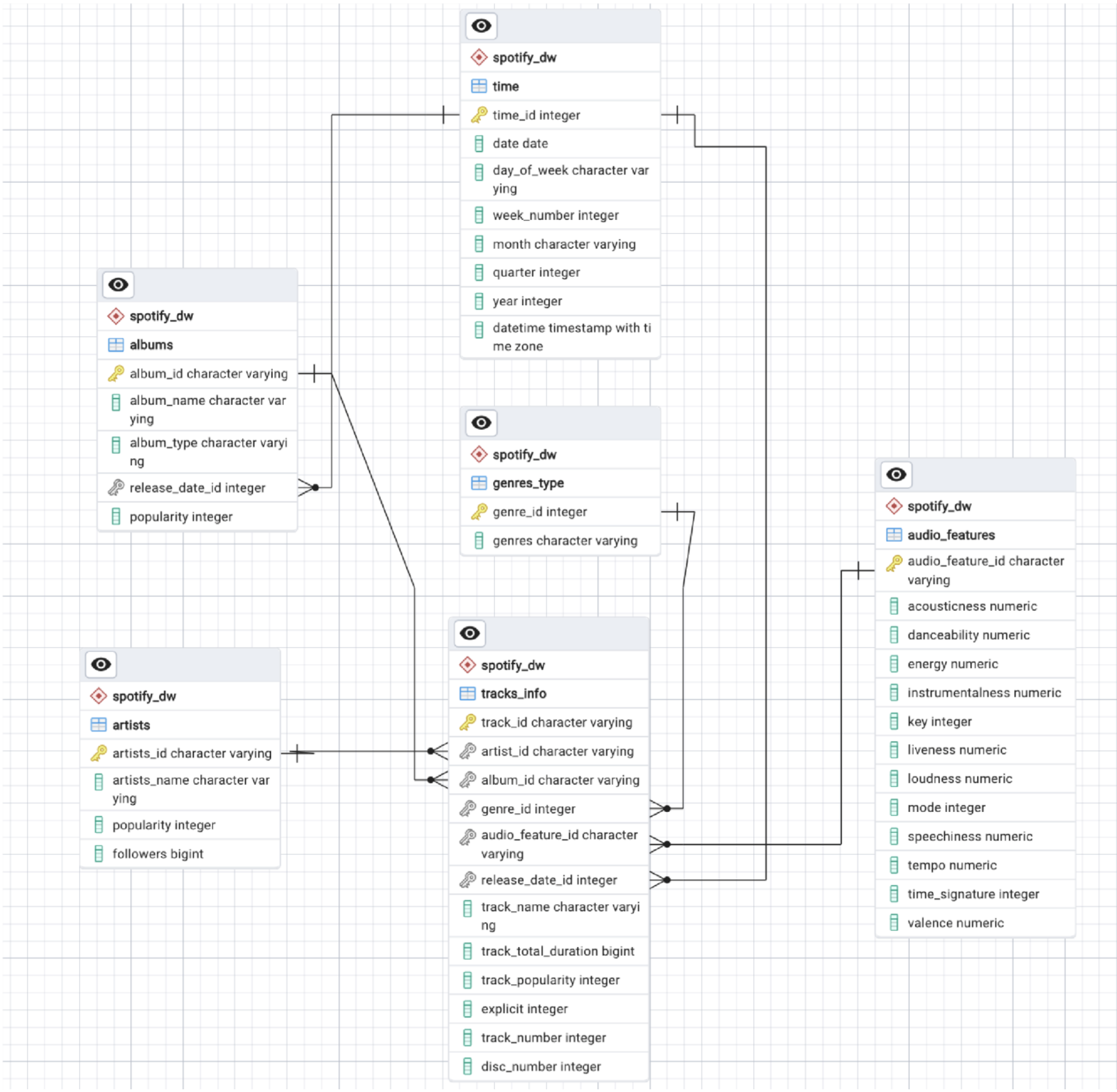


```
-- Database: spotify_datawarehouse

CREATE DATABASE spotify_datawarehouse
    WITH
    OWNER = postgres
    ENCODING = 'UTF8'
    LC_COLLATE = 'C'
    LC_CTYPE = 'C'
    LOCALE_PROVIDER = 'libc'
    TABLESPACE = pg_default
    CONNECTION LIMIT = -1
    IS_TEMPLATE = False;


-- SCHEMA: spotify_dw

-- DROP SCHEMA IF EXISTS spotify_dw ;

CREATE SCHEMA IF NOT EXISTS spotify_dw
    AUTHORIZATION postgres;

-- Table: spotify_dw.albums

-- DROP TABLE IF EXISTS spotify_dw.albums;

CREATE TABLE IF NOT EXISTS spotify_dw.albums
(
    album_id character varying COLLATE pg_catalog."default" NOT NULL,
    album_name character varying COLLATE pg_catalog."default",
    album_type character varying COLLATE pg_catalog."default",
    release_date_id integer,
    popularity integer,
    CONSTRAINT albums_pkey PRIMARY KEY (album_id),
    CONSTRAINT albums_release_date_id_fkey FOREIGN KEY (release_date_id)
        REFERENCES spotify_dw."time" (time_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS spotify_dw.albums
    OWNER to postgres;

-- Table: spotify_dw.artists

-- DROP TABLE IF EXISTS spotify_dw.artists;

CREATE TABLE IF NOT EXISTS spotify_dw.artists
(
    artists_id character varying COLLATE pg_catalog."default" NOT NULL,
    artists_name character varying COLLATE pg_catalog."default",
    popularity integer, followers bigint,
    CONSTRAINT artists_pkey PRIMARY KEY (artists_id)
)
```

```
TABLESPACE pg_default;

ALTER TABLE IF EXISTS spotify_dw.artists
    OWNER to postgres;

-- Table: spotify_dw.audio_features

-- DROP TABLE IF EXISTS spotify_dw.audio_features;

CREATE TABLE IF NOT EXISTS spotify_dw.audio_features
(
    audio_feature_id character varying COLLATE pg_catalog."default" NOT NULL,
    acousticness numeric, danceability numeric, energy numeric,
    instrumentalness numeric, key integer, liveness numeric, loudness numeric,
    mode integer, speechiness numeric, tempo numeric, time_signature integer,
    valence numeric,
    CONSTRAINT audio_features_pkey PRIMARY KEY (audio_feature_id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS spotify_dw.audio_features
    OWNER to postgres;

-- Table: spotify_dw.genres_type

-- DROP TABLE IF EXISTS spotify_dw.genres_type;

CREATE TABLE IF NOT EXISTS spotify_dw.genres_type

(

    genre_id integer NOT NULL, genres character varying

    COLLATE pg_catalog."default",

    CONSTRAINT "Genres_type_pkey" PRIMARY KEY (genre_id)

)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS spotify_dw.genres_type

    OWNER to postgres;

-- Table: spotify_dw.time
-- DROP TABLE IF EXISTS spotify_dw."time";

CREATE TABLE IF NOT EXISTS spotify_dw."time"
(
   time_id integer NOT NULL,
   date date,
   day_of_week character varying COLLATE pg_catalog."default",
   week_number integer,
   month character varying COLLATE
   pg_catalog."default", quarter integer, year integer,
   datetime timestamp with time zone,
   CONSTRAINT time_pkey PRIMARY KEY (time_id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS spotify_dw."time"
   OWNER to postgres;

-- Table: spotify_dw.tracks_info

-- DROP TABLE IF EXISTS spotify_dw.tracks_info;

CREATE TABLE IF NOT EXISTS spotify_dw.tracks_info
(
   track_id character varying COLLATE pg_catalog."default" NOT NULL,
   artist_id character varying COLLATE pg_catalog."default", album_id
   character varying COLLATE pg_catalog."default", genre_id integer,
   audio_feature_id character varying COLLATE pg_catalog."default",
   release_date_id integer,
   track_name character varying COLLATE pg_catalog."default",
   track_total_duration bigint,
   track_popularity integer,
   explicit integer,
   track_number integer,
   disc_number integer,
   CONSTRAINT tracks_info_pkey PRIMARY KEY (track_id),
   CONSTRAINT tracks_info_album_id_fkey FOREIGN KEY (album_id)
      REFERENCES spotify_dw.albums (album_id) MATCH SIMPLE
      ON UPDATE NO ACTION
      ON DELETE NO ACTION
      NOT VALID,
   CONSTRAINT tracks_info_artist_id_fkey FOREIGN KEY (artist_id)
      REFERENCES spotify_dw.artists (artists_id) MATCH SIMPLE
      ON UPDATE NO ACTION
```

```
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT tracks_info_audio_feature_id_fkey FOREIGN KEY (audio_feature_id)
        REFERENCES spotify_dw.audio_features (audio_feature_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT tracks_info_genre_id_fkey FOREIGN KEY (genre_id)
        REFERENCES spotify_dw.genres_type (genre_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT tracks_info_release_date_id_fkey FOREIGN KEY (release_date_id)
        REFERENCES spotify_dw."time" (time_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS spotify_dw.tracks_info
    OWNER to postgres;
```

Implementation of ROLAP schema in Postgres:



Primary events in the Data Warehouse:

- **Track Creation and Information Gathering:** This event involves the creation of new tracks by artists. It includes gathering detailed information about each track, such as its name, duration, popularity, explicitness, and its position within albums. This information is stored in the Tracks_Info fact table.

- **Album Release:** When an album is released, it becomes a significant event. Details about the album, including its name, type, release date, and popularity, are recorded in the Albums dimension table. The release date is linked to the time dimension to provide temporal context.

- **Artist Creation and Activity:** Artists are central figures in the music industry. Whenever a new artist emerges or an existing artist produces new tracks, their information is stored in the Artists dimension table. This includes details such as their name, popularity, and followers.

- **Genre Classification:** Genre classification is essential for organizing and categorizing music. Whenever a new genre is identified or assigned to a track, its details are recorded in the Genres_Type dimension table. This table serves as a reference for categorizing tracks based on their genre.

- **Audio Feature Extraction:** Audio features play a crucial role in analyzing and understanding music tracks. The extraction of audio features such as acousticness, danceability, energy, instrumentalness, and others is performed and stored in the Audio_Features dimension table. These features provide insights into the characteristics of each track.

- **Time Dimension Updates:** The time dimension table is continuously updated to include new time-related attributes such as dates, days of the week, week numbers, months, quarters, and years. These updates ensure that tracks and albums are associated with the appropriate temporal context.

OLAP operations that can be performed on the Tracks_Info fact table:

ROLLUP to Analyze Track Popularity by Year, Quarter, and Month:

- ROLLUP (Tracks_info, Time -> Year, dimTime -> Quarter, dimTime -> Month, SUM(track_popularity))

SORT to List Tracks by Popularity in Descending Order:

- SORT (Tracks_info, track_popularity DESC)

PIVOT to Compare Track Popularity by Genre and Year:

- PIVOT (Tracks_info, Genre -> X, Time -> Year -> Y, SUM(track_popularity))

SLICE to Analyze Popularity of Explicit Tracks:

- SLICE (Tracks_info, explicit, 1)

DICE to Focus on Tracks Released in 2020 and 2021:
- DICE (Tracks_info, (release_date_id IN (SELECT time_id FROM Time WHERE year IN (2020, 2021))))

DRILLDOWN to Examine Popularity of Tracks by Track Number:
- DRILLDOWN (Tracks_info, track_number)

PIVOT to Compare Track Popularity by Album and Artist:
- PIVOT (Tracks_info, Albums -> X, Artist -> Y, SUM(track_popularity))

Data Sources:

https://www.kaggle.com/datasets/maltegrosse/8-m-spotify-tracks-genre-audio-features

https://drive.google.com/file/d/1pVF06vpOSs3o1m5iDY2JXKYvyKcpGkhK/view?usp=sharing

https://drive.google.com/file/d/1--hjvwohmDm5P_8025iuG-u1G684BbO8/view?usp=sharing

https://drive.google.com/file/d/1-0Dzz0qNx5w4eWF-5Xj5yYbAY9y2FptW/view?usp=sharing

https://drive.google.com/file/d/1Xup9O_DU6sZeYvrRCOOnkuPnucxMDxbd/view?usp=sharing

Note: We first created an OLTP DB in postgres with all the data that is given above and we are ingesting data in talend using a db connection.

ETL Overview:

Extract: For the Spotify project, data is extracted from OLTP database containing track information, audio features, artist details, album data, and genre information.
Transform: Lookups were used to enrich data by referencing existing tables. This was done to increase reusability of jobs for inserting in empty tables or updating new records.
Load: Transformed data is loaded into the fact table (Tracks_info) and dimension tables (Time, Albums, AudioFeatures, Artist). Loading involves inserting, updating records in the database tables. Reused database connections are utilized to optimize the loading process.

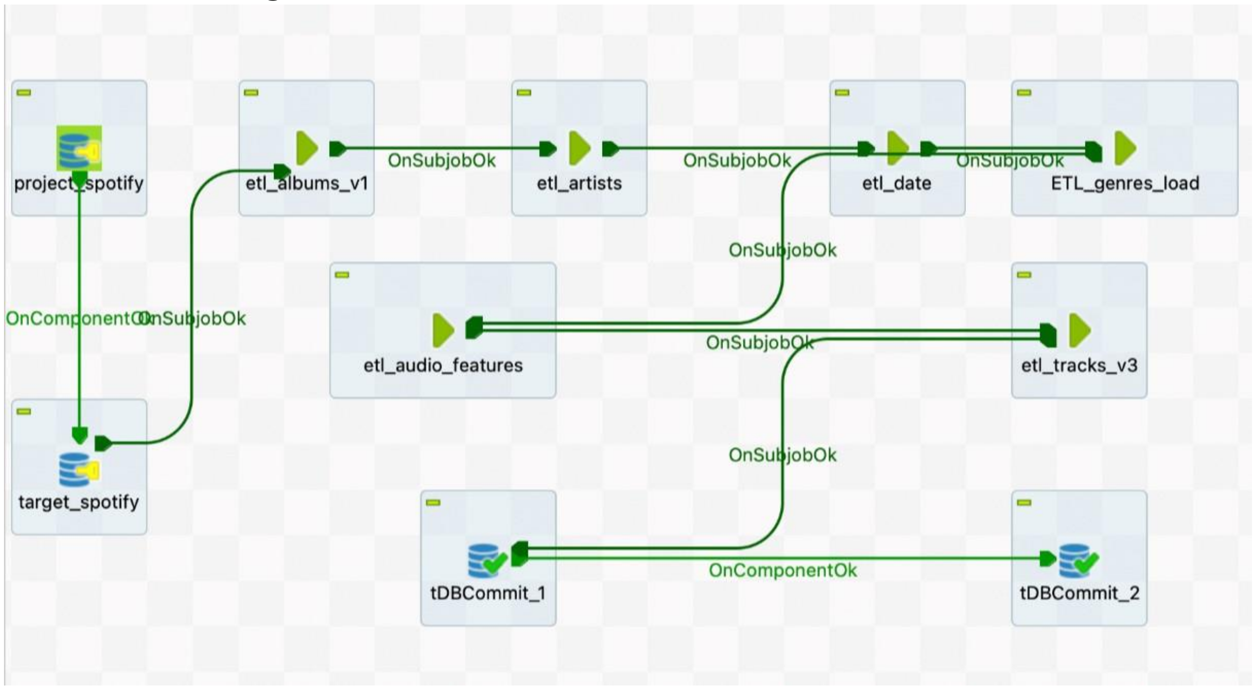
Transformations:

- Normalized data to ensure consistency and reduce redundancy. Ensured that track names are stored uniformly without leading or trailing spaces.
- Cleansed data to remove inconsistencies, errors, or missing values. Standardized data formats to ensure consistency. Ensured that date attributes are stored in a uniform format across all records
- Converted textual representations of dates to date data types for efficient date-based analysis.
- Identified and removed duplicate records based on unique identifiers like track_id or album_id while data loading.
- Standardized data formats to ensure consistency. Ensured that date attributes are stored in a uniform format across all records.
- Combined tracks information with artist, albums, and audio feature details for easy OLAP operations.

Complete and Incremental Loads:

- Performed a complete load when initially populating the data warehouse tables. This involves loading all records from the source data into the warehouse tables.
- Used TMaps to compare source data with existing data in the warehouse and loading only the changes. Used Keys and Constraints to insert new records, update existing records, and delete records that are no longer available in the source.

ETL Data flow diagram:



Documentation for ETL data flow:

Project_spotify: This is the db connection for our postgres operational database that we used to ingest data.

target_spotify: This is the db connection for our postgres data warehouse and is given to all the tables which includes connection to the data warehouse.

- Next all the process represent the data flow sequentially which starts from the albums table and ends with the tracks table.
- The commit tool will commit all the processes and will close the connection which is shared between all the entities.
- The data flow is given according to which the data was transformed.