Data Warehousing & Integration
IE6750
Spring 2025

Music Collaboration Platform

Milestone 4 (starts on pg. 19)
Group 5

Anushka Jami
Barath Keshav Sriram Kumaran

jami.a@northeastern.edu
sriramkumaran.b@northeastern.edu

3/30/2025

Problem Statement:

Those trying to make it in the music industry like independent musicians, struggle to be recognized. Most streaming services lean towards promoting mainstream music, making it harder for independent artists to put their music out and have a broad reach. This makes it difficult for producers, talent scouts, and record labels that are looking for unique and new talent as there is no good organization of independent music that they can get their hands on.

Moreover, many independent artists struggle to keep track of their streams, engagements, or even the interest that their work generates among producers. This alludes to a lack of transparency and fairness as it makes it hard for the independent artists to scope out their niches and make their music mainstream.

Project Overview:

The project aims to develop an open music-streaming platform where independent artists and aspiring musicians can upload their music to gain visibility and recognition from listeners and industry professionals alike. The platform will serve as a bridge between artists and music producers, talent scouts, and record labels by offering a robust database and discovery system. The primary focus is to create a fair, transparent, and efficient ecosystem for showcasing talent and fostering meaningful connections within the music industry.

Static Reference Data & Transactional Data:

Static Reference:

- Artists
- Listeners
- Producers
- Talent Scouts
- Record Labels

Transactional:

- Music uploaded
- Music streamed by listeners

- Songs added to playlists (added to Milestone 1)

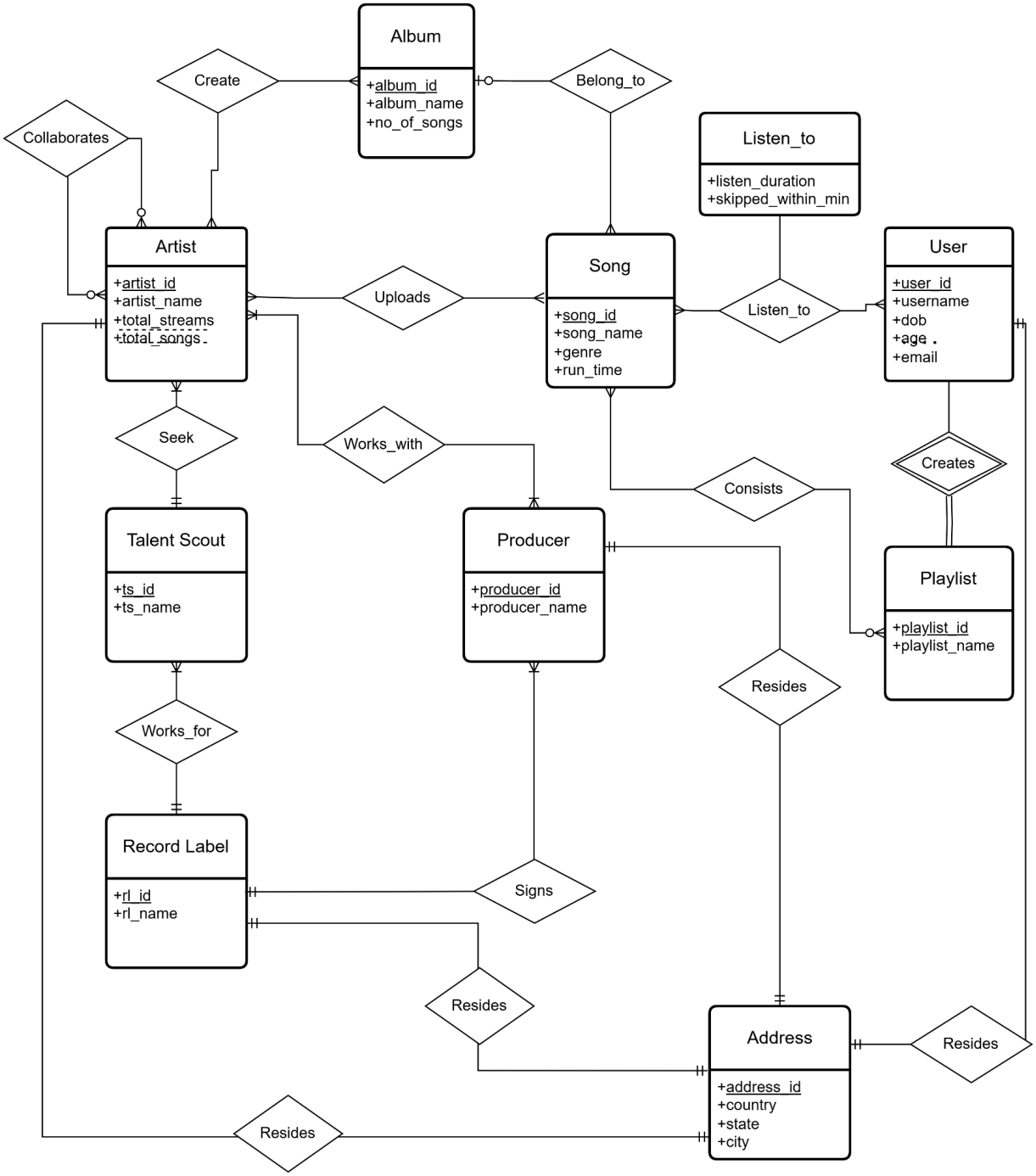
Analysis:

Streaming Rate Analysis: Analyzing the streaming trends of songs by particular artists help provide context to producers who are looking to recruit new upcoming talent. By analyzing trends in artist streaming rates, the artists with the fastest growing or largest fanbases can be identified for recruiters.

Genre Analysis: Analyzing trends in listening habits by listeners for different genres allows for prediction of what upcoming artists to invest in. For example, some genres could have fanbases who prefer independent artists, so producers should tend to avoid upcoming artists in that area. Using listener information can help inform general public listening habits to artists of the same genres.

Music Uploaded: Analyzing trends in the songs being uploaded to the platform can help predict what types of music artists want to create and users want to listen to. Using song length, number of collaborators, genres covered in new songs, and other characteristics, a picture of the upcoming music landscape can be created to allow for producers to select the talent with the most potential.

Entity Relationship Diagram:



Relational Model:

Album: album_id, album_name, no_of_songs
Create_Album: artist_id (FK), album_id (FK)
Artist: artist_id, artist_name, total_streams, total_songs, address_id (FK), ts_id (FK)(NOTNULL)
Collaborates: artist_id1 (FK), artist_id2 (FK), artist_id3 (FK)
Song: song_id, song_name, genre, length, run_time, album_id (FK)
Song_Upload: artist_id (FK), song_id (FK)
Users: user_id, username, DOB, email, age, address_id (FK)
Listen: user_id (FK), song_id (FK), listen_duration, skipped_within_min
Talent_Scout: ts_id, ts_name, rl_id (FK) (NOTNULL)
Record_Label: rl_id, rl_name, address_id (FK)
Producer: producer_id, producer_name, address_id (FK), rl_id (FK)(NOTNULL)
Artist_Prod: artist_id (FK), producer_id (FK)
Playlist: playlist_id, playlist_name, user_id (FK)(NOTNULL)
Playlist_Songs: playlist_id (FK), song_id (FK)
Address: address_id, country, state, city

DDL to Create Schema

```
CREATE TABLE Address (
    address_id INT PRIMARY KEY,
    country VARCHAR(100),
    state VARCHAR(100),
    city VARCHAR(100)
);

CREATE TABLE Record_Label (
    rl_id INT PRIMARY KEY,
    rl_name VARCHAR(255),
    address_id INT,
    FOREIGN KEY (address_id) REFERENCES Address(address_id)
);

CREATE TABLE Talent_Scout (
    ts_id INT PRIMARY KEY,
    ts_name VARCHAR(255),
    rl_id INT NOT NULL,
    FOREIGN KEY (rl_id) REFERENCES Record_Label(rl_id)
);

CREATE TABLE Producer (
    producer_id INT PRIMARY KEY,
    producer_name VARCHAR(255),
    address_id INT,
    rl_id INT NOT NULL,
    FOREIGN KEY (address_id) REFERENCES Address(address_id),
    FOREIGN KEY (rl_id) REFERENCES Record_Label(rl_id)
);

CREATE TABLE Artist (
    artist_id INT PRIMARY KEY,
    artist_name VARCHAR(255),
    total_streams INT,
    total_songs INT,
```

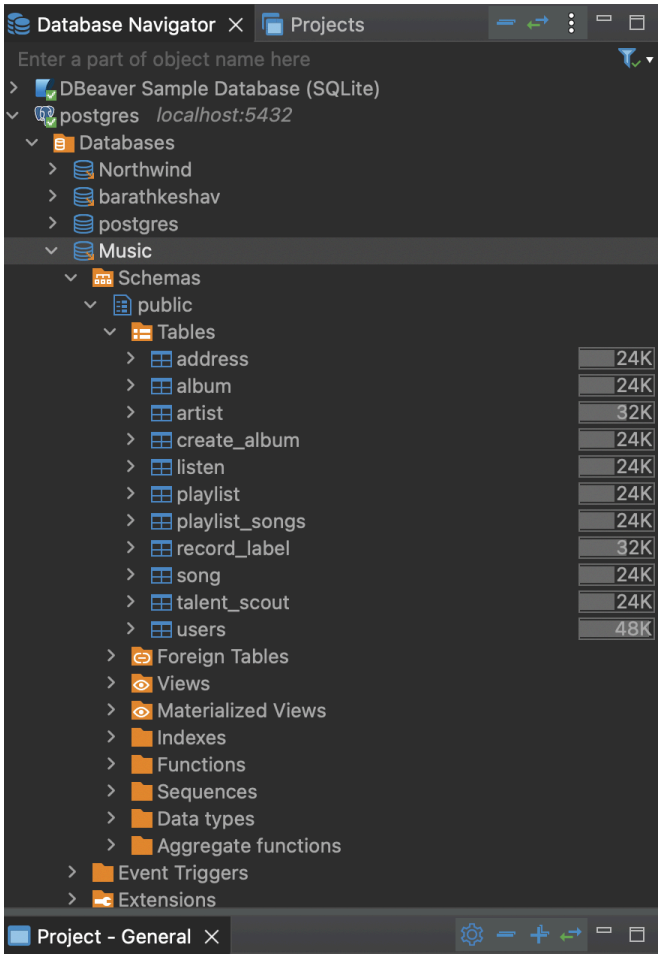
```
    address_id INT,
    ts_id INT NOT NULL,
    FOREIGN KEY (address_id) REFERENCES Address(address_id),
    FOREIGN KEY (ts_id) REFERENCES Talent_Scout(ts_id)
);

CREATE TABLE Album (
    album_id INT PRIMARY KEY,
    album_name VARCHAR(255),
    no_of_songs INT
);

CREATE TABLE Create_Album (
    artist_id INT,
    album_id INT,
    PRIMARY KEY (artist_id, album_id),
    FOREIGN KEY (artist_id) REFERENCES Artist(artist_id),
    FOREIGN KEY (album_id) REFERENCES Album(album_id)
);

CREATE TABLE Collaborates (
    artist_id1 INT,
    artist_id2 INT,
    artist_id3 INT,
    PRIMARY KEY (artist_id1, artist_id2, artist_id3),
    FOREIGN KEY (artist_id1) REFERENCES Artist(artist_id),
    FOREIGN KEY (artist_id2) REFERENCES Artist(artist_id),
    FOREIGN KEY (artist_id3) REFERENCES Artist(artist_id)
);

CREATE TABLE Song (
    song_id INT PRIMARY KEY,
    song_name VARCHAR(255),
    genre VARCHAR(100),
    length TIME,
    album_id INT,
    run_time INT,
    FOREIGN KEY (album_id) REFERENCES Album(album_id)
```

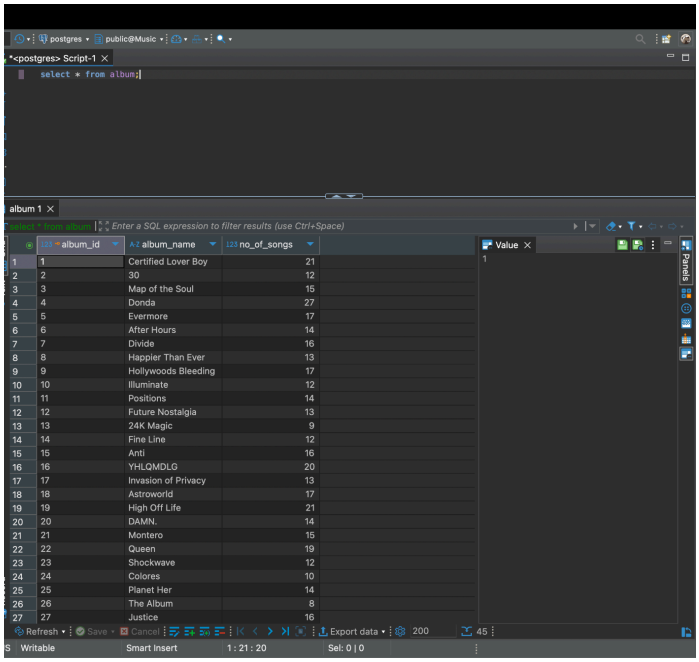
```
);



CREATE TABLE Song_Upload (
    artist_id INT,
    song_id INT,
    PRIMARY KEY (artist_id, song_id),
    FOREIGN KEY (artist_id) REFERENCES Artist(artist_id),
    FOREIGN KEY (song_id) REFERENCES Song(song_id)
);

CREATE TABLE Users (
    user_id INT PRIMARY KEY,
    username VARCHAR(255),
    DOB DATE,
    email VARCHAR(255),
    age INT,
    address_id INT,
    FOREIGN KEY (address_id) REFERENCES Address(address_id)
);

CREATE TABLE Listen (
    user_id INT,
    song_id INT,
    listen_duration float,
    skipped_within_min INT,
    PRIMARY KEY (user_id, song_id),
    FOREIGN KEY (user_id) REFERENCES Users(user_id),
    FOREIGN KEY (song_id) REFERENCES Song(song_id)
);

CREATE TABLE Artist_Prod (
    artist_id INT,
    producer_id INT,
    PRIMARY KEY (artist_id, producer_id),
```

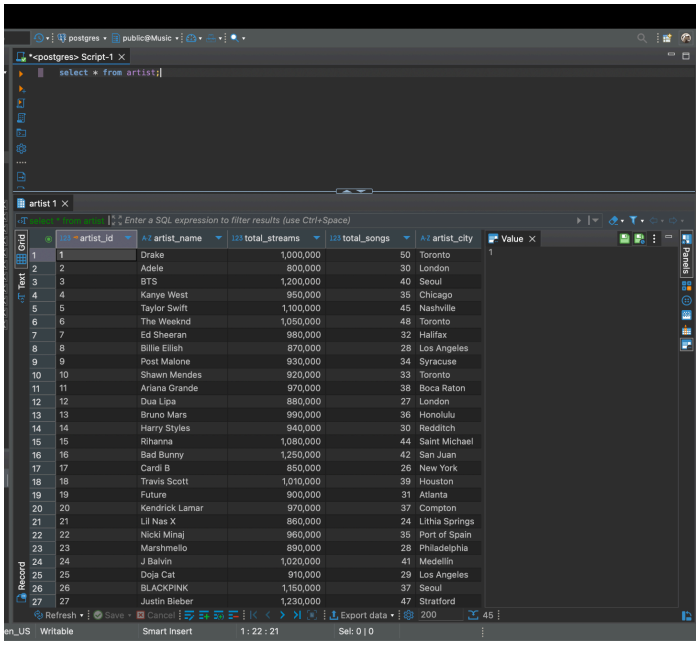
```
    FOREIGN KEY (artist_id) REFERENCES Artist(artist_id),
    FOREIGN KEY (producer_id) REFERENCES Producer(producer_id)
);

CREATE TABLE Playlist (
    playlist_id INT PRIMARY KEY,
    playlist_name VARCHAR(255),
    user_id INT NOT NULL,
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);

CREATE TABLE Playlist_Songs (
    playlist_id INT,
    song_id INT,
    PRIMARY KEY (playlist_id, song_id),
    FOREIGN KEY (playlist_id) REFERENCES Playlist(playlist_id),
    FOREIGN KEY (song_id) REFERENCES Song(song_id)
);
```

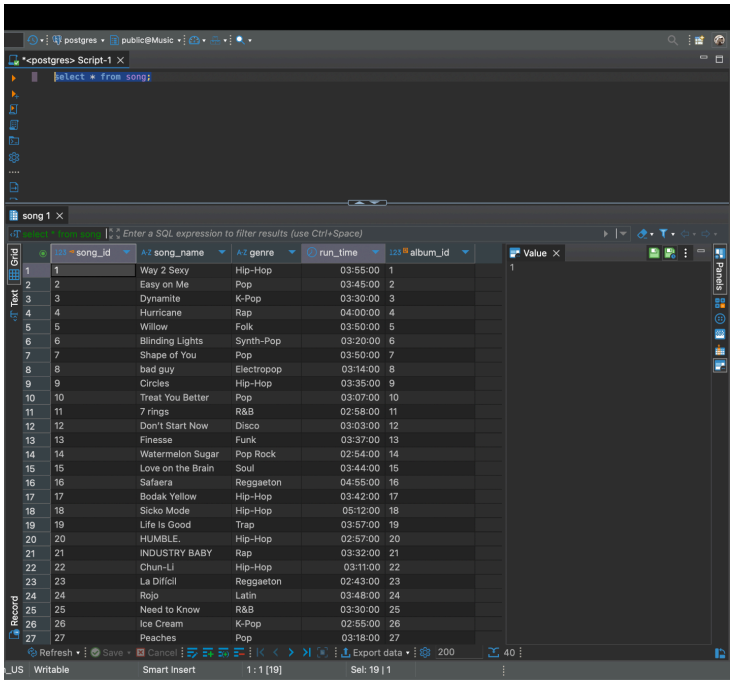
DML Data Population with Generated Data

Below is a screenshot of the tables within the schema as well as some examples of the data within the tables

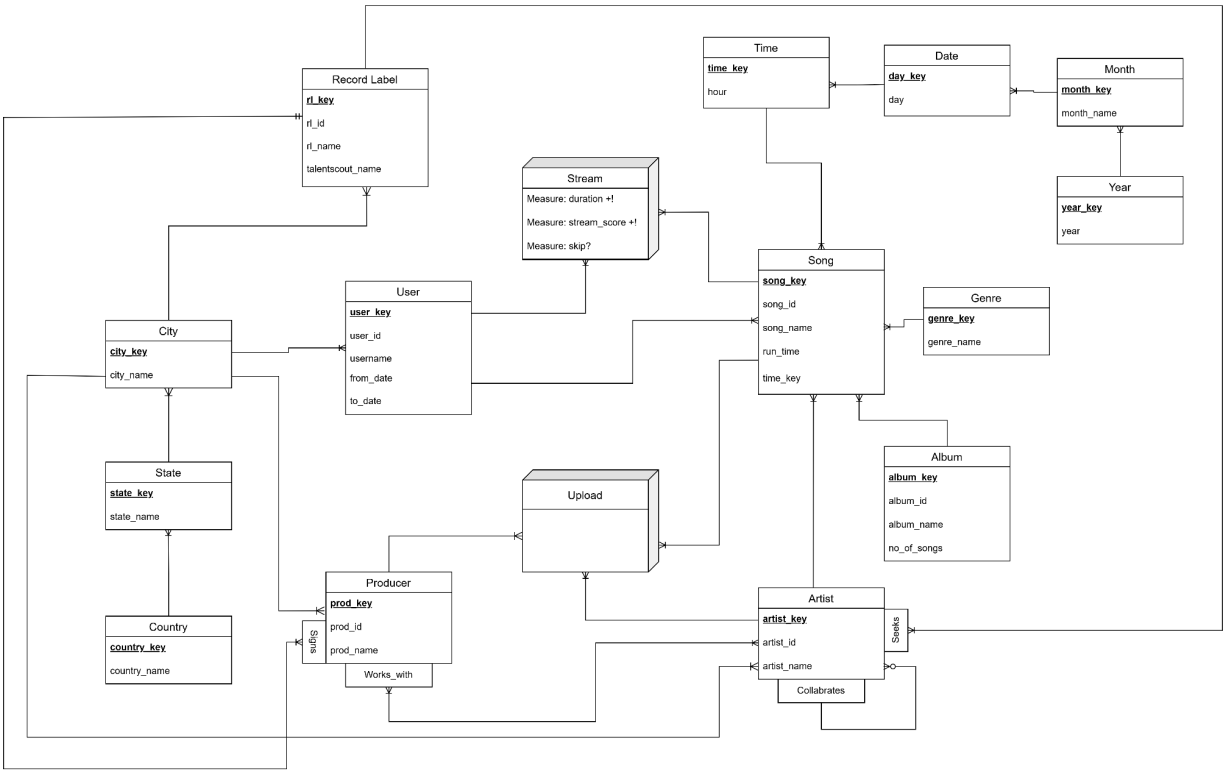

Database Navigator
Projects
Enter a part of object name here
DBeaver Sample Database (SQLite)
postgres localhost:5432
Databases
Northwind
barathkeshav
postgres
Music
Schemas
public
Tables
address 24K
album 24K
artist 32K
create_album 24K
listen 24K
playlist 24K
playlist_songs 24K
record_label 32K
song 24K
talent_scout 24K
users 48K
Foreign Tables
Views
Materialized Views
Indexes
Functions
Sequences
Data types
Aggregate functions
Event Triggers
Extensions
Project - General


postgres
public@Music
*<postgres> Script-1
select * from album;
album 1
Enter a SQL expression to filter results (use Ctrl+Space)
album_id
album_name
no_of_songs
Value
1 1 Certified Lover Boy 21
2 2 30 12
3 3 Map of the Soul 15
4 4 Donda 27
5 5 Evermore 17
6 6 After Hours 14
7 7 Divide 16
8 8 Happier Than Ever 13
9 9 Hollywoods Bleeding 17
10 10 Illuminate 12
11 11 Positions 14
12 12 Future Nostalgia 13
13 13 24K Magic 9
14 14 Fine Line 12
15 15 Anti 16
16 16 YHLQMDLG 20
17 17 Invasion of Privacy 13
18 18 Astroworld 17
19 19 High Off Life 21
20 20 DAMN. 14
21 21 Montero 15
22 22 Queen 19
23 23 Shockwave 12
24 24 Colores 10
25 25 Planet Her 14
26 26 The Album 8
27 27 Justice 16
Refresh
Save
Cancel
Export data
200
45
Writable
Smart Insert
1 : 21 : 20
Sel: 0 | 0


postgres
public@Music
*<postgres> Script-1
select * from artist;
artist 1
Enter a SQL expression to filter results (use Ctrl+Space)
artist_id
artist_name
total_streams
total_songs
artist_city
Value
1 1 Drake 1,000,000 50 Toronto
2 2 Adele 800,000 30 London
3 3 BTS 1,200,000 40 Seoul
4 4 Kanye West 950,000 35 Chicago
5 5 Taylor Swift 1,100,000 45 Nashville
6 6 The Weeknd 1,050,000 48 Toronto
7 7 Ed Sheeran 980,000 32 Halifax
8 8 Billie Eilish 870,000 28 Los Angeles
9 9 Post Malone 930,000 34 Syracuse
10 10 Shawn Mendes 920,000 33 Toronto
11 11 Ariana Grande 970,000 38 Boca Raton
12 12 Dua Lipa 880,000 27 London
13 13 Bruno Mars 990,000 36 Honolulu
14 14 Harry Styles 940,000 30 Redditch
15 15 Rihanna 1,080,000 44 Saint Michael
16 16 Bad Bunny 1,250,000 42 San Juan
17 17 Cardi B 850,000 26 New York
18 18 Travis Scott 1,010,000 39 Houston
19 19 Future 900,000 31 Atlanta
20 20 Kendrick Lamar 970,000 37 Compton
21 21 Lil Nas X 860,000 24 Lithia Springs
22 22 Nicki Minaj 960,000 35 Port of Spain
23 23 Marshmello 890,000 28 Philadelphia
24 24 J Balvin 1,020,000 41 Medellín
25 25 Doja Cat 910,000 29 Los Angeles
26 26 BLACKPINK 1,150,000 37 Seoul
27 27 Justin Bieber 1,230,000 47 Stratford
Refresh
Save
Cancel
Export data
200
45
en_US
Writable
Smart Insert
1 : 22 : 21
Sel: 0 | 0


postgres
public@Music
*<postgres> Script-1
select * from song;
song 1
Enter a SQL expression to filter results (use Ctrl+Space)
song_id
song_name
genre
run_time
album_id
Value
1 1 Way 2 Sexy Hip-Hop 03:55:00 1
2 2 Easy on Me Pop 03:45:00 2
3 3 Dynamite K-Pop 03:30:00 3
4 4 Hurricane Rap 04:00:00 4
5 5 Willow Folk 03:50:00 5
6 6 Blinding Lights Synth-Pop 03:20:00 6
7 7 Shape of You Pop 03:50:00 7
8 8 bad guy Electropop 03:14:00 8
9 9 Circles Hip-Hop 03:35:00 9
10 10 Treat You Better Pop 03:07:00 10
11 11 7 rings R&B 02:58:00 11
12 12 Don't Start Now Disco 03:03:00 12
13 13 Finesse Funk 03:37:00 13
14 14 Watermelon Sugar Pop Rock 02:54:00 14
15 15 Love on the Brain Soul 03:44:00 15
16 16 Safaera Reggaeton 04:55:00 16
17 17 Bodak Yellow Hip-Hop 03:42:00 17
18 18 Sicko Mode Hip-Hop 05:12:00 18
19 19 Life Is Good Trap 03:57:00 19
20 20 HUMBLE. Hip-Hop 02:57:00 20
21 21 INDUSTRY BABY Rap 03:32:00 21
22 22 Chun-Li Hip-Hop 03:11:00 22
23 23 La Difícil Reggaeton 02:43:00 23
24 24 Rojo Latin 03:48:00 24
25 25 Need to Know R&B 03:30:00 25
26 26 Ice Cream K-Pop 02:55:00 26
27 27 Peaches Pop 03:18:00 27
Refresh
Save
Cancel
Export data
200
40
Writable
Smart Insert
1 : 1 [19]
Sel: 19 | 1

MultiDim Schema



Logical Data Warehouse Model

Stream: stream_key(PK), users_key (PK, FK), song_key (PK, FK), time_key(PK, FK), duration, stream_score, skip?

Upload:upload_key(PK), prod_key(PK, FK), song_key(PK, FK), artist_key(PK, FK), time_key(PK, FK)

Year:year_key(PK), year

Month:month_key(PK), month_name, year_key(FK)

Date:day_key(PK), day, month_key(FK)

Time:time_key(PK), hour, day_key(FK)

Artist: artist_key, artist_id, artist_name, rl_key(FK), collaborator_key(FK), city_key(FK)

Prod_Artist: prod_key(PK, FK), artist_key(PK, FK)

Producer: prod_key, prod_id, prod_name, rl_key(FK), city_key(FK)

Country: country_key, country_name

State: state_key, state_name, country_key(FK)

City: city_key, city_name, state_key(FK)

Users: users_key, users_id, username, from_date, to_date, city_key(FK)

Genre: genre_key, genre_name

Album: album_key, album_id, album_name, no_of_songs

Songs: song_key, song_id, song_name, run_time, album_key(FK), genre_key(FK), artist_key (FK), users_key (FK)

Record_label: rl_key, rl_id, rl_name, talentscout_name, city_key (FK)

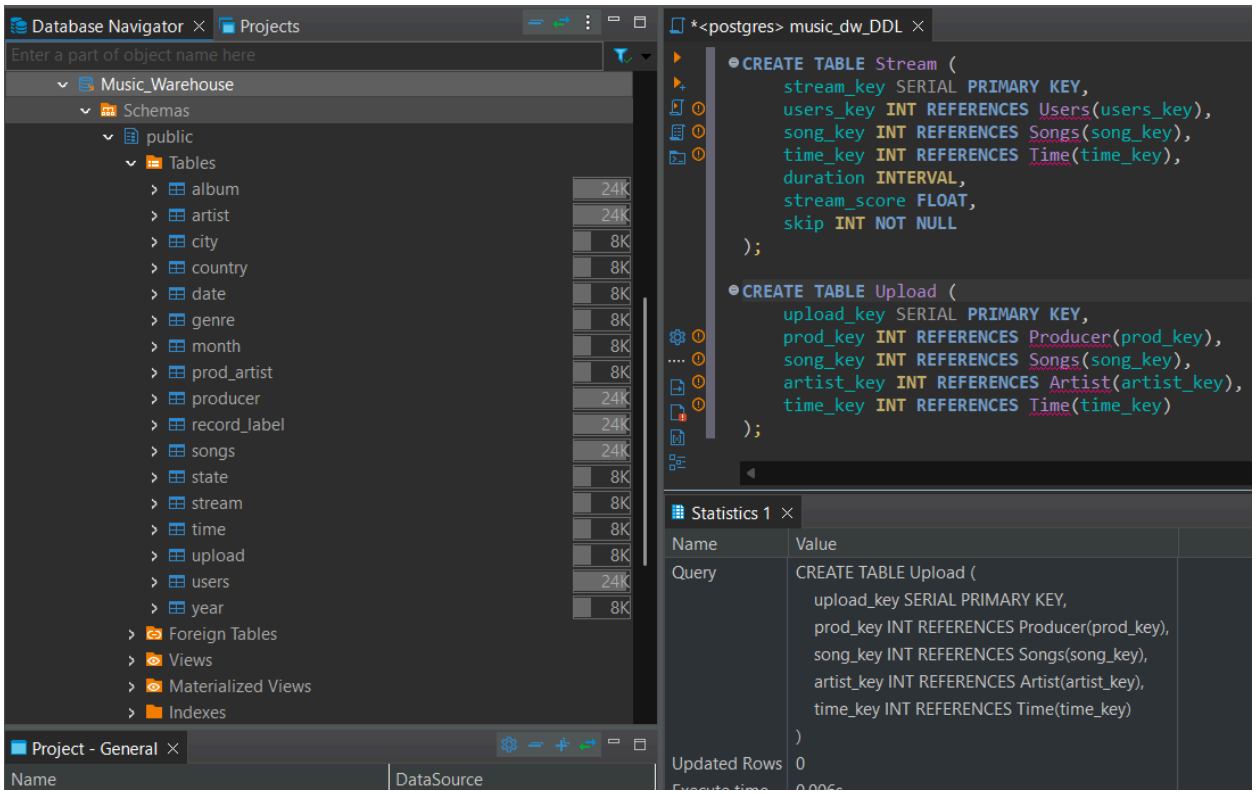
DB Implementation

```
CREATE TABLE Country (
    country_key VARCHAR(255) PRIMARY KEY,
    country_name VARCHAR(255) NOT NULL
);

CREATE TABLE State (
    state_key VARCHAR(255) PRIMARY KEY,
    state_name VARCHAR(255) NOT NULL,
    country_key VARCHAR(255) REFERENCES Country(country_key)
);

CREATE TABLE City (
    city_key VARCHAR(255) PRIMARY KEY,
    city_name VARCHAR(255) NOT NULL,
    state_key VARCHAR(255) REFERENCES State(state_key)
);

CREATE TABLE Record_label (
    rl_key VARCHAR(255) PRIMARY KEY,
    rl_id VARCHAR(255) UNIQUE NOT NULL,
    rl_name VARCHAR(255) NOT NULL,
    talentscout_name VARCHAR(255),
    ts_id VARCHAR(255) UNIQUE NOT NULL,
    city_key VARCHAR(255) REFERENCES City(city_key)
);


CREATE TABLE Producer (
    prod_key VARCHAR(255) PRIMARY KEY,
    prod_id VARCHAR(255) UNIQUE NOT NULL,
    prod_name VARCHAR(255) NOT NULL,
    rl_key VARCHAR(255) REFERENCES Record_label(rl_key),
    city_key VARCHAR(255) REFERENCES City(city_key)
);
```

```
CREATE TABLE Artist (
    artist_key VARCHAR(255) PRIMARY KEY,
    artist_id VARCHAR(255) UNIQUE NOT NULL,
    artist_name VARCHAR(255) NOT NULL,
    rl_key VARCHAR(255) REFERENCES Record_label(rl_key),
    collaborator_key VARCHAR(255) REFERENCES Artist(artist_key),
    city_key VARCHAR(255) REFERENCES City(city_key)
);



CREATE TABLE Prod_Artist (
    prod_key VARCHAR(255) REFERENCES Producer(prod_key),
    artist_key VARCHAR(255) REFERENCES Artist(artist_key),
    PRIMARY KEY (prod_key, artist_key)
);

CREATE TABLE Users (
    users_key VARCHAR(255) PRIMARY KEY,
    users_id VARCHAR(255) UNIQUE NOT NULL,
    username VARCHAR(255) NOT NULL,
    users_street_add TEXT,
    from_date DATE,
    to_date DATE,
    city_key VARCHAR(255) REFERENCES City(city_key)
);



CREATE TABLE Genre (
    genre_key VARCHAR(255) PRIMARY KEY,
    genre_name VARCHAR(255) NOT NULL
);

CREATE TABLE Album (
    album_key VARCHAR(255) PRIMARY KEY,
    album_id VARCHAR(255) UNIQUE NOT NULL,
    album_name VARCHAR(255) NOT NULL,
    no_of_songs INT
```

```
);

CREATE TABLE Songs (
    song_key VARCHAR(255) PRIMARY KEY,
    song_id VARCHAR(255) UNIQUE NOT NULL,
    song_name VARCHAR(255) NOT NULL,
    run_time INTERVAL,
    album_key VARCHAR(255) REFERENCES Album(album_key),
    genre_key VARCHAR(255) REFERENCES Genre(genre_key),
    artist_key VARCHAR(255) REFERENCES Artist(artist_key),
    users_key VARCHAR(255) REFERENCES Users(users_key)
);


CREATE TABLE Year (
    year_key VARCHAR(255) PRIMARY KEY,
    year INT NOT NULL
);

CREATE TABLE Month (
    month_key VARCHAR(255) PRIMARY KEY,
    month_name VARCHAR(255) NOT NULL,
    year_key VARCHAR(255) REFERENCES Year(year_key)
);

CREATE TABLE Date (
    day_key VARCHAR(255) PRIMARY KEY,
    day INT NOT NULL,
    month_key VARCHAR(255) REFERENCES Month(month_key)
);



CREATE TABLE Time (
    time_key VARCHAR(255) PRIMARY KEY,
    hour INT NOT NULL,
    day_key VARCHAR(255) REFERENCES Date(day_key)
);
```

```
CREATE TABLE Stream (
    stream_key VARCHAR(255) PRIMARY KEY,
    users_key VARCHAR(255) REFERENCES Users(users_key),
    song_key VARCHAR(255) REFERENCES Songs(song_key),
    duration INTERVAL,
    stream_score FLOAT,
    skip INT NOT NULL
);

CREATE TABLE Upload (
    upload_key VARCHAR(255) PRIMARY KEY,
    prod_key VARCHAR(255) REFERENCES Producer(prod_key),
    song_key VARCHAR(255) REFERENCES Songs(song_key),
    artist_key VARCHAR(255) REFERENCES Artist(artist_key),
);
```



OLAP Operations

1. No. of song streams from Los Angeles, CA

   Stream1 ← ROLLUP* (Stream, Users → Users, City → city_name, count(stream_amt))
   Result ← DICE (Stream1,city → 'Los Angles')

2. No. of Artists that make 'Rap' Music

   Upload1 ← ROLLUP* (Upload, Artist → Artist, Genre → genre_name, count(DISTINCT artist_id))
   Result ← DICE (Upload1, Genre → 'Rap")

3. Avg number of songs by album with the genre 'Indie'

   Upload1 ← ROLLUP* (Upload, Song → Album, Genre → genre_name, avg(no_of_songs))
   Result ← DICE (Upload1, Genre → 'Indie")

4. Avg skip rate for songs in albums with 3 or more songs

   Stream1 ← DICE(Stream, Album.no_of_songs >= 3)
   Result ← ROLLUP* (Stream1, Song → Album, avg(skip_rate))

5. Stream amount by country

   Result ← ROLLUP*(Stream, User→ Country, sum(stream_amt))

6. Top artist that has uploaded the most songs

   Upload1 ← ADDMEASURE(Upload, Rank = RANK(sum(upload_amt) DESC)
   OVER Artist)
   Upload2 ← ROLLUP*(Upload1, Artist → Artist, Rank)
   Result ← SLICE(Upload2, Rank = 1)

7. Average duration of songs that users that talent scouts from MA are interested in

   Stream1 ← DICE(Stream, Talent_Scout.State = 'Massachusetts')
   Result ← ROLLUP*(Stream1, Talent_Scout → State, avg(duration))

Primary Events

1. User streaming a song
2. Artist uploading a song

SCD Implementation

Implemented SCD Type 2 for address attribute in User by adding columns "from_date" and "to_date" to the table. These dates represent the validity of the instances.

Below, in row 40 it can be seen that a record was changed the same day it was input, creating a new record that is valid from that day and updating the "to_date" of the old record.

users_key	users_id	username	from_date	to_date	
28	30	cara_nelson	2025-03-30	[NULL]	CT8
29	31	dylan_carter	2025-03-30	[NULL]	CT8
30	32	eva_mitchell	2025-03-30	[NULL]	CT1
31	33	finn_perez	2025-03-30	[NULL]	CT1
32	34	gina_roberts	2025-03-30	[NULL]	CT1
33	35	hank_turner	2025-03-30	[NULL]	CT7
34	36	iris_phillips	2025-03-30	[NULL]	CT1
35	37	jake_campbell	2025-03-30	[NULL]	CT1
36	38	kara_parker	2025-03-30	[NULL]	CT4
37	39	liam_evans	2025-03-30	[NULL]	CT1
38	40	mona_rivera	2025-03-30	[NULL]	CT3
39	1	john_doe	2025-03-30	[NULL]	CT1
40	2	jane_smith	2025-03-30	2025-03-30	CT1
41	2	jane_smith	2025-03-30	[NULL]	CT1

Calculated Measure

Our calculated measure is stream_score that is within our "Stream" fact. The formula is found below.

Formula: Stream_score = (Stream duration / Song run_time)

The lower the stream duration, which is how long a user is streaming a song, the lower the score will be, as it indicates the fraction of the song the stream listened to.

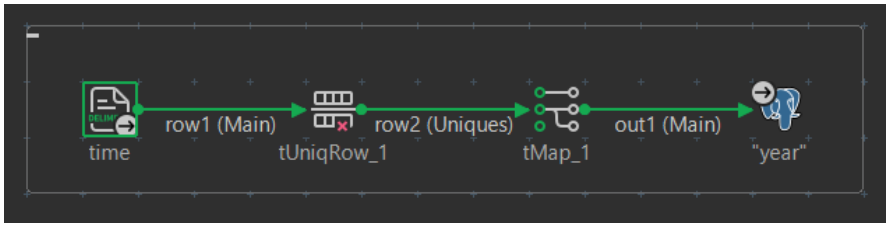
ETL Execution

Prepopulated Time dimension, extracting from csv: years 2022-2025

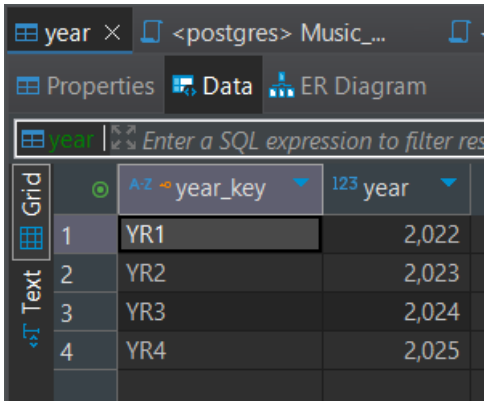
	A	B	C	D	E
1	year	month_nar	date	day	hour
2	2022	January	1/1/2022	1	0
3	2022	January	1/2/2022	2	0
4	2022	January	1/3/2022	3	0
5	2022	January	1/4/2022	4	0
6	2022	January	1/5/2022	5	0
7	2022	January	1/6/2022	6	0
8	2022	January	1/7/2022	7	0
9	2022	January	1/8/2022	8	0
10	2022	January	1/9/2022	9	0
11	2022	January	1/10/2022	10	0
12	2022	January	1/11/2022	11	0
13	2022	January	1/12/2022	12	0
14	2022	January	1/13/2022	13	0
15	2022	January	1/14/2022	14	0
16	2022	January	1/15/2022	15	0
17	2022	January	1/16/2022	16	0
18	2022	January	1/17/2022	17	0
19	2022	January	1/18/2022	18	0
20	2022	January	1/19/2022	19	0

1. Year

   Extracted unique values using tUniqRow and populated “year” table in warehouse

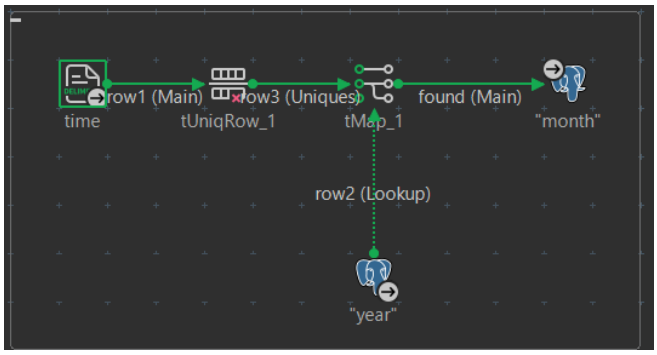


Result in warehouse: 4 rows



2. Month

Similarly, extracted unique month values using tUniqRow and populated “month” table in warehouse
Looked up year_key to populate as well



Result in warehouse: 48 rows

month | <postgres> Music_DDL_target | <postgres> Test_

Properties | Data | ER Diagram

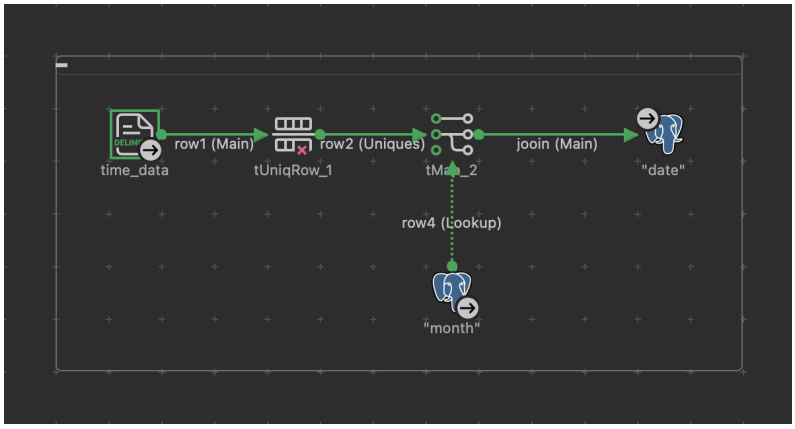
month | Enter a SQL expression to filter results (use Ctrl+Space)

Grid | Text

	month_key	month_name	year_key
1	MO1	January	YR1
2	MO2	February	YR1
3	MO3	March	YR1
4	MO4	April	YR1
5	MO5	May	YR1
6	MO6	June	YR1
7	MO7	July	YR1
8	MO8	August	YR1
9	MO9	September	YR1
10	MO10	October	YR1
11	MO11	November	YR1
12	MO12	December	YR1
13	MO13	January	YR2
14	MO14	February	YR2
15	MO15	March	YR2
16	MO16	April	YR2
17	MO17	May	YR2
18	MO18	June	YR2

3. Date

Similarly, extracted unique date values using tUniqRow and populated “date” table in warehouse

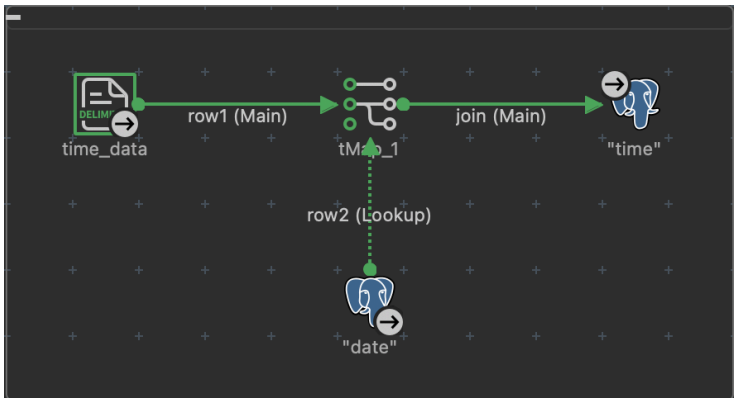


Result in warehouse: 1461 rows

	day_key	day	month_key
1	2022-01-01	1	MO37
2	2022-01-02	2	MO37
3	2022-01-03	3	MO37
4	2022-01-04	4	MO37
5	2022-01-05	5	MO37
6	2022-01-06	6	MO37
7	2022-01-07	7	MO37
8	2022-01-08	8	MO37
9	2022-01-09	9	MO37
10	2022-01-10	10	MO37
11	2022-01-11	11	MO37
12	2022-01-12	12	MO37
13	2022-01-13	13	MO37
14	2022-01-14	14	MO37
15	2022-01-15	15	MO37
16	2022-01-16	16	MO37

4. Time

   Similarly, extracted unique time values using tUniqRow and populated “time” table in warehouse



Result in warehouse: 10,000 rows

	time_key	hour	day_key
1	H1	0	2022-01-01
2	H2	0	2022-01-02
3	H3	0	2022-01-03
4	H4	0	2022-01-04
5	H5	0	2022-01-05
6	H6	0	2022-01-06
7	H7	0	2022-01-07
8	H8	0	2022-01-08
9	H9	0	2022-01-09
10	H10	0	2022-01-10
11	H11	0	2022-01-11
12	H12	0	2022-01-12
13	H13	0	2022-01-13
14	H14	0	2022-01-14
15	H15	0	2022-01-15
16	H16	0	2022-01-16

Prepopulated Geography dimension, extracting from csv

5. Country

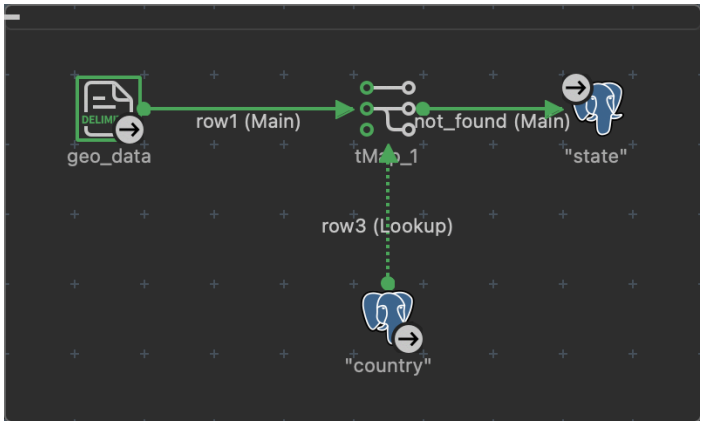
   Extracted unique country values using tAggregateRow and populated “country” table in warehouse

Result in warehouse: 27 rows

	country_key	country_name
1	CO1	United Arab Emirates
2	CO2	Australia
3	CO3	United States of America
4	CO4	Netherlands
5	CO5	New Zealand
6	CO6	India
7	CO7	Thailand
8	CO8	Spain
9	CO9	China
10	CO10	United Kingdom
11	CO11	Brazil
12	CO12	Germany
13	CO13	France
14	CO14	Argentina
15	CO15	Egypt
16	CO16	Canada

6. State

   Similarly, extracted unique state values using tUniqRow and populated “state” table in warehouse

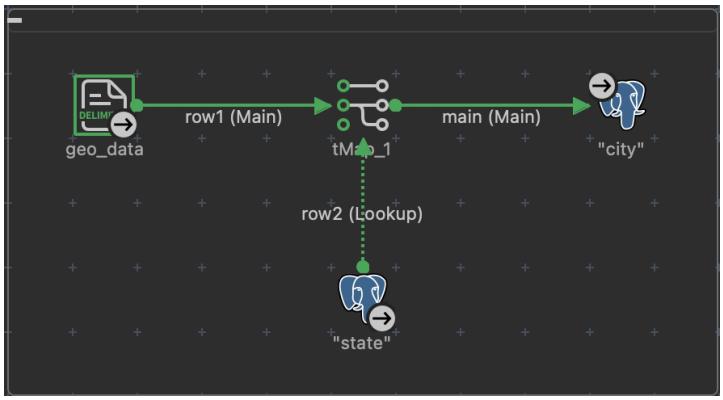


Result in warehouse: 169 rows

	state_key	state_name	country_key
1	ST1	Abu Dhabi	CO1
2	ST2	South Australia	CO2
3	ST3	Ajman	CO1
4	ST4	New York	CO3
5	ST5	North Holland	CO4
6	ST6	Georgia	CO3
7	ST7	Auckland	CO5
8	ST8	Texas	CO3
9	ST9	California	CO3
10	ST10	Karnataka	CO6
11	ST11	Bangkok	CO7
12	ST12	Catalonia	CO8
13	ST13	Beijing	CO9
14	ST14	Northern Ireland	CO10
15	ST15	Minas Gerais	CO11
16	ST16	Karnataka	CO6

7. City

Similarly, extracted unique city values using tUniqRow and populated “city” table in warehouse

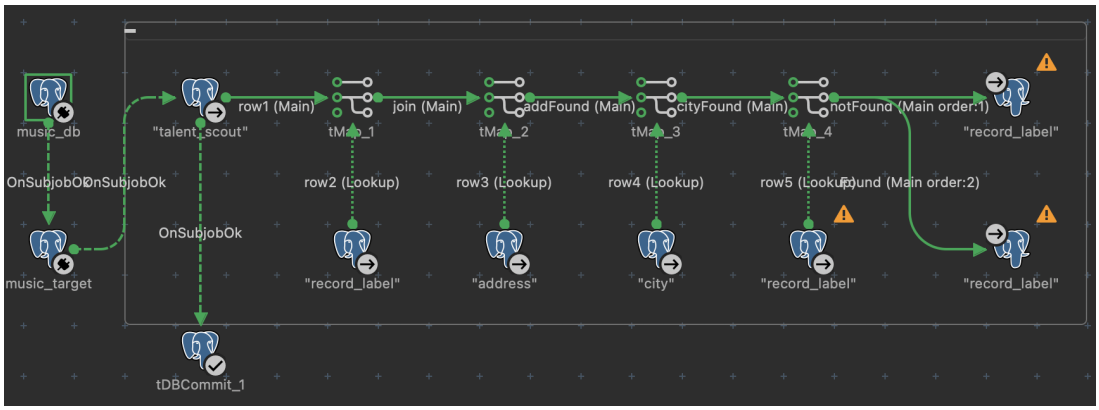


Result in warehouse: 169 rows

	city_key	city_name	state_key
1	CT1	Abu Dhabi	ST1
2	CT2	Adelaide	ST2
3	CT3	Ajman	ST3
4	CT4	Albany	ST130
5	CT5	Amsterdam	ST5
6	CT6	Atlanta	ST6
7	CT7	Auckland	ST7
8	CT8	Austin	ST135
9	CT9	Bakersfield	ST138
10	CT10	Bangalore	ST16
11	CT11	Bangkok	ST11
12	CT12	Barcelona	ST12
13	CT13	Beijing	ST13
14	CT14	Belfast	ST14
15	CT15	Belo Horizonte	ST15
16	CT16	Bengaluru	ST16

8. Record Label

To populate the “record_label” table in the warehouse, attributes are required from several tables in the oltp database such as “record_label”, “talent_scout”, and “address”. To do so, tmap components are used to join tables, extract desired attributes, and perform look-ups. Foreign keys were connected in this way as well.

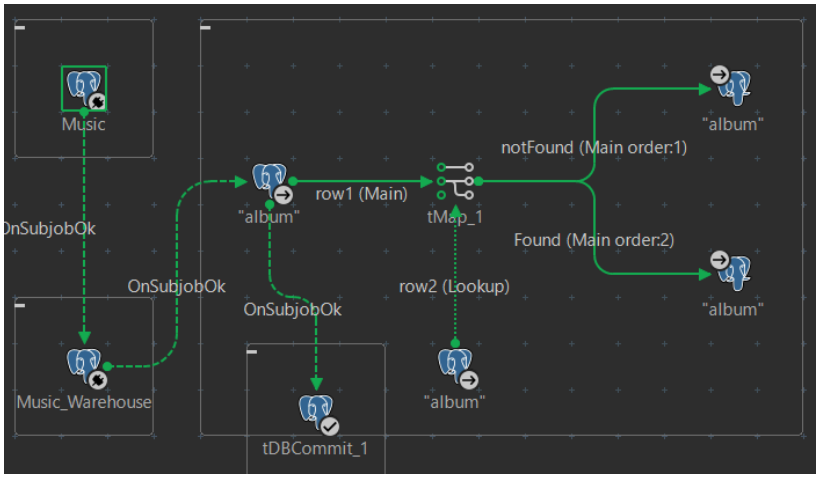


Result in warehouse: 40 rows

rl_key	rl_id	rl_name	talentscout_name	city_key
RL1	1	Universal Music Group	John Smith	CT84
RL2	2	Sony Music Entertainment	Emily Johnson	CT111
RL3	3	Warner Music Group	Michael Brown	CT72
RL4	4	EMI Records	Jessica Davis	CT158
RL5	5	Atlantic Records	David Wilson	CT164
RL6	6	Columbia Records	Sarah Martinez	CT83
RL7	7	Island Records	Daniel Garcia	CT102
RL8	8	Def Jam Recordings	Laura Anderson	CT117
RL9	9	Republic Records	James Thomas	CT95
RL10	10	Capitol Records	Sophia Lee	CT12
RL11	11	RCA Records	Robert Harris	CT153
RL12	12	Virgin Records	Olivia Clark	CT91
RL13	13	Arista Records	Matthew Lewis	CT157
RL14	14	Interscope Records	Emma Walker	CT13
RL15	15	Motown Records	Joshua Hall	CT101
RL16	17	Polydor Records	Andrew Young	CT93

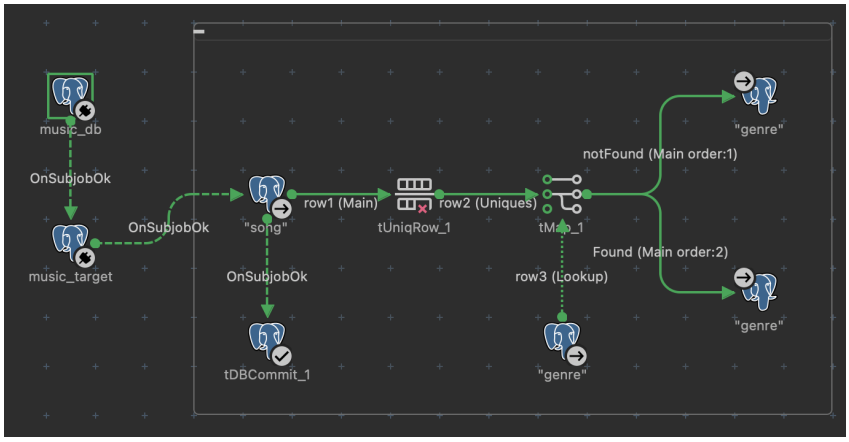
9. Album

To populate the “album” table in the warehouse, attributes are needed only from the “album” table in the oltp database. A tMap component is used to extract the values and to perform a lookup.



10. Genre

    To populate the "genre" table in the warehouse, attributes are needed only from the "song" table in the oltp database. A tUniqRow component is used to extract the unique values and a tMap is used to perform a lookup.

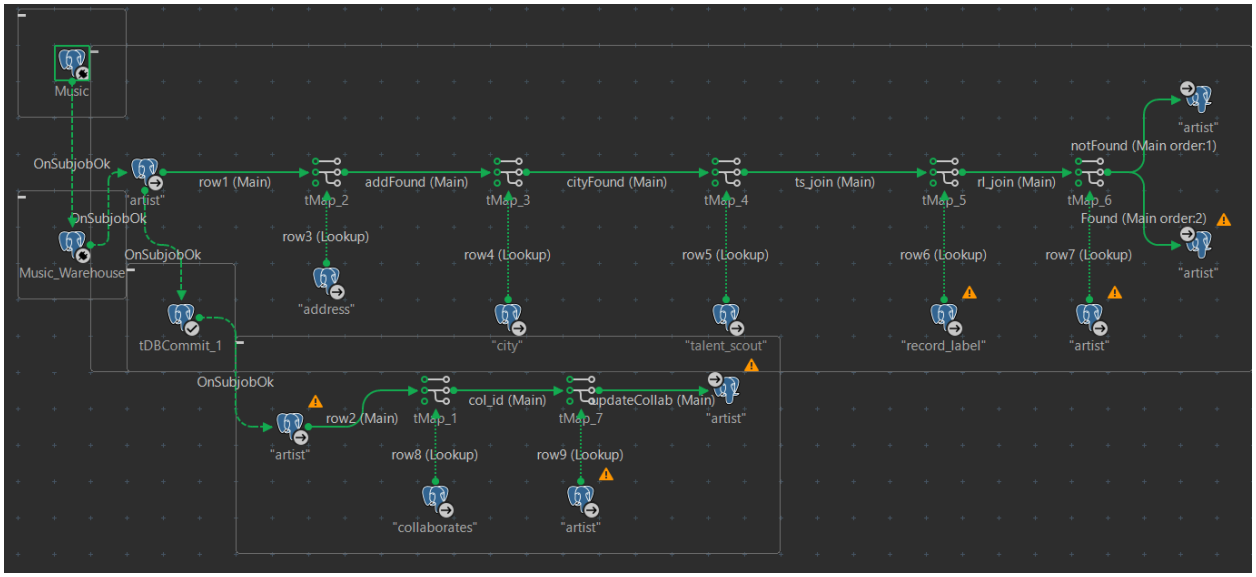


Result in warehouse: 12 rows

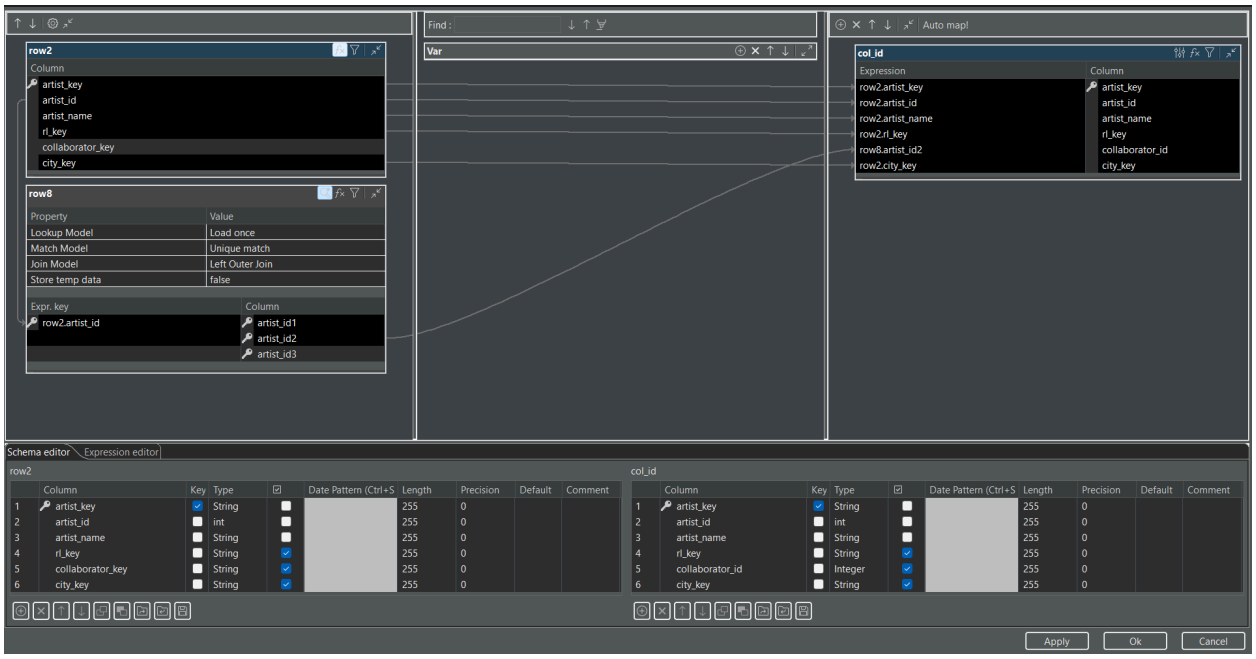
genre_key	genre_name
GE1	Pop
GE2	Rock
GE3	Electronic
GE4	Ambient
GE5	EDM
GE6	Jazz
GE7	Hip-Hop
GE8	Country
GE9	R&B
GE10	New Age
GE11	World
GE12	Classical

11. Artist

To populate the "artist" table in the warehouse, attributes are required from the "artist" table in the oltp database. Several foreign keys are also present. Thus, tMap components are used to join tables, extract desired attributes and foreign-keys, and perform look-ups. A sub-job is created to populate the null-allowed "collaborator_key" foreign key as it references the "artist" table recursively.



Below, the setup for the tMap component adding the foreign key "collaborator_key" is shown.

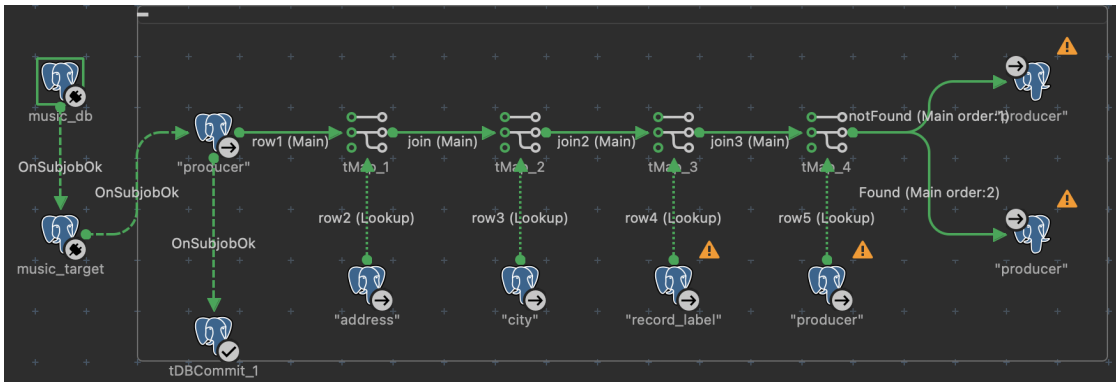


Result in warehouse: 40 rows

artist_key	artist_id	artist_name	rl_key	collaborator_key
AR35	37	Jake Campbell	RL36	[NULL]
AR36	38	Kara Parker	RL37	[NULL]
AR37	39	Liam Evans	RL38	[NULL]
AR38	40	Mona Rivera	RL39	[NULL]
AR1	1	John Doe	RL1	AR13
AR2	2	Jane Smith	RL2	AR23
AR3	3	Alice Johnson	RL3	AR20
AR4	4	Bob Brown	RL4	AR16
AR5	5	Charlie Davis	RL5	AR17
AR6	6	Eve White	RL6	AR21
AR7	7	Frank Wilson	RL7	AR14
AR8	8	Grace Lee	RL8	AR16
AR10	10	Ivy Thompson	RL10	AR24
AR11	11	Jack Anderson	RL11	AR23
AR15	15	Noah Robinson	RL15	AR25
AR19	20	Steve Hall	RL19	AR27

12. Producer

To populate the “producer” table in the warehouse, attributes are required from the “producer” table in the oltp database. Foreign keys “city_key” and “rl_key” are required as well. To do so, tMap components are used to join tables, extract desired attributes, and perform look-ups. Foreign keys were connected in this way as well.

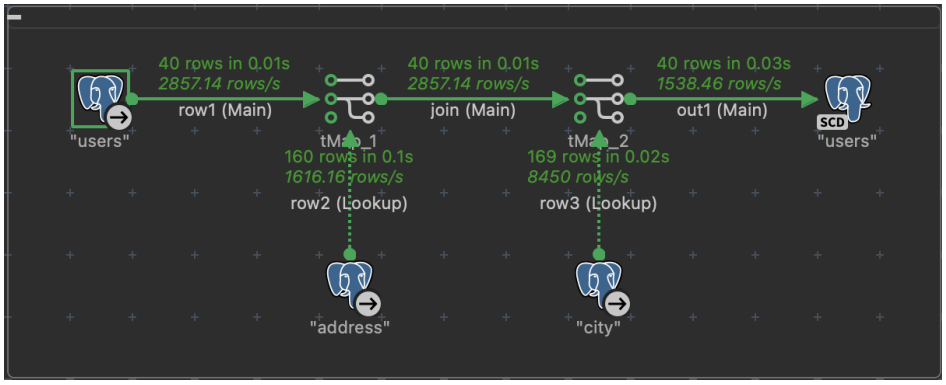


Result in warehouse: 40 rows

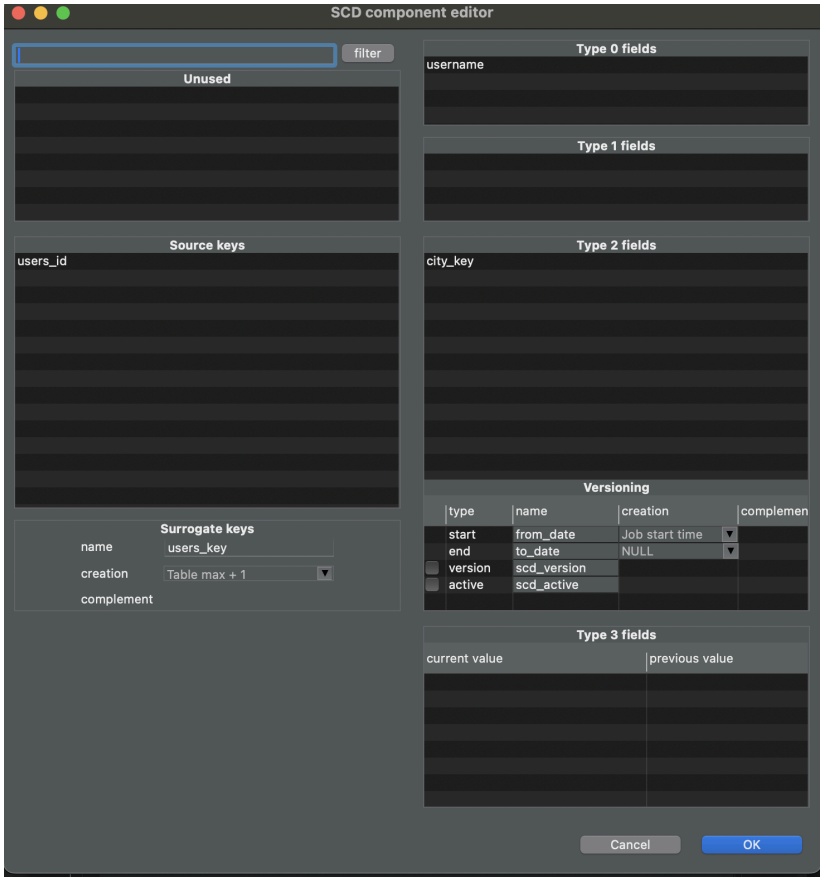
prod_key	prod_id	prod_name	rl_key	city_key
PR1	1	Rick Rubin	RL1	CT22
PR2	2	Max Martin	RL2	CT29
PR3	3	Quincy Jones	RL3	CT32
PR4	4	Dr. Dre	RL4	CT67
PR5	5	George Martin	RL5	CT21
PR6	6	Pharrell Williams	RL6	CT55
PR7	7	Brian Eno	RL7	CT162
PR8	8	Timbaland	RL8	CT2
PR9	9	Mark Ronson	RL9	CT167
PR10	10	Nigel Godrich	RL10	CT142
PR11	11	Butch Vig	RL11	CT65
PR12	12	Linda Perry	RL12	CT75
PR13	13	Danger Mouse	RL13	CT134
PR14	14	Jeff Lynne	RL14	CT98
PR15	15	Steve Albini	RL15	CT50
PR16	17	Mike Will Made It	RL16	CT149

13. User

To populate the “users” table in the warehouse, attributes are required from the “users” table in the oltp database. Foreign key “city_key” is required as well. To do so, tMap components are used to extract required columns. SCD type 2 was implemented in this table to allow the user changing their address by way of the tDBSCD component.



Below the configuration of the tDBSCD component can be found.

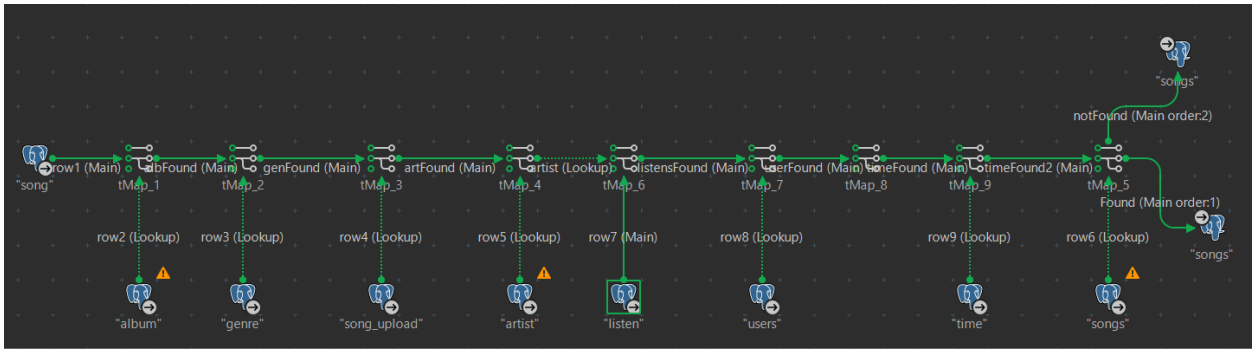


Result in warehouse: 41 rows

users_key	users_id	username	from_date	to_date	
28	30	cara_nelson	2025-03-30	[NULL]	CT8
29	31	dylan_carter	2025-03-30	[NULL]	CT8
30	32	eva_mitchell	2025-03-30	[NULL]	CT1
31	33	finn_perez	2025-03-30	[NULL]	CT1
32	34	gina_roberts	2025-03-30	[NULL]	CT1
33	35	hank_turner	2025-03-30	[NULL]	CT7
34	36	iris_phillips	2025-03-30	[NULL]	CT1
35	37	jake_campbell	2025-03-30	[NULL]	CT1
36	38	kara_parker	2025-03-30	[NULL]	CT4
37	39	liam_evans	2025-03-30	[NULL]	CT1
38	40	mona_rivera	2025-03-30	[NULL]	CT3
39	1	john_doe	2025-03-30	[NULL]	CT1
40	2	jane_smith	2025-03-30	2025-03-30	CT1
41	2	jane_smith	2025-03-30	[NULL]	CT1

14. Song

To populate the “songs” table in the warehouse, attributes are required from the “song” table in the oltp database. Foreign keys “album_key”, “genre_key”, “artist_key”, “users_key”, and “time_key” are required as well. To do so, tMap components are used to join tables, extract desired attributes, and perform look-ups. Foreign keys were connected in this way as well.

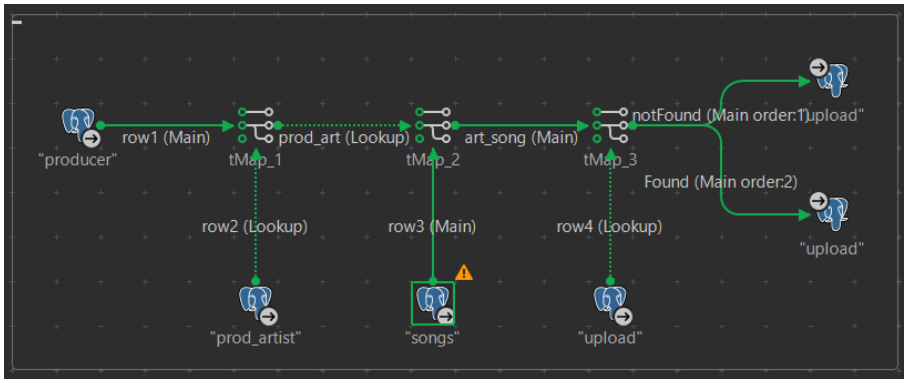


Result in warehouse: 200 rows

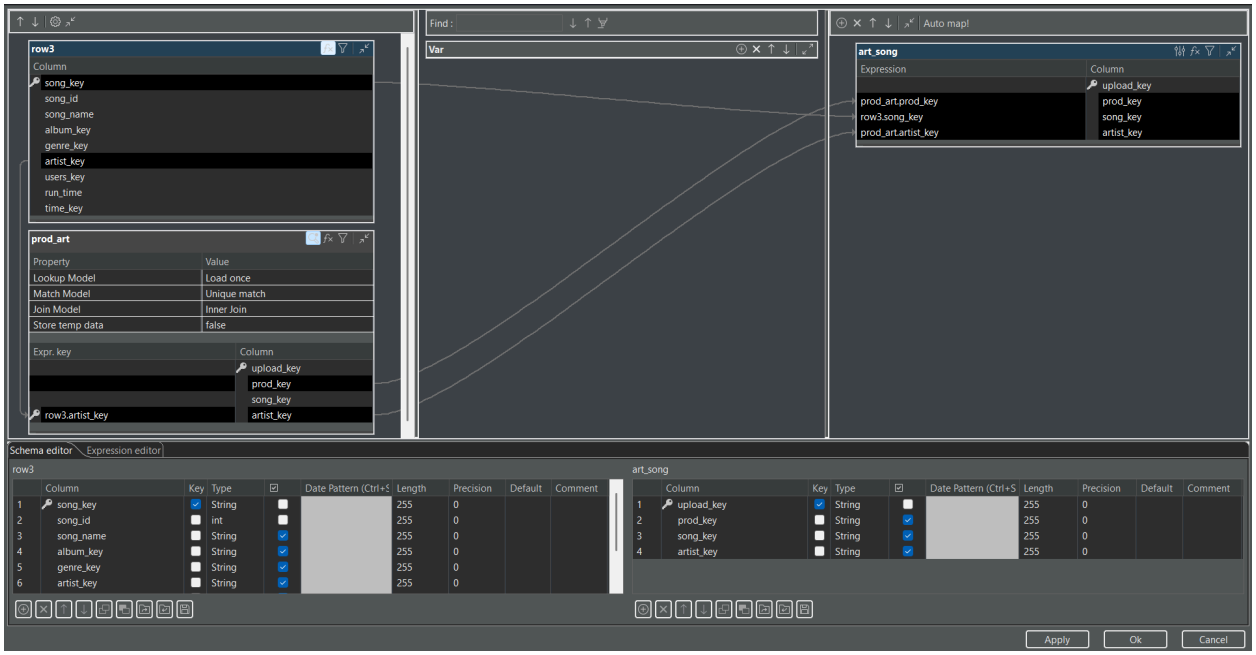
song_key	song_id	song_name	album_key	genre_key	artist_key	users_key	run_time	time_key
SO1	28	Aether	A28	GE4	AR27	39	263	H8860
SO2	2	Echoes	A2	GE2	AR2	12	261	H9015
SO3	28	Aether	A28	GE4	AR27	15	263	H5927
SO4	59	Polaris Rising	A19	GE1	AR18	30	287	H8466
SO5	64	Xanadu Dreams	A24	GE10	AR23	22	275	H5722
SO6	67	Aetherial Waves	A27	GE4	AR26	25	284	H9682
SO7	3	Golden Days	A3	GE1	AR3	3	331	H16080
SO8	66	Zenith Sky	A26	GE2	AR25	34	241	H11388
SO9	13	Crimson Dawn	A13	GE2	AR13	20	193	H64
SO10	6	Electric Love	A6	GE5	AR6	9	155	H14578
SO11	63	Whispering Winds	A23	GE6	AR22	21	180	H11035
SO12	19	Lunar Eclipse	A19	GE2	AR18	17	221	H6727
SO13	61	Twilight Shadows	A21	GE3	AR20	27	224	H577
SO14	72	Fading Stars	A32	GE1	AR31	28	237	H5444
SO15	72	Fading Stars	A32	GE1	AR31	1	237	H12579
SO16	48	Moonlight Sonata	A8	GE12	AR8	19	200	H17480
SO17	20	Polaris	A20	GE1	AR19	29	330	H15111

15. Upload

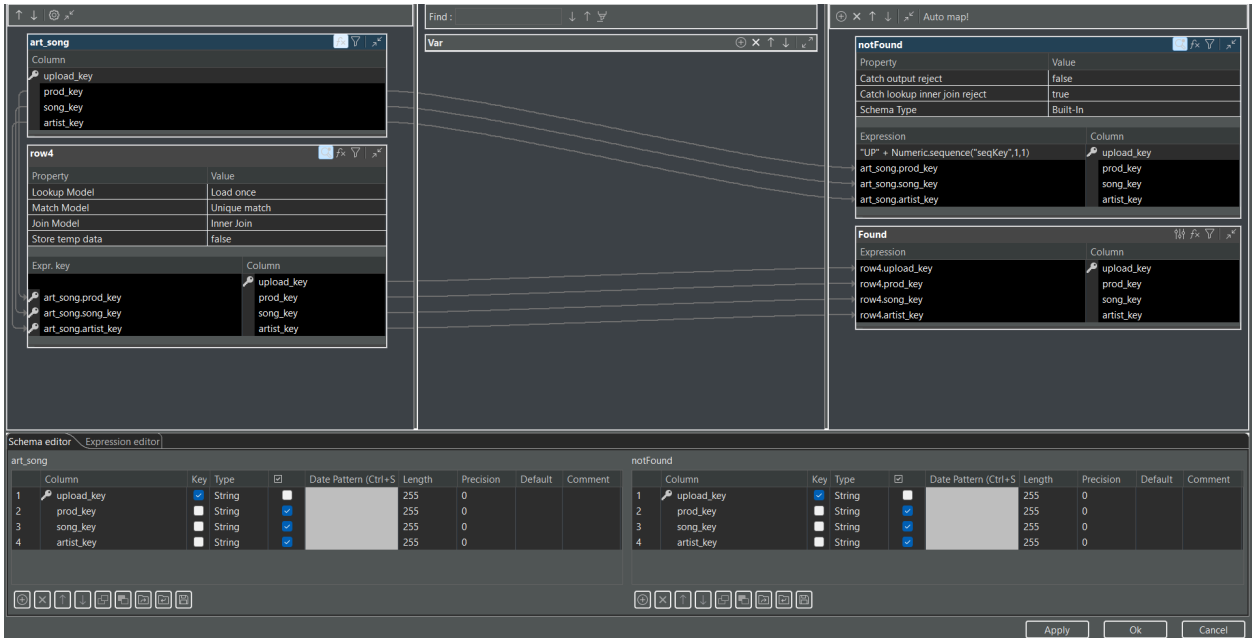
To populate the “upload” fact table, foreign keys “prod_key”, “song_key”, and “artist_key” are needed which are retrieved from the warehouse tables “producer”, “prod_artist”, and “songs”. From there a lookup is done and where a particular row is not found, an “upload_key” is generated.



Below is the configuration of the second tMap:



Below is the configuration of the last tMap:



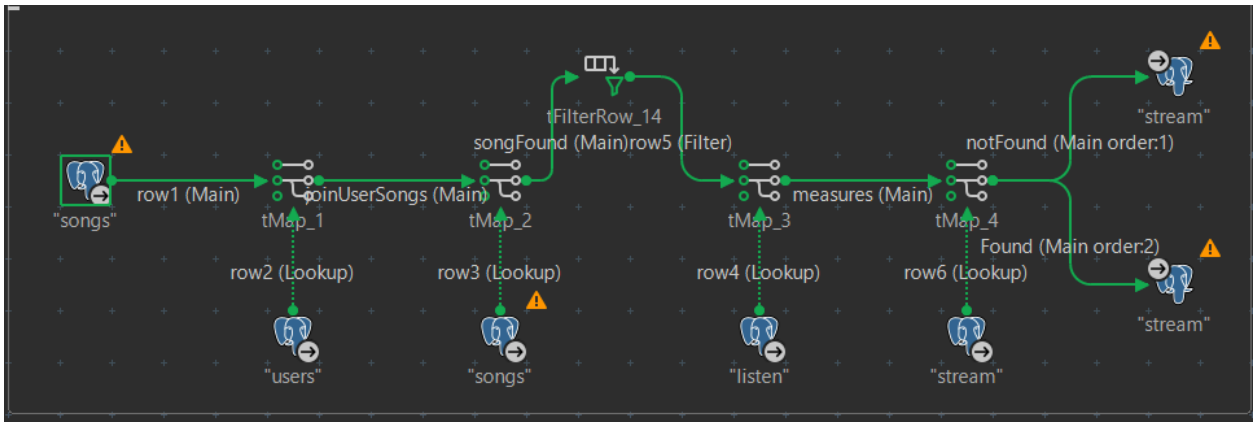
Result in warehouse: 202 rows

upload_key	prod_key	song_key	artist_key
UP1	PR17	SO1	AR27
UP2	PR4	SO2	AR2
UP3	PR17	SO3	AR27
UP4	PR13	SO4	AR18
UP5	PR30	SO5	AR23
UP6	PR29	SO6	AR26
UP7	PR32	SO7	AR3
UP8	PR8	SO8	AR25
UP9	PR20	SO9	AR13
UP10	PR1	SO10	AR6
UP11	PR24	SO11	AR22
UP12	PR13	SO12	AR18
UP13	PR21	SO13	AR20
UP14	PR25	SO14	AR31
UP15	PR25	SO15	AR31
UP16	PR33	SO16	AR8

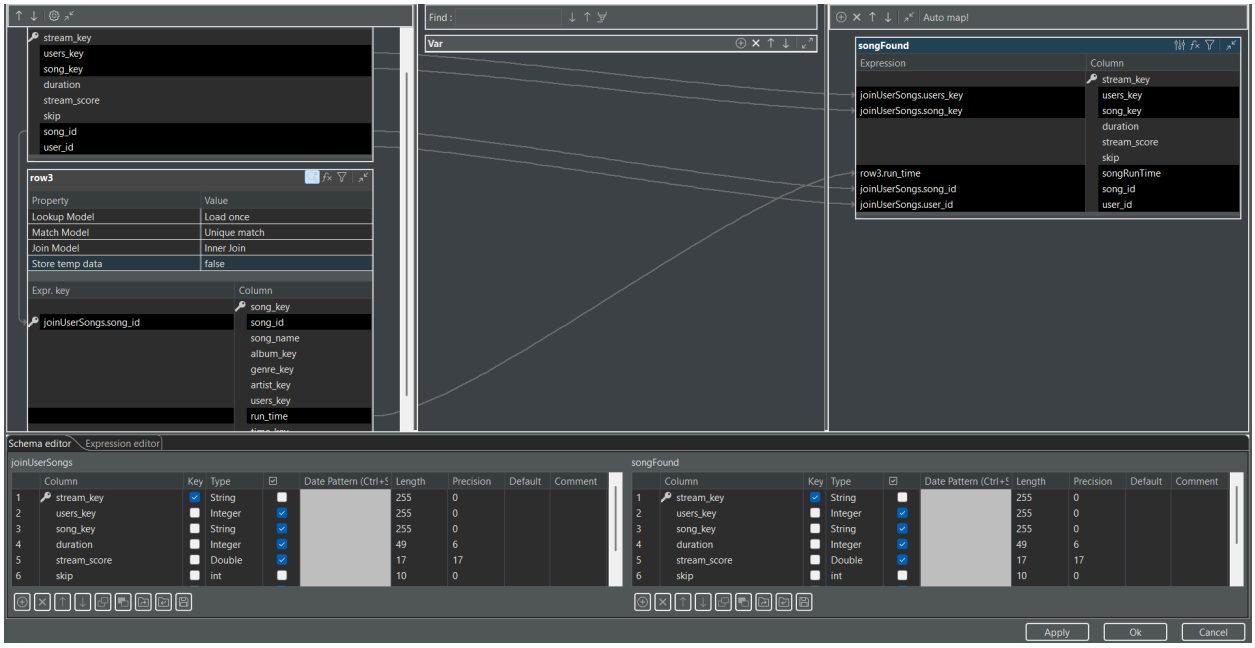
16. Stream

To populate the “stream” fact table, foreign keys “users_key” and “song_key” are needed which are retrieved from the warehouse tables “users” and “songs”. A tFilterRow component is used to filter some null values from the result. From there, the measures “stream_score”, “skip”, and “duration” need to be populated. This information comes from the oltp database table “listen”. Skip and duration are attributes in this table, but

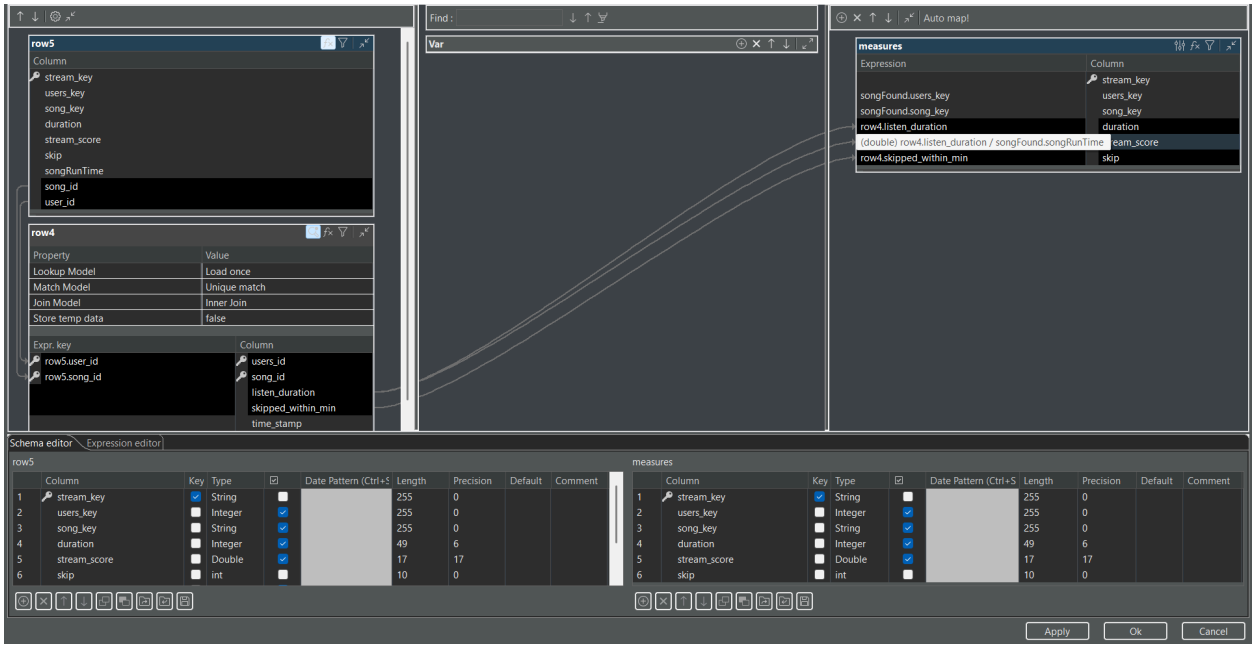
stream_score is a calculated measure that is calculated within the tMap component. Then, a lookup is done and where a particular row is not found, an “stream_key” is generated.



Below is the tMap configuration for the songs lookup, getting the run_time attribute to later create the calculated measure:



Below is the tMap configuration for extracting the measures for the warehouse:



Result in warehouse: 219 rows

stream_key	users_key	song_key	stream_score	skip	duration
ST1	39	SO1	0.825095057	0	217
ST2	12	SO2	0.877394636	0	229
ST3	15	SO3	0.2205323194	0	58
ST4	30	SO4	0.3902439024	0	112
ST5	22	SO5	0.3563636364	1	98
ST6	25	SO6	0.2112676056	1	60
ST7	3	SO7	0.4320241692	1	143
ST8	34	SO8	0.5767634855	1	139
ST9	20	SO9	0.8186528497	1	158
ST10	9	SO10	0.2451612903	0	38
ST11	21	SO11	0.55	0	99
ST12	17	SO12	0.6380090498	0	141
ST13	27	SO13	0.9553571429	1	214
ST14	28	SO14	0.8860759494	1	210
ST15	1	SO15	0.7594936709	1	180
ST16	19	SO16	0.045	0	9
ST17	29	SO17	0.0787878788	0	26
ST18	25	SO18	0.8242074928	0	286
ST19	20	SO20	0.2258064516	1	35
ST20	40	SO21	0.4488636364	1	79
ST21	29	SO22	0.0212121212	1	7
ST22	30	SO23	0.8020477816	1	235