EAS 550 Project: Music Inventory System

Phase 2 Report: Advanced Analytical Queries and Indexing Report

Team: Halle Bryant, Arun Ghontale, Phalguni Raparla

Introduction

For Phase 2 of the Music Inventory System implementation, we crafted four analytical queries with high informativity and use value for our target audience. These queries focus on genre fingerprinting (or returning information about the typical audio characteristics of each genre), top artists yearly, hidden gems (measured by above average popularity despite normal or low listen counts), and artist profiling. Each of these queries includes multi-table joins and aggregations at minimum. (See Appendix.)

Indexing Justification

To facilitate faster joins, we have created indices for all foreign keys used in the `JOIN` arguments. This is an important step, as SQL automatically indexes along primary keys but does not transfer these indexes to tables referencing the primary keys. For example, `artist_id` is indexed in the Artists table but artist_id is not indexed in the Tracks table. Without adding the additional index on the Tracks table, SQL must perform a full table scan in order to match on `artist_id`. Focusing on the features needed for the queries we have constructed for Phase 2, recall the original entities. (Primary keys are underlined.)

- **Albums:** `album_id`, `album_title`, `artist_id`, `album_type`, `album_favorites`, `album_tracks`, `album_date_released`, `album_listens`
- **Artists:** `artist_id`, `artist_name`, `artist_active_year_begin`, `artists_favorites`, `artist_handle`, `artist_website`
- **Genres:** `genre_id`, `parent_genre_id`, `genre_name`
- **Labels:** `label_id`, `label_name`
- **Tracks**: `track_id`, `album_id`, `track_title`, `track_language_code`, `track_listens`, `track_url`, `license_id`, `track_date_recorded`, `track_bit_rate`, `track_duration`, `track_explicit`, `track_favorites`
    - **Audio:** `track_id`, `acousticness`, `danceability`, `energy`, `instrumentalness`, `liveness`, `speechiness`, `tempo`, `valence`
    - **Social:** `track_id`, `artist_discovery`, `artist_familiarity`, `artist_hotttnesss`, `song_currency`, `song_hotttnesss`

- **ArtistLabels**: artist_id, label_id
- **TrackGenres:** track_id, genre_id

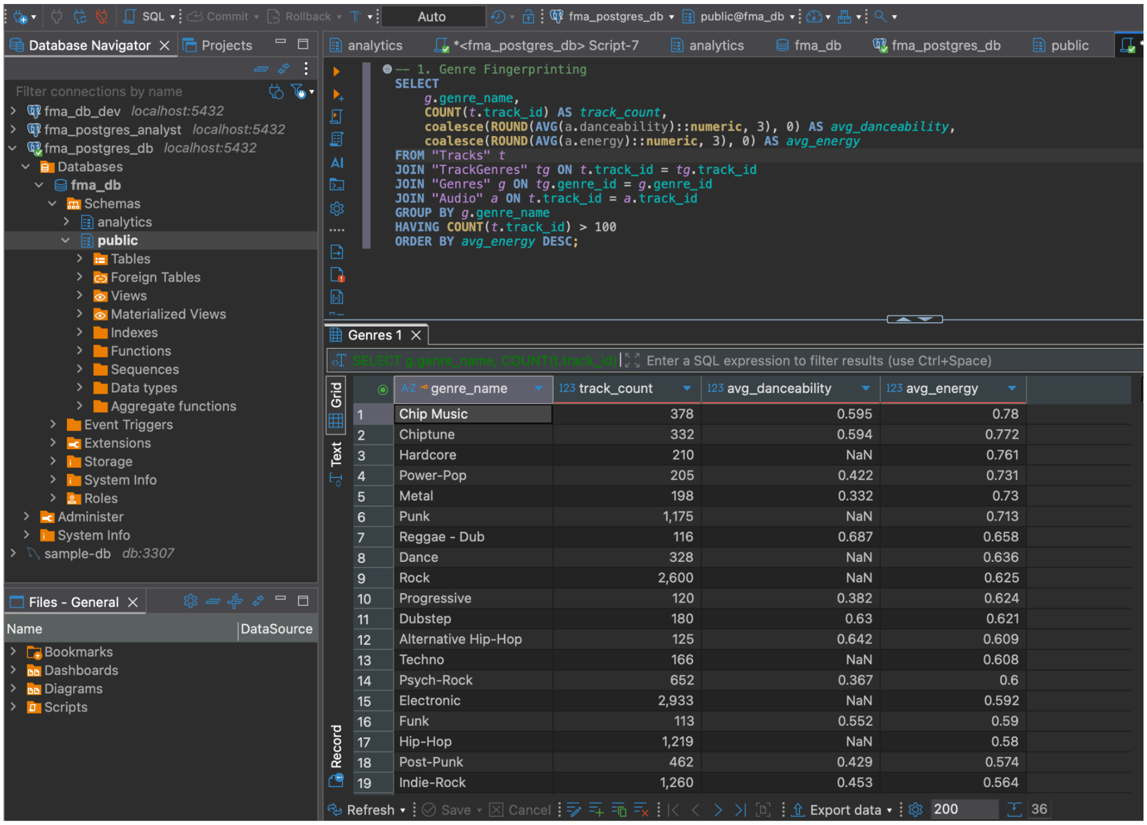
To enable more efficient joins between these tables, the database has been updated to include the following indexes:

- `idx_trackgenres_genre_id ON "TrackGenres"(genre_id)`
- `idx_trackgenres_track_id ON "TrackGenres"(track_id)`
- `idx_audio_track_id ON "Audio"(track_id)`
- `idx_tracks_date_recorded ON "Tracks"(track_date_recorded)`
- `idx_social_hotttnesss ON "Social"(song_hotttnesss)`
- `idx_artistlabels_label_id ON "ArtistLabels"(label_id)`

Implementing these indices significantly improves the execution times of the four analytical test queries, which can be accessed in the Appendix to this report.

Figure 1. Query results

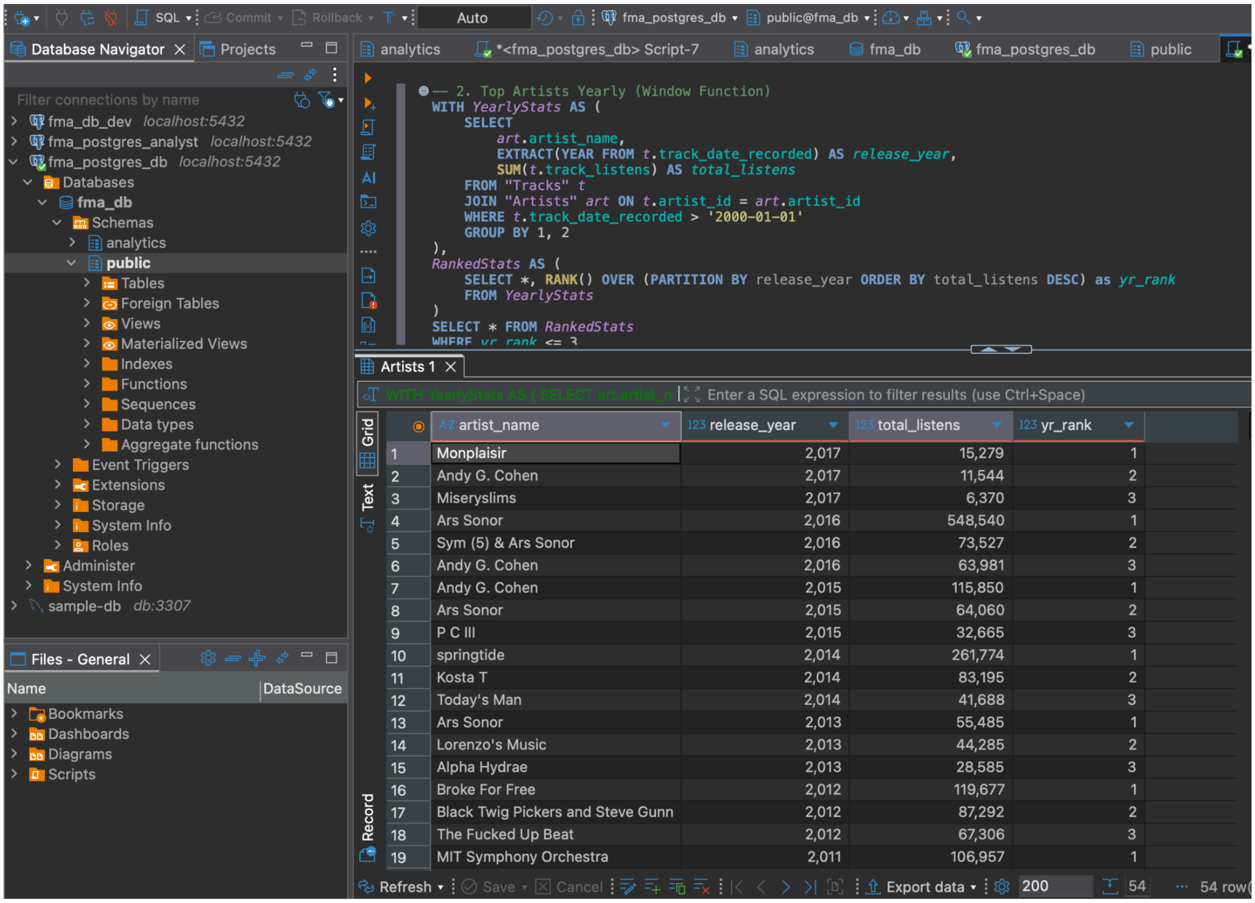
1- Genre Fingerprinting



```sql
-- 1. Genre Fingerprinting
SELECT
    g.genre_name,
    COUNT(t.track_id) AS track_count,
    coalesce(ROUND(AVG(a.danceability)::numeric, 3), 0) AS avg_danceability,
    coalesce(ROUND(AVG(a.energy)::numeric, 3), 0) AS avg_energy
FROM "Tracks" t
JOIN "TrackGenres" tg ON t.track_id = tg.track_id
JOIN "Genres" g ON tg.genre_id = g.genre_id
JOIN "Audio" a ON t.track_id = a.track_id
GROUP BY g.genre_name
HAVING COUNT(t.track_id) > 100
ORDER BY avg_energy DESC;
```

	genre_name	track_count	avg_danceability	avg_energy
1	Chip Music	378	0.595	0.78
2	Chiptune	332	0.594	0.772
3	Hardcore	210	NaN	0.761
4	Power-Pop	205	0.422	0.731
5	Metal	198	0.332	0.73
6	Punk	1,175	NaN	0.713
7	Reggae - Dub	116	0.687	0.658
8	Dance	328	NaN	0.636
9	Rock	2,600	NaN	0.625
10	Progressive	120	0.382	0.624
11	Dubstep	180	0.63	0.621
12	Alternative Hip-Hop	125	0.642	0.609
13	Techno	166	NaN	0.608
14	Psych-Rock	652	0.367	0.6
15	Electronic	2,933	NaN	0.592
16	Funk	113	0.552	0.59
17	Hip-Hop	1,219	NaN	0.58
18	Post-Punk	462	0.429	0.574
19	Indie-Rock	1,260	0.453	0.564

2- Top Artists Yearly

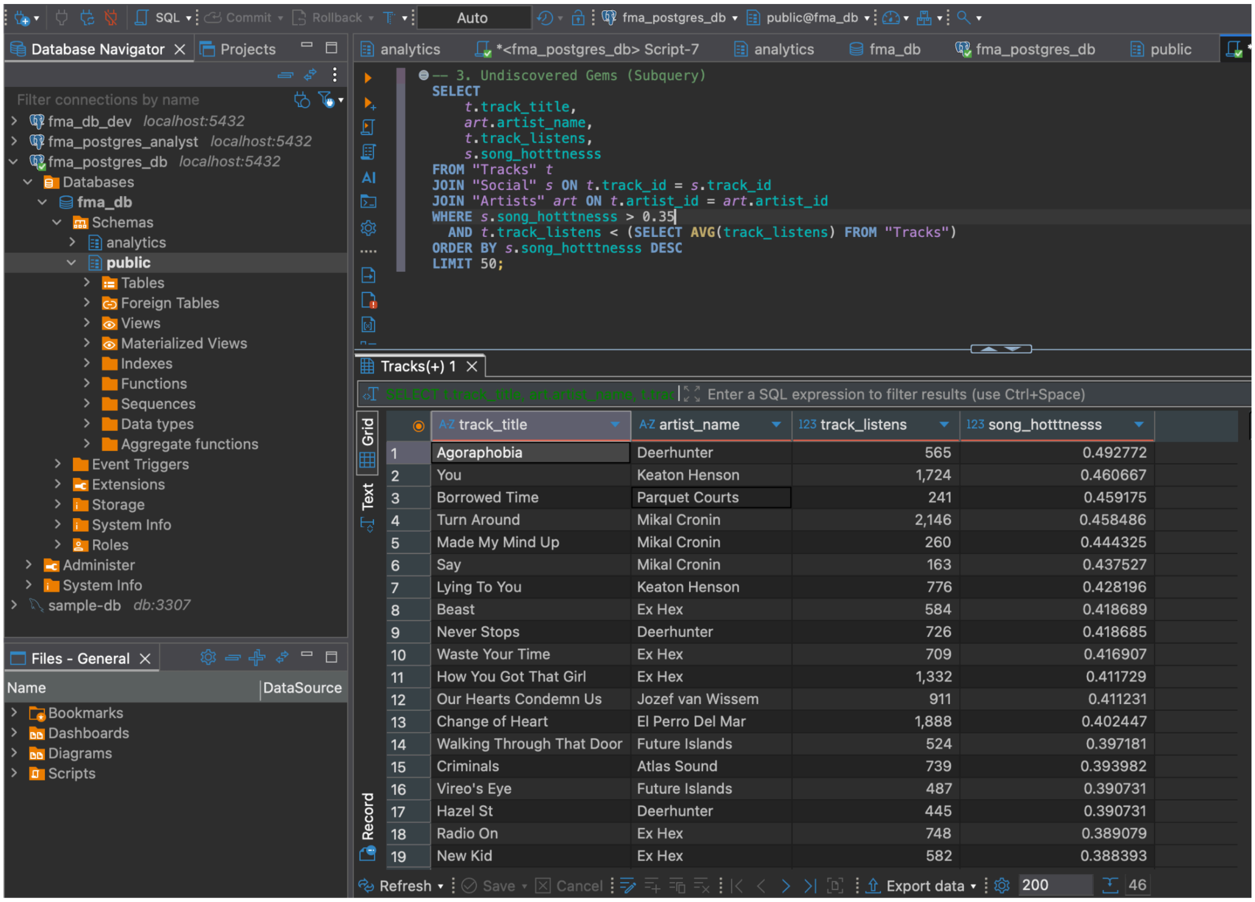


```sql
-- 2. Top Artists Yearly (Window Function)
WITH YearlyStats AS (
    SELECT
        art.artist_name,
        EXTRACT(YEAR FROM t.track_date_recorded) AS release_year,
        SUM(t.track_listens) AS total_listens
    FROM "Tracks" t
    JOIN "Artists" art ON t.artist_id = art.artist_id
    WHERE t.track_date_recorded > '2000-01-01'
    GROUP BY 1, 2
),
RankedStats AS (
    SELECT *, RANK() OVER (PARTITION BY release_year ORDER BY total_listens DESC) as yr_rank
    FROM YearlyStats
)
SELECT * FROM RankedStats
WHERE yr_rank <= 3
```

WITH YearlyStats AS (SELECT art.artist_n | Enter a SQL expression to filter results (use Ctrl+Space)

	artist_name	release_year	total_listens	yr_rank
1	Monplaisir	2,017	15,279	1
2	Andy G. Cohen	2,017	11,544	2
3	Miseryslims	2,017	6,370	3
4	Ars Sonor	2,016	548,540	1
5	Sym (5) & Ars Sonor	2,016	73,527	2
6	Andy G. Cohen	2,016	63,981	3
7	Andy G. Cohen	2,015	115,850	1
8	Ars Sonor	2,015	64,060	2
9	P C III	2,015	32,665	3
10	springtide	2,014	261,774	1
11	Kosta T	2,014	83,195	2
12	Today's Man	2,014	41,688	3
13	Ars Sonor	2,013	55,485	1
14	Lorenzo's Music	2,013	44,285	2
15	Alpha Hydrae	2,013	28,585	3
16	Broke For Free	2,012	119,677	1
17	Black Twig Pickers and Steve Gunn	2,012	87,292	2
18	The Fucked Up Beat	2,012	67,306	3
19	MIT Symphony Orchestra	2,011	106,957	1

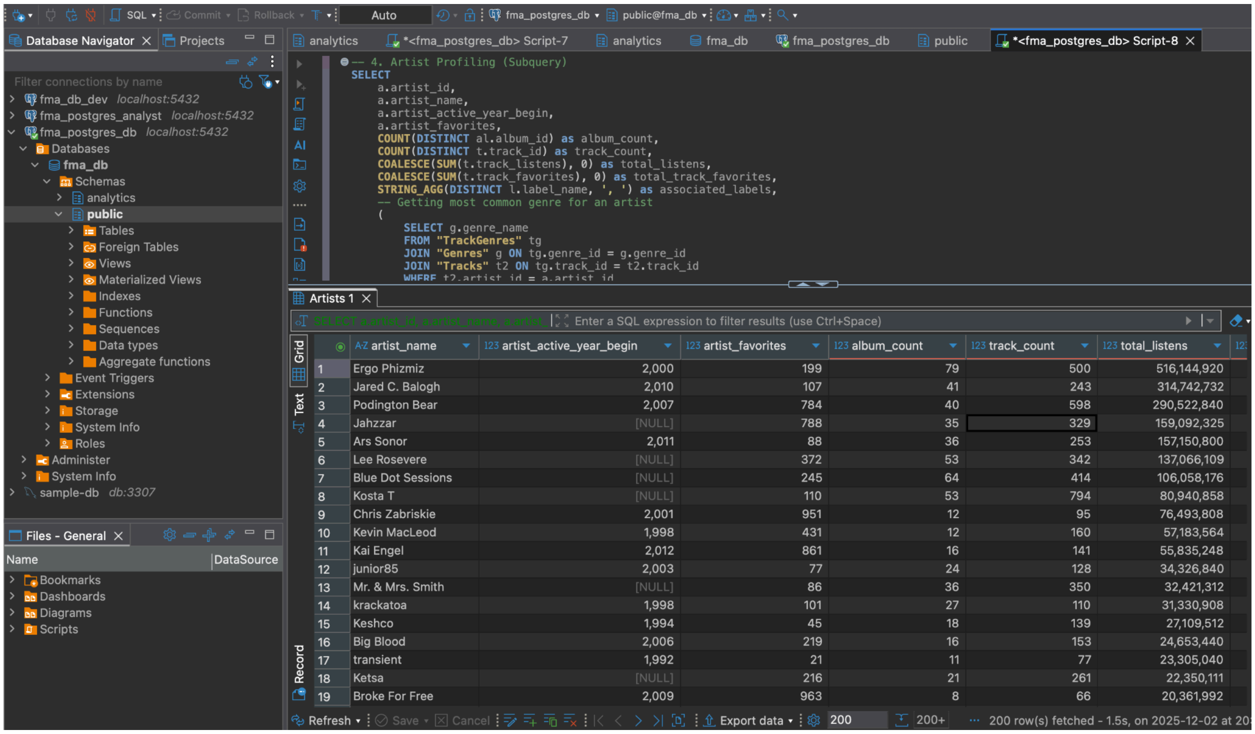
3- Undiscovered Gems



```sql
-- 3. Undiscovered Gems (Subquery)
SELECT
    t.track_title,
    art.artist_name,
    t.track_listens,
    s.song_hotttnesss
FROM "Tracks" t
JOIN "Social" s ON t.track_id = s.track_id
JOIN "Artists" art ON t.artist_id = art.artist_id
WHERE s.song_hotttnesss > 0.35
  AND t.track_listens < (SELECT AVG(track_listens) FROM "Tracks")
ORDER BY s.song_hotttnesss DESC
LIMIT 50;
```

	track_title	artist_name	track_listens	song_hotttnesss
1	Agoraphobia	Deerhunter	565	0.492772
2	You	Keaton Henson	1,724	0.460667
3	Borrowed Time	Parquet Courts	241	0.459175
4	Turn Around	Mikal Cronin	2,146	0.458486
5	Made My Mind Up	Mikal Cronin	260	0.444325
6	Say	Mikal Cronin	163	0.437527
7	Lying To You	Keaton Henson	776	0.428196
8	Beast	Ex Hex	584	0.418689
9	Never Stops	Deerhunter	726	0.418685
10	Waste Your Time	Ex Hex	709	0.416907
11	How You Got That Girl	Ex Hex	1,332	0.411729
12	Our Hearts Condemn Us	Jozef van Wissem	911	0.411231
13	Change of Heart	El Perro Del Mar	1,888	0.402447
14	Walking Through That Door	Future Islands	524	0.397181
15	Criminals	Atlas Sound	739	0.393982
16	Vireo's Eye	Future Islands	487	0.390731
17	Hazel St	Deerhunter	445	0.390731
18	Radio On	Ex Hex	748	0.389079
19	New Kid	Ex Hex	582	0.388393

4- Artist Profiling



```sql
-- 4. Artist Profiling (Subquery)
SELECT
    a.artist_id,
    a.artist_name,
    a.artist_active_year_begin,
    a.artist_favorites,
    COUNT(DISTINCT al.album_id) as album_count,
    COUNT(DISTINCT t.track_id) as track_count,
    COALESCE(SUM(t.track_listens), 0) as total_listens,
    COALESCE(SUM(t.track_favorites), 0) as total_track_favorites,
    STRING_AGG(DISTINCT l.label_name, ', ') as associated_labels,
    -- Getting most common genre for an artist
    (
        SELECT g.genre_name
        FROM "TrackGenres" tg
        JOIN "Genres" g ON tg.genre_id = g.genre_id
        JOIN "Tracks" t2 ON tg.track_id = t2.track_id
        WHERE t2.artist_id = a.artist_id
```

SELECT a.artist_id, a.artist_name, a.artist_ | Enter a SQL expression to filter results (use Ctrl+Space)

	artist_name	artist_active_year_begin	artist_favorites	album_count	track_count	total_listens
1	Ergo Phizmiz	2,000	199	79	500	516,144,920
2	Jared C. Balogh	2,010	107	41	243	314,742,732
3	Podington Bear	2,007	784	40	598	290,522,840
4	Jahzzar	[NULL]	788	35	329	159,092,325
5	Ars Sonor	2,011	88	36	253	157,150,800
6	Lee Rosevere	[NULL]	372	53	342	137,066,109
7	Blue Dot Sessions	[NULL]	245	64	414	106,058,176
8	Kosta T	[NULL]	110	53	794	80,940,858
9	Chris Zabriskie	2,001	951	12	95	76,493,808
10	Kevin MacLeod	1,998	431	12	160	57,183,564
11	Kai Engel	2,012	861	16	141	55,835,248
12	junior85	2,003	77	24	128	34,326,840
13	Mr. & Mrs. Smith	[NULL]	86	36	350	32,421,312
14	krackatoa	1,998	101	27	110	31,330,908
15	Keshco	1,994	45	18	139	27,109,512
16	Big Blood	2,006	219	16	153	24,653,440
17	transient	1,992	21	11	77	23,305,040
18	Ketsa	[NULL]	216	21	261	22,350,111
19	Broke For Free	2,009	963	8	66	20,361,992

200 row(s) fetched - 1.5s, on 2025-12-02 at 20:

Table 1. Query runtime under indexing and star schema (This experiment was done on a MacBook Air M2 device with 16GB of RAM)

Query	**Non-indexed 3NF schema**	**Indexed 3NF schema**	**Star schema (dbt)**
1- Genre Fingerprinting	62.047 ms	15.031 ms **(4.1x)**	0.050 ms **(1,241x)**
2- Top Artists Yearly	74.213 ms	5.719 ms **(13.0x)**	0.120 ms **(618x)**
3- Undiscovered Gems	3.096 ms	0.014 ms **(221x)**	0.040 ms **(77x)**
4- Artist Profiling	710.696 ms	548.719 ms **(1.3x)**	1.919 ms **(370x)**

Understanding the Performance:

Genre Fingerprinting (4.1x Speedup): The index `idx_audio_track_id` allowed the database to switch from scanning entire tables ("Hash Join") to instantly pinpointing matching audio features ("Nested Loop"), significantly reducing I/O overhead.

Top Artists Yearly (13.0x Speedup): Instead of reading all of the rows sequentially, the `idx_tracks_date_recorded` index allowed the query to specific data blocks where the data exists, ignoring the rest of the data

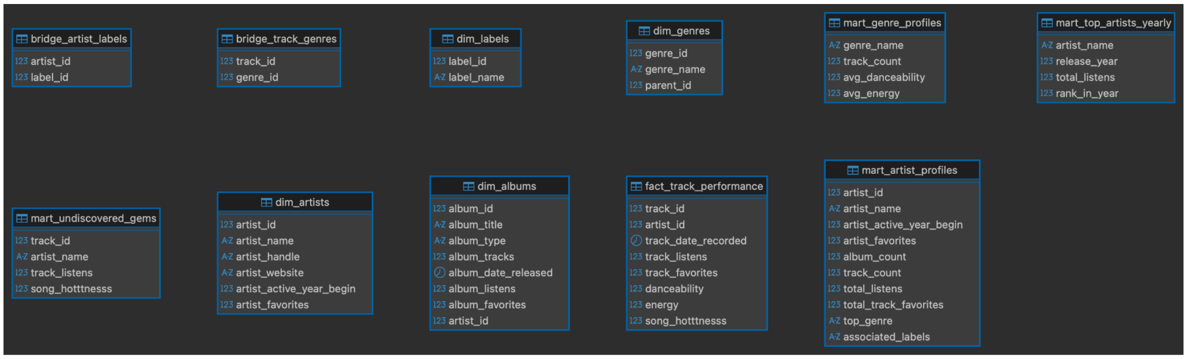
Undiscovered Gems (221x Speedup): This was the largest gain. Using the `idx_social_hotttnesss` index, the database reads the top values backwards directly from the B-Tree structure, eliminating the expensive in-memory "Sort" step.

Artist Profiling (1.3x Speedup): While a marginal speedup is observed (due to `idx_tracks_artist_id` index), the indices can't speed up aggregations (sum/avg). dbt tables ofc saw a significant speedup because the values are pre-calculated and fetched

dbt Modeling

The original schema developed during Phase 1 was normalized up to 3NF. Because the schema is in a highly normalized form, multiple `JOINs` and table scans (or index traversals) are required to execute our four test queries. Implementing a Star Schema in dbt allows us to denormalize the database to facilitate more efficient information retrieval with respect to these queries. The dbt model includes the fact and dimension tables listed below, with the indicated attributes. Mart views derived from the core tables store updated results on the four queries of interest, eliminating the need for repetitive table scans.

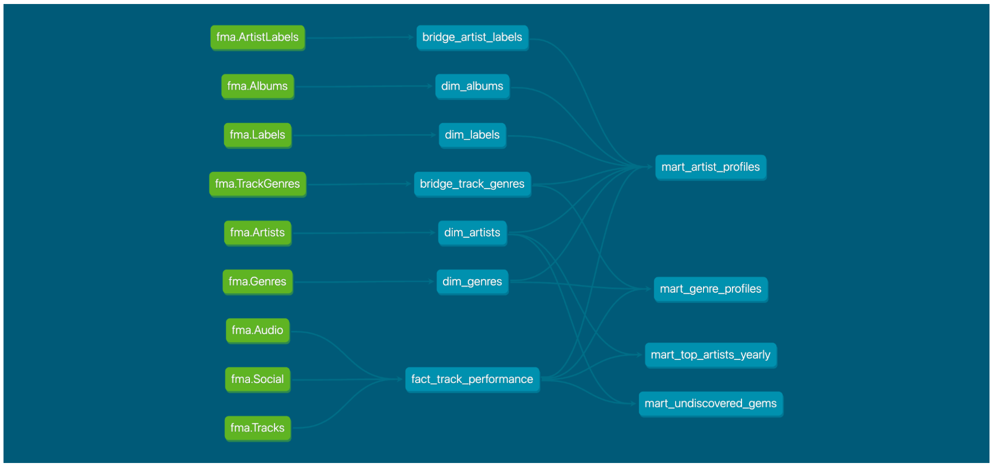
Figure 2. ER diagram for dbt schema



- **Fact tables**
  - **fact_track_performance:** `track_id`, `artist_id`, `track_date_recorded`, `track_listens`, `danceability`, `energy`, `song_hotttnesss`
- **Dimension tables**
  - **dim_albums:** `album_id`, `album_title`, `artist_id`, `album_type`, `album_favorites`, `album_tracks`, `album_date_released`, `album_listens`
  - **dim_artists:** `artist_id`, `artist_name`, `artist_active_year_begin`, `artist_favorites`
  - **dim_genre:** `genre_id`, `genre_name`, `parent_id`

  - **dim_labels:** [label_id](), label_name
- **Query-informed mart tables**
  - **mart_genre_profiles:** genre_name, track_count, avg_danceability, avg_energy
  - **mart_top_artists_yearly:** artist_name, release_year, total_listens, rank_in_year
  - **mart_undiscovered_gems:** track_id, artist_name, track_listens, song_hotttnesss
  - **mart_artist_profiles:** artist_id, artist_name, artist_active_year_begin, artist_favorites, album_count, track_count, total_listens, total_track_favorites, top_genre, associated_labels

Figure 3. Dbt-dag (STAR Schema)



Appendix: Analytical Queries

```
-- 1. Genre Fingerprinting
SELECT
    g.genre_title,
    COUNT(t.track_id) AS track_count,
    ROUND(AVG(a.danceability)::numeric, 3) AS avg_danceability,
    ROUND(AVG(a.energy)::numeric, 3) AS avg_energy
FROM "Tracks" t
JOIN "TrackGenres" tg ON t.track_id = tg.track_id
JOIN "Genres" g ON tg.genre_id = g.genre_id
JOIN "Audio" a ON t.track_id = a.track_id
GROUP BY g.genre_title
HAVING COUNT(t.track_id) > 100
ORDER BY avg_energy DESC;

-- 2. Top Artists Yearly (Window Function)
WITH YearlyStats AS (
    SELECT
        art.artist_name,
        EXTRACT(YEAR FROM t.track_date_recorded) AS release_year,
        SUM(t.track_listens) AS total_listens
    FROM "Tracks" t
    JOIN "Artists" art ON t.artist_id = art.artist_id
    WHERE t.track_date_recorded > '2000-01-01'
    GROUP BY 1, 2
),
RankedStats AS (
    SELECT *, RANK() OVER (PARTITION BY release_year ORDER BY total_listens DESC)
as yr_rank
    FROM YearlyStats
)
SELECT * FROM RankedStats
WHERE yr_rank <= 3
ORDER BY release_year DESC, yr_rank ASC;

-- 3. Undiscovered Gems (Subquery)
SELECT
    t.track_title,
    art.artist_name,
    t.track_listens,
    s.song_hotttnesss
FROM "Tracks" t
JOIN "Social" s ON t.track_id = s.track_id
JOIN "Artists" art ON t.artist_id = art.artist_id
WHERE s.song_hotttnesss > 0.4
  AND t.track_listens < (SELECT AVG(track_listens) FROM "Tracks")
ORDER BY s.song_hotttnesss DESC
LIMIT 50;

-- 4. Artist Profiling (Subquery)

SELECT
    a.artist_id,
    a.artist_name,
    a.artist_active_year_begin,
    a.artist_favorites,
    COUNT(DISTINCT al.album_id) as album_count,
    COUNT(DISTINCT t.track_id) as track_count,
    -- May need to use COALESCE(SUM(t.track_listens), 0) as total_listens,
    COALESCE(SUM(t.track_listens), 0) as total_listens,
    COALESCE(SUM(t.track_favorites), 0) as total_track_favorites,
    STRING_AGG(DISTINCT l.label_name, ', ') as associated_labels,
    -- Getting most common genre for an artist
    (
        SELECT g.genre_name
        FROM "TrackGenres" tg
        JOIN "Genres" g ON tg.genre_id = g.genre_id
        JOIN "Tracks" t2 ON tg.track_id = t2.track_id
        WHERE t2.artist_id = a.artist_id
        GROUP BY g.genre_name
        ORDER BY COUNT(*) DESC
        LIMIT 1
    ) as top_genre
FROM "Artists" a
LEFT JOIN "Albums" al ON a.artist_id = al.artist_id
LEFT JOIN "Tracks" t ON a.artist_id = t.artist_id
LEFT JOIN "Social" s ON t.track_id = s.track_id
LEFT JOIN "ArtistLabels" arl ON a.artist_id = arl.artist_id
LEFT JOIN "Labels" l ON arl.label_id = l.label_id
GROUP BY a.artist_id, a.artist_name, a.artist_active_year_begin,
a.artist_favorites
ORDER BY total_listens DESC, artist_favorites DESC
```