LAPORAN TUGAS BESAR BASIS DATA

Music Streaming Service

Laporan ini disusun untuk memenuhi Tugas Mata Kuliah Sistem Basis Data.

Disusun oleh:

Arkan Ramadhan Nugraha	241524033
Fauzi Ismail	241524042
Nurahma Rahayu	241524058
Virli Nasyila Putri	241524062

PROGRAM STUDI SARJANA TERAPAN TEKNIK INFORMATIKA

JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA

POLITEKNIK NEGERI BANDUNG

2025

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

1. DESKRIPSI

Layanan *streaming* musik digital, seperti Spotify, telah menjadi platform utama bagi masyarakat untuk mengonsumsi konten audio. Keberhasilan layanan ini sangat bergantung pada kemampuannya untuk mengelola basis data yang masif dan kompleks. Masalah utama yang dihadapi adalah bagaimana menyimpan, mengelola, dan menghubungkan jutaan data lagu, album, dan artis, sekaligus melayani permintaan jutaan pengguna secara *real-time* dengan akurat. Kita perlu memperhatikan, bagaimana data-data tersebut disimpan, bagaimana juga data tersebut berkembang seiring berjalannya waktu, bagaimana data tersebut ketika dijalankan, bagaimana keamanannya. Sistem basis data yang dirancang dengan baik sangat krusial untuk memastikan data dapat diakses dengan cepat, rekomendasi yang dipersonalisasi dapat diberikan, dan integritas data tetap terjaga.

Proses pengolahan data (proses bisnis) utama dapat dibagi menjadi dua alur besar: alur pengguna (konsumen) dan alur konten (kreator/admin).

1. Alur Pengguna (Konsumen)

   Proses dimulai saat User mendaftar dan melakukan login menggunakan email dan password. Setelah masuk, user dapat melakukan berbagai aktivitas inti seperti streaming (mendengarkan) Lagu, mencari Artis, menelusuri Album (atau EP dan kompilasi), dan memfilter berdasarkan Genre. Alur data yang dipersonalisasi terjadi ketika user mulai berinteraksi dengan sistem, seperti:

   1. Menyukai Lagu, album, atau EP
   2. Mengikuti (Follow) Artis
   3. Mengikuti (Follow) User lain
   4. Membuat Playlist, yaitu koleksi lagu custom dengan urutan custom yang spesifik.
   5. Dapat memberi rating dan memberi *review* lagu/album/EP.

2. Alur Konten (Kreator/Admin)

   Alur ini mengelola bagaimana konten musik itu sendiri direpresentasikan dalam database

   1. Satu lagu dapat memiliki satu atau lebih genre.
   2. Satu lagu dapat dimiliki oleh satu atau lebih Artis
   3. Setiap *Lagu* memiliki metadata audio (seperti *valence*, *acousticness*) yang mendeskripsikan "vibe" atau *mood* lagu tersebut.
   4. *Artis* memiliki *Lagu* dan *Collection* (sebuah entitas yang merepresentasikan Album, EP, atau Kompilasi).

5. Sistem juga mengelola data *Tour*, yang mencatat jadwal dan lokasi tur seorang *Artis* selama periode waktu tertentu. Tour bisa berupa kolaborasi artis.
6. Artis bisa promote Satu release nya entah itu lagu atau collection di page-nya dengan message tertentu.
7. Collection bisa terdiri dari beberapa disc. Setiap disc di-reset numbering nya

Masalah unik yang diangkat dalam studi kasus ini adalah implementasi sistem *review* kuantitatif yang tidak ada di layanan referensi. Sistem ini memungkinkan pengguna memberikan rating numerik (skala 1–100) dan komentar terhadap lagu maupun koleksi musik (album/EP). Ini menciptakan alur pengolahan data baru di mana data *rating* dan komentar yang dibuat pengguna harus disimpan, diagregasi (untuk menghitung *rating* rata-rata), dan dihubungkan secara efisien ke setiap entitas musik untuk dapat ditampilkan kembali kepada *user* lain.

Untuk mendukung proses tersebut, sistem basis data harus mampu melakukan agregasi, relasi, dan pengambilan data secara efisien tanpa mengorbankan performa.

Dalam skala besar seperti Spotify yang memiliki lebih dari 100 juta lagu, hampir 7 juta podcast, serta lebih dari 713 juta pengguna termasuk 281 juta pengguna yang berlangganan, desain database tidak boleh dilakukan secara sembarangan. Terdapat 3 permasalahan utama dalam hal menyusun database design ini, yaitu:

- Performance. Basis data harus mampu menangani permintaan (query) dalam jumlah besar secara cepat dan efisien.
- Scalability. Struktur data harus mudah dikembangkan seiring berjalannya waktu.
- Security. Data pengguna dan artis harus terlindungi dari kejahatan siber.

Perancangan basis data ini dilakukan untuk membangun sebuah basis data yang efisien, terstruktur, dan mudah dikembangkan untuk mendukung operasi layanan dengan skala besar. Sistem ini juga dirancang agar mampu mengelola banyak entitas seperti users, lagu, artis, collection (Album/EP/Kompilasi), Genre, Playlist, Tour, serta interaksi antar entitas secara real-time dengan tingkat konsistensi data yang cukup tinggi.

Tujuan utama mencakup:

- Mewujudkan pengelolaan data musik digital yang terintegrasi dan terorganisir.
- Mendukung seluruh proses bisnis, mulai dari registrasi pengguna hingga pengelolaan tur artis.
- Meningkatkan kecepatan pencarian dan akses data untuk mendukung fitur real-time seperti rekomendasi dan histori pemutaran lagu.

- Menjamin integritas, konsistensi, dan keamanan data pengguna maupun konten.
- Menyediakan fondasi untuk fitur analitik dan personalisasi, seperti tren popularitas, rating rata-rata, dan rekomendasi berbasis preferensi pengguna.

Sistem basis data yang dirancang untuk layanan Music Streaming Service ini memberikan berbagai manfaat dalam konteks aplikasi musik digital, antara lain:

- Efisiensi pengelolaan data, memungkinkan penyimpanan dan pengolahan jutaan lagu, artis, album, dan pengguna dengan performa tinggi.
- Personalisasi pengalaman pengguna, melalui fitur rekomendasi lagu, playlist dinamis, dan sistem rating yang interaktif.
- Kemudahan bagi kreator dan admin, dalam mengelola konten musik, tur, serta promosi album secara terstruktur.
- Peningkatan keterlibatan pengguna, karena adanya fitur interaksi sosial seperti like, follow, dan rating.
- Keamanan dan integritas data yang lebih baik, mendukung keberlanjutan layanan musik digital yang andal dan modern.

2. RUANG LINGKUP

Ruang lingkup dari studi kasus ini berfokus pada perancangan dan implementasi sistem basis data relasional untuk layanan Music Streaming Service yang meniru konsep aplikasi musik digital seperti Spotify. Sistem ini mencakup pengelolaan data pengguna, artis, lagu, koleksi (album/EP/kompilasi), genre, playlist, serta aktivitas interaksi pengguna seperti follow, like, dan rating.

Perancangan difokuskan pada bagaimana data antar entitas tersebut saling berelasi, disimpan, dan diolah secara efisien untuk mendukung proses bisnis utama layanan streaming musik digital.

Berikut adalah komponen utama yang termasuk dalam ruang lingkup sistem:

1. Manajemen User
    a. Registrasi dan Login

       Pengguna dapat membuat akun baru dan masuk ke sistem menggunakan email dan password.
    b. Profil Pengguna

       Pengguna dapat mengubah nama dan profile picture.
    c. Streaming dan Riwayat Listening

       Pengguna dapat streaming/memutar lagu dan histori pemutaran disimpan untuk mendukung fitur rekomendasi pengguna.
    d. Interaksi Sosial

       Pengguna dapat follow pengguna lain, follow artis, melihat aktivitas dan playlist pengguna lain yang publik, like lagu, review collection, review bisa di like oleh pengguna lain, pengguna dapat memblokir pengguna lain, mem-blacklist artis, dan memasukkan playlist orang lain dan collection ke library. Playlist dapat bersifat publik atau privat, serta dapat dibuat untuk digunakan (termasuk menambah lagu) bersama (collaborative) oleh beberapa pengguna.
2. Manajemen Konten Musik
    a. Entitas Lagu

Menyimpan informasi dasar seperti judul, durasi, artis, genre, serta metadata audio seperti *valence*, *energy, danceability*, dan *acousticness*.

b. Entitas Collection (Album/EP/Koleksi/Single)

Menyimpan beberapa lagu, satu lagu dapat dimiliki oleh beberapa collection. Setiap lagu dalam sebuah collection memiliki nomor disc dan nomor urut track.

c. Genre

Lagu dapat dikategorikan dalam satu atau lebih genre.

d. Kolaborasi Artis

Satu lagu dapat dimiliki oleh beberapa artis (many-to-many).

3. Manajemen Playlist dan Koleksi Pribadi Pengguna

a. Pembuatan Playlist

Menyimpan beberapa lagu user dapat memiliki beberapa playlist. Playlist dapat diatur agar tampil di profil pengguna (isOnProfile), bersifat publik/privat (isPublic), serta dapat dibuat bersama pengguna lain (isCollaborative).

b. Pengaturan Urutan Lagu

Urutan lagu dapat disusun manual oleh user atau berdasarkan jumlah pemutaran yang tercatat.

c. Penambahan Lagu ke Playlist

4. Manajemen Artis

a. Profil Artis

Menyimpan data seperti biografi, profile picture, banner, jumlah pengikut, daftar lagu dan collections milik artis.

b. Rilis Collection

Artis dapat merilis lagu atau collection, dicatat tanggal rilis nya. Suatu rilis bisa berupa Pre-Release, dimana wujudnya sudah dapat terlihat (tracklist, judul collection, tipe collection, cover album/lagu), tetapi lagu-lagu yang belum dirilis belum dapat didengar sampai release date.

c. Promosi Rilis

Artis dapat memilih satu rilisnya (lagu atau collection) untuk di rekomendasi kan di profilnya.

d. Manajemen Tur

Daftar tur artis tersebut, termasuk artis lain yang ada bareng, tanggal tur, tempat tur, dan lain-lain.

5. Sistem Rating dan Review
    a. Pemberian Rating

       Pengguna dapat memberikan nilai numerik (skala 1-100) terhadap lagu atau koleksi. Nilai ini menjadi indikator tingkat kepuasan pengguna terhadap konten tersebut.

    b. Komentar Review

       Pengguna dapat menulis komentar yang berisikan pendapat atau ulasan pribadi tentang kolesi. Untuk lagu, hanya ada rating saja.

    c. Agregasi

       Menghitung rata-rata rating dari seluruh pengguna yang memberi penilaian terhadap suatu lagu atau koleksi. Nilai agregasi ini disimpan dan digunakan untuk menampilkan peringkat atau rekomendasi

    d. Tampilan Rating

       Nilai rata-rata rating serta jumlah review akan ditampilkan pada halaman detail lagu (hanya rating dan jumlah rating) atau koleksi.

6. Pencarian dan Rekomendasi

   Fitur ini merupakan proses turunan dari data yang tersimpan dalam basis data relasional

    a. Pencarian berdasarkan entitas

       Pengguna dapat mencari lagu, artis, atau album berdasarkan nama, genre, atau kata kunci lainnya.

    b. Filter dan Sortir

       Hasil pencarian dapat difilter berdasarkan genre, popularitas, rating, atau waktu rilis.

    c. Personalized Recommendations

       Sistem dapat menampilkan rekomendasi berdasarkan histori mendengarkan, artis yang diikuti, atau rating yang diberikan pengguna.

7. Keamanan dan Integritas Data
    a. Keamanan Akses

       Sistem mendukung pengaturan hak akses di tingkat aplikasi untuk memastikan keamanan data pengguna.

b. Integritas Relasi

Seluruh entitas (User, Song, Artist, Collection, Genre, Playlist, Rating, Tour) terhubung melalui kunci utama dan kunci asing untuk menjaga konsistensi.

8. Batasan Sistem

Ruang lingkup kasus ini tidak mencakup:

a. Implementasi streaming audio secara aktual

b. Sistem pembayaran, langganan, atau manajemen lisensi lagu.

3. BUSINESS RULE

Pada bagian ini dijelaskan aturan-aturan bisnis yang mengatur proses pengolahan data dalam sistem Music Streaming Service. Aturan bisnis ini berfungsi untuk menggambarkan bagaimana setiap entitas saling berinteraksi serta batasan-batasan yang berlaku dalam pengelolaan data di sistem.

Tabel berikut menyajikan daftar proses bisnis utama beserta entitas yang terlibat dan aturan yang diterapkan dalam masing-masing proses:

Tabel 1 Aturan Bisnis Pengolahan Data

No.	Nama Proses Bisnis	Entitas	Aturan
1.	Registrasi dan Login User	User	Email dan username harus unik. Password harus memenuhi panjang minimum (hashnya harus lebih dari 8 karakter). Email dan username wajib ada. Username tidak boleh berupa whitespace. Email harus mengikuti format email valid.
		User	User tidak dapat mengakses fitur seperti streaming, like, dan rating tanpa melakukan login terlebih dahulu.

		User	User dapat memiliki satu atau lebih playlist pribadi setelah berhasil login.
2.	Pembuatan Playlist oleh user	User, Playlist	User dapat membuat banyak playlist dengan nama yang berbeda
		Playlist	Satu playlist dapat berisi satu atau lebih lagu. Playlist harus memiliki satu pemilik. Playlist default bersifat private dan non-collaborative.
		Song	Satu lagu dapat dimasukkan ke banyak playlist yang berbeda.
		Song	Satu playlist dapat terdiri dari beberapa lagu.
		Song	User dapat mengatur urutan lagu dalam playlist.
		User, Playlist	User dapat menyimpan playlist user lain
		User, Playlist, Song	Urutan lagu harus bernilai positif dan berurutan untuk setiap playlist.
3.	Streaming dan History (Riwayat Listening)	User, Song	Pengguna dapat streaming/memutar lagu dan Riwayat listening-nya disimpan sebagai history.
		User, Song	Durasi dengar harus positif. Setiap record memiliki timestamp otomatis.
4.	Rating dan Review	User, Collection	Satu user hanya dapat memberi satu review per

			koleksi besrta rating yang berada pada rentang 1–100.
		User, Song	User hanya boleh memberikan satu rating per lagu. Rating berada dalam rentang 1–100.
		Review	Satu user hanya dapat memberikan satu review untuk setiap collection
		Song, Collection	Nilai rata – rata rating dihitung dari data review (untuk collection) dan data rating (untuk song)
		User, Review	User dapat memberikan like pada hasil review dari orang lain. Satu user hanya boleh like satu review sekali. Tidak boleh like review berulang.
5.	Sistem Follow	User, Artis	User dapat mengikuti artis. User hanya dapat follow artis satu kali.
		User	User dapat mengikuti user lain. User tidak dapat mem-follow dirinya sendiri. Satu user tidak dapat follow user yang sama lebih dari satu kali.
6.	Manajemen Koleksi Musik	Artis, Collection, Song	Satu artis dapat memiliki banyak koleksi (collection)

		Collection	Tipe koleksi harus salah satu dari: Album, EP, Single, Compilation.
		Collection, Song	Satu lagu dapat masuk ke beberapa koleksi . Nomor disc dan nomor track harus bernilai positif untuk setiap lagu dalam suatu koleksi.
		Collection	Koleksi berstatus prerelease harus memiliki release_date di masa depan. Pada hari rilis, status prerelease tidak lagi berlaku.
		Collection, Genre	Koleksi hanya dapat menyimpan maksimal 3 genre teratas berdasarkan lagu-lagunya.
7.	Manajemen Lagu	Artis, Song, Genre	Lagu dapat dibuat oleh satu atau lebih artis (kolaborasi). Setiap lagu harus memiliki minimal satu artis. Pasangan lagu-artis tidak boleh duplikat
		Song, Genre	Lagu dapat memiliki satu atau lebih genre(maks 3). Satu lagu dapat memiliki beberapa genre, tetapi tidak boleh duplikat pasangan.
		Song	Durasi lagu harus positif. Popularitas berada dalam rentang 0–100. Nilai audio metrics berada dalam rentang 0–1 dengan 3 angka di desimal. Popularitas

			dihitung dengan rumus berikut: popularitas = (jumlah_play_30_hari_terakhir / jumlah_play_lagu_terbanyak_di_sistem_30_hari_terakhir) * 100
		User, Song	User hanya boleh like satu lagu satu kali. Timestamp tercatat otomatis.
8.	Promosi Lagu oleh Artis	Artis, Collection, Song	Artis hanya dapat mempromosikan koleksi yang benar-benar dirilis oleh artis tersebut.
9.	Manajemen Tur Artis	Artis, Tour	Artis dapat memiliki satu atau lebih jadwal tour
		Tour	Tour mencatat tour_name, venue, dan tour_date. Tanggal tour tidak boleh berada di masa lalu.
		Artis, Tour	Satu tour dapat melibatkan beberapa artis
10.	Pencarian dan Rekomendasi	User, Song, Artis, Genre	User dapat mencari lagu, artis, atau koleksi berdasarkan kata kunci
		User, Song	Sistem rekomendasi didasarkan pada histori mendengarkan dan artis yang diikuti.
11.	Keamanan dan Integritas Data	User	Atribut password pada User dilindungi dengan function/algoritma bcrypt beserta salt.

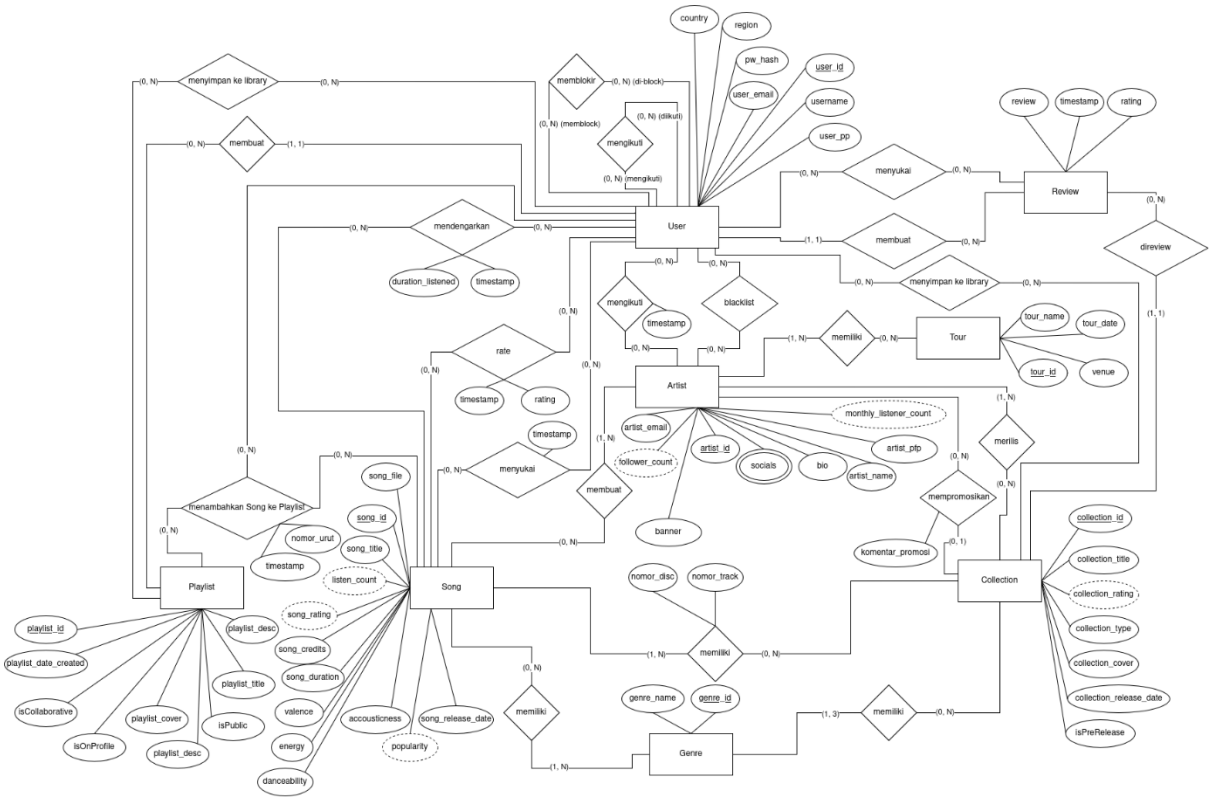
12.	Blocking User dan Blacklist Artis	User	User tidak dapat mem-block dirinya sendiri. Block bersifat unik per pasangan user.
		User, Artis	Setiap user dapat mem-blacklist seorang/beberapa artis. User hanya dapat mem-block artis satu kali.
13.	Manajemen sosial media artis	Artis	Artis dapat memiliki banyak social media link
14.	Manajemen Library User	User, Playlist	User hanya dapat menyimpan playlist satu kali; tidak boleh duplikat
		User, Collection	Satu koleksi hanya dapat disimpan sekali per user. Tidak boleh duplikat.

4. PERANCANGAN PENGOLAHAN DATA

4.1 Entity-Relationship Diagram (E-RD)

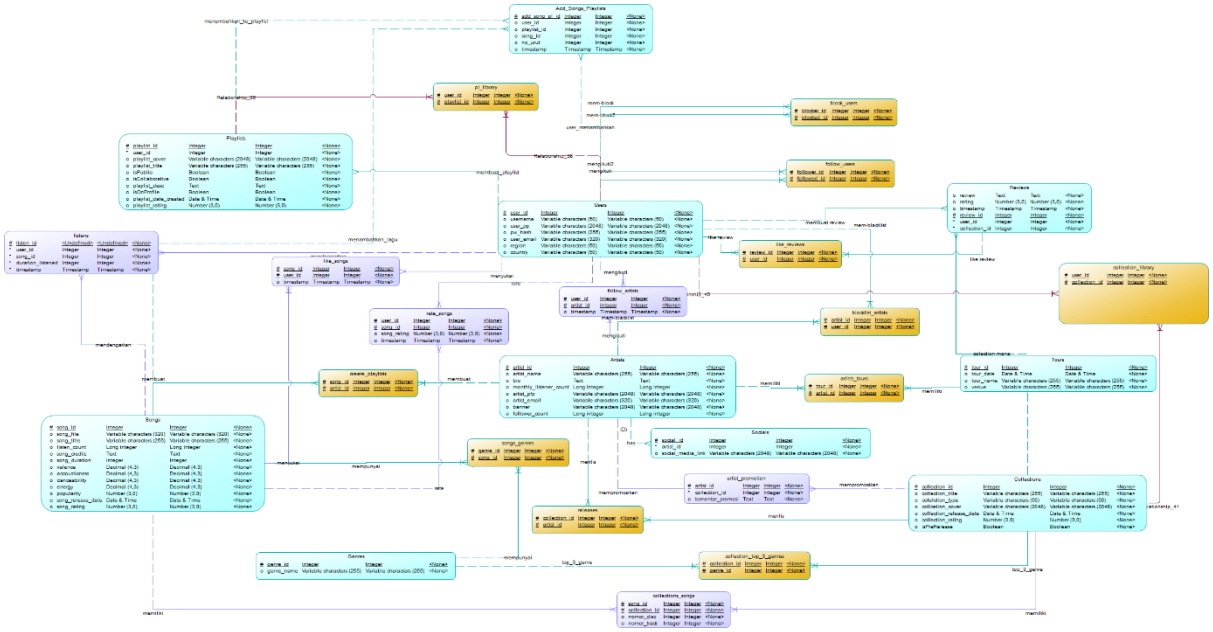
Untuk menggambarkan hubungan antar entitas dalam sistem yang dirancang, dibuat sebuah Entity-Relationship Diagram (ERD) dengan menggunakan notasi Chen. Diagram ini bertujuan untuk memodelkan struktur data secara konseptual, termasuk entitas utama, atribut yang dimiliki, serta hubungan antar entitas beserta kardinalitasnya (minimum dan maksimum).

ERD berikut menunjukkan bagaimana setiap entitas saling berinteraksi dalam sistem basis data yang akan dibangun:



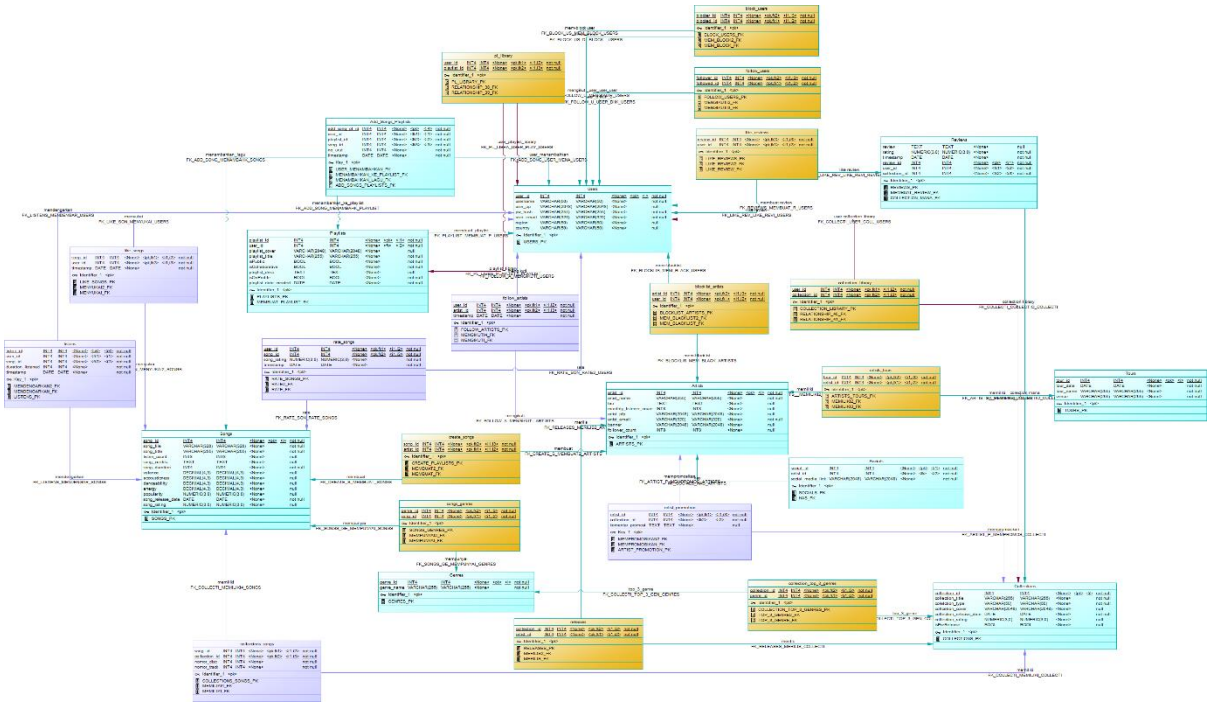
Gambar 4. 1 Entity-Relationship Diagram (ERD)

4.2 Logical Data Model



Gambar 4. 2 Logical Data Model (LDM)

4.3 Physical Data Model



Gambar 4. 3 Physical Data Model (PDM)

4.4 Kamus Data

4.4.1 Table

Tabel 2 Tabel Users

Nama	USERS
Deskripsi	Tabel ini menyimpan data pengguna aplikasi, termasuk identitas dasar, kredensial login, serta informasi wilayah pengguna. Digunakan sebagai entitas utama dalam sistem karena berhubungan dengan playlist, like, follow, rating, dan aktivitas lainnya.

No	Nama Kolom	Tipe Data	Constraint	Deskripsi	Contoh Data
1	USER_ID	INT4	PK, NOT NULL	ID unik untuk setiap user	1023
2	USERNAME	VARCHAR(50)	NOT NULL, LENGTH(TRIM (USERNAME)) > 0	Nama akun yang ditampilkan	"aurorabeats"
3	USER_PFP	VARCHAR(2048)	NULL	URL foto profil user	"https://img.com/u/pp123.png"
4	PW_HASH	VARCHAR(255)	NOT NULL, LENGTH >= 8	Hash password user	"$2a$10$9sd8asd8asd..."
5	USER_EMAIL	VARCHAR(320)	NOT NULL, LIKE '%_@__%.__%'	Email untuk login/verifikasi	"aurora@mail.com"
6	REGION	VARCHAR(50)	NULL	Wilayah tertentu dari user	"West Java"
7	COUNTRY	VARCHAR(50)	NULL	Negara asal user	"Indonesia"

Tabel 3 Tabel Artists

Nama	ARTISTS
Deskripsi	Menyimpan data artis, termasuk nama, foto profil, banner, email, jumlah pendengar, dan koleksi (album/EP) yang terkait. Tabel ini menggunakan composite primary key (ARTIST_ID, COLLECTION_ID).

No	Nama Kolom	Tipe Data	Constraint	Deskripsi	Contoh Data
1	ARTIST_ID	INT4	PK, NOT NULL	ID unik artis	11
2	ARTIST_NAME	VARCHAR(255)	NOT NULL, LENGTH(TRIM(ARTIST_NAME)) > 0	Nama artis	“The Weekend”
3	BIO	TEXT	NULL	Biografi artis	“Canadian singer and producer...”
4	MONTHLY_LISTENER_COUNT	INT8	NULL, default 0, value >= 0	Jumlah pendengar bulanan	67500000
5	ARTIST_PFP	VARCHAR(2048)	NULL	Foto profil artis	“https://cdn.com/a11.png”
6	ARTIST_EMAIL	VARCHAR(320)	NOT NULL	Email artis	“tw@music.com”
7	BANNER	VARCHAR(2048)	NULL	Link gambar banner profil artis	https://cdn.com/banner11.png
8	FOLLOWER_COUNT	INT8	NULL, default 0	Jumlah pengikut	45000000

Tabel 4 Tabel Songs

Nama	SONGS
Deskripsi	Menampung informasi detail lagu seperti file path, judul, durasi, statistik popularitas, metrik audio (valence, acousticness, energy), tanggal rilis, dan rating lagu.

No	Nama Kolom	Tipe Data	Constraint	Deskripsi	Contoh Data
1	SONG_ID	INT4	PK, NOT NULL	ID unik lagu	510
2	SONG_FILE	VARCHAR(320)	NOT NULL	Lokasi file audio	“/songs/track510.mp3”
3	SONG_TITLE	VARCHAR(255)	NOT NULL	Judul lagu	“Blinding Lights”

4	LISTEN_COUNT	INT8	NULL, default 0	Total jumlah diputar	2300000000
5	SONG_CREDITS	TEXT	NULL	Keterangan produser/artis	“Produced by Max Martin”
6	SONG_DURATION	INT4	NOT NULL, value > 0	Durasi lagu (detik)	201
7	VALENCE	DECIMAL(4,3)	NULL, 0 <= value <= 1	Mood positif lagu	0.520
8	ACCOUSTICNESS	DECIMAL(4,3)	NULL, 0 <= value <= 1	Kelembutan suara	0.11
9	DANCEABILITY	DECIMAL(4,3)	NULL, 0 <= value <= 1	Kelayakan lagu untuk menari	0.803
10	ENERGY	DECIMAL(4,3)	NULL, 0 <= value <= 1	Energi audio	0.734
11	POPULARITY	NUMERIC(3,0)	NULL, 0 <= value <= 100	Skor popularitas (0-100). Rumus seperti berikut: popularitas = (jumlah_play_30_hari_terakhir / jumlah_play_lagu_terbanyak_di_sistem_30_hari_terakhir) * 100	87
12	SONG_RELEASE_DATE	DATE	NOT NULL	Tanggal rilis	“2020-03-20”
13	SONG_RATING	NUMERIC(3,0)	NULL	Rating rata-rata	9

Tabel 5 Tabel Genres

Nama	GENRES				
Deskripsi	Menyimpan daftar genre musik yang digunakan untuk mengategorikan lagu dan koleksi.				
No	**Nama Kolom**	**Tipe Data**	**Constraint**	**Deskripsi**	**Contoh Data**
1	GENRE_ID	INT4	PK, NOT NULL	ID unik genre	30

2	GENRE_NAME	VARCHAR(255)	NOT NULL	Nama genre	“Noise Pop”

Tabel 6 Tabel Playlists

Nama	PLAYLISTS
Deskripsi	Menyimpan informasi playlist yang dibuat oleh user. Playlist dapat berisi banyak lagu dan memiliki sampul, deskripsi, serta status publik/privat.

No	**Nama Kolom**	**Tipe Data**	**Constraint**	**Deskripsi**	**Contoh Data**
1	PLAYLIST_ID	INT4	PK, NOT NULL	ID unik playlist	7001
2	PLAYLIST_TITLE	VARCHAR(255)	NOT NULL	Nama playlist	"Late Night Vibes"
3	ISPUBLIC	BOOL	NOT NULL, default false	Status publik/privat	TRUE
4	PLAYLIST_COVER	VARCHAR(2048)	NULL	URL sampul playlist	"https://img.com/pl7001.jpg"
5	USER_ID	INT4	FK USERS.USER_ID	Pembuat playlist	101
6	ISCOLLABORATIVE	BOOL	NOT NULL, default false	Menandakan apakah playlist bersifat kolaboratif	TRUE
7	PLAYLIST_DESC	TEXT	NULL	Deskripsi playlist	"A curated mix of synthwave and chill electronic beats."
8	ISONPROFILE	BOOL	NOT NULL	Menandakan apakah playlist ditampilkan di halaman profil user	TRUE

9	PLAYLIST_DATE_CREATE	DATE	NOT NULL	Tanggal saat playlist dibuat oleh user.	"2024-12-20"

Tabel 7 Tabel Pl_Library

Nama	PL_LIBRARY				
Deskripsi	Menunjukkan playlist apa saja yang disimpan (added to library) oleh user. Berbeda dari playlist yang dibuat sendiri.				
No	**Nama Kolom**	**Tipe Data**	**Constraint**	**Deskripsi**	**Contoh Data**
1	USER_ID	INT4	PK, FK USER.USER_ID, NOT NULL	User pemilik library	101
2	PLAYLIST_ID	INT4	PK, FK PLAYLISTS.PLAYLIST_ID, NOT NULL	Playlist yang disimpan	7001

Tabel 8 Tabel Add_Songs_Playlists

Nama	ADD_SONGS_PLAYLISTS				
Deskripsi	Menyimpan daftar lagu yang ditambahkan ke playlist beserta urutan lagu dalam playlist.				
No	**Nama Kolom**	**Tipe Data**	**Constraint**	**Deskripsi**	**Contoh Data**
1	PLAYLIST_ID	INT4	FK PLAYLISTS.PLAYLIST_ID, NOT NULL	Playlist	7001
2	SONG_ID	INT4	FK SONGS.SONG_ID, NOT NULL	Lagu yang masuk playlist	510
3	NO_URUT	INT4	NOT NULL, value > 0	Urutan lagu dalam playlist	1
4	USER_ID	INT4	FK, NOT NULL	User yang menambahkan lagu ke playlist	101

5	TIMESTAMP	TIMESTAMP	NULL	Timestamp lagu ditambahkan ke playlist	“2025-01-19 00:00:00"
6	ADD_SONG_PL_ID	INT4	PK	ID entry user memasukkan lagu ke playlist	1

Tabel 9 Tabel Like_Songs

Nama	LIKE_SONGS				
Deskripsi	Menyimpan daftar lagu yang disukai oleh user.				
No	**Nama Kolom**	**Tipe Data**	**Constraint**	**Deskripsi**	**Contoh Data**
1	USER_ID	INT4	PK, FK USERS.USER_ID	User yang menyukai	101
2	SONG_ID	INT4	PK, FK SONGS.SONG_ID	Lagu yang disukai	510
3	TIMESTAMP	TIMESTAMP	NOT NULL, default current_timestamp	Timestamp user memberikan like pada lagu	"2025-01-19 00:00:00"

Tabel 10 Tabel Listens

Nama	LISTENS				
Deskripsi	Mencatat riwayat pemutaran lagu oleh user (untuk rekomendasi dan statistik personal).				
No	**Nama Kolom**	**Tipe Data**	**Constraint**	**Deskripsi**	**Contoh Data**
1	USER_ID	INT4	FK USERS.USER_ID, NOT NULL	User yang mendengarkan	101
2	SONG_ID	INT4	FK SONGS.SONG_ID, NOT NULL	Lagu yang diputar	510

3	DURATION_LISTENED	INT4	NOT NULL	Selama berapa detik lagu diputar	122
4	TIMESTAMP	DATE	NOT NULL, default current_timestamp	Timestamp aktivitas mendengarkan lagu	"2025-01-20"
5	LISTEN_ID	INT4	NOT NULL, PK	ID satu aktivitas listening	3

Tabel 11 Tabel Follow_Users

Nama	FOLLOW_USERS				
Deskripsi	Mencatat hubungan "follow" antar sesama pengguna. User dapat mengikuti user lain untuk melihat aktivitas dan playlist mereka.				
No	**Nama Kolom**	**Tipe Data**	**Constraint**	**Deskripsi**	**Contoh Data**
1	FOLLOWER_ID	INT4	PK, FK USERS.USER_ID, FOLLOWER_ID <> FOLLOWED_ID	User yang mengikuti	101
2	FOLLOWED_ID	INT4	PK, FK USERS.USER_ID, FOLLOWER_ID <> FOLLOWED_ID	User yang di-follow	205

Tabel 12 Tabel Follow_Artists

Nama	FOLLOW_ARTISTS				
Deskripsi	Mencatat user yang mengikuti artis. Data ini digunakan untuk personalisasi konten dan statistik follower artis.				
No	**Nama Kolom**	**Tipe Data**	**Constraint**	**Deskripsi**	**Contoh Data**
1	USER_ID	INT4	PK, FK USERS.USER_ID	User yang mengikuti artis	101

2	ARTIST_ID	INT4	PK, FK ARTISTS.ARTIST_ID	Artis yang di-follow	11
3	TIMESTAMP	TIMESTAMP	NOT NULL, default current_timestamp	Timestamp mengikuti artist	"2025-01-20 00:00:00"

Tabel 13 Tabel Collections

Nama	COLLECTIONS
Deskripsi	Merepresentasikan kumpulan lagu seperti Album, EP, atau Kompilasi. Setiap collection dimiliki oleh artis, memiliki tanggal rilis, deskripsi, rating, dan dapat memiliki banyak disc.

No	Nama Kolom	Tipe Data	Constraint	Deskripsi	Contoh Data
1	COLLECTION_ID	INT4	PK, NOT NULL	ID unik collection	2
2	COLLECTION_TITLE	VARCHAR(255)	NOT NULL	Nama album/EP/kompilasi	"After Hours"
3	COLLECTION_RELEASE_DATE	DATE	NOT NULL	Tanggal rilis	"2020-03-20"
4	COLLECTION_TYPE	CHAR(11)	NOT NULL, value in ('Album', 'EP', 'Single', 'Compilation')	Jenis collection (Album/EP/etc.)	"Album"
5	COLLECTION_COVER	VARCHAR(2048)	NULL	URL cover album atau EP	https://img.com/coll1001.jpg
6	COLLECTION_RATING	NUMERIC(3,0)	NULL	Rating rata-rata	9
7	ISPRERELEASE	BOOL	NULL	Menandakan apakah collection masih prerelease	FALSE

				(belum dirilis penuh)	

Tabel 14 Tabel Songs_Genres

Nama	SONGS_GENRES				
Deskripsi	Menghubungkan lagu dan genre dalam relasi many-to-many (karena satu lagu dapat memiliki beberapa genre).				
No	**Nama Kolom**	**Tipe Data**	**Constraint**	**Deskripsi**	**Contoh Data**
1	GENRE_ID	INT4	PK, FK GENRES.GENRE_ID, NOT NULL	Genre lagu	30
2	SONG_ID	INT4	PK, FK SONGS.SONG_ID, NOT NULL	Lagu yang memiliki genre	510

Tabel 15 Tabel Collection_Top_3_Genre

Nama	COLLECTION_TOP_3_GENRE				
Deskripsi	Menghubungkan collection (album/EP/kompilasi) dengan genre. Satu album dapat mencakup beberapa genre.				
No	**Nama Kolom**	**Tipe Data**	**Constraint**	**Deskripsi**	**Contoh Data**
1	COLLECTION_ID	INT4	PK, FK COLLECTIONS.COLLECTION_ID, NOT NULL	Koleksi/album	20
2	GENRE_ID	INT4	PK, FK GENRES.GENRE_ID, NOT NULL	Genre yang terkait	30

Tabel 16 Tabel Collections_Songs

Nama	COLLECTIONS_SONGS
Deskripsi	Relasi yang menghubungkan lagu dengan collection (album/EP/kompilasi). Karena satu collection dapat berisi banyak lagu, dan sebuah lagu dapat

	muncul di beberapa collection (misalnya kompilasi), tabel ini menjadi penghubung many-to-many.

No	Nama Kolom	Tipe Data	Constraint	Deskripsi	Contoh Data
1	COLLECTION_ID	INT4	PK, FK COLLECTIONS.COLLECTION_ID	ID collection	1001
2	SONG_ID	INT4	PK, FK SONGS.SONG_ID	ID lagu	510
3	NOMOR_DISC	INT4	NULL, value > 0	Disc ke berapa lagu ditempatkan (untuk album dengan banyak disc)	1
4	NOMOR_TRACK	INT4	NULL, value > 0	Track number (urutan lagu) dalam disc	3

Tabel 17 Tabel Releases

Nama	RELEASES
Deskripsi	Menghubungkan artis dengan collection (album/ EP/ kompilasi) secara eksplisit. Diperlukan untuk kasus kolaborasi album atau kompilasi dengan banyak artis.

No	Nama Kolom	Tipe Data	Constraint	Deskripsi	Contoh Data
1	ARTIST_ID	INT4	PK, FK ARTISTS.ARTIST_ID, NOT NULL	Artis pemilik	11
2	COLLECTION_ID	INT4	PK, FK COLLECTIONS.COLLECTION_ID, NOT NULL	Collection terkait	12

Tabel 18 Tabel Create_Songs

Nama	CREATE_SONGS

Deskripsi	Menghubungkan artis dengan lagu. Satu lagu bisa memiliki banyak artis (kolaborasi), sementara satu artis punya banyak lagu.

No	Nama Kolom	Tipe Data	Constraint	Deskripsi	Contoh Data
1	ARTIST_ID	INT4	PK, FK ARTISTS.ARTIST_ID, NOT NULL	Artis yang berkontribusi	11
2	SONG_ID	INT4	PK, FK SONGS.SONG_ID, NOT NULL	Lagu hasil karya/artis tersebut	510

Tabel 19 Tabel Artist_Promotion

Nama	ARTIST_PROMOTION
Deskripsi	Fitur unik sistem: artis dapat mempromosikan satu lagu atau satu collection di halaman profilnya beserta pesan promosi.

No	Nama Kolom	Tipe Data	Constraint	Deskripsi	Contoh Data
1	ARTIST_ID	INT4	PK, FK ARTISTS.ARTIST_ID, NOT NULL, REFERENCES RELEASES (ARTIST_ID, COLLECTION_ID)	Artis yang melakukan promosi	11
2	COLLECTION_ID	INT4	FK COLLECTIONS.CLLECTION_ID, NOT NULL, REFERENCES RELEASES (ARTIST_ID, COLLECTION_ID)	Collection yang dipromosikan (opsional)	34
3	KOMENTAR_PROMOSI	TEXT	NULL	Pesan promosi	"New album out now — enjoy!"

Tabel 20 Tabel Tours

Nama	TOURS
Deskripsi	Menyimpan informasi dasar mengenai tur artis, termasuk nama tur dan jangka waktu penyelenggaraan. Satu artis dapat memiliki banyak tur, dan setiap tur dapat berlangsung di beberapa kota.

No	**Nama Kolom**	**Tipe Data**	**Constraint**	**Deskripsi**	**Contoh Data**
1	TOUR_ID	INT4	PK, NOT NULL	ID unik tur	301
2	TOUR_NAME	VARCHAR(255)	NOT NULL	Nama tur	"After Hours World Tour"
3	TOUR_DATE	DATE	NOT NULL	Tanggal mulai tur	"2023-07-10"
4	VENUE	VARCHAR(255)	NOT NULL	Tanggal selesai tur	“VENUE WAWA, Jakarta, ID”

Tabel 21 Tabel Artists_Tours

Nama	ARTISTS_TOURS
Deskripsi	Menghubungkan artis dengan tur tertentu. Diperlukan untuk tur dengan lebih dari satu artis (kolaborasi event atau festival).

No	**Nama Kolom**	**Tipe Data**	**Constraint**	**Deskripsi**	**Contoh Data**
1	ARTIST_ID	INT4	PK, FK ARTISTS.ARTIST_ID	Artis yang tampil	11
2	TOUR_ID	INT4	PK, FK TOURS.TOUR_ID	Tur yang diikuti	301

Tabel 22 Tabel Rate_Songs

Nama	RATE_SONGS
Deskripsi	Menyimpan rating numerik (1–100) yang diberikan user terhadap lagu tertentu. Digunakan untuk menghitung rata-rata rating lagu yang tersimpan di tabel SONGS.

No	Nama Kolom	Tipe Data	Constraint	Deskripsi	Contoh Data
1	USER_ID	INT4	PK, FK USERS.USER_ID	User yang memberi rating	101
2	SONG_ID	INT4	PK, FK SONGS.SONG_ID	Lagu yang diberi rating	510
3	SONG_RATING	NUMERIC(3,0)	NOT NULL, 1-100	Nilai rating (1–100)	9
4	TIMESTAMP	TIMESTAMP	NOT NULL, default current_timestamp	TIMESTAMP rating songn	"2025-01-20 00:00:00"

Tabel 23 Tabel Block_Users

Nama	BLOCK_USERS				
Deskripsi	Menyimpan relasi “user mem-block user lain”. Digunakan agar user tertentu tidak dapat melihat/berinteraksi satu sama lain.				
No	**Nama Kolom**	**Tipe Data**	**Constraint**	**Deskripsi**	**Contoh**
1	BLOCKER_ID	INT4	PK, FK USERS.USER_ID, NOT NULL, BLOCKER_ID <> BLOCKED_ID	User yang memblokir	5
2	BLOCKED_ID	INT4	PK, FK USERS.USER_ID, NOT NULL, BLOCKER_ID <> BLOCKED_ID	User yang diblokir	12

Tabel 24 Tabel Blocklist_Users

Nama	BLOCKLIST_ARTISTS				
Deskripsi	Daftar artist yang diblokir oleh user (hide artist).				
No	**Nama Kolom**	**Tipe Data**	**Constraint**	**Deskripsi**	**Contoh**

1	ARTIST_ID	INT4	PK, FK ARTISTS.ARTIST_ID, NOT NULL	Artist yang diblokir	7
2	USER_ID	INT4	PK, FK USERS.USER_ID, NOT NULL	User yang memblokir	3

Tabel 25 Tabel Collection_Library

Nama	COLLECTION_LIBRARY				
Deskripsi	Menyimpan koleksi (album/EP/single/Compilation) yang disimpan oleh user (add to library).				
No	**Nama Kolom**	**Tipe Data**	**Constraint**	**Deskripsi**	**Contoh**
1	USER_ID	INT4	PK, FK USERS.USER_ID, NOT NULL	User yang menyimpan koleksi	10
2	COLLECTION_ID	INT4	PK, FK COLLECTIONS.COLLECTION_ID, NOT NULL	Koleksi yang disimpan	2

Tabel 26 Tabel Like_Reviews

Nama	LIKE_REVIEWS				
Deskripsi	Daftar “like” yang diberikan user terhadap ulasan (review).				
No	**Nama Kolom**	**Tipe Data**	**Constraint**	**Deskripsi**	**Contoh**
1	REVIEW_ID	INT4	PK, FK REVIEWS.REVIEW_I), NOT NULL	Review yang dilike	14
2	USER_ID	INT4	PK, FK USERS.USER_ID, NOT NULL	User yang memberikan like	22

Tabel 27 Tabel Reviews

Nama	REVIEWS

Deskripsi	Review dan rating user terhadap sebuah collection (album/EP/single).				
No	**Nama Kolom**	**Tipe Data**	**Constraint**	**Deskripsi**	**Contoh**
1	REVIEW	TEXT	NULL	Isi review	"Albumnya sangat keren!"
2	RATING	NUMERIC(3,0)	NOT NULL, CHECK 1–100	Rating koleksi	85
3	TIMESTAMP	TIMESTAMP	NOT NULL, DEFAULT current_timestamp	Waktu review ditulis	2025-11-26 10:00
4	REVIEW_ID	INT4	PK	ID review	18
5	USER_ID	INT4	FK USERS.USER_ID, NOT NULL	User yang review	4
6	COLLECTION_ID	INT4	FK COLLECTIONS.COLLECTION_ID, NOT NULL	Koleksi yang direview	7

Tabel 28 Tabel Socials

Nama	SOCIALS				
Deskripsi	Daftar link sosial media milik artis.				
No	**Nama Kolom**	**Tipe Data**	**Constraint**	**Deskripsi**	**Contoh**
1	SOCIAL_ID	INT4	PK	ID sosial media	5
2	ARTIST_ID	INT4	FK ARTISTS.ARTIST_ID, NOT NULL	Artist pemilik akun	12
3	SOCIAL_MEDIA_LINK	VARCHAR(2048)	NOT NULL	URL sosial media	https://instagram.com/artist

4.4.2 Sequence

Pada studi kasus ini, sequence digunakan untuk mengotomatisasi pengisian nilai primary key pada seluruh tabel yang memerlukan ID unik. Sequence memastikan bahwa setiap record baru mendapatkan nilai ID yang berurutan, konsisten, serta tidak terjadi duplikasi ID. Penggunaan sequence juga mempermudah proses insert data dan menjaga integritas referensial antar tabel:

Tabel 29 Sequence

No.	Nama Tabel	Nama Kolom	Deskripsi
1.	Users	user_id	Sequence ini digunakan untuk menghasilkan nilai user_id secara otomatis dan berurutan saat data pengguna baru ditambahkan. Sequence ini menjamin setiap pengguna memiliki ID unik tanpa perlu input manual.
2.	Artists	artist_id	Sequence menghasilkan nilai artist_id secara otomatis untuk setiap artis baru. Digunakan untuk menjaga konsistensi ID pada tabel artis.
3.	Songs	Song_id	Sequence digunakan sebagai auto-increment pada kolom song_id saat lagu baru ditambahkan ke sistem.
4.	Playlists	playlist_id	Sequence menghasilkan nilai playlist_id secara otomatis sehingga ID playlist selalu unik dan tidak bentrok.
5.	Collections	collection_id	Sequence digunakan untuk mengisi collection_id secara otomatis ketika koleksi baru ditambahkan.

6.	Genres	genre_id	Sequence menghasilkan nilai genre_id secara otomatis untuk setiap genre yang dibuat.
7.	Reviews	review_id	Sequence digunakan untuk mengisi kolom review_id secara otomatis agar setiap review memiliki ID unik tanpa input manual.
8.	Tours	tour_id	Sequence menghasilkan tour_id secara otomatis untuk setiap data tur konser yang ditambahkan.
9.	Socials	social_id	Sequence pada kolom social_id memastikan data media sosial yang terhubung pada artis memiliki ID unik dan bertambah otomatis.
10.	Listen	Listen_id	Sequence pada kolom listen_id memastikan data listening memiliki ID unik dan bertambah otomatis
11.	Add_Songs_Playlist	Add_Song_PL_id	Sequence pada kolom ini memastikan data entry user menambahkan lagu ke playlist memiliki ID unik dan bertambah secara otomatis

4.4.3 Indexing

Pada studi kasus ini, indexing digunakan untuk meningkatkan performa query dan menjaga integritas data. Setiap primary key otomatis memiliki index unik yang memastikan pencarian record berdasarkan ID dapat dilakukan dengan cepat. Selain itu, foreign key juga dibuatkan index untuk mempercepat operasi join antar tabel serta mendukung proses update dan delete secara efisien. Penggunaan index secara tepat membantu mengoptimalkan akses data, mengurangi waktu eksekusi query, dan menjaga konsistensi referensial di seluruh tabel.

Tabel 30 Indexing

No.	Nama Tabel	Nama Index	Nama Kolom	Keterangan
1	ADD_SONGS_PLAYLISTS	USER_MENAMBAHKAN_FK	USER_ID	index FK ke USERS
2	ADD_SONGS_PLAYLISTS	MENAMBAHKAN_KE_PLAYLIST_FK	PLAYLIST_ID	index FK ke PLAYLISTS
3	ADD_SONGS_PLAYLISTS	MENAMBAHKAN_LAGU_FK	SONG_ID	index FK ke SONGS
4	ARTISTS	ARTISTS_PK	ARTIST_ID	primary key
5	ARTISTS_TOURS	ARTISTS_TOURS_PK	TOUR_ID, ARTIST_ID	primary key
6	ARTISTS_TOURS	MEMILIKI2_FK	ARTIST_ID	index FK ke ARTISTS
7	ARTISTS_TOURS	MEMILIKI3_FK	TOUR_ID	index FK ke TOURS
8	ARTIST_PROMOTION	PK_ARTIST_PROMOTION	ARTIST_ID, COLLECTION_ID	primary key
9	ARTIST_PROMOTION	MEMPROMOSIKAN2_FK	ARTIST_ID	index FK ke ARTISTS
10	ARTIST_PROMOTION	MEMPROMOSIKAN_FK	COLLECTION_ID	index FK ke COLLECTIONS
11	BLOCKLIST_ARTISTS	BLOCKLIST_ARTISTS_PK	ARTIST_ID, USER_ID	primary key
12	BLOCKLIST_ARTISTS	MEM_BLACKLIST_FK	ARTIST_ID	index FK ke ARTISTS
13	BLOCKLIST_ARTISTS	MEM_BLACKLIST2_FK	USER_ID	index FK ke USERS

14	BLOCK_USERS	BLOCK_USERS_PK	BLOCKER_ID, BLOCKED_ID	primary key
15	BLOCK_USERS	MEM_BLOCK_FK	BLOCKER_ID	index FK ke USERS
16	BLOCK_USERS	MEM_BLOCK2_FK	BLOCKED_ID	index FK ke USERS
17	COLLECTIONS	COLLECTIONS_PK	COLLECTION_ID	primary key
18	COLLECTIONS_SONGS	COLLECTIONS_SONGS_PK	SONG_ID, COLLECTION_ID	primary key
19	COLLECTIONS_SONGS	MEMILIKI_FK	COLLECTION_ID	index FK ke COLLECTIONS
20	COLLECTIONS_SONGS	MEMILIKI4_FK	SONG_ID	index FK ke SONGS
21	COLLECTION_LIBRARY	COLLECTION_LIBRARY_PK	USER_ID, COLLECTION_ID	primary key
22	COLLECTION_LIBRARY	RELATIONSHIP_40_FK	USER_ID	index FK ke USERS
23	COLLECTION_LIBRARY	RELATIONSHIP_41_FK	COLLECTION_ID	index FK ke COLLECTIONS
24	COLLECTION_TOP_3_GENRES	COLLECTION_TOP_3_GENRES_PK	COLLECTION_ID, GENRE_ID	primary key
25	COLLECTION_TOP_3_GENRES	TOP_3_GENRE_FK	COLLECTION_ID	index FK ke COLLECTIONS
26	COLLECTION_TOP_3_GENRES	TOP_3_GENRE2_FK	GENRE_ID	index FK ke GENRES

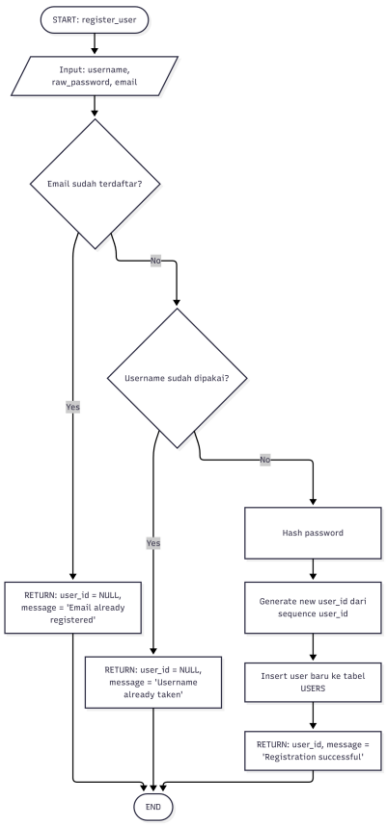
27	CREATE_PLAYLISTS	CREATE_PLAYLISTS_PK	SONG_ID, ARTIST_ID	primary key
28	CREATE_PLAYLISTS	MEMBUAT_FK	SONG_ID	index FK ke SONGS
29	CREATE_PLAYLISTS	MEMBUAT2_FK	ARTIST_ID	index FK ke ARTISTS
30	FOLLOW_ARTISTS	FOLLOW_ARTISTS_PK	USER_ID, ARTIST_ID	primary key
31	FOLLOW_ARTISTS	MENGIKUTI_FK	ARTIST_ID	index FK ke ARTISTS
32	FOLLOW_ARTISTS	MENGIKUTI4_FK	USER_ID	index FK ke USERS
33	FOLLOW_USERS	FOLLOW_USERS_PK	FOLLOWER_ID, FOLLOWED_ID	primary key
34	FOLLOW_USERS	MENGIKUTI2_FK	FOLLOWED_ID	index FK ke USERS
35	FOLLOW_USERS	MENGIKUTI3_FK	FOLLOWER_ID	index FK ke USERS
36	GENRES	GENRES_PK	GENRE_ID	primary key
37	LIKE_REVIEWS	LIKE_REVIEWS_PK	REVIEW_ID, USER_ID	primary key
38	LIKE_REVIEWS	LIKE_REVIEW_FK	REVIEW_ID	index FK ke REVIEWS
39	LIKE_REVIEWS	LIKE_REVIEW2_FK	USER_ID	index FK ke USERS
40	LIKE_SONGS	LIKE_SONGS_PK	SONG_ID, USER_ID	primary key
41	LIKE_SONGS	MENYUKAI_FK	USER_ID	index FK ke USERS
42	LIKE_SONGS	MENYUKAI2_FK	SONG_ID	index FK ke SONGS
43	LISTENS	LISTENS_PK	USER_ID, SONG_ID	primary key

44	LISTENS	MENDENGARKAN_FK	SONG_ID	index FK ke SONGS
45	LISTENS	MENDENGARKAN2_FK	USER_ID	index FK ke USERS
46	PLAYLISTS	PLAYLISTS_PK	PLAYLIST_ID	primary key
47	PLAYLISTS	MEMBUAT_PLAYLIST_FK	USER_ID	index FK ke USERS
48	PL_LIBRARY	PL_LIBRARY_PK	USER_ID, PLAYLIST_ID	primary key
49	PL_LIBRARY	RELATIONSHIP_38_FK	USER_ID	index FK ke USERS
50	PL_LIBRARY	RELATIONSHIP_39_FK	PLAYLIST_ID	index FK ke PLAYLISTS
51	RATE_SONGS	RATE_SONGS_PK	USER_ID, SONG_ID	primary key
52	RATE_SONGS	RATE_FK	SONG_ID	index FK ke SONGS
53	RATE_SONGS	RATE2_FK	USER_ID	index FK ke USERS
54	RELEASES	RELEASES_PK	COLLECTION_ID, ARTIST_ID	primary key
55	RELEASES	MERILIS_FK	COLLECTION_ID	index FK ke COLLECTIONS
56	RELEASES	MERILIS2_FK	ARTIST_ID	index FK ke ARTISTS
57	REVIEWS	REVIEWS_PK	REVIEW_ID	primary key
58	REVIEWS	MEMBUAT_REVIEW_FK	USER_ID	index FK ke USERS

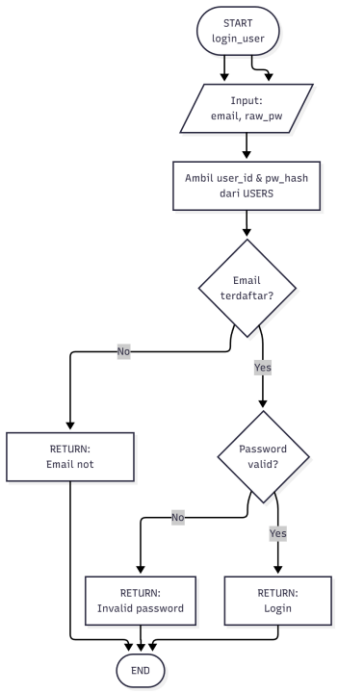
59	REVIEWS	COLLECTION_MANA_FK	COLLECTION_ID	index FK ke COLLECTIONS
60	SOCIALS	SOCIALS_PK	SOCIAL_ID	primary key
61	SOCIALS	HAS_FK	ARTIST_ID	index FK ke ARTISTS
62	SONGS	SONGS_PK	SONG_ID	primary key
63	SONGS_GENRES	SONGS_GENRES_PK	GENRE_ID, SONG_ID	primary key
64	SONGS_GENRES	MEMPUNYAI_FK	GENRE_ID	index FK ke GENRES
65	SONGS_GENRES	MEMPUNYAI2_FK	SONG_ID	index FK ke SONGS
66	TOURS	TOURS_PK	TOUR_ID	primary key
67	USERS	USERS_PK	USER_ID	primary key

4.5 Function/ Procedure

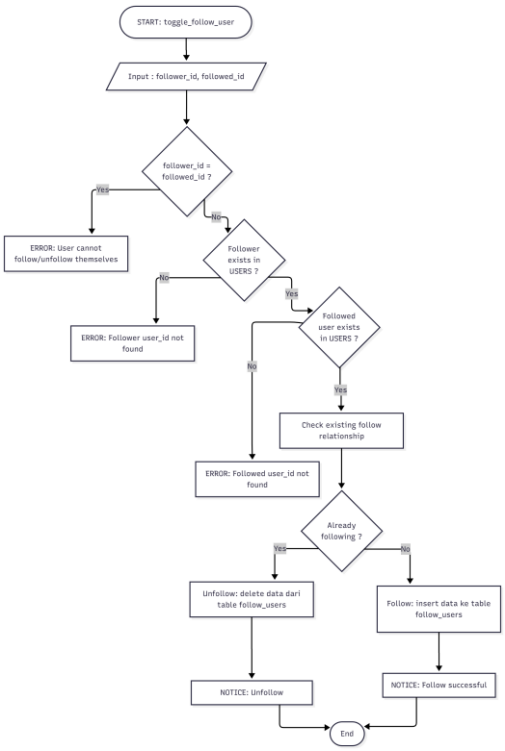
Gambar 4. 4 Flow Chart Function register_user



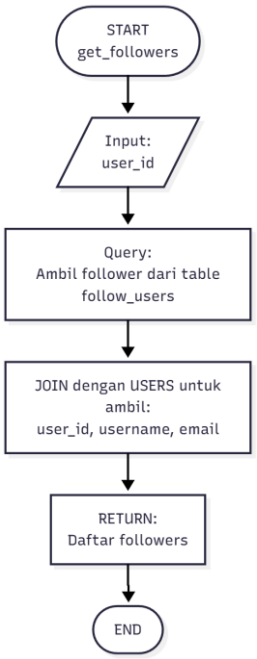
Gambar 4. 5 Flow Chart Function register_user



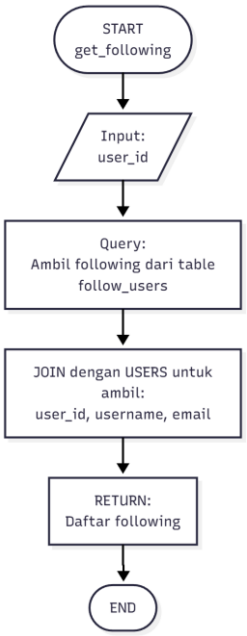
Gambar 4. 6 Flow Chart Function login_user



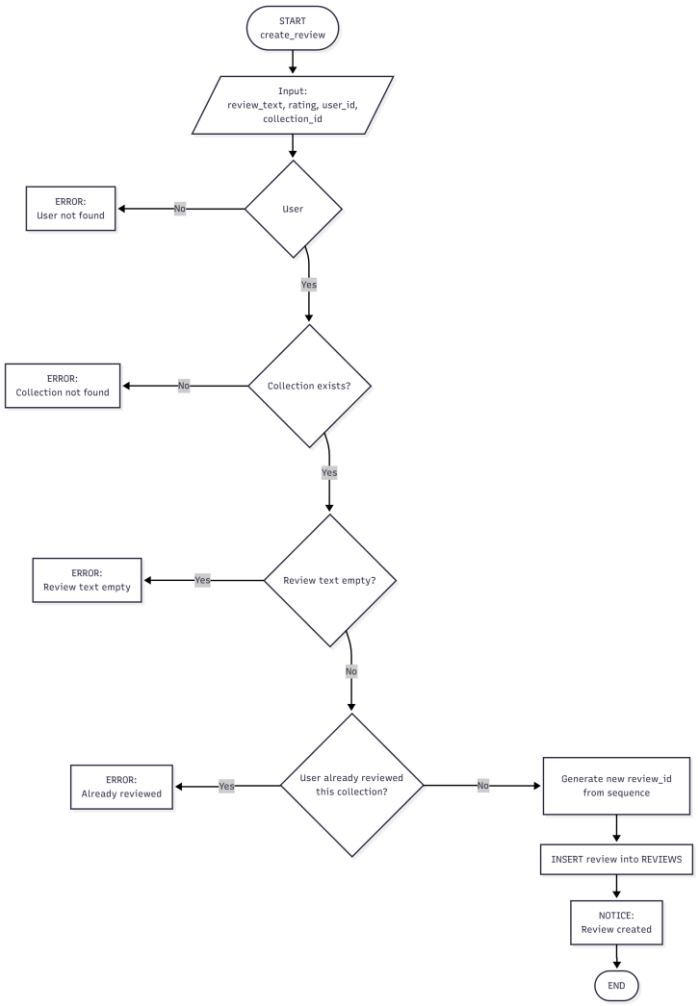
Gambar 4. 7 Flow Chart Procedure toggle_follow_user



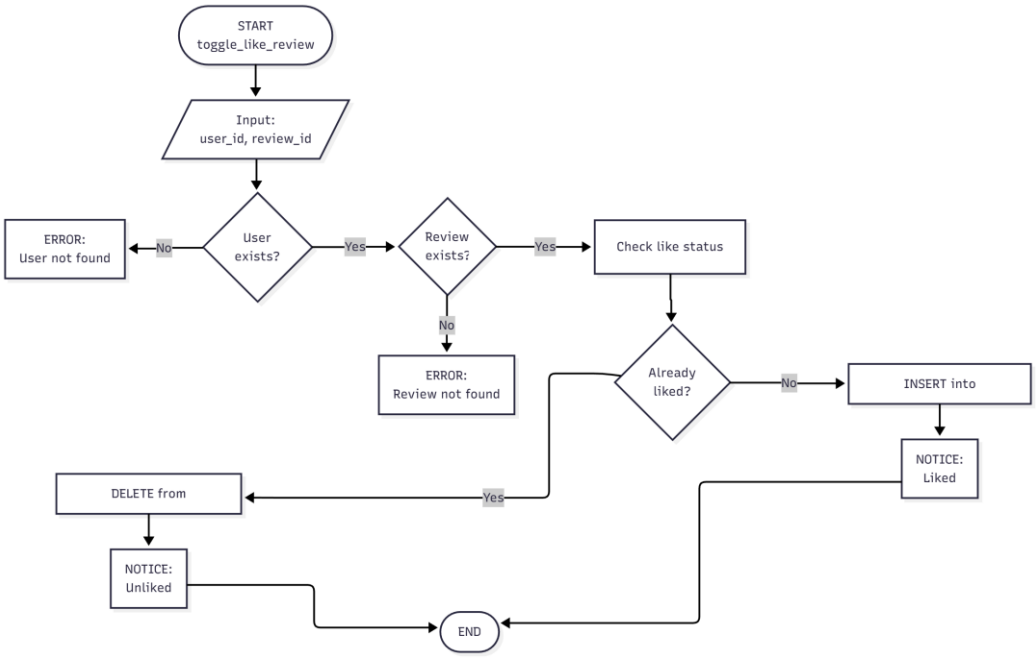
Gambar 4. 8 Flow Chart Function get_followers



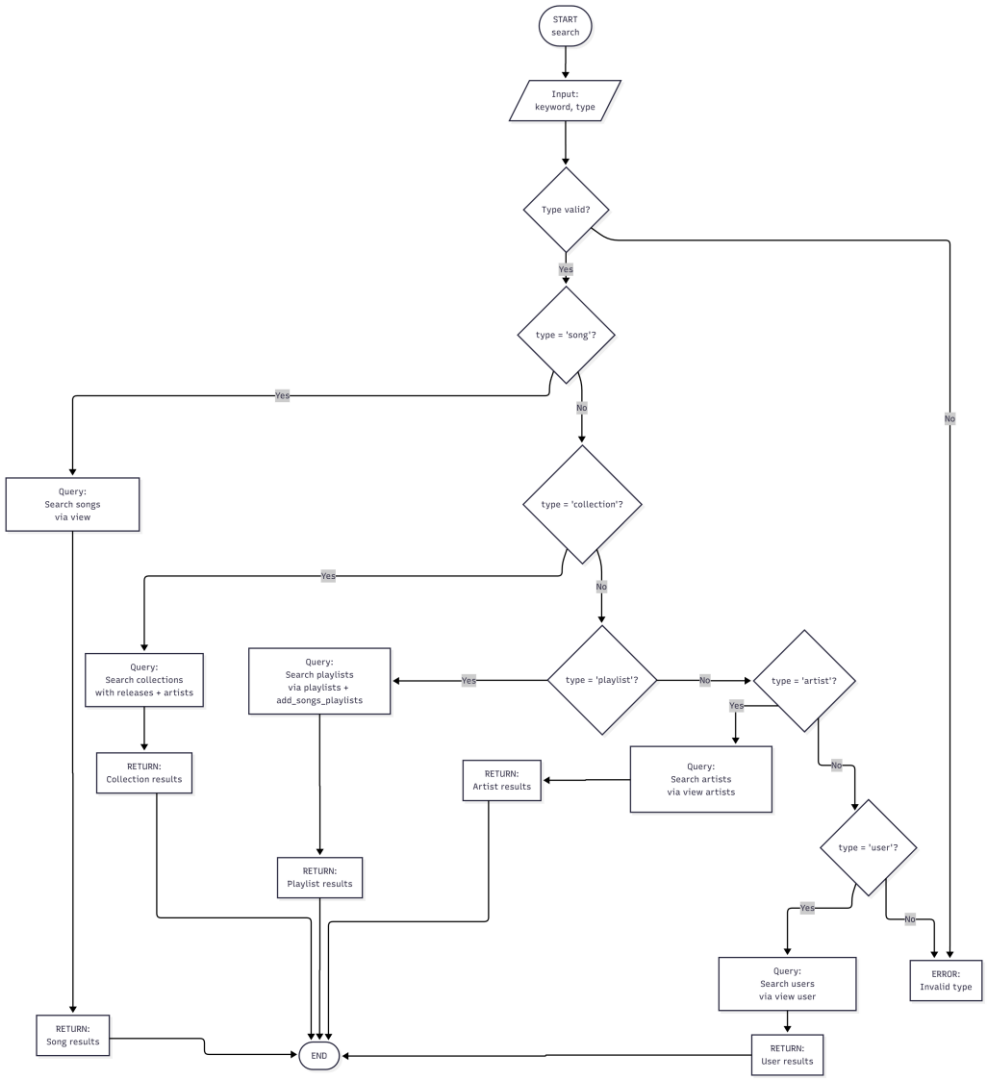
Gambar 4. 9 Flow Chart Function get_following



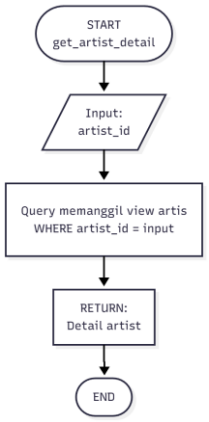
Gambar 4. 10 Flow Chart Procedure create_review



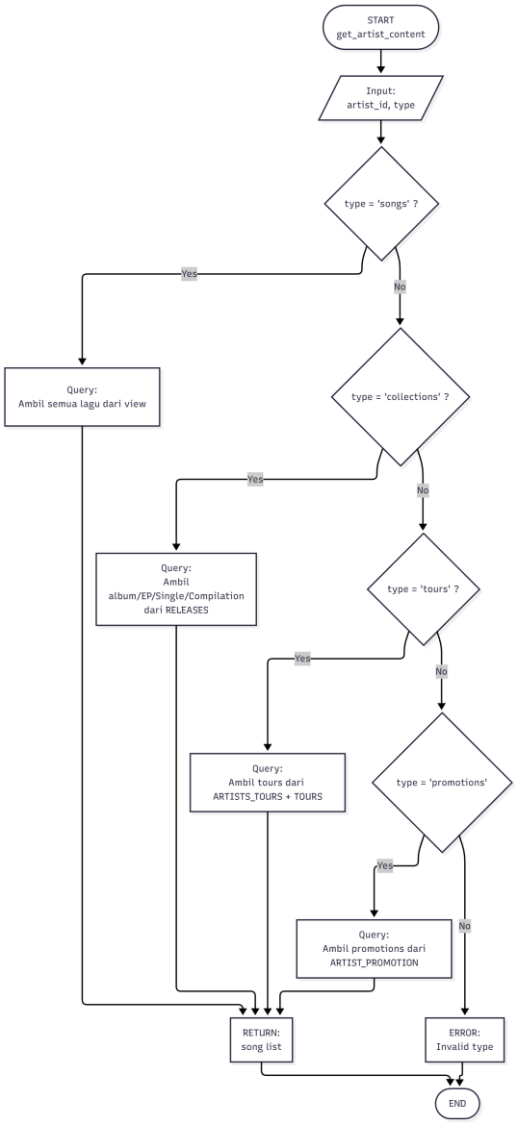
Gambar 4. 11 Flow Chart Procedure toggle_like_review



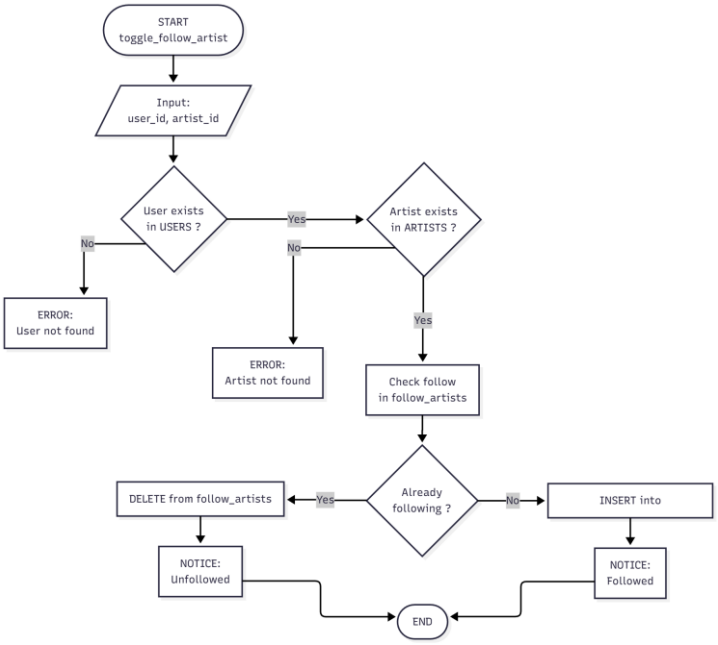
Gambar 4. 12 Flow Chart Function search



Gambar 4. 13 Flow Chart Function get_artist_detail

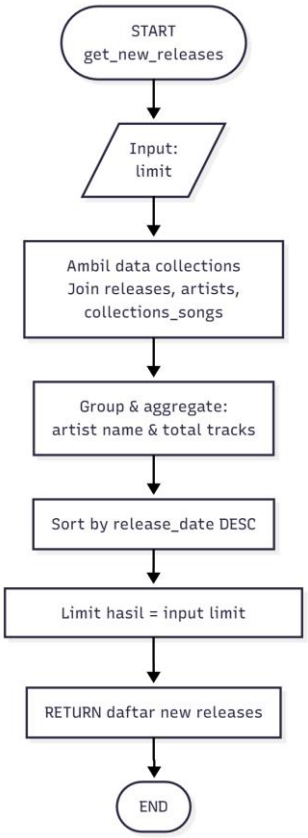


Gambar 4. 14 Flow Chart Function get_artist_content

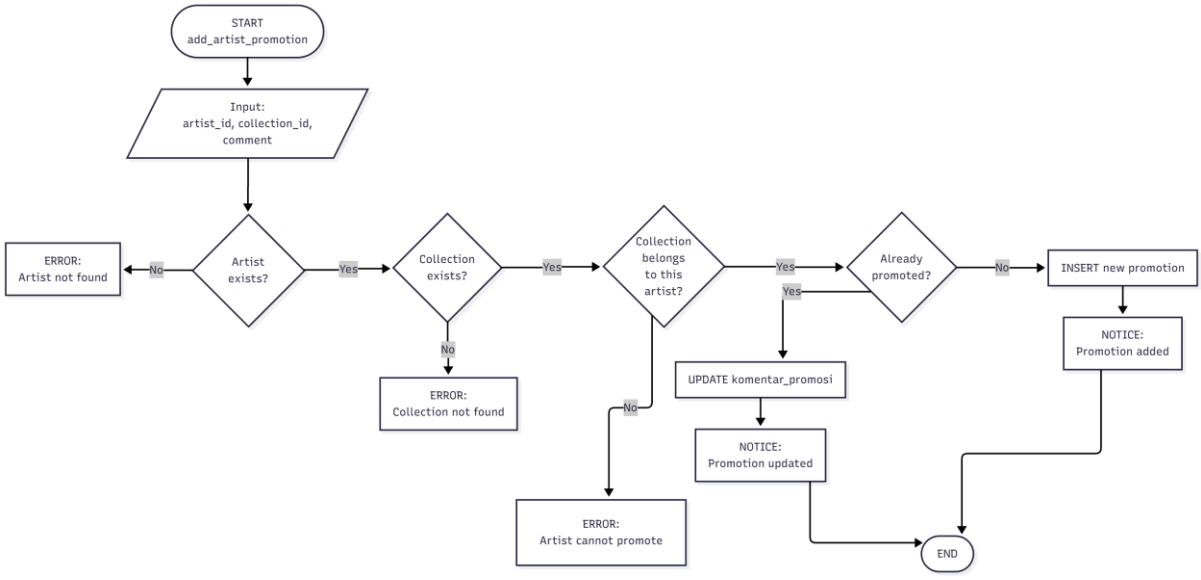


Gambar 4. 15 Flow Chart Procedure toggle_follow_artist

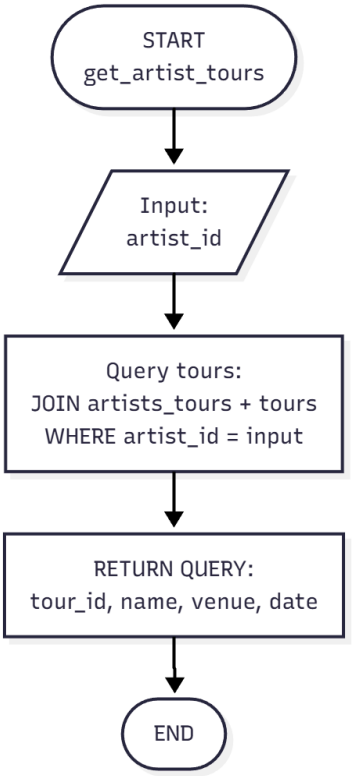
Gambar 4. 16 Flow Chart Function get_collection_tracks



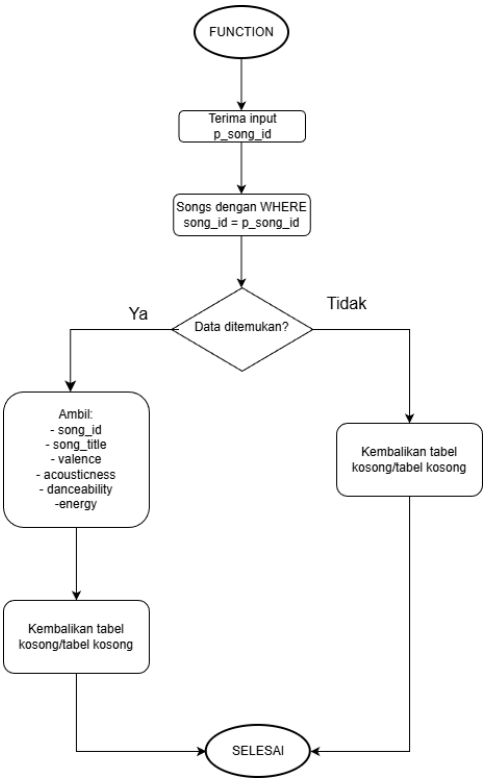
Gambar 4. 17 Flow Chart Function get_new_releases



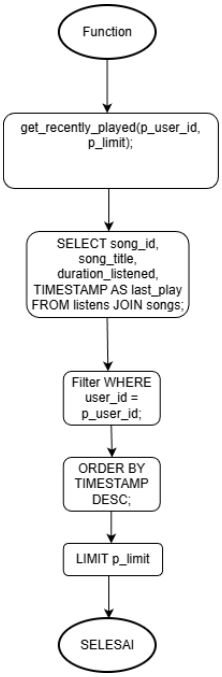
Gambar 4. 18 Flow Chart Procedure add_artist_promotion



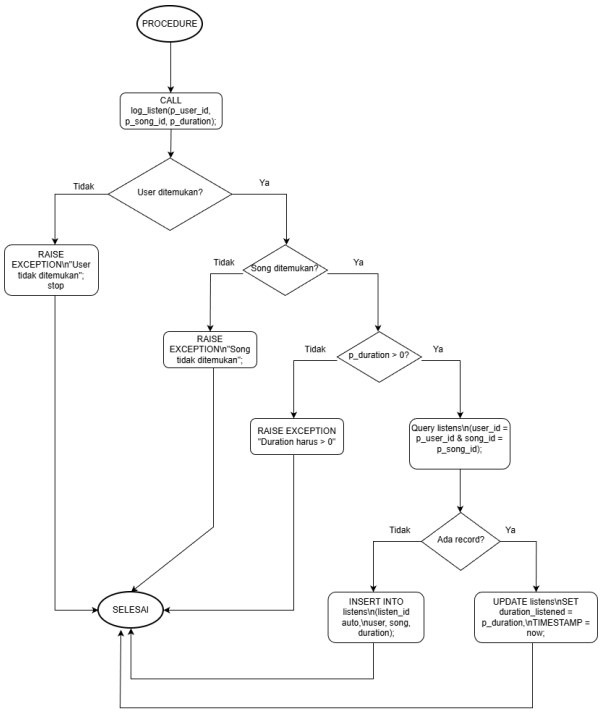
Gambar 4. 19 Flow Chart Function get_artist_tours



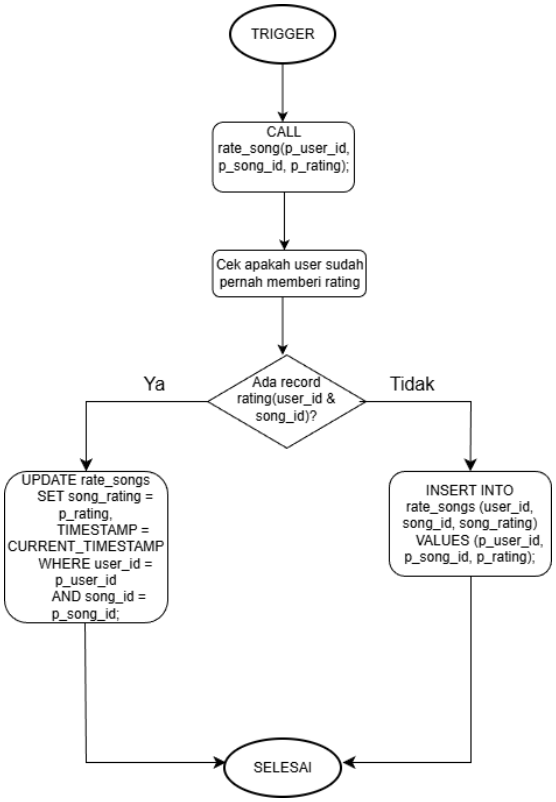
Gambar 4. 20 Flow Chart Function get_song_audio_features



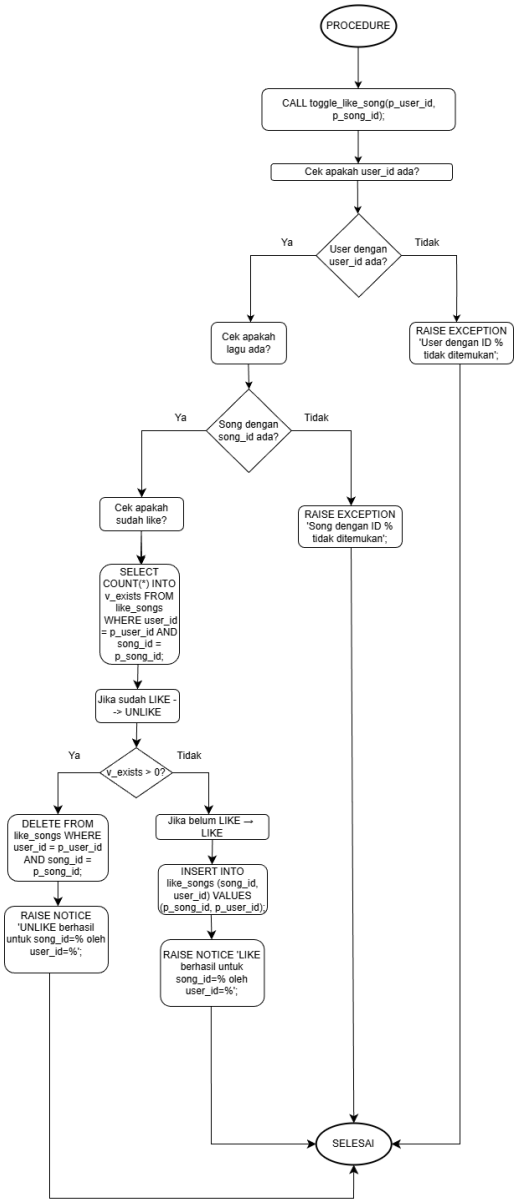
Gambar 4. 21 Flow Chart Function get_recently_played



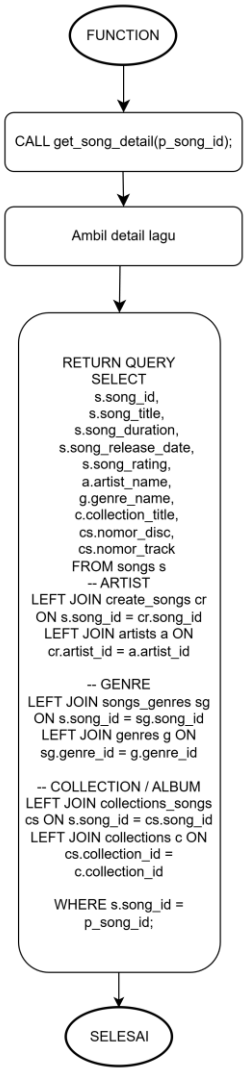
Gambar 4. 22 Flow Chart Procedure log_listen



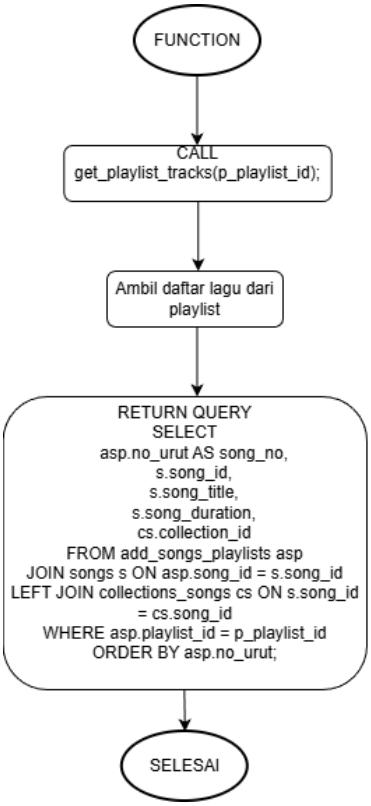
Gambar 4. 23 Flow Chart Procedure rate_song



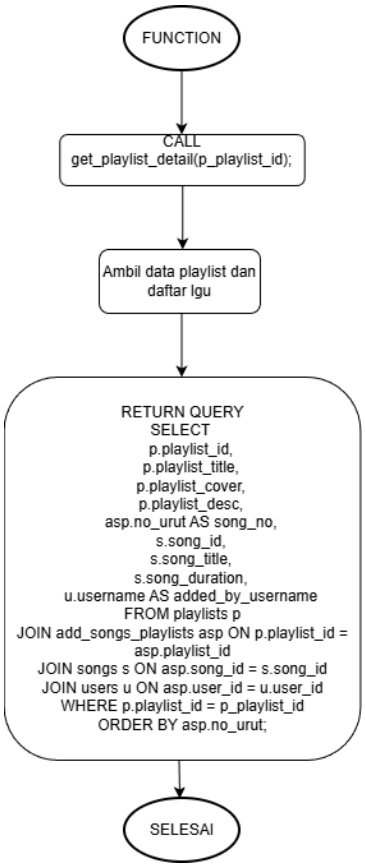
Gambar 4. 24 Flow Chart Procedure toggle_like_song



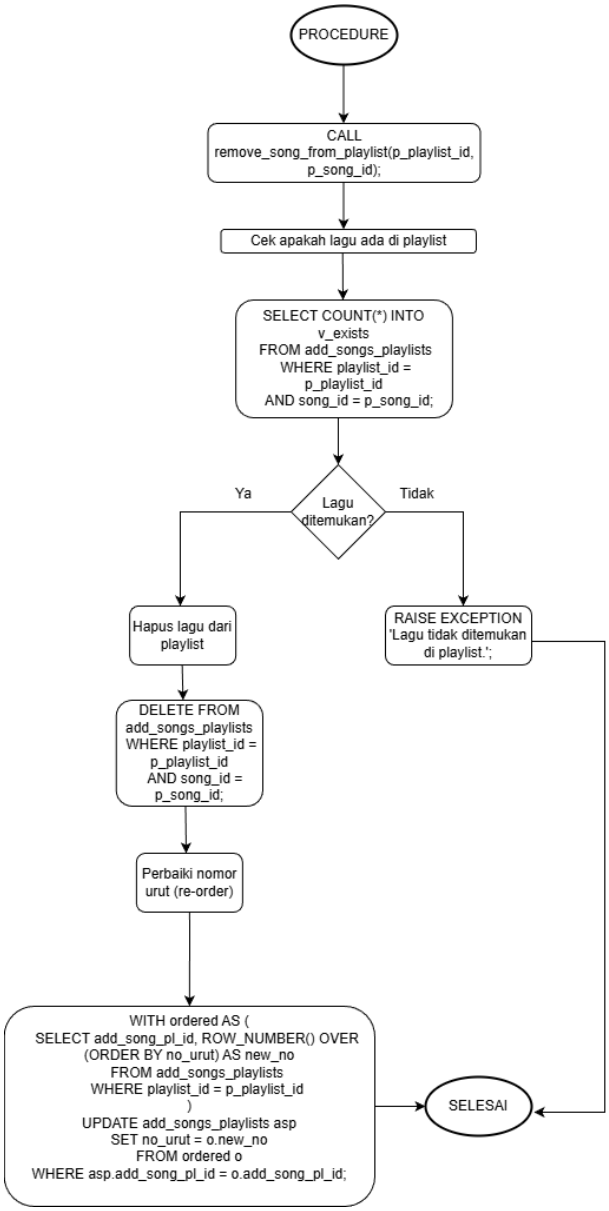
Gambar 4. 25 Flow Chart Function get_song_detail



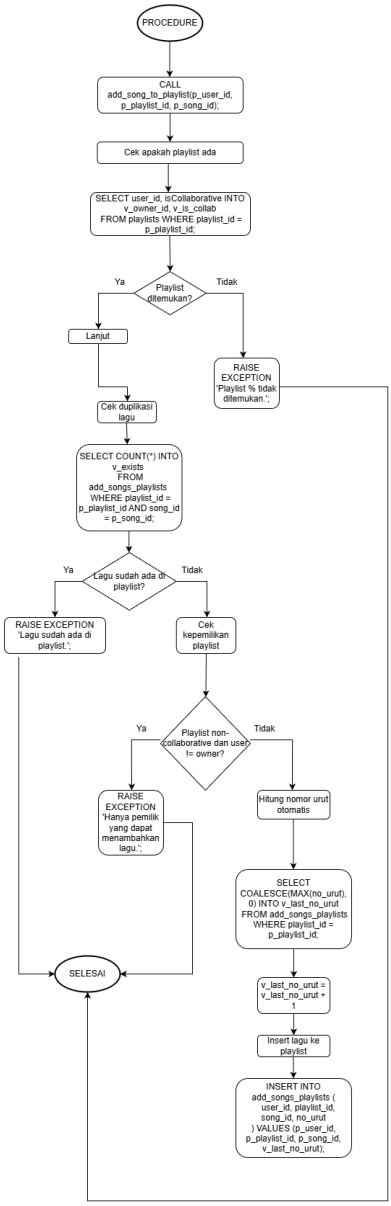
Gambar 4. 26 Flow Chart Function get_playlist_tracks



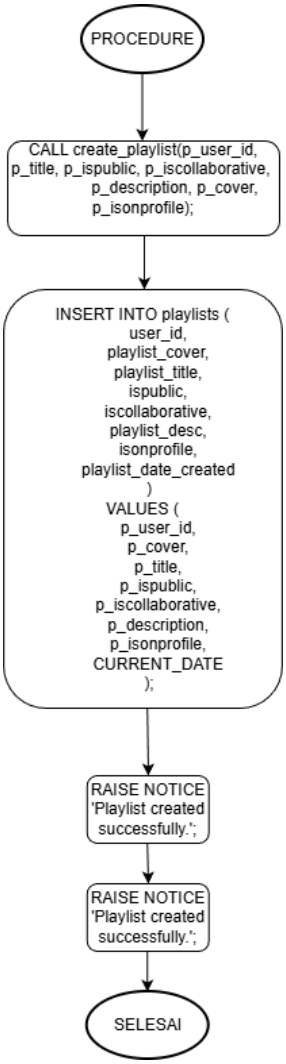
Gambar 4. 27 Flow Chart Function get_playlist_detail



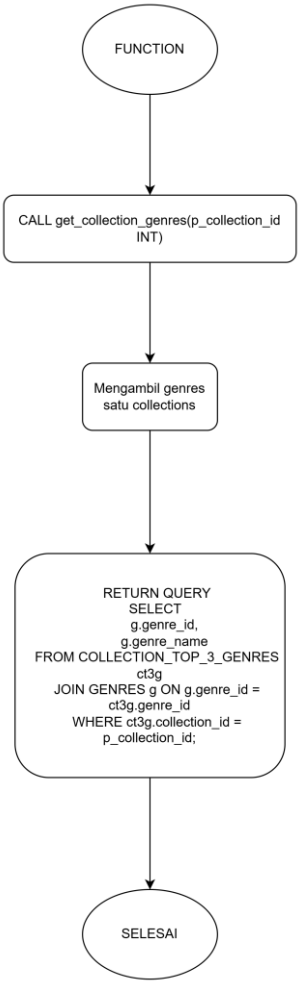
Gambar 4. 28 Flow Chart Procedure remove_song_from_playlist



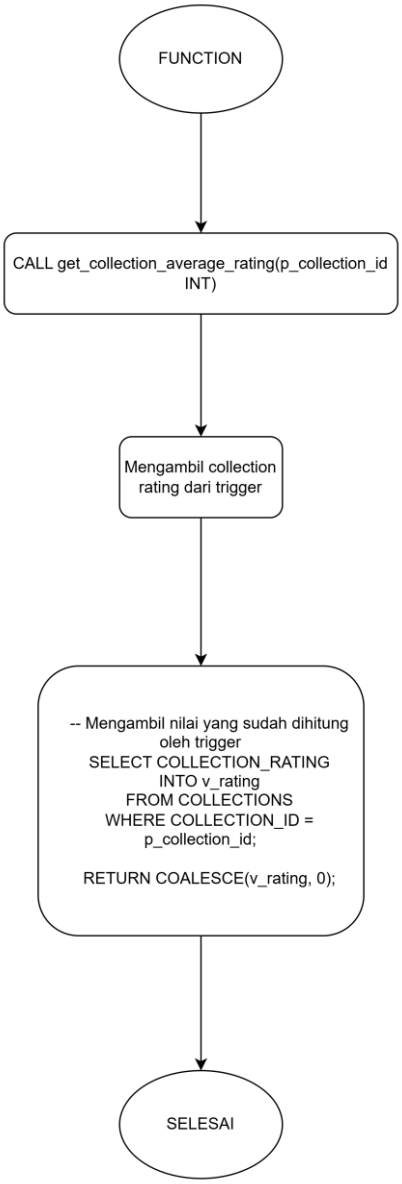
Gambar 4. 29 Flow Chart Procedure add_song_to_playlist



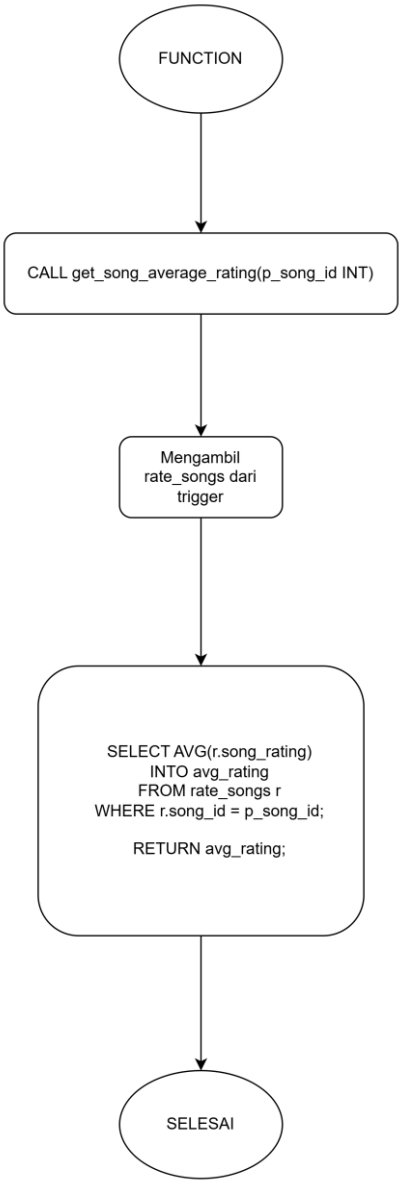
Gambar 4. 30 Flow Chart Procedure create_playlist



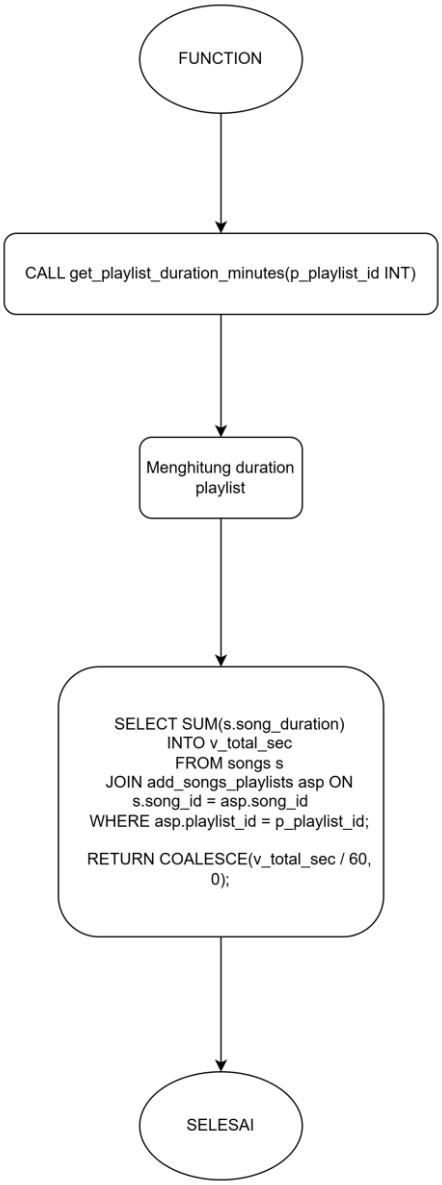
Gambar 4. 31 Flow Chart Function get_collection_genres



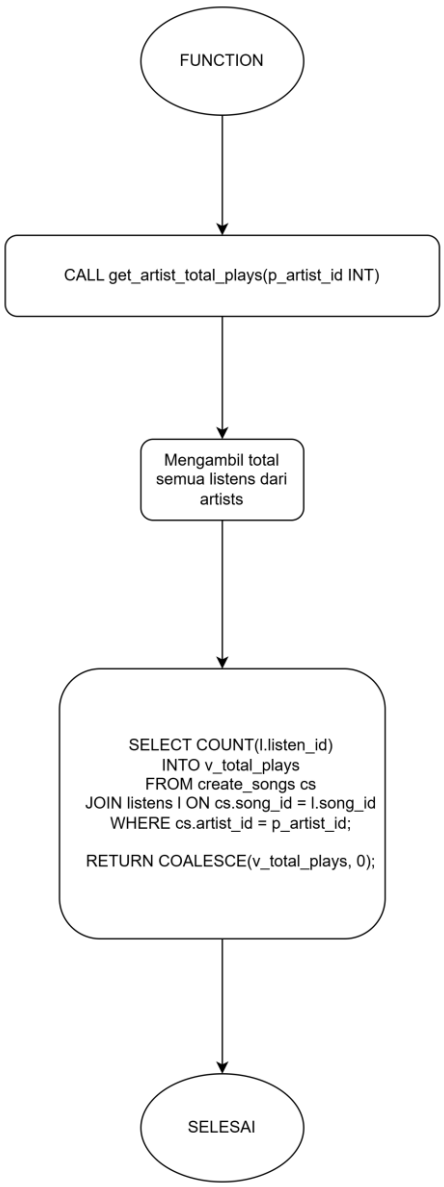
Gambar 4. 32 Flow Chart Function get_collection_average_rating



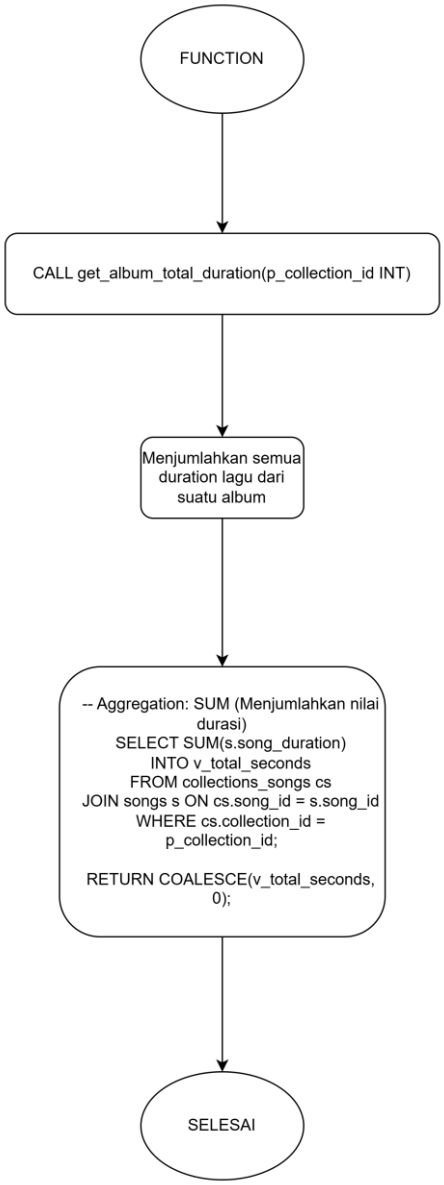
Gambar 4. 33 Flow Chart Function get_song_average_rating



Gambar 4. 34 Flow Chart Function get_playlist_duration_minutes

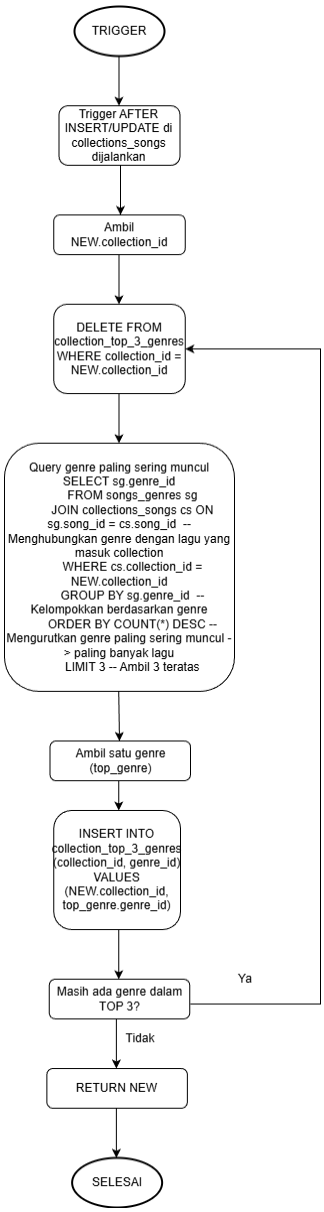


Gambar 4. 35 Flow Chart Function get_artist_total_plays

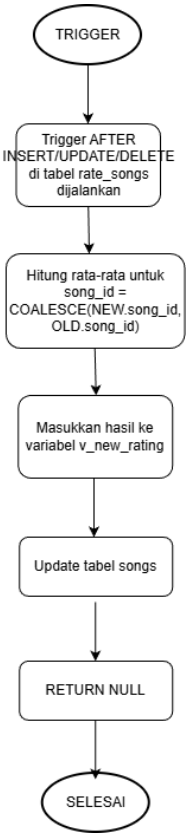


Gambar 4. 36 Flow Chart Function get_album_total_duration

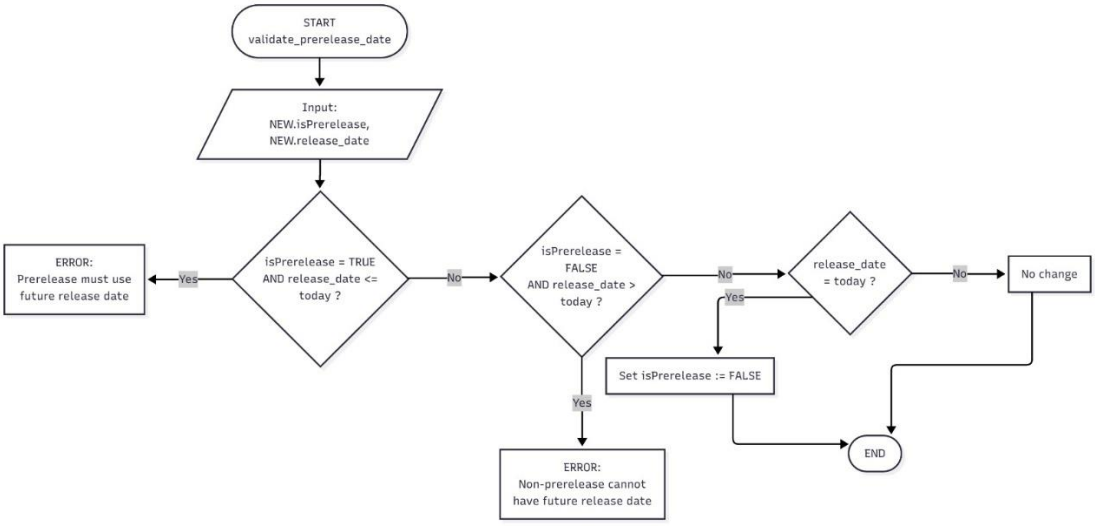
4.6 Trigger



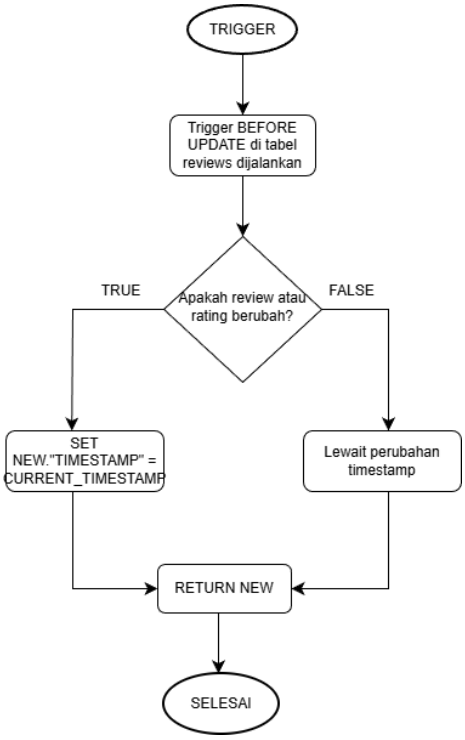
Gambar 4. 37 Flow Chart Trigger update_collection_top3_genres



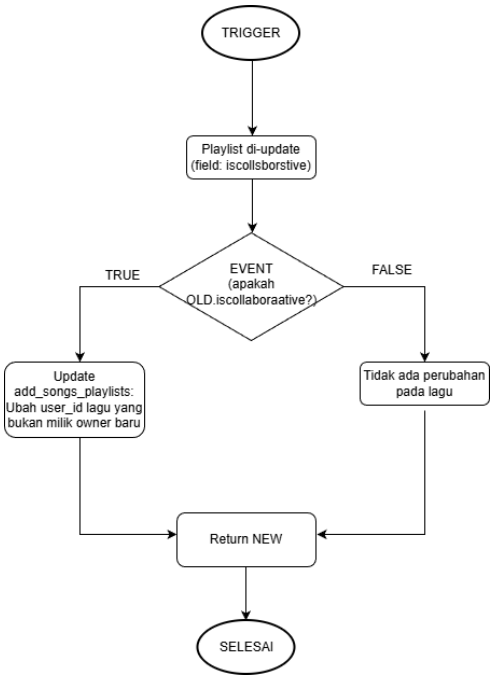
Gambar 4. 38 Flow Chart Trigger update_song_rating



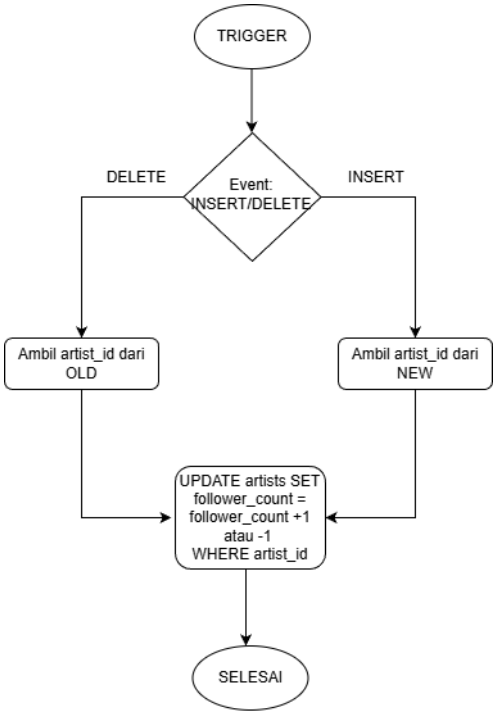
Gambar 4. 39 Flow Chart Trigger trg_validate_prerelease_date



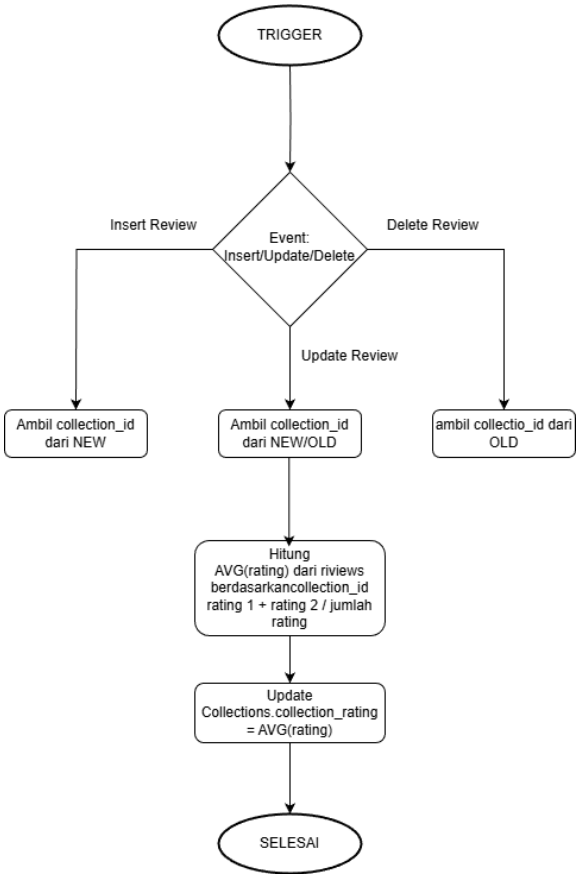
Gambar 4. 40 Flow Chart Trigger update_review_timestamp



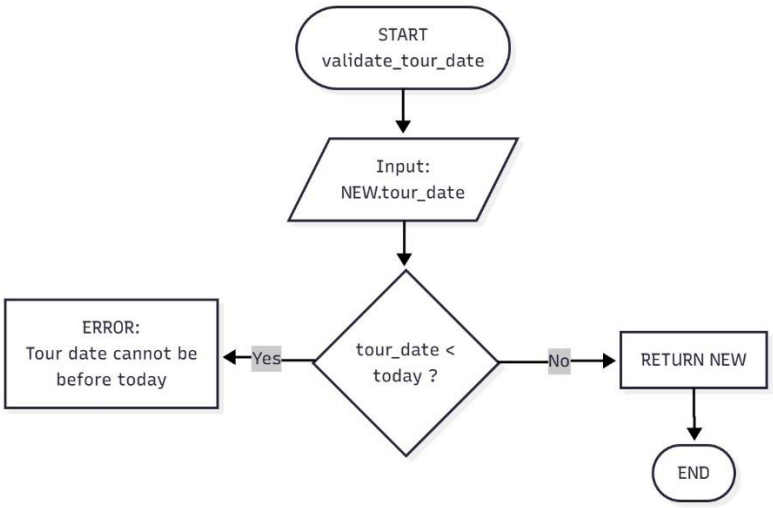
Gambar 4. 41 Flow Chart Trigger trg_fix_playlist_collaborators



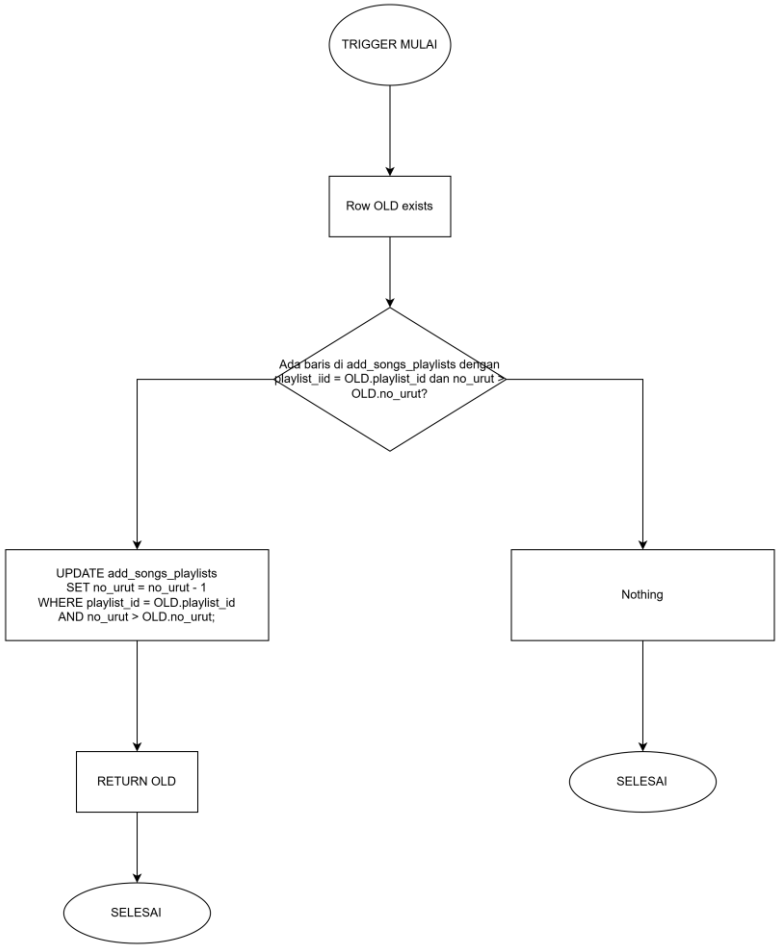
Gambar 4. 42 Flow Chart Trigger trg_update_artist_follow_count



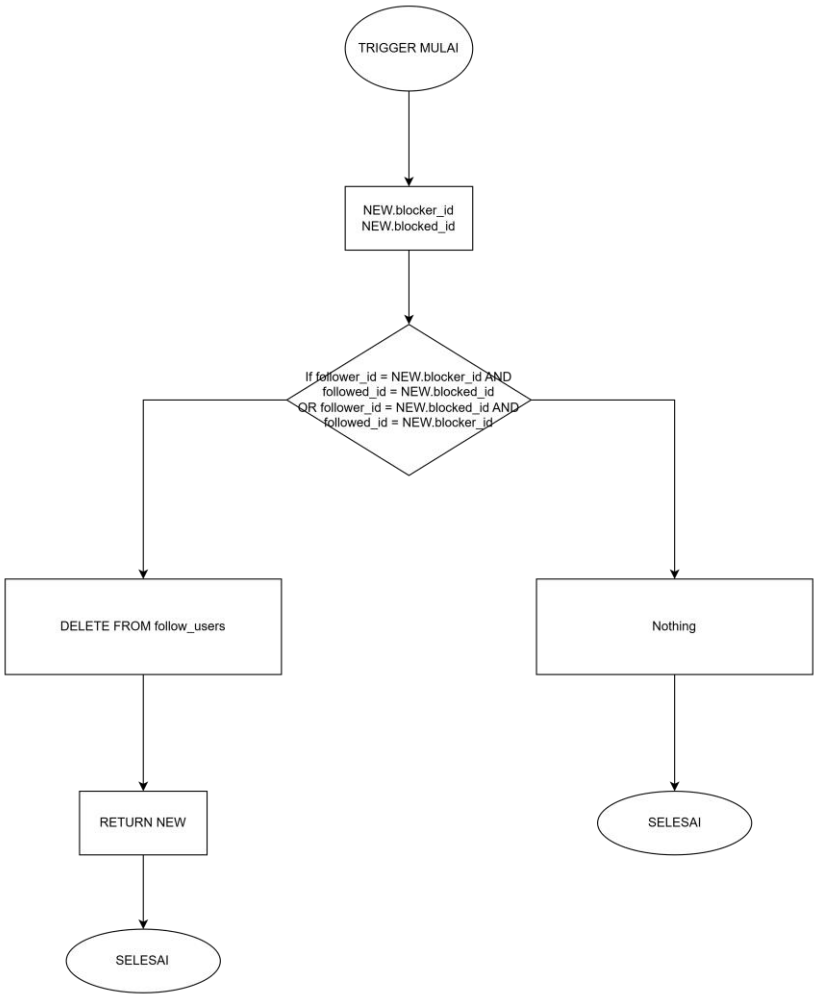
Gambar 4. 43 Flow Chart Trigger trg_update_collection_rating



Gambar 4. 44 Flow Chart Trigger trg_validate_tour_date



Gambar 4. 45 Flow Chart Trigger reorder_playlist_sequence



Gambar 4. 46 Flow Chart Trigger enforce_block_logic

4.7 View

Tabel 31 View Album Tracklist

Nama	VIEW_ALBUM_TRACKLIST	
Deskripsi	View ini menampilkan daftar lagu dalam sebuah collection beserta artis album, artis lagu, urutan track, dan durasi yang sudah diformat. Digunakan untuk menampilkan tracklist album secara lengkap tanpa duplikasi.	
No.	**Nama Kolom**	**Tipe Data**
1.	COLLECTION_ID	INT4
2.	COLLECTION_TITLE	VARCHAR(255)
3.	ALBUM_ARTISTS	TEXT
4.	NOMOR_DISC	INT4

5.	NOMOR_TRACK	INT4
6.	SONG_TITLE	VARCHAR(255)
7.	SONG_ARTISTS	TEXT
8.	DURATION	TEXT

Tabel 32 View Artist Header

Nama	VIEW_ARTIST_PROFILE_HEADER	
Deskripsi	View ini ditujukan untuk memudahkan pengambilan data header artis termasuk data dasar, total album, total lagu, follower count, dan monthly listener.	
No.	**Nama Kolom**	**Tipe Data**
1.	ARTIST_ID	INT4
2.	ARTIST_NAME	VARCHAR(255)
3.	BIO	TEXT
4.	ARTIST_PFP	VARCHAR(2048)
5.	BANNER	VARCHAR(2048)
6.	ARTIST_EMAIL	VARCHAR(320)
7.	MONTHLY_LISTENER_COUNT	INT8
8.	FOLLOWER_COUNT	INT8
9.	TOTAL_ALBUMS	INT8
10.	TOTAL_TRACKS	INT8

Tabel 33 View Full Song Details

Nama	VIEW_FULL_SONG_DETAILS	
Deskripsi	View ini ditujukan untuk memudahkan pengambilan data lagu termasuk collection-collection apa saja yang lagu terdapat didalamnya, sehingga terdapat duplikat lagu dengan asosiasi collection yang berbeda.	
No.	**Nama Kolom**	**Tipe Data**

1.	SONG_ID	INT4
2.	SONG_TITLE	VARCHAR(255)
3.	ARTISTS_NAME	TEXT
4.	ALBUM_NAME	VARCHAR(255)
5.	SONG_DURATION	INT4
6.	POPULARITY	NUMERIC(3,0)
7.	SONG_FILE	VARCHAR(320)
8.	DURATION_FORMATTED	TEXT

Tabel 34 View Top Charts

Nama	VIEW_TOP_CHARTS	
Deskripsi	View ini ditujukan untuk memudahkan pengambilan data lagu-lagu diurutkan berdasarkan popularitas. Berbeda dengan VIEW_FULL_SONG_DETAILS, disini lagu tidak akan duplikat, supaya urutan sesuai.	
No.	**Nama Kolom**	**Tipe Data**
1.	RANK	INT8
2.	SONG_TITLE	VARCHAR(255)
3.	ARTISTS	TEXT
4.	ALBUM	VARCHAR(255)
5.	POPULARITY	NUMERIC(3,0)

Tabel 35 View User Library Stats

Nama	VIEW_USER_LIBRARY_STATS	
Deskripsi	View ini ditujukan untuk memudahkan pengambilan data user termasuk jumlah follower, jumlah following user dan artis, dan jumlah playlist public	
No.	**Nama Kolom**	**Tipe Data**
1.	USER_ID	INT4

2.	USERNAME	VARCHAR(50)
3.	USER_PFP	VARCHAR(2048)
4.	PUBLIC_PLAYLIST	INT8
5.	FOLLOWING_COUNT	INT8
6.	FOLLOWERS_COUNT	INT8

4.8 Cron

Cron adalah mekanisme task scheduling yang memungkinkan sistem basis data menjalankan perintah secara otomatis pada waktu tertentu tanpa interaksi pengguna. Dalam sistem ini, cron digunakan melalui pg_cron, sebuah ekstensi PostgreSQL yang memungkinkan penjadwalan fungsi PL/pgSQL secara periodik (harian, mingguan, atau interval tertentu). Ekstensi ini tidak tersedia di Sistem Operasi Windows, tetapi tersedia di Sistem Operasi Linux, sehingga sangat cocok digunakan pada lingkungan server database yang umumnya berjalan pada sistem operasi Linux. Di Arch Linux, ada package di AUR (Arch User Repository) bernama 'pg_cron'.

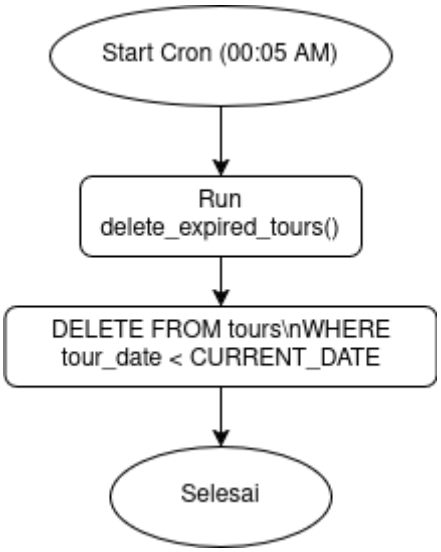
Secara default, pg_cron menggunakan timezone UTC, bukan timezone lokal server. Artinya, jika server berada di zona waktu Asia/Jakarta (UTC+7), maka job yang dijadwalkan pada pukul 08:00 pagi harus dikonversi menjadi 01:00 UTC.

Penggunaan cron penting untuk proses-proses yang bersifat time-driven, yaitu proses yang harus berjalan berdasarkan waktu, bukan berdasarkan event database seperti INSERT, UPDATE, atau DELETE. Oleh karena itu, cron tidak dapat digantikan oleh trigger.

Dalam perancangan sistem layanan musik ini, terdapat beberapa proses otomatis yang tidak dapat mengandalkan trigger, karena harus dijalankan secara berkala berdasarkan tanggal dan waktu tertentu. Contoh proses tersebut:

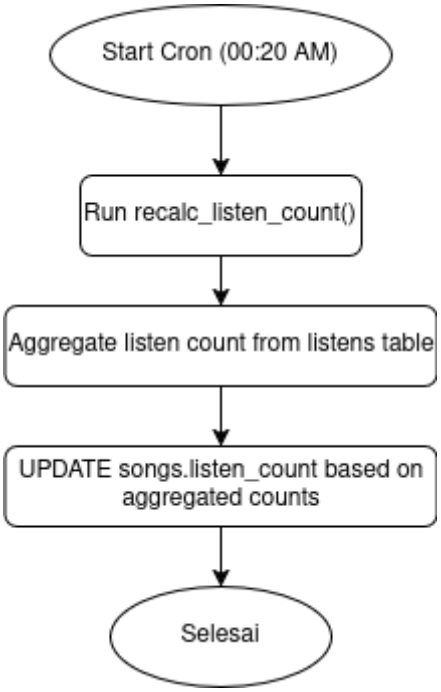
1. Menghapus tour yang sudah lewat tanggalnya.
2. Menonaktifkan status prerelease pada collection yang hari rilisnya sudah tiba.
3. Menghitung ulang metrik tertentu secara periodik seperti popularitas global, monthly listeners, atau aggregasi data lain yang tidak perlu dihitung real-time.

Cron memastikan data tetap konsisten dan bersih meskipun tidak ada aktivitas user terhadap tabel terkait.



Gambar 4. 47 Flow Chart Cron delete_expired_tours_daily

Cronjob ini dijalankan setiap hari pukul 00:05. Fungsinya adalah menghapus semua entry tour yang tanggalnya sudah lewat hari ini (tour_date < CURRENT_DATE). Karena trigger tidak dapat berjalan berdasarkan waktu, cronjob ini memastikan bahwa daftar tour tetap bersih dan hanya menampilkan tour yang masih relevan. Jika ada tour yang sudah lewat, sistem akan otomatis menghapusnya tanpa perlu intervensi pengguna.



Gambar 4. 48 Flow Chart Cron recalc_listen_count_daily

Cronjob ini berjalan setiap hari pukul 00:20 untuk menghitung ulang total jumlah pemutaran (listen_count) setiap lagu berdasarkan tabel listens.

Pendekatan ini lebih efisien dibanding memperbarui listen_count melalui trigger setiap kali user memutar lagu, karena trigger akan memberikan beban tinggi pada transaksi insert. Dengan cronjob, perhitungan dilakukan sekali sehari, tetap akurat, dan jauh lebih efisien.

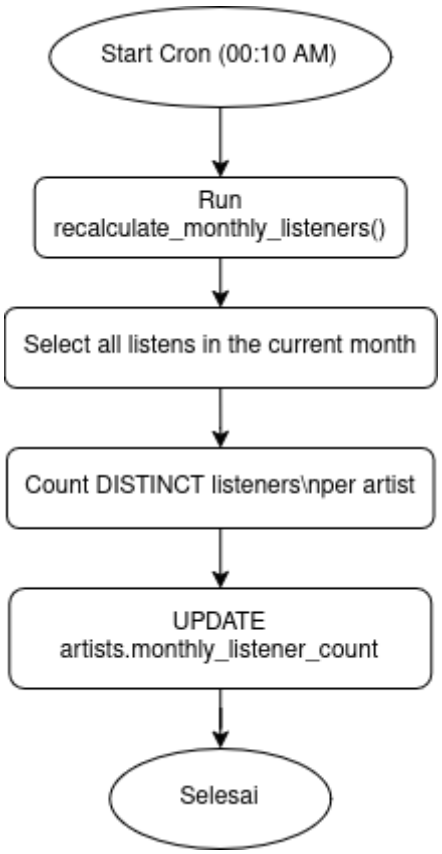
Gambar 4. 49 Flow Chart Cron recalculate_song_popularity_daily

Cronjob ini menghitung ulang popularitas seluruh lagu setiap hari pada pukul 00:15. Popularitas dihitung menggunakan formula berikut.

popularitas =

(jumlah_play_30_hari_terakhir / jumlah_play_lagu_terbanyak_di_sistem_30_hari_terakhir) * 100

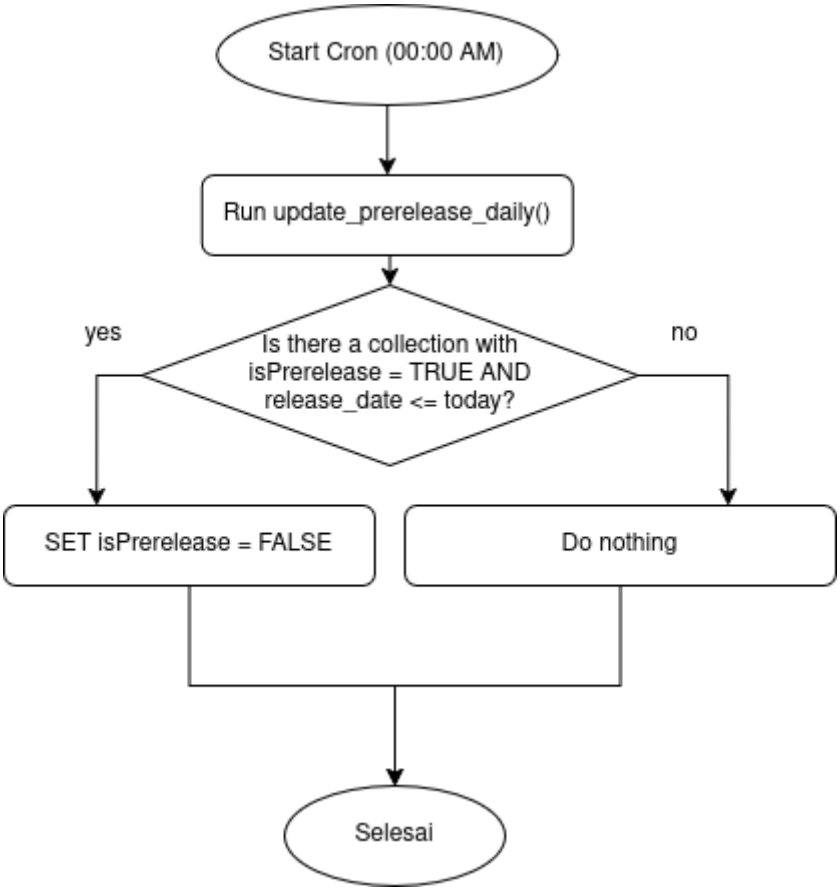
Cronjob ini menjaga agar popularitas bersifat dinamis, terbarui, dan stabil, tanpa membebani sistem saat pemutaran terjadi.



Gambar 4. 50 Flow Chart Cron recalculate_monthly_listeners_daily

Cronjob ini memperbarui jumlah monthly listeners setiap artis setiap hari pukul 00:10. Nilai dihitung berdasarkan jumlah user unik (DISTINCT user_id) yang memutar lagu artis dalam bulan berjalan.

Karena perhitungan ini memerlukan agregasi besar dan melibatkan beberapa tabel (listens dan create_songs), menjalankannya melalui trigger akan sangat tidak efisien. Cronjob menjadi solusi optimal untuk melakukan pembaruan terjadwal yang ringan bagi sistem.



Gambar 4. 51 Flow Chart Cron update_prerelease_job_daily

Cronjob ini dijalankan setiap hari tepat pukul 00:00, berfungsi untuk menonaktifkan status prerelease dari koleksi (album/EP/single) yang tanggal rilisnya telah tiba. Sistem secara otomatis mengecek apakah collection_release_date sudah sama atau lebih kecil dari CURRENT_DATE, kemudian mengubah isPrerelease menjadi FALSE.

Cronjob ini sangat penting untuk menghindari koleksi yang tetap tampil sebagai “belum dirilis” padahal sudah melewati hari rilisnya.

5. IMPLEMENTASI PENGOLAHAN DATA

5.1 DDL

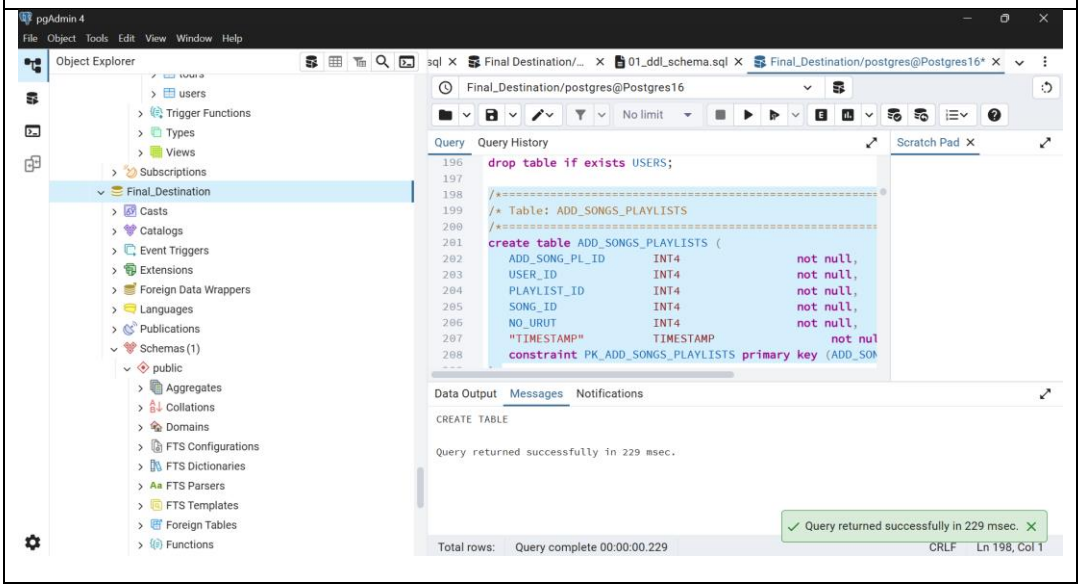
Tabel 36 DDL – Tabel add_songs_playlist

Nama Tabel	add_songs_playlist
Deskripsi	Tabel ini berfungsi sebagai hubungan menambahkan lagu apa ke suatu playlist

Script SQL

```
/*==============================================================*/
/* Table: ADD_SONGS_PLAYLISTS                                   */
/*==============================================================*/
create table ADD_SONGS_PLAYLISTS (
   ADD_SONG_PL_ID       INT4                 not null,
   USER_ID              INT4                 not null,
   PLAYLIST_ID          INT4                 not null,
   SONG_ID              INT4                 not null,
   NO_URUT              INT4                 not null,
   "TIMESTAMP"          DATE                 not null,
   constraint PK_ADD_SONGS_PLAYLISTS primary key (ADD_SONG_PL_ID)
);
```

Screenshot Hasil



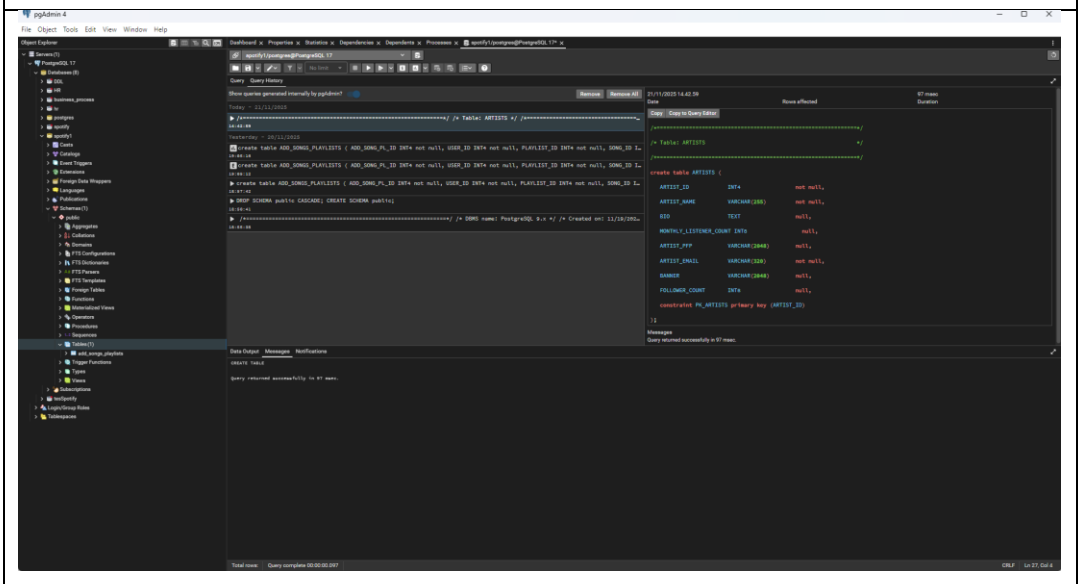
Tabel 37 DDL – Tabel Artists

Nama Tabel	Artists
Deskripsi	Tabel untuk menyimpan data artis, termasuk nama, foto profil, banner, email, jumlah pendengar, dan koleksi (album/EP) yang terkait.

Script SQL

```
/*==============================================================*/
/* Table: ARTISTS                                               */
/*==============================================================*/
create table ARTISTS (
   ARTIST_ID            INT4                 not null,
   ARTIST_NAME          VARCHAR(255)         not null,
   BIO                  TEXT                 null,
   MONTHLY_LISTENER_COUNT INT8                 null default 0,
   ARTIST_PFP           VARCHAR(2048)        null,
   ARTIST_EMAIL         VARCHAR(320)         not null unique,
   BANNER               VARCHAR(2048)        null,
   FOLLOWER_COUNT       INT8                 null default 0,
   constraint PK_ARTISTS primary key (ARTIST_ID)
);
```

Screenshot Hasil



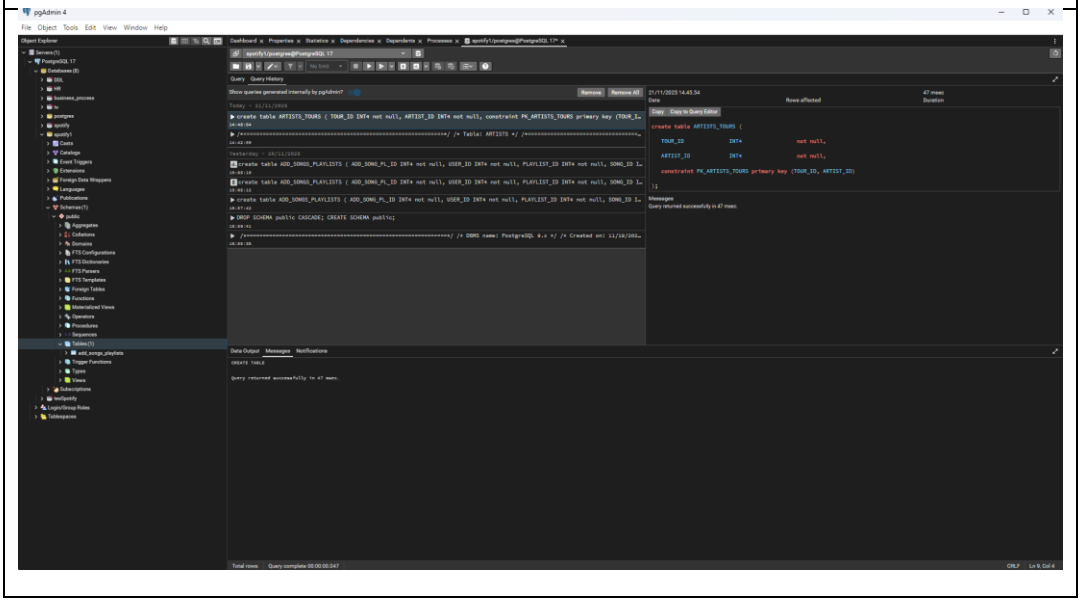
Tabel 38 DDL – Tabel Artists_tours

Nama Tabel	Artists_tours
Deskripsi	Tabel ini menyimpan hubungan antara tour dan artists

Script SQL

```
/*==============================================================*/
/* Table: ARTISTS_TOURS                                         */
/*==============================================================*/
create table ARTISTS_TOURS (
   TOUR_ID              INT4                 not null,
   ARTIST_ID            INT4                 not null,
   constraint PK_ARTISTS_TOURS primary key (TOUR_ID, ARTIST_ID)
);
```

Screenshot Hasil



Tabel 39 DDL – Tabel Artist_Promotion

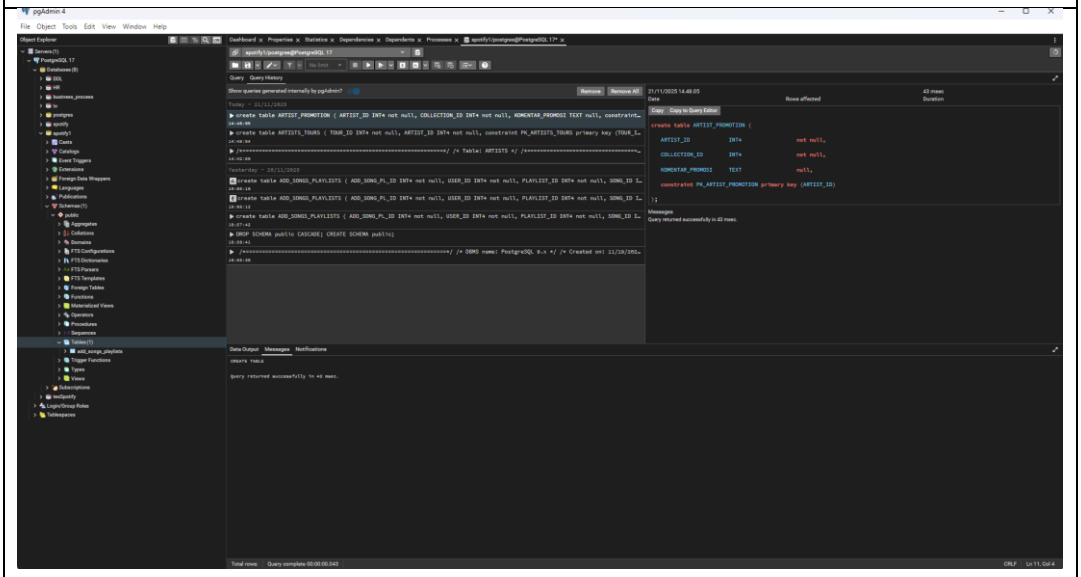
Nama Tabel	Artist_promotion
Deskripsi	Tabel ini menyimpan text promosi dan hubungan tabel artist dan collection

Script SQL

```
/*==============================================================*/
```

```
/* Table: ARTIST_PROMOTION                                      */
/*==============================================================*/
create table ARTIST_PROMOTION (
   ARTIST_ID            INT4                 not null,
   COLLECTION_ID        INT4                 not null,
   KOMENTAR_PROMOSI     TEXT                 null,
   constraint PK_ARTIST_PROMOTION primary key (ARTIST_ID)
);
```

Screenshot Hasil



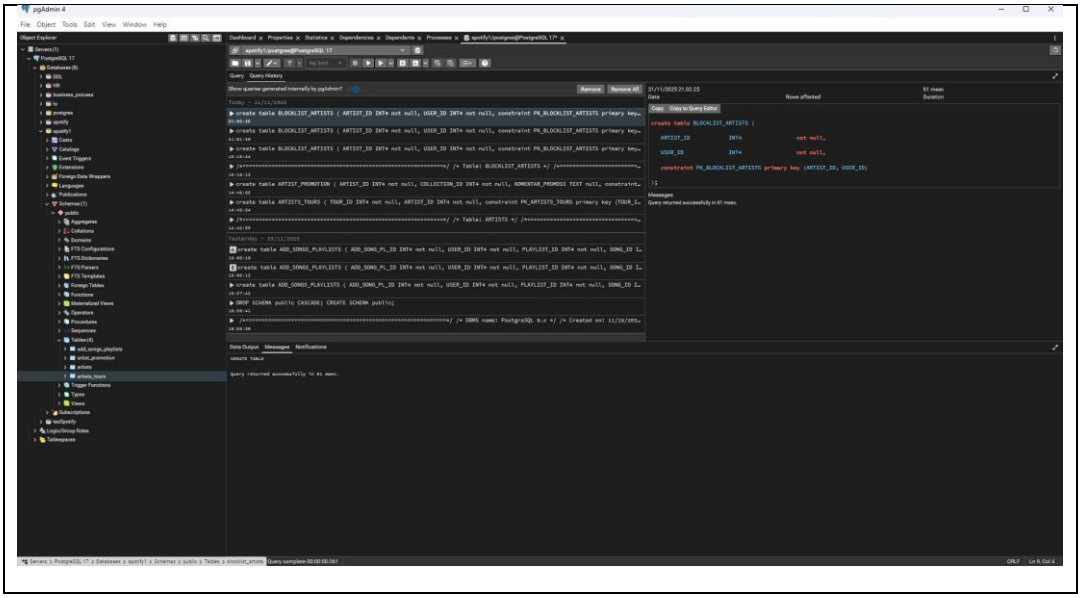
Tabel 40 DDL – Tabel Blocklist_artists

Nama Tabel	Blocklist_artists
Deskripsi	Tabel ini menyimpan hubungan antara artists dan user, untuk siapa saja artist yang di block oleh user

Script SQL

```
/*==============================================================*/
/* Table: BLOCKLIST_ARTISTS                                     */
/*==============================================================*/
create table BLOCKLIST_ARTISTS (
   ARTIST_ID            INT4                 not null,
   USER_ID              INT4                 not null,
   constraint PK_BLOCKLIST_ARTISTS primary key (ARTIST_ID, USER_ID)
);
```

Screenshot Hasil

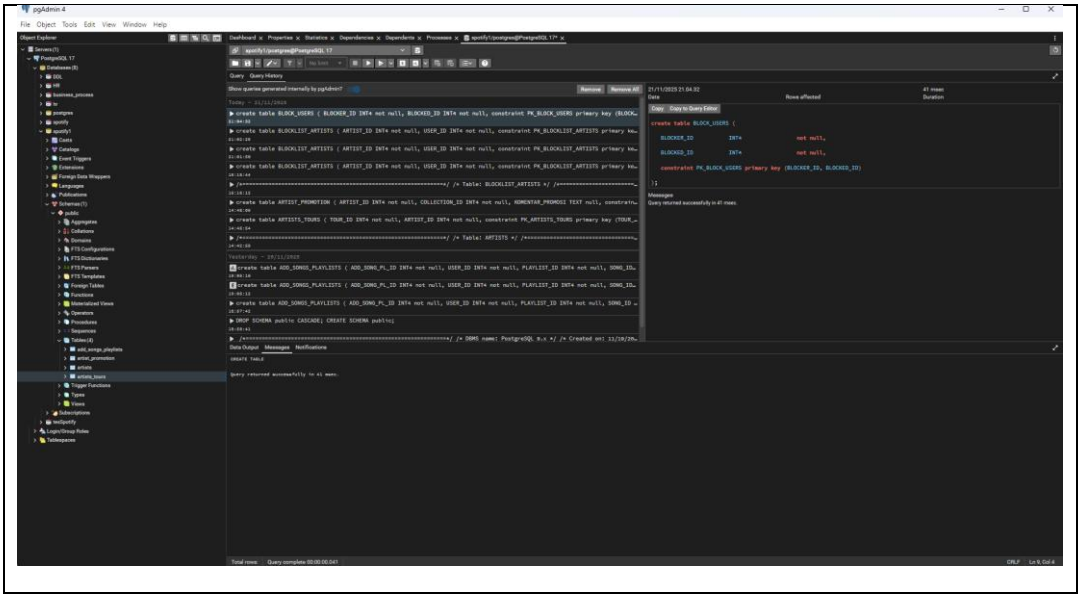


Tabel 41 DDL – Tabel block_users

Nama Tabel	Block_users
Deskripsi	Tabel ini berfungsi untuk menyimpan data user yang diblock dan yang terblock
Script SQL	

```
/*==============================================================*/
/* Table: BLOCK_USERS                                           */
/*==============================================================*/
create table BLOCK_USERS (
   BLOCKER_ID           INT4                 not null,
   BLOCKED_ID           INT4                 not null,
   constraint PK_BLOCK_USERS primary key (BLOCKER_ID, BLOCKED_ID)
);
```

Screenshot Hasil



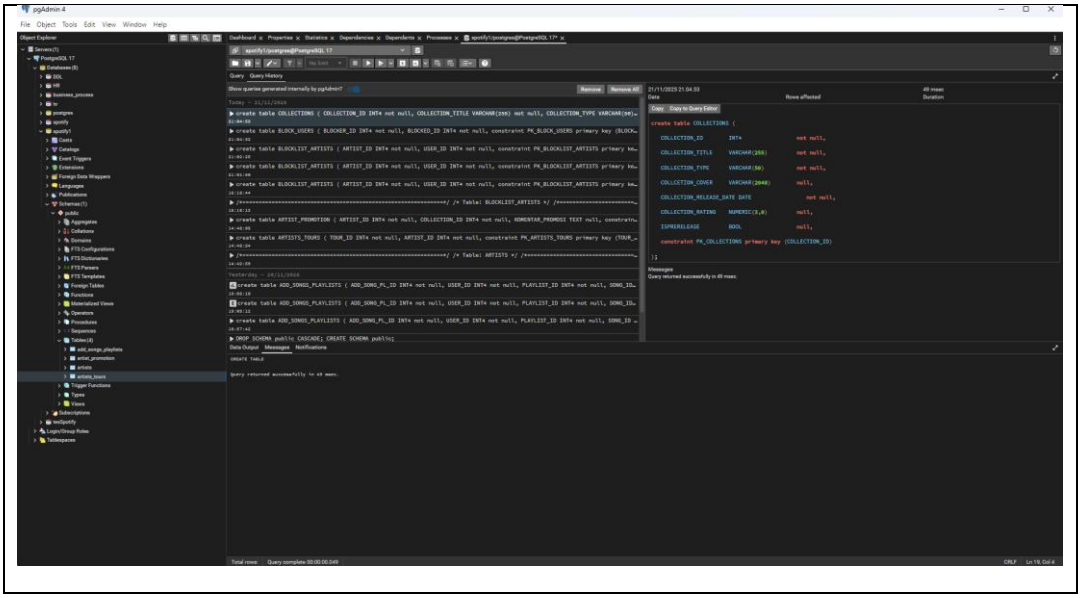
Tabel 42 DDL – Tabel collections

Nama Tabel	collections
Deskripsi	Tabel ini berfungsi untuk menyimpan collections entah itu album

Script SQL

```
/*==============================================================*/
/* Table: COLLECTIONS                                           */
/*==============================================================*/
create table COLLECTIONS (
   COLLECTION_ID        INT4                 not null,
   COLLECTION_TITLE     VARCHAR(255)         not null,
   COLLECTION_TYPE      VARCHAR(50)          not null,
   COLLCETION_COVER     VARCHAR(2048)        null,
   COLLECTION_RELEASE_DATE DATE                 not null,
   COLLECTION_RATING    NUMERIC(3,0)         null,
   ISPRERELEASE         BOOL                 null,
   constraint PK_COLLECTIONS primary key (COLLECTION_ID)
);
```

Screenshot Hasil



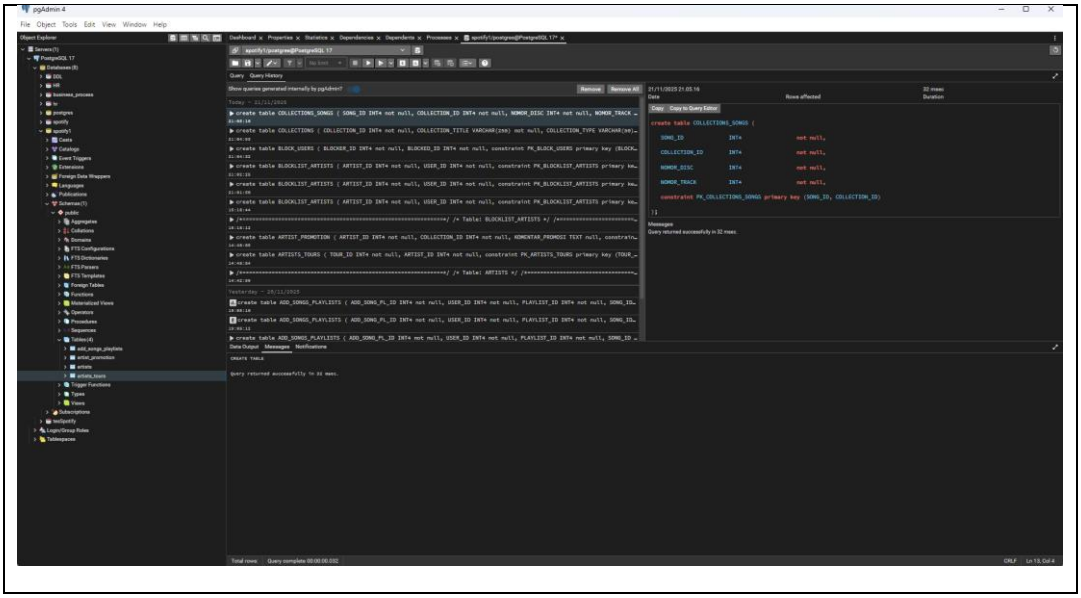
Tabel 43 DDL – Tabel Collections_Songs

Nama Tabel	Collections_songs
Deskripsi	Tabel ini berfungsi untuk menyimpan hubungan antara songs dan collections, beserta dengan nomor disc dan nomor track

Script SQL

```
/*==============================================================*/
/* Table: COLLECTIONS_SONGS                                     */
/*==============================================================*/
create table COLLECTIONS_SONGS (
   SONG_ID              INT4                 not null,
   COLLECTION_ID        INT4                 not null,
   NOMOR_DISC           INT4                 not null,
   NOMOR_TRACK          INT4                 not null,
   constraint PK_COLLECTIONS_SONGS primary key (SONG_ID,
COLLECTION_ID)
);
```

Screenshot Hasil



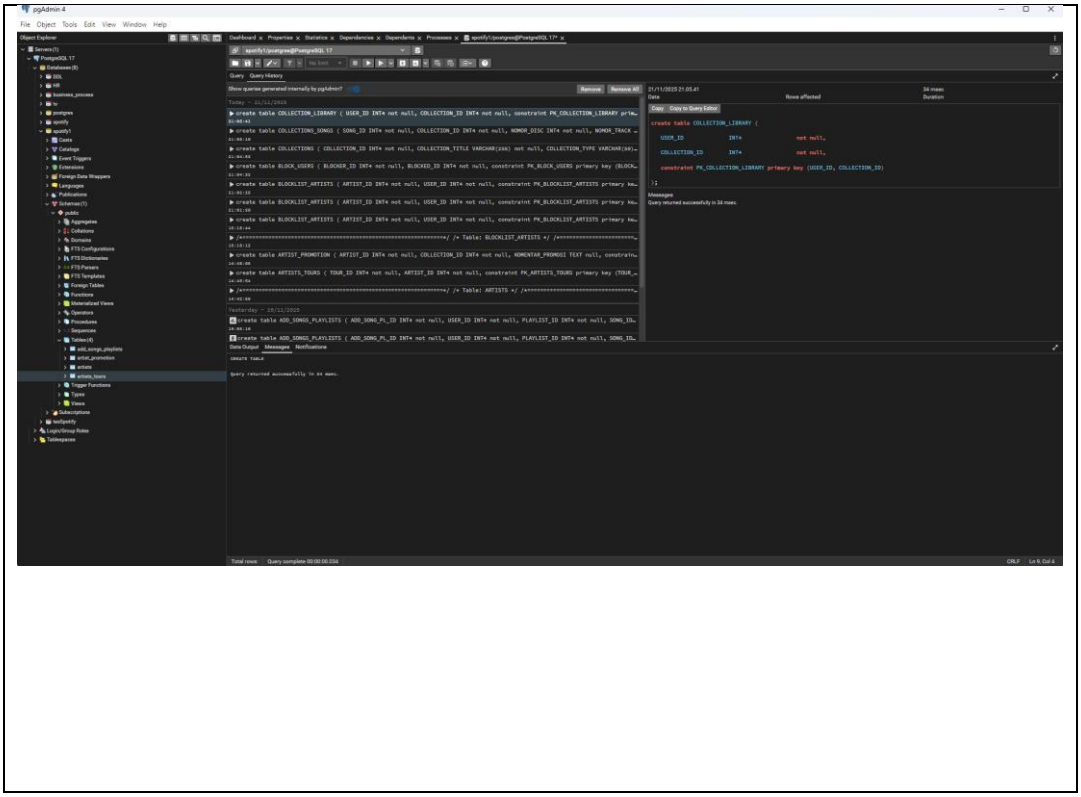
Tabel 44 DDL – Tabel Collection_library

Nama Tabel	Collection_library
Deskripsi	Tabel collection library berfungsin sebagai hubungan antara users dan collections, collections apa saja yang disimpan users

Script SQL

```
/*==============================================================*/
/* Table: COLLECTION_LIBRARY                                    */
/*==============================================================*/
create table COLLECTION_LIBRARY (
   USER_ID              INT4                 not null,
   COLLECTION_ID        INT4                 not null,
   constraint PK_COLLECTION_LIBRARY primary key (USER_ID,
COLLECTION_ID)
);
```

Screenshot Hasil

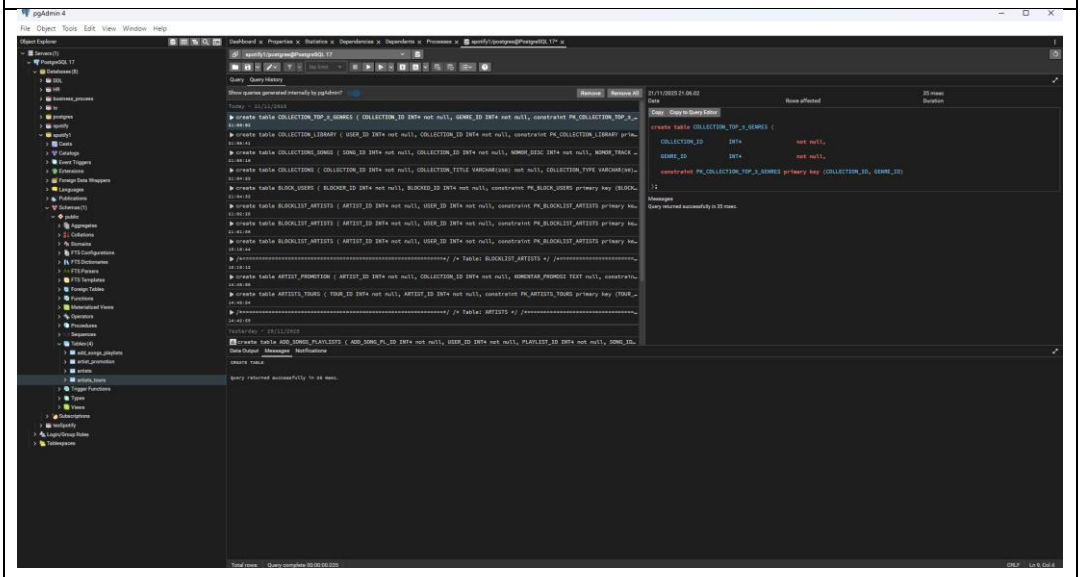


Tabel 45 DDL – Tabel collection_top_3_genres

Nama Tabel	Collection_top_3_genres
Deskripsi	Tabel ini berfungsi menyimpan 3 genre utama dari suatu collection
Script SQL	

```
/*==============================================================*/
/* Table: COLLECTION_TOP_3_GENRES                               */
/*==============================================================*/
create table COLLECTION_TOP_3_GENRES (
   COLLECTION_ID        INT4                 not null,
   GENRE_ID             INT4                 not null,
   constraint PK_COLLECTION_TOP_3_GENRES primary key
(COLLECTION_ID, GENRE_ID)
);
```

Screenshot Hasil

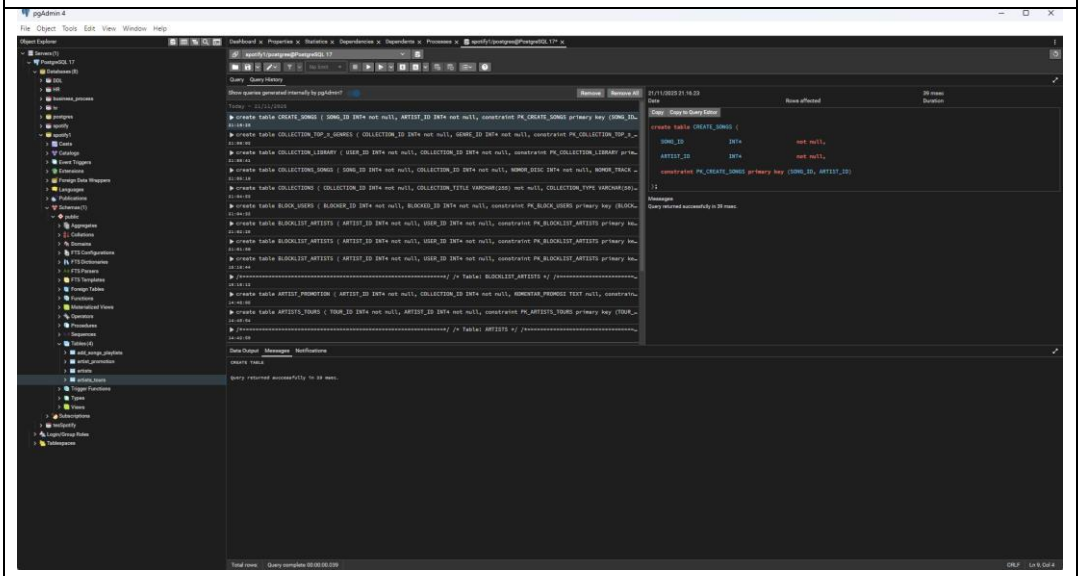


Tabel 46 DDL – Tabel create_songs

Nama Tabel	Create_songs
Deskripsi	Tabel ini berfungsi pembuatan songs, hubungan antara songs dan artists

Script SQL

```
/*==============================================================*/
/* Table: CREATE_SONGS                                          */
/*==============================================================*/
create table CREATE_SONGS (
   SONG_ID              INT4                 not null,
   ARTIST_ID            INT4                 not null,
   constraint PK_CREATE_SONGS primary key (SONG_ID, ARTIST_ID)
);
```

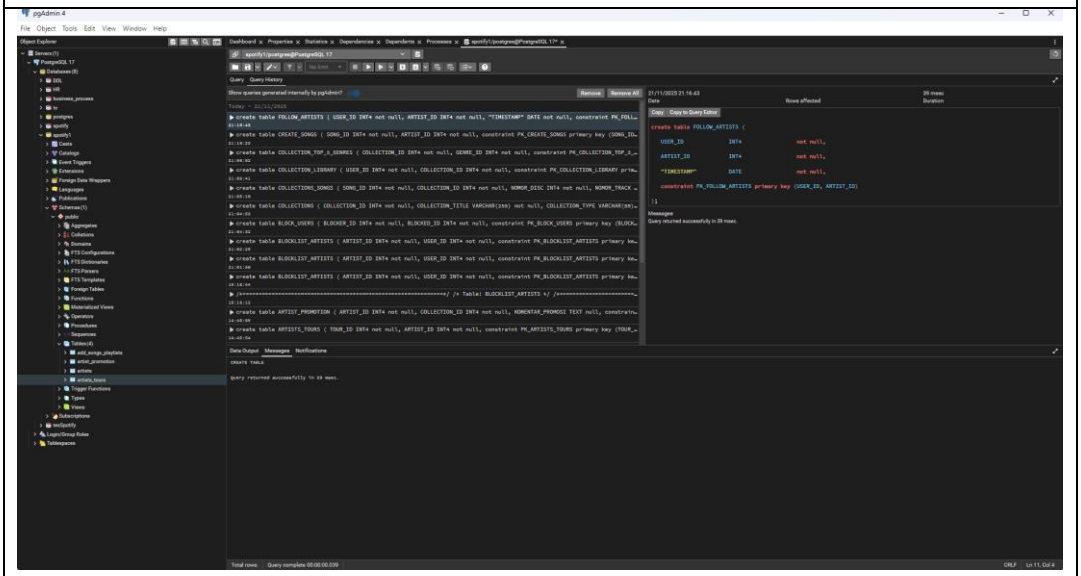
Screenshot Hasil


Tabel 47 DDL – Tabel follow_artists

Nama Tabel	Follow_artists
Deskripsi	Tabel ini berfungsi sebagai hubungan antara users dan artists yang difollow users

Script SQL

```
/*==============================================================*/
/* Table: FOLLOW_ARTISTS                                        */
/*==============================================================*/
create table FOLLOW_ARTISTS (
   USER_ID              INT4                 not null,
   ARTIST_ID            INT4                 not null,
   "TIMESTAMP"          DATE                 not null,
   constraint PK_FOLLOW_ARTISTS primary key (USER_ID, ARTIST_ID)
);
```

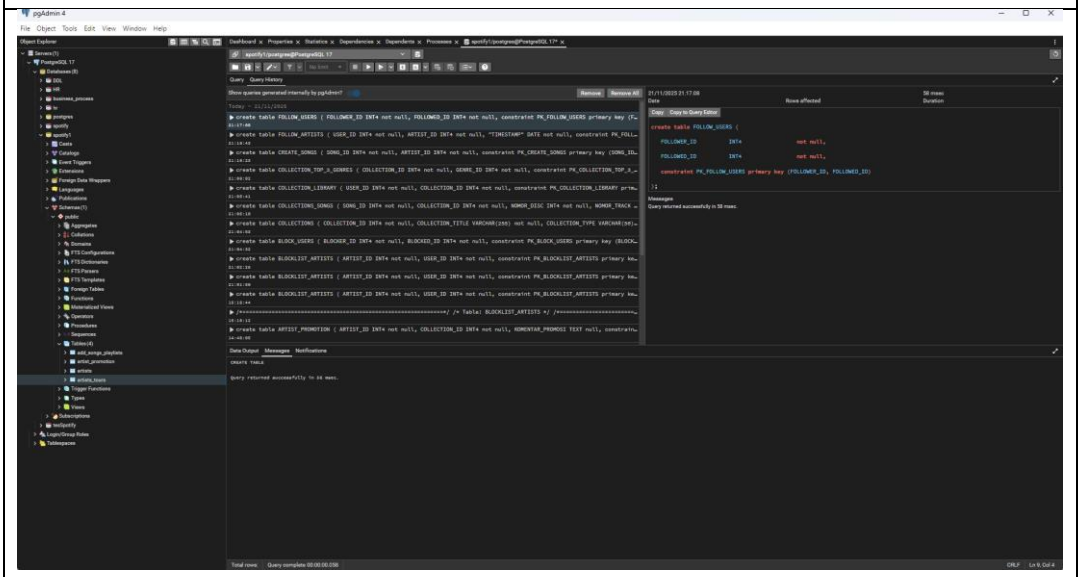
Screenshot Hasil


Tabel 48 DDL – Tabel follow_users

Nama Tabel	Follow_users
Deskripsi	Tabel ini sebagai hubungan antara users dan users lainnya, follow
Script SQL	

```
/*==============================================================*/
/* Table: FOLLOW_USERS                                          */
/*==============================================================*/
create table FOLLOW_USERS (
   FOLLOWER_ID          INT4                 not null,
   FOLLOWED_ID          INT4                 not null,
   constraint PK_FOLLOW_USERS primary key (FOLLOWER_ID,
FOLLOWED_ID)
);
```

Screenshot Hasil



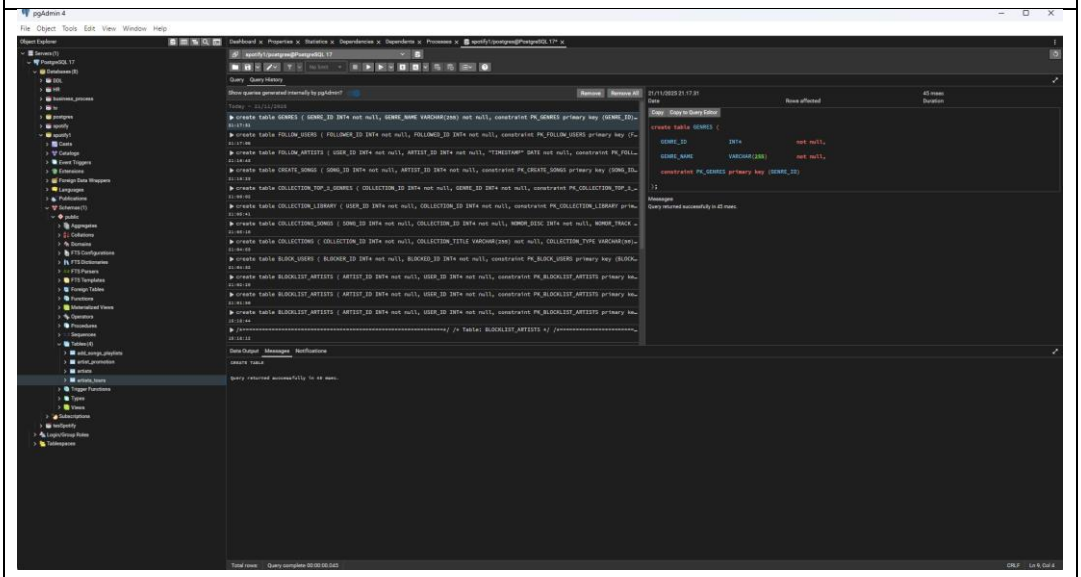
Tabel 49 DDL – Tabel genres

Nama Tabel	genres
Deskripsi	Tabel ini berfungsi menyimpan genres

Script SQL

```
/*==============================================================*/
/* Table: GENRES                                                */
/*==============================================================*/
create table GENRES (
   GENRE_ID             INT4                 not null,
   GENRE_NAME           VARCHAR(255)         not null,
   constraint PK_GENRES primary key (GENRE_ID)
);
```

Screenshot Hasil



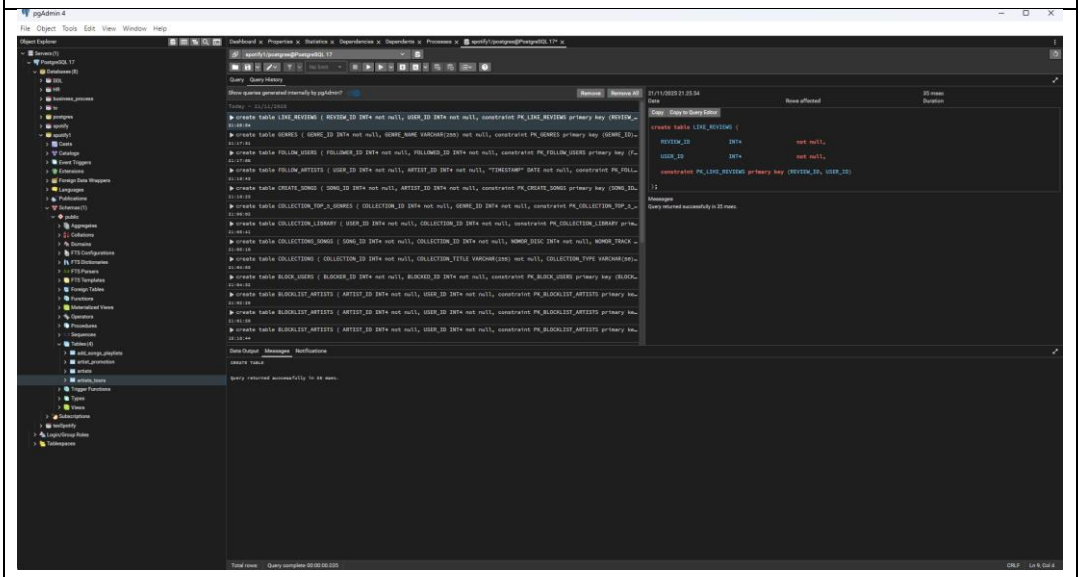
Tabel 50 DDL – Tabel like_reviews

Nama Tabel	Like_reviews
Deskripsi	Tabel ini bergungsi sebagai hubungan antara reviews dan userss

Script SQL

```
/*==============================================================*/
/* Table: LIKE_REVIEWS                                          */
/*==============================================================*/
create table LIKE_REVIEWS (
   REVIEW_ID            INT4                 not null,
   USER_ID              INT4                 not null,
   constraint PK_LIKE_REVIEWS primary key (REVIEW_ID, USER_ID)
);
```

Screenshot Hasil



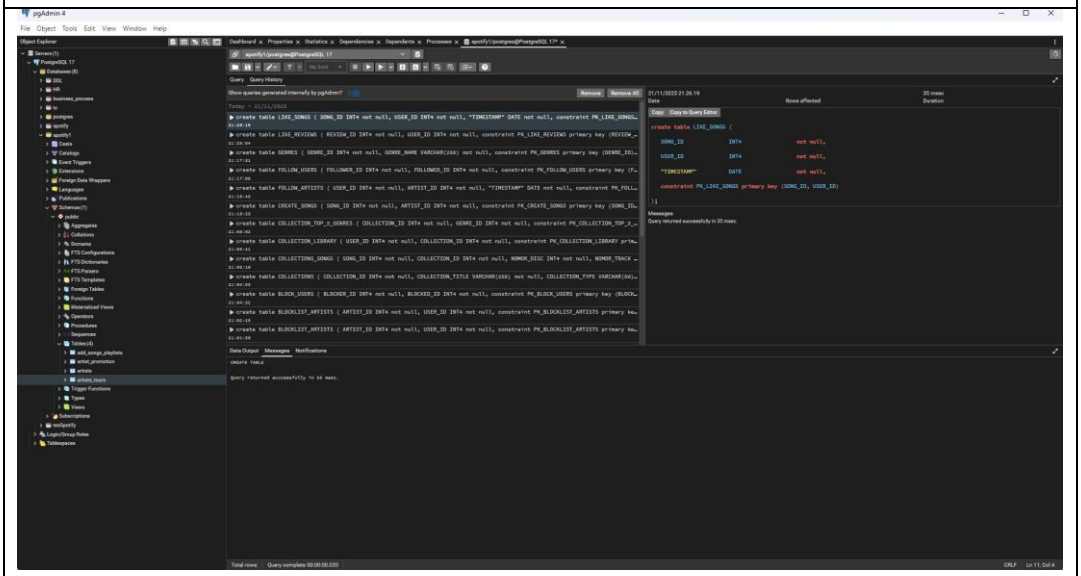
Tabel 51 DDL – Tabel like_songs

Nama Tabel	Like_songs
Deskripsi	Tabel lika_songs berfungsi sebagai hubungan antara users dan songs yang di-like

Script SQL

```
/*==============================================================*/
/* Table: LIKE_SONGS                                            */
/*==============================================================*/
create table LIKE_SONGS (
   SONG_ID              INT4                 not null,
   USER_ID              INT4                 not null,
   "TIMESTAMP"          DATE                 not null,
   constraint PK_LIKE_SONGS primary key (SONG_ID, USER_ID)
);
```

Screenshot Hasil



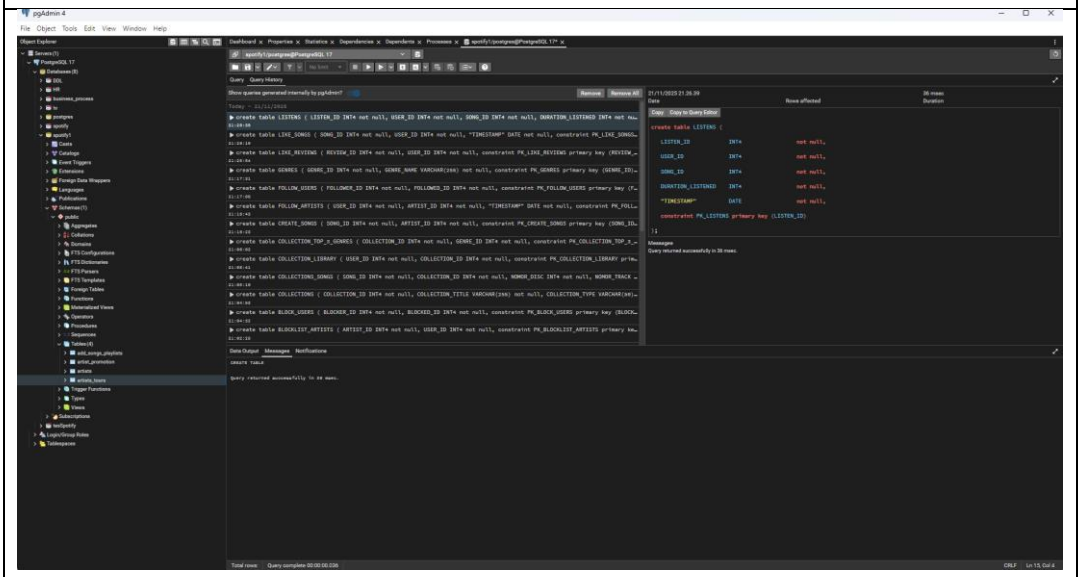
Tabel 52 DDL – Tabel listens

Nama Tabel	listens
Deskripsi	Tabel listens berfungsi sebagai penyimpanan lagu mana saja yang telah didengarkan users

Script SQL

```
/*==============================================================*/
/* Table: LISTENS                                               */
/*==============================================================*/
create table LISTENS (
   LISTEN_ID            INT4                 not null,
   USER_ID              INT4                 not null,
   SONG_ID              INT4                 not null,
   DURATION_LISTENED    INT4                 not null,
   "TIMESTAMP"          DATE                 not null,
   constraint PK_LISTENS primary key (LISTEN_ID)
);
```

Screenshot Hasil



Tabel 53 DDL – Tabel playlists

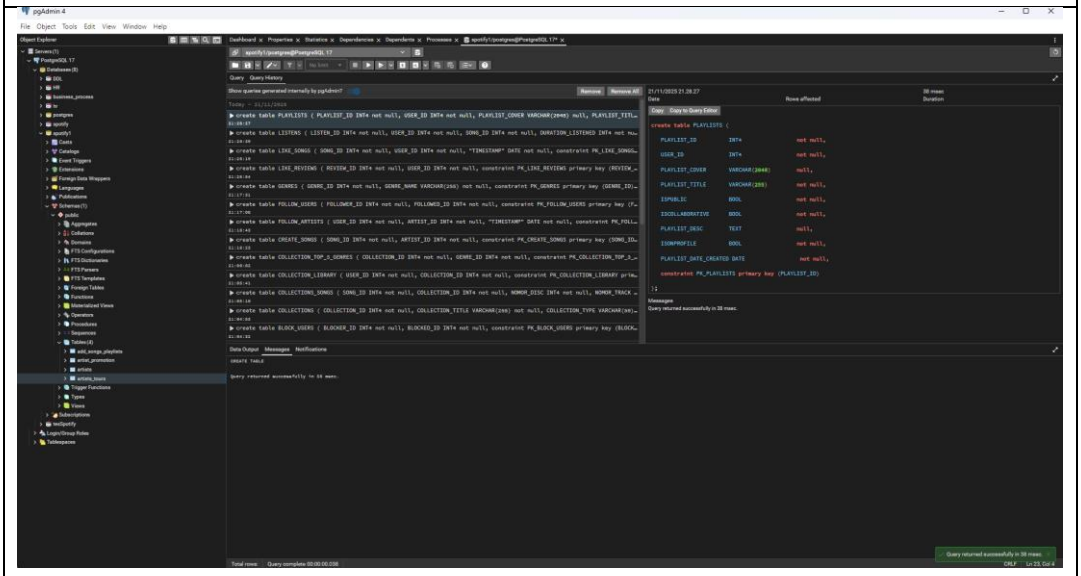
Nama Tabel	playlists
Deskripsi	Tabel ini berfungsi sebagai penyimpanan sejumlah songs

Script SQL

```
/*==============================================================*/
/* Table: PLAYLISTS                                             */
/*==============================================================*/
create table PLAYLISTS (
   PLAYLIST_ID          INT4                 not null,
   USER_ID              INT4                 not null,
   PLAYLIST_COVER       VARCHAR(2048)        null,
   PLAYLIST_TITLE       VARCHAR(255)         not null,
   ISPUBLIC             BOOL                 not null,
   ISCOLLABORATIVE      BOOL                 not null,
   PLAYLIST_DESC        TEXT                 null,
   ISONPROFILE          BOOL                 not null,
   PLAYLIST_DATE_CREATED DATE                 not null,
   constraint PK_PLAYLISTS primary key (PLAYLIST_ID)
```

```
);
```

Screenshot Hasil



Tabel 54 DDL – Tabel Pl_library

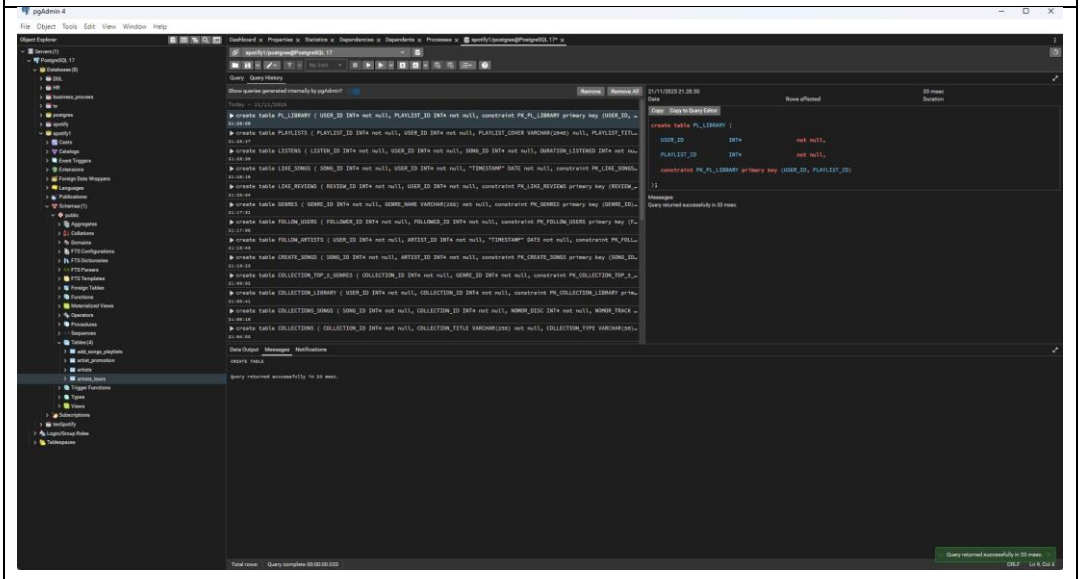
Nama Table	Pl_library
Deskripsi	Tabel pl_library berfungsi sebagai penyimpanan hubungan antara playlists yang dimiliki users

Script SQL

```
/*==============================================================*/
/* Table: PL_LIBRARY                                            */
/*==============================================================*/
create table PL_LIBRARY (
   USER_ID              INT4                 not null,
   PLAYLIST_ID          INT4                 not null,
   constraint PK_PL_LIBRARY primary key (USER_ID, PLAYLIST_ID)
```

```
);
```

Screenshot Hasil



Tabel 55 DDL – Tabel Rate_songs

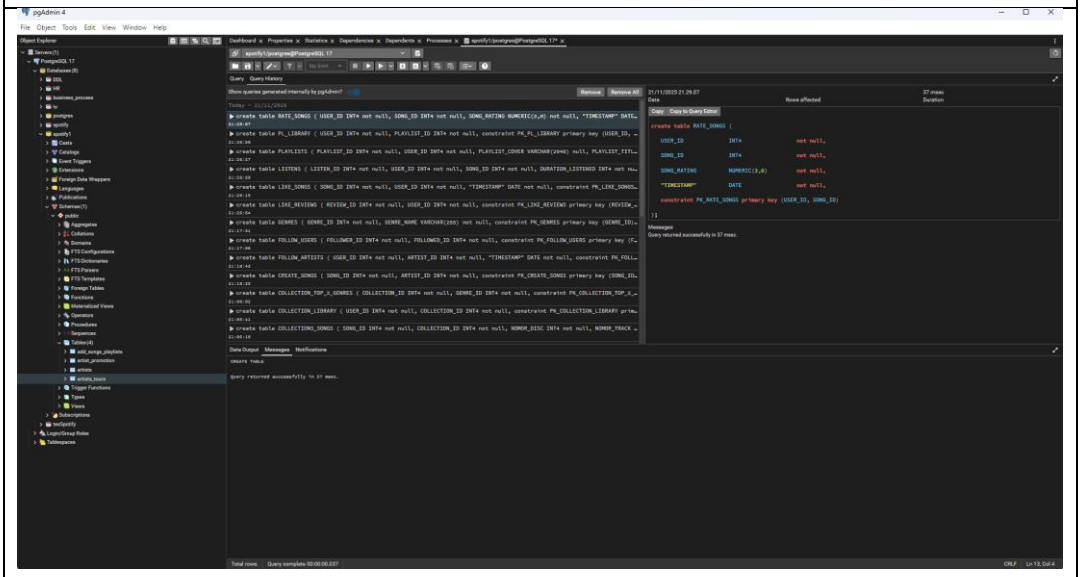
Nama Tabel	Rate_songs
Deskripsi	Tabel rate_songs berfungsi sebagai penyimpanan hubungan antara songs yang di-rate oleh users

Script SQL

```
/*==============================================================*/
/* Table: RATE_SONGS                                            */
/*==============================================================*/
create table RATE_SONGS (
   USER_ID              INT4                 not null,
   SONG_ID              INT4                 not null,
   SONG_RATING          NUMERIC(3,0)         not null,
```

```
   "TIMESTAMP"          DATE                    not null,
   constraint PK_RATE_SONGS primary key (USER_ID, SONG_ID)
);
```

Screenshot Hasil



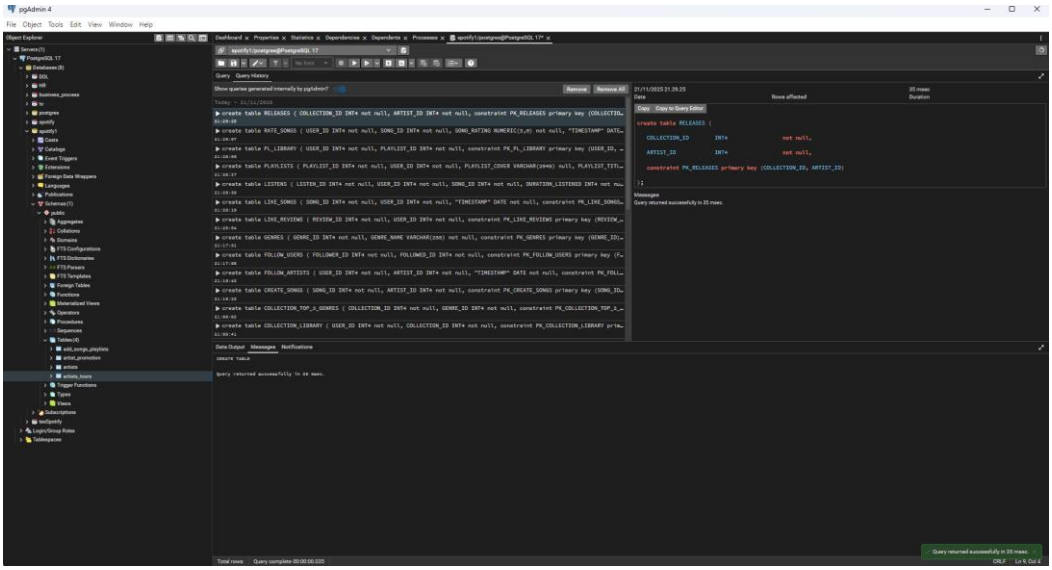
Tabel 56 DDL – Tabel releases

Nama Tabel	releases
Deskripsi	Tabel releases berfungsi sebagai hubungan antar collection yang akan direleases oleh artist

Script SQL

```
/*==============================================================*/
/* Table: RELEASES                                              */
/*==============================================================*/
create table RELEASES (
   COLLECTION_ID        INT4                    not null,
   ARTIST_ID            INT4                    not null,
   constraint PK_RELEASES primary key (COLLECTION_ID, ARTIST_ID)
);
```

Screenshot Hasil



Tabel 57 DDL – Tabel reviews

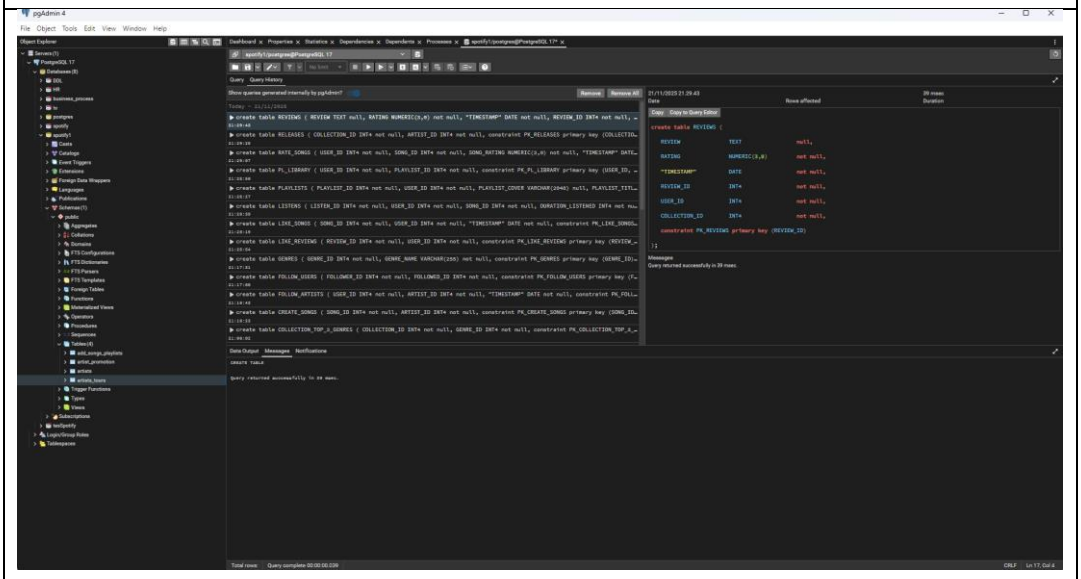
Nama Tabel	reviews
Deskripsi	Tabel ini merupakan penyimpanan review yang dibuat oleh users terhadap collections

Script SQL

```
/*==============================================================*/
/* Table: REVIEWS                                               */
/*==============================================================*/
create table REVIEWS (
   REVIEW               TEXT                 null,
   RATING               NUMERIC(3,0)         not null,
   "TIMESTAMP"          DATE                 not null,
   REVIEW_ID            INT4                 not null,
   USER_ID              INT4                 not null,
   COLLECTION_ID        INT4                 not null,
```

```
   constraint PK_REVIEWS primary key (REVIEW_ID)
);
```

Screenshot Hasil



Tabel 58 DDL – Tabel socials

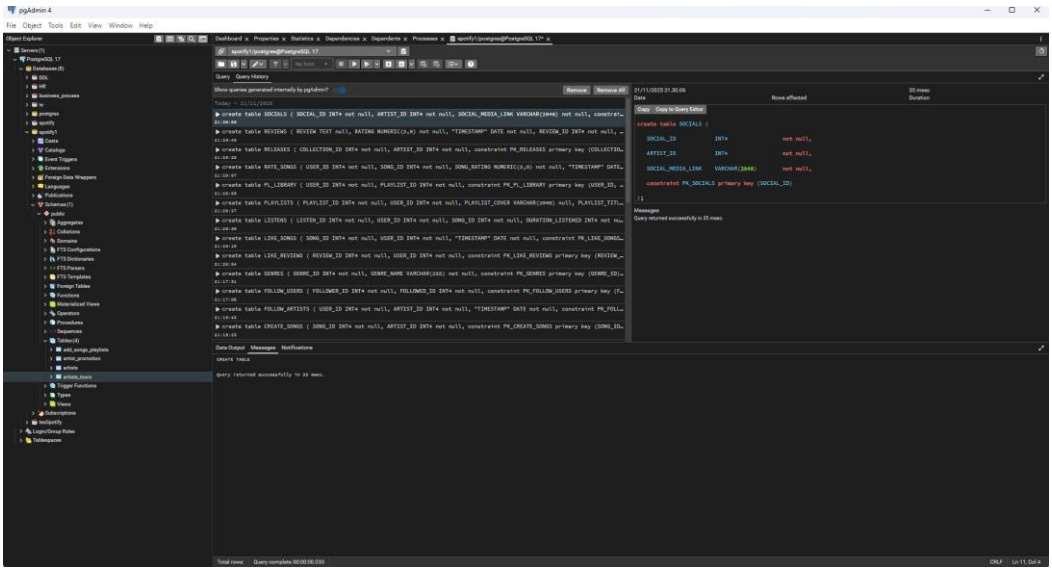
Nama Tabel	socials
Deskripsi	Tabel ini merupakan penyimpanan social media link artists

Script SQL

```
/*==============================================================*/
/* Table: SOCIALS                                               */
/*==============================================================*/
create table SOCIALS (
   SOCIAL_ID            INT4                 not null,
   ARTIST_ID            INT4                 not null,
   SOCIAL_MEDIA_LINK    VARCHAR(2048)        not null,
   constraint PK_SOCIALS primary key (SOCIAL_ID)
);
```

Screenshot Hasil

<Gambar Hasil Percobaan>



Tabel 59 DDL – Tabel songs

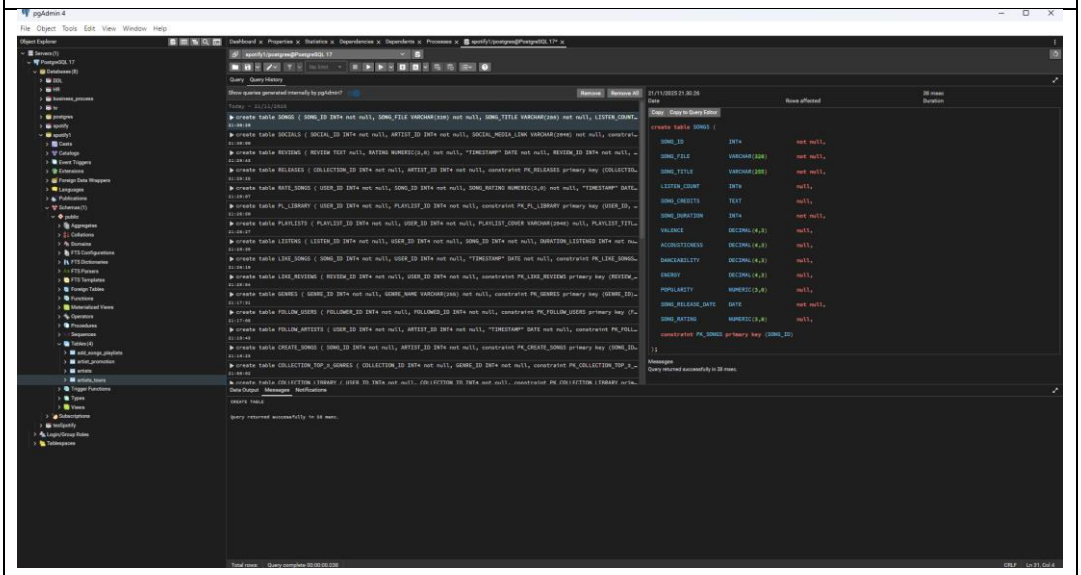
Nama Tabel	songs
Deskripsi	Tabel ini merupakan penyimpanan songs terdiri dari beberapa atribut

Script SQL

```
/*==============================================================*/
/* Table: SONGS                                                 */
/*==============================================================*/
create table SONGS (
   SONG_ID              INT4                 not null,
   SONG_FILE            VARCHAR(320)         not null,
   SONG_TITLE           VARCHAR(255)         not null,
   LISTEN_COUNT         INT8                 null,
   SONG_CREDITS         TEXT                 null,
   SONG_DURATION        INT4                 not null,
   VALENCE              DECIMAL(4,3)         null,
```

```
   ACCOUSTICNESS        DECIMAL(4,3)          null,
   DANCEABILITY         DECIMAL(4,3)          null,
   ENERGY               DECIMAL(4,3)          null,
   POPULARITY           NUMERIC(3,0)          null,
   SONG_RELEASE_DATE    DATE                  not null,
   SONG_RATING          NUMERIC(3,0)          null,
   constraint PK_SONGS primary key (SONG_ID)
);
```

Screenshot Hasil



Tabel 60 DDL – Tabel song_genres

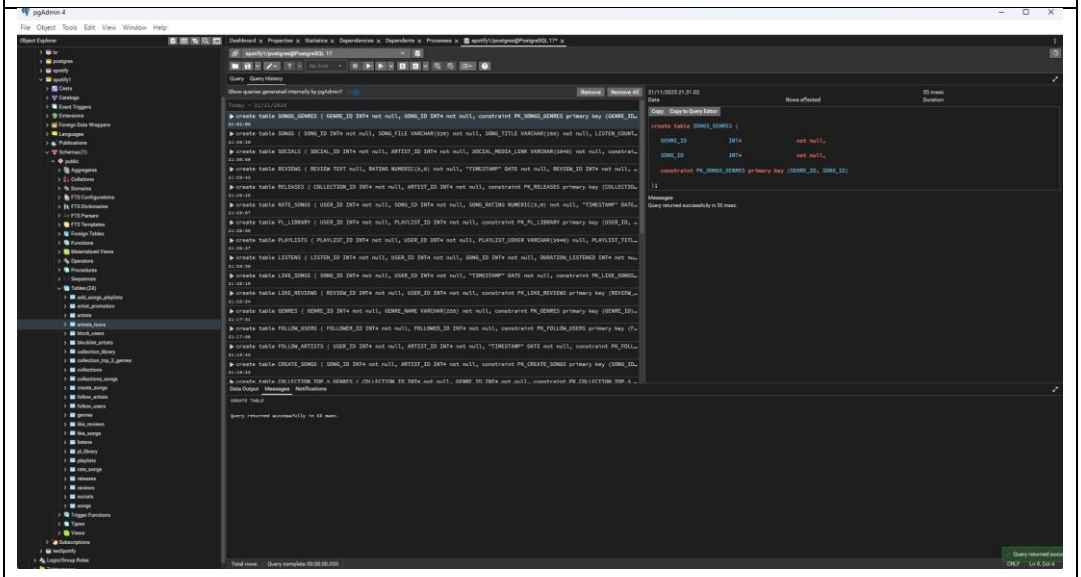
Nama Tabel	Song_genres
Deskripsi	Tabel ini berisi hubungan antara genres dan songs

Script SQL

```
/*==============================================================*/
/* Table: SONGS_GENRES                                          */
```

```
/*==============================================================*/
create table SONGS_GENRES (
   GENRE_ID             INT4                 not null,
   SONG_ID              INT4                 not null,
   constraint PK_SONGS_GENRES primary key (GENRE_ID, SONG_ID)
);
```

Screenshot Hasil



Tabel 61 DDL – Tabel tours

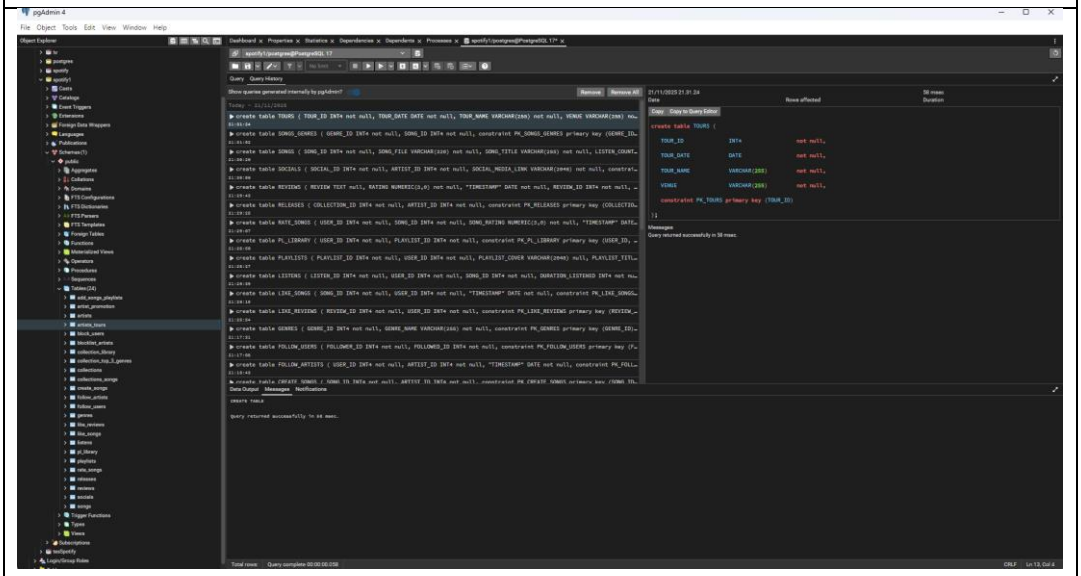
Nama Tabel	tours
Deskripsi	Tabel ini berfungsi menyimpan data tours dengan beberapa atributnya

Script SQL

```
/*==============================================================*/
/* Table: TOURS                                                 */
```

```
/*==============================================================*/
create table TOURS (
   TOUR_ID              INT4                 not null,
   TOUR_DATE            DATE                 not null,
   TOUR_NAME            VARCHAR(255)         not null,
   VENUE                VARCHAR(255)         not null,
   constraint PK_TOURS primary key (TOUR_ID)
);
```

Screenshot Hasil



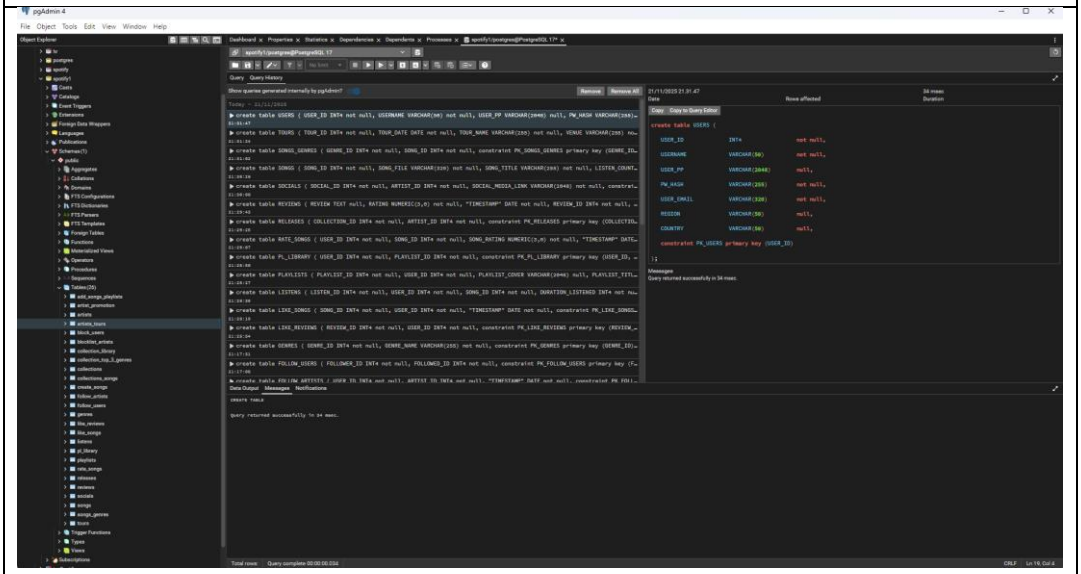
Tabel 62 DDL – Tabel users

Nama Tabel	users
Deskripsi	Tabel ini berfungsi menyimpan data utama users seperti user id, username, password, email dll.
Script SQL	

```
/*==============================================================*/
```

```
/* Table: USERS                                                 */
/*==============================================================*/
create table USERS (
   USER_ID              INT4                not null,
   USERNAME             VARCHAR(50)         not null,
   USER_PP              VARCHAR(2048)       null,
   PW_HASH              VARCHAR(255)        not null,
   USER_EMAIL           VARCHAR(320)        not null,
   REGION               VARCHAR(50)         null,
   COUNTRY              VARCHAR(50)         null,
   constraint PK_USERS primary key (USER_ID)
);
```

Screenshot Hasil



Tabel 63 DDL - Sequence

Nama Sequence	User_id, artist_id, song_id, playlist_id, collection_id, genre_id, review_id, tour_id, social_id, listen_id, add_songs_playlist_id

Deskripsi	Sequence yang digunakan untuk menghasilkan nilai ID unik pada berbagai entitas utama dalam sistem, seperti user, artist, song, playlist, collection, genre, review, tour, social, listening history, dan data penambahan lagu ke playlist. Sequence ini memastikan setiap record memiliki primary key yang konsisten, tidak duplikatif, dan tetap aman digunakan untuk operasi insert secara paralel

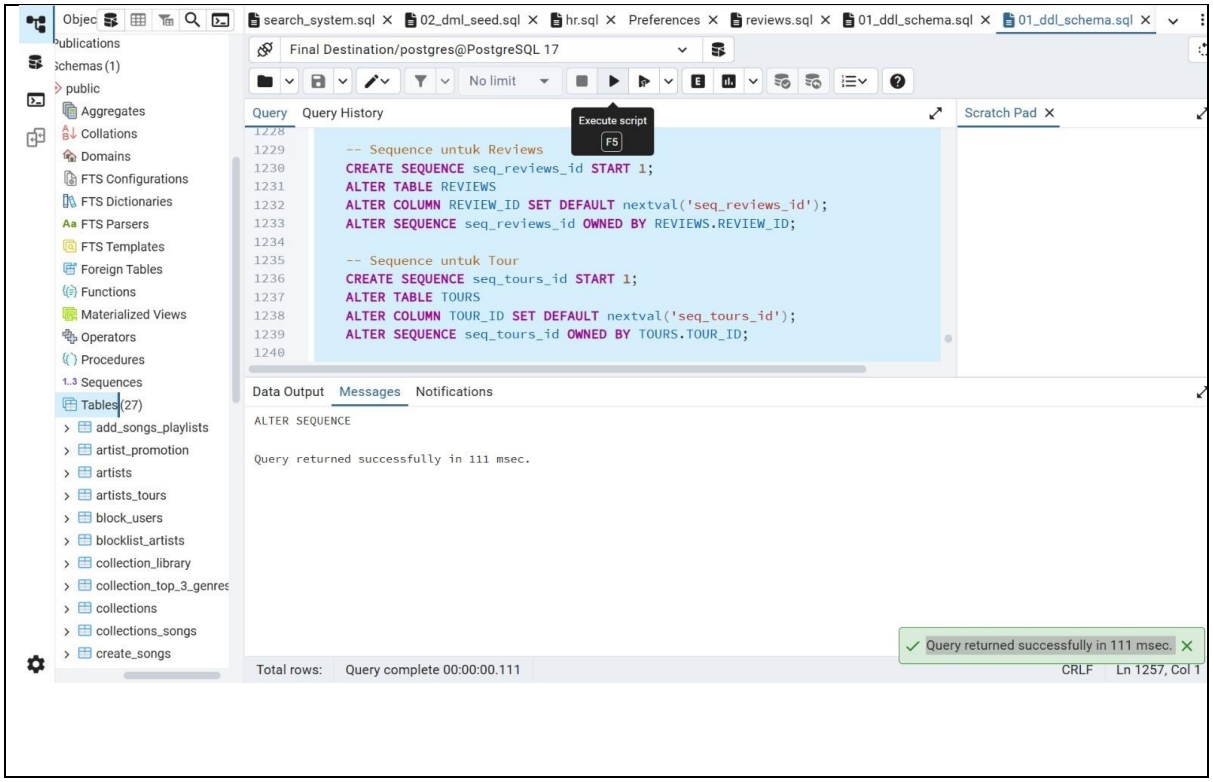
Script SQL

```
-- Sequence untuk artists

CREATE SEQUENCE seq_artists_id START 1;

ALTER TABLE ARTISTS

ALTER COLUMN ARTIST_ID SET DEFAULT nextval('seq_artists_id');

ALTER SEQUENCE seq_artists_id OWNED BY ARTISTS.ARTIST_ID;


-- Sequence untuk Songs

CREATE SEQUENCE seq_songs_id START 1;

ALTER TABLE SONGS

ALTER COLUMN SONG_ID SET DEFAULT nextval('seq_songs_id');

ALTER SEQUENCE seq_songs_id OWNED BY SONGS.SONG_ID;


-- Sequence untuk Users

CREATE SEQUENCE seq_users_id START 1;

ALTER TABLE USERS

ALTER COLUMN USER_ID SET DEFAULT nextval('seq_users_id');

ALTER SEQUENCE seq_users_id OWNED BY USERS.USER_ID;


-- Sequence untuk Playlists

CREATE SEQUENCE seq_playlists_id START 1;
```

```
ALTER TABLE PLAYLISTS
ALTER COLUMN PLAYLIST_ID SET DEFAULT nextval('seq_playlists_id');
ALTER SEQUENCE seq_playlists_id OWNED BY PLAYLISTS.PLAYLIST_ID;

-- Sequence untuk Collections
CREATE SEQUENCE seq_collections_id START 1;
ALTER TABLE COLLECTIONS
ALTER COLUMN COLLECTION_ID SET DEFAULT nextval('seq_collections_id');
ALTER SEQUENCE seq_collections_id OWNED BY COLLECTIONS.COLLECTION_ID;

-- Sequence untuk Genres
CREATE SEQUENCE seq_genres_id START 1;
ALTER TABLE GENRES
ALTER COLUMN GENRE_ID SET DEFAULT nextval('seq_genres_id');
ALTER SEQUENCE seq_genres_id OWNED BY GENRES.GENRE_ID;

-- Sequence untuk Reviews
CREATE SEQUENCE seq_reviews_id START 1;
ALTER TABLE REVIEWS
ALTER COLUMN REVIEW_ID SET DEFAULT nextval('seq_reviews_id');
ALTER SEQUENCE seq_reviews_id OWNED BY REVIEWS.REVIEW_ID;

-- Sequence untuk Tour
CREATE SEQUENCE seq_tours_id START 1;
ALTER TABLE TOURS
ALTER COLUMN TOUR_ID SET DEFAULT nextval('seq_tours_id');
ALTER SEQUENCE seq_tours_id OWNED BY TOURS.TOUR_ID;
```

```
-- Sequence untuk Socials

CREATE SEQUENCE seq_socials_id START 1;

ALTER TABLE SOCIALS

ALTER COLUMN SOCIAL_ID SET DEFAULT nextval('seq_socials_id');

ALTER SEQUENCE seq_socials_id OWNED BY SOCIALS.SOCIAL_ID;


-- Sequence untuk Listens

CREATE SEQUENCE seq_listens_id START 1;

ALTER TABLE LISTENS

ALTER COLUMN LISTEN_ID SET DEFAULT nextval('seq_listens_id');

ALTER SEQUENCE seq_listens_id OWNED BY LISTENS.LISTEN_ID;


-- Sequence untuk ADD_SONGS_PLAYLISTS

CREATE SEQUENCE seq_add_songs_playlist_id START 1;

ALTER TABLE ADD_SONGS_PLAYLISTS

ALTER COLUMN ADD_SONG_PL_ID SET DEFAULT nextval('seq_add_songs_playlist_id');

ALTER SEQUENCE seq_add_songs_playlist_id OWNED BY
ADD_SONGS_PLAYLISTS.ADD_SONG_PL_ID;
```

Screenshot Hasil



Tabel 64 DDL - Constraints

Nama Constraint	CHK_SONG_METRICS, CHK_SONG_DURATION_POSITIVE, CHK_SONG_POPULARITY_RANGE, CHK_RATE_SONG_RANGE, CHK_REVIEW_RATING_RANGE, CHK_LISTENER_POSITIVE, CHK_PLAYLIST_ORDER_POSITIVE, CHK_DISC_TRACK_POSITIVE, CHK_PW_HASH_LENGTH, CHK_USERNAME_NO_WHITESPACE, CHK_ARTISTNAME_NO_WHITESPACE, CHK_NO_SELF_FOLLOW, CHK_NO_SELF_BLOCK, CHK_EMAIL_FORMAT, CHK_COLLECTION_TYPE_VALID, set defaults, FK_PROMO_OWN_RELEASE, UQ_USER_COLLECTION_REVIEW
Deskripsi	Berbagai *check constraint* diterapkan untuk memastikan seluruh data yang tersimpan tetap valid dan konsisten sesuai aturan bisnis. Pada tabel SONGS, metrik audio seperti valence, danceability, energy,

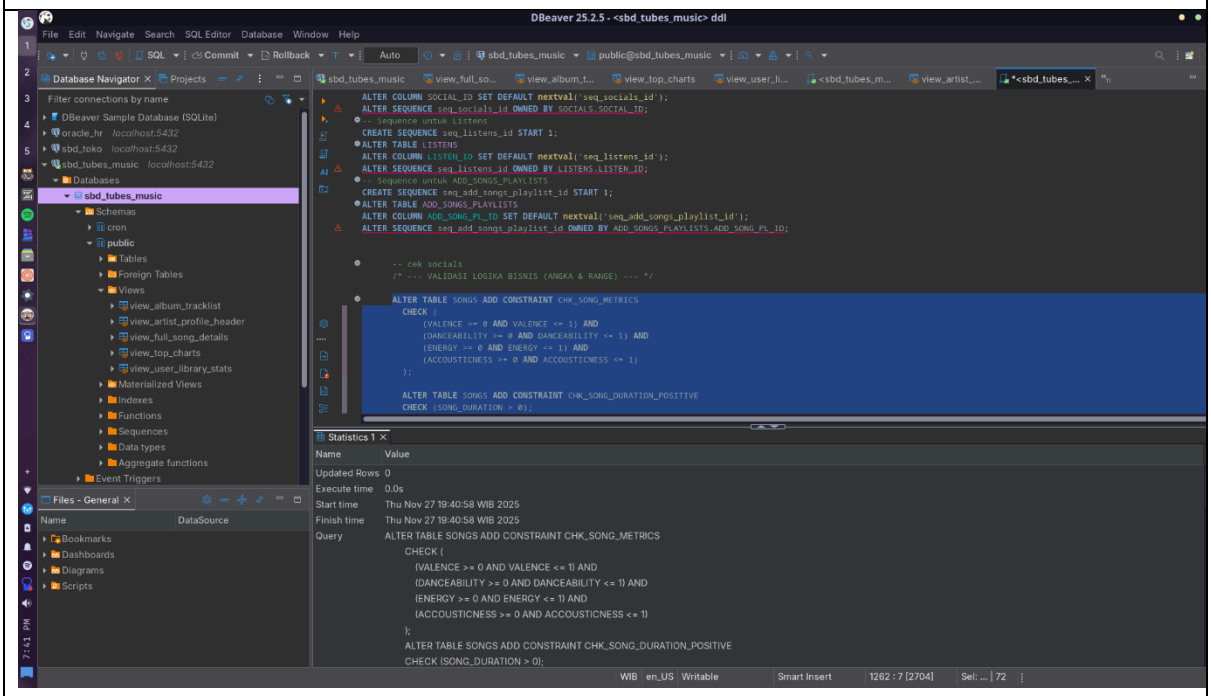
	dan acousticness dibatasi dalam rentang 0–1, sementara durasi lagu harus positif dan popularitas berada pada skala 0–100. Standar ini diperkuat dengan pembatasan nilai rating pada RATE_SONGS dan REVIEWS, yang harus berada dalam rentang 1–100. Selain itu, nomor urut lagu dalam playlist serta nomor disc dan track pada koleksi wajib bernilai positif, dan jumlah monthly listener artis tidak boleh negatif. Validasi atribut identitas dan format teks juga diterapkan untuk menjaga kualitas data. Password hash diwajibkan memiliki panjang minimal, username serta nama artis tidak boleh berupa whitespace, dan email pengguna harus sesuai format email yang valid. Untuk mencegah hubungan yang tidak logis, sistem melarang pengguna mem-follow atau mem-block dirinya sendiri. Tipe koleksi pun dibatasi hanya pada kategori yang diizinkan seperti Album, EP, Single, dan Compilation. Pada sisi integritas relasional, sistem memastikan bahwa artis hanya dapat mempromosikan koleksi yang benar-benar dirilis olehnya, serta mencegah satu pengguna memberikan lebih dari satu review terhadap koleksi yang sama.
Script SQL	
ALTER TABLE SONGS ADD CONSTRAINT CHK_SONG_METRICS CHECK ((VALENCE >= 0 AND VALENCE <= 1) AND (DANCEABILITY >= 0 AND DANCEABILITY <= 1) AND (ENERGY >= 0 AND ENERGY <= 1) AND (ACCOUSTICNESS >= 0 AND ACCOUSTICNESS <= 1)); ALTER TABLE SONGS ADD CONSTRAINT CHK_SONG_DURATION_POSITIVE CHECK (SONG_DURATION > 0);	

```
ALTER TABLE SONGS ADD CONSTRAINT CHK_SONG_POPULARITY_RANGE
CHECK (POPULARITY >= 0 AND POPULARITY <= 100);

ALTER TABLE RATE_SONGS ADD CONSTRAINT CHK_RATE_SONG_RANGE
CHECK (SONG_RATING >= 1 AND SONG_RATING <= 100);

ALTER TABLE REVIEWS ADD CONSTRAINT CHK_REVIEW_RATING_RANGE
CHECK (RATING >= 1 AND RATING <= 100);

ALTER TABLE ARTISTS ADD CONSTRAINT CHK_LISTENER_POSITIVE
CHECK (MONTHLY_LISTENER_COUNT >= 0);

ALTER TABLE ADD_SONGS_PLAYLISTS ADD CONSTRAINT
CHK_PLAYLIST_ORDER_POSITIVE
CHECK (NO_URUT > 0);

ALTER TABLE COLLECTIONS_SONGS ADD CONSTRAINT CHK_DISC_TRACK_POSITIVE
CHECK (NOMOR_DISC > 0 AND NOMOR_TRACK > 0);

ALTER TABLE USERS ADD CONSTRAINT CHK_PW_HASH_LENGTH
CHECK (LENGTH(PW_HASH) >= 8);

ALTER TABLE USERS ADD CONSTRAINT CHK_USERNAME_NO_WHITESPACE
CHECK (LENGTH(TRIM(USERNAME)) > 0);

ALTER TABLE ARTISTS ADD CONSTRAINT CHK_ARTISTNAME_NO_WHITESPACE
CHECK (LENGTH(TRIM(ARTIST_NAME)) > 0);
```

```
ALTER TABLE FOLLOW_USERS
ADD CONSTRAINT CHK_NO_SELF_FOLLOW
CHECK (FOLLOWER_ID <> FOLLOWED_ID);

ALTER TABLE BLOCK_USERS
ADD CONSTRAINT CHK_NO_SELF_BLOCK
CHECK (BLOCKER_ID <> BLOCKED_ID);
/* --- VALIDASI FORMAT TEKS & TIPE --- */

ALTER TABLE USERS ADD CONSTRAINT CHK_EMAIL_FORMAT
CHECK (USER_EMAIL LIKE '%_@__%.__%');

ALTER TABLE COLLECTIONS ADD CONSTRAINT CHK_COLLECTION_TYPE_VALID
CHECK (COLLECTION_TYPE IN ('Album', 'EP', 'Single', 'Compilation'));

/* --- NILAI DEFAULT --- */

-- 1. Playlist otomatis Private jika tidak diisi
ALTER TABLE PLAYLISTS ALTER COLUMN ISPUBLIC SET DEFAULT FALSE;

-- 2. Playlist otomatis Tidak Kolaboratif jika tidak diisi
ALTER TABLE PLAYLISTS ALTER COLUMN ISCOLLABORATIVE SET DEFAULT FALSE;



-- artist hanya bisa promosi rilisannya sendiri
ALTER TABLE ARTIST_PROMOTION
   ADD CONSTRAINT FK_PROMO_OWN_RELEASE
```

```
FOREIGN KEY (ARTIST_ID, COLLECTION_ID)
REFERENCES RELEASES (ARTIST_ID, COLLECTION_ID)
ON DELETE CASCADE ON UPDATE CASCADE;


ALTER TABLE REVIEWS
ADD CONSTRAINT UQ_USER_COLLECTION_REVIEW UNIQUE (user_id, collection_id);
```

Screenshot Hasil



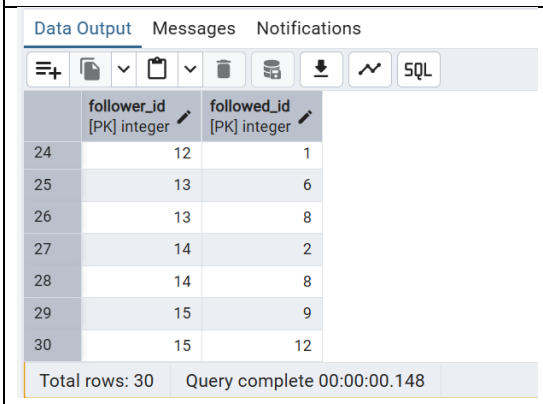
5.2 DML

Tabel 65 DML – Follow user

Nama Fitur	Follow User
Deskripsi	User dapat mengikuti atau berhenti mengikuti user lain.
Script SQL	

```
SELECT * FROM follow_users;
```

Screenshot Hasil



	follower_id [PK] integer	followed_id [PK] integer
24	12	1
25	13	6
26	13	8
27	14	2
28	14	8
29	15	9
30	15	12

Total rows: 30 Query complete 00:00:00.148

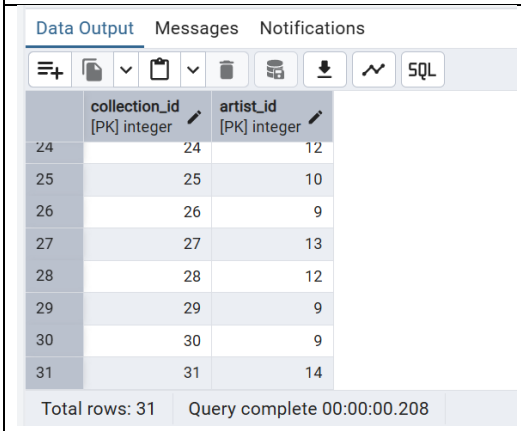
Tabel 66 DML – Rilis Collection

Nama Fitur	Rilis Collection
Deskripsi	Artis dapat merilis lagu atau collection, dicatat tanggal rilis nya. Suatu rilis bisa berupa Pre-Release, dimana wujudnya sudah dapat terlihat

Script SQL

```
select * from releases
```

Screenshot Hasil



	collection_id [PK] integer	artist_id [PK] integer
24	24	12
25	25	10
26	26	9
27	27	13
28	28	12
29	29	9
30	30	9
31	31	14

Total rows: 31 Query complete 00:00:00.208

Tabel 67 DML – Promosi Rilis

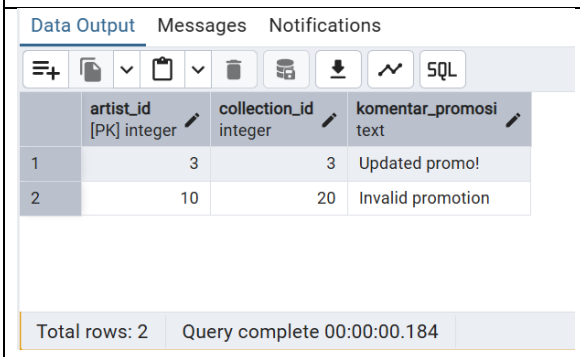
Nama Fitur	Promosi Rilis

Deskripsi	Artis dapat memilih satu rilisnya (lagu atau collection) untuk di rekomendasi kan di profilnya.

Script SQL

```
select * from artist_promotion
```

Screenshot Hasil



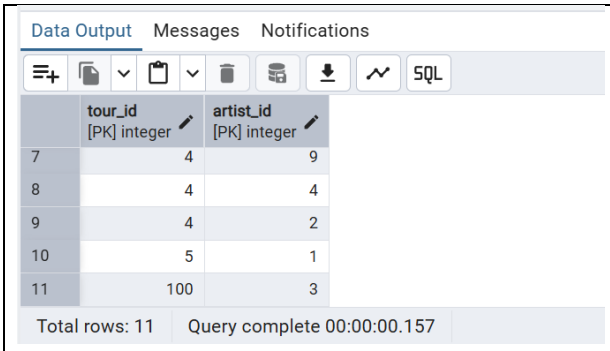
Tabel 68 DML – Manajemen Tur

Nama Fitur	Manajemen Tur
Deskripsi	Artist dapat melakukan tour. Seluruh data tentang tour disimpan di table tours dan relasinya di artist_tours.

Script SQL

```
select * from artists_tours
select * from tours

--tambah tour
INSERT INTO tours (tour_id, tour_name, venue, tour_date)
VALUES (100, 'World Tour 2025', 'Jakarta Convention Center', '2025-12-20');

-- relasikan artis ke tour
INSERT INTO artists_tours (artist_id, tour_id)
VALUES (3, 100);
```

Screenshot Hasil



Tabel 69 DML – Like Review

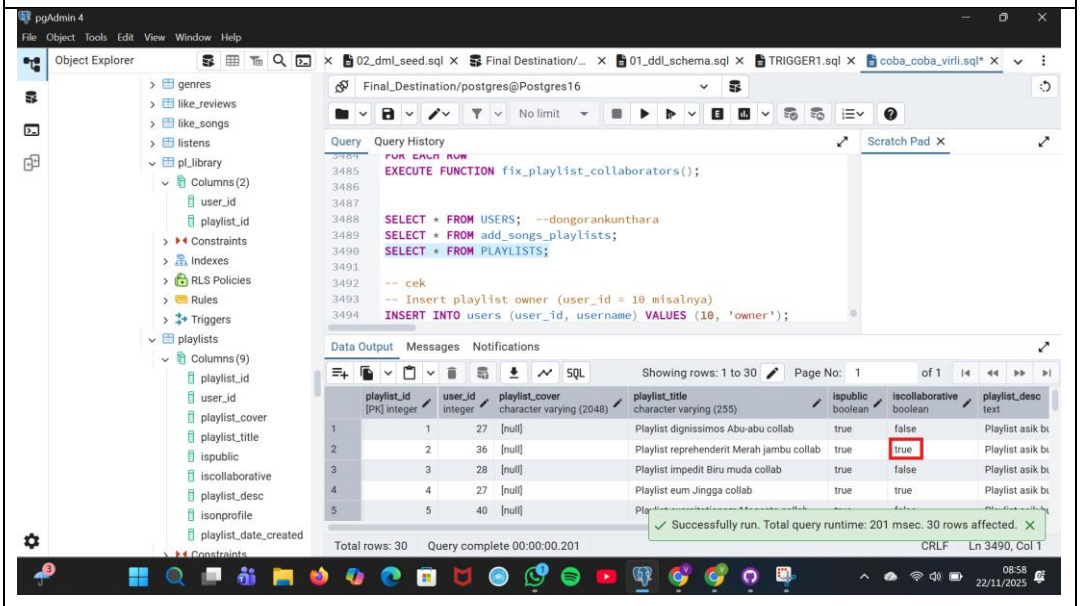
Nama Fitur	Like Review
Deskripsi	Review untuk collection yang dibuat oleh user dapat dilike.
Script SQL	
`select * from like_reviews`	
Screenshot Hasil	
	

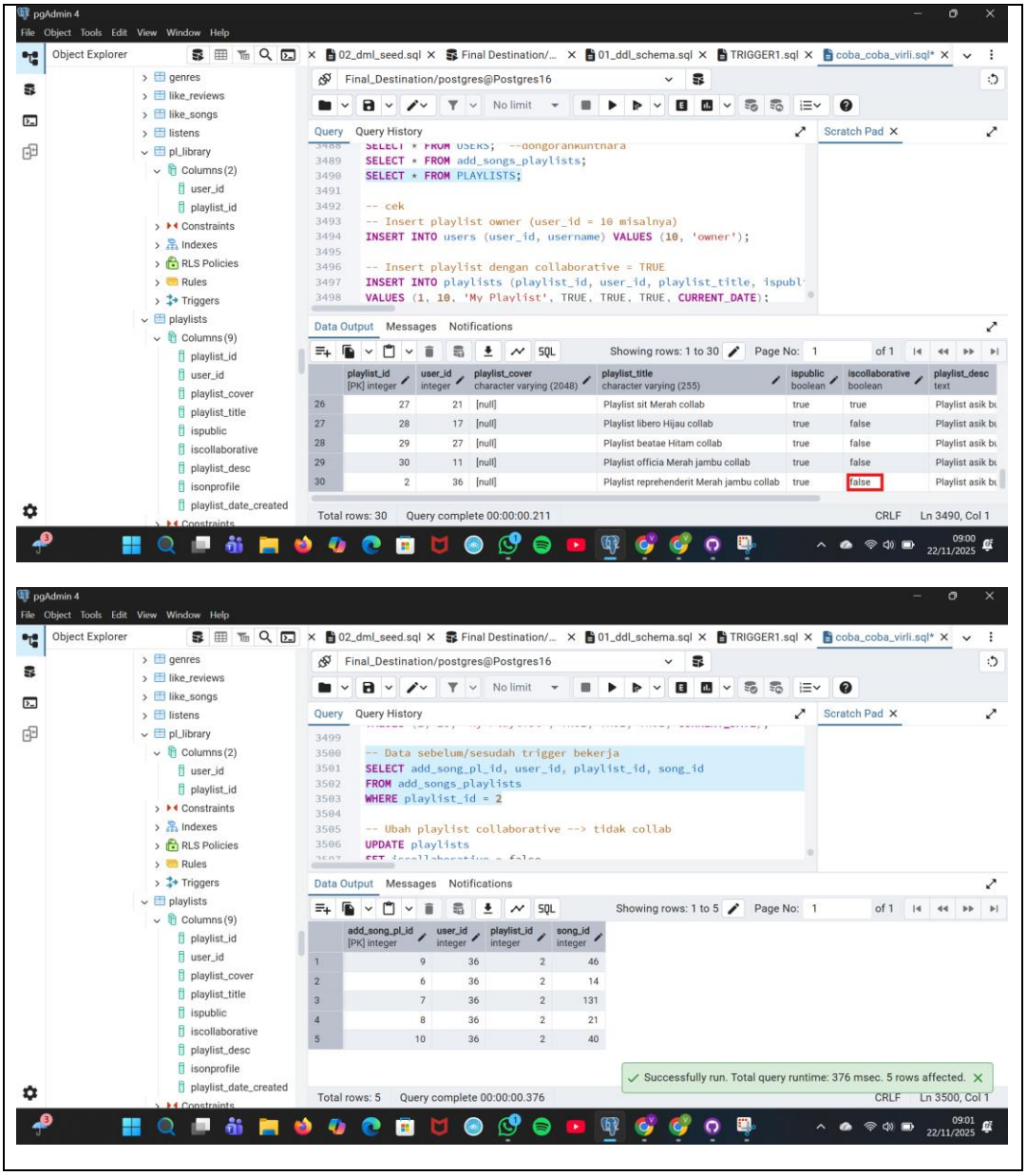
Tabel 70 DML – playlist_collaborators

Nama Fitur	playlist_collaborators
Deskripsi	Memperbarui seluruh user_id pada lagu dalam sebuah playlist ketika playlist tersebut berubah dari *collaborative* menjadi *non-collaborative*. Semua lagu yang sebelumnya ditambahkan oleh user lain akan dialihkan menjadi milik pemilik playlist.
Script SQL	

```
SELECT * FROM USERS;

SELECT * FROM add_songs_playlists;

SELECT * FROM PLAYLISTS;



-- cek

-- Data sebelum/sesudah trigger bekerja

SELECT add_song_pl_id, user_id, playlist_id, song_id

FROM add_songs_playlists

WHERE playlist_id = 2



-- Ubah playlist collaborative --> tidak collab

UPDATE playlists

SET iscollaborative = false

WHERE playlist_id = 2;
```

Screenshot Hasil



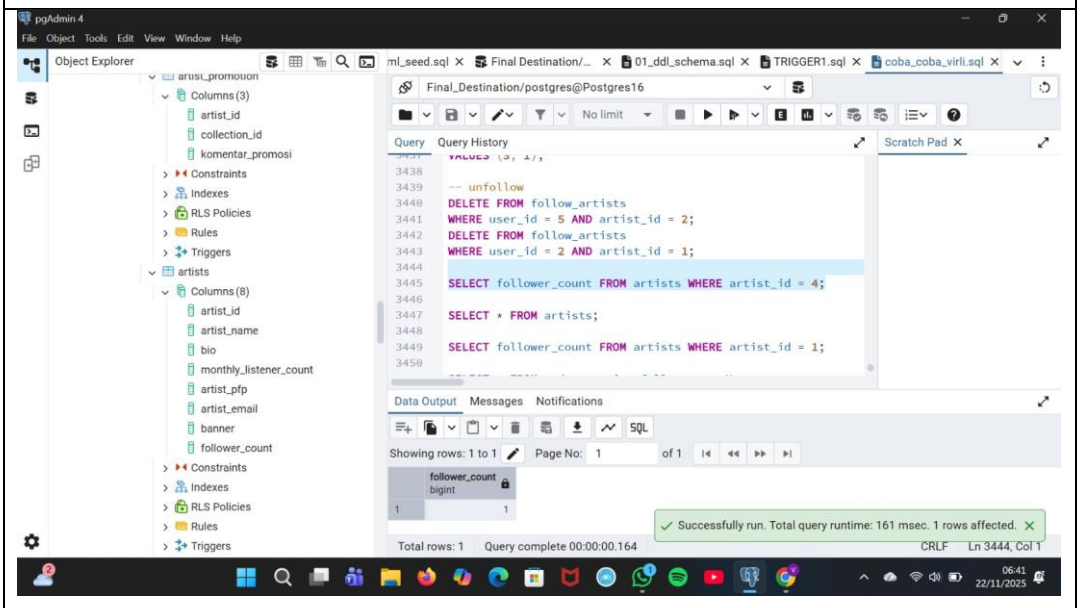


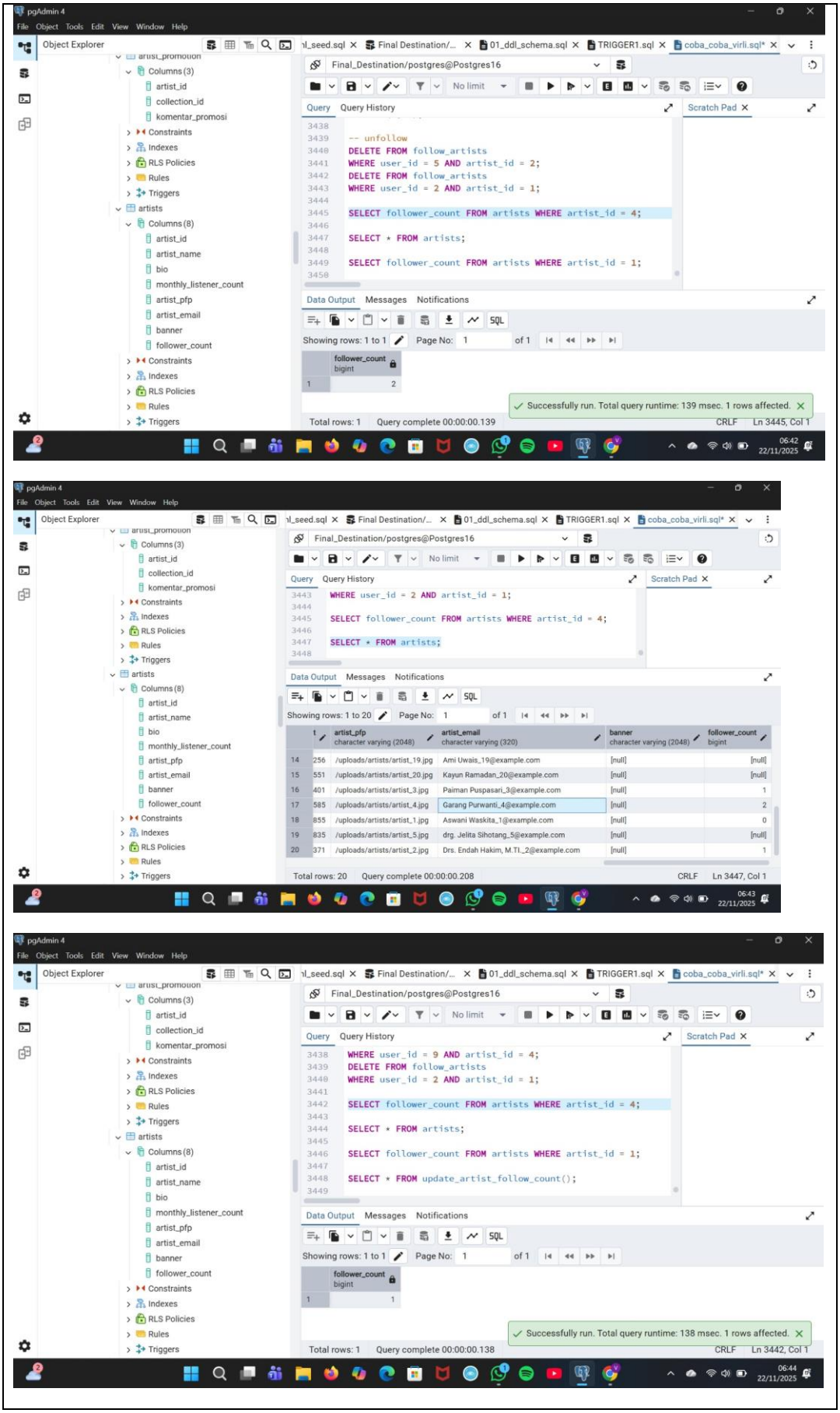
Tabel 71 DML – Update follow count

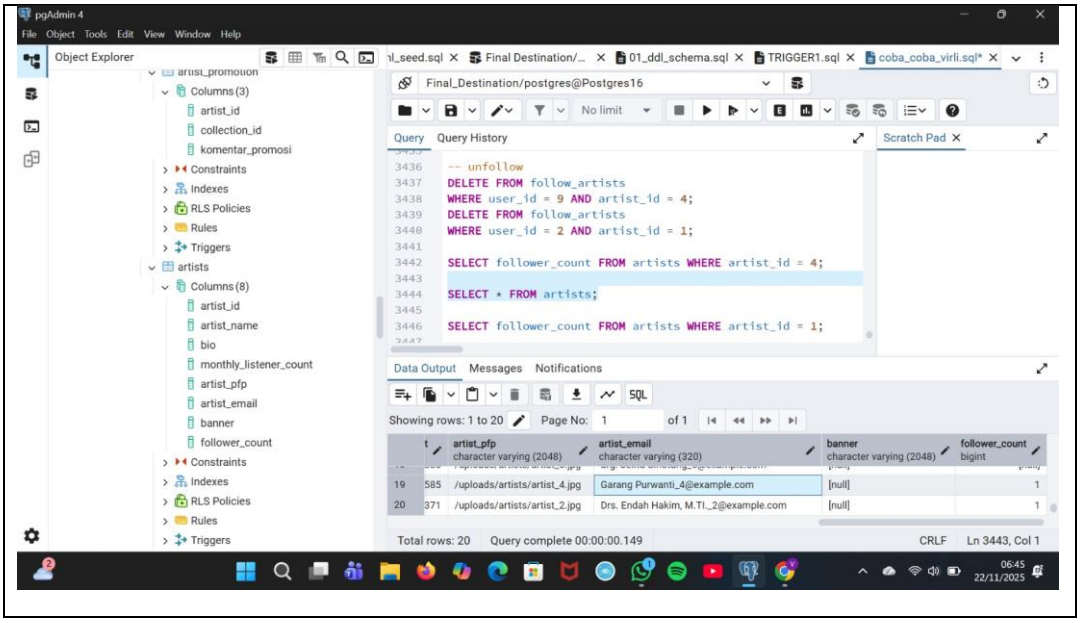
Nama Fitur	Update artist follow count
Deskripsi	Menambahkan otomatis ketika user follow/unfollow
Script SQL	
INSERT INTO follow_artists (user_id, artist_id) VALUES (9, 4);	

```
INSERT INTO follow_artists (user_id, artist_id)

VALUES (10, 4);

-- unfollow

DELETE FROM follow_artists

WHERE user_id = 9 AND artist_id = 4;

DELETE FROM follow_artists

WHERE user_id = 2 AND artist_id = 1;

SELECT follower_count FROM artists WHERE artist_id = 4;


SELECT * FROM artists;


SELECT follower_count FROM artists WHERE artist_id = 1;


SELECT * FROM update_artist_follow_count();
```

Screenshot Hasil




pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Final_Destination/postgres@Postgres16
Query Query History
Scratch Pad
-- unfollow
DELETE FROM follow_artists
WHERE user_id = 5 AND artist_id = 2;
DELETE FROM follow_artists
WHERE user_id = 2 AND artist_id = 1;
SELECT follower_count FROM artists WHERE artist_id = 4;
SELECT * FROM artists;
SELECT follower_count FROM artists WHERE artist_id = 1;
Data Output Messages Notifications
Showing rows: 1 to 1 Page No: 1 of 1
follower_count bigint
1 2
Successfully run. Total query runtime: 139 msec. 1 rows affected.
Total rows: 1 Query complete 00:00:00.139
CRLF Ln 3445, Col 1
06:42 22/11/2025
Showing rows: 1 to 20 Page No: 1 of 1
artist_pfp character varying (2048)
artist_email character varying (320)
banner character varying (2048)
follower_count bigint
14 256 /uploads/artists/artist_19.jpg Ami Uwais_19@example.com [null] [null]
15 551 /uploads/artists/artist_20.jpg Kayun Ramadan_20@example.com [null] [null]
16 401 /uploads/artists/artist_3.jpg Paiman Puspasari_3@example.com [null] 1
17 585 /uploads/artists/artist_4.jpg Garang Purwanti_4@example.com [null] 2
18 855 /uploads/artists/artist_1.jpg Aswani Waskita_1@example.com [null] 0
19 835 /uploads/artists/artist_5.jpg drg. Jelita Sihotang_5@example.com [null] [null]
20 371 /uploads/artists/artist_2.jpg Drs. Endah Hakim, M.TI._2@example.com [null] 1
Total rows: 20 Query complete 00:00:00.208
CRLF Ln 3447, Col 1
06:43 22/11/2025
WHERE user_id = 9 AND artist_id = 4;
DELETE FROM follow_artists
WHERE user_id = 2 AND artist_id = 1;
SELECT follower_count FROM artists WHERE artist_id = 4;
SELECT * FROM artists;
SELECT follower_count FROM artists WHERE artist_id = 1;
SELECT * FROM update_artist_follow_count();
Successfully run. Total query runtime: 138 msec. 1 rows affected.
Total rows: 1 Query complete 00:00:00.138
CRLF Ln 3442, Col 1
06:44 22/11/2025

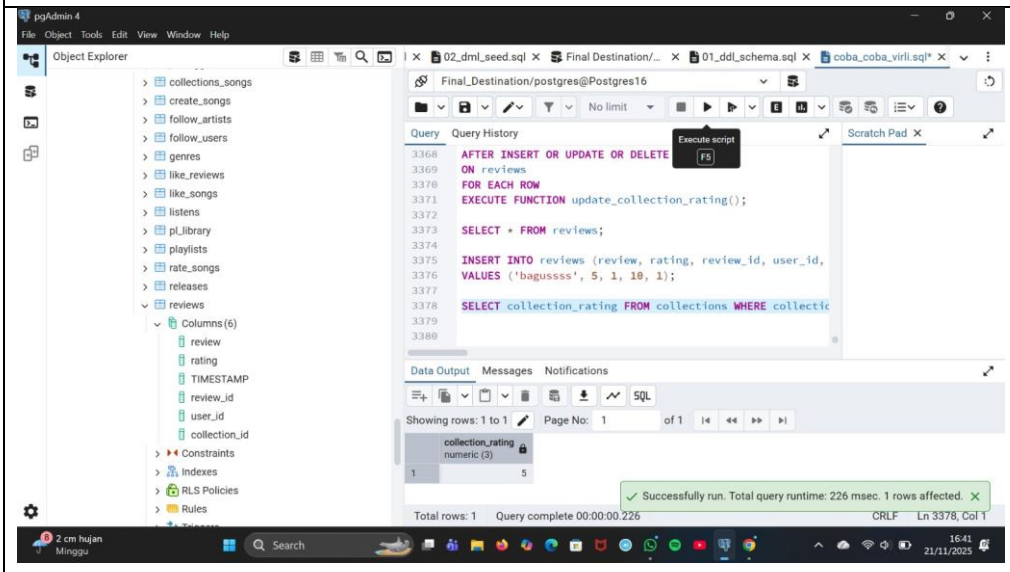


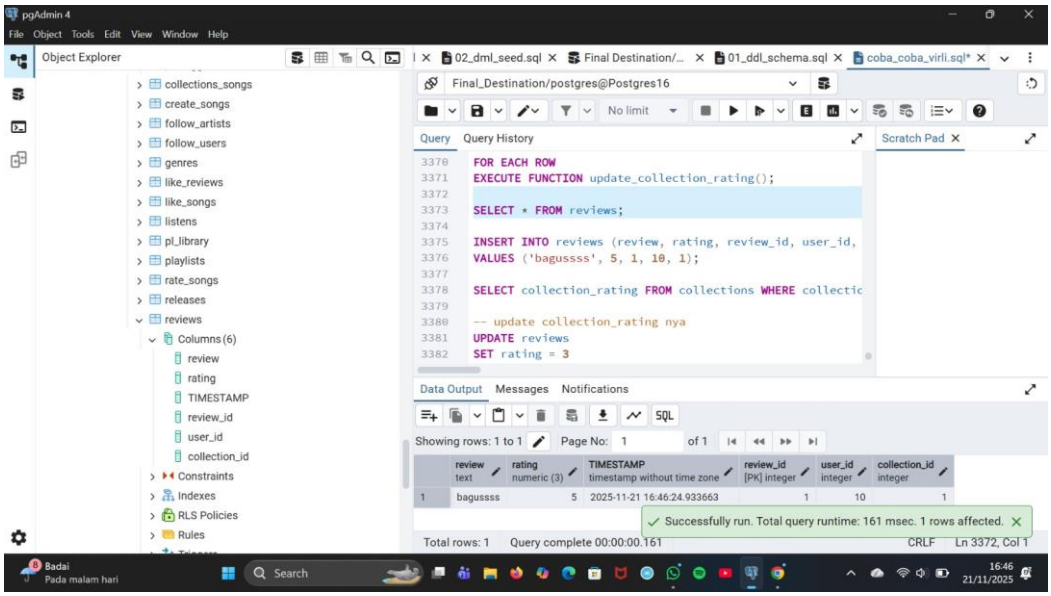
Tabel 72 DML – update collection rating

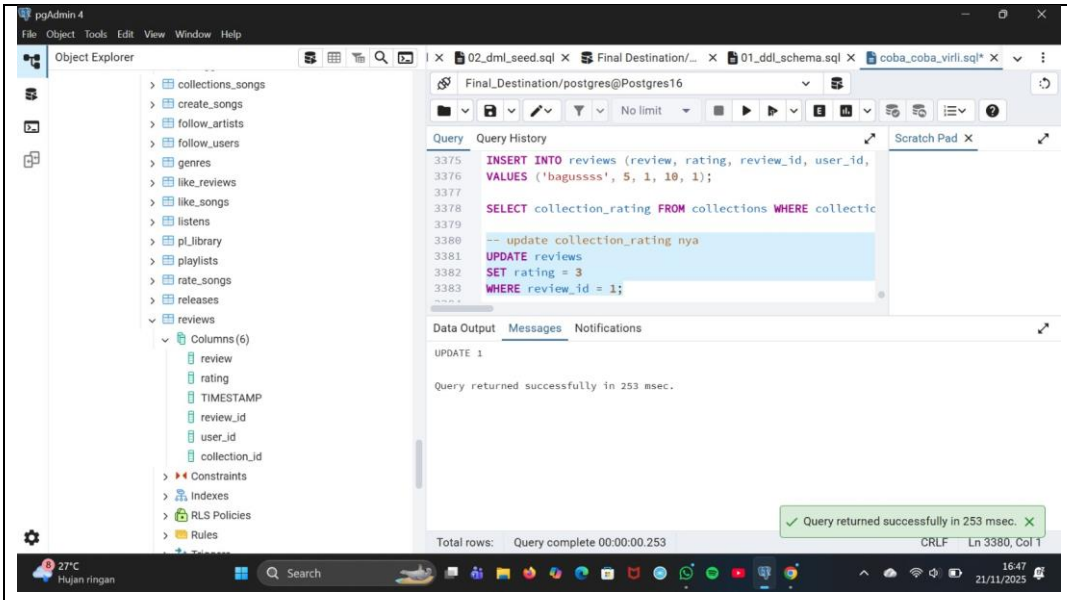
Nama Fitur	Update collection rating
Deskripsi	Menghitung ulang rata-rata rating pada tabel collections setiap kali ada perubahan pada tabel reviews. Nilai collection_rating diperbarui berdasarkan AVG(rating) dari semua review dengan collection_id yang sama.
Script SQL	
SELECT * FROM reviews; SELECT * FROM collections; INSERT INTO reviews (review, rating, review_id, user_id, collection_id) VALUES ('bagussss', 5, 1, 10, 1); INSERT INTO reviews (review, rating, review_id, user_id, collection_id) VALUES ('kerenn', 5, 2, 1, 1); SELECT collection_rating FROM collections WHERE collection_id = 1;	

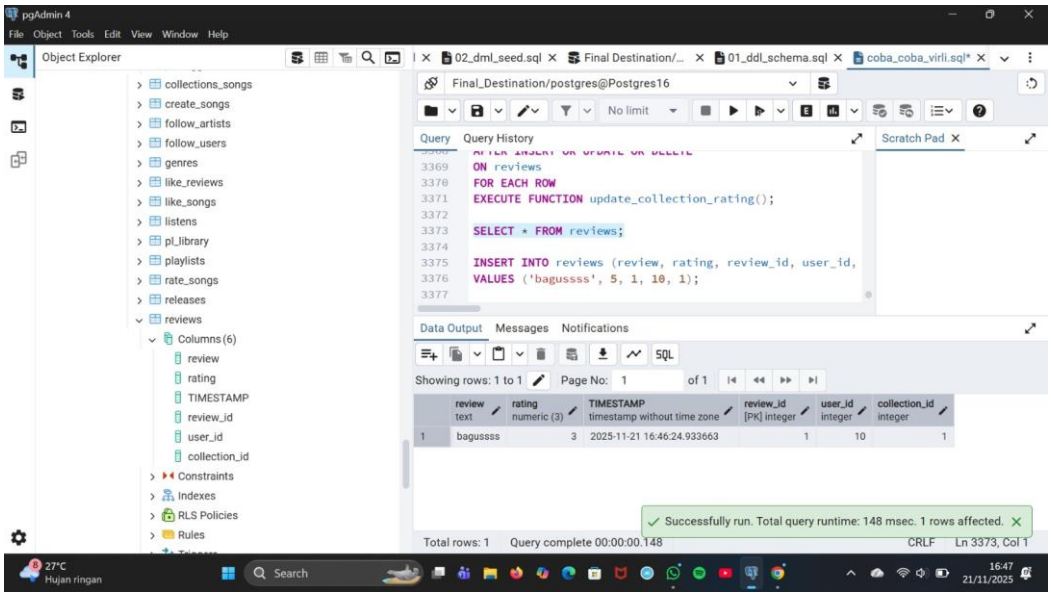
```
-- update collection_rating nya

UPDATE reviews

SET rating = 3

WHERE review_id = 1;


-- delete review

DELETE FROM reviews WHERE review_id = 1;
```

Screenshot Hasil

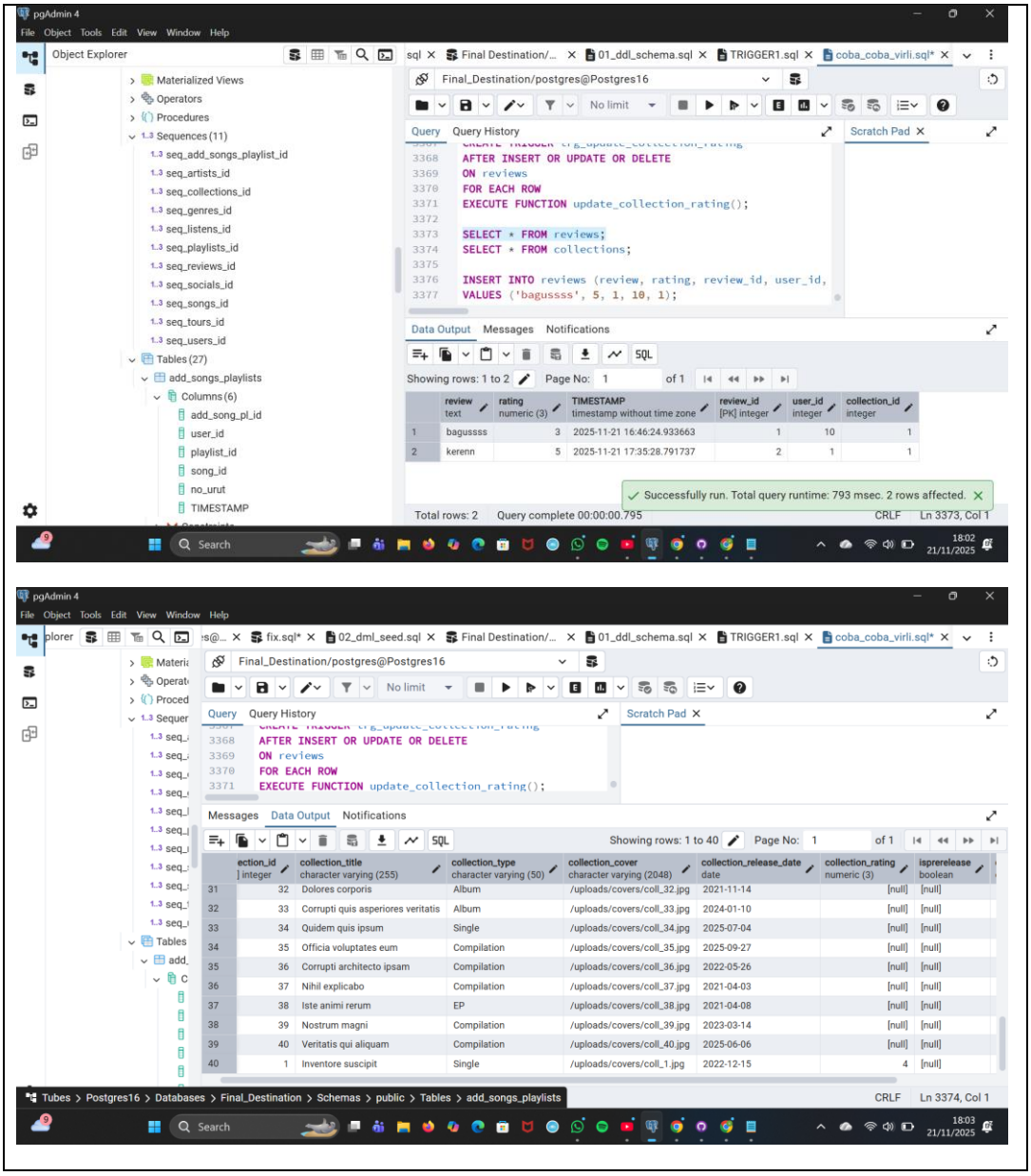








--- review baru dari user berbeda



Tabel 73 DML – Update collection top3 genres

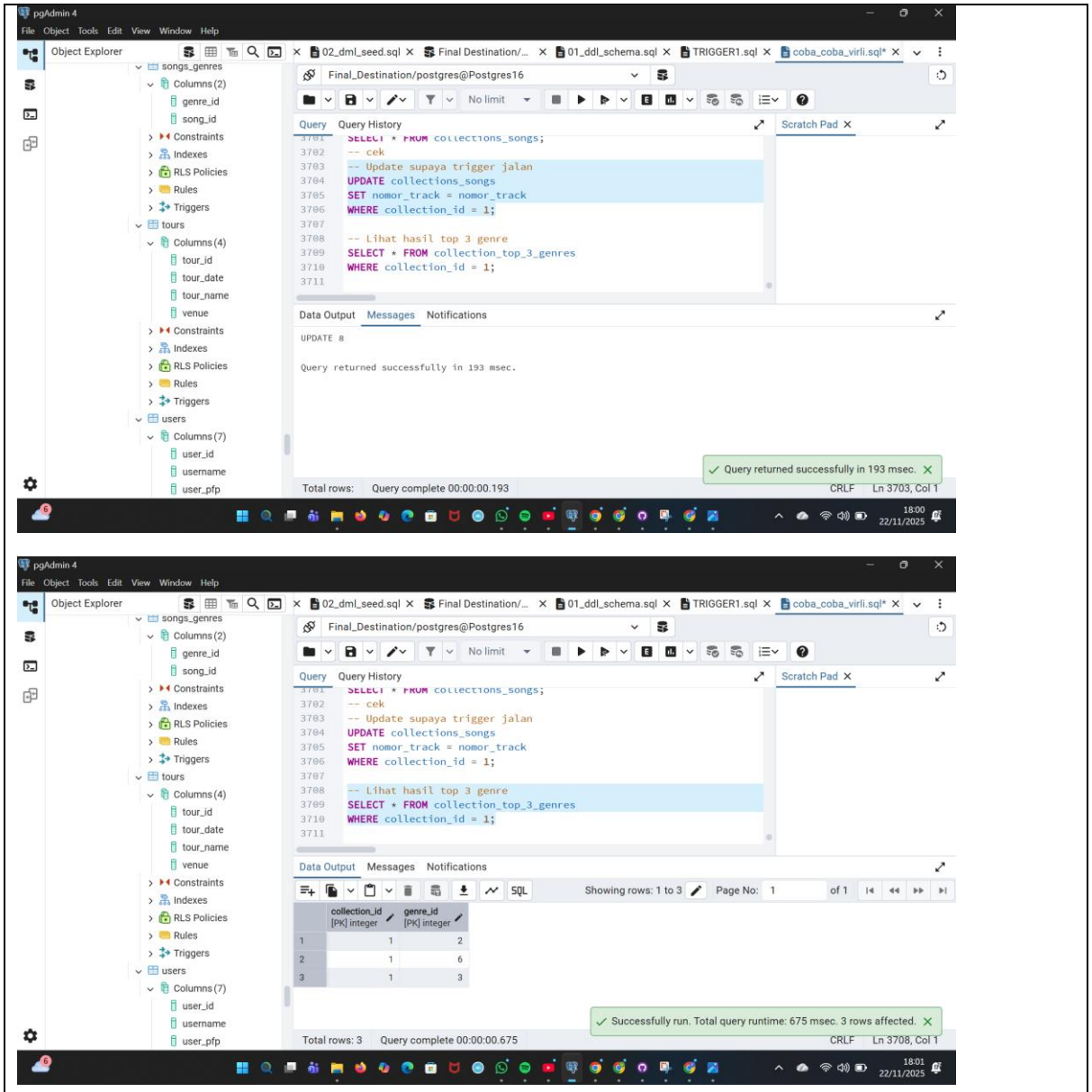
Nama Fitur	Update collection top3 genres
Deskripsi	Otomatis menghitung 3 genre terbanyak dalam sebuah collection setiap kali ada perubahan pada tabel collections_songs. Mekanismenya: 1. hapus data genre lama untuk collection tersebut, 2.hitung frekuensi tiap genre dari semua lagu melalui songs_genres,

	3.ambil 3 genre dengan jumlah tertinggi dan simpan ke collection_top_3_genres. Contoh: jika sebuah collection memiliki genre Pop (4), Jazz (2), Rock (2), dan Noise Pop (1), maka yang disimpan adalah Pop, Jazz, dan Rock sebagai top 3 genre.

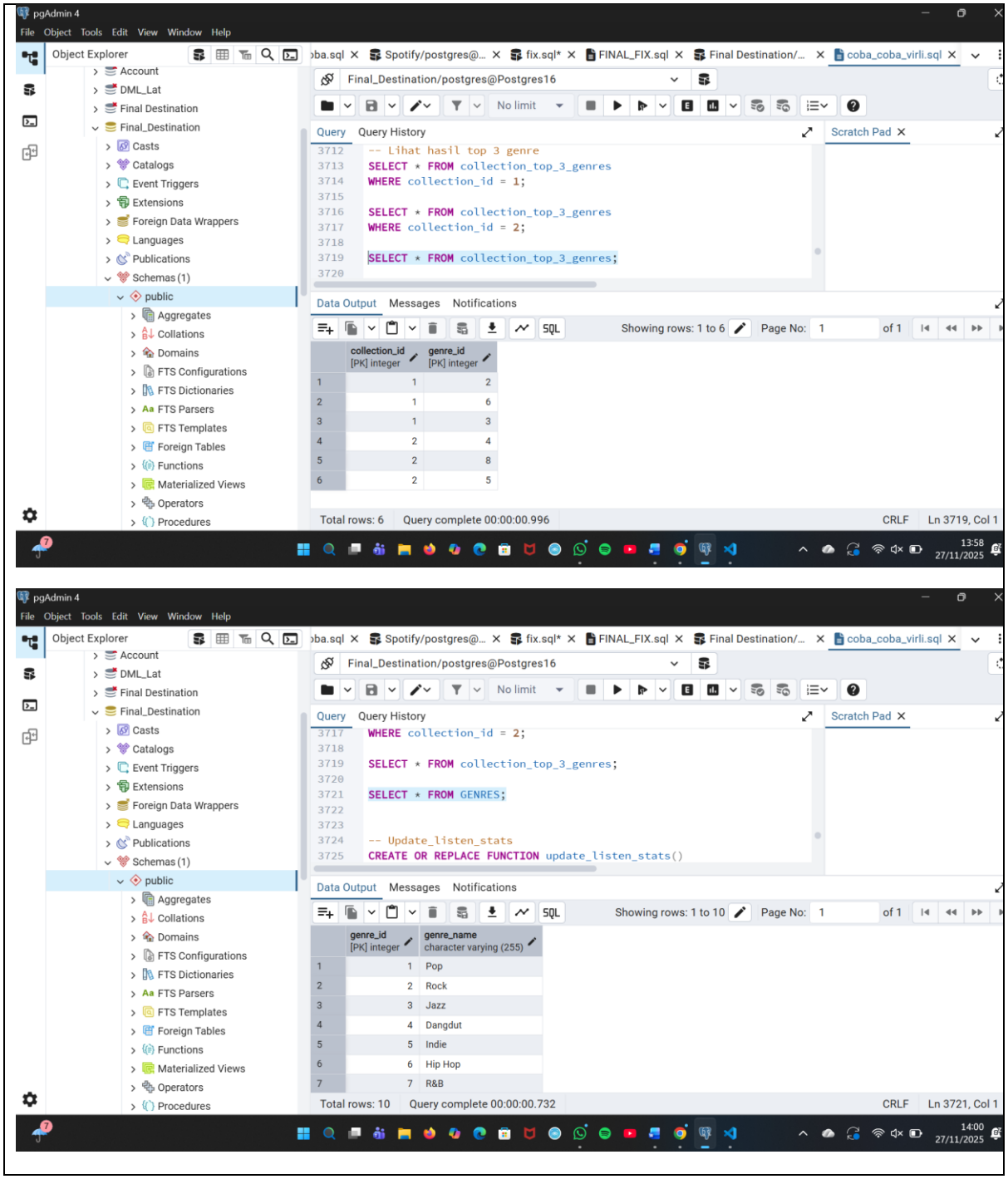
Script SQL

```
-- cek

-- Update supaya trigger jalan

UPDATE collections_songs

SET nomor_track = nomor_track

WHERE collection_id = 1;


UPDATE collections_songs

SET nomor_track = nomor_track

WHERE collection_id = 2;


-- Lihat hasil top 3 genre

SELECT * FROM collection_top_3_genres

WHERE collection_id = 1;


SELECT * FROM collection_top_3_genres

WHERE collection_id = 2;


SELECT * FROM collection_top_3_genres;
```

Screenshot Hasil


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
02_dml_seed.sql
Final Destination/...
01_ddl_schema.sql
TRIGGER1.sql
coba_coba_virli.sql*
Final_Destination/postgres@Postgres16
No limit
Query
Query History
Scratch Pad
3702 -- cek
3703 -- Update supaya trigger jalan
3704 UPDATE collections_songs
3705 SET nomor_track = nomor_track
3706 WHERE collection_id = 1;
3708 -- Lihat hasil top 3 genre
3709 SELECT * FROM collection_top_3_genres
3710 WHERE collection_id = 1;
Data Output
Messages
Notifications
UPDATE 8
Query returned successfully in 193 msec.
Query returned successfully in 193 msec.
Total rows:
Query complete 00:00:00.193
CRLF
Ln 3703, Col 1
songs_genres
Columns (2)
genre_id
song_id
Constraints
Indexes
RLS Policies
Rules
Triggers
tours
Columns (4)
tour_id
tour_date
tour_name
venue
users
Columns (7)
user_id
username
user_pfp
18:00
22/11/2025

pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
02_dml_seed.sql
Final Destination/...
01_ddl_schema.sql
TRIGGER1.sql
coba_coba_virli.sql*
Final_Destination/postgres@Postgres16
No limit
Query
Query History
Scratch Pad
3702 -- cek
3703 -- Update supaya trigger jalan
3704 UPDATE collections_songs
3705 SET nomor_track = nomor_track
3706 WHERE collection_id = 1;
3708 -- Lihat hasil top 3 genre
3709 SELECT * FROM collection_top_3_genres
3710 WHERE collection_id = 1;
Data Output
Messages
Notifications
Showing rows: 1 to 3
Page No: 1
of 1
collection_id [PK] integer
genre_id [PK] integer
1 1 2
2 1 6
3 1 3
Successfully run. Total query runtime: 675 msec. 3 rows affected.
Total rows: 3
Query complete 00:00:00.675
CRLF
Ln 3708, Col 1
songs_genres
Columns (2)
genre_id
song_id
Constraints
Indexes
RLS Policies
Rules
Triggers
tours
Columns (4)
tour_id
tour_date
tour_name
venue
users
Columns (7)
user_id
username
user_pfp
18:01
22/11/2025



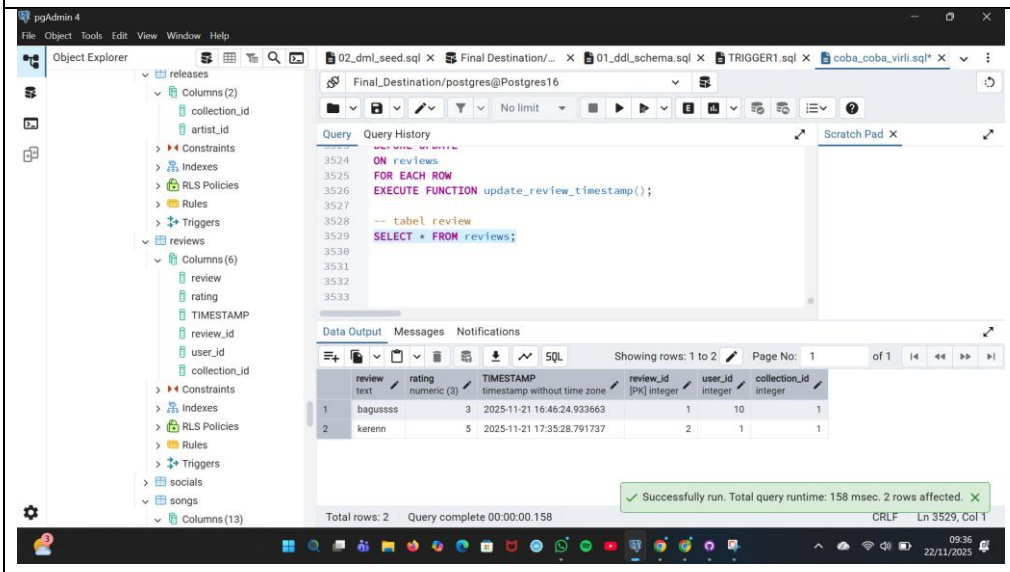
Tabel 74 DML – Update review timestamp

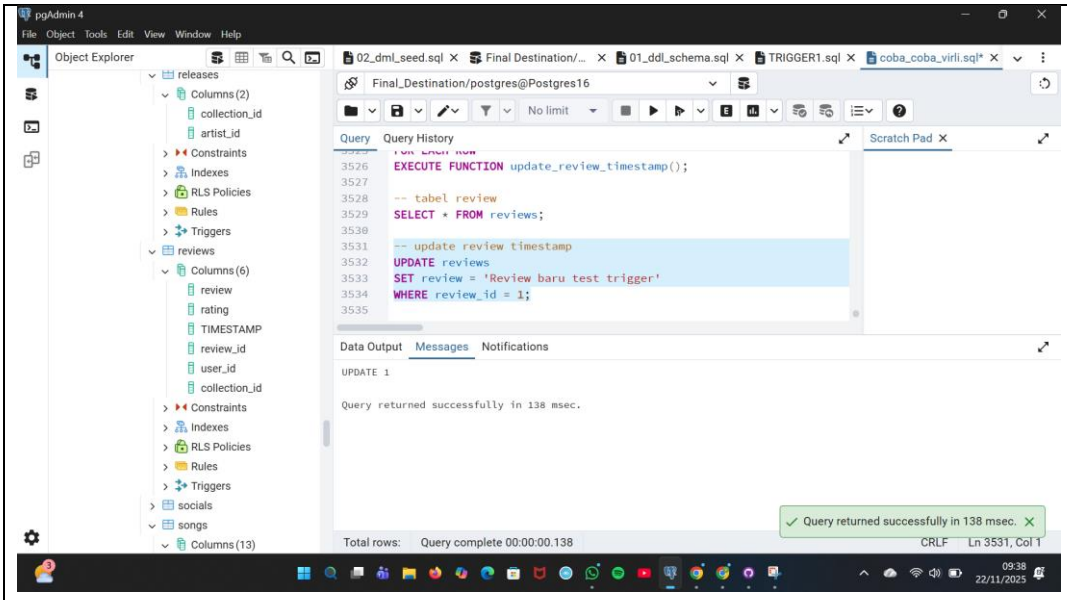
Nama Fitur	Update review timestamp
Deskripsi	Otomatis memperbarui kolom timestamp setiap kali isi review atau rating berubah. Jika salah satu di-update, nilai timestamp diganti dengan CURRENT_TIMESTAMP, sehingga riwayat perubahan review selalu tercatat dengan akurat.

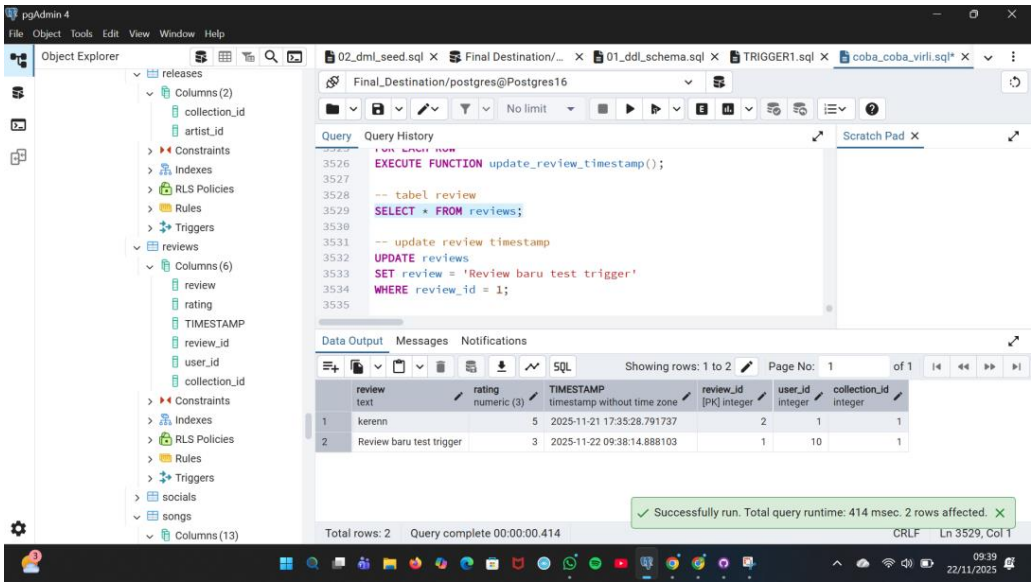
Script SQL

```
-- tabel review

SELECT * FROM reviews;


-- update review timestamp

UPDATE reviews

SET review = 'Review baru test trigger'

WHERE review_id = 1;


-- update rating timestamp

UPDATE reviews

SET rating = rating + 1

WHERE review_id = 1;


SELECT * FROM COLLECTIONS;
```

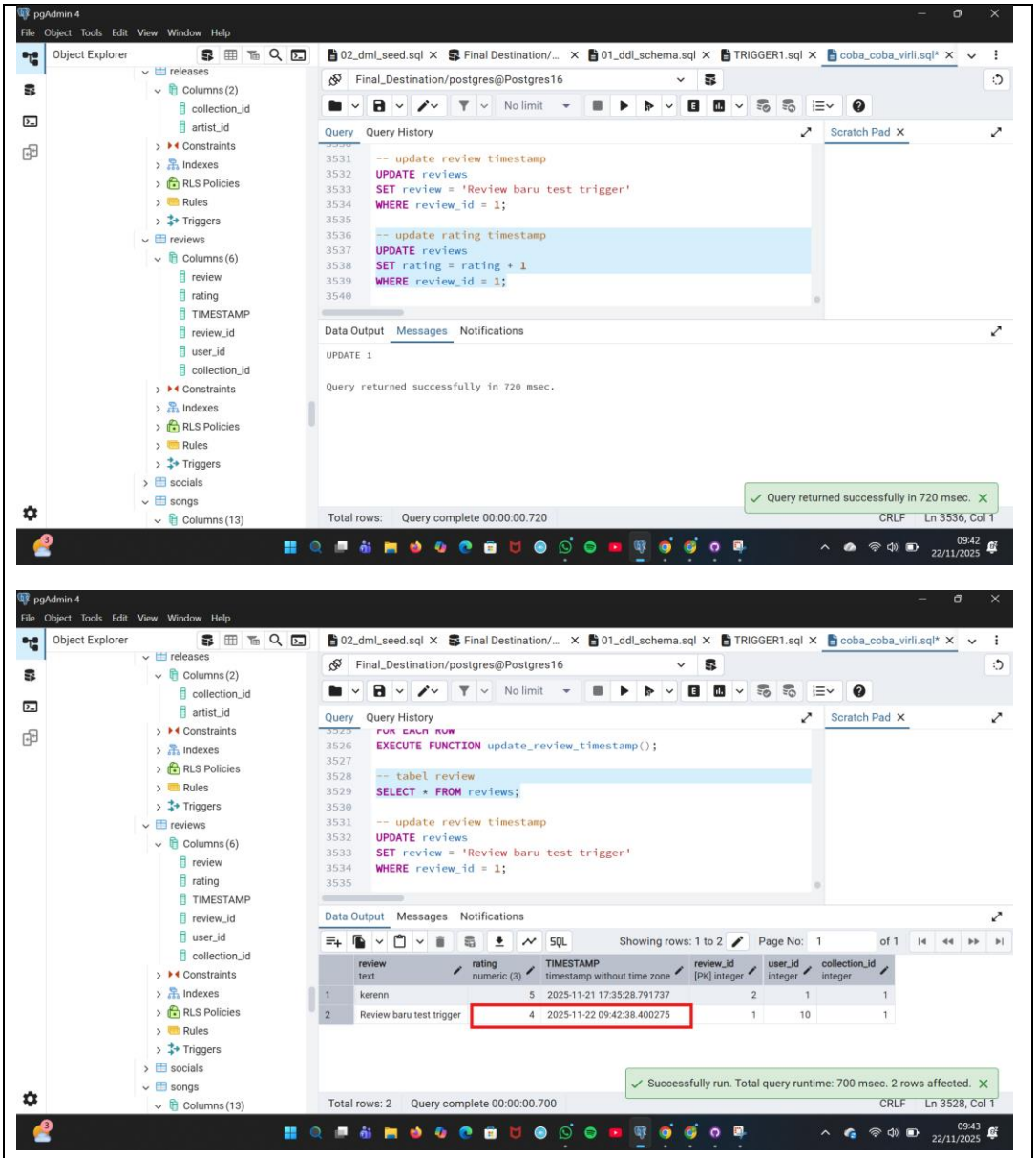
Screenshot Hasil







-- cek update rating



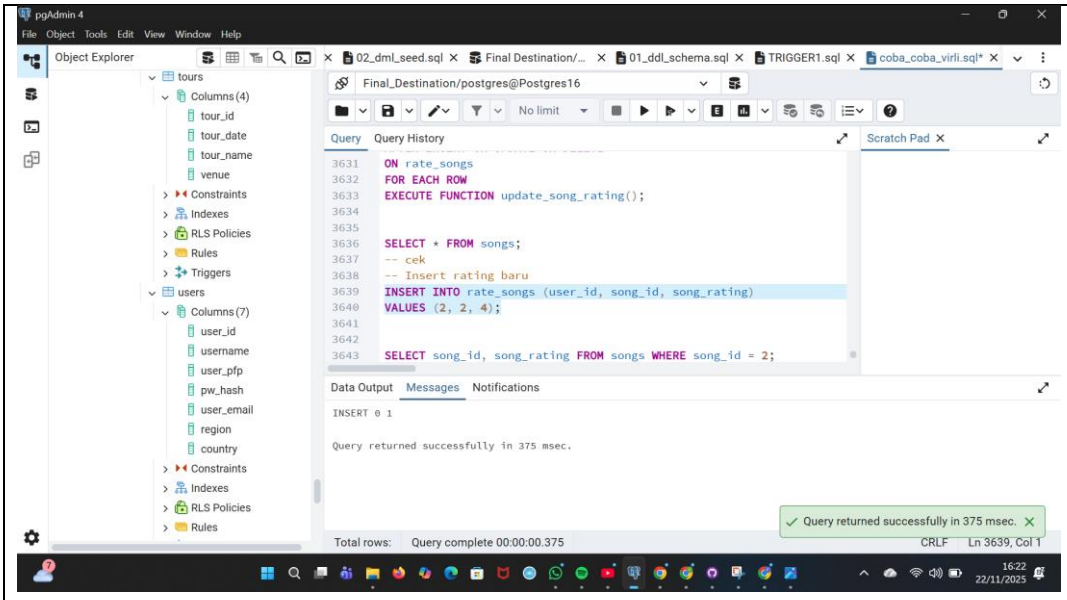
Tabel 75 DML – Update song rating

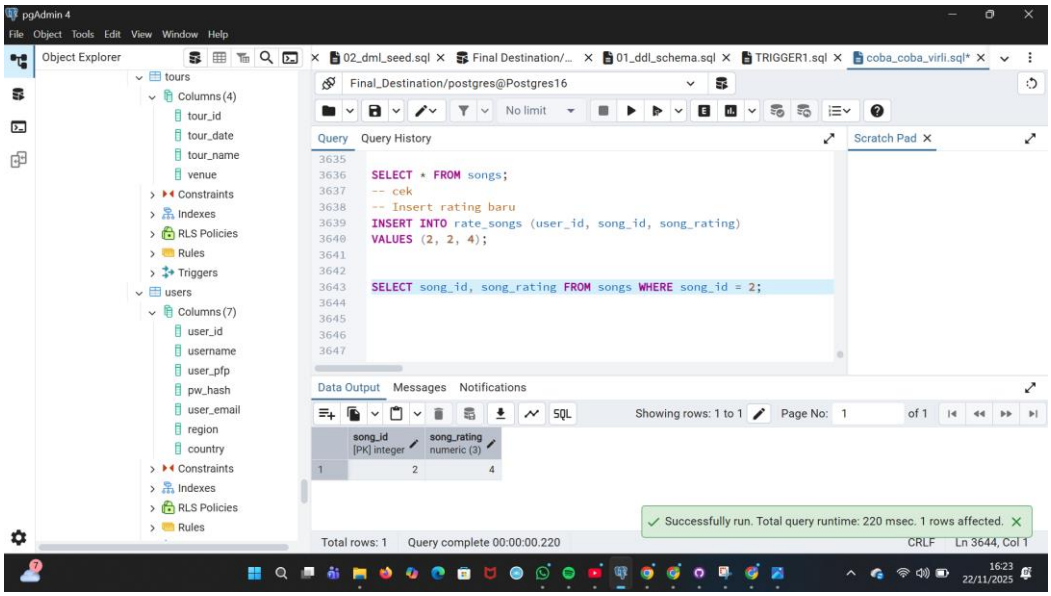
Nama Fitur	Update song rating
Deskripsi	Ketika ada INSERT, UPDATE, atau DELETE pada RATE_SONGS. Trigger akan menghitung ulang rata-rata rating untuk setiap song_id menggunakan AVG(song_rating) dari semua rating yang ada, lalu menyimpannya ke SONGS.song_rating. Dengan begitu, nilai rating lagu selalu akurat dan terbaru.
Script SQL	
SELECT * FROM songs;	

```
-- cek

-- Insert rating baru

INSERT INTO rate_songs (user_id, song_id, song_rating)

VALUES (2, 2, 4);


-- Update rating song

UPDATE rate_songs

SET song_rating = 5

WHERE user_id = 2 AND song_id = 2;


-- Update rating song

UPDATE rate_songs

SET song_rating = 5

WHERE user_id = 3 AND song_id = 2;


-- Insert baru

INSERT INTO rate_songs (user_id, song_id, song_rating)

VALUES (3, 2, 3);


-- Delete

DELETE FROM rate_songs

WHERE user_id = 2 AND song_id = 2;


SELECT song_id, song_rating FROM songs WHERE song_id = 2;
```

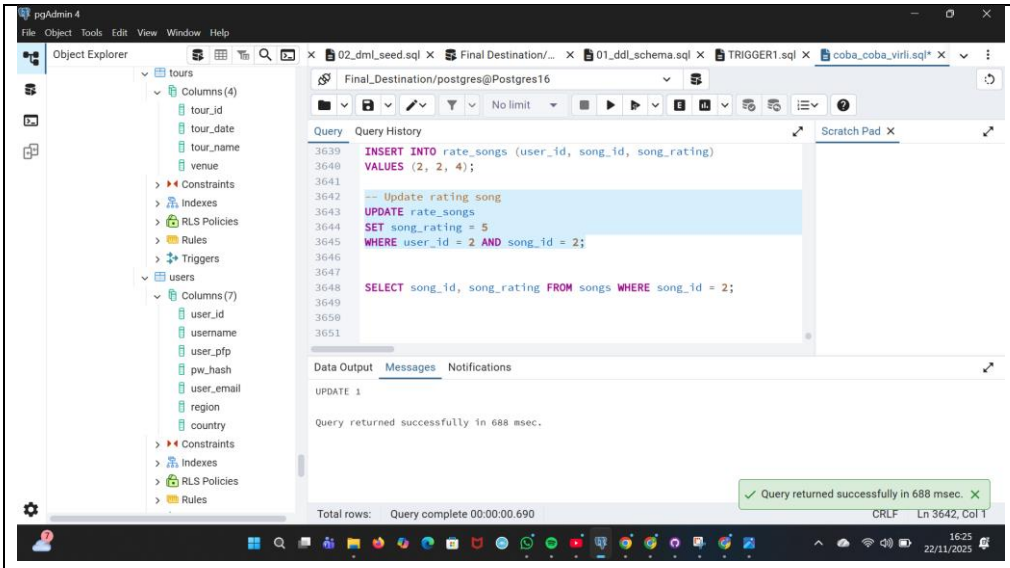
Screenshot Hasil

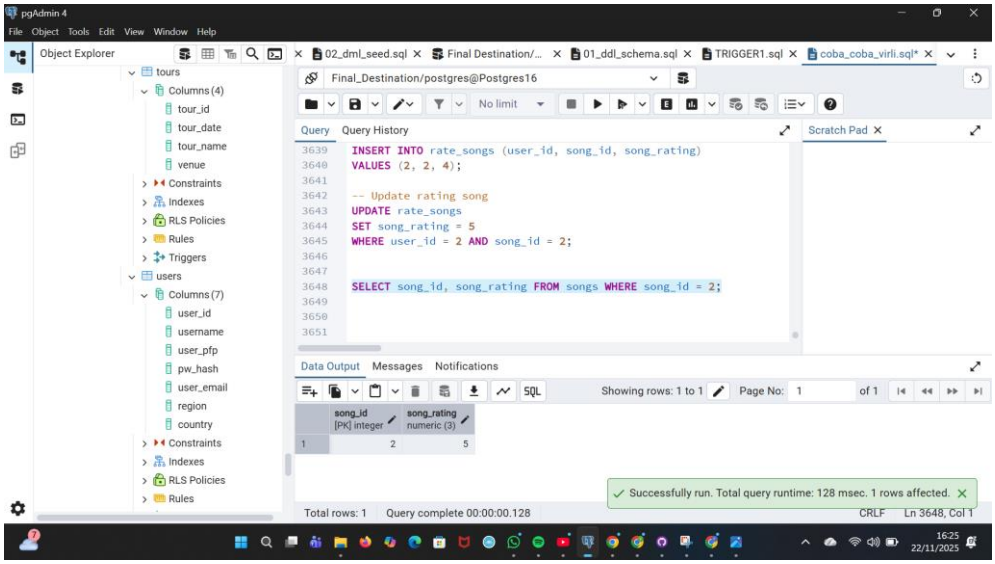
--- insert rating baru dengan user_id 2 rating_song 4



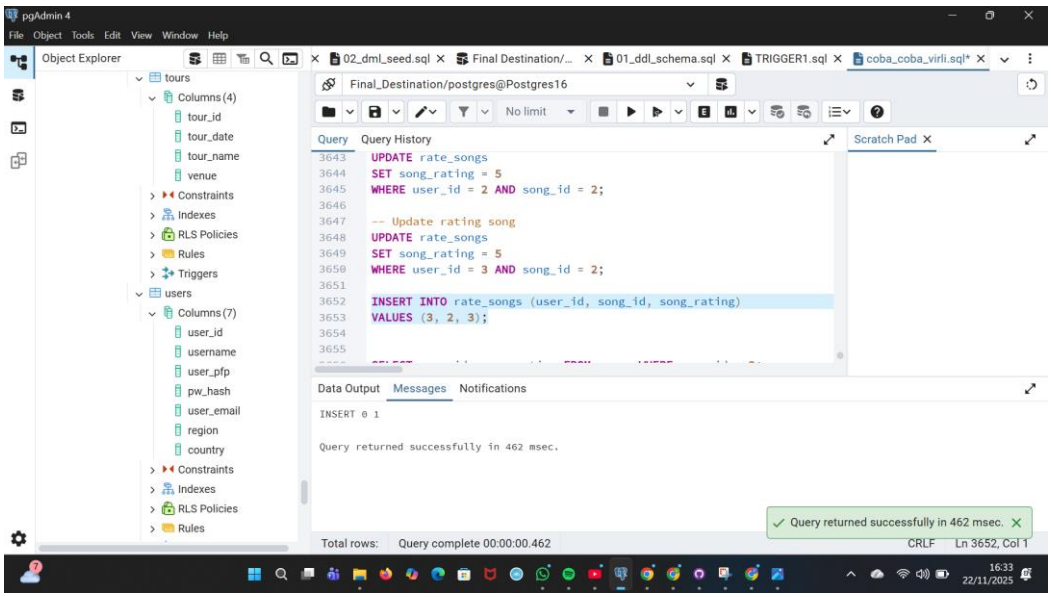


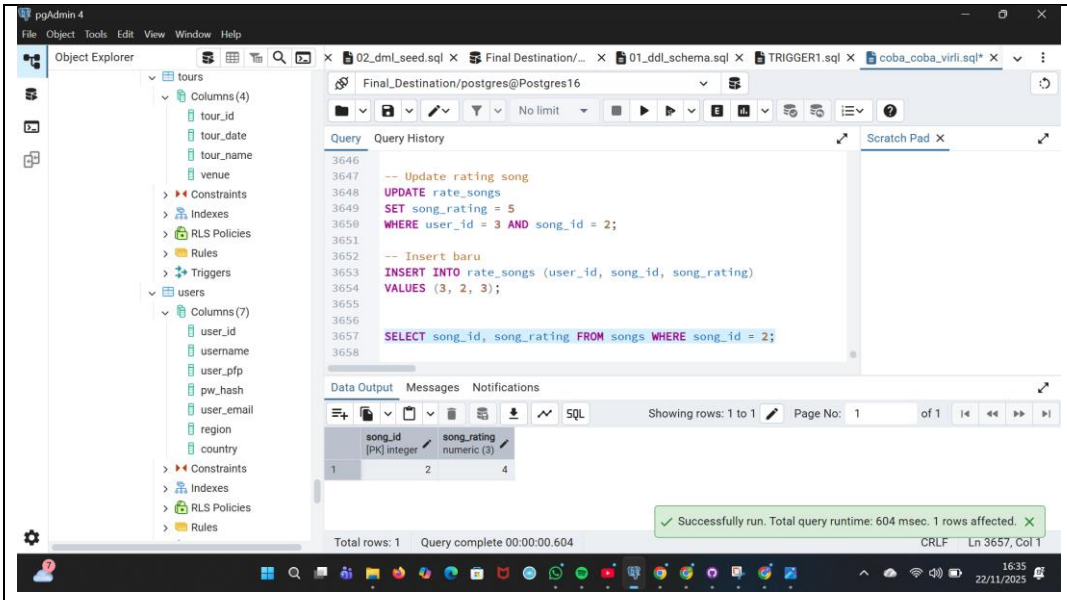
-- update rating song



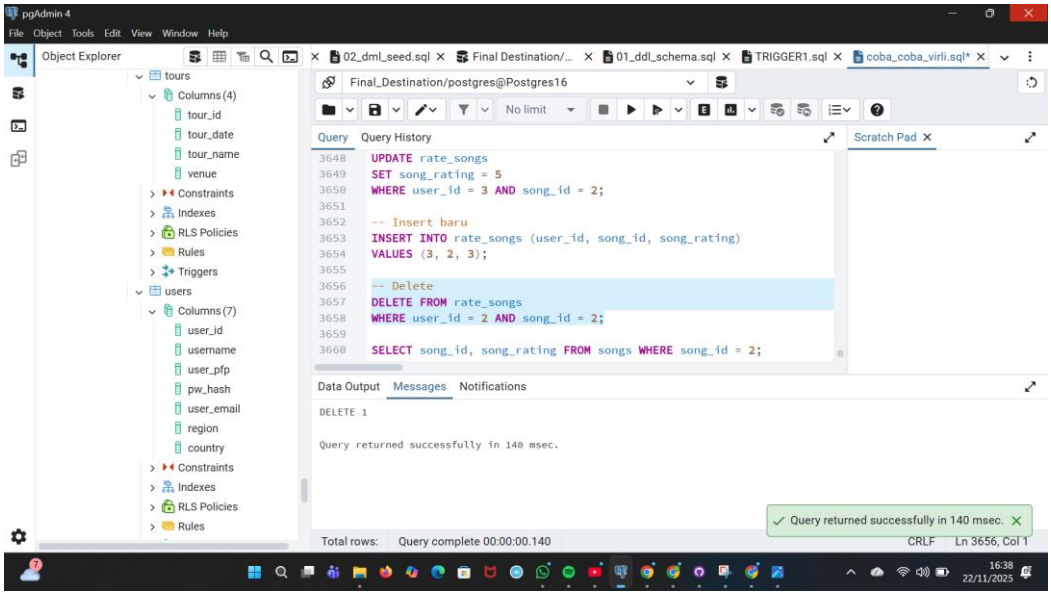


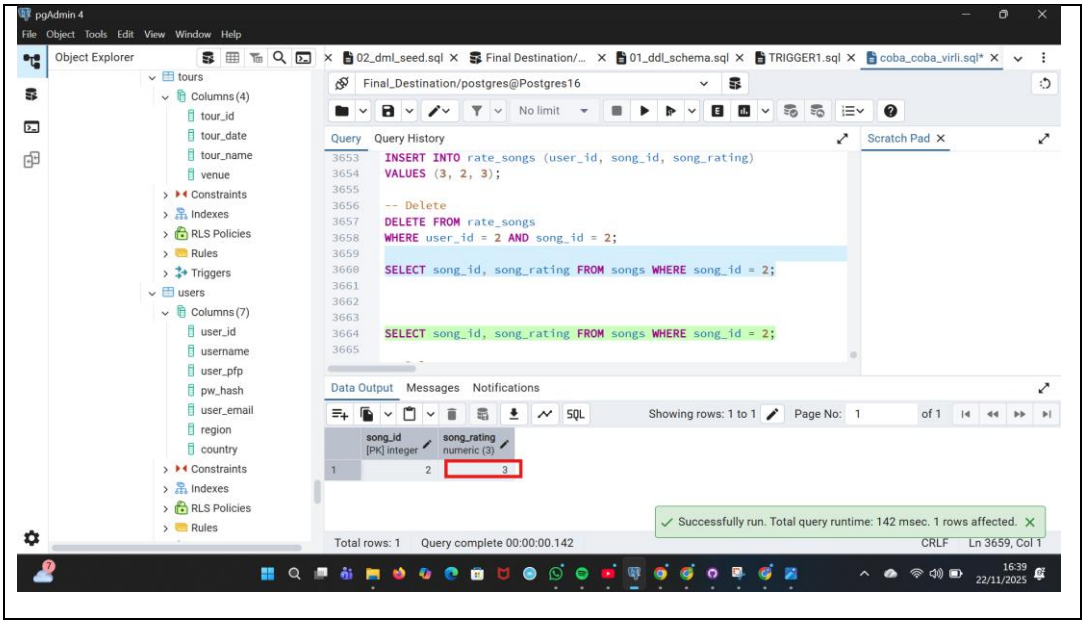
-- insert baru dengan rating 3 user id nya 3




pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
02_dml_seed.sql
Final Destination/...
01_ddl_schema.sql
TRIGGER1.sql
coba_coba_virli.sql*
Final_Destination/postgres@Postgres16
No limit
Query
Query History
Scratch Pad
-- Update rating song
UPDATE rate_songs
SET song_rating = 5
WHERE user_id = 3 AND song_id = 2;
-- Insert baru
INSERT INTO rate_songs (user_id, song_id, song_rating)
VALUES (3, 2, 3);
SELECT song_id, song_rating FROM songs WHERE song_id = 2;
Data Output
Messages
Notifications
Showing rows: 1 to 1
Page No: 1
of 1
song_id [PK] integer
song_rating numeric (3)
1
2
4
Successfully run. Total query runtime: 604 msec. 1 rows affected.
Total rows: 1
Query complete 00:00:00.604
CRLF
Ln 3657, Col 1
16:35
22/11/2025

-- Delete


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
02_dml_seed.sql
Final Destination/...
01_ddl_schema.sql
TRIGGER1.sql
coba_coba_virli.sql*
Final_Destination/postgres@Postgres16
No limit
Query
Query History
Scratch Pad
UPDATE rate_songs
SET song_rating = 5
WHERE user_id = 3 AND song_id = 2;
-- Insert baru
INSERT INTO rate_songs (user_id, song_id, song_rating)
VALUES (3, 2, 3);
-- Delete
DELETE FROM rate_songs
WHERE user_id = 2 AND song_id = 2;
SELECT song_id, song_rating FROM songs WHERE song_id = 2;
Data Output
Messages
Notifications
DELETE 1
Query returned successfully in 140 msec.
Query returned successfully in 140 msec.
Total rows:
Query complete 00:00:00.140
CRLF
Ln 3656, Col 1
16:38
22/11/2025



5.3 PL/SQL

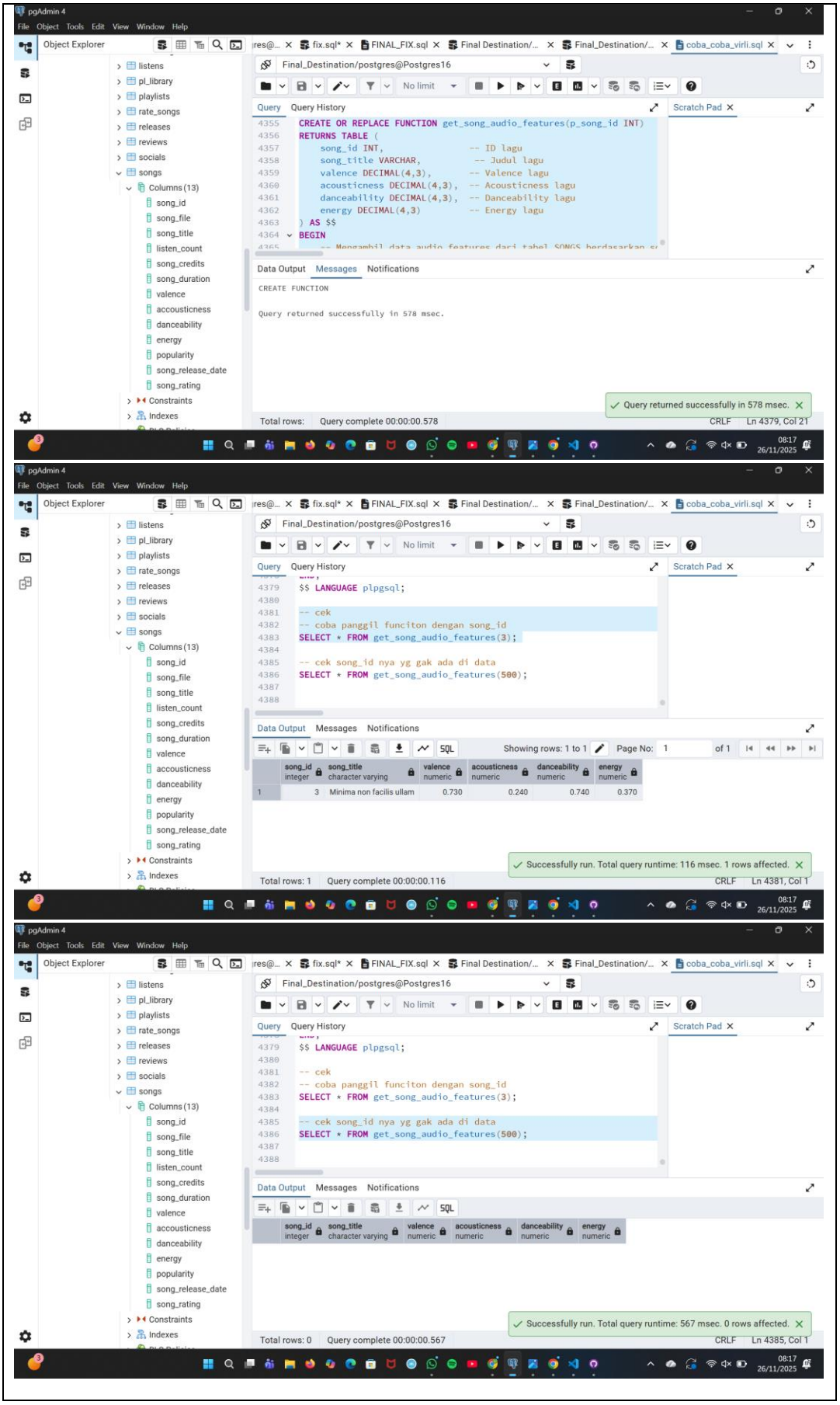
Tabel 76 PL/SQL – Function get_song_audio_features

Nama Function	get_song_audio_features
Parameter	p_song_id IN INT – ID lagu yang ingin diambil fiturnya
Deskripsi	Function ini digunakan untuk mengambil fitur audio numerik dari sebuah lagu berdasarkan song_id. Fitur audio ini mencerminkan karakteristik musik dari lagu, seperti “positif/ceria”, akustik, danceability, dan energi lagu. Function mengembalikan hasil berupa tabel dengan informasi lagu dan nilai fitur audio. Jika song_id tidak ditemukan, function mengembalikan tabel kosong.
Script SQL	

```
CREATE OR REPLACE FUNCTION get_song_audio_features(p_song_id INT)
RETURNS TABLE (
    song_id INT,                -- ID lagu
    song_title VARCHAR,          -- Judul lagu
```

```
    valence DECIMAL(4,3),        -- Valence lagu
    acousticness DECIMAL(4,3),   -- Acousticness lagu
    danceability DECIMAL(4,3),   -- Danceability lagu
    energy DECIMAL(4,3)          -- Energy lagu
) AS $$
BEGIN
    -- Mengambil data audio features dari tabel SONGS berdasarkan
song_id
    RETURN QUERY
    SELECT
        s.song_id,
        s.song_title,
        s.valence,  -- seberapa “positif/ceria”
        s.accousticness,   -- seberapa akustik
        s.danceability, -- dance
        s.energy -- seberapa enejik
    FROM songs s
    WHERE s.song_id = p_song_id;

    -- Jika song_id tidak ada, hasilnya akan kosong
END;
$$ LANGUAGE plpgsql;
```

Screenshot Hasil


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Final_Destination/postgres@Postgres16
Query Query History
Scratch Pad
CREATE OR REPLACE FUNCTION get_song_audio_features(p_song_id INT)
RETURNS TABLE (
song_id INT, -- ID lagu
song_title VARCHAR, -- Judul lagu
valence DECIMAL(4,3), -- Valence lagu
acousticness DECIMAL(4,3), -- Acousticness lagu
danceability DECIMAL(4,3), -- Danceability lagu
energy DECIMAL(4,3) -- Energy lagu
) AS $$
BEGIN
Data Output Messages Notifications
CREATE FUNCTION
Query returned successfully in 578 msec.
Query returned successfully in 578 msec.
Total rows: Query complete 00:00:00.578
CRLF Ln 4379, Col 21
$$ LANGUAGE plpgsql;
-- cek
-- coba panggil funciton dengan song_id
SELECT * FROM get_song_audio_features(3);
-- cek song_id nya yg gak ada di data
SELECT * FROM get_song_audio_features(500);
Showing rows: 1 to 1
Page No: 1 of 1
song_id integer
song_title character varying
valence numeric
acousticness numeric
danceability numeric
energy numeric
1 3 Minima non facilis ullam 0.730 0.240 0.740 0.370
Successfully run. Total query runtime: 116 msec. 1 rows affected.
Total rows: 1 Query complete 00:00:00.116
CRLF Ln 4381, Col 1
Successfully run. Total query runtime: 567 msec. 0 rows affected.
Total rows: 0 Query complete 00:00:00.567
CRLF Ln 4385, Col 1
listens
pl_library
playlists
rate_songs
releases
reviews
socials
songs
Columns (13)
song_id
song_file
song_title
listen_count
song_credits
song_duration
valence
accousticness
danceability
energy
popularity
song_release_date
song_rating
Constraints
Indexes
08:17
26/11/2025

Tabel 77 PL/SQL - Function get_recently_played

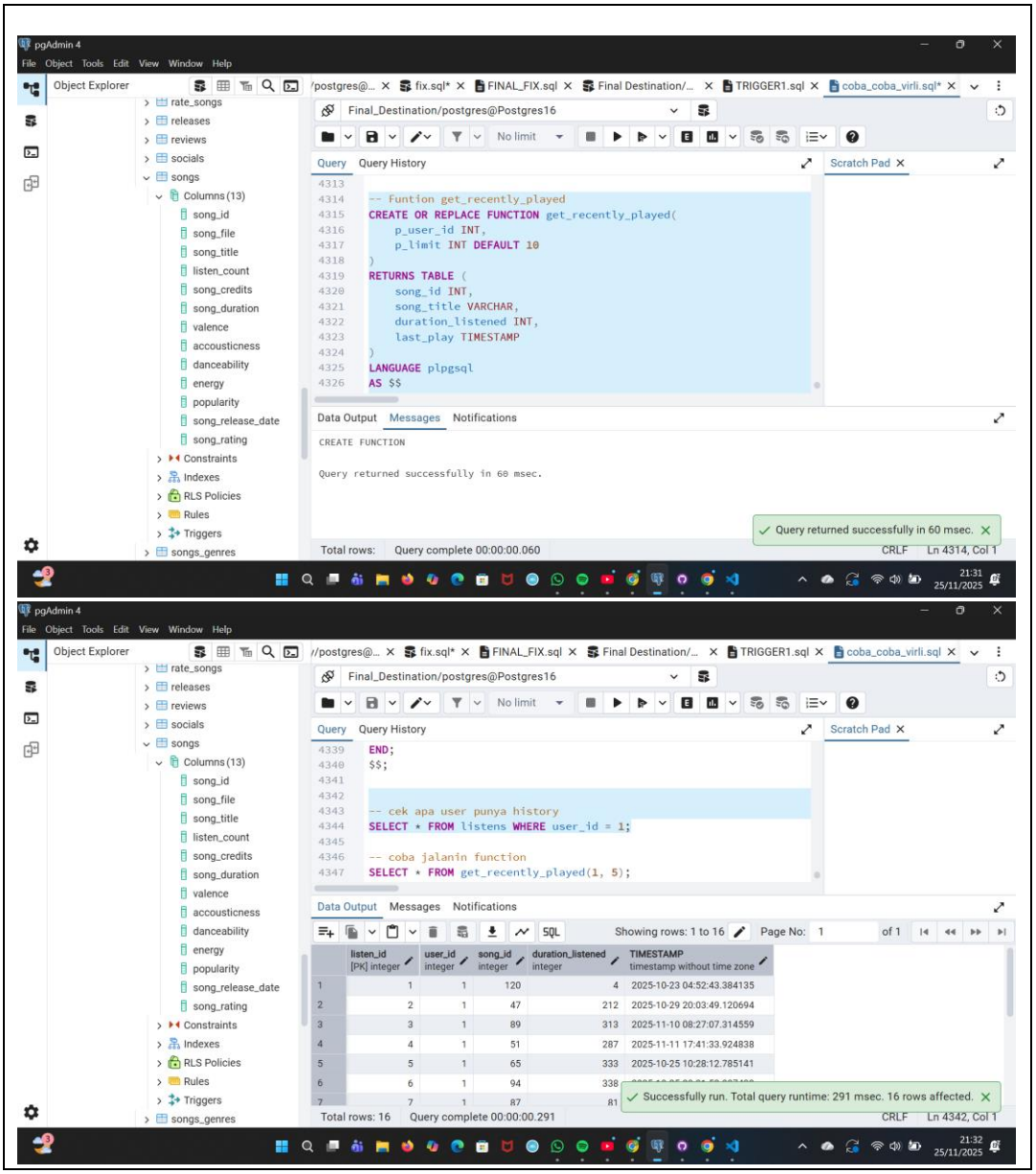
Nama Function	get_recently_played
Parameter	p_user_id INT --> ID user yang ingin diambil riwayat pemutarannya p_limit INT --> Jumlah maksimal lagu terbaru yang akan ditampilkan
Deskripsi	Function get_recently_played digunakan untuk mengambil daftar lagu yang paling terakhir diputar oleh seorang user. Data diambil dari tabel listens yang berisi catatan setiap kali user memutar lagu. Function ini menampilkan judul lagu, durasi pendengaran (dalam detik), dan timestamp pemutaran terakhir, kemudian mengurutkannya dari yang paling baru. Fitur yang disediakan: Mendapatkan riwayat pemutaran berdasarkan user_id. Mengurutkan berdasarkan waktu pemutaran (TIMESTAMP) secara descending. Limit output menggunakan parameter p_limit. Mengembalikan data dalam bentuk TABLE.

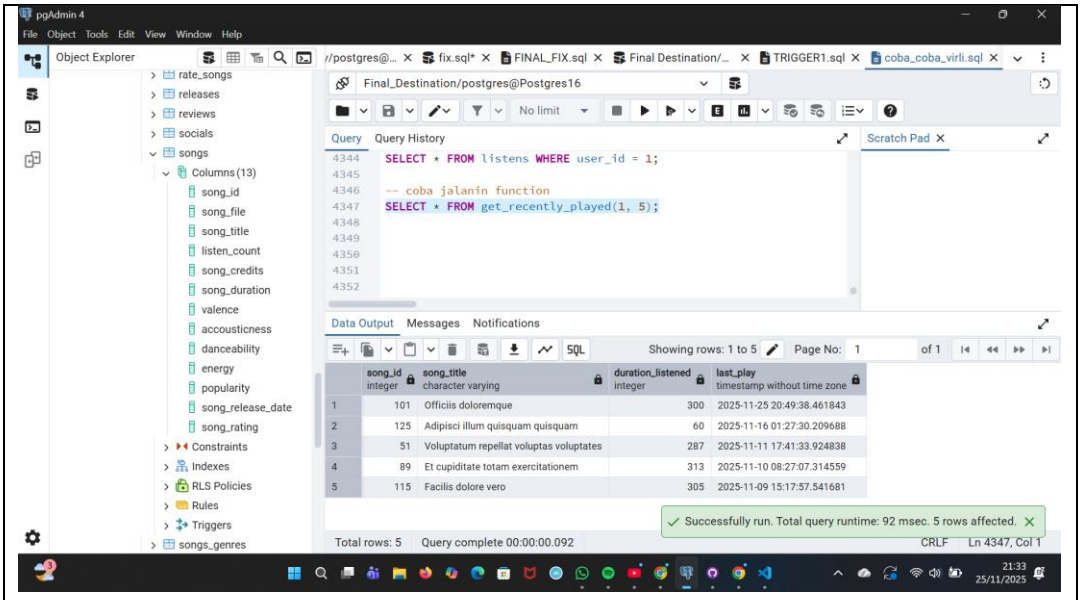
Script SQL

```
CREATE OR REPLACE FUNCTION get_recently_played(
    p_user_id INT,     -- Parameter input ID user
    p_limit INT DEFAULT 10 -- Limit jumlah hasil yang ditampilkan
)
```

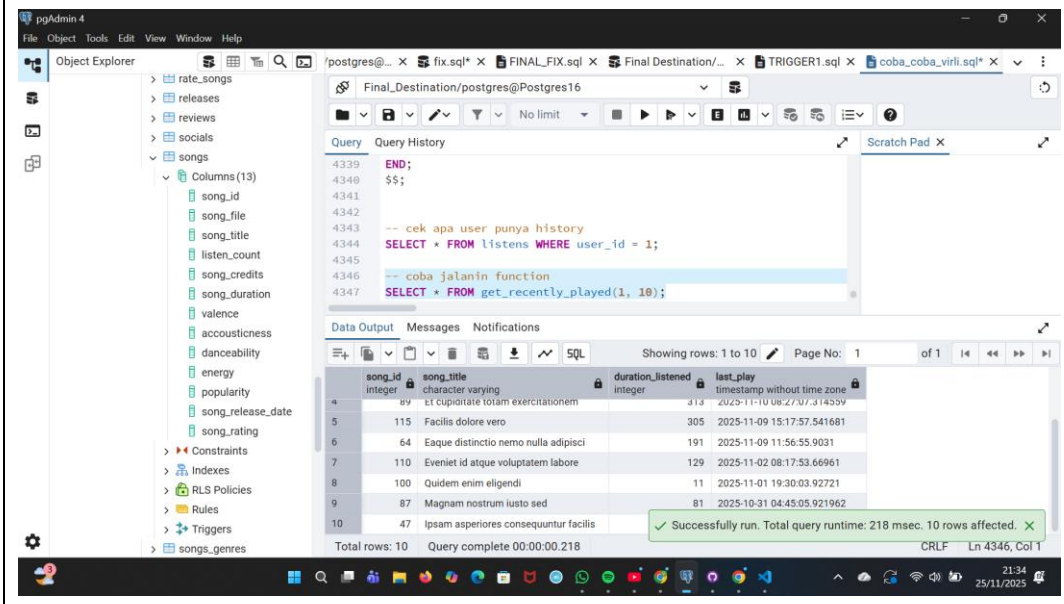
```
RETURNS TABLE (
    song_id INT,  -- Mengambil ID lagu dari tabel songs
    song_title VARCHAR,   -- Mengambil judul lagu dari tabel songs
    duration_listened INT,  -- Durasi mendengarkan dalam bentuk
detik
    last_play TIMESTAMP  -- Timestamp kapan lagu diputar
)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
    SELECT
        s.song_id,
        s.song_title,
        l.duration_listened, -- dalam detik
        l."TIMESTAMP" AS last_play
    FROM listens l
    JOIN songs s ON s.song_id = l.song_id  -- Join tabel listens
dengan songs berdasarkan song_id
    WHERE l.user_id = p_user_id   -- Memfilter hanya record yang
sesuai user yang diminta
    ORDER BY l."TIMESTAMP" DESC  -- Mengurutkan dari pemutaran
terbaru
    LIMIT p_limit;  -- Membatasi jumlah record sesuai parameter
p_limit
END;
$$;
```

Screenshot Hasil


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Final_Destination/postgres@Postgres16
Query Query History
Scratch Pad
-- Funtion get_recently_played
CREATE OR REPLACE FUNCTION get_recently_played(
p_user_id INT,
p_limit INT DEFAULT 10
)
RETURNS TABLE (
song_id INT,
song_title VARCHAR,
duration_listened INT,
last_play TIMESTAMP
)
LANGUAGE plpgsql
AS $$
Data Output Messages Notifications
CREATE FUNCTION
Query returned successfully in 60 msec.
Total rows:
Query complete 00:00:00.060
Query returned successfully in 60 msec.
CRLF Ln 4314, Col 1
END;
$$;
-- cek apa user punya history
SELECT * FROM listens WHERE user_id = 1;
-- coba jalanin function
SELECT * FROM get_recently_played(1, 5);
Showing rows: 1 to 16
Page No: 1 of 1
listen_id [PK] integer
user_id integer
song_id integer
duration_listened integer
TIMESTAMP timestamp without time zone
1 1 1 120 4 2025-10-23 04:52:43.384135
2 2 1 47 212 2025-10-29 20:03:49.120694
3 3 1 89 313 2025-11-10 08:27:07.314559
4 4 1 51 287 2025-11-11 17:41:33.924838
5 5 1 65 333 2025-10-25 10:28:12.785141
6 6 1 94 338
7 7 1 87 81
Successfully run. Total query runtime: 291 msec. 16 rows affected.
Total rows: 16
Query complete 00:00:00.291
CRLF Ln 4342, Col 1



-- jika di ganti jadi 10 riwayat lagu terakhir



Tabel 78 PL/SQL – Function get_song_detail

Nama Function	get_song_detail
Parameter	p_song_id IN NUMBER
Deskripsi	Function ini digunakan untuk mengambil metadata lengkap sebuah lagu berdasarkan song_id. Data yang dikembalikan mencakup informasi lagu, artis pembuat lagu, genre, serta

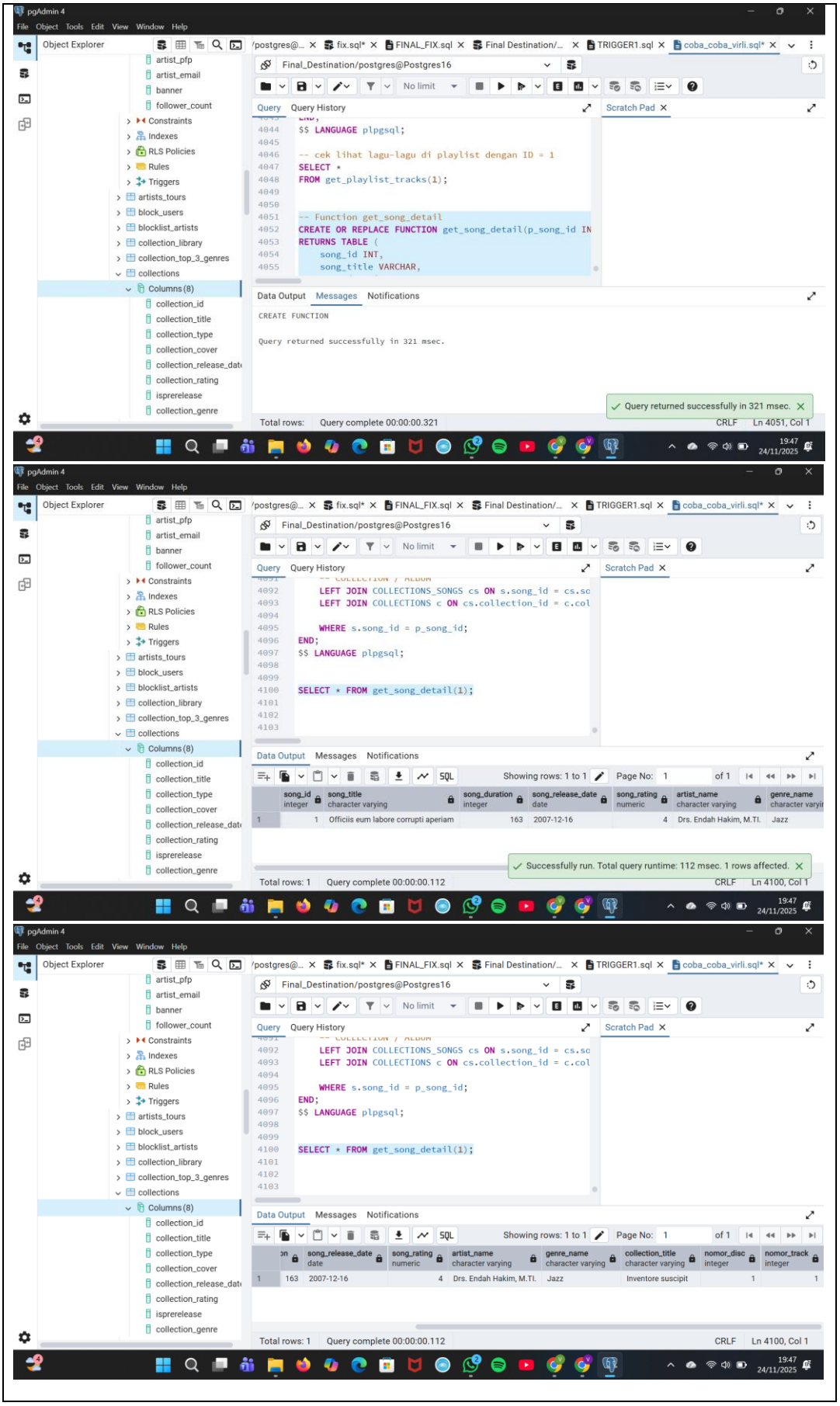
	album/collection tempat lagu berada. Function menggabungkan beberapa tabel: songs, create_songs, artists, songs_genres, genres, collections_songs, dan collections. Output berupa satu baris per kombinasi lagu–genre–collection.

Script SQL

```
CREATE OR REPLACE FUNCTION get_song_detail(p_song_id INT)
RETURNS TABLE (
    song_id INT,
    song_title VARCHAR,
    song_duration INT,
    song_release_date DATE,
    song_rating NUMERIC,
    artist_name VARCHAR,
    genre_name VARCHAR,
    collection_title VARCHAR,
    nomor_disc INT,
    nomor_track INT
)
AS $$
BEGIN
    RETURN QUERY
    SELECT
        -- Ambil ID lagu, judul lagu, durasi lagu, tanggal rilis
lagu, rating lagu
        s.song_id,
        s.song_title,
        s.song_duration,
        s.song_release_date,
        s.song_rating,

        a.artist_name,                -- artis dari CREATE_SONGS)
        g.genre_name,                 -- genre lagu dari
SONGS_GENRES)
        c.collection_title,           -- dari collection
        cs.nomor_disc,                -- nomor disc lagu dalam
collection
        cs.nomor_track                -- nomor track lagu dalam
collection

    FROM SONGS s

    -- ARTIST
```

```
    LEFT JOIN CREATE_SONGS cr ON s.song_id = cr.song_id  -- Join
untuk dapat artist_id dari CREATE_SONGS
    LEFT JOIN ARTISTS a ON cr.artist_id = a.artist_id

    -- GENRE
    LEFT JOIN SONGS_GENRES sg ON s.song_id = sg.song_id
    LEFT JOIN GENRES g ON sg.genre_id = g.genre_id

    -- COLLECTION / ALBUM
    LEFT JOIN COLLECTIONS_SONGS cs ON s.song_id = cs.song_id
    LEFT JOIN COLLECTIONS c ON cs.collection_id = c.collection_id

    WHERE s.song_id = p_song_id;   -- Filter hanya lagu dengan ID
sesuai parameter
END;
$$ LANGUAGE plpgsql;
```

Screenshot Hasil


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Final_Destination/postgres@Postgres16
Query Query History
Scratch Pad
Data Output Messages Notifications
CREATE FUNCTION
Query returned successfully in 321 msec.
Total rows: Query complete 00:00:00.321
Query returned successfully in 321 msec.
Ln 4051, Col 1
Showing rows: 1 to 1
Page No: 1 of 1
Officiis eum labore corrupti aperiam
Drs. Endah Hakim, M.TI.
Jazz
Successfully run. Total query runtime: 112 msec. 1 rows affected.
Total rows: 1 Query complete 00:00:00.112
Ln 4100, Col 1
Inventore suscipit
24/11/2025

Tabel 79 PL/SQL – Function get_playlist_tracks

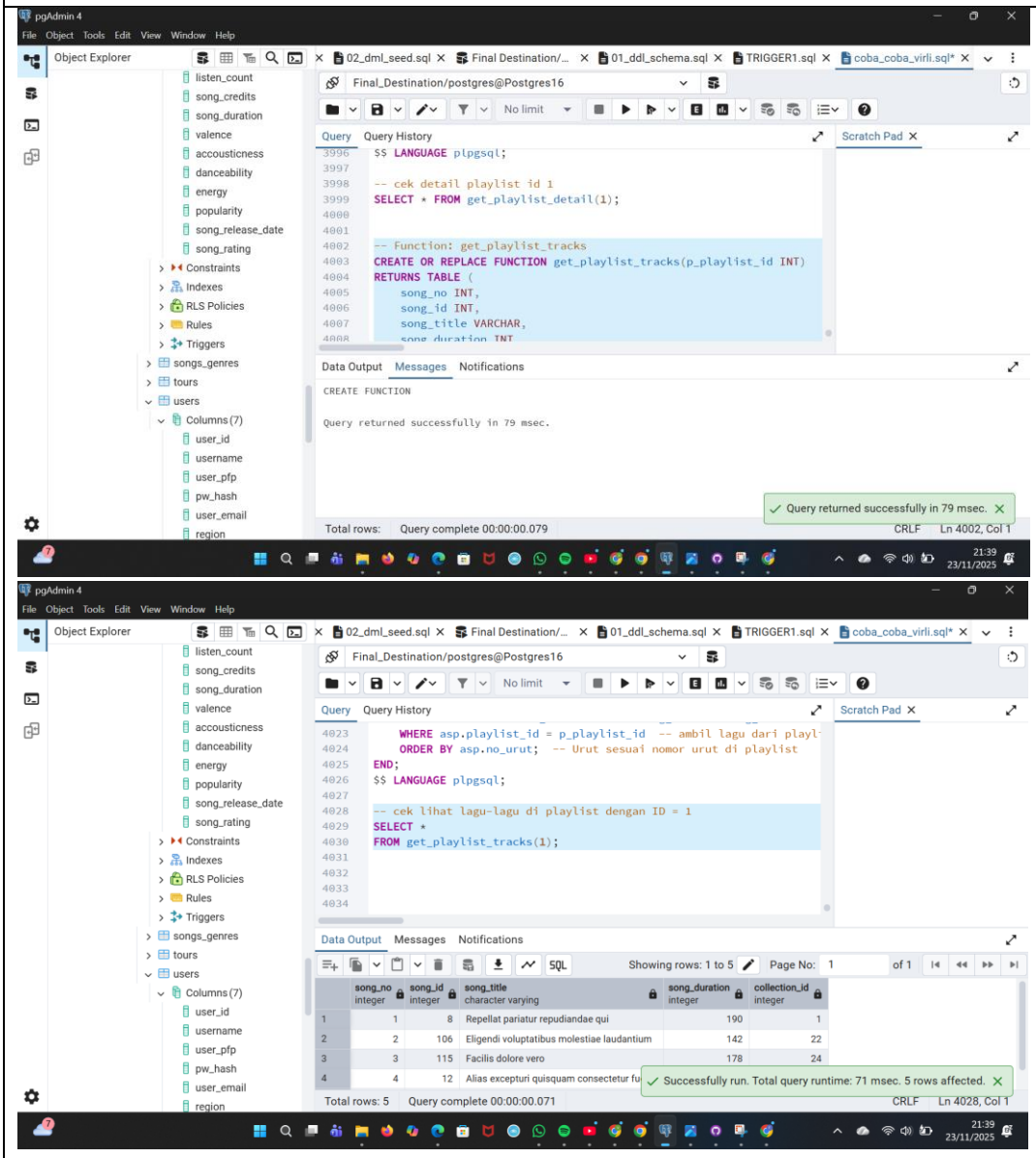
Nama Function	get_playlist_tracks
Parameter	p_playlist_id IN INT
Deskripsi	Mengambil semua lagu yang ada di suatu playlist berdasarkan playlist_id. Menampilkan detail lagu dari tabel SONGS dan informasi terkait koleksi (album) dari tabel COLLECTIONS_SONGS. Nomor urut lagu mengikuti urutan penambahan di playlist (NO_URUT dari ADD_SONGS_PLAYLISTS).

Script SQL

```
CREATE OR REPLACE FUNCTION get_playlist_tracks(p_playlist_id INT)
RETURNS TABLE (
    song_no INT,
    song_id INT,
    song_title VARCHAR,
    song_duration INT,
    collection_id INT
)
AS $$
BEGIN
    RETURN QUERY
    SELECT
        asp.no_urut AS song_no, -- nomor urut lagu di playlist
        s.song_id,
        s.song_title,
        s.song_duration,
        cs.collection_id   -- null jika lagu tidak ada di collection
    FROM ADD_SONGS_PLAYLISTS asp  -- Ambil tabel lagu yang ditambahkan
ke playlist (alias asp)
    JOIN SONGS s ON asp.song_id = s.song_id
    LEFT JOIN COLLECTIONS_SONGS cs ON s.song_id = cs.song_id -- cek
lagu di collection mana
    WHERE asp.playlist_id = p_playlist_id  -- ambil lagu dari playlist
tertentu
    ORDER BY asp.no_urut;  -- Urut sesuai nomor urut di playlist
```

```sql
END;
$$ LANGUAGE plpgsql;

-- cek lihat lagu-lagu di playlist dengan ID = 1
SELECT *
FROM get_playlist_tracks(1);
```

Screenshot Hasil



Tabel 80 PL/SQL - Function get_playlist_detail

Nama Function	get_playlist_detail
Parameter	p_playlist_id INT

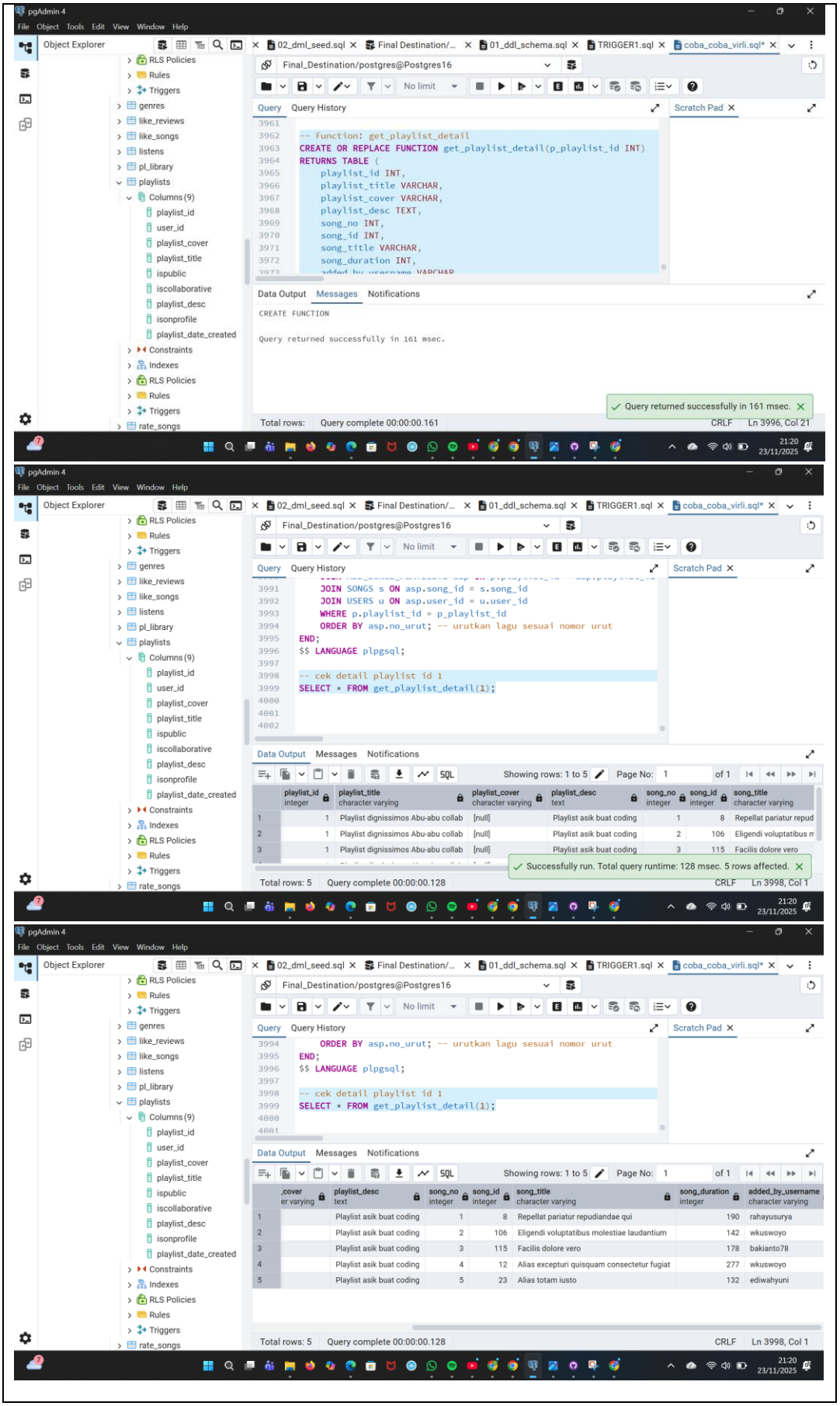
Deskripsi	Function ini mengembalikan detail dari sebuah playlist, termasuk informasi playlist itu sendiri, daftar lagu di dalamnya, nomor urut lagu, durasi lagu, dan username siapa yang menambahkan setiap lagu. Function mengembalikan data dalam bentuk tabel (set of rows).

Script SQL

```
CREATE OR REPLACE FUNCTION get_playlist_detail(p_playlist_id INT)
RETURNS TABLE (
    playlist_id INT,
    playlist_title VARCHAR,
    playlist_cover VARCHAR,
    playlist_desc TEXT,
    song_no INT,
    song_id INT,
    song_title VARCHAR,
    song_duration INT,
    added_by_username VARCHAR
)
AS $$
BEGIN
    -- Ambil data playlist + daftar lagu + siapa yang menambahkan
lagu
    RETURN QUERY
    SELECT
        p.playlist_id,
        p.playlist_title,
        p.playlist_cover,
        p.playlist_desc,
        asp.no_urut AS song_no,  -- nomor urut lagu di playlist
        s.song_id,
        s.song_title,
        s.song_duration,
        u.username AS added_by_username -- username yang
menambahkan lagu
    FROM PLAYLISTS p
    -- Join ke tabel add_songs_playlists untuk mendapatkan daftar
lagu yang ada di playlist
    JOIN ADD_SONGS_PLAYLISTS asp ON p.playlist_id = asp.playlist_id
    JOIN SONGS s ON asp.song_id = s.song_id  -- Join ke tabel songs
untuk mengambil detail lagu
    JOIN USERS u ON asp.user_id = u.user_id  -- Join ke tabel users
untuk mengetahui siapa yang nambahin lagu
```

```
    WHERE p.playlist_id = p_playlist_id  -- Filter hanya playlist
yang sesuai dengan parameter input
    ORDER BY asp.no_urut; -- urutkan lagu sesuai nomor urut
END;
$$ LANGUAGE plpgsql;

-- cek detail playlist id 1
SELECT * FROM get_playlist_detail(1);
```

Screenshot Hasil


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
02_dml_seed.sql
Final Destination/...
01_ddl_schema.sql
TRIGGER1.sql
coba_coba_virli.sql*
Final_Destination/postgres@Postgres16
No limit
Query
Query History
Scratch Pad
-- Function: get_playlist_detail
CREATE OR REPLACE FUNCTION get_playlist_detail(p_playlist_id INT)
RETURNS TABLE (
playlist_id INT,
playlist_title VARCHAR,
playlist_cover VARCHAR,
playlist_desc TEXT,
song_no INT,
song_id INT,
song_title VARCHAR,
song_duration INT,
Data Output
Messages
Notifications
CREATE FUNCTION
Query returned successfully in 161 msec.
Total rows:
Query complete 00:00:00.161
CRLF
Ln 3996, Col 21
JOIN SONGS s ON asp.song_id = s.song_id
JOIN USERS u ON asp.user_id = u.user_id
WHERE p.playlist_id = p_playlist_id
ORDER BY asp.no_urut; -- urutkan lagu sesuai nomor urut
END;
$$ LANGUAGE plpgsql;
-- cek detail playlist id 1
SELECT * FROM get_playlist_detail(1);
Showing rows: 1 to 5
Page No: 1
of 1
Successfully run. Total query runtime: 128 msec. 5 rows affected.
Total rows: 5
Query complete 00:00:00.128
Ln 3998, Col 1
RLS Policies
Rules
Triggers
genres
like_reviews
like_songs
listens
pl_library
playlists
Columns (9)
playlist_id
user_id
playlist_cover
playlist_title
ispublic
iscollaborative
playlist_desc
isonprofile
playlist_date_created
Constraints
Indexes
rate_songs
21:20
23/11/2025

Tabel 81 PL/SQL – Procedure log_listen

Nama Procedure	Log_listen
Parameter	p_user_id – ID user yang mendengarkan lagu p_song_id – ID lagu yang didengarkan p_duration – Durasi dalam detik user mendengarkan lagu
Deskripsi	log_listen procedure ini yang akan digunakan untuk mencatat histori mendengarkan lagu ke tabel LISTENS. Fungsionalitasnya adalah: 1. Validasi bahwa user dan lagu benar-benar ada di tabel USERS dan SONGS. 2. Validasi durasi harus lebih dari 0. 3. Mengecek apakah user sudah pernah mendengarkan lagu tersebut. 1. Jika belum pernah, procedure melakukan: - INSERT baris baru ke tabel LISTENS - Menambahkan listen_count + 1 pada tabel SONGS 2. Jika sudah pernah, procedure hanya: - UPDATE duration terbaru - Mengupdate timestamp - Tidak menambah listen_count lagi Dengan demikian penghitungan listen_count hanya dilakukan sekali

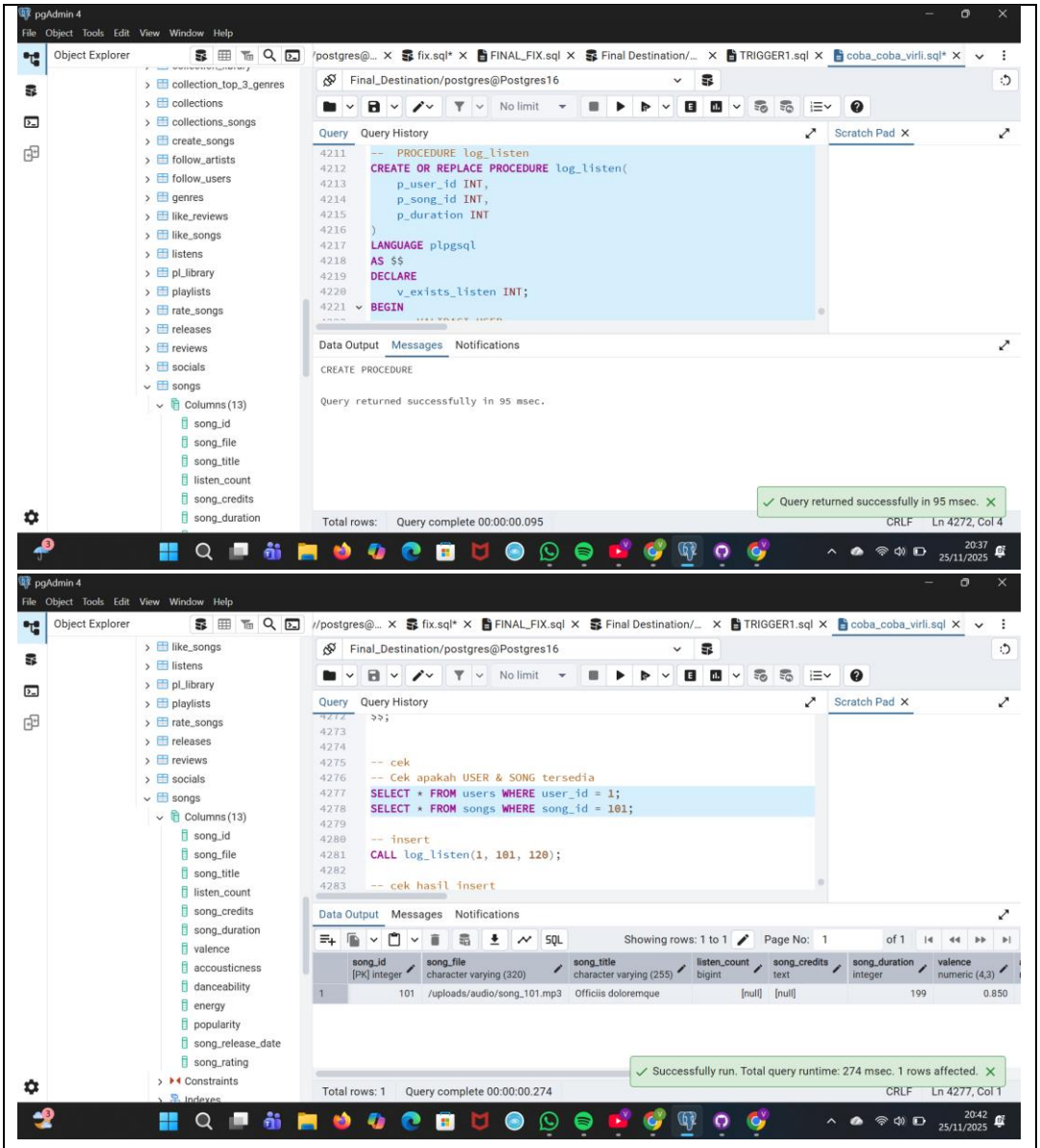
	untuk setiap pasangan user–song, dan histori durasi terakhir tetap diperbarui pada pemutaran ulang.

Script SQL

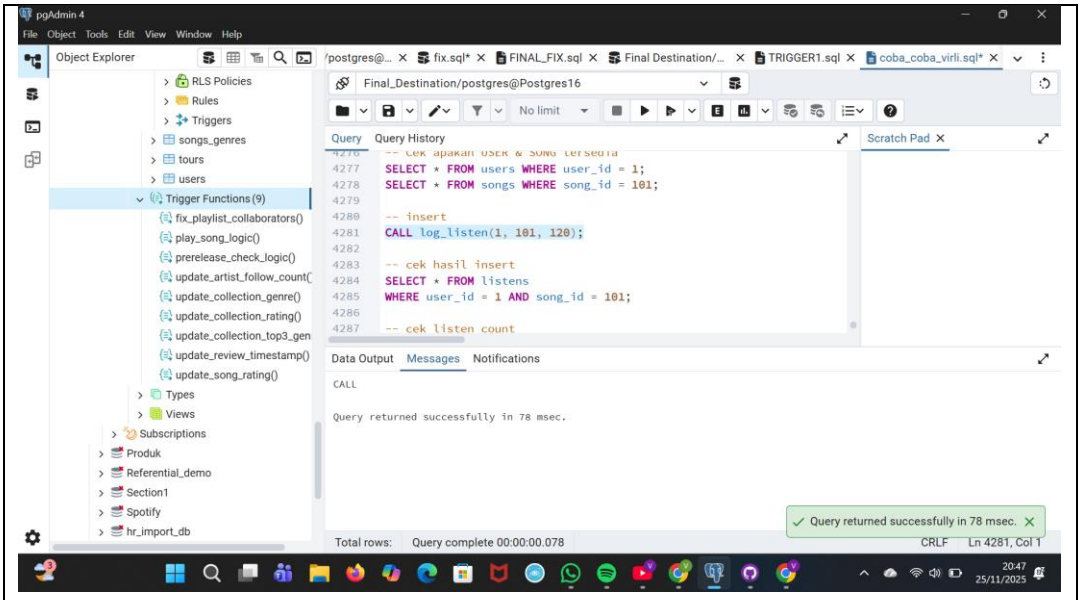
```
CREATE OR REPLACE PROCEDURE log_listen(
    p_user_id INT,  -- Parameter input: ID user yang mendengarkan
lagu, ID lagu yang didengarkan, durasi lagu yang didengarkan (dalam
detik)
    p_song_id INT,
    p_duration INT
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_exists_listen INT;  -- Variabel untuk menyimpan jumlah record
listens yang sudah ada untuk user & song
BEGIN
    -- VALIDASI USER
    IF NOT EXISTS (SELECT 1 FROM users WHERE user_id = p_user_id)
THEN
        RAISE EXCEPTION 'User % tidak ditemukan', p_user_id;
    END IF;

    -- VALIDASI SONG
    IF NOT EXISTS (SELECT 1 FROM songs WHERE song_id = p_song_id)
THEN
        RAISE EXCEPTION 'Song % tidak ditemukan', p_song_id;
    END IF;

    -- VALIDASI DURASI
    IF p_duration <= 0 THEN
        RAISE EXCEPTION 'Duration harus > 0';
    END IF;

    -- CEK APAKAH USER SUDAH PERNAH MENDENGARKAN LAGU INI
    SELECT COUNT(*)  -- Hitung jumlah record listens untuk user &
song
    INTO v_exists_listen   -- Simpan hasil hitungan ke variabel
v_exists_listen
    FROM listens
    WHERE user_id = p_user_id
      AND song_id = p_song_id;

    -- INSERT LISTEN BARU  → listen_count + 1
```

```
    IF v_exists_listen = 0 THEN  -- Jika user belum pernah
mendengarkan lagu ini

        INSERT INTO listens (listen_id, user_id, song_id,
duration_listened)
        VALUES (
            (SELECT COALESCE(MAX(listen_id), 0) + 1 FROM listens),
-- akan generate listen_id baru secara increment
            p_user_id,
            p_song_id,
            p_duration
        );

        -- TAMBAH COUNTER LISTEN HANYA UNTUK INSERT
        UPDATE songs
        SET listen_count = COALESCE(listen_count, 0) + 1   --
Increment listen_count lagu
        WHERE song_id = p_song_id;

    -- UPDATE LISTEN LAMA → TIDAK MENAMBAH COUNTER
    ELSE

        UPDATE listens
        SET duration_listened = p_duration,  -- Update durasi
terakhir mendengarkan
            "TIMESTAMP" = CURRENT_TIMESTAMP  -- Update timestamp
terakhir
        WHERE user_id = p_user_id
          AND song_id = p_song_id;

    END IF;

END;
$$;
```

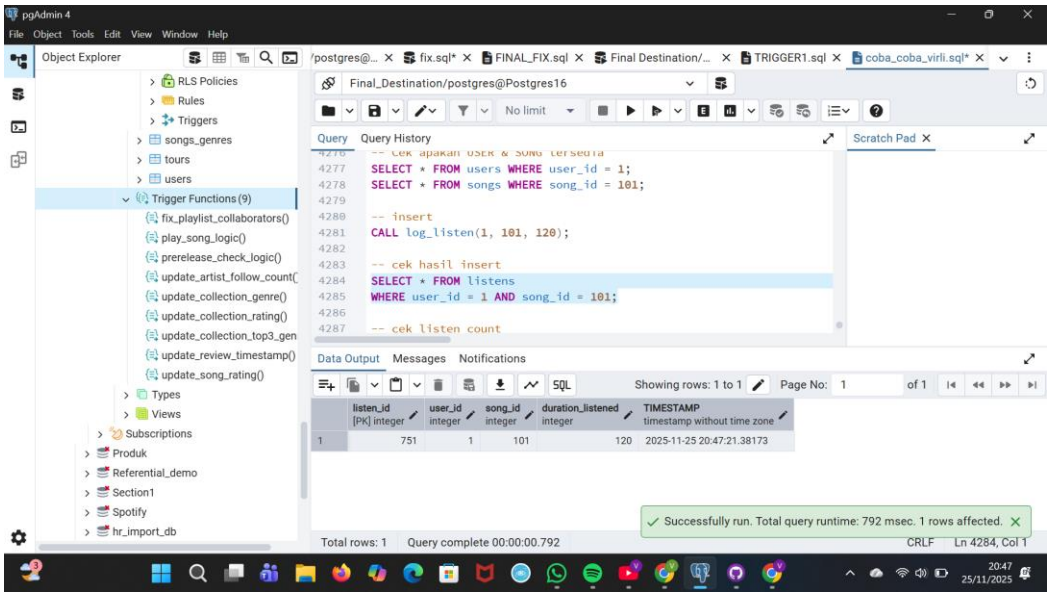
Screenshot Hasil


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Final_Destination/postgres@Postgres16
Query Query History
Scratch Pad
-- PROCEDURE log_listen
CREATE OR REPLACE PROCEDURE log_listen(
p_user_id INT,
p_song_id INT,
p_duration INT
)
LANGUAGE plpgsql
AS $$
DECLARE
v_exists_listen INT;
BEGIN
Data Output Messages Notifications
CREATE PROCEDURE
Query returned successfully in 95 msec.
Total rows: Query complete 00:00:00.095
Query returned successfully in 95 msec.
-- cek
-- Cek apakah USER & SONG tersedia
SELECT * FROM users WHERE user_id = 1;
SELECT * FROM songs WHERE song_id = 101;
-- insert
CALL log_listen(1, 101, 120);
-- cek hasil insert
song_id [PK] integer
song_file character varying (320)
song_title character varying (255)
listen_count bigint
song_credits text
song_duration integer
valence numeric (4,3)
101
/uploads/audio/song_101.mp3
Officiis doloremque
[null]
[null]
199
0.850
Successfully run. Total query runtime: 274 msec. 1 rows affected.
Total rows: 1 Query complete 00:00:00.274

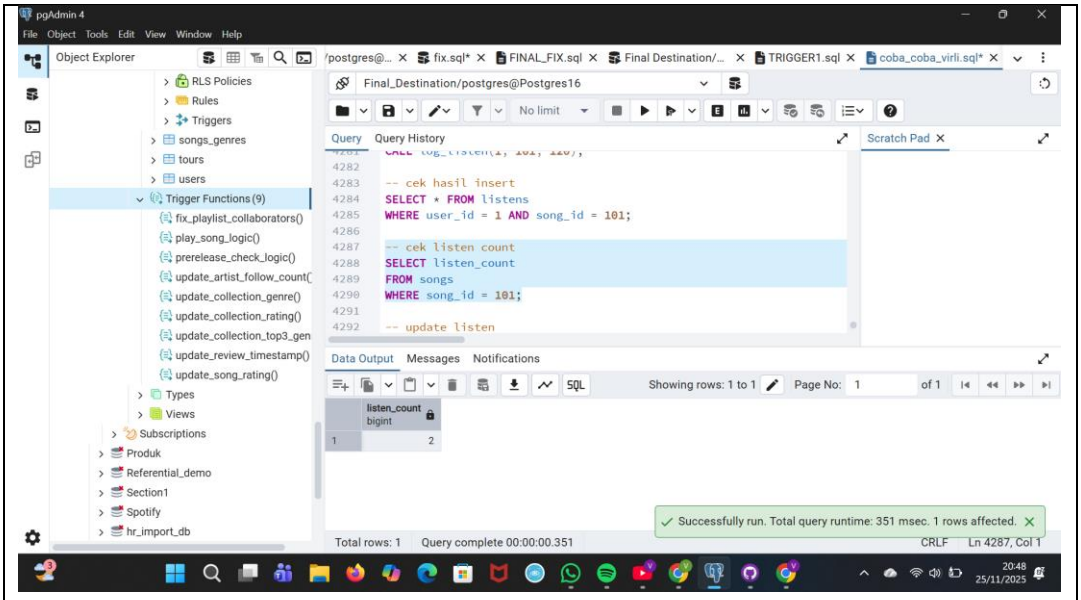
-- insert



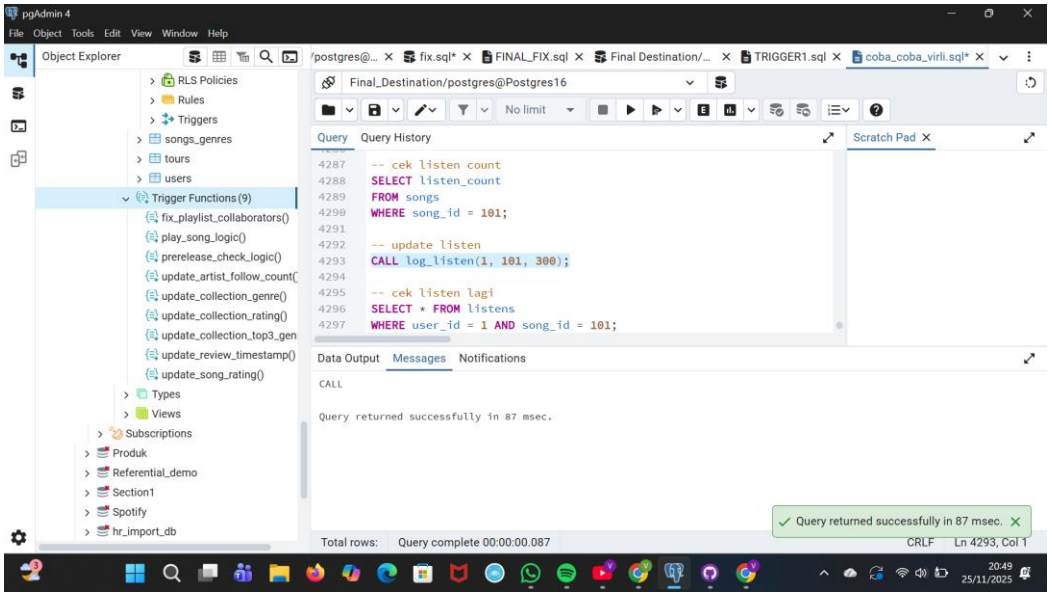
-- cek hasil insert



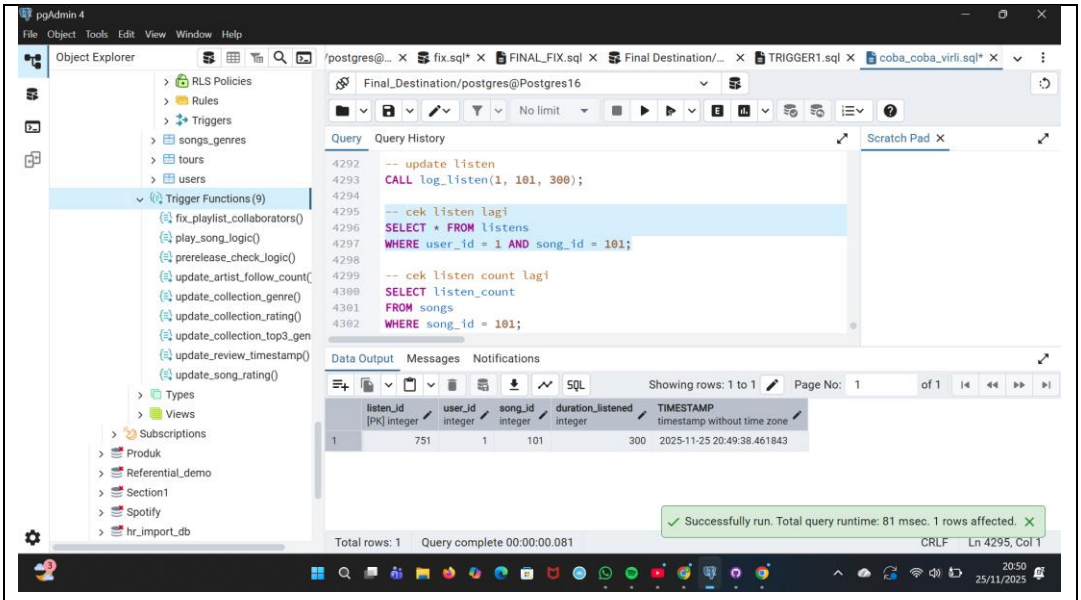
-- cek listen count



-- update listen



--



--

Tabel 82 PL/SQL – Procedure rate_song

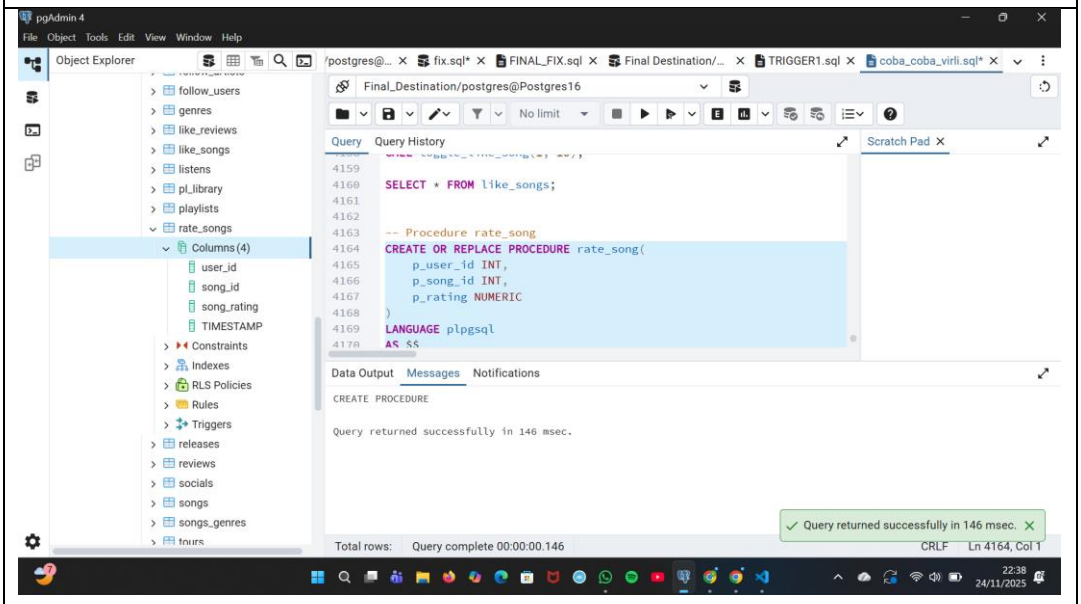
Nama Procedure	rate_song
Parameter	p_user_id INT, p_song_id INT, p_rating NUMERIC
Deskripsi	Procedure rate_song digunakan untuk memberikan atau memperbarui rating pada sebuah lagu oleh user. Logika utamanya: Mengecek apakah user sudah pernah melakukan rating pada lagu tersebut. Jika belum, maka dilakukan INSERT rating baru ke tabel rate_songs. Jika sudah, maka dilakukan UPDATE rating beserta timestamp-nya.

Script SQL

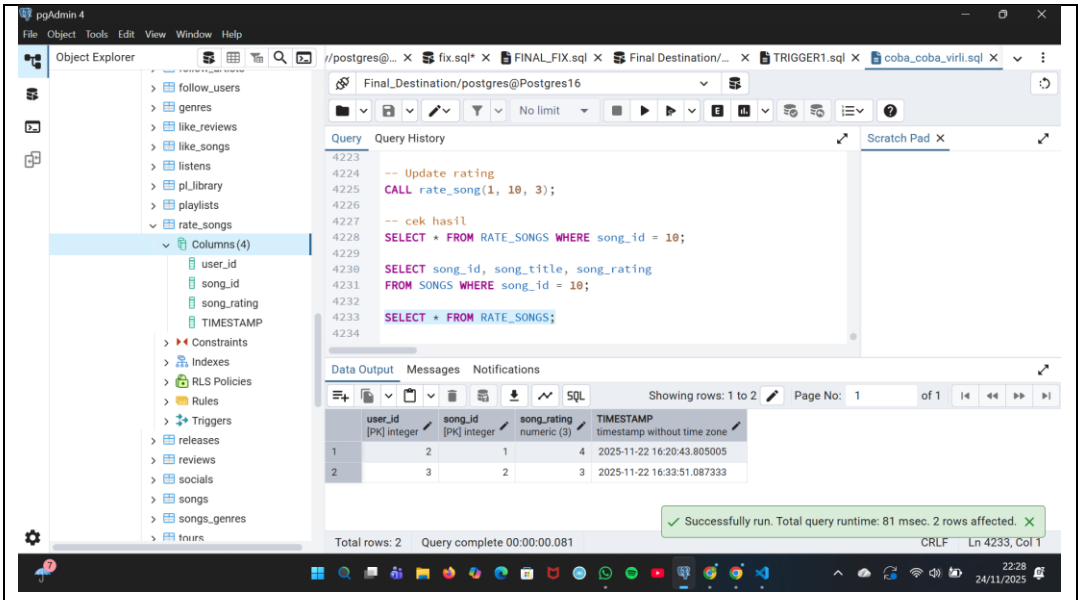
```
CREATE OR REPLACE PROCEDURE rate_song(
    p_user_id INT,  -- Parameter input: ID user yang memberi rating
```

```
    p_song_id INT,  -- Parameter input: ID lagu yang dirating
    p_rating NUMERIC  -- Parameter input: nilai rating lagu
)
LANGUAGE plpgsql
AS $$
BEGIN
    -- Cek apakah user sudah pernah memberi rating
    IF EXISTS (
        SELECT 1 FROM rate_songs
        WHERE user_id = p_user_id
          AND song_id = p_song_id
    ) THEN
        -- Kalo sudah ada rating → update rating lama dan timestamp
        UPDATE rate_songs
        SET song_rating = p_rating,  -- Set rating baru
            "TIMESTAMP" = CURRENT_TIMESTAMP  -- Update waktu rating
terakhir
        WHERE user_id = p_user_id
          AND song_id = p_song_id;  -- update record user & lagu
    ELSE
        -- Jika belum ada rating → insert rating baru
        INSERT INTO rate_songs (user_id, song_id, song_rating)
        VALUES (p_user_id, p_song_id, p_rating);   -- Masukkan
record baru
    END IF;
END;
$$;
```

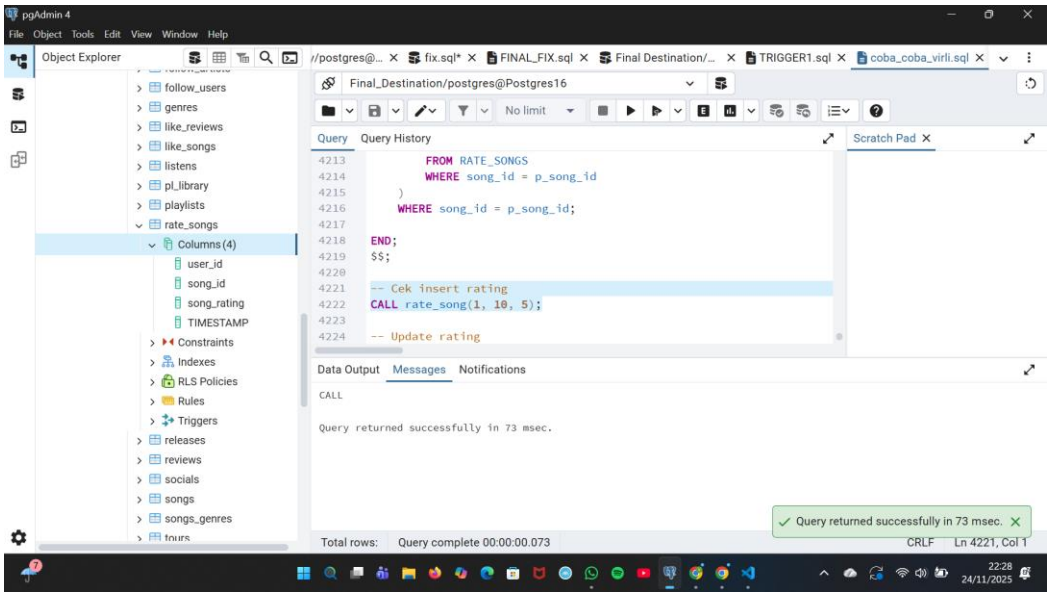
Screenshot Hasil

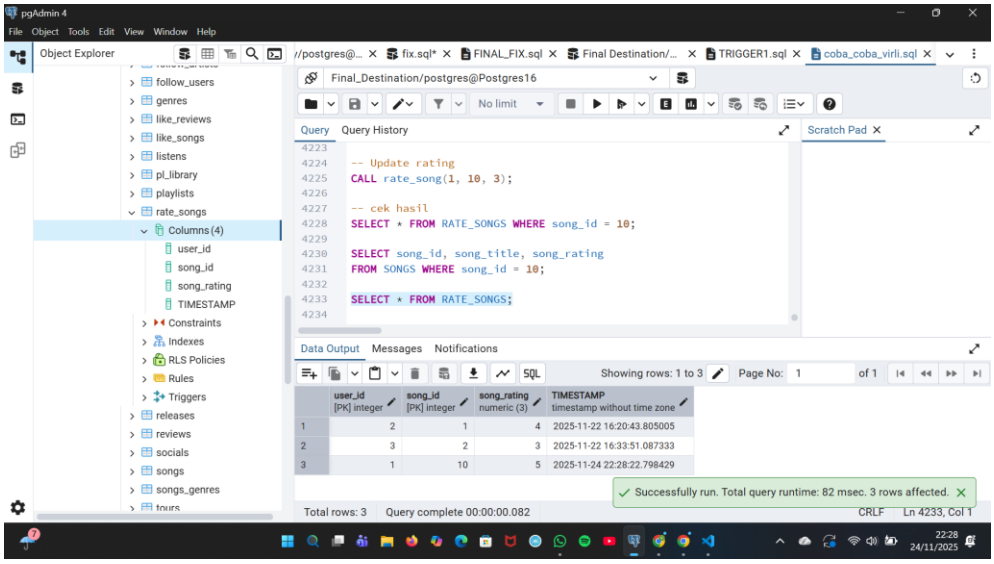


-- sebelum cek

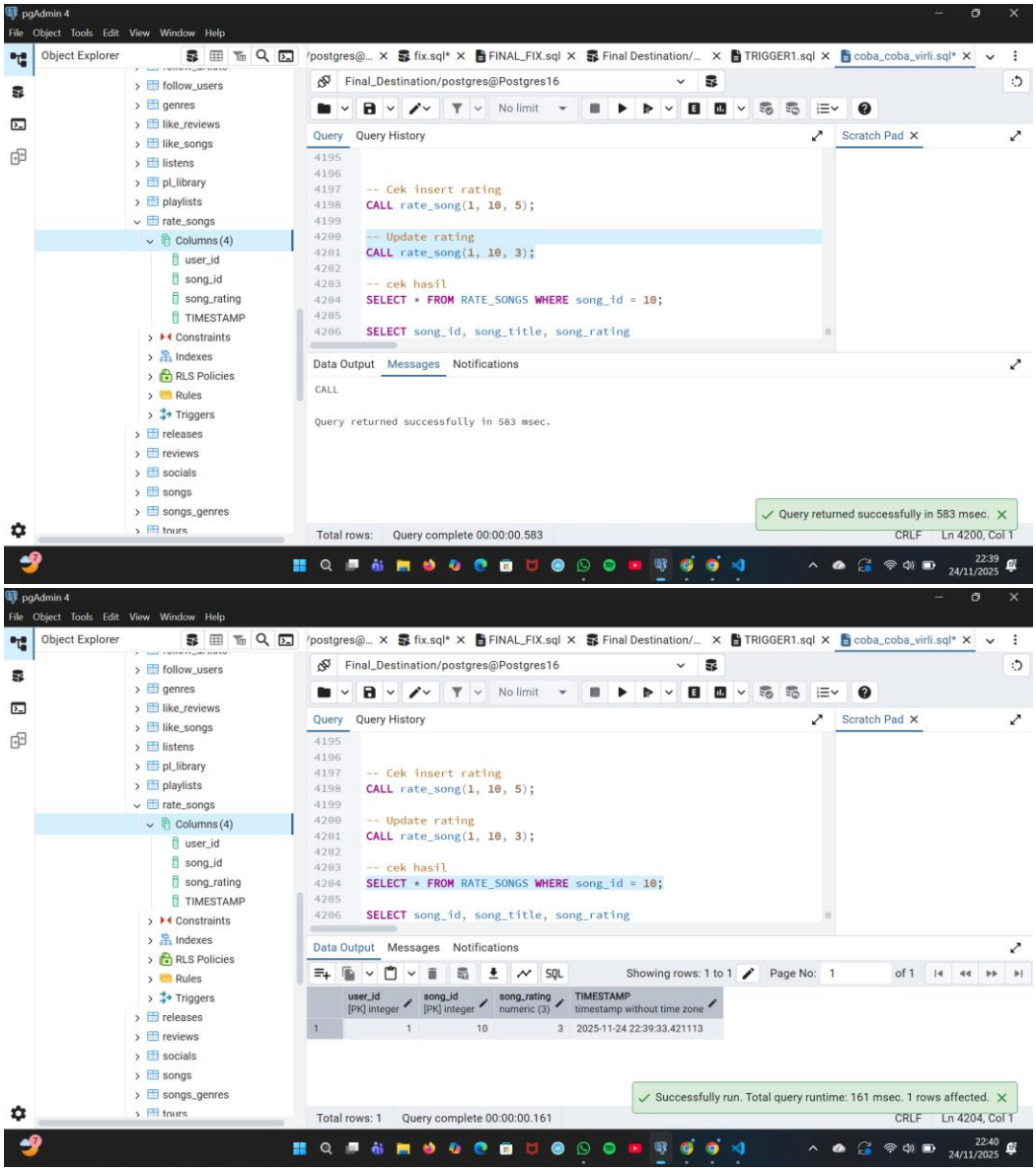

pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Final_Destination/postgres@Postgres16
Query Query History
Scratch Pad
-- Update rating
CALL rate_song(1, 10, 3);
-- cek hasil
SELECT * FROM RATE_SONGS WHERE song_id = 10;
SELECT song_id, song_title, song_rating
FROM SONGS WHERE song_id = 10;
SELECT * FROM RATE_SONGS;
Data Output Messages Notifications
Showing rows: 1 to 2
Page No: 1 of 1
user_id [PK] integer
song_id [PK] integer
song_rating numeric (3)
TIMESTAMP timestamp without time zone
1 2 1 4 2025-11-22 16:20:43.805005
2 3 2 3 2025-11-22 16:33:51.087333
Successfully run. Total query runtime: 81 msec. 2 rows affected.
Total rows: 2
Query complete 00:00:00.081
CRLF
Ln 4233, Col 1

-- cek


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Final_Destination/postgres@Postgres16
Query Query History
Scratch Pad
FROM RATE_SONGS
WHERE song_id = p_song_id
)
WHERE song_id = p_song_id;
END;
$$;
-- Cek insert rating
CALL rate_song(1, 10, 5);
-- Update rating
Data Output Messages Notifications
CALL
Query returned successfully in 73 msec.
Query returned successfully in 73 msec.
Total rows:
Query complete 00:00:00.073
CRLF
Ln 4221, Col 1


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Final_Destination/postgres@Postgres16
Query Query History
Scratch Pad
-- Update rating
CALL rate_song(1, 10, 3);
-- cek hasil
SELECT * FROM RATE_SONGS WHERE song_id = 10;
SELECT song_id, song_title, song_rating
FROM SONGS WHERE song_id = 10;
SELECT * FROM RATE_SONGS;
Data Output Messages Notifications
Showing rows: 1 to 3
Page No: 1 of 1
user_id [PK] integer
song_id [PK] integer
song_rating numeric (3)
TIMESTAMP timestamp without time zone
1 2 1 4 2025-11-22 16:20:43.805005
2 3 2 3 2025-11-22 16:33:51.087333
3 1 10 5 2025-11-24 22:28:22.798429
Successfully run. Total query runtime: 82 msec. 3 rows affected.
Total rows: 3
Query complete 00:00:00.082
CRLF
Ln 4233, Col 1

-- update rating



Tabel 83 PL/SQL – Procedure toggle_like_song

Nama Procedure	Toggle_like_song
Parameter	p_user_id INT p_user_id INT
Deskripsi	Procedure toggle_like_song digunakan untuk melakukan fitur LIKE/UNLIKE lagu oleh seorang user dalam satu perintah.

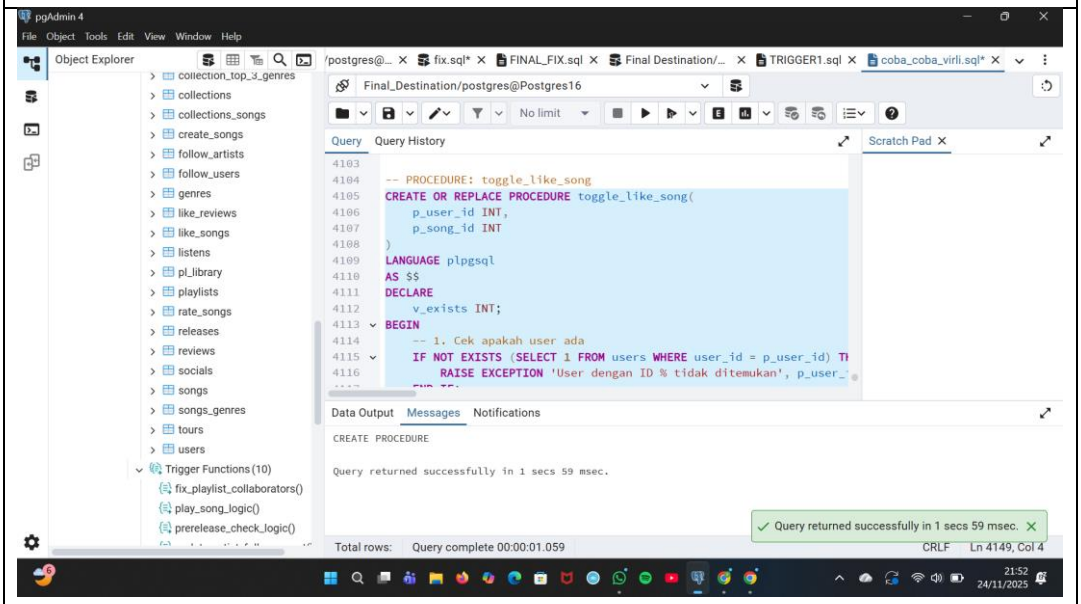
	Jika user belum like lagu → sistem akan INSERT ke tabel LIKE_SONGS Jika user sudah like lagu → sistem akan DELETE dari tabel LIKE_SONGS Procedure ini juga melakukan validasi apakah user dan song benar-benar ada pada tabel USERS dan SONGS.

Script SQL

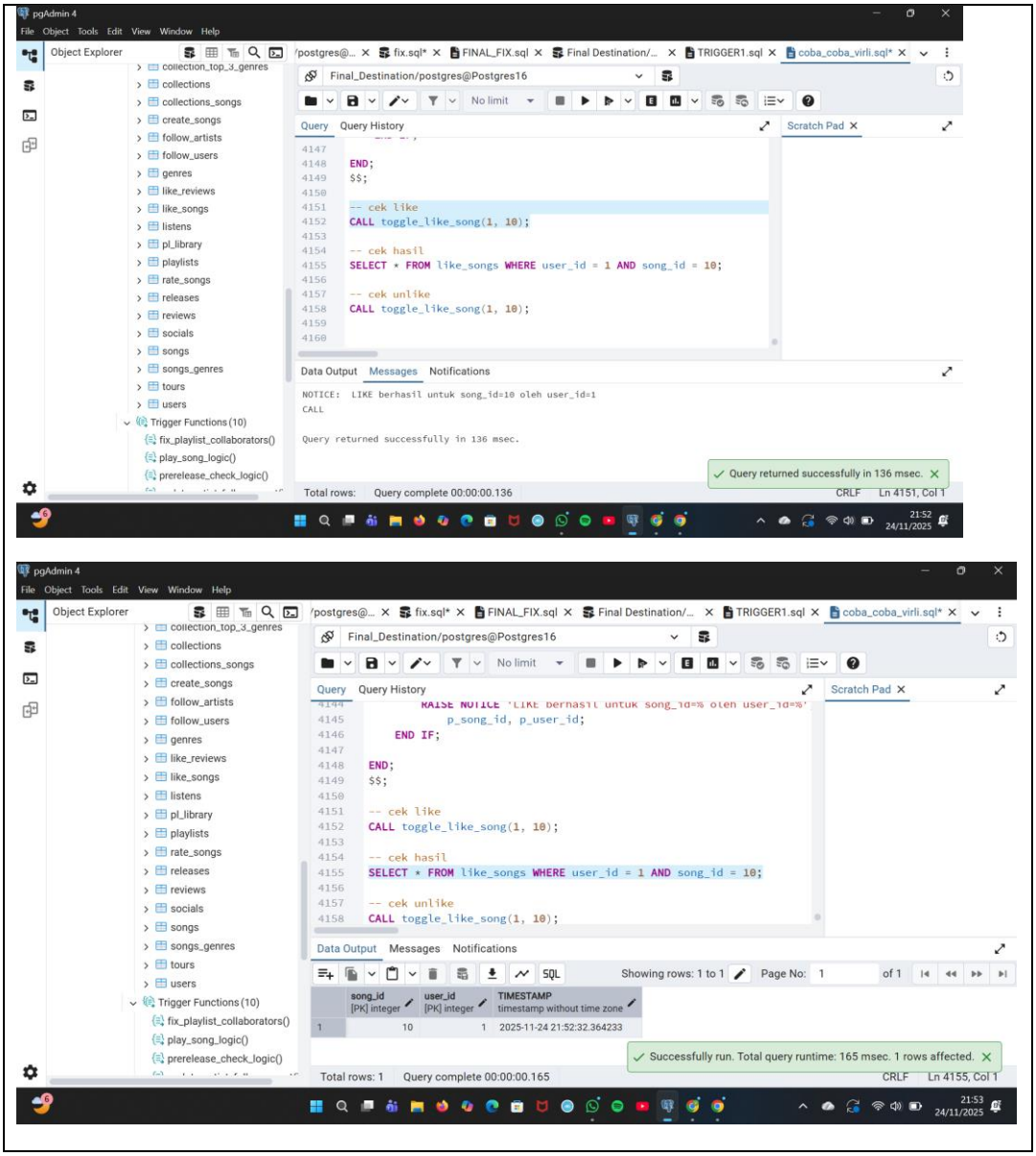
```
CREATE OR REPLACE PROCEDURE toggle_like_song(
    p_user_id INT, -- Parameter input: ID user yang ingin
like/unlike lagu
    p_song_id INT  -- Parameter input: ID lagu yang ingin di
like/unlike
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_exists INT;  -- Variabel untuk mengecek record like_songs
sudah ada apa belum
BEGIN
    -- 1. Cek apakah user ada
    IF NOT EXISTS (SELECT 1 FROM users WHERE user_id = p_user_id)
THEN
        RAISE EXCEPTION 'User dengan ID % tidak ditemukan',
p_user_id;
    END IF;

    -- 2. Cek apakah lagu ada
    IF NOT EXISTS (SELECT 1 FROM songs WHERE song_id = p_song_id)
THEN
        RAISE EXCEPTION 'Song dengan ID % tidak ditemukan',
p_song_id;
    END IF;

    -- 3. Cek apakah sudah like
    SELECT COUNT(*) INTO v_exists
    FROM like_songs
    WHERE user_id = p_user_id
      AND song_id = p_song_id;
    -- 4. Jika sudah LIKE → UNLIKE
    IF v_exists > 0 THEN
        DELETE FROM like_songs  -- Hapus record like (unlike)
        WHERE user_id = p_user_id
```

```
          AND song_id = p_song_id;
        RAISE NOTICE 'UNLIKE berhasil untuk song_id=% oleh
user_id=%',
            p_song_id, p_user_id;
    -- 5. Jika belum LIKE → LIKE
    ELSE
        INSERT INTO like_songs (song_id, user_id)  -- Tambah record
like baru
        VALUES (p_song_id, p_user_id);
        RAISE NOTICE 'LIKE berhasil untuk song_id=% oleh
user_id=%',
            p_song_id, p_user_id;
    END IF;
END;
$$;

-- cek like
CALL toggle_like_song(1,10);
```

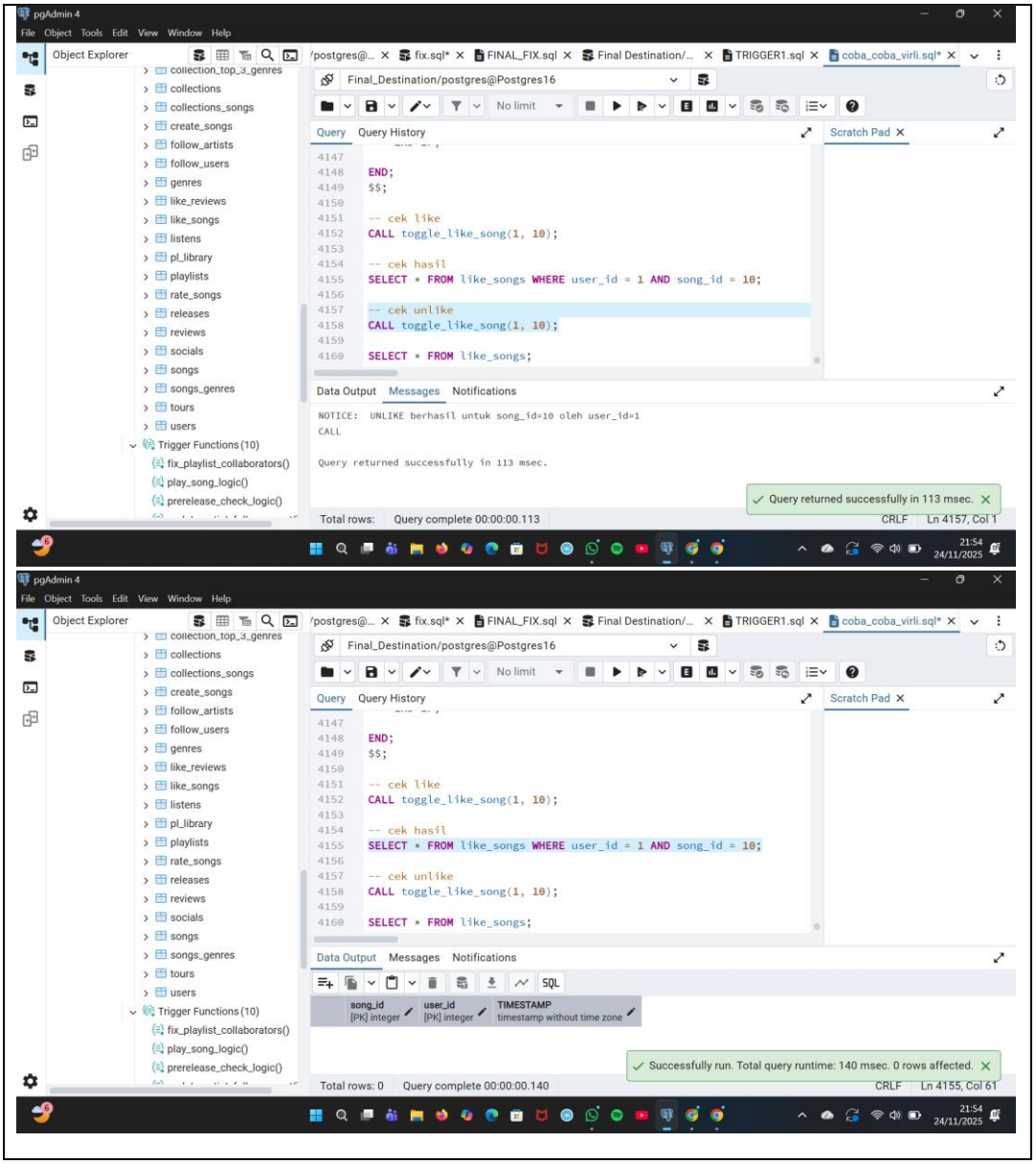
Screenshot Hasil



-- cek like


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Final_Destination/postgres@Postgres16
Query Query History
Scratch Pad
END;
$$;
-- cek like
CALL toggle_like_song(1, 10);
-- cek hasil
SELECT * FROM like_songs WHERE user_id = 1 AND song_id = 10;
-- cek unlike
CALL toggle_like_song(1, 10);
Data Output Messages Notifications
NOTICE: LIKE berhasil untuk song_id=10 oleh user_id=1
CALL
Query returned successfully in 136 msec.
Query returned successfully in 136 msec.
Total rows: Query complete 00:00:00.136
CRLF Ln 4151, Col 1

pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Final_Destination/postgres@Postgres16
Query Query History
Scratch Pad
p_song_id, p_user_id;
END IF;
END;
$$;
-- cek like
CALL toggle_like_song(1, 10);
-- cek hasil
SELECT * FROM like_songs WHERE user_id = 1 AND song_id = 10;
-- cek unlike
CALL toggle_like_song(1, 10);
Data Output Messages Notifications
Showing rows: 1 to 1 Page No: 1 of 1
song_id [PK] integer
user_id [PK] integer
TIMESTAMP timestamp without time zone
1 10 1 2025-11-24 21:52:32.364233
Successfully run. Total query runtime: 165 msec. 1 rows affected.
Total rows: 1 Query complete 00:00:00.165
CRLF Ln 4155, Col 1



Tabel 84 PL/SQL – Procedure remove_song_from_playlist

Nama Procedure	Remove_song_from_playlist
Parameter	p_playlist_id (INT) --> ID playlist tempat lagu akan dihapus p_song_id (INT) --> ID lagu yang ingin dihapus dari playlist
Deskripsi	Procedure remove_song_from_playlist ini prosedur yang digunakan untuk menghapus lagu tertentu dari sebuah playlist sekaligus

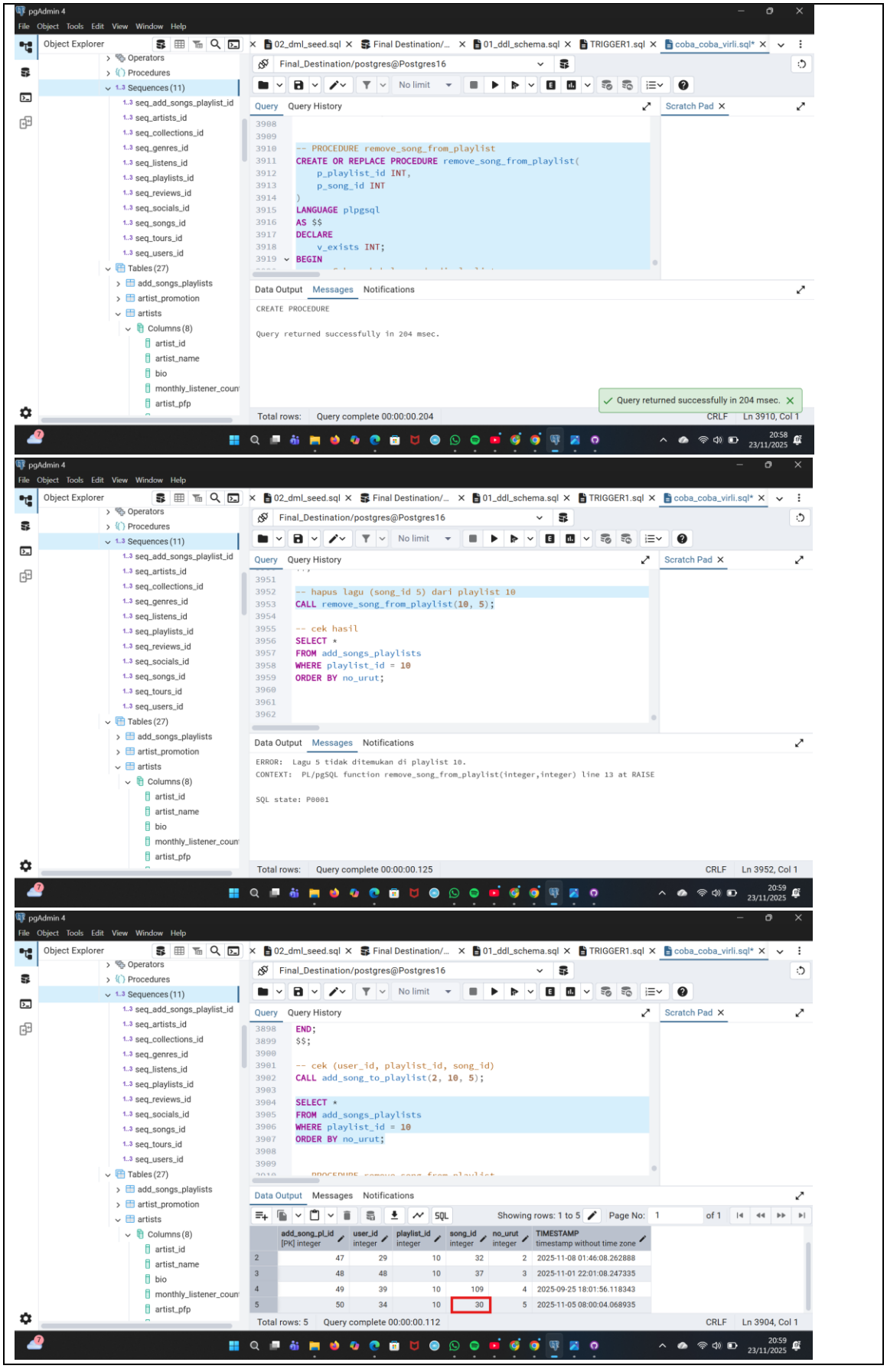
	memastikan struktur playlist tetap rapi dan valid. Prosedur ini terlebih dahulu memeriksa apakah lagu yang akan dihapus benar-benar ada pada playlist; jika tidak ditemukan, sistem akan memberikan error menggunakan RAISE EXCEPTION. Jika lagu ditemukan, prosedur akan menghapusnya dari tabel add_songs_playlists, lalu melakukan proses penataan ulang (re-ordering) nomor urut lagu agar tidak ada nomor yang hilang atau loncat setelah penghapusan—misalnya dari 1,2,4 menjadi 1,2,3. Dengan demikian, prosedur ini menjaga konsistensi data dan memastikan playlist tetap terstruktur secara benar.

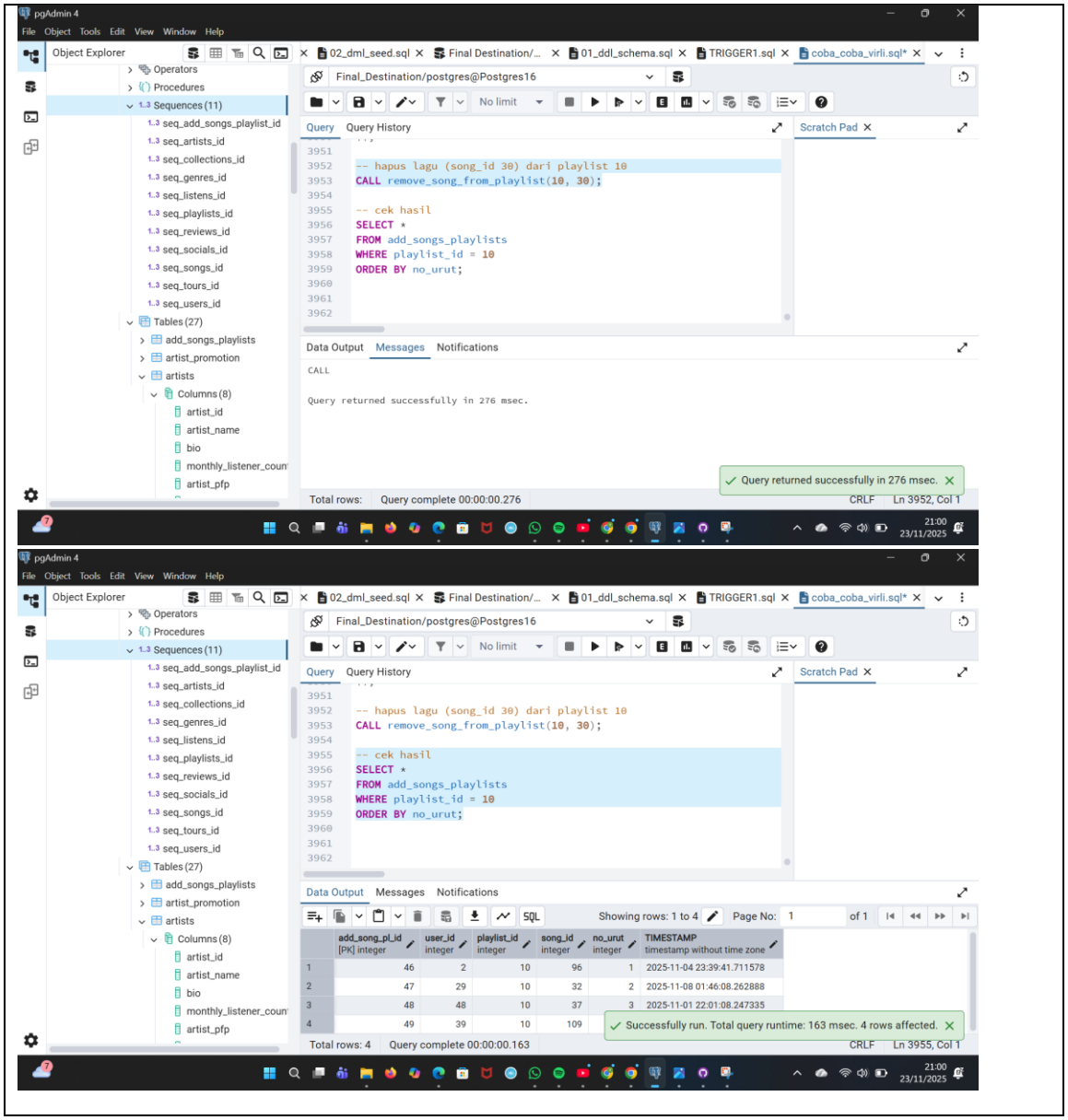
Script SQL

```
CREATE OR REPLACE PROCEDURE remove_song_from_playlist(
    p_playlist_id INT,  -- Parameter input: ID playlist yang ingin
dihapus lagunya
    p_song_id INT  -- Parameter input: ID lagu yang ingin dihapus
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_exists INT;  -- Variabel untuk mengecek apakah lagu ada di
playlist
BEGIN
    -- Cek apakah lagu ada di playlist
    SELECT COUNT(*)
    INTO v_exists
    FROM add_songs_playlists
    WHERE playlist_id = p_playlist_id
      AND song_id = p_song_id;

    IF v_exists = 0 THEN
        -- Kalo lagu tidak ditemukan di playlist,
        RAISE EXCEPTION 'Lagu % tidak ditemukan di playlist %.',
p_song_id, p_playlist_id;
    END IF;

    -- Hapus lagu dari playlist
    DELETE FROM add_songs_playlists
    WHERE playlist_id = p_playlist_id
      AND song_id = p_song_id;
```

```
    -- Perbaiki nomor urut (re-order) supaya urutan lagu tetap berurutan
    WITH ordered AS (
        SELECT
            add_song_pl_id,  -- Ambil ID record
            ROW_NUMBER() OVER (ORDER BY no_urut) AS new_no  -- Hitung
nomor urut baru
        FROM add_songs_playlists
        WHERE playlist_id = p_playlist_id  -- Hanya untuk playlist yang
sama
    )
    UPDATE add_songs_playlists asp
    SET no_urut = o.new_no  -- Update no_urut sesuai urutan baru
    FROM ordered o
    WHERE asp.add_song_pl_id = o.add_song_pl_id;  -- Cocokin dengan ID
record nya

END;
$$;

-- hapus lagu (song_id 30) dari playlist 10
CALL remove_song_from_playlist(10, 30);
```

Screenshot Hasil


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
02_dml_seed.sql
Final Destination/...
01_ddl_schema.sql
TRIGGER1.sql
coba_coba_virli.sql*
Final_Destination/postgres@Postgres16
No limit
Query
Query History
Scratch Pad
Operators
Procedures
Sequences (11)
seq_add_songs_playlist_id
seq_artists_id
seq_collections_id
seq_genres_id
seq_listens_id
seq_playlists_id
seq_reviews_id
seq_socials_id
seq_songs_id
seq_tours_id
seq_users_id
Tables (27)
add_songs_playlists
artist_promotion
artists
Columns (8)
artist_id
artist_name
bio
monthly_listener_count
artist_pfp
3908
3909
3910 -- PROCEDURE remove_song_from_playlist
3911 CREATE OR REPLACE PROCEDURE remove_song_from_playlist(
3912 p_playlist_id INT,
3913 p_song_id INT
3914)
3915 LANGUAGE plpgsql
3916 AS $$
3917 DECLARE
3918 v_exists INT;
3919 BEGIN
Data Output
Messages
Notifications
CREATE PROCEDURE
Query returned successfully in 204 msec.
Total rows:
Query complete 00:00:00.204
CRLF
Ln 3910, Col 1
20:58
23/11/2025
pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
02_dml_seed.sql
Final Destination/...
01_ddl_schema.sql
TRIGGER1.sql
coba_coba_virli.sql*
Final_Destination/postgres@Postgres16
No limit
Query
Query History
Scratch Pad
Operators
Procedures
Sequences (11)
seq_add_songs_playlist_id
seq_artists_id
seq_collections_id
seq_genres_id
seq_listens_id
seq_playlists_id
seq_reviews_id
seq_socials_id
seq_songs_id
seq_tours_id
seq_users_id
Tables (27)
add_songs_playlists
artist_promotion
artists
Columns (8)
artist_id
artist_name
bio
monthly_listener_count
artist_pfp
3951
3952 -- hapus lagu (song_id 5) dari playlist 10
3953 CALL remove_song_from_playlist(10, 5);
3954
3955 -- cek hasil
3956 SELECT *
3957 FROM add_songs_playlists
3958 WHERE playlist_id = 10
3959 ORDER BY no_urut;
3960
3961
3962
Data Output
Messages
Notifications
ERROR: Lagu 5 tidak ditemukan di playlist 10.
CONTEXT: PL/pgSQL function remove_song_from_playlist(integer,integer) line 13 at RAISE
SQL state: P0001
Total rows:
Query complete 00:00:00.125
CRLF
Ln 3952, Col 1
20:59
23/11/2025
pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
02_dml_seed.sql
Final Destination/...
01_ddl_schema.sql
TRIGGER1.sql
coba_coba_virli.sql*
Final_Destination/postgres@Postgres16
No limit
Query
Query History
Scratch Pad
Operators
Procedures
Sequences (11)
seq_add_songs_playlist_id
seq_artists_id
seq_collections_id
seq_genres_id
seq_listens_id
seq_playlists_id
seq_reviews_id
seq_socials_id
seq_songs_id
seq_tours_id
seq_users_id
Tables (27)
add_songs_playlists
artist_promotion
artists
Columns (8)
artist_id
artist_name
bio
monthly_listener_count
artist_pfp
3898 END;
3899 $$;
3900
3901 -- cek (user_id, playlist_id, song_id)
3902 CALL add_song_to_playlist(2, 10, 5);
3903
3904 SELECT *
3905 FROM add_songs_playlists
3906 WHERE playlist_id = 10
3907 ORDER BY no_urut;
3908
3909
Data Output
Messages
Notifications
SQL
Showing rows: 1 to 5
Page No: 1
of 1
add_song_pl_id [PK] integer
user_id integer
playlist_id integer
song_id integer
no_urut integer
TIMESTAMP timestamp without time zone
2 47 29 10 32 2 2025-11-08 01:46:08.262888
3 48 48 10 37 3 2025-11-01 22:01:08.247335
4 49 39 10 109 4 2025-09-25 18:01:56.118343
5 50 34 10 30 5 2025-11-05 08:00:04.068935
Total rows: 5
Query complete 00:00:00.112
CRLF
Ln 3904, Col 1
20:59
23/11/2025



Tabel 85 PL/SQL – Procedure add_song_to_playlist

Nama Procedure	Add_song_to_playlist
Parameter	p_user_id INT --> ID user yang menambahkan p_playlist_id INT --> ID playlist tujuan p_song_id INT --> ID lagu yang ingin ditambahkan
Deskripsi	Procedure ini digunakan untuk menambahkan lagu ke dalam playlist

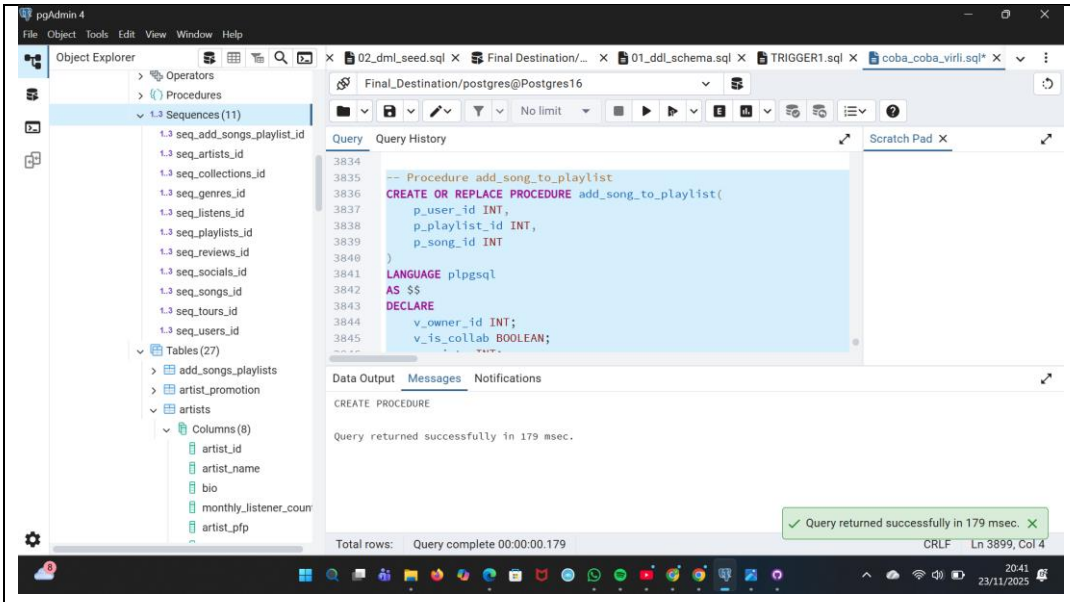
	dengan mekanisme pengecekan dan penomoran otomatis. Fungsi utamanya:
	
	- Memastikan playlist ada
	- Mencegah duplikasi lagu
	- Memastikan hak akses user
	- Menghitung nomor urut lagu otomatis
	  - Mengambil MAX(no_urut)
	  - Lagu baru akan berada pada urutan terakhir (seperti Spotify)
	- Melakukan insert data ke tabel add_songs_playlists

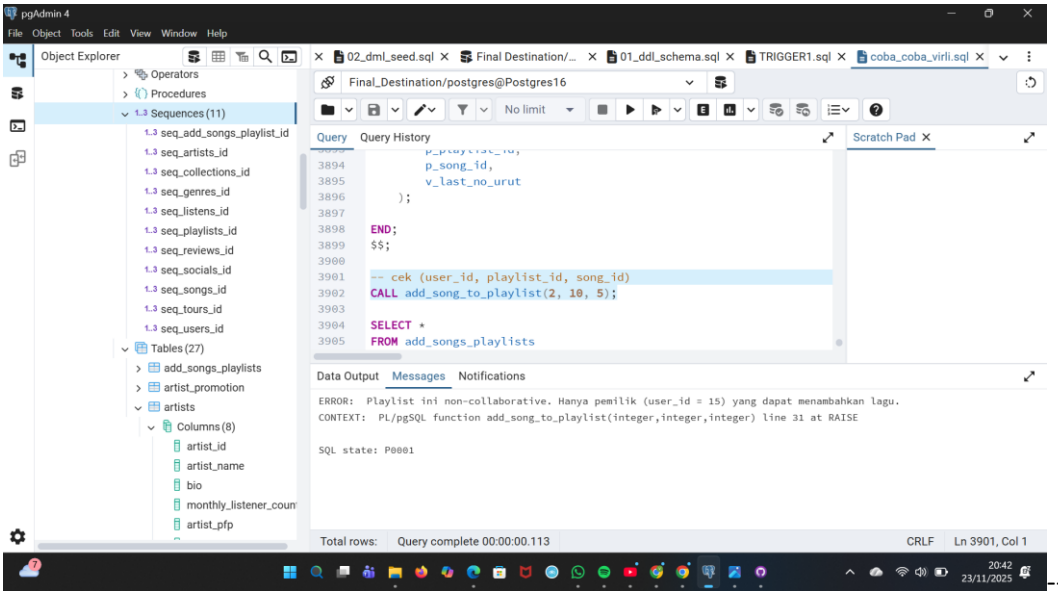
Script SQL

```
CREATE OR REPLACE PROCEDURE add_song_to_playlist(
    p_user_id INT,  -- Parameter input: ID user yang ingin menambahkan
lagu
    p_playlist_id INT,  -- Parameter input: ID playlist tujuan
    p_song_id INT  -- Parameter input: ID lagu yang ingin ditambahkan
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_owner_id INT;  -- Variabel untuk menyimpan ID pemilik playlist
    v_is_collab BOOLEAN;
    v_exists INT;  -- Variabel untuk mengecek apakah lagu sudah ada di
playlist
    v_last_no_urut INT; -- Variabel untuk menentukan nomor urut lagu
terakhir
BEGIN
    -- Cek apakah playlist ada
    SELECT user_id, isCollaborative
    INTO v_owner_id, v_is_collab
    FROM playlists
    WHERE playlist_id = p_playlist_id;

    IF NOT FOUND THEN
    -- Kalo playlist tidak ditemukan,
        RAISE EXCEPTION 'Playlist % tidak ditemukan.', p_playlist_id;
    END IF;

    -- Cek duplikasi lagu
    SELECT COUNT(*)
    INTO v_exists
    FROM add_songs_playlists
```

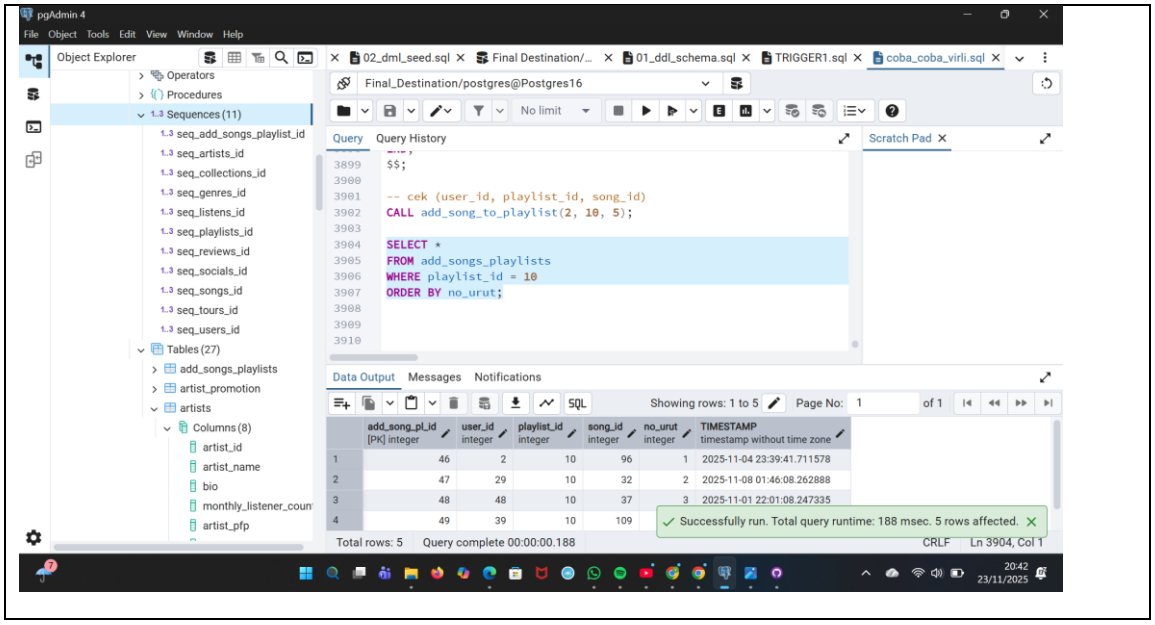
```
    WHERE playlist_id = p_playlist_id
      AND song_id = p_song_id;

    IF v_exists > 0 THEN
    -- Kalo lagu sudah ada di playlist,
        RAISE EXCEPTION 'Lagu % sudah ada di playlist %.', p_song_id,
p_playlist_id;
    END IF;

    -- Cek kepemilikan playlist (non-collaborative)
    IF v_is_collab = FALSE AND p_user_id <> v_owner_id THEN
        RAISE EXCEPTION
            'Playlist ini non-collaborative. Hanya pemilik (user_id = %)
yang dapat menambahkan lagu.',
            v_owner_id;
    END IF;

    -- Hitung nomor urut otomatis
    SELECT COALESCE(MAX(no_urut), 0)
    INTO v_last_no_urut
    FROM add_songs_playlists
    WHERE playlist_id = p_playlist_id;

    v_last_no_urut := v_last_no_urut + 1;  -- Tambah 1 untuk nomor urut
lagu baru

    -- Insert lagu ke playlist
    INSERT INTO add_songs_playlists (
        user_id,
        playlist_id,
        song_id,
        no_urut
    ) VALUES (
        p_user_id,
        p_playlist_id,
        p_song_id,
        v_last_no_urut
    );

END;
$$;

-- cek (user_id, playlist_id, song_id)
CALL add_song_to_playlist(2, 10, 5);
```

Screenshot Hasil


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Final_Destination/postgres@Postgres16
Query Query History
Scratch Pad
-- Procedure add_song_to_playlist
CREATE OR REPLACE PROCEDURE add_song_to_playlist(
p_user_id INT,
p_playlist_id INT,
p_song_id INT
)
LANGUAGE plpgsql
AS $$
DECLARE
v_owner_id INT;
v_is_collab BOOLEAN;
Data Output Messages Notifications
CREATE PROCEDURE
Query returned successfully in 179 msec.
Total rows: Query complete 00:00:00.179
CRLF Ln 3899, Col 4


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Final_Destination/postgres@Postgres16
Query Query History
Scratch Pad
p_song_id,
v_last_no_urut
);
END;
$$;
-- cek (user_id, playlist_id, song_id)
CALL add_song_to_playlist(2, 10, 5);
SELECT *
FROM add_songs_playlists
Data Output Messages Notifications
ERROR: Playlist ini non-collaborative. Hanya pemilik (user_id = 15) yang dapat menambahkan lagu.
CONTEXT: PL/pgSQL function add_song_to_playlist(integer,integer,integer) line 31 at RAISE
SQL state: P0001
Total rows: Query complete 00:00:00.113
CRLF Ln 3901, Col 1

-- cek no urut



Tabel 86 PL/SQL – Procedure create_playlist

Nama Procedure	create_playlist
Parameter	p_user_id (INT) p_title (VARCHAR) p_ispublic (BOOLEAN) p_iscollaborative (BOOLEAN) p_description (TEXT) p_cover (VARCHAR) p_isonprofile (BOOLEAN)
Deskripsi	Procedure create_playlist digunakan untuk menambahkan playlist baru ke dalam tabel playlists. Ketika procedure ini dipanggil, sistem akan memasukkan data playlist baru berdasarkan parameter yang diberikan, seperti user pembuat playlist, judul playlist, status publik atau privat, status kolaboratif,

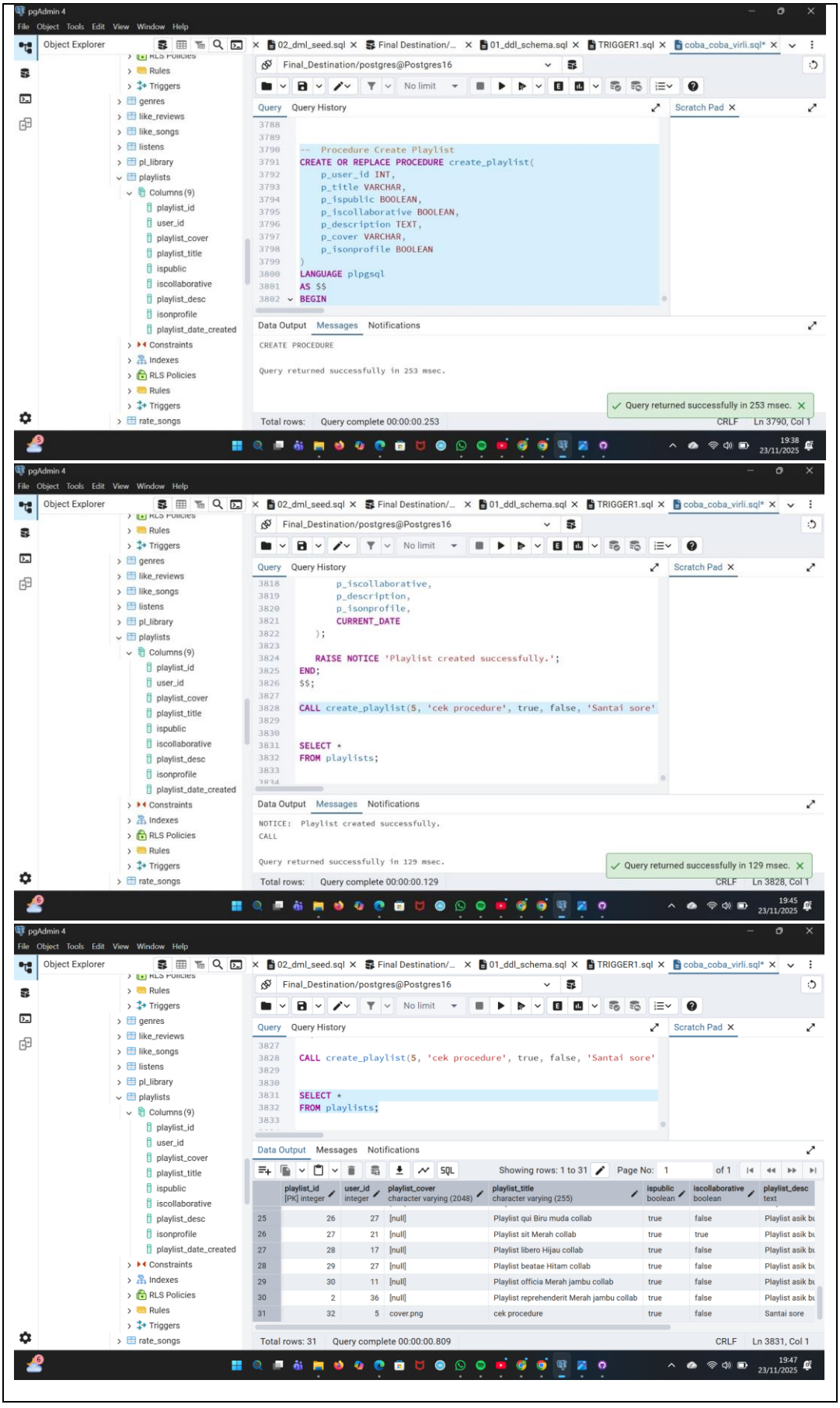
	deskripsi, cover, dan apakah playlist ingin ditampilkan di profil pengguna. Procedure ini juga mencatat tanggal pembuatan playlist secara otomatis menggunakan CURRENT_DATE, serta menampilkan pesan konfirmasi menggunakan RAISE NOTICE jika proses berhasil.

Script SQL

```
CREATE OR REPLACE PROCEDURE create_playlist(
    p_user_id INT,
    p_title VARCHAR,
    p_ispublic BOOLEAN,
    p_iscollaborative BOOLEAN,
    p_description TEXT,
    p_cover VARCHAR,
    p_isonprofile BOOLEAN
)
LANGUAGE plpgsql
AS $$
BEGIN
   INSERT INTO playlists (
       user_id,
       playlist_cover,
       playlist_title,
       ispublic,
       iscollaborative,
       playlist_desc,
       isonprofile,
       playlist_date_created
   )
   VALUES (
       p_user_id,
       p_cover,
       p_title,
       p_ispublic,
       p_iscollaborative,
       p_description,
       p_isonprofile,
       CURRENT_DATE
   );

   RAISE NOTICE 'Playlist created successfully.';
END;
```

```
$$;

-- CALL user_id, playlist_title, ispublic, iscollaborative,
playlist_desc, playlist_cover, isonprofile
CALL create_playlist(5, 'cek procedure', true, false, 'Santai
sore', 'cover.png', true);
```

Screenshot Hasil


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
02_dml_seed.sql
Final Destination/...
01_ddl_schema.sql
TRIGGER1.sql
coba_coba_virli.sql*
Final_Destination/postgres@Postgres16
No limit
Query
Query History
Scratch Pad
-- Procedure Create Playlist
CREATE OR REPLACE PROCEDURE create_playlist(
p_user_id INT,
p_title VARCHAR,
p_ispublic BOOLEAN,
p_iscollaborative BOOLEAN,
p_description TEXT,
p_cover VARCHAR,
p_isonprofile BOOLEAN
)
LANGUAGE plpgsql
AS $$
BEGIN
Data Output
Messages
Notifications
CREATE PROCEDURE
Query returned successfully in 253 msec.
Total rows:
Query complete 00:00:00.253
CRLF
Ln 3790, Col 1
Query returned successfully in 129 msec.
Total rows:
Query complete 00:00:00.129
Ln 3828, Col 1
NOTICE: Playlist created successfully.
CALL
Query returned successfully in 129 msec.
p_iscollaborative,
p_description,
p_isonprofile,
CURRENT_DATE
);
RAISE NOTICE 'Playlist created successfully.';
END;
$$;
CALL create_playlist(5, 'cek procedure', true, false, 'Santai sore');
SELECT *
FROM playlists;
Showing rows: 1 to 31
Page No: 1 of 1
playlist_id [PK] integer
user_id integer
playlist_cover character varying (2048)
playlist_title character varying (255)
ispublic boolean
iscollaborative boolean
playlist_desc text
25 26 27 [null] Playlist qui Biru muda collab true false Playlist asik b
26 27 21 [null] Playlist sit Merah collab true true Playlist asik b
27 28 17 [null] Playlist libero Hijau collab true false Playlist asik b
28 29 27 [null] Playlist beatae Hitam collab true false Playlist asik b
29 30 11 [null] Playlist officia Merah jambu collab true false Playlist asik b
30 2 36 [null] Playlist reprehenderit Merah jambu collab true false Playlist asik b
31 32 5 cover.png cek procedure true false Santai sore
Total rows: 31
Query complete 00:00:00.809
Ln 3831, Col 1
Columns (9)
playlist_id
user_id
playlist_cover
playlist_title
ispublic
iscollaborative
playlist_desc
isonprofile
playlist_date_created
Constraints
Indexes
RLS Policies
Rules
Triggers
genres
like_reviews
like_songs
listens
pl_library
playlists
rate_songs

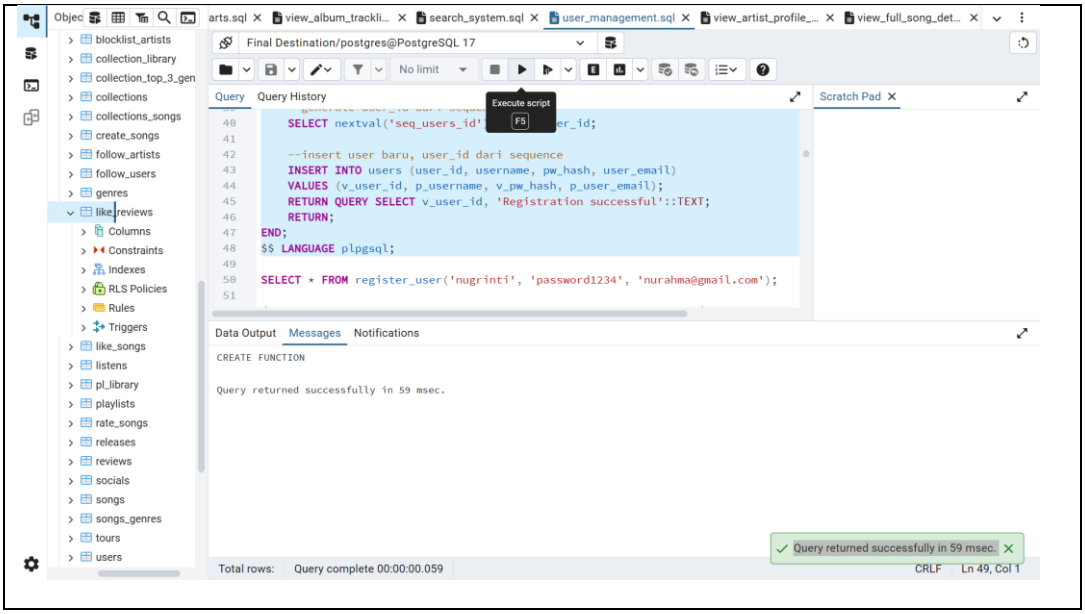
Tabel 87 PL/SQL – Function Register User

Nama Function	register_user
Parameter	p_username VARCHAR, p_raw_pw VARCHAR, p_user_email VARCHAR
Deskripsi	Mendaftarkan user baru dengan memvalidasi email & username, meng-hash password, lalu menghasilkan user_id baru dari sequence.

Script SQL

```
CREATE OR REPLACE FUNCTION register_user(p_username VARCHAR, p_raw_pw VARCHAR,
p_user_email VARCHAR)
RETURNS TABLE (
    user_id INT4,
    message TEXT
)
AS $$
DECLARE
    v_user_id INT4;
    v_pw_hash TEXT;
BEGIN
    --cek apakah email sudah dipakai atau belum
    IF EXISTS (SELECT 1 FROM users WHERE user_email = p_user_email) THEN
        RETURN QUERY SELECT NULL::INT, 'Email already registered'::TEXT;
        RETURN;
    END IF;

    --cek apakah username sudah dipakai atau belum
    IF EXISTS (SELECT 1 FROM users WHERE username = p_username) THEN
        RETURN QUERY SELECT NULL::INT, 'Username already taken'::TEXT;
        RETURN;
    END IF;

    --hash password menggunakan bcrypt
    v_pw_hash := crypt(p_raw_pw, gen_salt('bf'));

    --generate user_id dari sequence
    SELECT nextval('seq_users_id') INTO v_user_id;

    --insert user baru, user_id dari sequence
    INSERT INTO users (user_id, username, pw_hash, user_email)
    VALUES (v_user_id, p_username, v_pw_hash, p_user_email);
    RETURN QUERY SELECT v_user_id, 'Registration successful'::TEXT;
    RETURN;
END;
$$ LANGUAGE plpgsql;
```

Screenshot Hasil



Tabel 88 PL/SQL – Function Login User

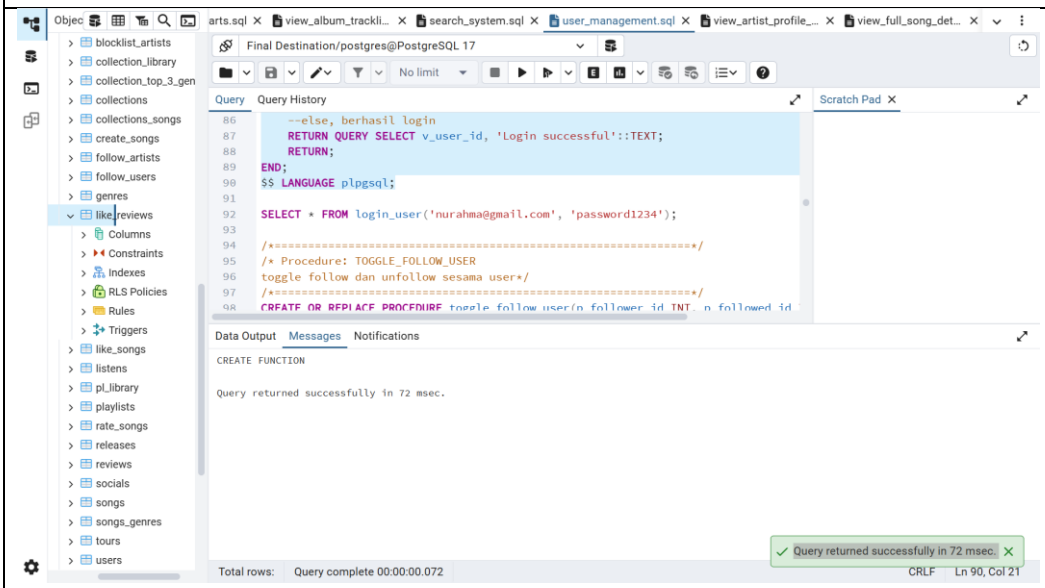
Nama Function	login_user
Parameter	p_user_email VARCHAR, p_raw_pw VARCHAR
Deskripsi	Melakukan verifikasi login user dengan cek email dan mencocokkan password yang di-hash.

Script SQL

```
CREATE OR REPLACE FUNCTION login_user(p_user_email VARCHAR, p_raw_pw VARCHAR)
RETURNS TABLE (
    user_id INT4,
    message TEXT
)
AS $$
DECLARE
    v_user_id INT4;
    v_pw_hash VARCHAR(255);
BEGIN
    --cek apakah username sudah terdaftar atau belum
    SELECT u.user_id, u.pw_hash
    INTO v_user_id, v_pw_hash
    FROM users u
    WHERE u.user_email = p_user_email;

    --jika email belum terdaftar
    IF v_user_id IS NULL THEN
        RETURN QUERY SELECT NULL::INT4, 'Email not registered'::TEXT;
        RETURN;
    END IF;

    --cek password
    IF crypt(p_raw_pw, v_pw_hash) <> v_pw_hash THEN
        RETURN QUERY SELECT NULL::INT4, 'Invalid password'::TEXT;
        RETURN;
    END IF;

    --else, berhasil login
    RETURN QUERY SELECT v_user_id, 'Login successful'::TEXT;
    RETURN;
END;
```

```
$$ LANGUAGE plpgsql;
```

Screenshot Hasil



Tabel 89 PL/SQL – Procedure Toggle Follow User

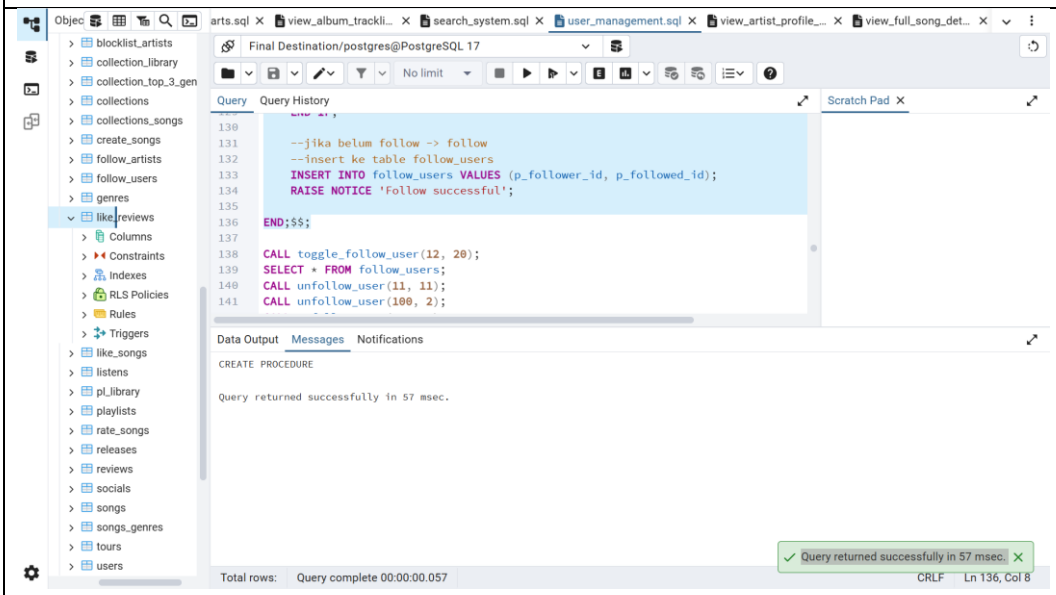
Nama Function	toggle_follow_user
Parameter	p_follower_id INT, p_followed_id INT
Deskripsi	Mengikuti atau berhenti mengikuti user lain secara otomatis berdasarkan apakah relationship follow sudah ada atau belum.

Script SQL

```
CREATE OR REPLACE PROCEDURE toggle_follow_user(p_follower_id INT, p_followed_id INT)
LANGUAGE PLPGSQL
AS $$
DECLARE
    v_exists BOOLEAN;
BEGIN
    --tidak boleh follow/unfollow diri sendiri
    IF p_follower_id = p_followed_id THEN
        RAISE EXCEPTION 'User cannot follow/unfollow themselves';
    END IF;

    --cek apakah kedua user valid
    IF NOT EXISTS (SELECT 1 FROM users WHERE user_id = p_follower_id) THEN
        RAISE EXCEPTION 'Follower user_id % not found', p_follower_id;
    END IF;

    IF NOT EXISTS (SELECT 1 FROM users WHERE user_id = p_followed_id) THEN
        RAISE EXCEPTION 'Followed user_id % not found', p_followed_id;
    END IF;

    --cek apakah sudah follow, jika sudah -> unfollow
    SELECT EXISTS (SELECT 1 FROM follow_users
        WHERE follower_id = p_follower_id AND followed_id = p_followed_id)
    INTO v_exists;

    IF v_exists THEN
```

```
        --delete relationship from follow_users
        DELETE from follow_users
            WHERE follower_id = p_follower_id AND followed_id = p_followed_id;
        RAISE NOTICE 'Unfollow successful';
        RETURN;
    END IF;

    --jika belum follow -> follow
    --insert ke table follow_users
    INSERT INTO follow_users VALUES (p_follower_id, p_followed_id);
    RAISE NOTICE 'Follow successful';

END;$$;
```

Screenshot Hasil



Tabel 90 PL/SQL – Function Get Followers

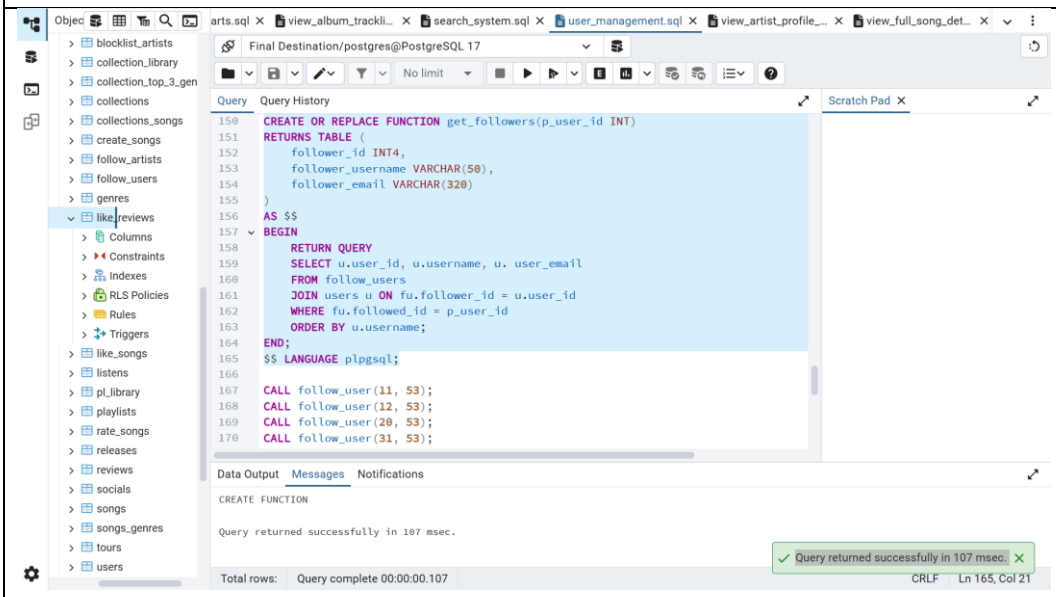
Nama Function	get_followers
Parameter	p_user_id INT
Deskripsi	Mengambil daftar user yang mengikuti user tertentu.

Script SQL

```
CREATE OR REPLACE FUNCTION get_followers(p_user_id INT)
RETURNS TABLE (
    follower_id INT4,
    follower_username VARCHAR(50),
    follower_email VARCHAR(320)
)
AS $$
BEGIN
    RETURN QUERY
    SELECT u.user_id, u.username, u. user_email
    FROM follow_users
    JOIN users u ON fu.follower_id = u.user_id
    WHERE fu.followed_id = p_user_id
    ORDER BY u.username;
END;
```

```
$$ LANGUAGE plpgsql;
```

Screenshot Hasil



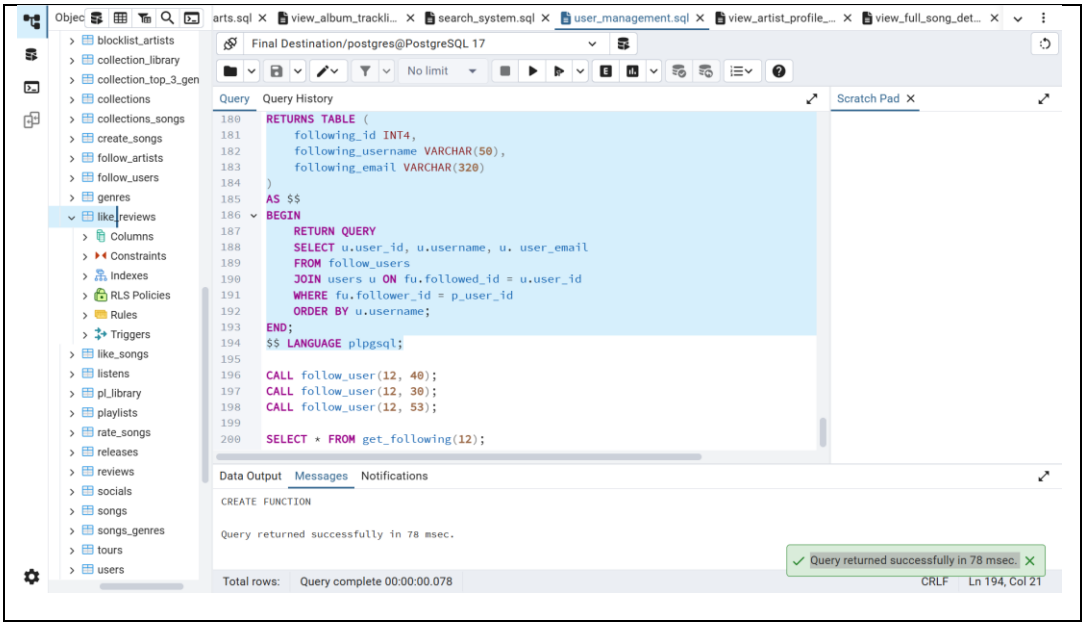
Tabel 91 PL/SQL – Function Get Following

Nama Function	get_following
Parameter	p_user_id INT
Deskripsi	Mengambil daftar user yang diikuti oleh user tertentu.

Script SQL

```
CREATE OR REPLACE FUNCTION get_following(p_user_id INT)
RETURNS TABLE (
    following_id INT4,
    following_username VARCHAR(50),
    following_email VARCHAR(320)
)
AS $$
BEGIN
    RETURN QUERY
    SELECT u.user_id, u.username, u. user_email
    FROM follow_users
    JOIN users u ON fu.followed_id = u.user_id
    WHERE fu.follower_id = p_user_id
    ORDER BY u.username;
END;
$$ LANGUAGE plpgsql;
```

Screenshot Hasil



Tabel 92 PL/SQL – Procedure Create Review

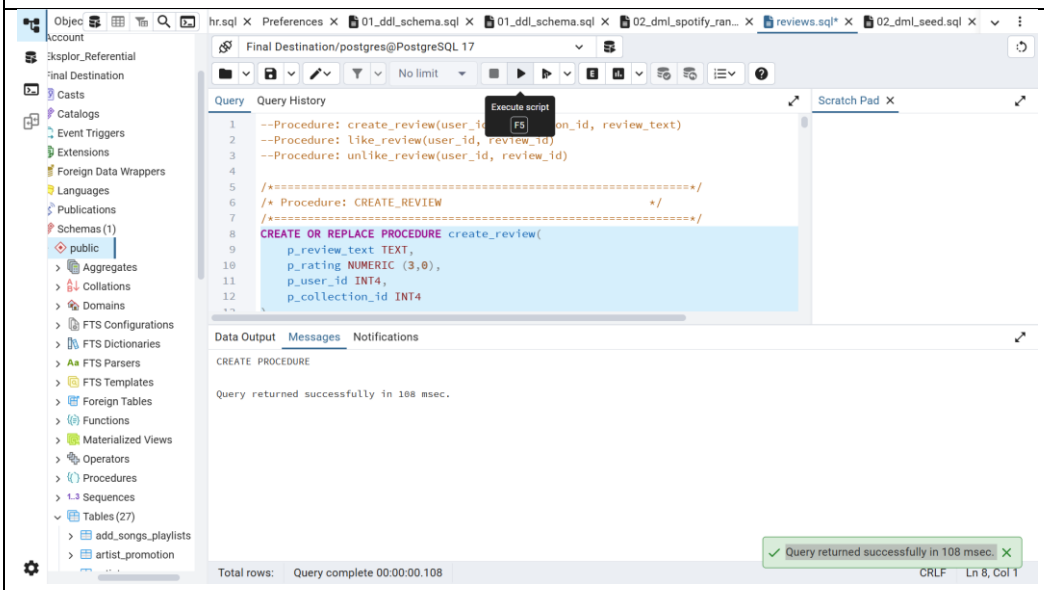
Nama Function	create_review
Parameter	p_review_text TEXT, p_rating NUMERIC (3,0), p_user_id INT4, p_collection_id INT4
Deskripsi	Membuat review baru untuk sebuah koleksi setelah memvalidasi user, koleksi, isi review, dan memastikan user belum pernah memberi review yang sama.

Script SQL

```
CREATE OR REPLACE PROCEDURE create_review(
    p_review_text TEXT,
    p_rating NUMERIC (3,0),
    p_user_id INT4,
    p_collection_id INT4
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_review_id INT;
BEGIN
    -- cek user
    IF NOT EXISTS (SELECT 1 FROM users WHERE user_id = p_user_id) THEN
        RAISE EXCEPTION 'User_id % not found', p_user_id;
    END IF;

    -- cek collection
    IF NOT EXISTS (SELECT 1 FROM collections WHERE collection_id = p_collection_id) THEN
        RAISE EXCEPTION 'Collection_id % not found', p_collection_id;
    END IF;

    -- cek review kosong
    IF p_review_text IS NULL OR LENGTH(TRIM(p_review_text)) = 0 THEN
        RAISE EXCEPTION 'Review text cannot be empty';
    END IF;

    -- cek apakah user sudah review koleksi ini
    IF EXISTS (SELECT 1 FROM reviews WHERE user_id = p_user_id
```

```
        AND collection_id = p_collection_id
    ) THEN
        RAISE EXCEPTION 'User has already reviewed this collection';
    END IF;

    -- ambil review_id dari sequence
    SELECT NEXTVAL('seq_reviews_id')
    INTO v_review_id;

    -- insert review (trigger akan update timestamp jika review diganti)
    INSERT INTO reviews(review, rating, "TIMESTAMP", review_id, user_id, collection_id)
    VALUES (p_review_text, p_rating, CURRENT_TIMESTAMP, v_review_id, p_user_id,
p_collection_id);

    RAISE NOTICE 'Review created with ID %', v_review_id;
END;
$$;
```

Screenshot Hasil



Tabel 93 PL/SQL – Procedure Toggle Like Reivew

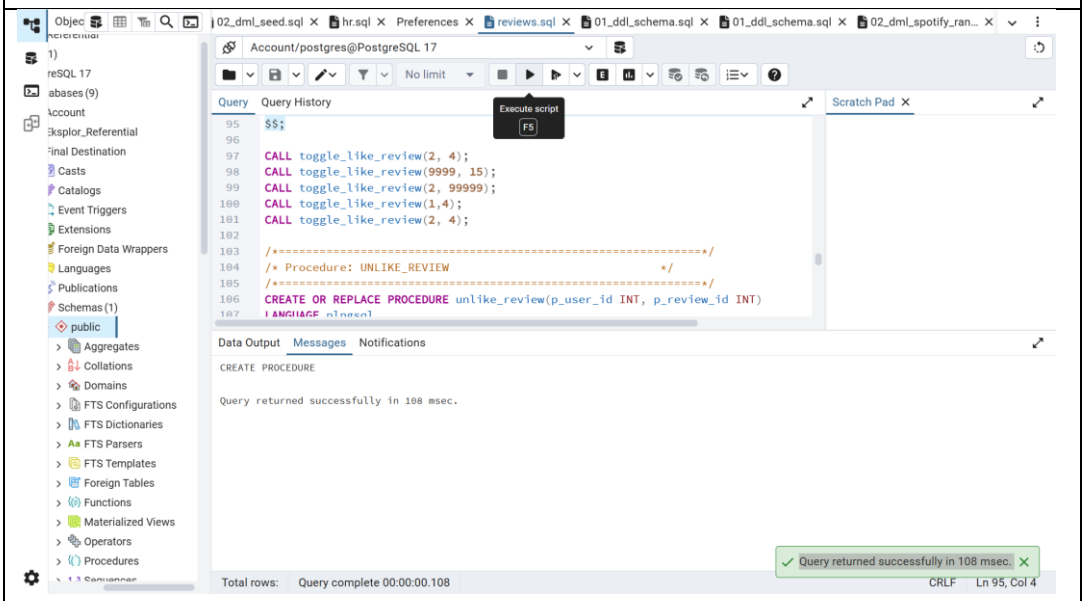
Nama Function	toggle_like_review
Parameter	p_user_id INT, p_review_id INT
Deskripsi	Melakukan like/unlike pada review tergantung apakah user sudah pernah menyukai review tersebut.

Script SQL

```
CREATE OR REPLACE PROCEDURE toggle_like_review(p_user_id INT, p_review_id INT)
LANGUAGE plpgsql
AS $$
DECLARE
    v_exists BOOLEAN;
BEGIN
    -- cek user
    IF NOT EXISTS (SELECT 1 FROM users WHERE user_id = p_user_id) THEN
```

```
        RAISE EXCEPTION 'User_id % not found', p_user_id;
    END IF;

    -- cek review
    IF NOT EXISTS (SELECT 1 FROM reviews WHERE review_id = p_review_id) THEN
        RAISE EXCEPTION 'Review_id % not found', p_review_id;
    END IF;

    --cek toggle
    SELECT EXISTS (SELECT 1 FROM like_reviews WHERE review_id = p_review_id AND user_id =
p_user_id)
    INTO v_exists;

    --jika sudah like -> unlike
    IF v_exists THEN
        DELETE FROM like_reviews WHERE review_id = p_review_id AND user_id = p_user_id;
        RAISE NOTICE 'Review % unliked by user %', p_review_id, p_user_id;
        RETURN;
    END IF;

    -- jika belum like -> insert like
    INSERT INTO like_reviews VALUES (p_review_id, p_user_id);
    RAISE NOTICE 'Review % liked by user %', p_review_id, p_user_id;
END;
$$;
```

Screenshot Hasil



Tabel 94 PL/SQL – Function Search

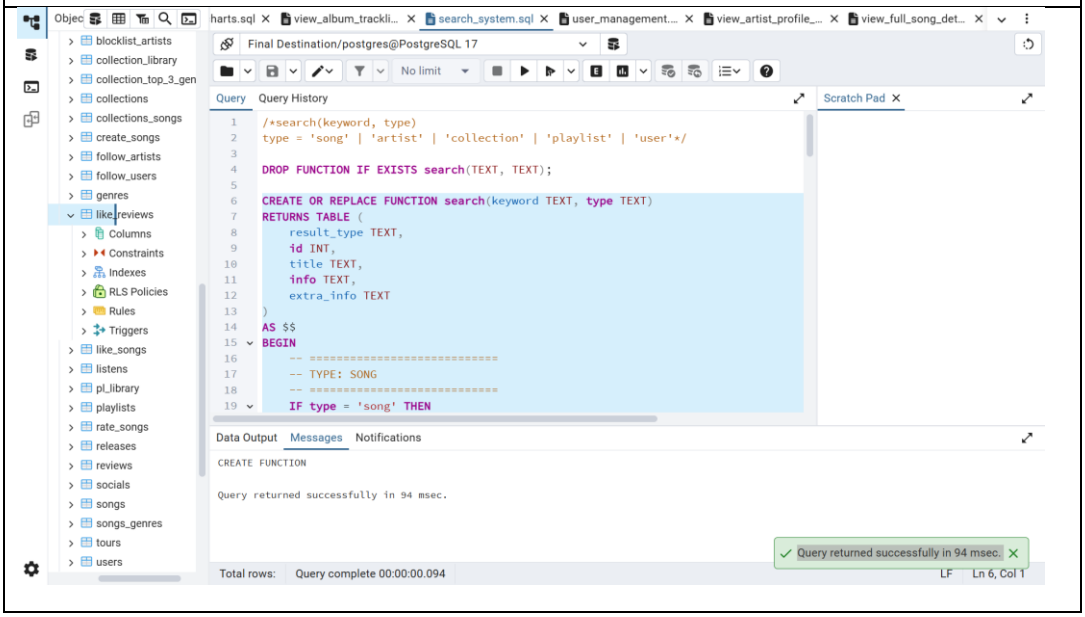
Nama Function	search
Parameter	keyword TEXT, type TEXT
Deskripsi	Mencari lagu, album, playlist, artis, atau user berdasarkan keyword dan type pencarian, lalu mengembalikan hasil dalam format unified.

Script SQL

```sql
CREATE OR REPLACE FUNCTION search(keyword TEXT, type TEXT)
RETURNS TABLE (
    result_type TEXT,
    id INT,
    title TEXT,
    info TEXT,
    extra_info TEXT
)
AS $$
BEGIN
    -- ============================
    -- TYPE: SONG
    -- ============================
    IF type = 'song' THEN
        RETURN QUERY
        SELECT
            'song' AS result_type,
            v.song_id AS id,
            v.song_title ::TEXT AS title,
            'Artist: ' || COALESCE(v.artists_name, '-') AS info,
            'Album: ' || COALESCE(v.album_name, '-') ||
            ' | Popularity: ' || COALESCE(v.popularity, 0) AS extra_info
        FROM view_full_song_details v
        WHERE v.song_title ILIKE '%' || keyword || '%'
           OR v.artists_name ILIKE '%' || keyword || '%'
           OR v.album_name ILIKE '%' || keyword || '%'
        ORDER BY v.popularity DESC NULLS LAST;

    -- ============================
    -- TYPE: COLLECTION (ALBUM)
    -- ============================
    ELSIF type = 'collection' THEN
        RETURN QUERY
        SELECT
            'collection',
            c.collection_id,
            c.collection_title ::TEXT,
            'Artist: ' || COALESCE(aa.album_artists,'-') AS info,
            'Release: ' || COALESCE(c.collection_release_date::TEXT, '-') AS extra_info
        FROM collections c
        JOIN (
            SELECT collection_id,
                STRING_AGG(artist_name, ', ') AS album_artists
            FROM releases r
            JOIN artists a ON a.artist_id = r.artist_id
            GROUP BY collection_id
        ) aa ON aa.collection_id = c.collection_id
        WHERE c.collection_title ILIKE '%' || keyword || '%'
           OR aa.album_artists ILIKE '%' || keyword || '%';

    -- ============================
    -- TYPE: PLAYLIST
    -- ============================
    ELSIF type = 'playlist' THEN
        RETURN QUERY
        SELECT
            'playlist',
            p.playlist_id,
            p.playlist_title ::TEXT,
            COALESCE(p.playlist_desc, ''),
            COUNT(asp.song_id)::TEXT AS extra
        FROM playlists p
        LEFT JOIN add_songs_playlists asp ON asp.playlist_id = p.playlist_id
        LEFT JOIN view_full_song_details v ON v.song_id = asp.song_id
        WHERE p.playlist_title ILIKE '%' || keyword || '%'
           OR COALESCE(p.playlist_desc, '') ILIKE '%' || keyword || '%'
           OR v.song_title ILIKE '%' || keyword || '%'
           OR v.artists_name ILIKE '%' || keyword || '%'
           OR v.album_name ILIKE '%' || keyword || '%'
        GROUP BY p.playlist_id, p.playlist_title, p.playlist_desc;
```

```
    -- ============================
    -- TYPE: ARTIST
    -- ============================
    ELSIF type = 'artist' THEN
        RETURN QUERY
        SELECT
            'artist',
            v.artist_id,
            v.artist_name ::TEXT,
            v.follower_count::TEXT || ' followers',
            'Albums: ' || total_albums || ' | Tracks: ' || total_tracks AS extra
        FROM view_artist_profile_header v
        WHERE v.artist_name ILIKE '%' || keyword || '%';

    -- ============================
    -- TYPE: USER
    -- ============================
    ELSIF type = 'user' THEN
        RETURN QUERY
        SELECT
            'user',
            u.user_id,
            u.username ::TEXT,
            u.followers_count::TEXT || ' followers',
            'Following: ' || u.following_count ||
            ' | Public playlists: ' || u.public_playlists AS extra
        FROM view_user_library_stats u
        WHERE u.username ILIKE '%' || keyword || '%';

    ELSE
        RAISE EXCEPTION 'Invalid type: %, valid types = song | artist | collection |
playlist | user', type;
    END IF;
END;
$$ LANGUAGE plpgsql STABLE;
```

Screenshot Hasil



Tabel 95 PL/SQL – Function Get Artist Detail

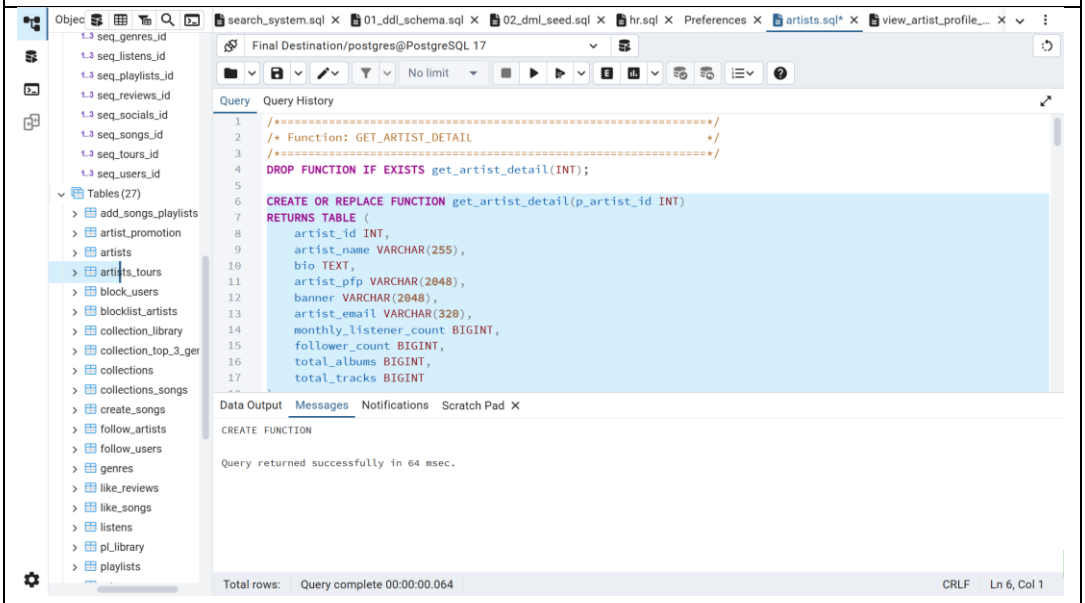
Nama Function	get_artist_detail
Parameter	p_artist_id INT

Deskripsi	Mengambil informasi lengkap profil artis dari view_artist_profile_header berdasarkan artist_id.

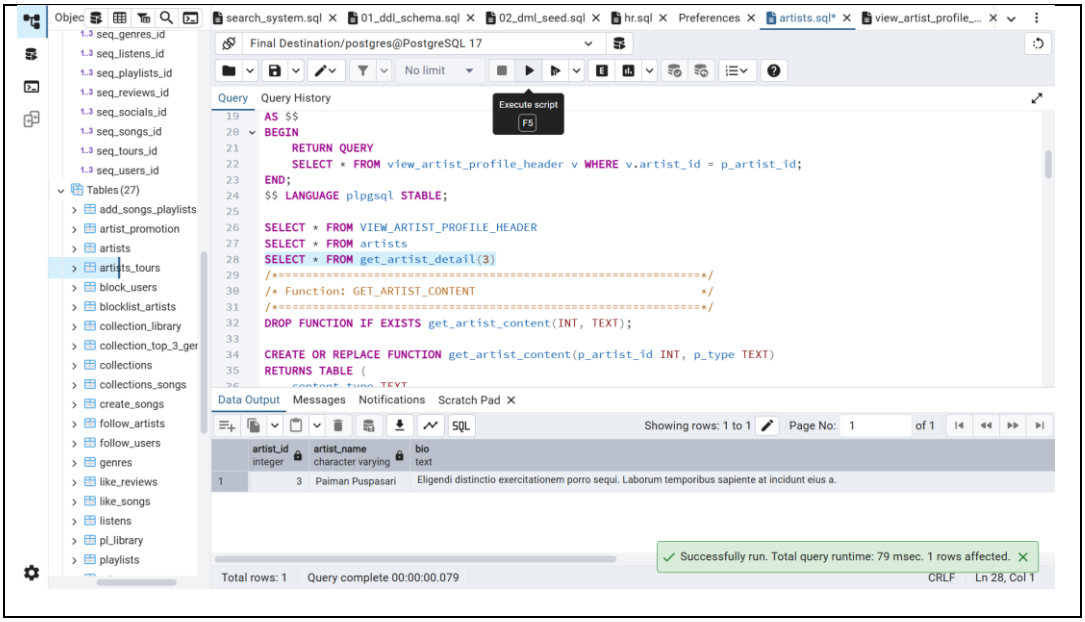
Script SQL

```sql
CREATE OR REPLACE FUNCTION get_artist_detail(p_artist_id INT)
RETURNS TABLE (
    artist_id INT,
    artist_name VARCHAR(255),
    bio TEXT,
    artist_pfp VARCHAR(2048),
    banner VARCHAR(2048),
    artist_email VARCHAR(320),
    monthly_listener_count BIGINT,
    follower_count BIGINT,
    total_albums BIGINT,
    total_tracks BIGINT
)
AS $$
BEGIN
    RETURN QUERY
    SELECT * FROM view_artist_profile_header v WHERE v.artist_id = p_artist_id;
END;
$$ LANGUAGE plpgsql STABLE;
```

Screenshot Hasil




```sql
SELECT * FROM get_artist_detail(3)
```



Tabel 96 PL/SQL – Function Get Artist Content

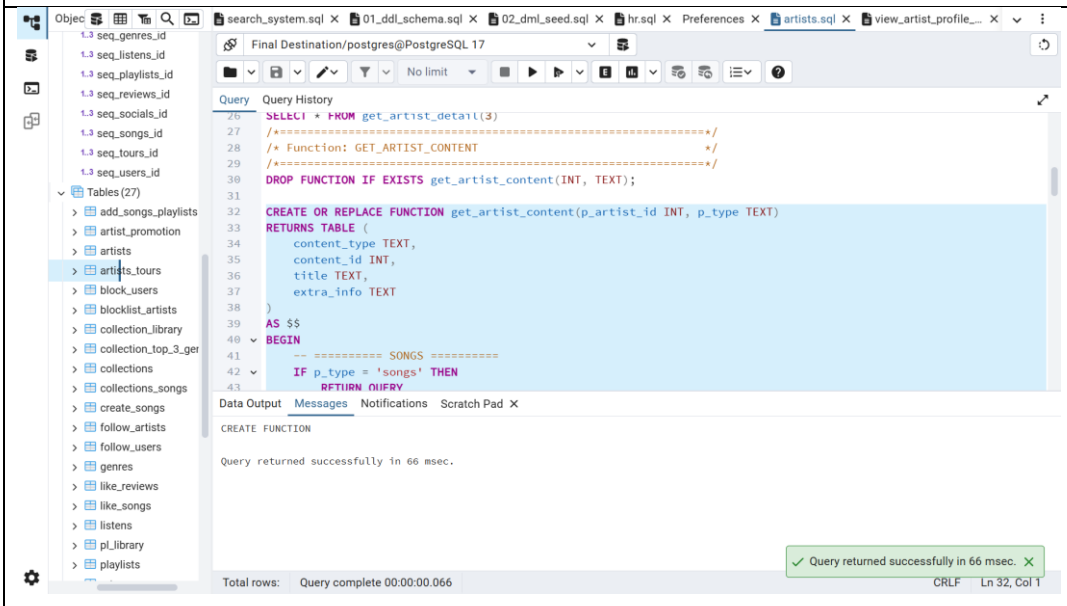
Nama Function	get_artist_content
Parameter	p_artist_id INT, p_type TEXT
Deskripsi	Mengambil daftar konten artis (lagu, koleksi, tur, atau promosi) sesuai kategori yang diminta.

Script SQL

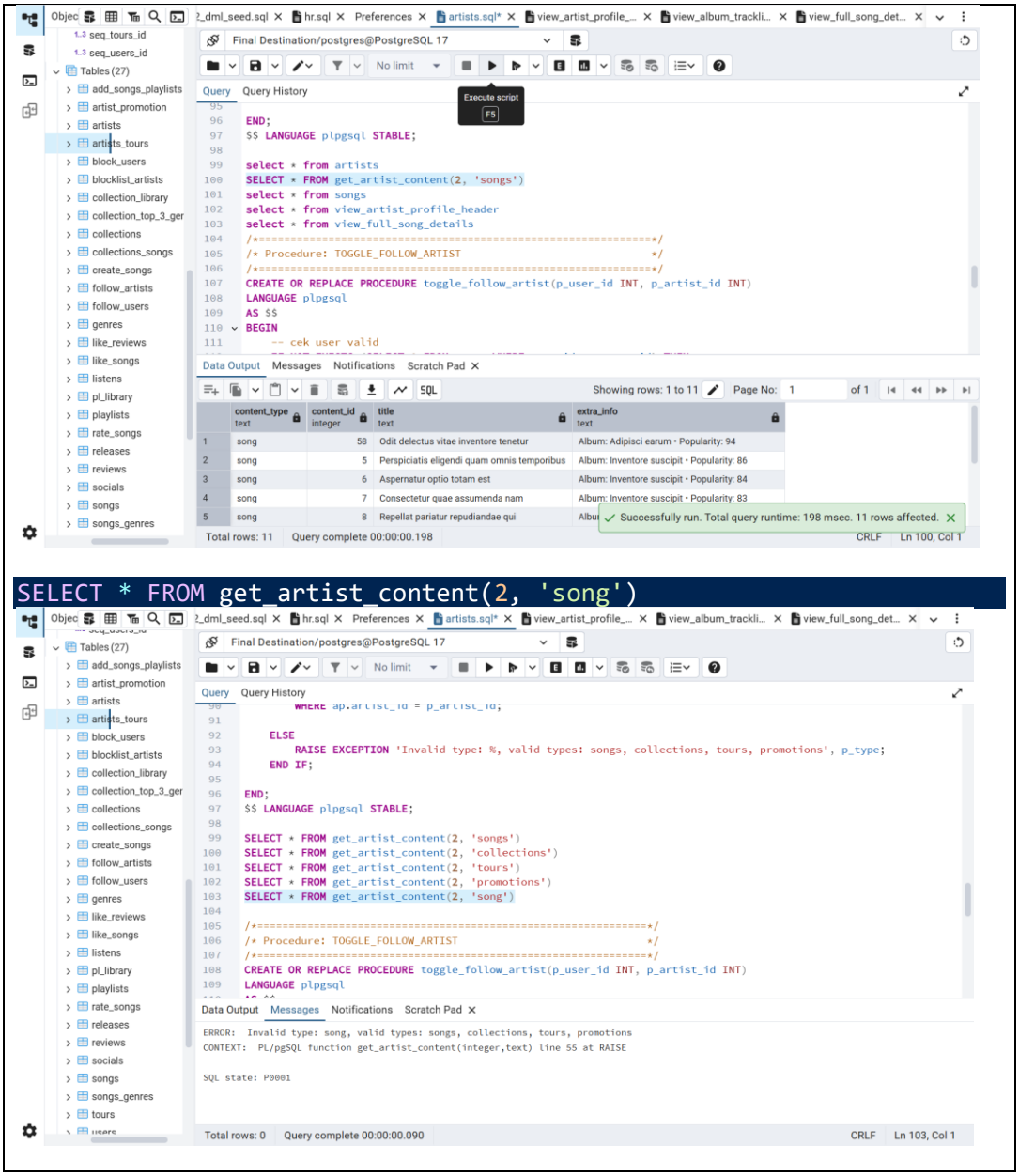
```
CREATE OR REPLACE FUNCTION get_artist_content(p_artist_id INT, p_type TEXT)
RETURNS TABLE (
    content_type TEXT,
    content_id INT,
    title TEXT,
    extra_info TEXT
)
AS $$
BEGIN
    -- ========== SONGS ==========
    IF p_type = 'songs' THEN
        RETURN QUERY
        SELECT
            'song',
            v.song_id,
            v.song_title ::TEXT,
            'Album: ' || COALESCE(v.album_name, '-') || ' • Popularity: ' || v.popularity
        FROM view_full_song_details v
        WHERE v.artist_name ILIKE (
            SELECT '%' || artist_name || '%'
            FROM artists WHERE artist_id = p_artist_id
        )
        ORDER BY v.popularity DESC;

    -- ========== COLLECTIONS ==========
    ELSIF p_type = 'collections' THEN
        RETURN QUERY
        SELECT
            'collection',
            c.collection_id,
            c.collection_title ::TEXT,
            c.collection_type ::TEXT
```

```
        FROM releases r
        JOIN collections c ON c.collection_id = r.collection_id
        WHERE r.artist_id = p_artist_id;

    -- ========== TOURS ==========
    ELSIF p_type = 'tours' THEN
        RETURN QUERY
        SELECT
            'tour',
            t.tour_id,
            t.tour_name ::TEXT,
            t.venue || ' • ' || t.tour_date
        FROM artists_tours at
        JOIN tours t ON t.tour_id = at.tour_id
        WHERE at.artist_id = p_artist_id;

    -- ========== PROMOTIONS ==========
    ELSIF p_type = 'promotions' THEN
        RETURN QUERY
        SELECT
            'promotion',
            ap.collection_id,
            c.collection_title ::TEXT,
            ap.komentar_promosi
        FROM artist_promotion ap
        LEFT JOIN collections c ON c.collection_id = ap.collection_id
        WHERE ap.artist_id = p_artist_id;

    ELSE
        RAISE EXCEPTION 'Invalid type: %, valid types: songs, collections, tours,
promotions', p_type;
    END IF;

END;
$$ LANGUAGE plpgsql STABLE;
```

Screenshot Hasil




```
SELECT * FROM get_artist_content(2, 'songs')
```



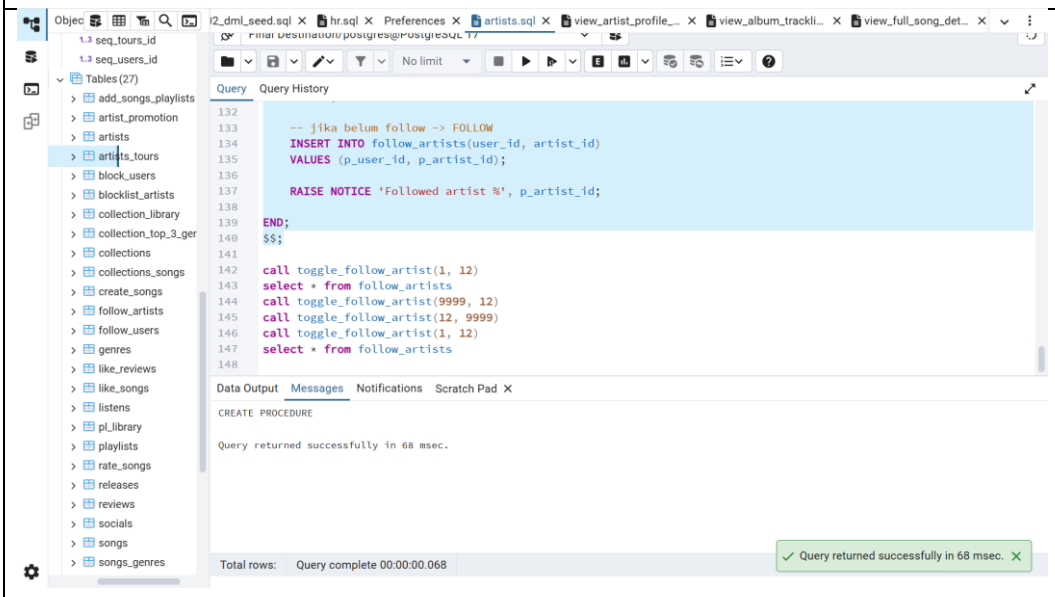
Tabel 97 PL/SQL – Procedure Toggle Follow Artist

Nama Function	toggle_follow_artist
Parameter	p_user_id INT, p_artist_id INT
Deskripsi	Mengaktifkan fitur follow/unfollow artis dalam satu prosedur berdasarkan status relasi sebelumnya.
Script SQL	

```
CREATE OR REPLACE PROCEDURE toggle_follow_artist(p_user_id INT, p_artist_id INT)
LANGUAGE plpgsql
AS $$
BEGIN
    -- cek user valid
    IF NOT EXISTS (SELECT 1 FROM users WHERE user_id = p_user_id) THEN
        RAISE EXCEPTION 'User_id % not found', p_user_id;
```

```
    END IF;

    -- cek artist valid
    IF NOT EXISTS (SELECT 1 FROM artists WHERE artist_id = p_artist_id) THEN
        RAISE EXCEPTION 'Artist_id % not found', p_artist_id;
    END IF;

    -- jika sudah follow -> UNFOLLOW
    IF EXISTS (
        SELECT 1 FROM follow_artists
        WHERE user_id = p_user_id AND artist_id = p_artist_id
    ) THEN
        DELETE FROM follow_artists
        WHERE user_id = p_user_id AND artist_id = p_artist_id;

        RAISE NOTICE 'Unfollowed artist %', p_artist_id;
        RETURN;
    END IF;

    -- jika belum follow -> FOLLOW
    INSERT INTO follow_artists(user_id, artist_id)
    VALUES (p_user_id, p_artist_id);

    RAISE NOTICE 'Followed artist %', p_artist_id;

END;
$$;
```

Screenshot Hasil

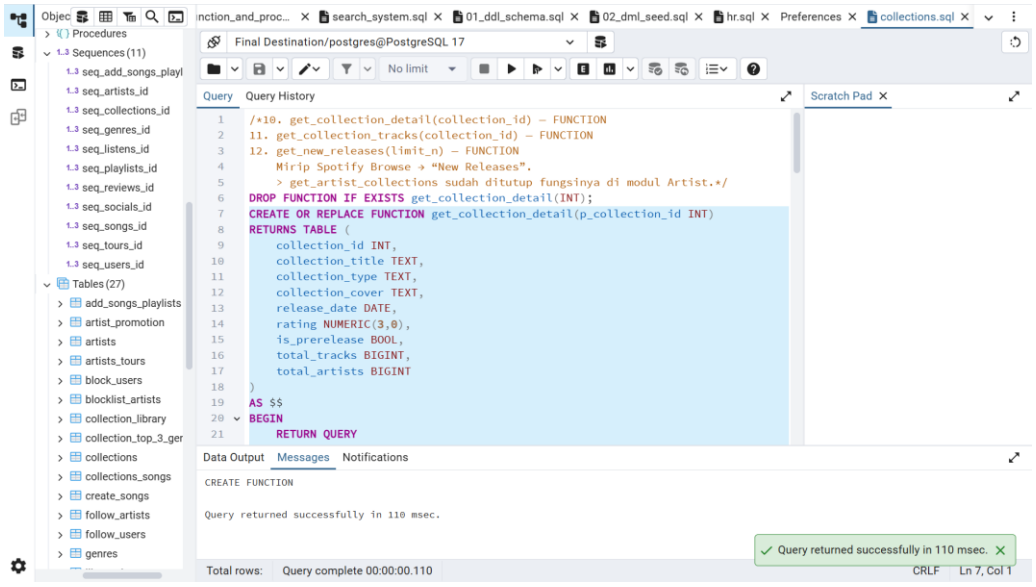


Tabel 98 PL/SQL – Function Get Collection Detail

Nama Function	get_collection_detail
Parameter	p_collection_id INT
Deskripsi	Mengambil detail lengkap sebuah koleksi (album/EP/single) beserta jumlah lagu dan jumlah artis yang terlibat.
Script SQL	

```sql
CREATE OR REPLACE FUNCTION get_collection_detail(p_collection_id INT)
RETURNS TABLE (
    collection_id INT,
    collection_title TEXT,
    collection_type TEXT,
    collection_cover TEXT,
    release_date DATE,
    rating NUMERIC(3,0),
    is_prerelease BOOL,
    total_tracks BIGINT,
    total_artists BIGINT
)
AS $$
BEGIN
    RETURN QUERY
    SELECT
        c.collection_id,
        c.collection_title ::TEXT,
        c.collection_type ::TEXT,
        c.collection_cover ::TEXT,
        c.collection_release_date,
        c.collection_rating,
        c.isprerelease,
        (SELECT COUNT(*) FROM collections_songs cs
         WHERE cs.collection_id = c.collection_id) AS total_tracks,
        (SELECT COUNT(DISTINCT r.artist_id)
         FROM releases r
         WHERE r.collection_id = c.collection_id) AS total_artists
    FROM collections c
    WHERE c.collection_id = p_collection_id;
END;
$$ LANGUAGE plpgsql STABLE;
```

Screenshot Hasil



Tabel 99 PL/SQL – Function Get Collection Tracks

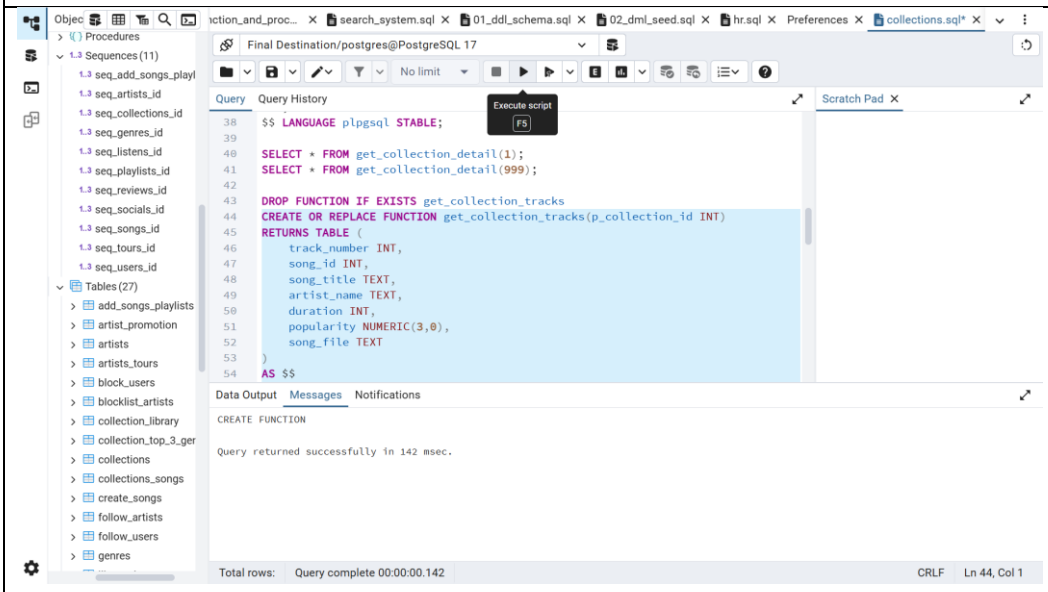
Nama Function	get_collection_tracks
Parameter	p_collection_id INT

Deskripsi	Mengembalikan daftar track dalam sebuah koleksi lengkap dengan artis, durasi, file audio, dan urutan track.

Script SQL

```sql
CREATE OR REPLACE FUNCTION get_collection_tracks(p_collection_id INT)
RETURNS TABLE (
    track_number INT,
    song_id INT,
    song_title TEXT,
    artist_name TEXT,
    duration INT,
    popularity NUMERIC(3,0),
    song_file TEXT
)
AS $$
BEGIN
    RETURN QUERY
    SELECT
        cs.nomor_track,
        v.song_id,
        v.song_title ::TEXT,
        v.artist_name ::TEXT,
        v.song_duration,
        v.popularity,
        v.song_file ::TEXT
    FROM collections_songs cs
    JOIN view_full_song_details v ON v.song_id = cs.song_id
    WHERE cs.collection_id = p_collection_id
    ORDER BY cs.nomor_disc, cs.nomor_track;
END;
$$ LANGUAGE plpgsql STABLE;
```

Screenshot Hasil



Tabel 100 PL/SQL – Function Get New Releases

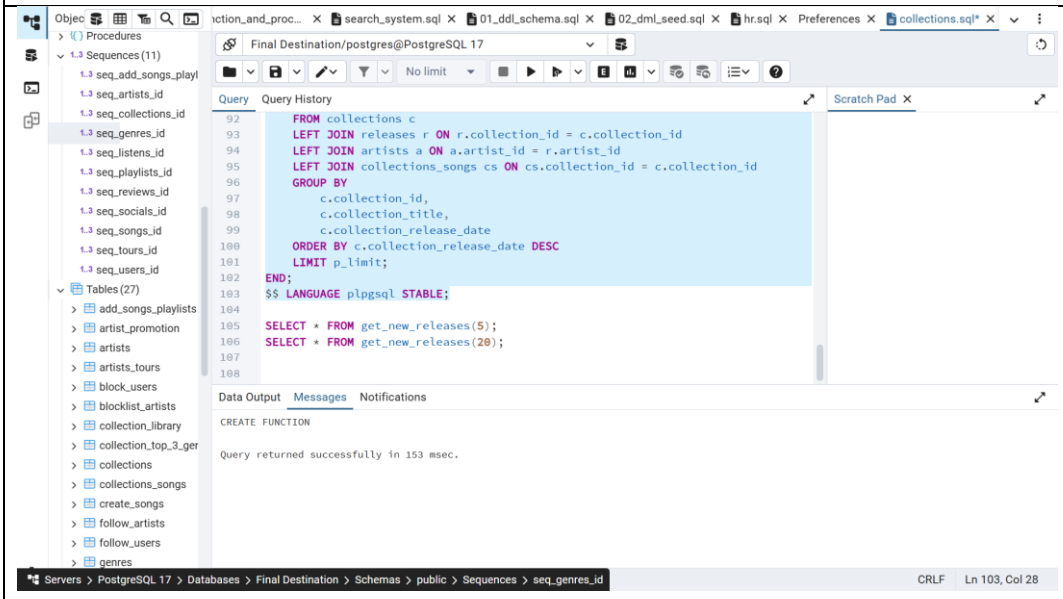
Nama Function	get_new_realeses
Parameter	p_limit INT

Deskripsi	Mengambil daftar rilis terbaru (dibatasi p_limit) beserta artis dan total lagu dalam setiap koleksi

Script SQL

```
CREATE OR REPLACE FUNCTION get_new_releases(p_limit INT)
RETURNS TABLE (
    collection_id INT,
    title TEXT,
    artist_name TEXT,
    release_date DATE,
    total_tracks BIGINT
)
AS $$
BEGIN
    RETURN QUERY
    SELECT
        c.collection_id,
        c.collection_title ::TEXT,
        STRING_AGG(DISTINCT a.artist_name, ', ') AS artist_name,
        c.collection_release_date,
        COUNT(cs.song_id) AS total_tracks
    FROM collections c
    LEFT JOIN releases r ON r.collection_id = c.collection_id
    LEFT JOIN artists a ON a.artist_id = r.artist_id
    LEFT JOIN collections_songs cs ON cs.collection_id = c.collection_id
    GROUP BY
        c.collection_id,
        c.collection_title,
        c.collection_release_date
    ORDER BY c.collection_release_date DESC
    LIMIT p_limit;
END;
$$ LANGUAGE plpgsql STABLE;
```

Screenshot Hasil



Tabel 101 PL/SQL – Procedure Add Artist Promotion

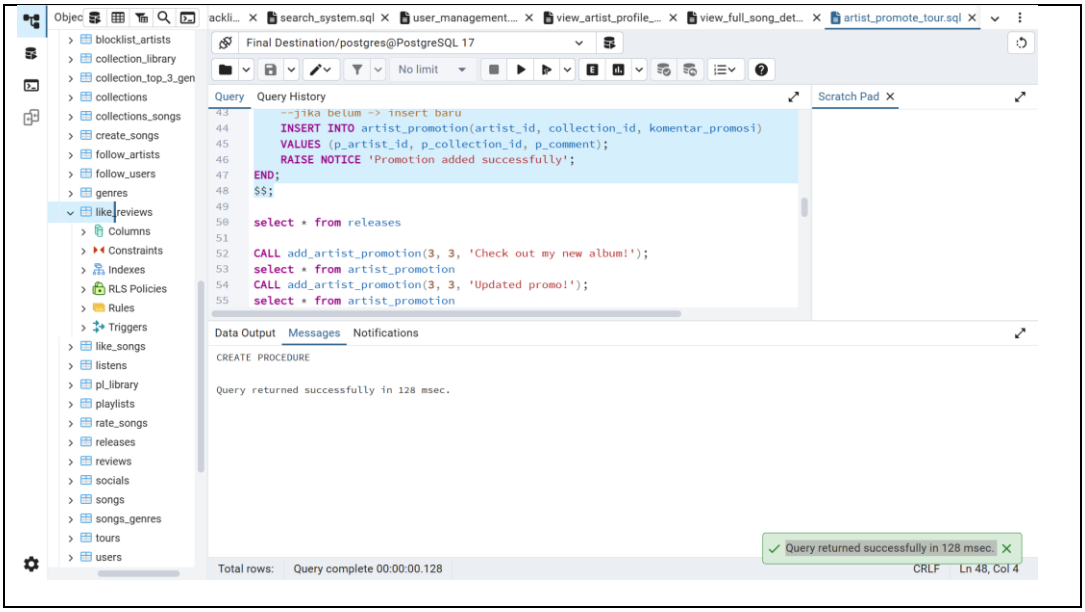
Nama Function	add_artist_promotion
Parameter	p_artist_id INT, p_collection_id INT, p_comment TEXT

Deskripsi	Menambahkan atau memperbarui komentar promosi seorang artis untuk sebuah koleksi; hanya diizinkan jika koleksi tersebut memang dirilis oleh artis tersebut.

Script SQL

```
CREATE OR REPLACE PROCEDURE add_artist_promotion(p_artist_id INT, p_collection_id INT,
p_comment TEXT)
LANGUAGE plpgsql
AS $$
BEGIN

    -- cek artist valid
    IF NOT EXISTS (SELECT 1 FROM artists WHERE artist_id = p_artist_id) THEN
        RAISE EXCEPTION 'Artist_id % not found', p_artist_id;
    END IF;

    -- cek collection ada
    IF NOT EXISTS (SELECT 1 FROM collections WHERE collection_id = p_collection_id) THEN
        RAISE EXCEPTION 'Collection_id % not found', p_collection_id;
    END IF;

    --validasi, apakah koleksi ini dirilis oleh artis tersebut?
    IF NOT EXISTS (SELECT 1 FROM releases WHERE artist_id = p_artist_id
        AND collection_id = p_collection_id
    ) THEN
        RAISE EXCEPTION 'Artist % cannot promote collection %, because it does not belong
to them',
            p_artist_id, p_collection_id;
    END IF;

    --jika sudah pernah promosi -> update komentar
    IF EXISTS (SELECT 1 FROM artist_promotion WHERE artist_id = p_artist_id
        AND collection_id = p_collection_id
    ) THEN
        UPDATE artist_promotion SET komentar_promosi = p_comment WHERE artist_id =
p_artist_id
        AND collection_id = p_collection_id;

        RAISE NOTICE 'Promotion updated successfully';
        RETURN;
    END IF;

    --jika belum -> insert baru
    INSERT INTO artist_promotion(artist_id, collection_id, komentar_promosi)
    VALUES (p_artist_id, p_collection_id, p_comment);
    RAISE NOTICE 'Promotion added successfully';
END;
$$;
```

Screenshot Hasil



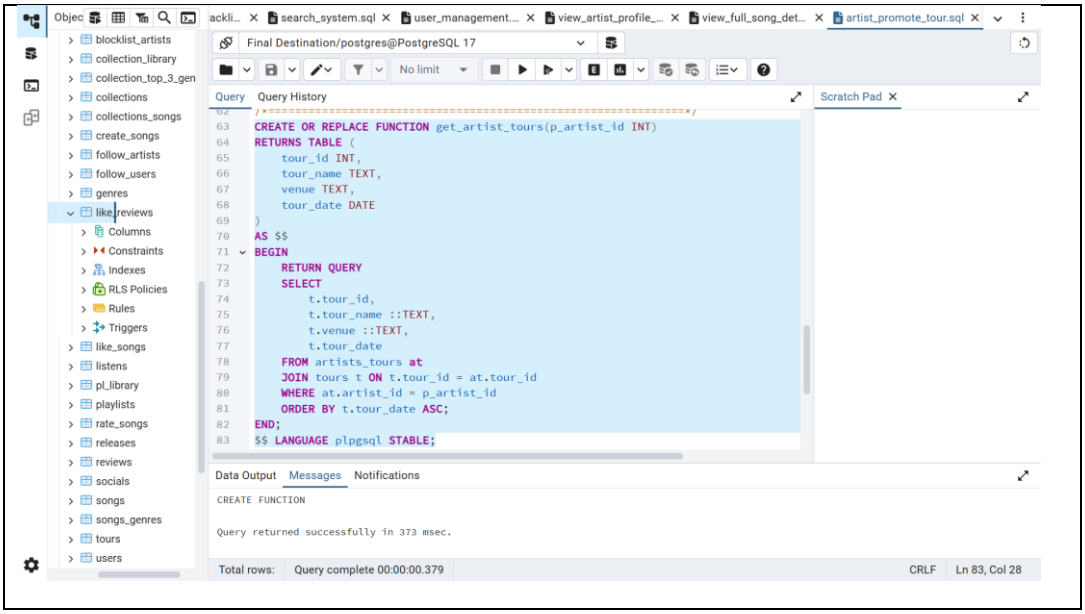
Tabel 102 PL/SQL – Function Get Artist Tours

Nama Function	get_artists_tours
Parameter	p_artist_id INT
Deskripsi	Mengambil daftar tur dari artis berdasarkan tabel artists_tours dan tours, diurutkan berdasarkan tanggal tur.

Script SQL

```
CREATE OR REPLACE FUNCTION get_artist_tours(p_artist_id INT)
RETURNS TABLE (
    tour_id INT,
    tour_name TEXT,
    venue TEXT,
    tour_date DATE
)
AS $$
BEGIN
    RETURN QUERY
    SELECT
        t.tour_id,
        t.tour_name ::TEXT,
        t.venue ::TEXT,
        t.tour_date
    FROM artists_tours at
    JOIN tours t ON t.tour_id = at.tour_id
    WHERE at.artist_id = p_artist_id
    ORDER BY t.tour_date ASC;
END;
$$ LANGUAGE plpgsql STABLE;
```

Screenshot Hasil


Final Destination/postgres@PostgreSQL 17
search_system.sql
user_management....
view_artist_profile_...
view_full_song_det...
artist_promote_tour.sql
blocklist_artists
collection_library
collection_top_3_gen
collections
collections_songs
create_songs
follow_artists
follow_users
genres
like_reviews
Columns
Constraints
Indexes
RLS Policies
Rules
Triggers
like_songs
listens
pl_library
playlists
rate_songs
releases
reviews
socials
songs
songs_genres
tours
users
Query
Query History
Scratch Pad
63 CREATE OR REPLACE FUNCTION get_artist_tours(p_artist_id INT)
64 RETURNS TABLE (
65 tour_id INT,
66 tour_name TEXT,
67 venue TEXT,
68 tour_date DATE
69)
70 AS $$
71 BEGIN
72 RETURN QUERY
73 SELECT
74 t.tour_id,
75 t.tour_name ::TEXT,
76 t.venue ::TEXT,
77 t.tour_date
78 FROM artists_tours at
79 JOIN tours t ON t.tour_id = at.tour_id
80 WHERE at.artist_id = p_artist_id
81 ORDER BY t.tour_date ASC;
82 END;
83 $$ LANGUAGE plpgsql STABLE;
Data Output
Messages
Notifications
CREATE FUNCTION
Query returned successfully in 373 msec.
Total rows:
Query complete 00:00:00.379
CRLF
Ln 83, Col 28

Tabel 103 PL/SQL – Function delete_expired_tours

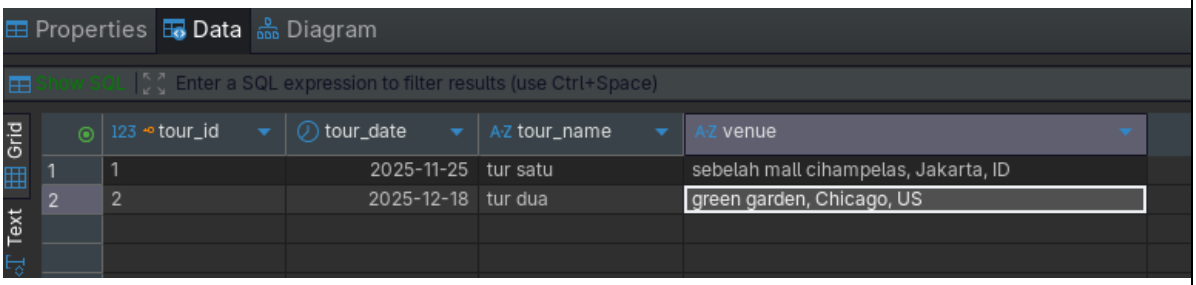
Nama Function	delete_expired_tours
Parameter	
Deskripsi	Menghapus semua entry tour yang tanggalnya sudah lewat hari ini (tour_date < CURRENT_DATE). Dalam bentuk function karena merupakan sebuah cron.

Script SQL

```
CREATE OR REPLACE FUNCTION delete_expired_tours()
RETURNS VOID AS $$
BEGIN
    DELETE FROM tours
    WHERE tour_date < CURRENT_DATE;
END;
$$ LANGUAGE plpgsql;
```

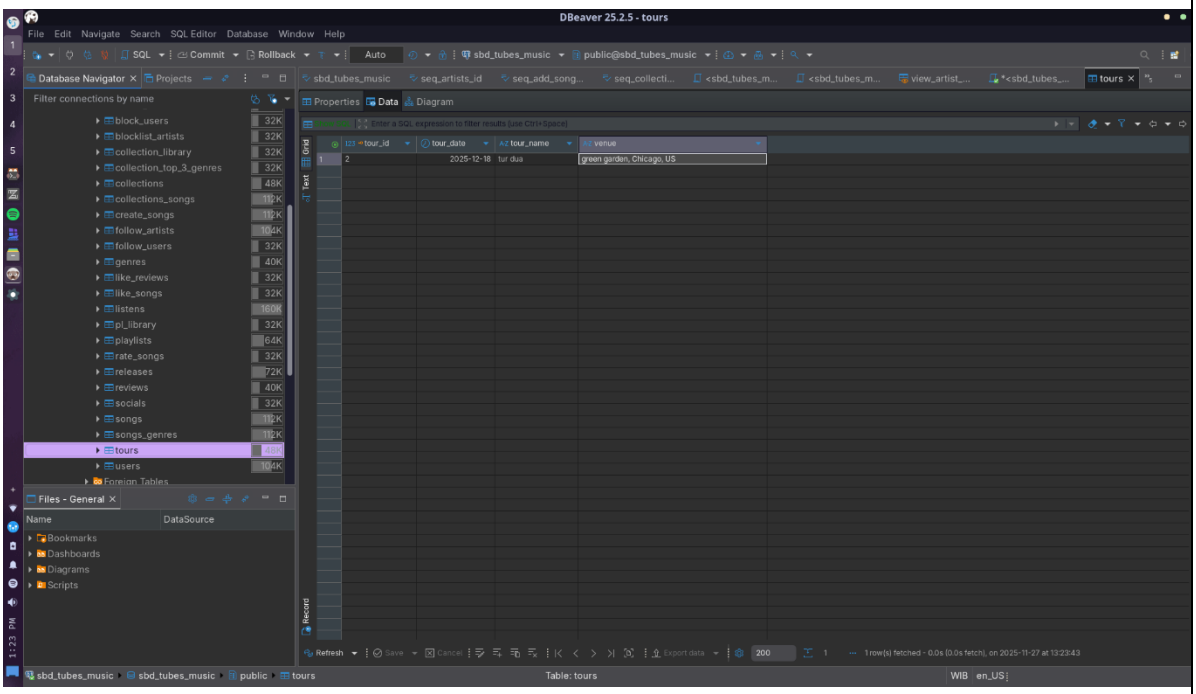
Screenshot Hasil

Before



*current_date = 2025-11-27

After



Tabel 104 PL/SQL – Function recalc_listen_count

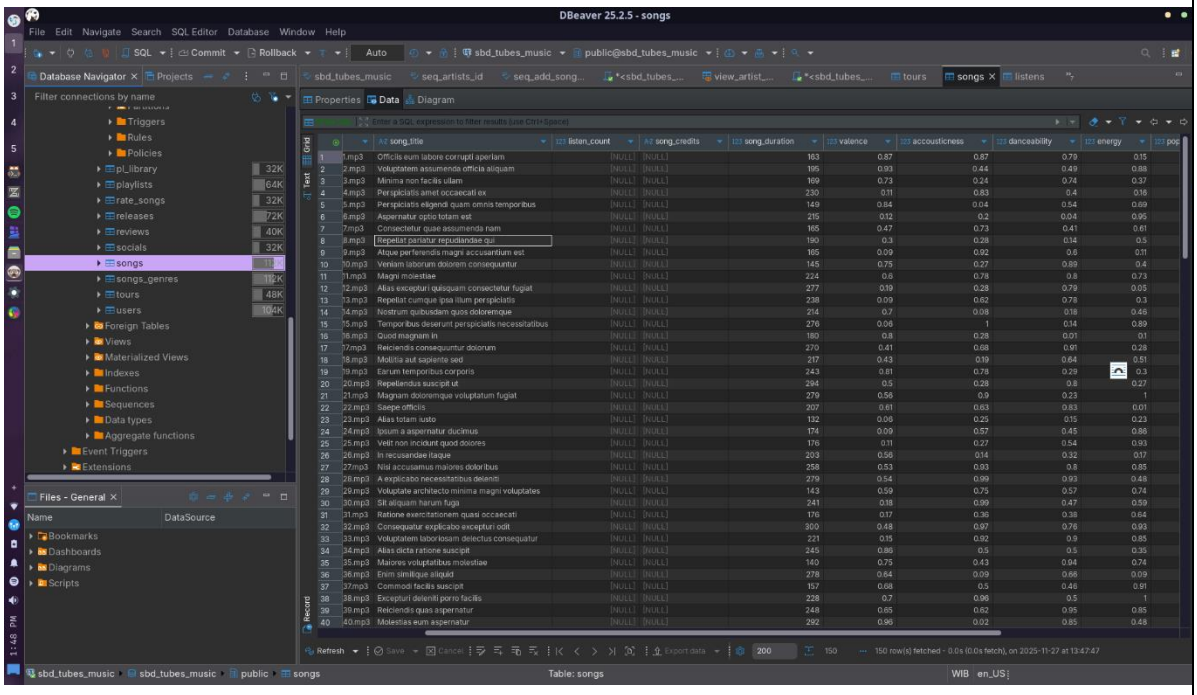
Nama Function	recalc_listen_count
Parameter	
Deskripsi	Menghitung listen_count dan mengupdate nya untuk setiap lagu. Dalam bentuk function karena merupakan sebuah cron.

Script SQL

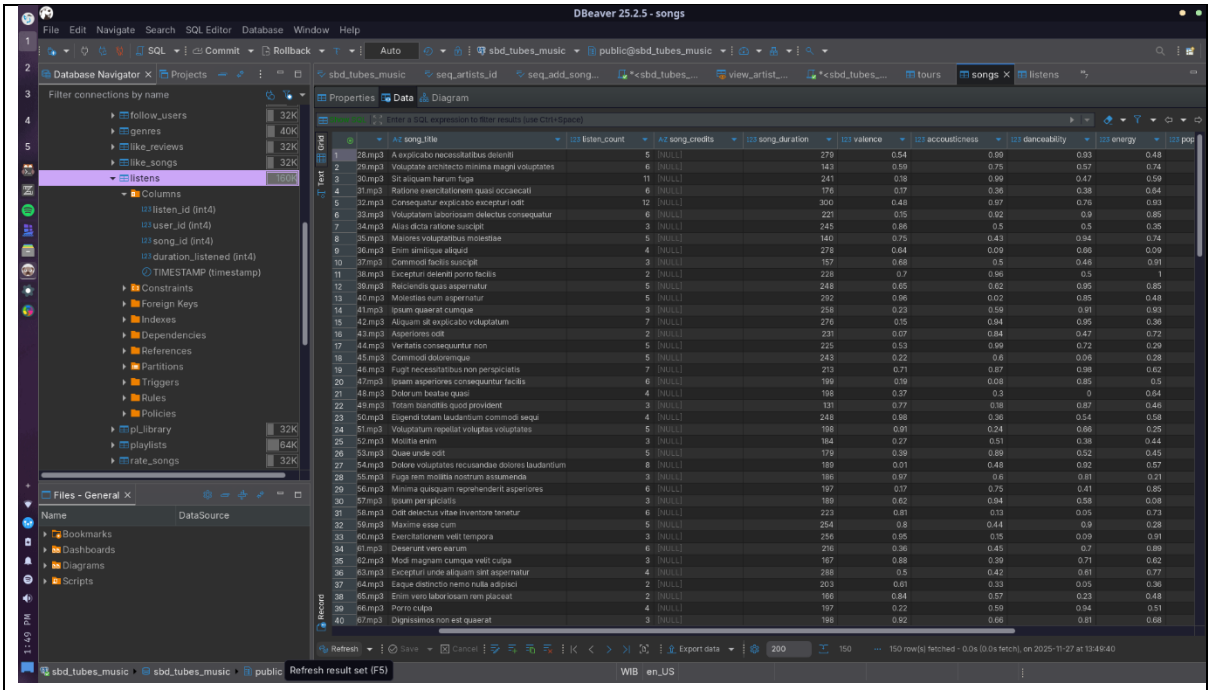
```
CREATE OR REPLACE FUNCTION recalc_listen_count()
RETURNS void AS $$
BEGIN
    UPDATE songs s
    SET listen_count = sub.cnt
    FROM (
        SELECT song_id, COUNT(*) AS cnt
        FROM listens
        GROUP BY song_id
    ) sub
    WHERE s.song_id = sub.song_id;
END;
$$ LANGUAGE plpgsql;
```

Screenshot Hasil

Before



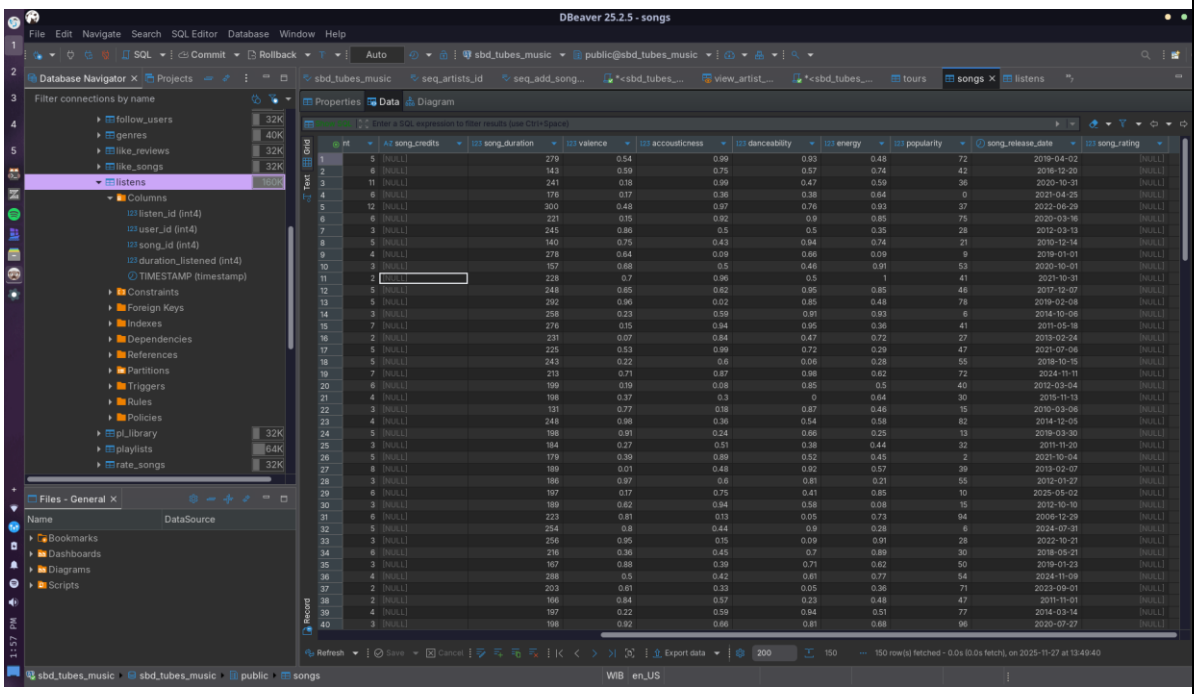
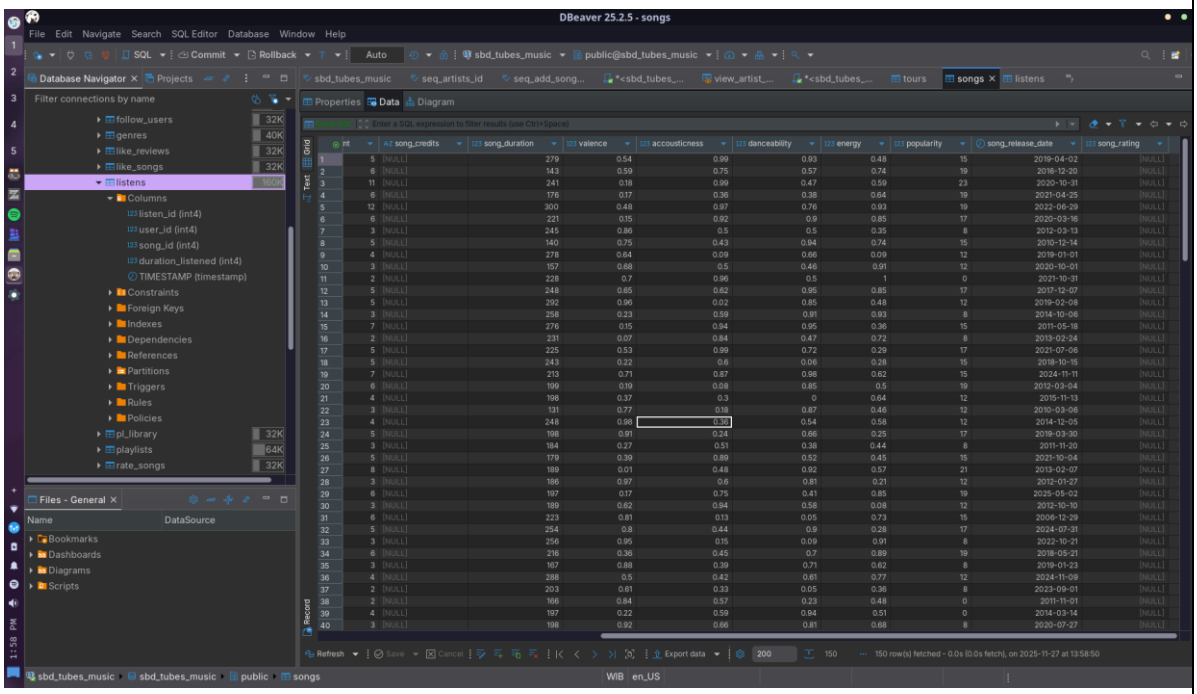
After



Tabel 105 PL/SQL – Function recalculate_all_song_popularity

Nama Function	recalculate_all_song_popularity
Parameter	
Deskripsi	Menghitung popularity suatu lagu dengan rumus tertentu dan mengupdate nya untuk setiap lagu. Dalam bentuk function karena merupakan sebuah cron.
Script SQL	

```
CREATE OR REPLACE FUNCTION recalculate_all_song_popularity()
RETURNS VOID AS $$
BEGIN
    -- hitung play maksimum dalam 30 hari terakhir
    WITH play_counts AS (
        SELECT
            song_id,
            COUNT(*) AS plays_30
        FROM listens
        WHERE "TIMESTAMP" >= CURRENT_DATE - INTERVAL '30 days'
        GROUP BY song_id
    ),
    max_play AS (
        SELECT MAX(plays_30) AS max_plays FROM play_counts
    )
    UPDATE songs s
    SET popularity =
        CASE
            WHEN pc.plays_30 IS NULL THEN 0    -- tidak ada play sama sekali -> 0
            WHEN mp.max_plays = 0 THEN 0       -- kalau kosong
            ELSE ROUND( (pc.plays_30::numeric / mp.max_plays::numeric) * 100 )
        END
    FROM play_counts pc, max_play mp
    WHERE s.song_id = pc.song_id;
END;
$$ LANGUAGE plpgsql;
```

Screenshot Hasil
Before  After 

Tabel 106 PL/SQL - Function recalculate_monthly_listeners

Nama Function	recalculate_monthly_listeners

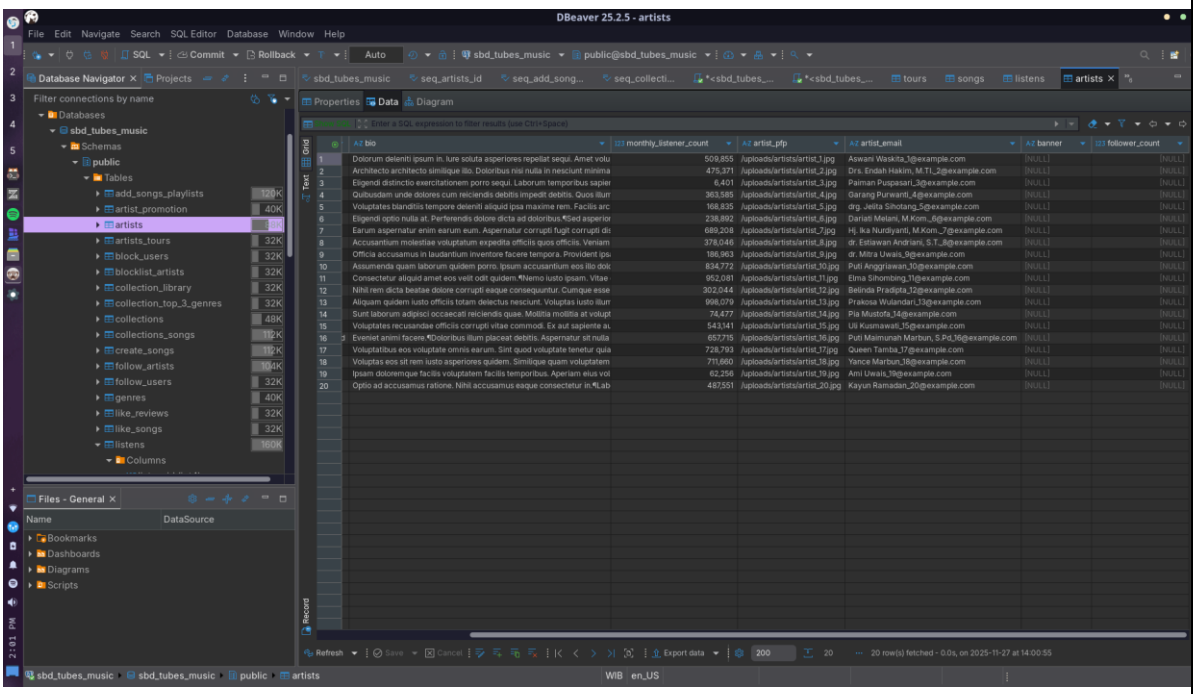
Parameter	
Deskripsi	Menghitung monthly_listeners suatu artis. Dalam bentuk function karena merupakan sebuah cron.

Script SQL

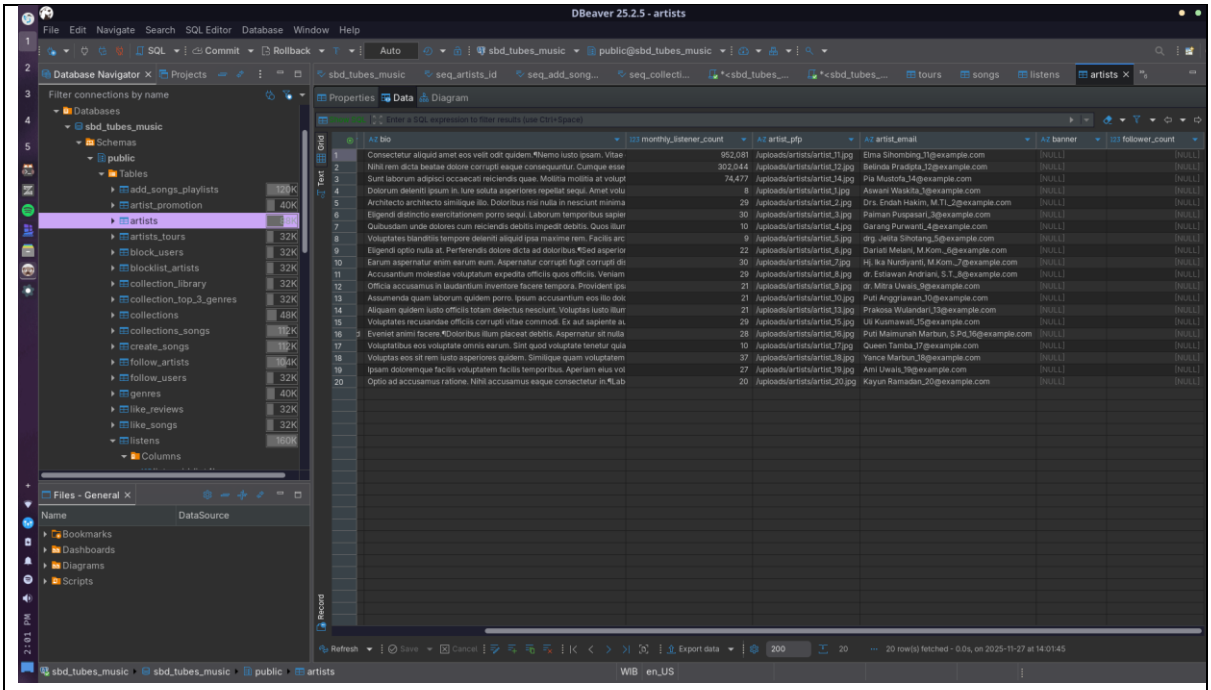
```
CREATE OR REPLACE FUNCTION recalculate_monthly_listeners()
RETURNS VOID AS $$
BEGIN
    UPDATE artists a
    SET monthly_listener_count = sub.cnt
    FROM (
        SELECT
            cs.artist_id,
            COUNT(DISTINCT l.user_id) AS cnt
        FROM listens l
        JOIN create_songs cs ON cs.song_id = l.song_id
        WHERE DATE_TRUNC('month', l."TIMESTAMP") = DATE_TRUNC('month', CURRENT_DATE)
        GROUP BY cs.artist_id
    ) sub
    WHERE a.artist_id = sub.artist_id;
END;
$$ LANGUAGE plpgsql;
```

Screenshot Hasil

Before



After



Tabel 107 PL/SQL - Function update_prerelease_daily

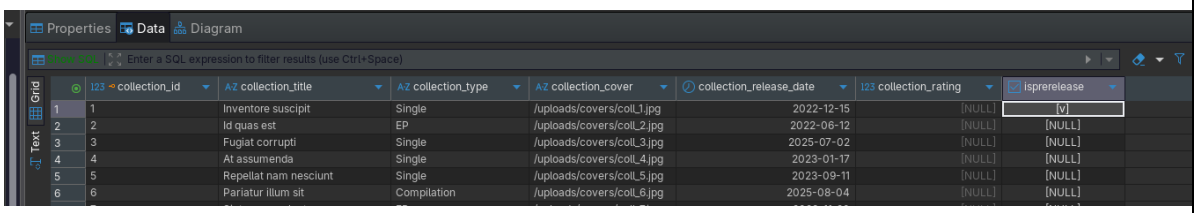
Nama Function	update_prerelease_daily
Parameter	
Deskripsi	Mengubah value isPrerelease suatu collection jika sudah lewat tanggal rilis collection tersebut. Dalam bentuk function karena merupakan sebuah cron.

Script SQL

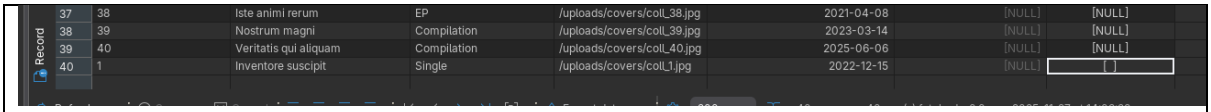
```
CREATE OR REPLACE FUNCTION update_prerelease_daily()
RETURNS VOID AS $$
BEGIN
    UPDATE collections
    SET isPrerelease = FALSE
    WHERE isPrerelease = TRUE
      AND collection_release_date <= CURRENT_DATE;
END;
$$ LANGUAGE plpgsql;
```

Screenshot Hasil

Before



After



37	38	Iste animi rerum	EP	/uploads/covers/coll_38.jpg	2021-04-08	[NULL]	[NULL]
38	39	Nostrum magni	Compilation	/uploads/covers/coll_39.jpg	2023-03-14	[NULL]	[NULL]
39	40	Veritatis qui aliquam	Compilation	/uploads/covers/coll_40.jpg	2025-06-06	[NULL]	[NULL]
40	1	Inventore suscipit	Single	/uploads/covers/coll_1.jpg	2022-12-15	[NULL]	[]

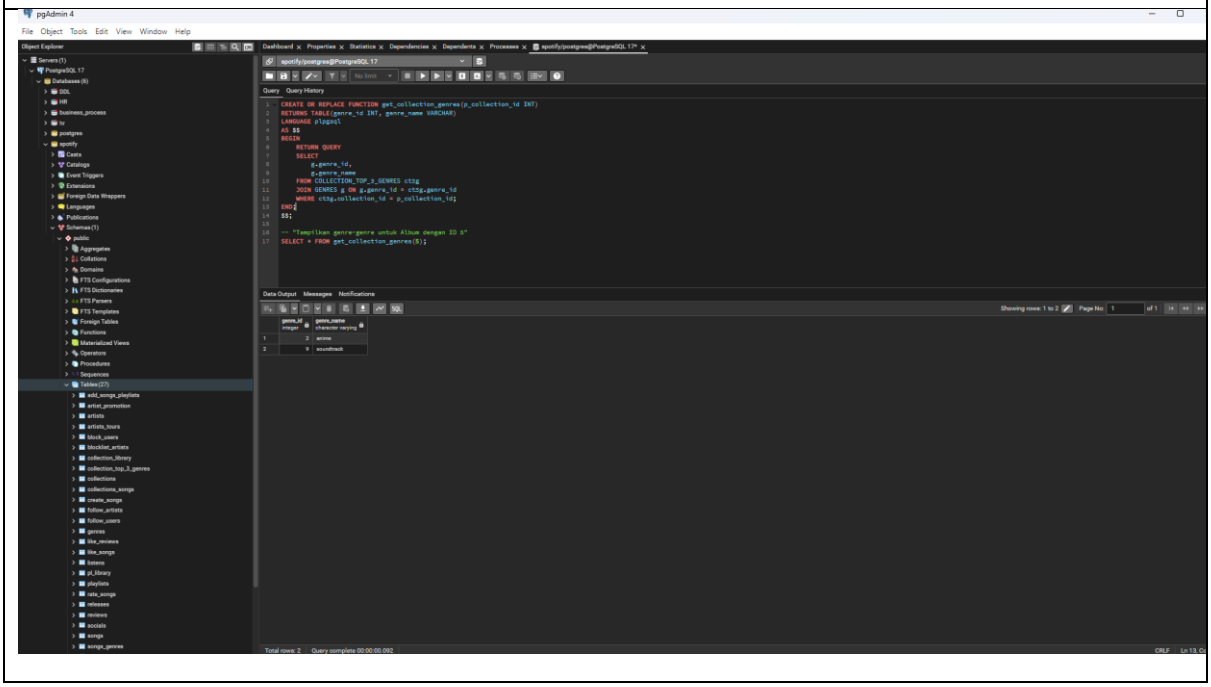
Tabel 108 PL/SQL - Function get_collection_genres

Nama Function	get_collection_genres
Parameter	p_collection_id INT
Deskripsi	Function ini bertujuan untuk mengambil daftar genre utama

Script SQL

```
CREATE OR REPLACE FUNCTION get_collection_genres(p_collection_id INT)
RETURNS TABLE(genre_id INT, genre_name VARCHAR)
LANGUAGE plpgsql
AS $$
BEGIN
   RETURN QUERY
   SELECT
      g.genre_id,
      g.genre_name
   FROM COLLECTION_TOP_3_GENRES ct3g
   JOIN GENRES g ON g.genre_id = ct3g.genre_id
   WHERE ct3g.collection_id = p_collection_id;
END;
$$;
```

Screenshot Hasil



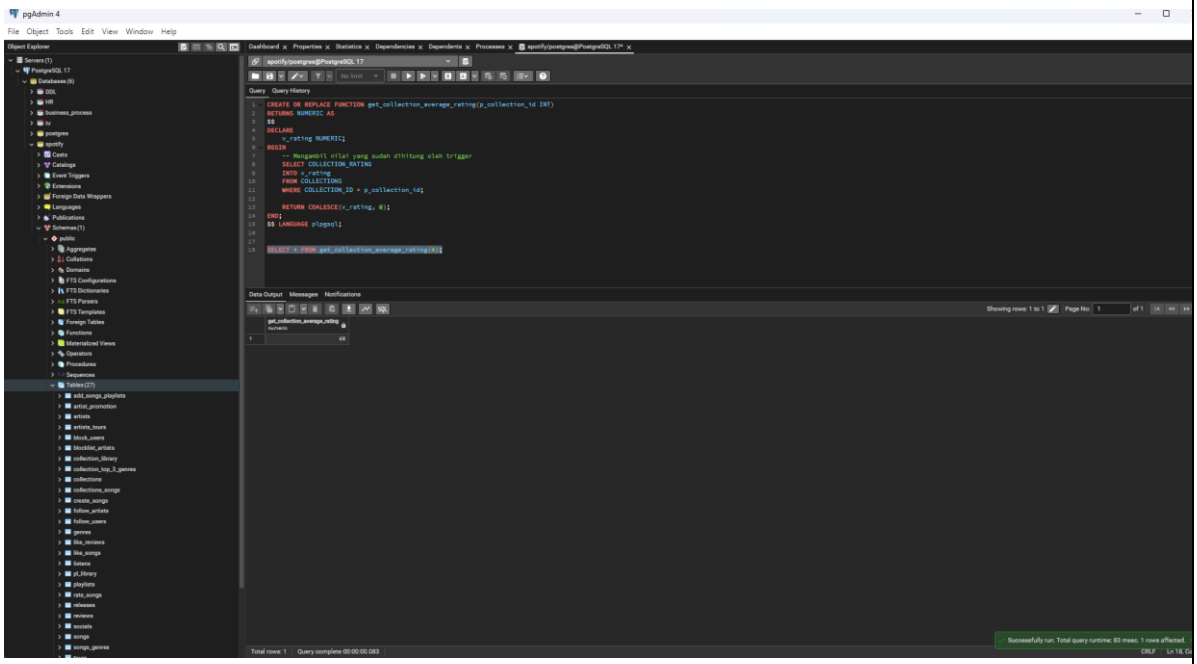
Tabel 109 PL/SQL - Function get_collection_average_rating

Nama Function	get_collection_average_rating
Parameter	p_collection_id INT
Deskripsi	Function ini berfungsi untuk mengambil hasil average rating dari trigger

Script SQL

```
CREATE OR REPLACE FUNCTION get_collection_average_rating(p_collection_id INT)
RETURNS NUMERIC AS
$$
DECLARE
   v_rating NUMERIC;
BEGIN
   -- Mengambil nilai yang sudah dihitung oleh trigger
   SELECT COLLECTION_RATING
   INTO v_rating
   FROM COLLECTIONS
   WHERE COLLECTION_ID = p_collection_id;

   RETURN COALESCE(v_rating, 0);
END;
$$ LANGUAGE plpgsql;
```

Screenshot Hasil



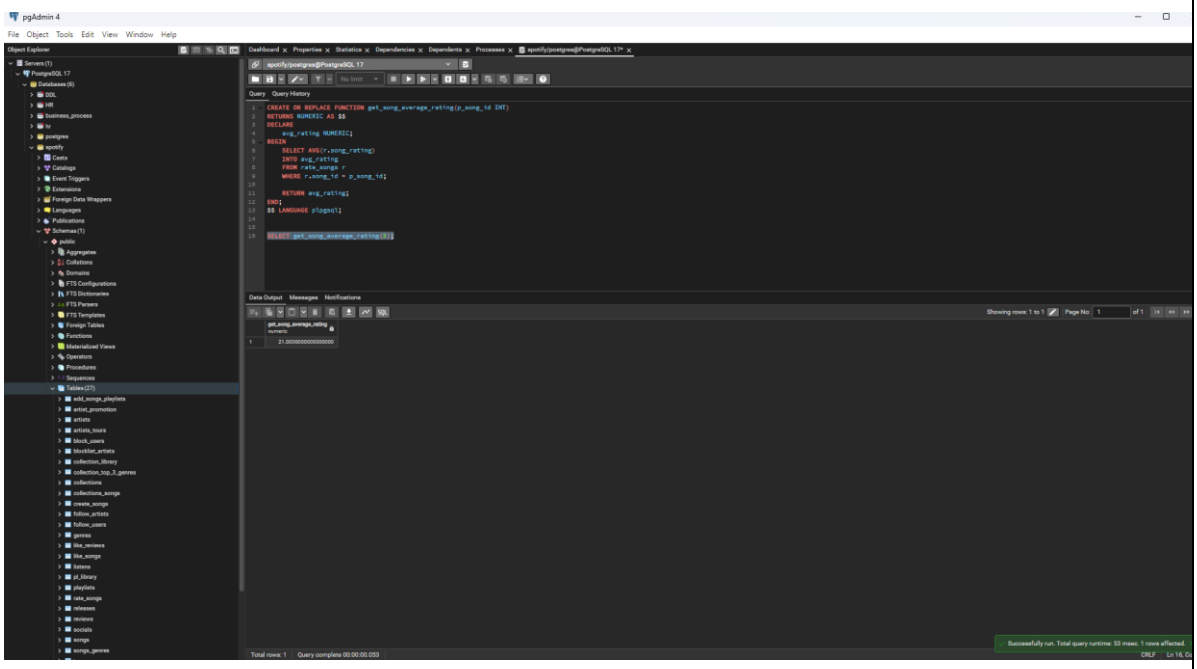
Tabel 110 PL/SQL - Function get_song_average_rating

Nama Function	get_song_average_rating(p_song_id INT)
Parameter	p_song_id INT
Deskripsi	Function ini berfungsi untuk menghitung rata rata song dari rate_songs

Script SQL

```
CREATE OR REPLACE FUNCTION get_song_average_rating(p_song_id INT)
RETURNS NUMERIC AS $$
DECLARE
   avg_rating NUMERIC;
BEGIN
   SELECT AVG(r.song_rating)
   INTO avg_rating
   FROM rate_songs r
   WHERE r.song_id = p_song_id;

   RETURN avg_rating;
END;
$$ LANGUAGE plpgsql;
```

Screenshot Hasil



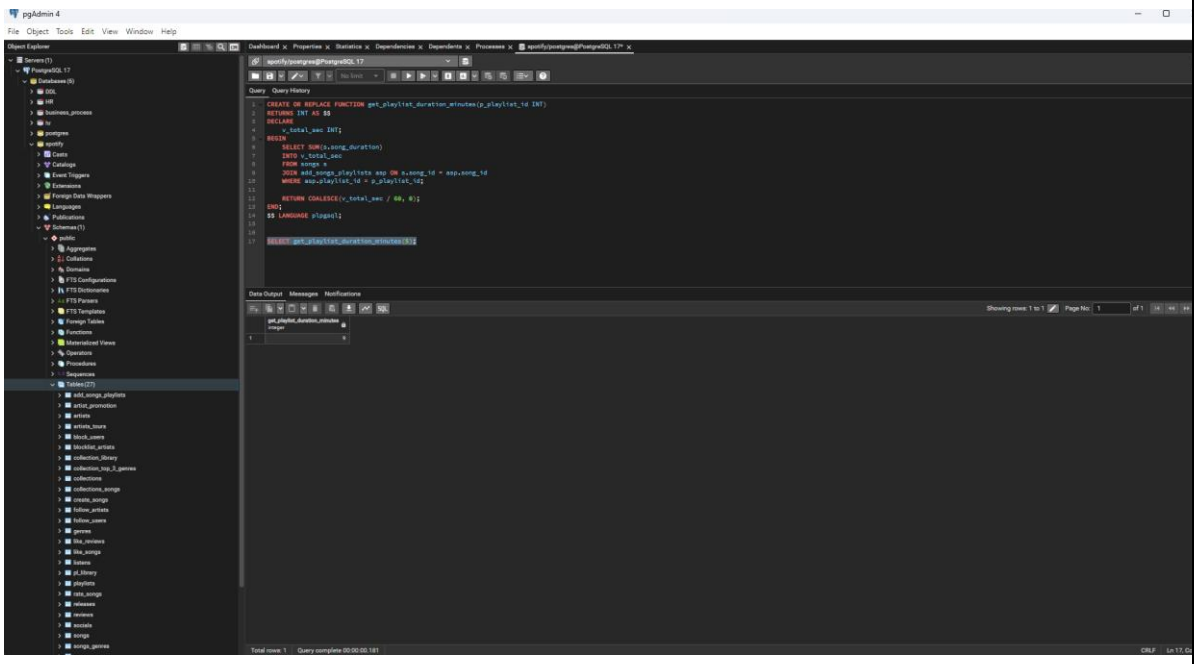
Tabel 111 PL/SQL - Function get_playlist_duration_minutes

Nama Function	get_playlist_duration_minutes(p_playlist_id INT)
Parameter	p_playlist_id INT
Deskripsi	Function ini berfungsi untuk menjumlahkan durasi seluruh lagu dalam satu playlist

Script SQL

```
CREATE OR REPLACE FUNCTION get_playlist_duration_minutes(p_playlist_id INT)
RETURNS INT AS $$
DECLARE
   v_total_sec INT;
BEGIN
   SELECT SUM(s.song_duration)
   INTO v_total_sec
   FROM songs s
   JOIN add_songs_playlists asp ON s.song_id = asp.song_id
   WHERE asp.playlist_id = p_playlist_id;

   RETURN COALESCE(v_total_sec / 60, 0);
END;
$$ LANGUAGE plpgsql;
```

Screenshot Hasil



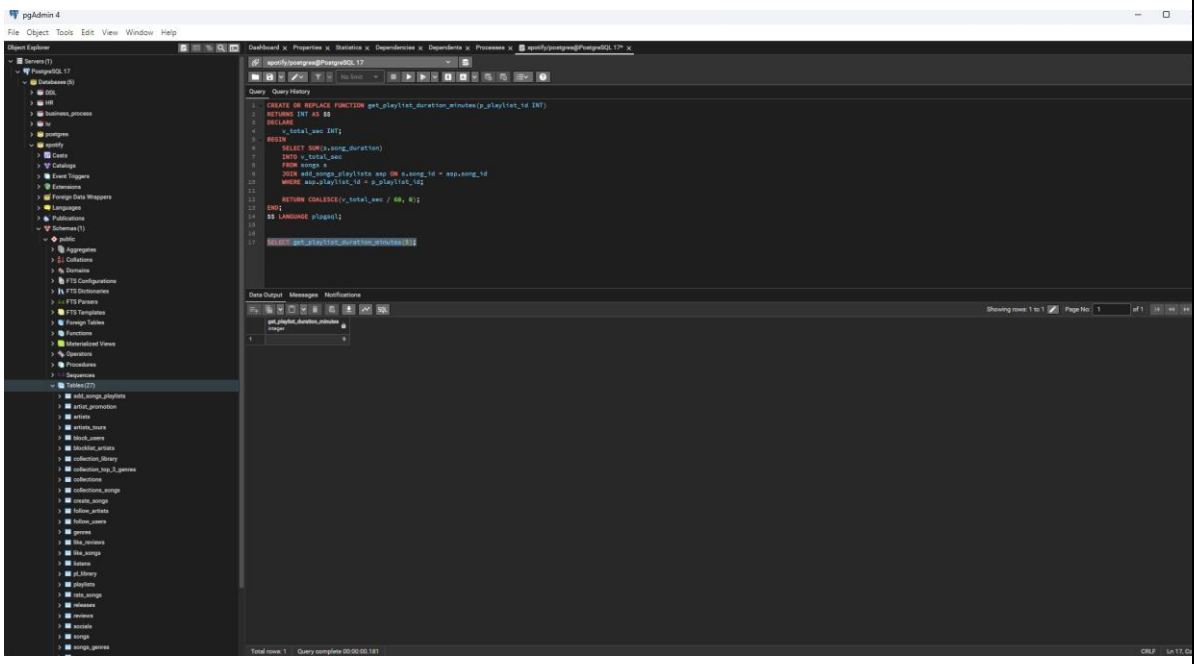
Tabel 112 PL/SQL - Function get_artist_total_plays

Nama Function	get_artist_total_plays(p_artist_id INT)
Parameter	p_artist_id INT
Deskripsi	Function ini berfungsi untuk menghitung total jumlah stream suatu artists

Script SQL

```
CREATE OR REPLACE FUNCTION get_artist_total_plays(p_artist_id INT)
RETURNS BIGINT AS $$
DECLARE
   v_total_plays BIGINT;
BEGIN
   SELECT COUNT(l.listen_id)
   INTO v_total_plays
   FROM create_songs cs
   JOIN listens l ON cs.song_id = l.song_id
   WHERE cs.artist_id = p_artist_id;

   RETURN COALESCE(v_total_plays, 0);
END;
$$ LANGUAGE plpgsql;
```

Screenshot Hasil



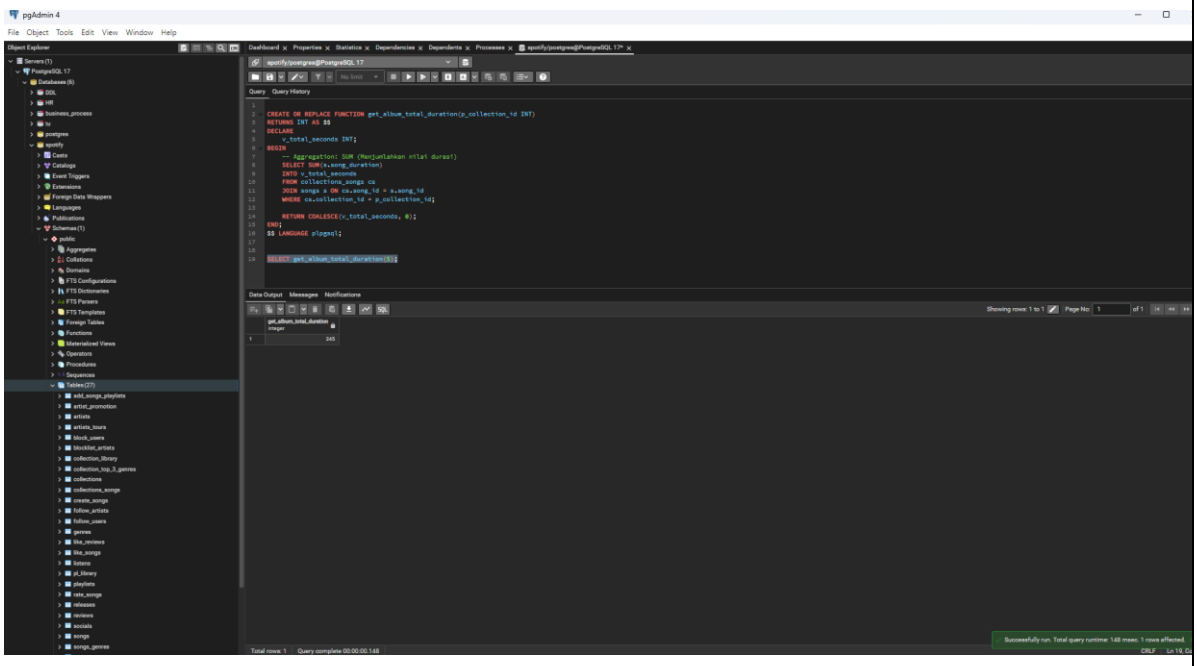
Tabel 113 PL/SQL - Function get_album_total_duration

Nama Function	get_album_total_duration(p_collection_id INT)
Parameter	p_collection_id INT
Deskripsi	Function ini berfungsi untuk menghitung durasi dari suatu album

Script SQL

```
CREATE OR REPLACE FUNCTION get_album_total_duration(p_collection_id INT)
RETURNS INT AS $$
DECLARE
   v_total_seconds INT;
BEGIN
   -- Aggregation: SUM (Menjumlahkan nilai durasi)
   SELECT SUM(s.song_duration)
   INTO v_total_seconds
   FROM collections_songs cs
   JOIN songs s ON cs.song_id = s.song_id
   WHERE cs.collection_id = p_collection_id;

   RETURN COALESCE(v_total_seconds, 0);
END;
$$ LANGUAGE plpgsql;
```

Screenshot Hasil



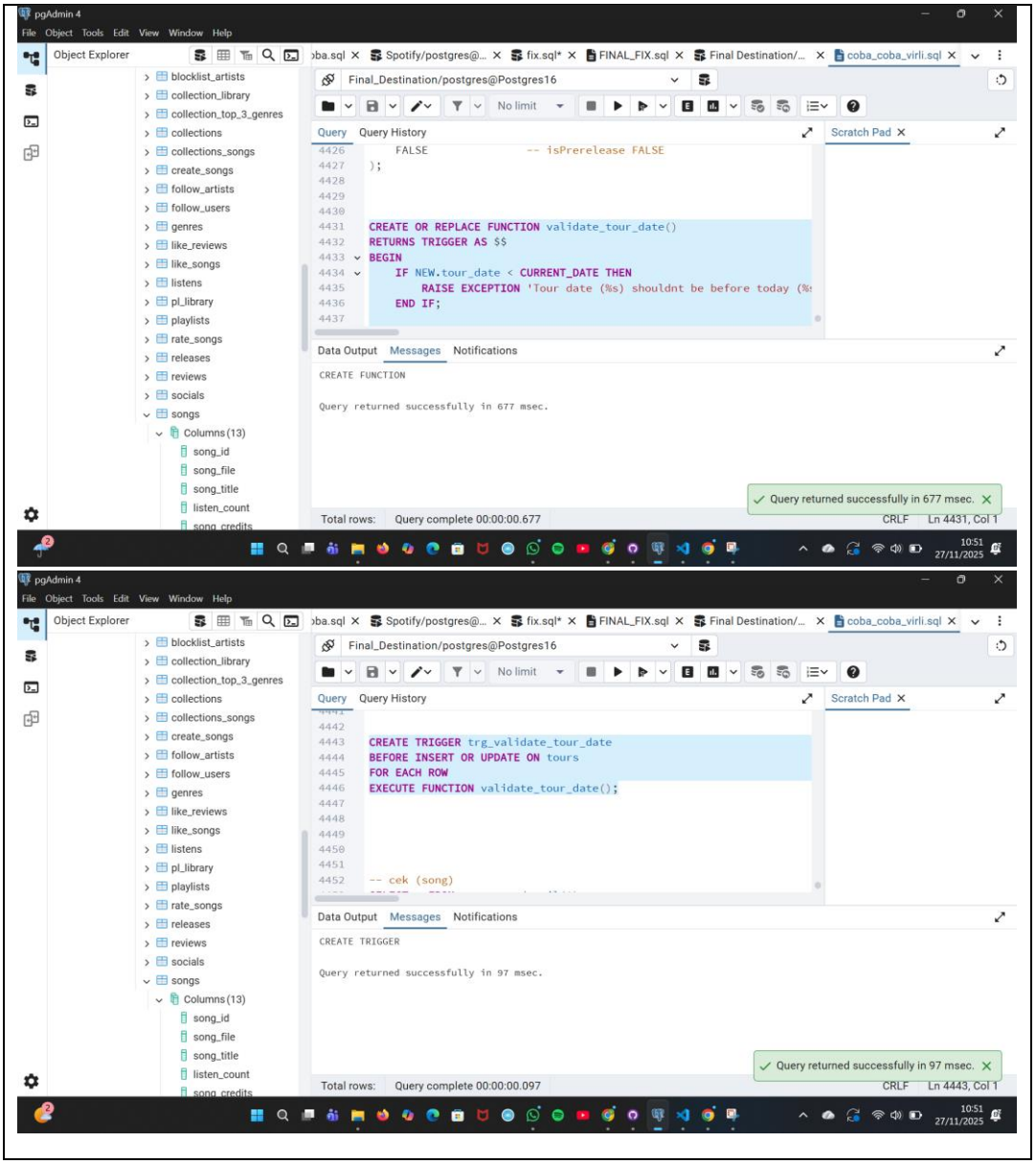
Tabel 113 PL/SQL - Trigger validate_tour_date

Nama Trigger	validate_tour_date
Aksi	Melakukan validasi otomatis terhadap kolom tour_date setiap kali data pada tabel tours ditambahkan atau diperbarui. Trigger ini memastikan bahwa tanggal tour tidak boleh berada di masa lalu. Jika tanggal yang dimasukkan lebih kecil dari tanggal hari ini (CURRENT_DATE), proses akan dibatalkan dan sistem melempar error.
Jenis	BEFORE INSERT OR UPDATE
Deskripsi	Trigger ini digunakan untuk menjamin bahwa setiap jadwal tour (konser, event, atau show) hanya dapat dibuat dengan tanggal yang valid, yaitu hari ini atau masa depan. Setiap ada operasi INSERT atau UPDATE pada tabel tours, trigger akan memanggil function validate_tour_date(), yang kemudian melakukan pengecekan: 1. Jika tour_date < CURRENT_DATE, maka fungsi akan mengeluarkan exception yang menghentikan proses insert/update. 2. Jika tanggal valid, baris data akan dilanjutkan untuk diproses. Trigger ini memastikan integritas data sehingga tidak ada tour yang terdaftar dengan tanggal kedaluwarsa.

Script SQL

```
CREATE OR REPLACE FUNCTION validate_tour_date()
RETURNS TRIGGER AS $$
BEGIN
IF NEW.tour_date < CURRENT_DATE THEN
RAISE EXCEPTION 'Tour date (%s) shouldnt be before today
(%s)', NEW.tour_date, CURRENT_DATE;
END IF;

RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER trg_validate_tour_date
BEFORE INSERT OR UPDATE ON tours
FOR EACH ROW
EXECUTE FUNCTION validate_tour_date();
```

Screenshot Hasil



Tabel 115 PL/SQL - Trigger trg_update_collection_top3_genres

Nama Trigger	trg_update_collection_top3_genres
Aksi	Update otomatis tabel collection_top_3_genres saat ada perubahan atau penambahan lagu di collections_songs.
Jenis	AFTER INSERT OR UPDATE

Deskripsi	Trigger ini akan secara otomatis menghitung 3 genre teratas yang paling sering muncul dari seluruh lagu dalam sebuah collection setiap kali ada lagu baru ditambahkan atau update di tabel collections_songs. Langkah kerjanya: 1. Hapus data lama untuk collection terkait di tabel collection_top_3_genres. 2. Hitung jumlah kemunculan tiap genre dari lagu-lagu collection menggunakan tabel songs_genres. 3. Ambil 3 genre terbanyak dan masukkan ke tabel collection_top_3_genres. 4. Trigger dijalankan per baris (row-level), sehingga setiap lagu yang diinsert/update langsung memicu perhitungan ulang. Contoh Perhitungan : Misalkan sebuah collection memiliki 4 lagu dengan genre sebagai berikut: Lagu 1: Pop, Jazz Lagu 2: Pop, Rock, Noise Pop Lagu 3: Pop, Rock Lagu 4: Pop, Jazz Maka jumlah kemunculan genre: Pop = 4 Jazz = 2 Rock = 2

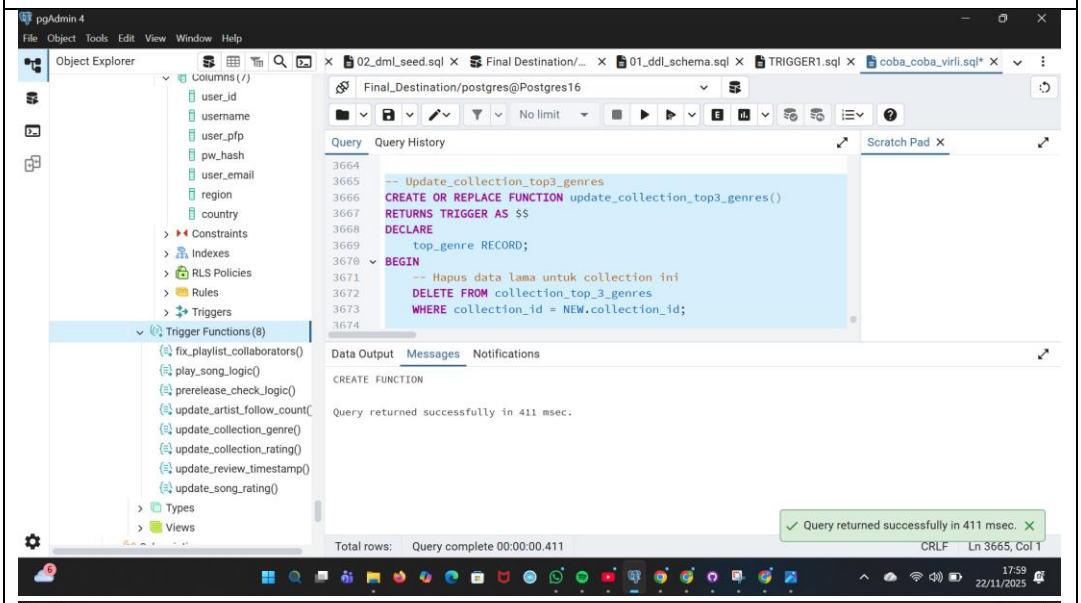
	Noise Pop = 1 Top 3 genre (Pop, Jazz, Rock) akan di-insert ke collection_top_3_genres.

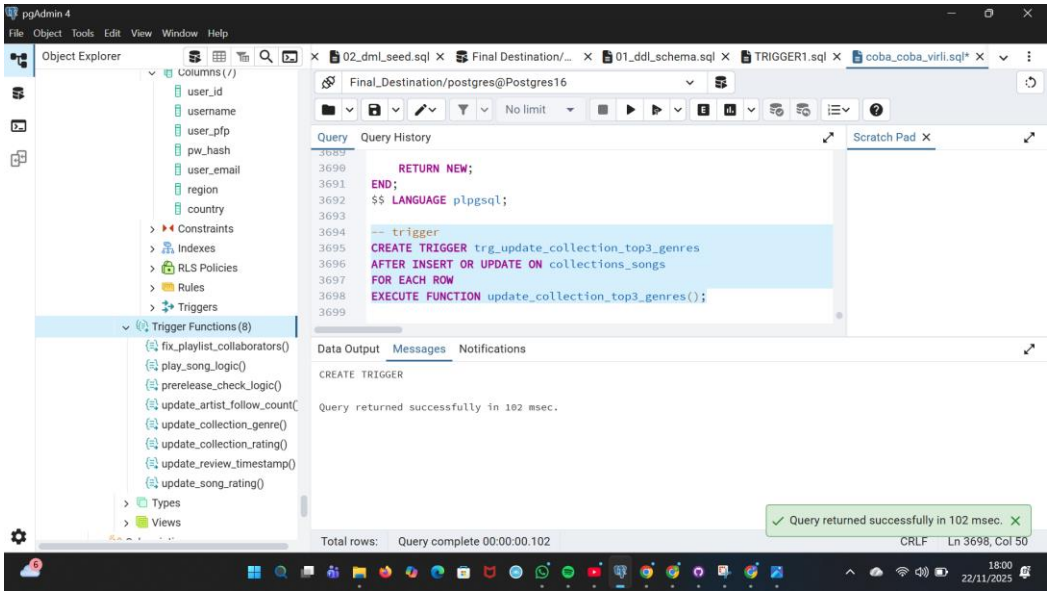
Script SQL

```
CREATE OR REPLACE FUNCTION update_collection_top3_genres()
RETURNS TRIGGER AS $$
DECLARE
    top_genre RECORD;
BEGIN
    -- Hapus data lama untuk collection ini
    DELETE FROM collection_top_3_genres
    WHERE collection_id = NEW.collection_id;

    -- Ambil 3 genre paling sering muncul dari lagu-lagu collection
    FOR top_genre IN
        SELECT sg.genre_id
        FROM songs_genres sg
        JOIN collections_songs cs ON sg.song_id = cs.song_id  -- Menghubungkan genre dengan lagu yang masuk collection
        WHERE cs.collection_id = NEW.collection_id
        GROUP BY sg.genre_id  -- Kelompokkan berdasarkan genre
        ORDER BY COUNT(*) DESC -- Mengurutkan genre paling sering muncul -> paling banyak lagu
        LIMIT 3 -- Ambil 3 teratas
    LOOP
        -- Insert ke COLLECTION_TOP_3_GENRES
```

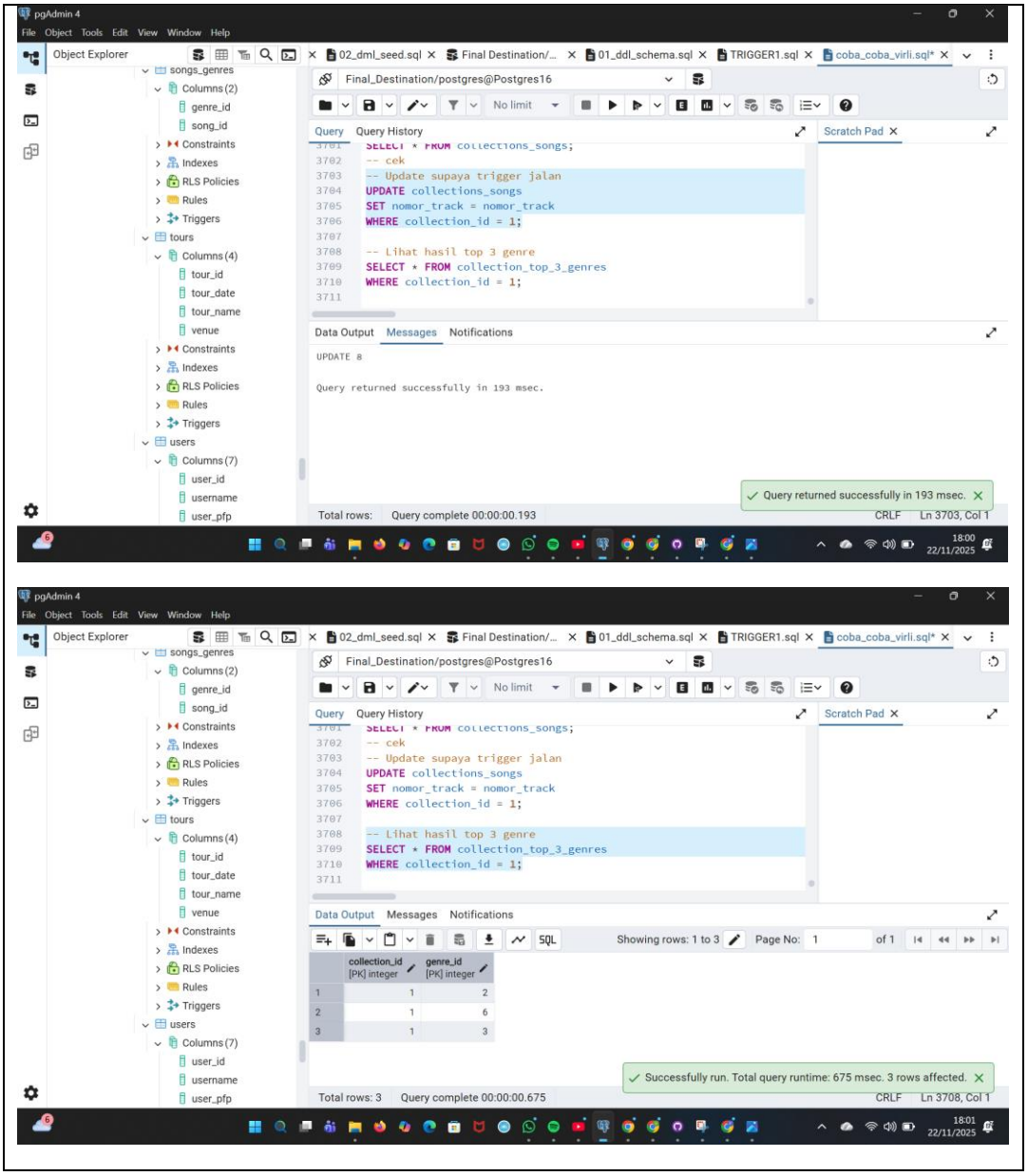
```
        INSERT INTO collection_top_3_genres (collection_id,
genre_id)
        VALUES (NEW.collection_id, top_genre.genre_id);
    END LOOP;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Screenshot Hasil





-- update



Tabel 116 PL/SQL - Trigger trg_update_song_rating

Nama Trigger	Trg_update_song_rating
Aksi	Update otomatis kolom SONG_RATING saat ada perubahan rating lagu.
Jenis	AFTER INSERT OR UPDATE OR DELETE

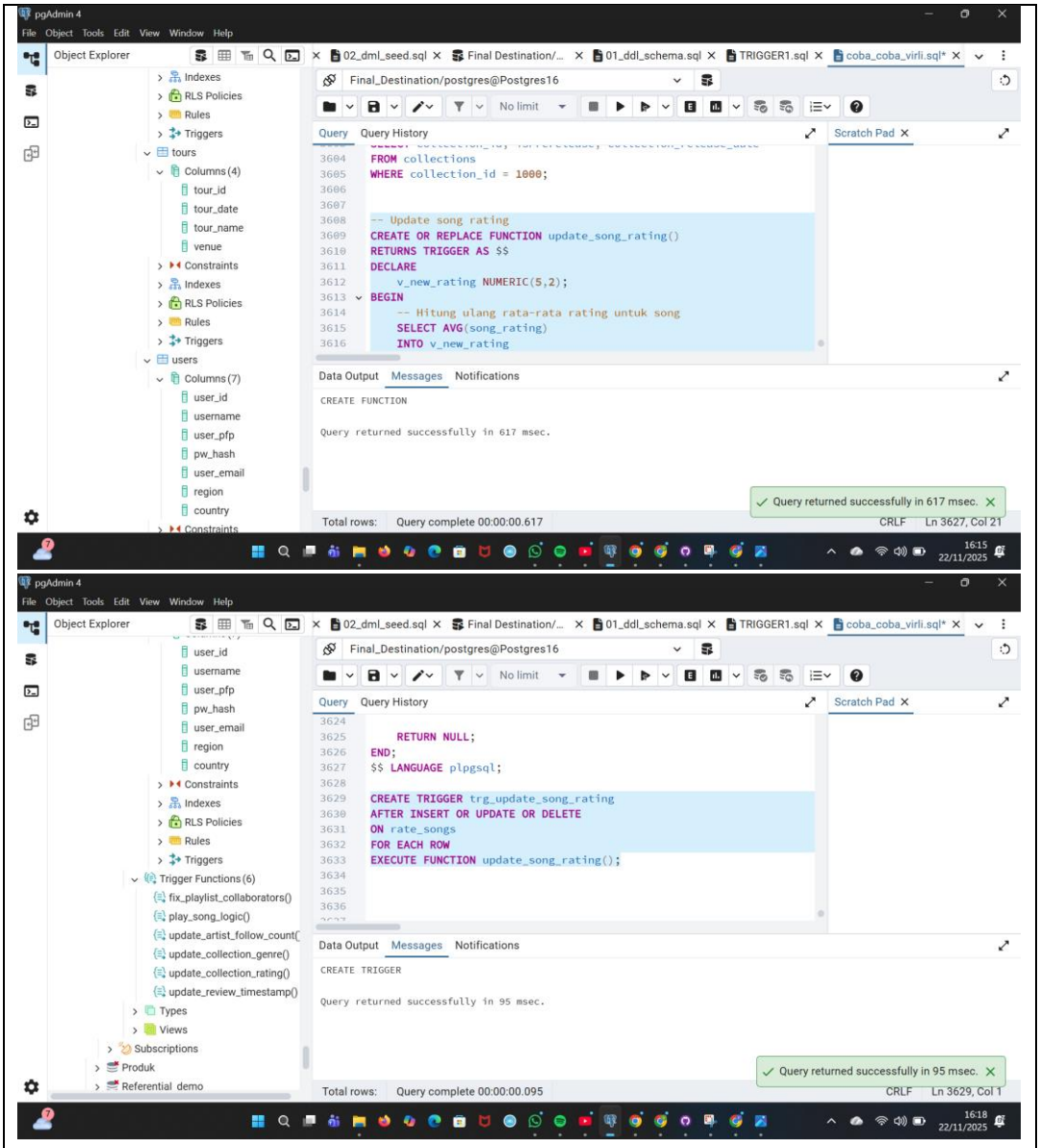
Deskripsi	Trigger ini dijalankan setiap kali ada perubahan di tabel RATE_SONGS, baik itu INSERT, UPDATE, maupun DELETE. Fungsi utamanya adalah menghitung ulang rata-rata rating untuk sebuah lagu (song_id) dan menyimpannya di kolom SONGS.SONG_RATING. Logika perhitungan AVG: - Sistem akan mengambil semua rating yang ada untuk lagu tertentu (song_id). - Kemudian dihitung rata-ratanya menggunakan AVG(song_rating). - Hasilnya akan otomatis di-update ke song_rating. Dengan mekanisme ini, kolom SONG_RATING selalu mencerminkan rata-rata rating terkini dari semua user untuk lagu tersebut.

Script SQL

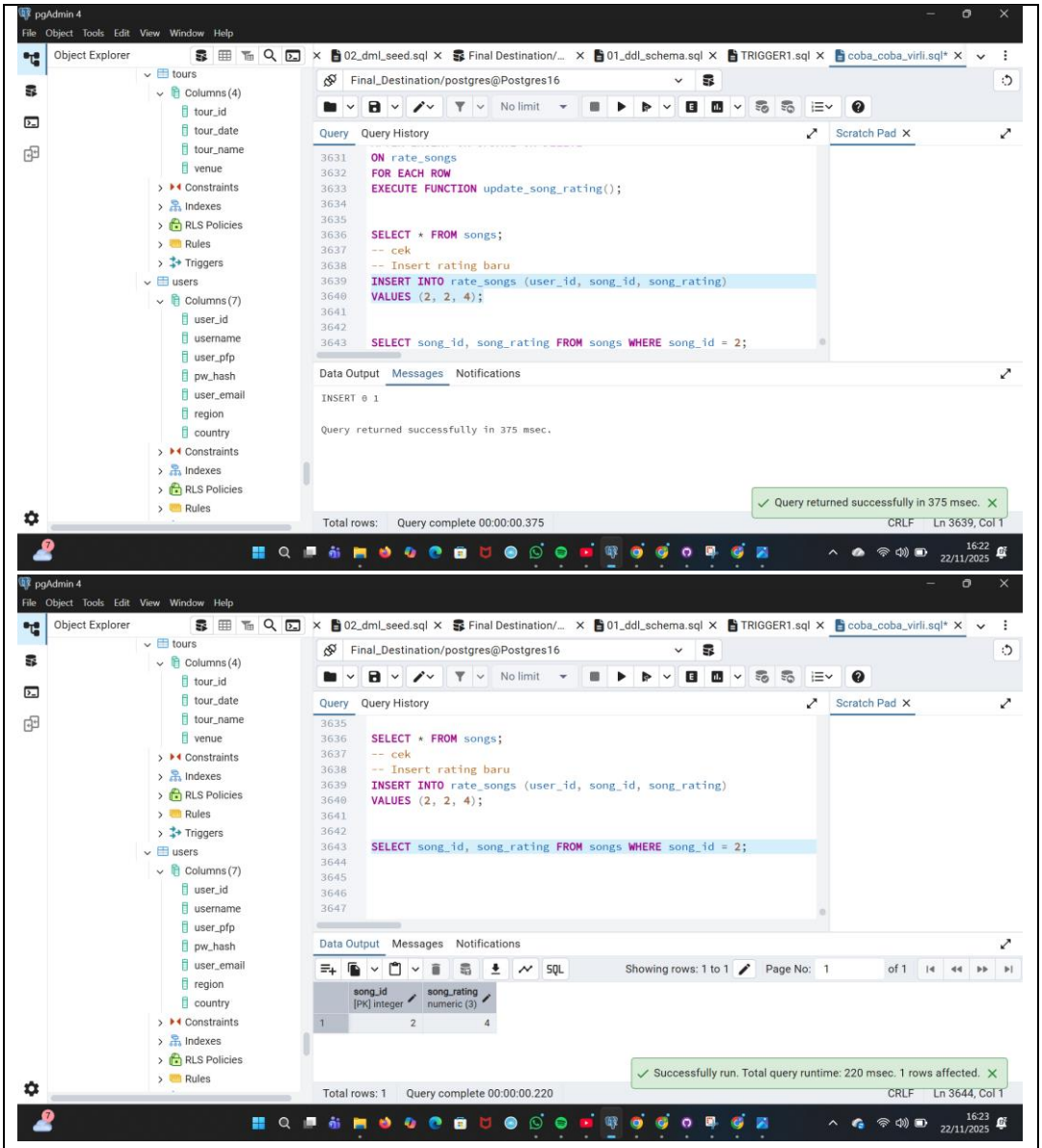
```
-- Update song rating
-- Fungsi ini digunakan untuk menghitung ulang rata-rata rating
sebuah lagu
-- setiap kali terjadi perubahan pada tabel rate_songs.
-- Trigger akan memanggil fungsi ini saat INSERT, UPDATE, atau
DELETE.
CREATE OR REPLACE FUNCTION update_song_rating()
RETURNS TRIGGER AS $$
DECLARE
    v_new_rating NUMERIC(5,2);  -- Variabel untuk menyimpan rata-
rata rating terbaru
BEGIN
    -- Hitung ulang rata-rata rating untuk song
        --    NEW.song_id digunakan saat INSERT/UPDATE.
        --    OLD.song_id digunakan saat DELETE.
        --    COALESCE memastikan salah satu pasti terpakai.
    SELECT AVG(song_rating)
    INTO v_new_rating
    FROM rate_songs
    WHERE song_id = COALESCE(NEW.song_id, OLD.song_id);
```

```
    -- Update nilai rata-rata rating ke tabel songs.
    -- Jika semua rating dihapus, AVG akan NULL → lagu tidak punya
rating.
    UPDATE songs
    SET song_rating = v_new_rating
    WHERE song_id = COALESCE(NEW.song_id, OLD.song_id);

    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

-- Trigger
CREATE TRIGGER trg_update_song_rating
AFTER INSERT OR UPDATE OR DELETE
ON rate_songs
FOR EACH ROW
EXECUTE FUNCTION update_song_rating();
```

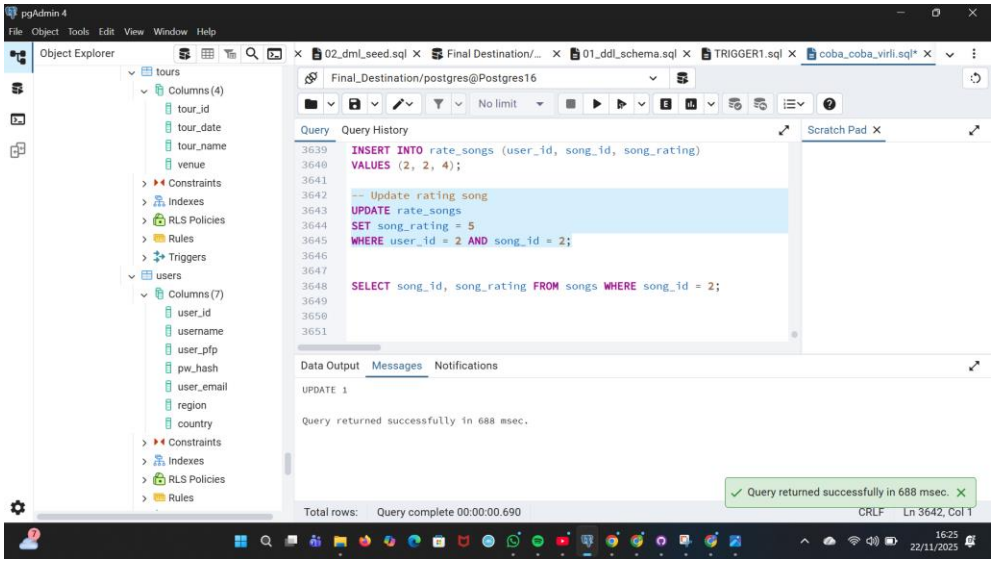
Screenshot Hasil

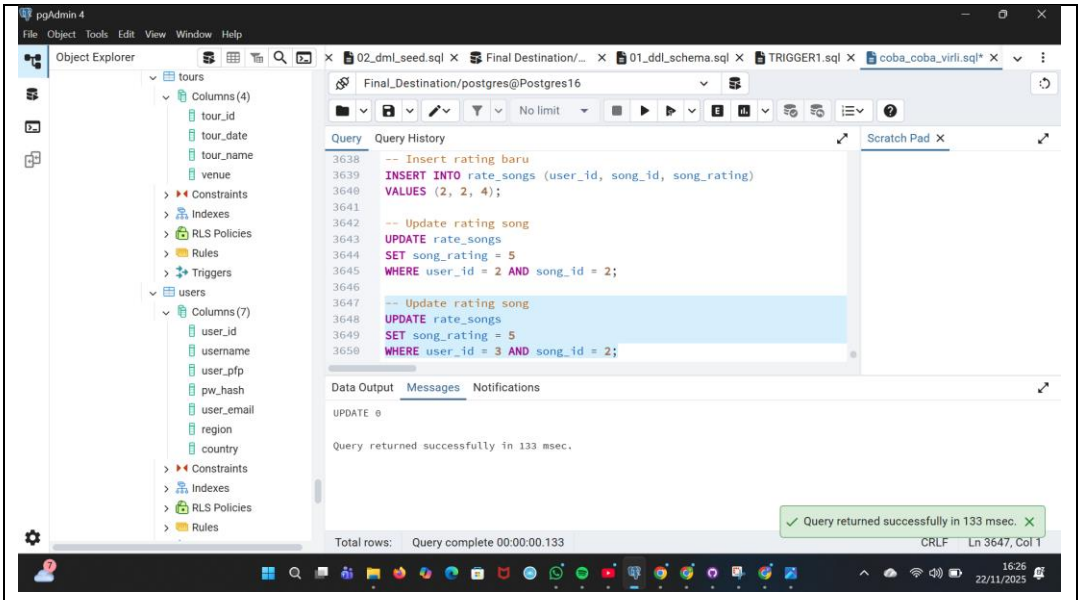


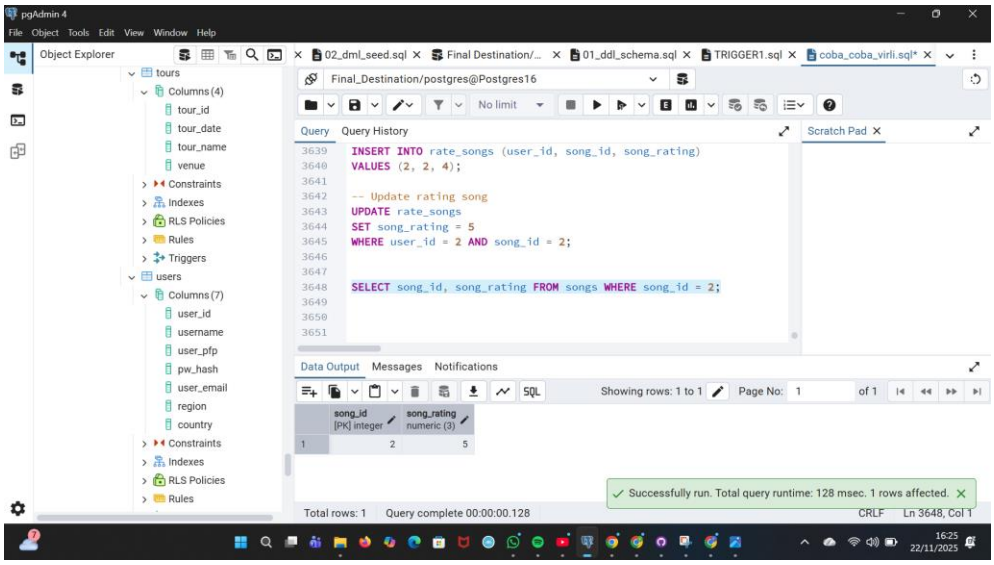
-- insert rating baru



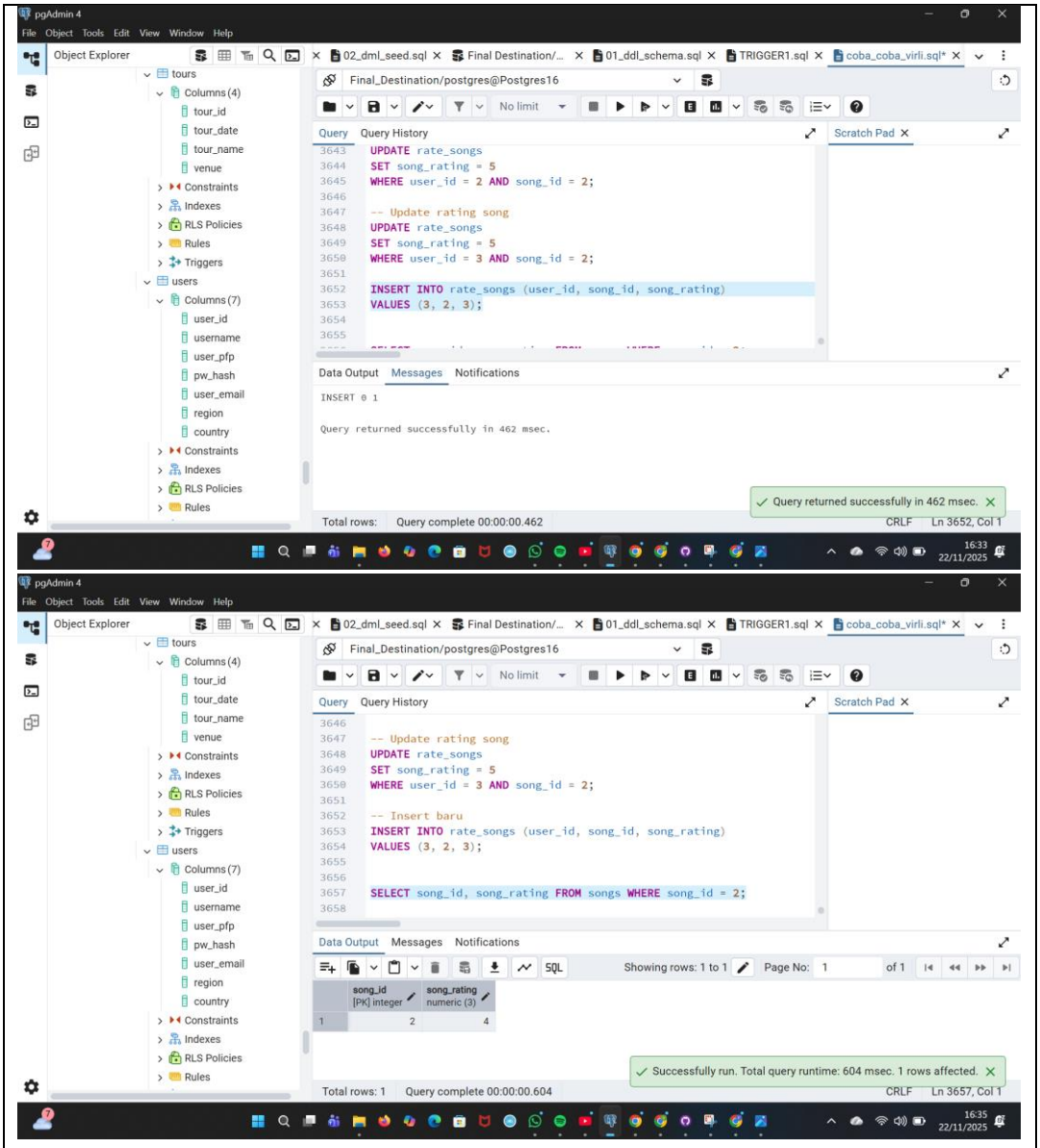
-- update rating song




pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
02_dml_seed.sql
Final Destination/...
01_ddl_schema.sql
TRIGGER1.sql
coba_coba_virli.sql*
Final_Destination/postgres@Postgres16
No limit
Query
Query History
Scratch Pad
-- Insert rating baru
INSERT INTO rate_songs (user_id, song_id, song_rating)
VALUES (2, 2, 4);
-- Update rating song
UPDATE rate_songs
SET song_rating = 5
WHERE user_id = 2 AND song_id = 2;
-- Update rating song
UPDATE rate_songs
SET song_rating = 5
WHERE user_id = 3 AND song_id = 2;
Data Output
Messages
Notifications
UPDATE 0
Query returned successfully in 133 msec.
Query returned successfully in 133 msec.
Total rows:
Query complete 00:00:00.133
CRLF
Ln 3647, Col 1
16:26
22/11/2025

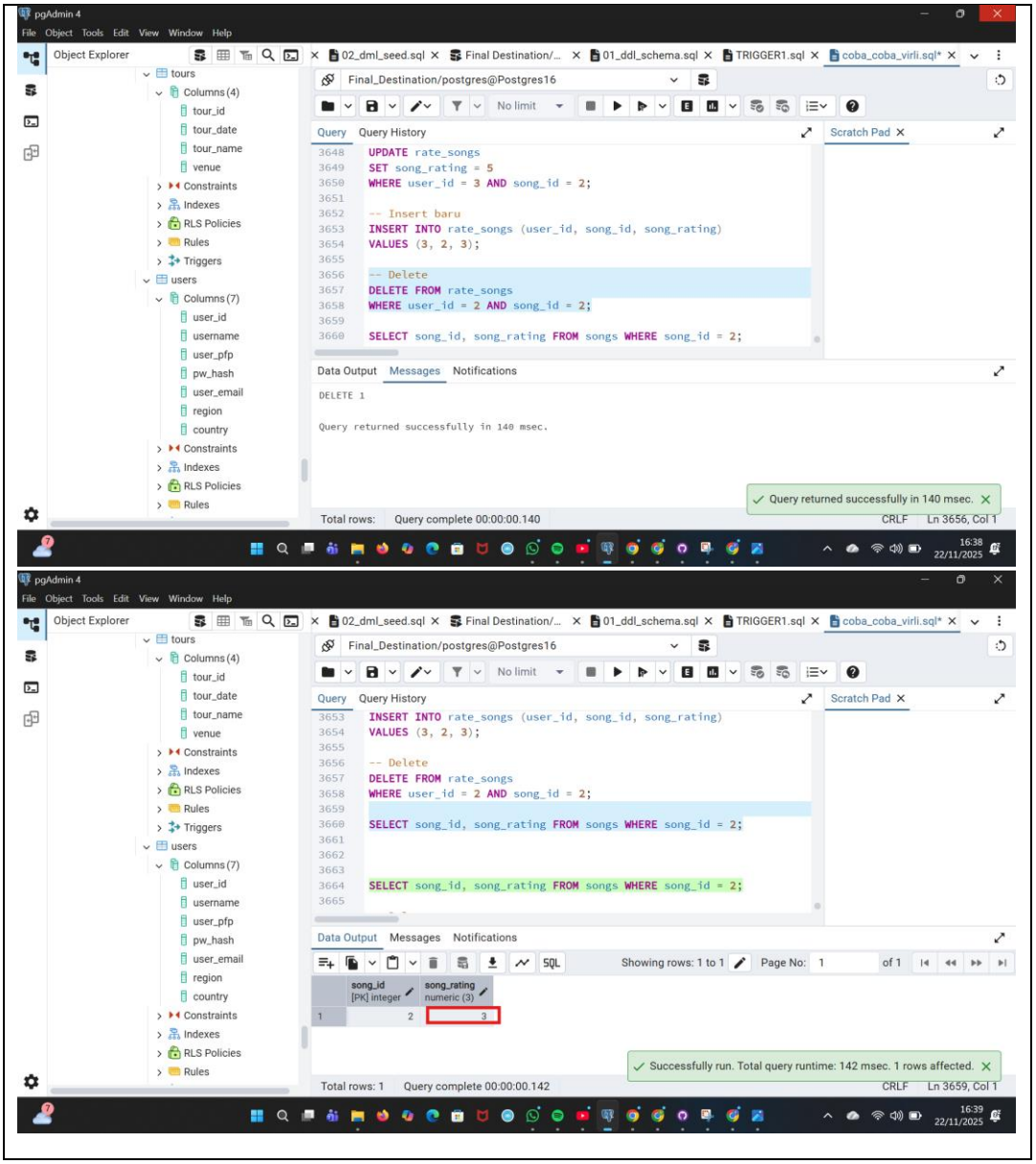

pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
02_dml_seed.sql
Final Destination/...
01_ddl_schema.sql
TRIGGER1.sql
coba_coba_virli.sql*
Final_Destination/postgres@Postgres16
No limit
Query
Query History
Scratch Pad
INSERT INTO rate_songs (user_id, song_id, song_rating)
VALUES (2, 2, 4);
-- Update rating song
UPDATE rate_songs
SET song_rating = 5
WHERE user_id = 2 AND song_id = 2;
SELECT song_id, song_rating FROM songs WHERE song_id = 2;
Data Output
Messages
Notifications
Showing rows: 1 to 1
Page No: 1
of 1
song_id
[PK] integer
song_rating
numeric (3)
1
2
5
Successfully run. Total query runtime: 128 msec. 1 rows affected.
Total rows: 1
Query complete 00:00:00.128
CRLF
Ln 3648, Col 1
16:25
22/11/2025

-- insert baru


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
02_dml_seed.sql
Final Destination/...
01_ddl_schema.sql
TRIGGER1.sql
coba_coba_virli.sql*
Final_Destination/postgres@Postgres16
No limit
Query
Query History
Scratch Pad
3643 UPDATE rate_songs
3644 SET song_rating = 5
3645 WHERE user_id = 2 AND song_id = 2;
3646
3647 -- Update rating song
3648 UPDATE rate_songs
3649 SET song_rating = 5
3650 WHERE user_id = 3 AND song_id = 2;
3651
3652 INSERT INTO rate_songs (user_id, song_id, song_rating)
3653 VALUES (3, 2, 3);
Data Output
Messages
Notifications
INSERT 0 1
Query returned successfully in 462 msec.
Query returned successfully in 462 msec.
Total rows:
Query complete 00:00:00.462
CRLF
Ln 3652, Col 1
16:33
22/11/2025
tours
Columns (4)
tour_id
tour_date
tour_name
venue
Constraints
Indexes
RLS Policies
Rules
Triggers
users
Columns (7)
user_id
username
user_pfp
pw_hash
user_email
region
country

pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
02_dml_seed.sql
Final Destination/...
01_ddl_schema.sql
TRIGGER1.sql
coba_coba_virli.sql*
Final_Destination/postgres@Postgres16
No limit
Query
Query History
Scratch Pad
3646
3647 -- Update rating song
3648 UPDATE rate_songs
3649 SET song_rating = 5
3650 WHERE user_id = 3 AND song_id = 2;
3651
3652 -- Insert baru
3653 INSERT INTO rate_songs (user_id, song_id, song_rating)
3654 VALUES (3, 2, 3);
3655
3656
3657 SELECT song_id, song_rating FROM songs WHERE song_id = 2;
3658
Data Output
Messages
Notifications
SQL
Showing rows: 1 to 1
Page No: 1
of 1
song_id
[PK] integer
song_rating
numeric (3)
1
2
4
Successfully run. Total query runtime: 604 msec. 1 rows affected.
Total rows: 1
Query complete 00:00:00.604
CRLF
Ln 3657, Col 1
16:35
22/11/2025
tours
Columns (4)
tour_id
tour_date
tour_name
venue
Constraints
Indexes
RLS Policies
Rules
Triggers
users
Columns (7)
user_id
username
user_pfp
pw_hash
user_email
region
country

-- Delete



Tabel 117 PL/SQL – Trigger trg_validate_prerelease_date

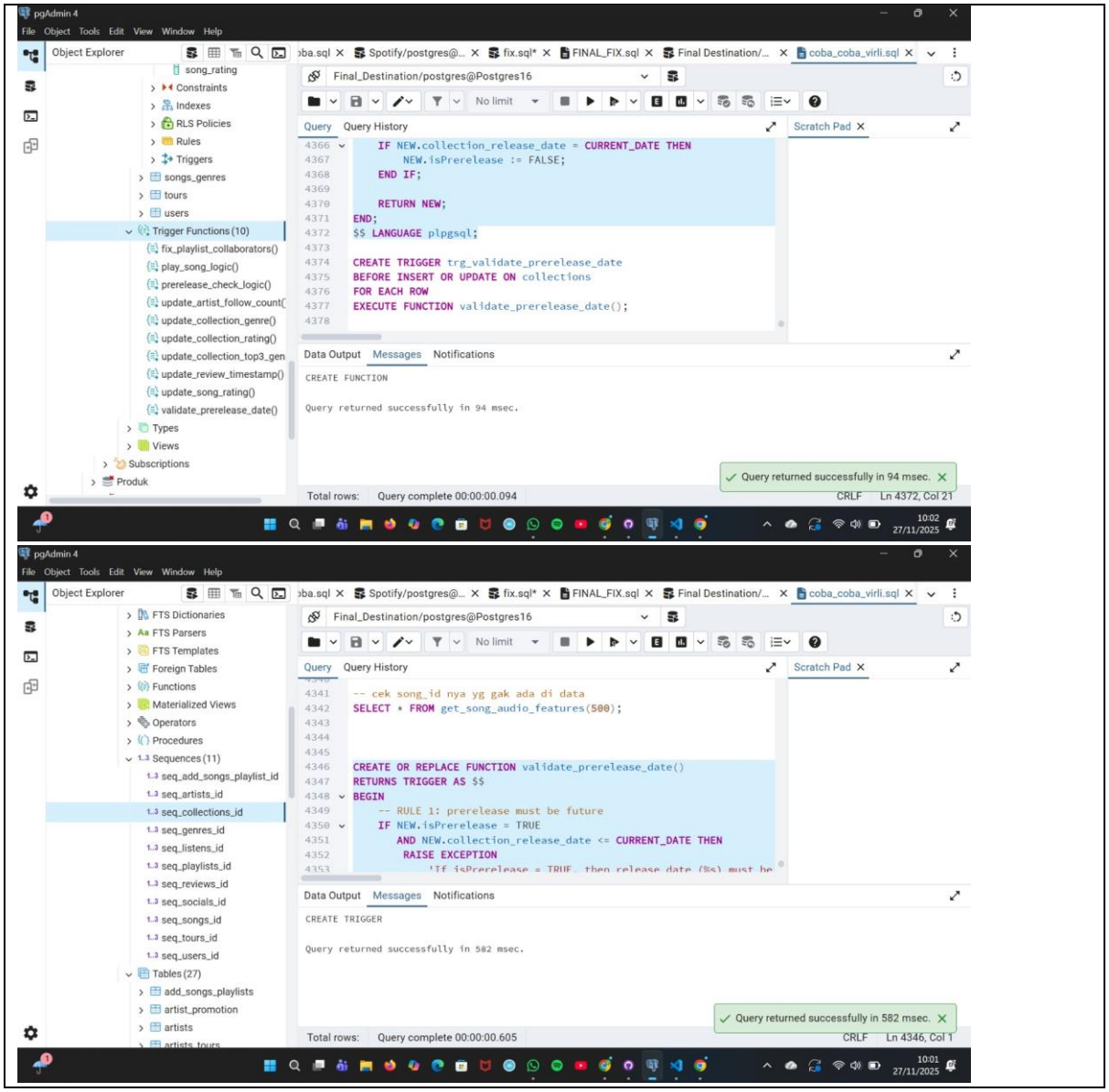
Nama Trigger	trg_validate_prerelease_date
Aksi	Melakukan validasi otomatis terhadap data koleksi sebelum proses INSERT atau UPDATE, meliputi: 1. Melarang koleksi berstatus prerelease (isPrerelease = TRUE) memiliki tanggal rilis yang tidak di masa depan. 2. Melarang koleksi yang bukan prerelease (isPrerelease = FALSE) memiliki tanggal rilis di masa depan.

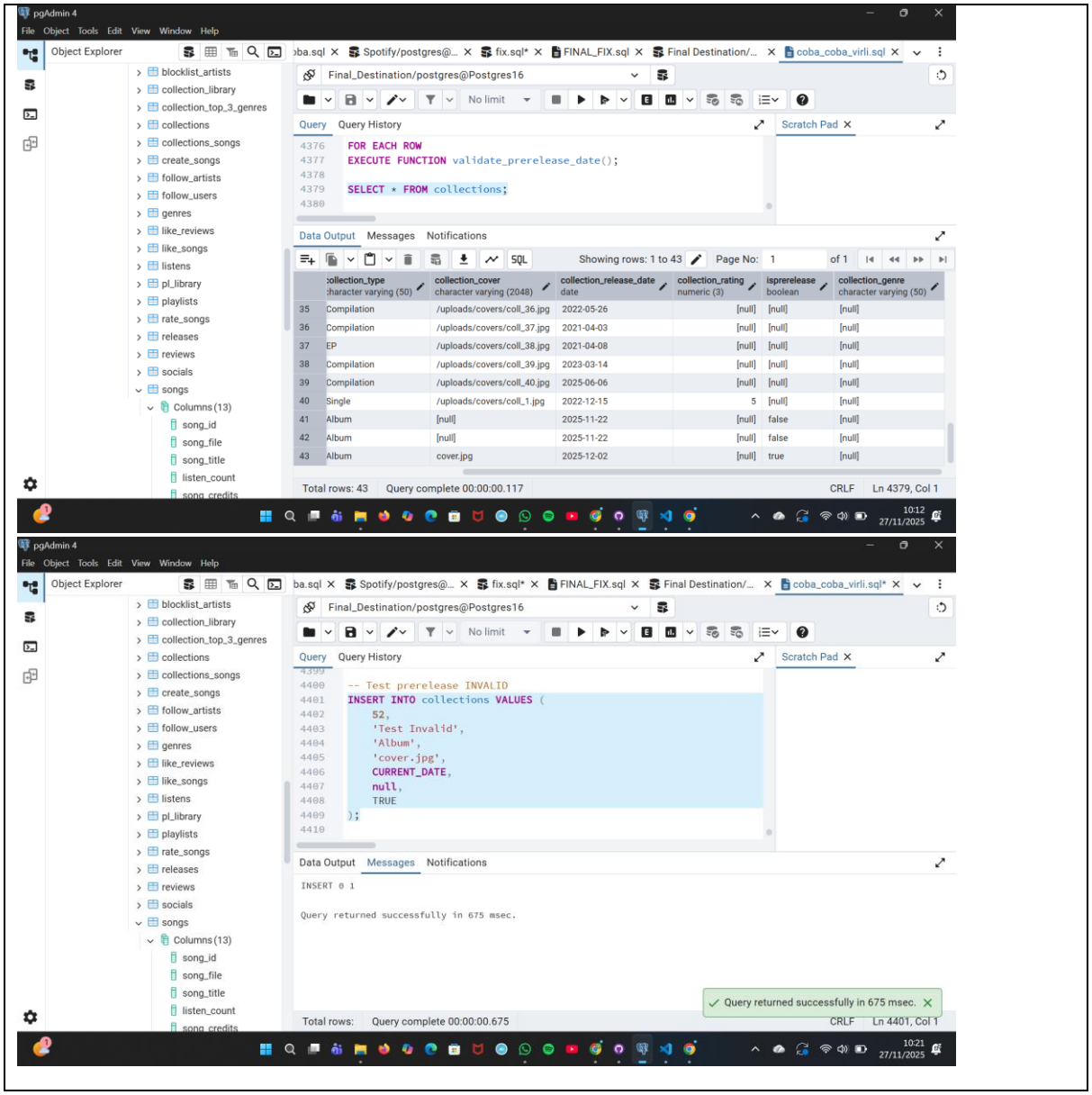
	3. Mengubah otomatis nilai isPrerelease menjadi FALSE apabila tanggal rilis koleksi adalah hari ini (CURRENT_DATE).
Jenis	BEFORE INSERT OR UPDATE
Deskripsi	Trigger trg_validate_prerelease_date digunakan untuk memastikan bahwa setiap data koleksi (album/playlist) pada tabel collections mengikuti aturan validasi terkait status prerelease dan tanggal rilis. Setiap kali entri koleksi ditambahkan atau diperbarui, trigger ini akan memanggil function validate_prerelease_date() untuk melakukan pengecekan: 1. Validasi Prerelease (isPrerelease = TRUE) Jika sebuah koleksi ditandai sebagai prerelease, maka tanggal rilisnya harus berada di masa depan. Jika tanggal rilis tidak di masa depan (lebih kecil atau sama dengan CURRENT_DATE), maka proses akan dibatalkan dengan ERROR. 2. Validasi Non-Prerelease (isPrerelease = FALSE) Jika koleksi bukan prerelease, maka tanggal rilisnya tidak boleh berada di masa depan. Jika tanggal rilis masih di masa depan, trigger akan menghentikan proses dengan ERROR. 3. Pengubahan Otomatis isPrerelease Apabila tanggal rilis koleksi sama dengan hari ini, trigger akan otomatis mengubah nilai isPrerelease menjadi FALSE. Ini memastikan sistem mencerminkan bahwa koleksi tersebut sudah rilis pada hari itu. Dengan mekanisme ini, trigger menjamin integritas data prerelease dan rilis sesuai aturan bisnis yang ditetapkan, sekaligus menghindari ketidaksesuaian status dan tanggal rilis di tabel collections.

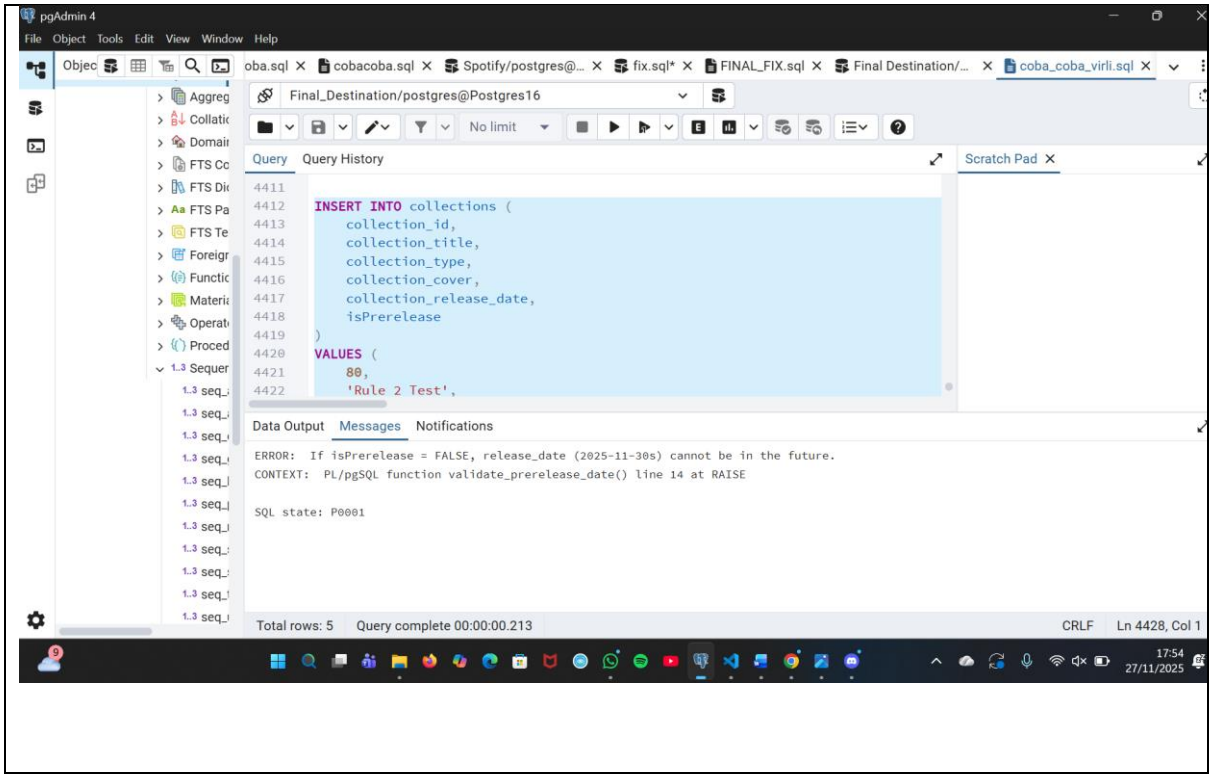
Script SQL

```
CREATE OR REPLACE FUNCTION validate_prerelease_date()
RETURNS TRIGGER AS $$
BEGIN
    -- RULE 1: prerelease must be future
    IF NEW.isPrerelease = TRUE  -- Jika isPrerelease = TRUE
       AND NEW.collection_release_date <= CURRENT_DATE THEN    -- Tetapi
tanggal rilis tidak di masa depan
        RAISE EXCEPTION
            'If isPrerelease = TRUE, then release_date (%s) must be in the
future.',  -- Pesan error
            NEW.collection_release_date;
    END IF;

    -- RULE 2: NOT prerelease must be today or past
    IF NEW.isPrerelease = FALSE  -- Jika isPrerelease = FALSE
       AND NEW.collection_release_date > CURRENT_DATE THEN -- Tetapi tanggal
rilis berada di masa depan
        RAISE EXCEPTION
            'If isPrerelease = FALSE, release_date (%s) cannot be in the
future.',  -- Pesan error
            NEW.collection_release_date;    -- Menampilkan tanggal yang salah
    END IF;

    -- RULE 3: If release date is today → force prerelease = FALSE
    IF NEW.collection_release_date = CURRENT_DATE THEN    -- Jika tanggal
rilis sama dengan hari ini
        NEW.isPrerelease := FALSE;  -- isPrerelease menjadi FALSE
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_validate_prerelease_date
BEFORE INSERT OR UPDATE ON collections
FOR EACH ROW
EXECUTE FUNCTION validate_prerelease_date();
```

Screenshot Hasil


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Final_Destination/postgres@Postgres16
Query Query History
Scratch Pad
Trigger Functions (10)
fix_playlist_collaborators()
play_song_logic()
prerelease_check_logic()
update_artist_follow_count()
update_collection_genre()
update_collection_rating()
update_collection_top3_gen
update_review_timestamp()
update_song_rating()
validate_prerelease_date()
CREATE FUNCTION
Query returned successfully in 94 msec.
Data Output Messages Notifications
Total rows: Query complete 00:00:00.094
CRLF Ln 4372, Col 21
Sequences (11)
seq_collections_id
CREATE TRIGGER
Query returned successfully in 582 msec.
Total rows: Query complete 00:00:00.605
CRLF Ln 4346, Col 1


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Final_Destination/postgres@Postgres16
Query Query History
Scratch Pad
FOR EACH ROW
EXECUTE FUNCTION validate_prerelease_date();
SELECT * FROM collections;
Data Output Messages Notifications
Total rows: 43 Query complete 00:00:00.117
CRLF Ln 4379, Col 1
-- Test prerelease INVALID
INSERT INTO collections VALUES (
52,
'Test Invalid',
'Album',
'cover.jpg',
CURRENT_DATE,
null,
TRUE
);
INSERT 0 1
Query returned successfully in 675 msec.
Total rows: Query complete 00:00:00.675
CRLF Ln 4401, Col 1



Tabel 118 PL/SQL - Trigger trg_update_review_timestamp

Nama Trigger	trg_update_review_timestamp
Aksi	Memperbarui kolom TIMESTAMP pada tabel reviews ketika nilai review atau rating diubah.
Jenis	BEFORE UPDATE

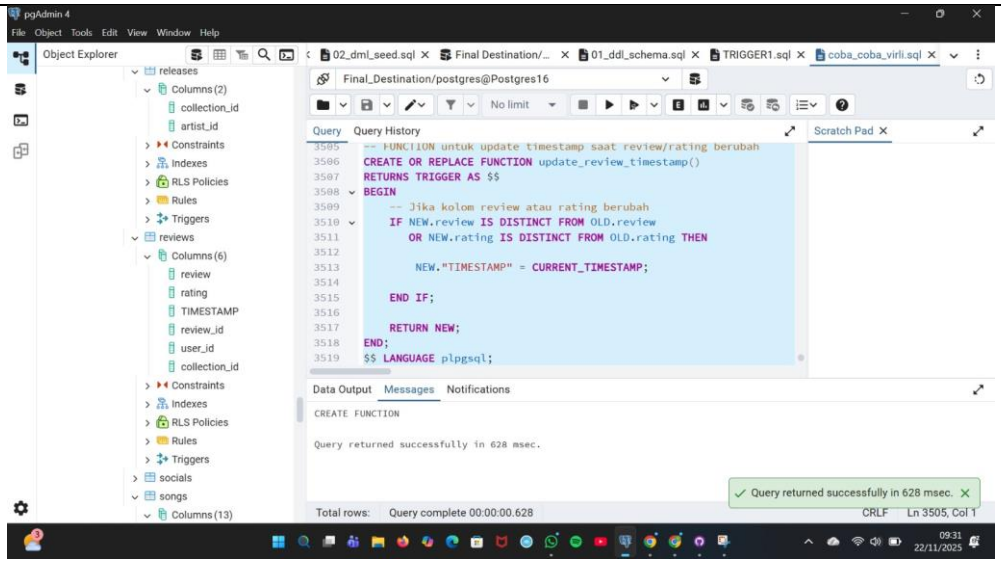
Deskripsi	Trigger ini digunakan untuk meng-update otomatis timestamp setiap kali user mengubah: 1. Isi review (review) 2. Nilai rating (rating) Jika salah satu dari dua kolom tersebut berubah, maka kolom "TIMESTAMP" akan diisi ulang dengan waktu saat perubahan terjadi (CURRENT_TIMESTAMP). Trigger memastikan data riwayat perubahan review selalu tercatat akurat.

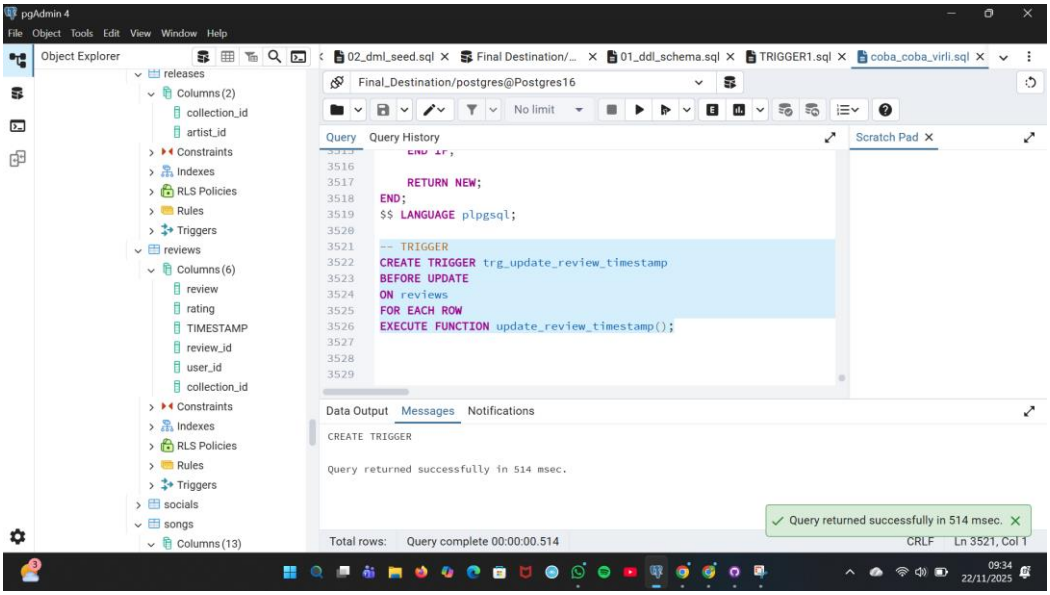
Script SQL

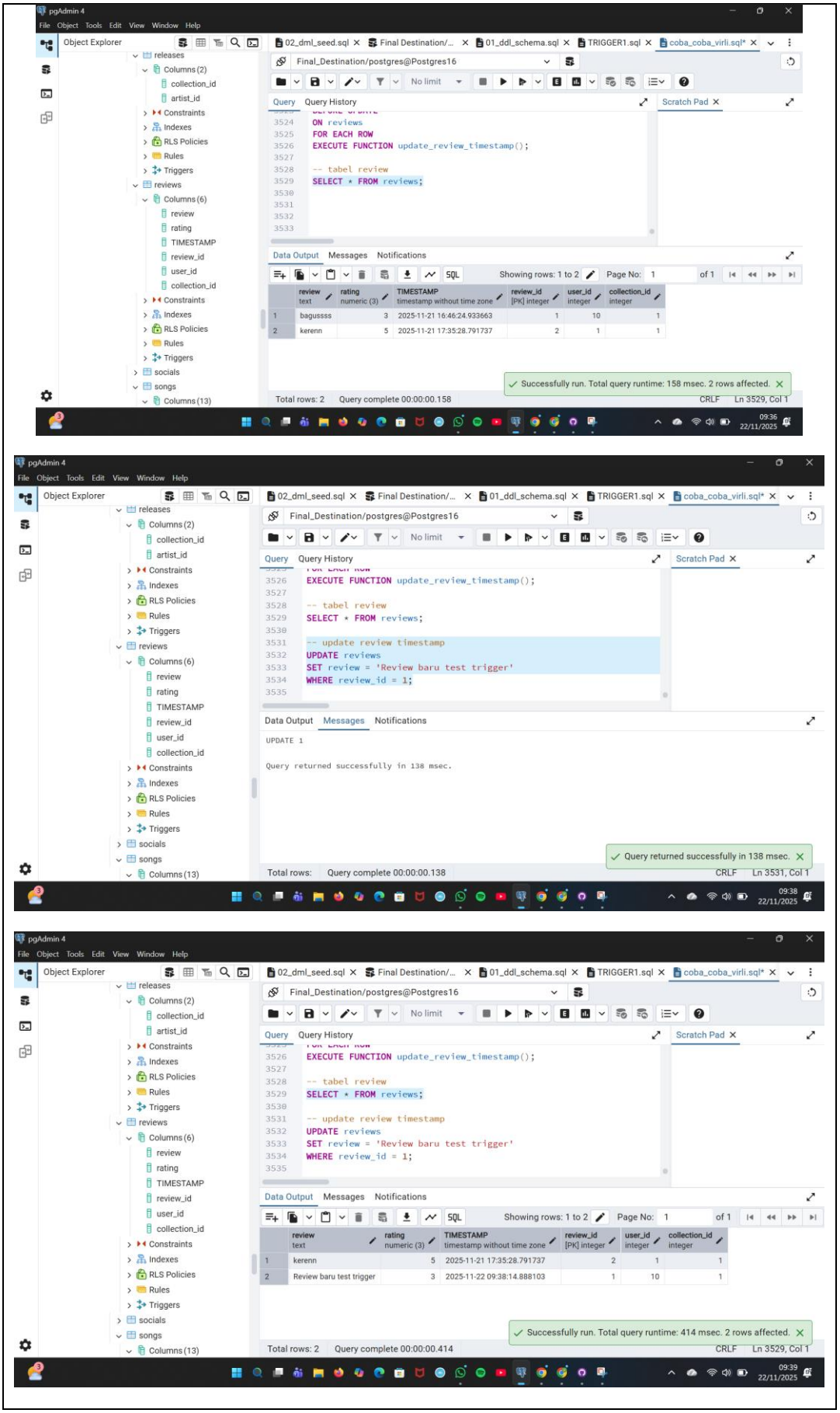
```
-- FUNCTION untuk update timestamp saat review/rating berubah
CREATE OR REPLACE FUNCTION update_review_timestamp()
RETURNS TRIGGER AS $$
BEGIN
    -- Jika kolom review atau rating berubah
    IF NEW.review IS DISTINCT FROM OLD.review  -- IS DISTINCT FROM
digunakan agar aman untuk nilai NULL.
       OR NEW.rating IS DISTINCT FROM OLD.rating THEN

        NEW."TIMESTAMP" = CURRENT_TIMESTAMP; -- Jika ada perubahan
pada review atau rating, update kolom TIMESTAMP dengan waktu saat
ini.

    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- TRIGGER
CREATE TRIGGER trg_update_review_timestamp
BEFORE UPDATE
```

```
ON reviews
FOR EACH ROW
EXECUTE FUNCTION update_review_timestamp();
```

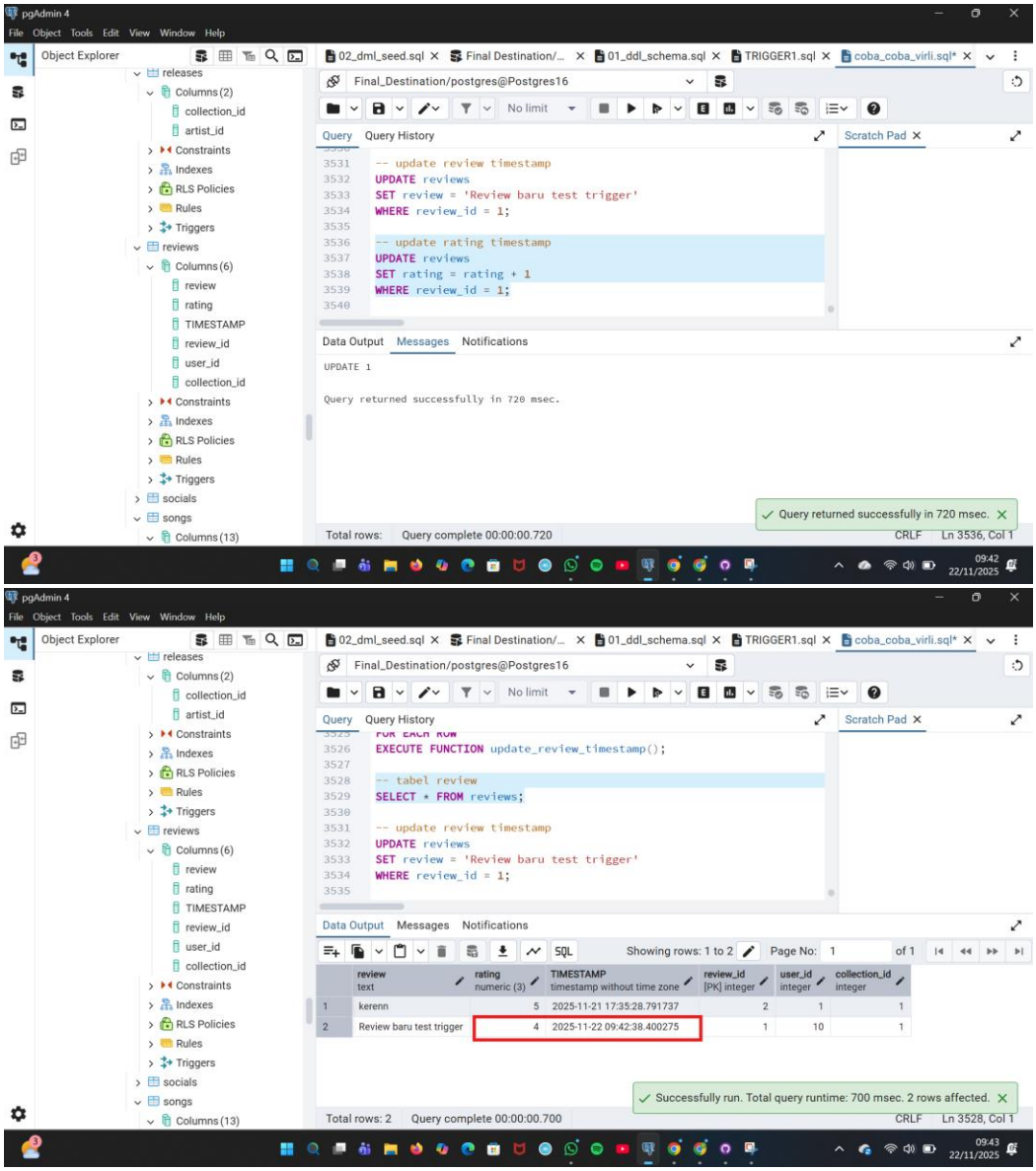
Screenshot Hasil






pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
02_dml_seed.sql
Final Destination/...
01_ddl_schema.sql
TRIGGER1.sql
coba_coba_virli.sql*
Final_Destination/postgres@Postgres16
Query
Query History
Scratch Pad
ON reviews
FOR EACH ROW
EXECUTE FUNCTION update_review_timestamp();
-- tabel review
SELECT * FROM reviews;
Data Output
Messages
Notifications
Showing rows: 1 to 2
Page No: 1 of 1
review text
rating numeric (3)
TIMESTAMP timestamp without time zone
review_id [PK] integer
user_id integer
collection_id integer
1 bagussss 3 2025-11-21 16:46:24.933663 1 10 1
2 kerenn 5 2025-11-21 17:35:28.791737 2 1 1
Successfully run. Total query runtime: 158 msec. 2 rows affected.
Total rows: 2
Query complete 00:00:00.158
CRLF
Ln 3529, Col 1
09:36
22/11/2025
-- update review timestamp
UPDATE reviews
SET review = 'Review baru test trigger'
WHERE review_id = 1;
UPDATE 1
Query returned successfully in 138 msec.
Total rows:
Query complete 00:00:00.138
Ln 3531, Col 1
09:38
22/11/2025
Successfully run. Total query runtime: 414 msec. 2 rows affected.
Total rows: 2
Query complete 00:00:00.414
Ln 3529, Col 1
09:39
22/11/2025

-- cek update rating



Tabel 119 PL/SQL - Trigger trg_fix_playlist_collaborators

Nama Trigger	trg_fix_playlist_collaborators
Aksi	UPDATE (kolom: iscollaborative)
Jenis	AFTER UPDATE

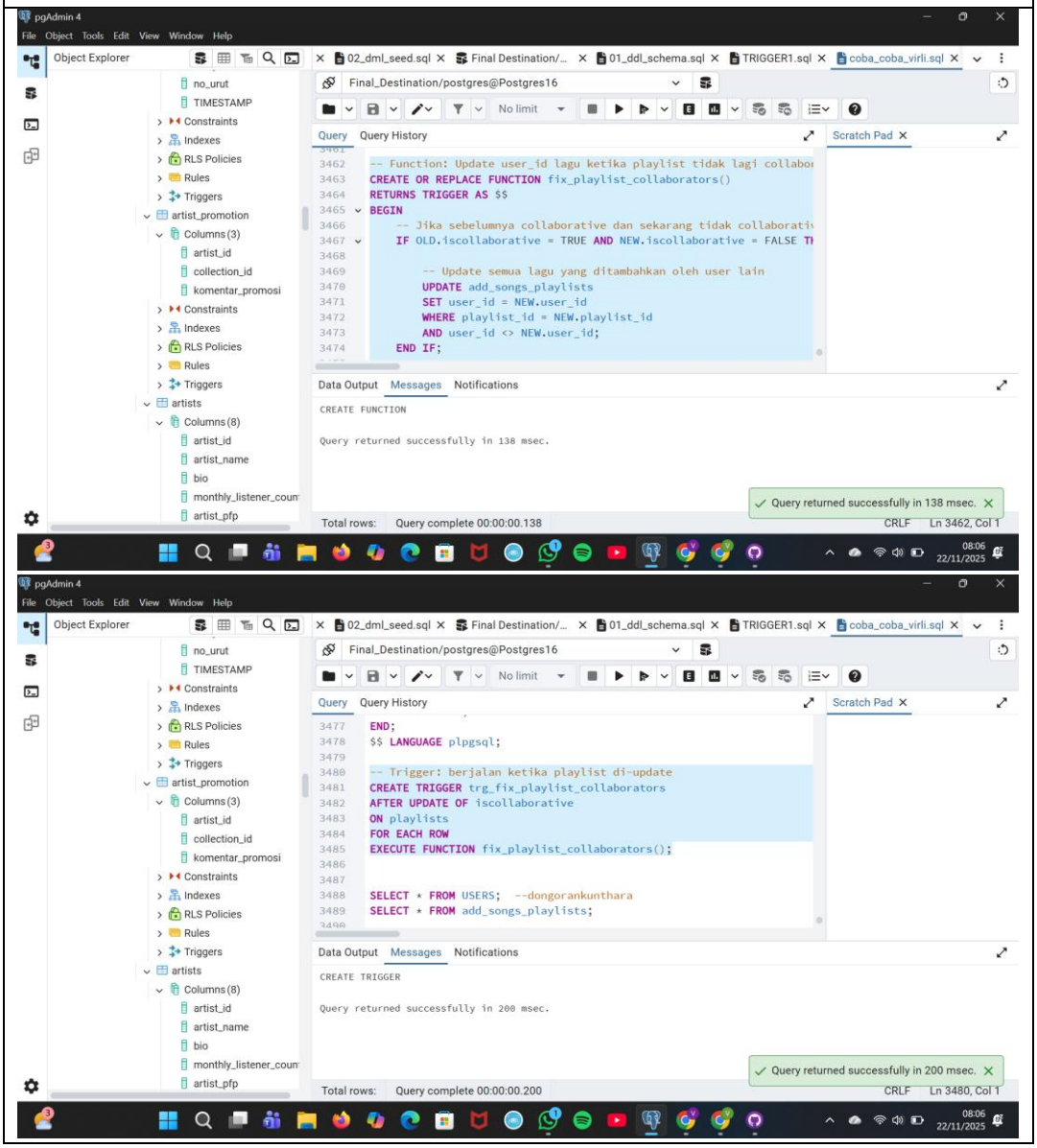
Deskripsi	Trigger ini digunakan ketika sebuah playlist yang sebelumnya collaborative (true) diubah menjadi non-collaborative (false). Saat perubahan ini terjadi, seluruh lagu yang sebelumnya ditambahkan oleh user selain pemilik playlist akan diperbarui user_id-nya menjadi user_id pemilik playlist. Tujuan trigger ini adalah memastikan bahwa ketika playlist tidak lagi bersifat kolaboratif, seluruh lagu di dalamnya dianggap berasal dari pemilik playlist. Logika Perhitungan: Jika OLD.iscollaborative = TRUE dan NEW.iscollaborative = FALSE → maka untuk setiap lagu dalam playlist tersebut: Jika user_id != owner (NEW.user_id) → ganti menjadi NEW.user_id.

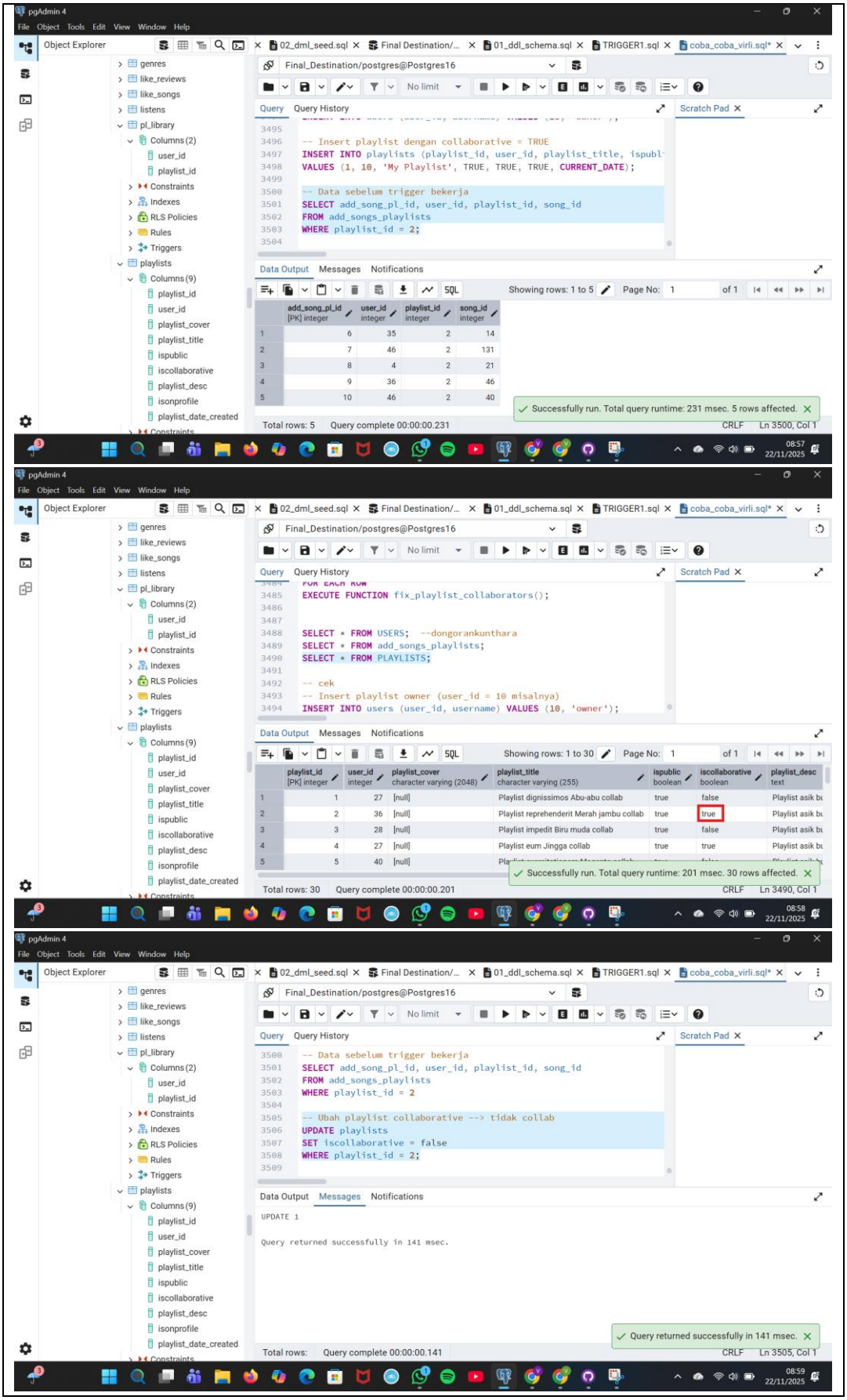
Script SQL

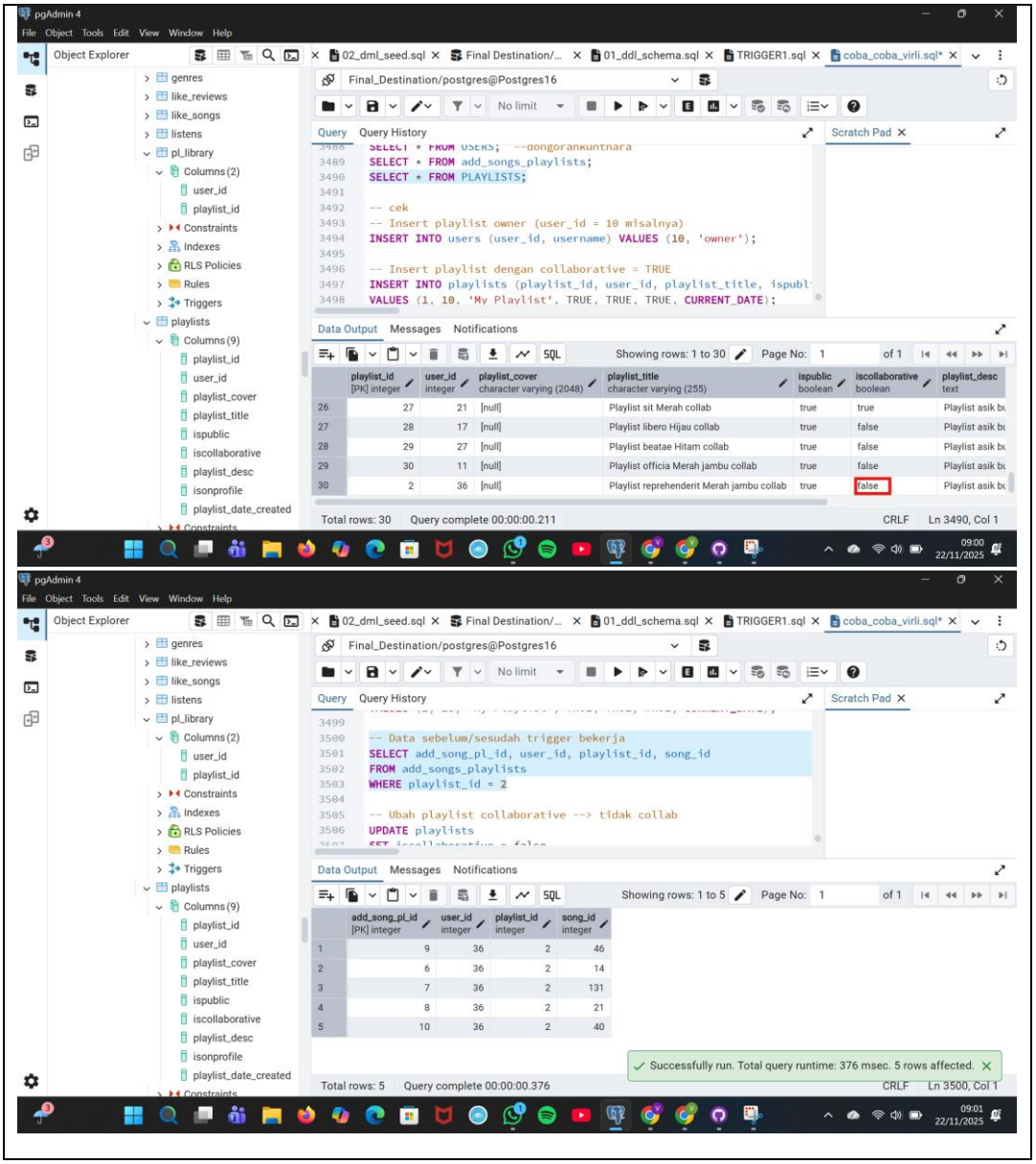
```
-- Function: Update user_id lagu ketika playlist tidak lagi
collaborative
CREATE OR REPLACE FUNCTION fix_playlist_collaborators()
RETURNS TRIGGER AS $$
BEGIN
    -- Jika sebelumnya collaborative dan sekarang tidak
collaborative
    IF OLD.iscollaborative = TRUE AND NEW.iscollaborative = FALSE
THEN

        -- Update semua lagu yang ditambahkan oleh user lain
        UPDATE add_songs_playlists
        SET user_id = NEW.user_id  -- ganti pemilik lagu
        WHERE playlist_id = NEW.playlist_id  -- hanya untuk
playlist ini
        AND user_id <> NEW.user_id;  -- hanya yang bukan owner
    END IF;
```

```
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger: berjalan ketika playlist di-update
CREATE TRIGGER trg_fix_playlist_collaborators
AFTER UPDATE OF iscollaborative
ON playlists  -- trigger untuk tabel playlists
FOR EACH ROW
EXECUTE FUNCTION fix_playlist_collaborators();
```

Screenshot Hasil




pgAdmin 4
Object Explorer
Final_Destination/postgres@Postgres16
Query
Query History
Scratch Pad
Data Output
Messages
Notifications
Total rows: 5
Query complete 00:00:00.231
Successfully run. Total query runtime: 231 msec. 5 rows affected.
Total rows: 30
Query complete 00:00:00.201
Successfully run. Total query runtime: 201 msec. 30 rows affected.
UPDATE 1
Query returned successfully in 141 msec.
Total rows:
Query complete 00:00:00.141
Query returned successfully in 141 msec.



Tabel 120 PL/SQL - Trigger trg_update_artist_follower_count

Nama Trigger	trg_update_artist_follow_count
Aksi	INSERT OR DELETE
Jenis	AFTER

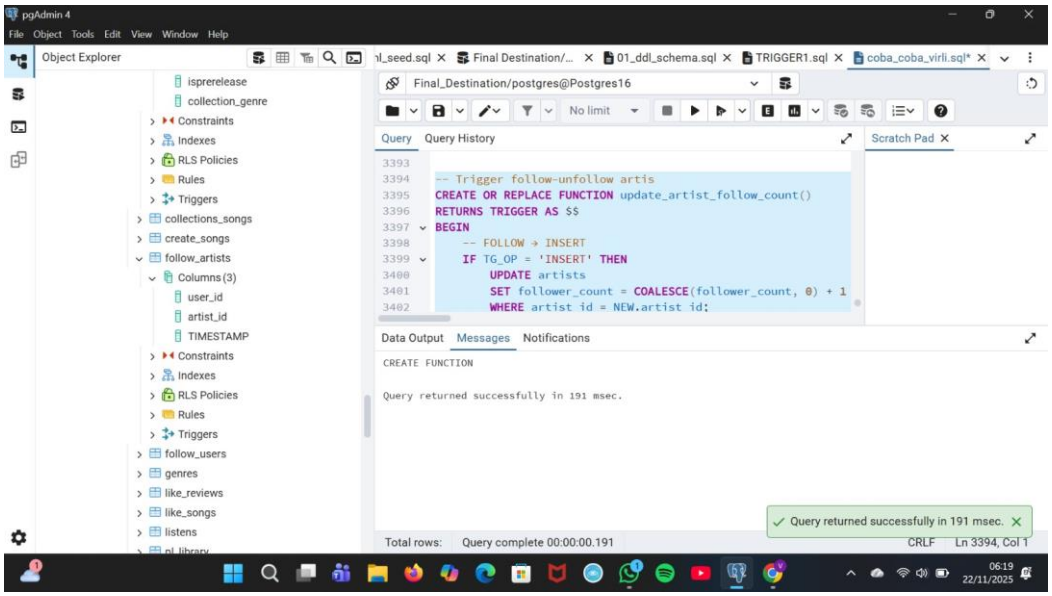
Deskripsi	Trigger ini digunakan untuk menjaga konsistensi jumlah followers artis pada tabel artists. Setiap kali terjadi aksi FOLLOW (insert ke tabel follow_artists), nilai follower_count artis akan bertambah 1. Sebaliknya, saat terjadi UNFOLLOW (delete dari tabel follow_artists), nilai follower_count akan berkurang 1. Dan trigger ini memastikan bahwa jumlah followers selalu sesuai dengan jumlah baris aktual dalam tabel follow_artists.

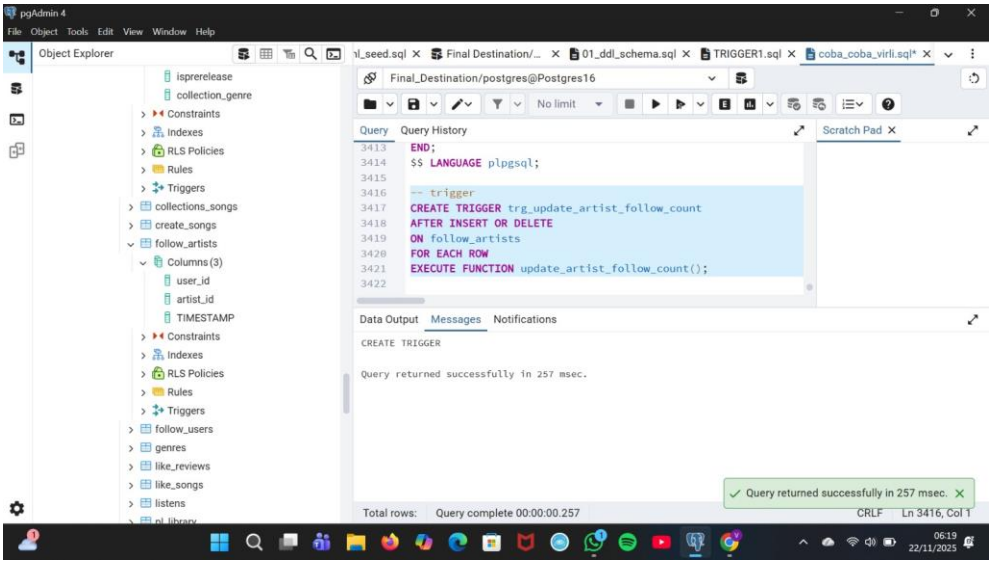
Script SQL

```
-- Trigger follow-unfollow artis
CREATE OR REPLACE FUNCTION update_artist_follow_count()
RETURNS TRIGGER AS $$
BEGIN
    -- FOLLOW → INSERT
    IF TG_OP = 'INSERT' THEN  -- TG_OP berisi string yang
menunjukkan operasi apa yang memicu trigger nya
        UPDATE artists
        SET follower_count = COALESCE(follower_count, 0) + 1
        WHERE artist_id = NEW.artist_id;  -- artist_id dari baris
baru
    END IF;

    -- UNFOLLOW → DELETE
    IF TG_OP = 'DELETE' THEN  -- TG_OP berisi string yang
menunjukkan operasi apa yang memicu trigger nya
        UPDATE artists
        SET follower_count = COALESCE(follower_count, 0) - 1
        WHERE artist_id = OLD.artist_id;  -- artist_id dari baris
lama
    END IF;
```

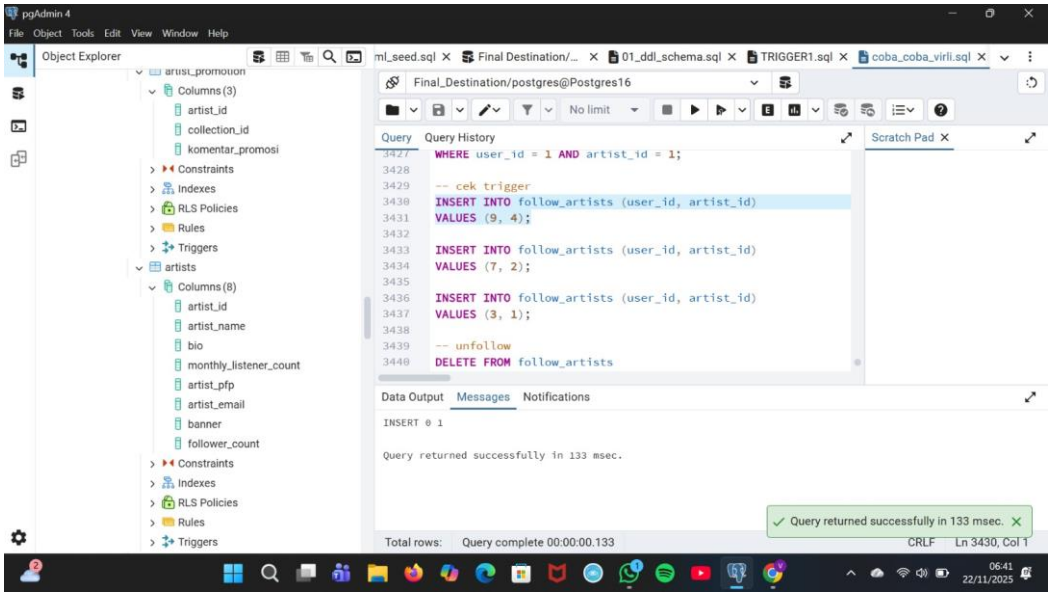
```
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

-- trigger
CREATE TRIGGER trg_update_artist_follow_count
AFTER INSERT OR DELETE
ON follow_artists
FOR EACH ROW
EXECUTE FUNCTION update_artist_follow_count();
```

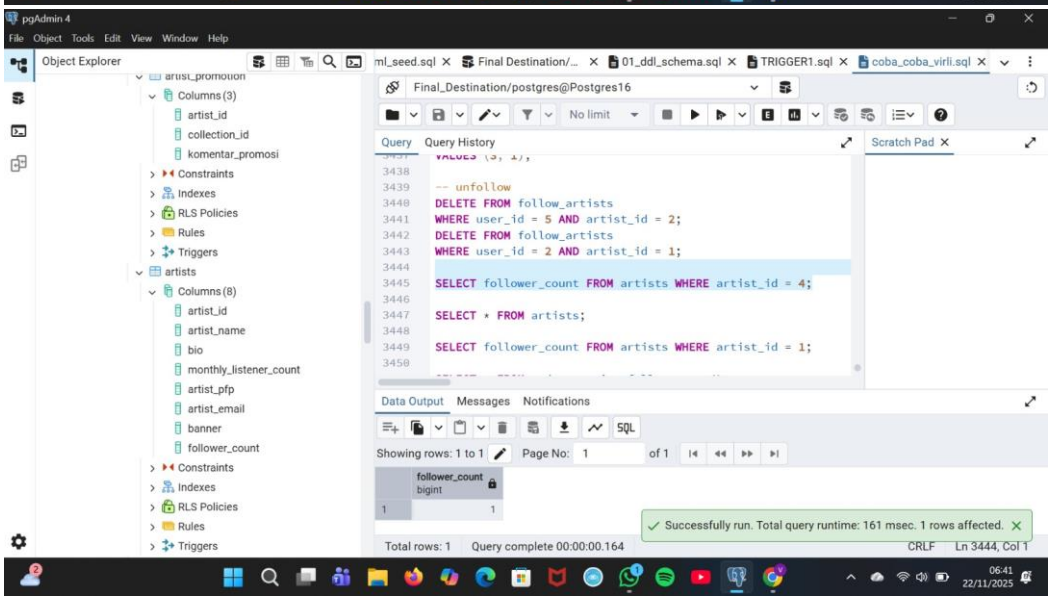
Screenshot Hasil

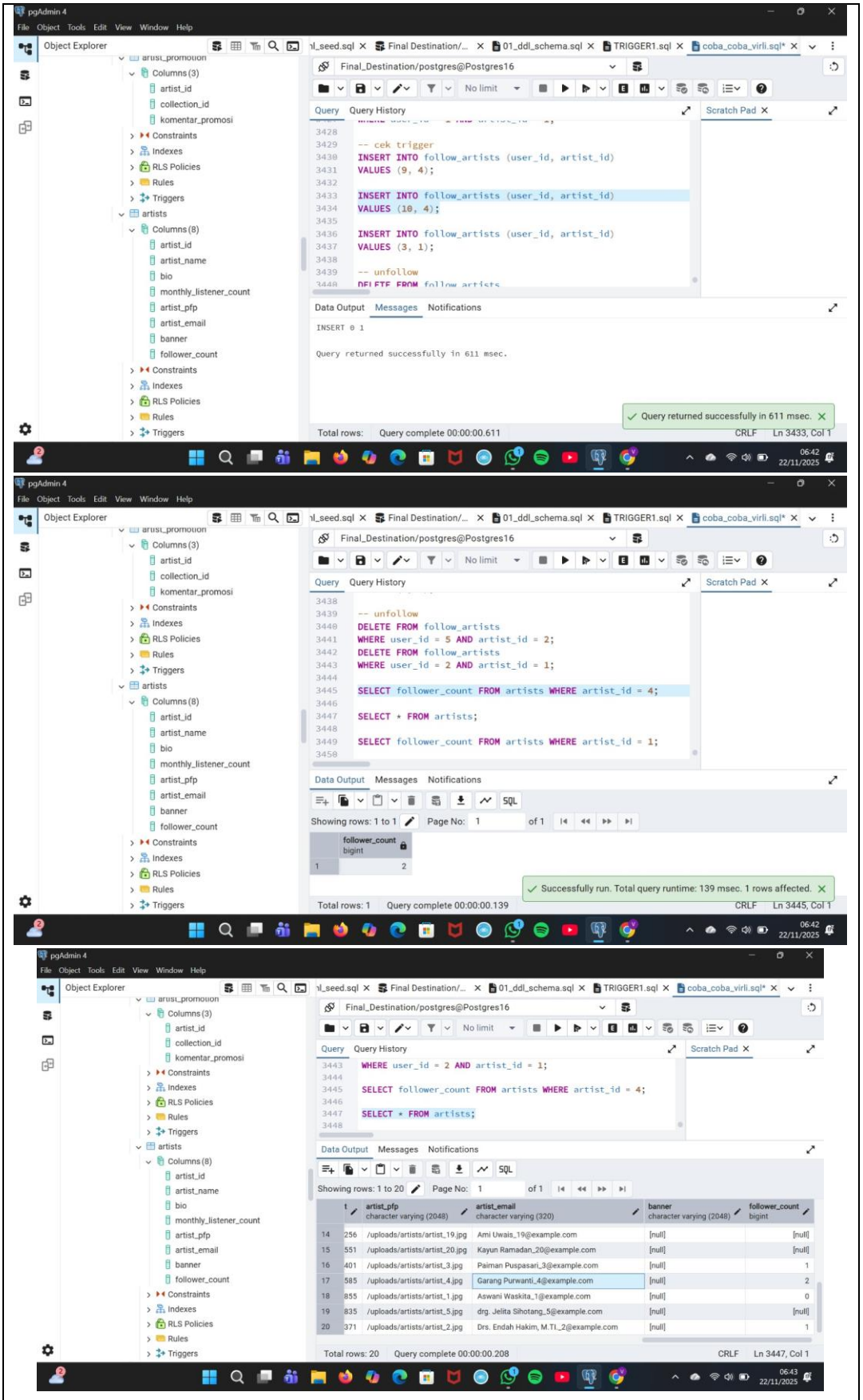




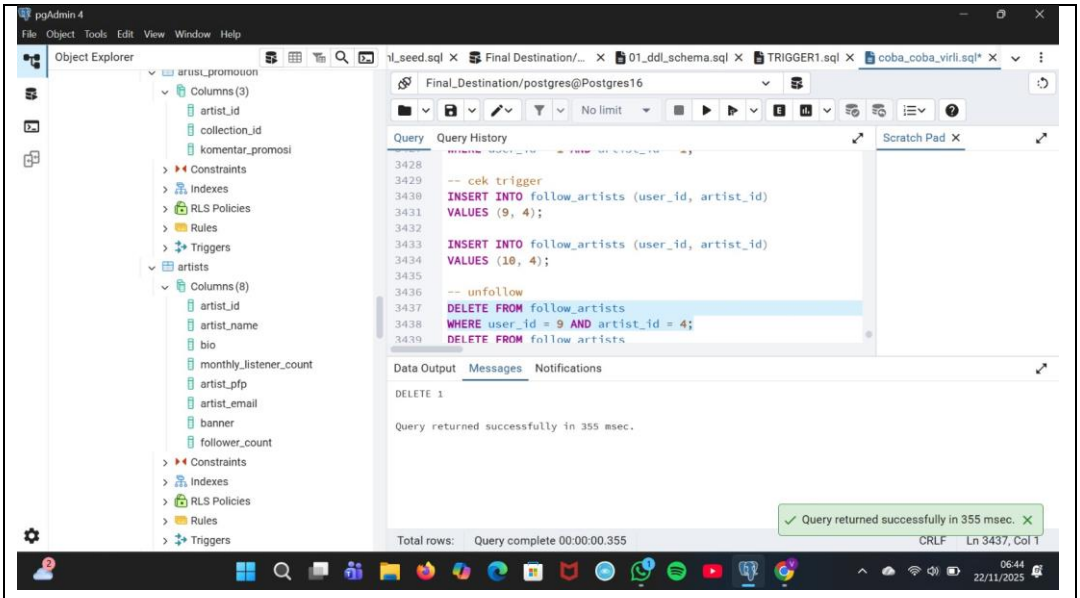
-- Follow (Insert)

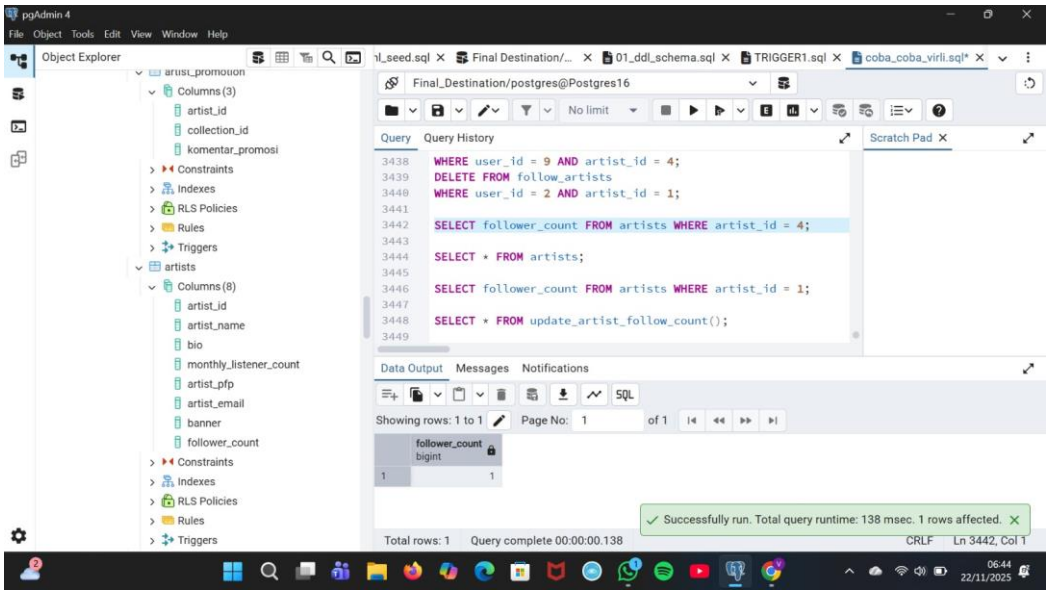


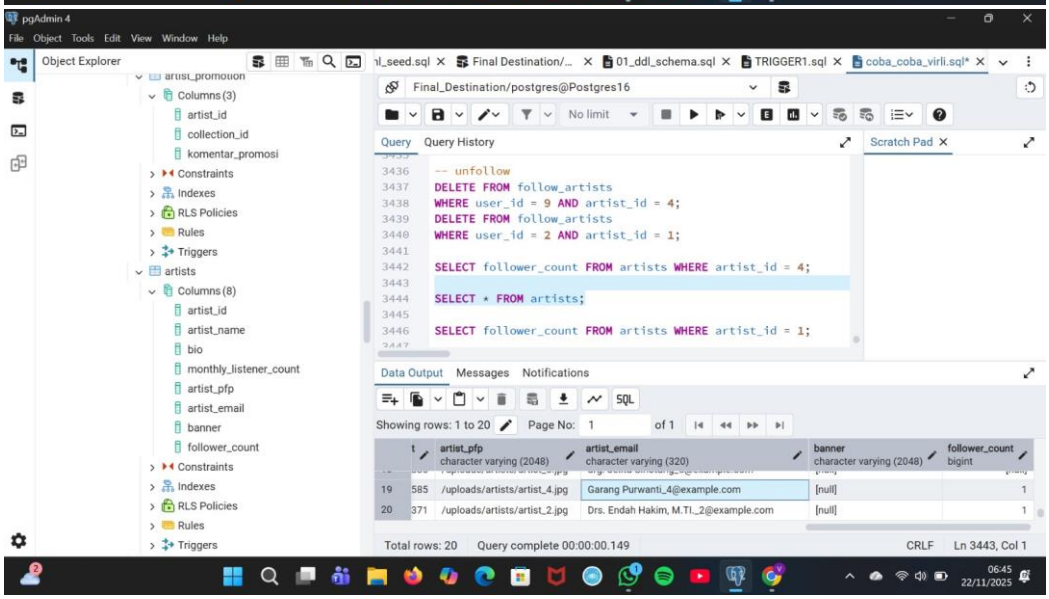




-- Unffolow (Delete)


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Final_Destination/postgres@Postgres16
Query Query History
Scratch Pad
-- cek trigger
INSERT INTO follow_artists (user_id, artist_id)
VALUES (9, 4);
INSERT INTO follow_artists (user_id, artist_id)
VALUES (10, 4);
-- unfollow
DELETE FROM follow_artists
WHERE user_id = 9 AND artist_id = 4;
Data Output Messages Notifications
DELETE 1
Query returned successfully in 355 msec.
Total rows: Query complete 00:00:00.355
Query returned successfully in 355 msec.
CRLF Ln 3437, Col 1
06:44
22/11/2025


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Final_Destination/postgres@Postgres16
Query Query History
Scratch Pad
WHERE user_id = 9 AND artist_id = 4;
DELETE FROM follow_artists
WHERE user_id = 2 AND artist_id = 1;
SELECT follower_count FROM artists WHERE artist_id = 4;
SELECT * FROM artists;
SELECT follower_count FROM artists WHERE artist_id = 1;
SELECT * FROM update_artist_follow_count();
Data Output Messages Notifications
Showing rows: 1 to 1
Page No: 1 of 1
follower_count bigint
1
Successfully run. Total query runtime: 138 msec. 1 rows affected.
Total rows: 1 Query complete 00:00:00.138
CRLF Ln 3442, Col 1
06:44
22/11/2025


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Final_Destination/postgres@Postgres16
Query Query History
Scratch Pad
-- unfollow
DELETE FROM follow_artists
WHERE user_id = 9 AND artist_id = 4;
DELETE FROM follow_artists
WHERE user_id = 2 AND artist_id = 1;
SELECT follower_count FROM artists WHERE artist_id = 4;
SELECT * FROM artists;
SELECT follower_count FROM artists WHERE artist_id = 1;
Data Output Messages Notifications
Showing rows: 1 to 20
Page No: 1 of 1
artist_pfp character varying (2048)
artist_email character varying (320)
banner character varying (2048)
follower_count bigint
19 585 /uploads/artists/artist_4.jpg Garang Purwanti_4@example.com [null] 1
20 371 /uploads/artists/artist_2.jpg Drs. Endah Hakim, M.TI._2@example.com [null] 1
Total rows: 20 Query complete 00:00:00.149
CRLF Ln 3443, Col 1
06:45
22/11/2025

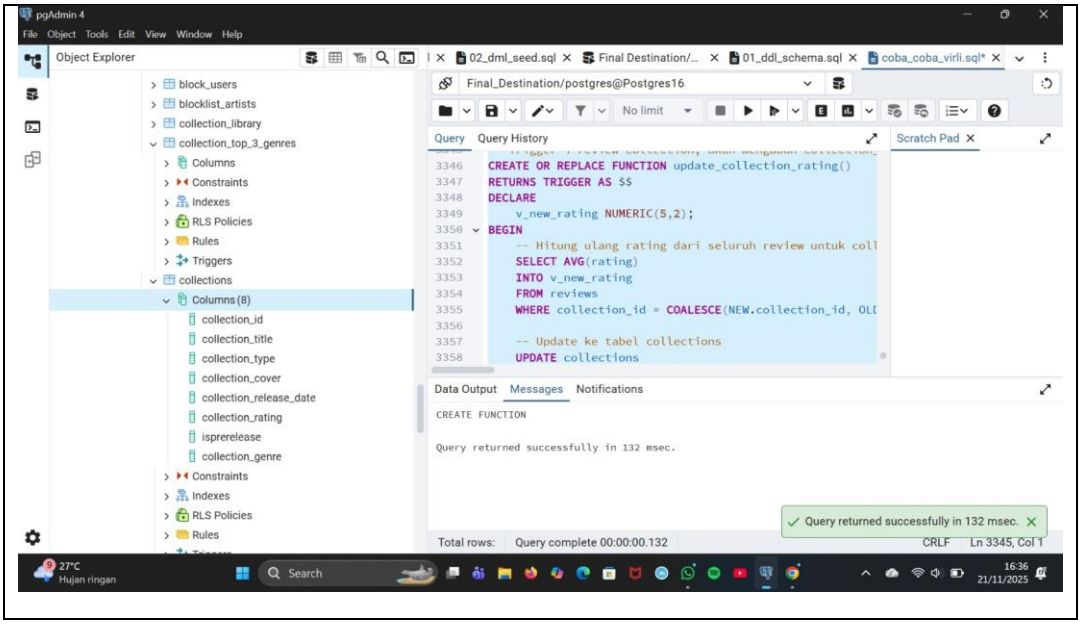
Tabel 121 PL/SQL - Trigger trg_update_collection_rating

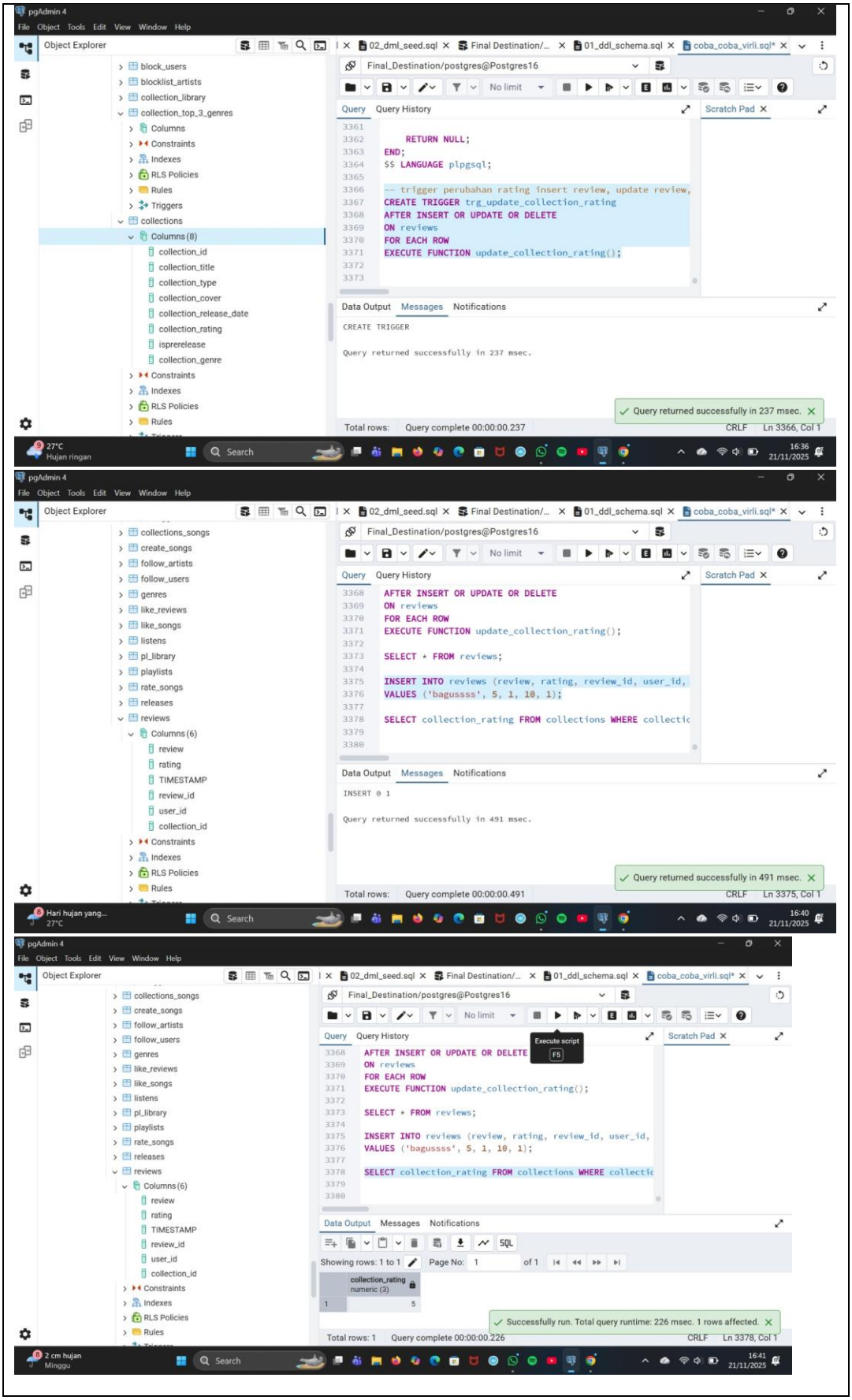
Nama Trigger	trg_update_collection_rating
Aksi	INSERT, UPDATE, DELETE
Jenis	AFTER
Deskripsi	Trigger (trg_update_collection_rating) digunakan untuk memperbarui nilai collection_rating pada tabel collections setiap kali terjadi perubahan data pada tabel reviews. Trigger ini menghitung ulang rata-rata rating dari seluruh review berdasarkan collection_id terkait, lalu menyimpannya kembali ke kolom collection_rating. Proses perhitungan dilakukan dengan cara mengambil rerata (AVG) dari semua nilai rating pada tabel reviews untuk koleksi tersebut. Contoh perhingan nya terjadi selama pengujian: - Review 1 memberikan rating 3 - Review 2 nya memberikan rating 5 Maka perhitungan rata – rata: (3 + 5) / 2 = 4 Nilai 4 ini kemudian disimpan ke kolom collection_rating pada tabel collections. Dengan adanya trigger ini, nilai rating koleksi akan selalu akurat tanpa perlu diupdate secara manual.
Script SQL	

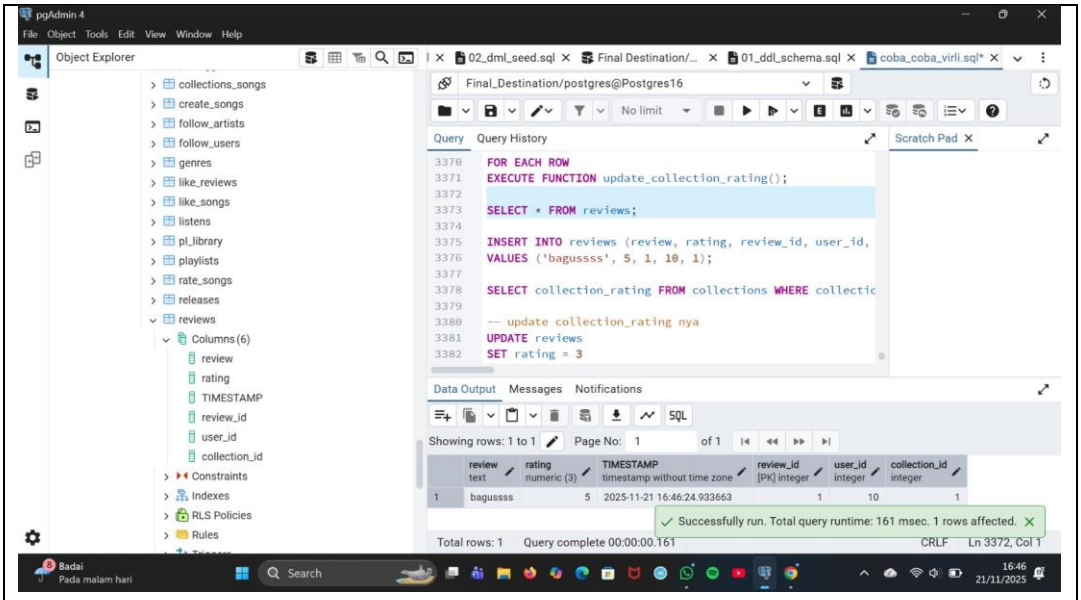
```
-- Trigger review collection, akan mengubah collection_rating
```

```sql
CREATE OR REPLACE FUNCTION update_collection_rating()
RETURNS TRIGGER AS $$
DECLARE
    v_new_rating NUMERIC(5,2);  -- Variable untuk menyimpan rata-rata rating terbaru
BEGIN
    -- Hitung ulang rating dari seluruh review untuk collection
    SELECT AVG(rating)  -- Mengambil nilai rata-rata (AVG) dari semua rating pada review
    INTO v_new_rating  -- Menyimpan hasil rata-rata tersebut ke variabel v_new_rating
    FROM reviews  -- Mengambil data dari tabel reviews
    --   INSERT → pakai NEW.collection_id
    --   UPDATE → pakai NEW.collection_id
    --   DELETE → pakai OLD.collection_id
    WHERE collection_id = COALESCE(NEW.collection_id, OLD.collection_id);

    -- Update ke tabel collections
    UPDATE collections
    SET collection_rating = v_new_rating  -- Menuliskan nilai rata-rata terbaru ke kolom collection_rating
    WHERE collection_id = COALESCE(NEW.collection_id, OLD.collection_id);  -- Menentukan collection mana yang harus di-update

    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

-- trigger perubahan rating insert review, update review, delete review
CREATE TRIGGER trg_update_collection_rating
AFTER INSERT OR UPDATE OR DELETE
ON reviews
FOR EACH ROW
EXECUTE FUNCTION update_collection_rating();
```

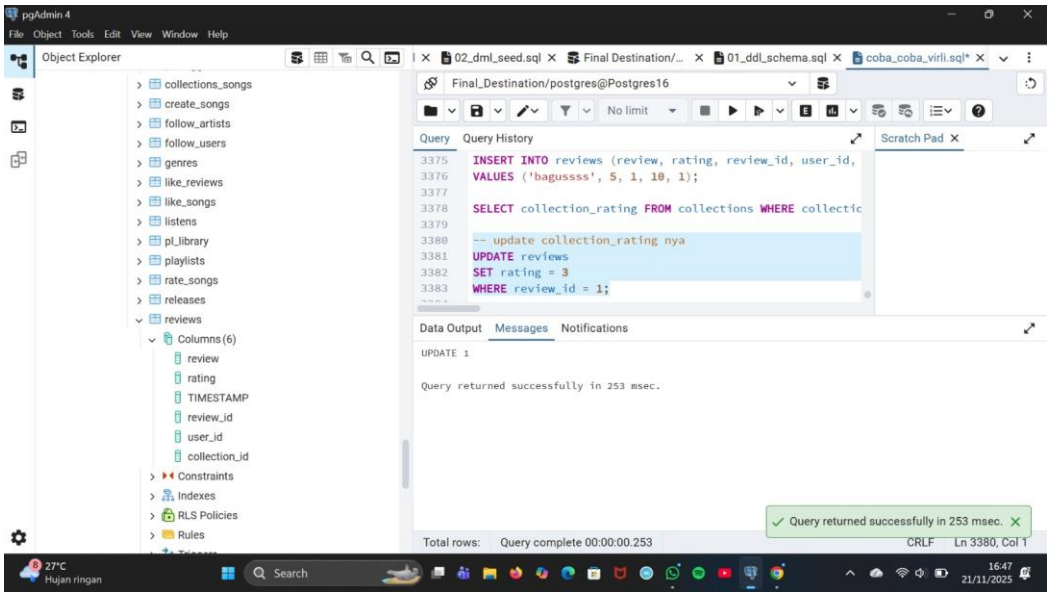
Screenshot Hasil

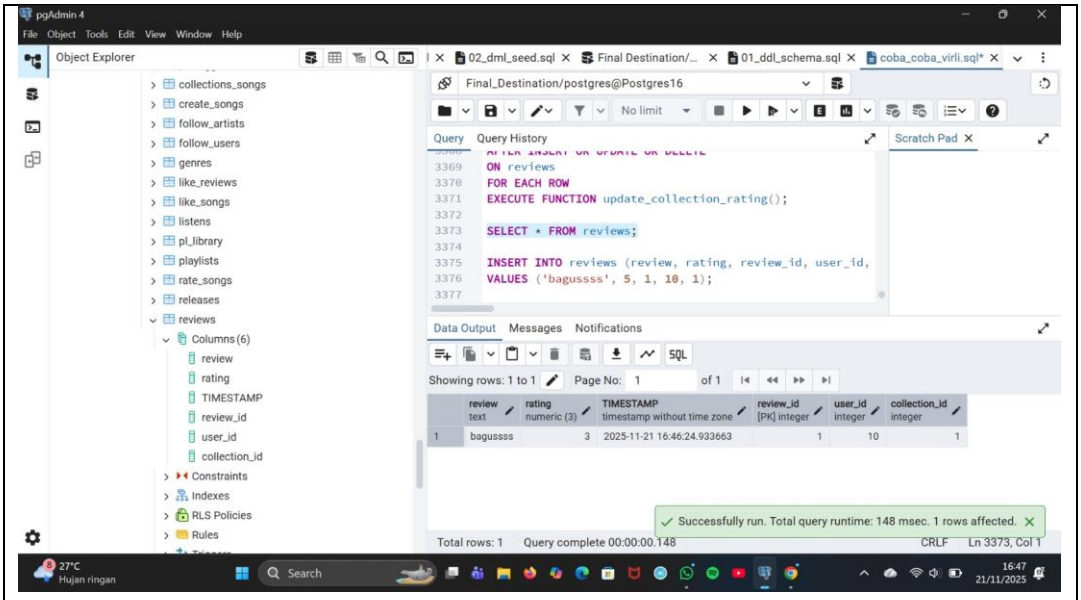

pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
block_users
blocklist_artists
collection_library
collection_top_3_genres
Columns
Constraints
Indexes
RLS Policies
Rules
Triggers
collections
Columns (8)
collection_id
collection_title
collection_type
collection_cover
collection_release_date
collection_rating
isprerelease
collection_genre
Constraints
Indexes
RLS Policies
Rules
02_dml_seed.sql
Final Destination/...
01_ddl_schema.sql
coba_coba_virli.sql*
Final_Destination/postgres@Postgres16
No limit
Query
Query History
Scratch Pad
3346 CREATE OR REPLACE FUNCTION update_collection_rating()
3347 RETURNS TRIGGER AS $$
3348 DECLARE
3349 v_new_rating NUMERIC(5,2);
3350 BEGIN
3351 -- Hitung ulang rating dari seluruh review untuk coll
3352 SELECT AVG(rating)
3353 INTO v_new_rating
3354 FROM reviews
3355 WHERE collection_id = COALESCE(NEW.collection_id, OLD
3356
3357 -- Update ke tabel collections
3358 UPDATE collections
Data Output
Messages
Notifications
CREATE FUNCTION
Query returned successfully in 132 msec.
Query returned successfully in 132 msec.
Total rows:
Query complete 00:00:00.132
CRLF
Ln 3345, Col 1
27°C
Hujan ringan
Search
16:36
21/11/2025


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
02_dml_seed.sql
Final Destination/...
01_ddl_schema.sql
coba_coba_virli.sql*
Final_Destination/postgres@Postgres16
No limit
Query
Query History
Scratch Pad
block_users
blocklist_artists
collection_library
collection_top_3_genres
Columns
Constraints
Indexes
RLS Policies
Rules
Triggers
collections
Columns (8)
collection_id
collection_title
collection_type
collection_cover
collection_release_date
collection_rating
isprerelease
collection_genre
Constraints
Indexes
RLS Policies
Rules
3361
RETURN NULL;
END;
$$ LANGUAGE plpgsql;
-- trigger perubahan rating insert review, update review,
CREATE TRIGGER trg_update_collection_rating
AFTER INSERT OR UPDATE OR DELETE
ON reviews
FOR EACH ROW
EXECUTE FUNCTION update_collection_rating();
Data Output
Messages
Notifications
CREATE TRIGGER
Query returned successfully in 237 msec.
Query returned successfully in 237 msec.
Total rows:
Query complete 00:00:00.237
CRLF
Ln 3366, Col 1
27°C
Hujan ringan
Search
16:36
21/11/2025
collections_songs
create_songs
follow_artists
follow_users
genres
like_reviews
like_songs
listens
pl_library
playlists
rate_songs
releases
reviews
Columns (6)
review
rating
TIMESTAMP
review_id
user_id
collection_id
Constraints
Indexes
RLS Policies
Rules
AFTER INSERT OR UPDATE OR DELETE
ON reviews
FOR EACH ROW
EXECUTE FUNCTION update_collection_rating();
SELECT * FROM reviews;
INSERT INTO reviews (review, rating, review_id, user_id,
VALUES ('bagussss', 5, 1, 10, 1);
SELECT collection_rating FROM collections WHERE collectic
INSERT 0 1
Query returned successfully in 491 msec.
Query returned successfully in 491 msec.
Total rows:
Query complete 00:00:00.491
CRLF
Ln 3375, Col 1
Hari hujan yang...
27°C
Search
16:40
21/11/2025
Execute script
F5
SQL
Showing rows: 1 to 1
Page No: 1
of 1
collection_rating
numeric (3)
1
5
Successfully run. Total query runtime: 226 msec. 1 rows affected.
Total rows: 1
Query complete 00:00:00.226
CRLF
Ln 3378, Col 1
2 cm hujan
Minggu
Search
16:41
21/11/2025

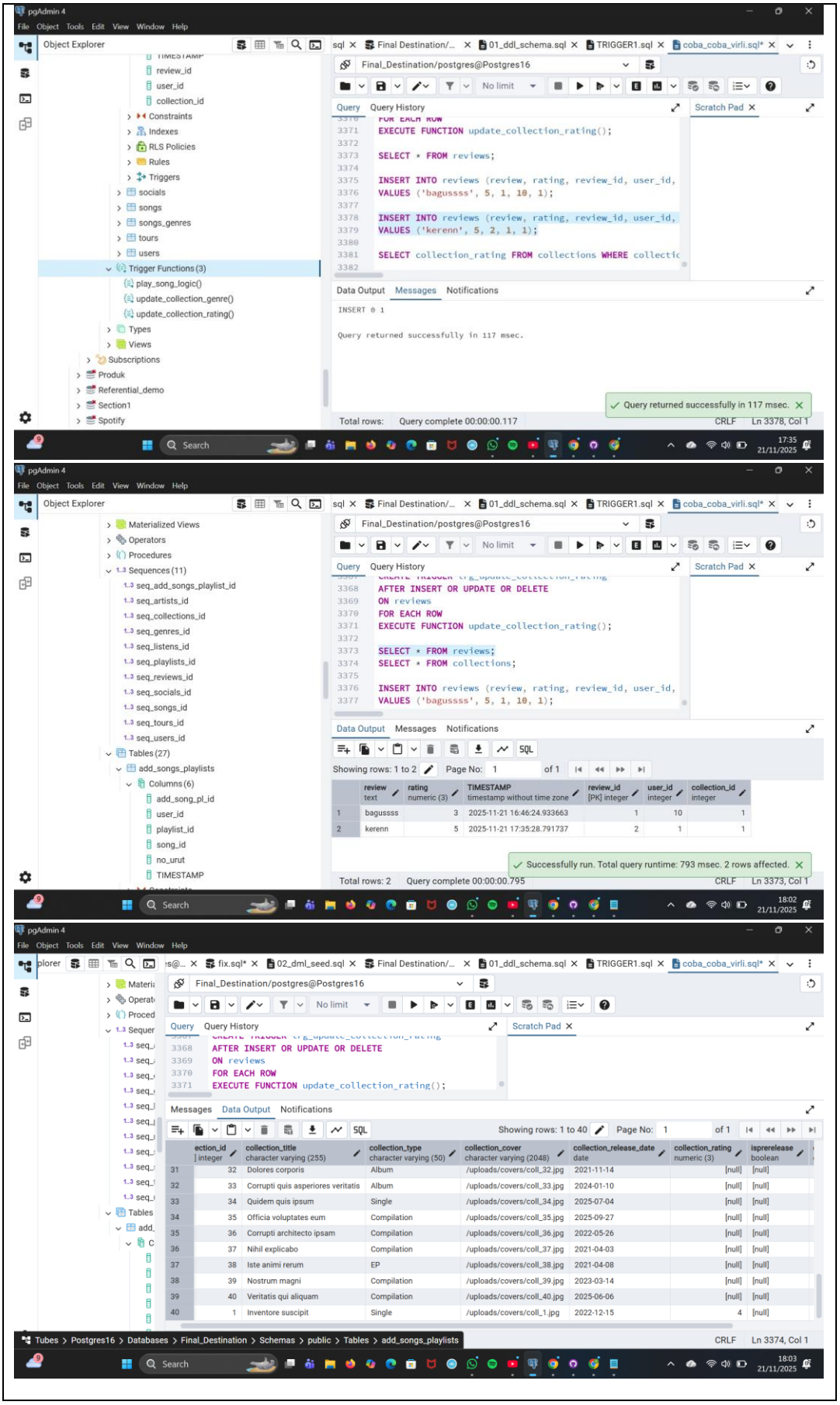

pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Final_Destination/postgres@Postgres16
Query Query History
Scratch Pad
FOR EACH ROW
EXECUTE FUNCTION update_collection_rating();
SELECT * FROM reviews;
INSERT INTO reviews (review, rating, review_id, user_id,
VALUES ('bagussss', 5, 1, 10, 1);
SELECT collection_rating FROM collections WHERE collectic
-- update collection_rating nya
UPDATE reviews
SET rating = 3
Data Output Messages Notifications
Showing rows: 1 to 1 Page No: 1 of 1
review text
rating numeric (3)
TIMESTAMP timestamp without time zone
review_id [PK] integer
user_id integer
collection_id integer
bagussss 5 2025-11-21 16:46:24.933663 1 10 1
Successfully run. Total query runtime: 161 msec. 1 rows affected.
Total rows: 1 Query complete 00:00:00.161 CRLF Ln 3372, Col 1
Badai Pada malam hari
Search
16:46 21/11/2025

-- update


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Final_Destination/postgres@Postgres16
Query Query History
Scratch Pad
INSERT INTO reviews (review, rating, review_id, user_id,
VALUES ('bagussss', 5, 1, 10, 1);
SELECT collection_rating FROM collections WHERE collectic
-- update collection_rating nya
UPDATE reviews
SET rating = 3
WHERE review_id = 1;
Data Output Messages Notifications
UPDATE 1
Query returned successfully in 253 msec.
Query returned successfully in 253 msec.
Total rows: Query complete 00:00:00.253 CRLF Ln 3380, Col 1
27°C Hujan ringan
Search
16:47 21/11/2025



-- nambah review baru dari user_id berbeda


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Final_Destination/postgres@Postgres16
Query Query History
Scratch Pad
FOR EACH ROW
EXECUTE FUNCTION update_collection_rating();
SELECT * FROM reviews;
INSERT INTO reviews (review, rating, review_id, user_id,
VALUES ('bagussss', 5, 1, 10, 1);
INSERT INTO reviews (review, rating, review_id, user_id,
VALUES ('kerenn', 5, 2, 1, 1);
SELECT collection_rating FROM collections WHERE collectic
Data Output Messages Notifications
INSERT 0 1
Query returned successfully in 117 msec.
Total rows: Query complete 00:00:00.117
Query returned successfully in 117 msec.
CRLF Ln 3378, Col 1
Trigger Functions (3)
play_song_logic()
update_collection_genre()
update_collection_rating()
Sequences (11)
Tables (27)
add_songs_playlists
Columns (6)
AFTER INSERT OR UPDATE OR DELETE
ON reviews
FOR EACH ROW
EXECUTE FUNCTION update_collection_rating();
SELECT * FROM reviews;
SELECT * FROM collections;
Showing rows: 1 to 2
Page No: 1 of 1
review text
rating numeric (3)
TIMESTAMP timestamp without time zone
review_id [PK] integer
user_id integer
collection_id integer
bagussss 3 2025-11-21 16:46:24.933663 1 10 1
kerenn 5 2025-11-21 17:35:28.791737 2 1 1
Successfully run. Total query runtime: 793 msec. 2 rows affected.
Total rows: 2 Query complete 00:00:00.795
CRLF Ln 3373, Col 1
Showing rows: 1 to 40
collection_id integer
collection_title character varying (255)
collection_type character varying (50)
collection_cover character varying (2048)
collection_release_date date
collection_rating numeric (3)
isprerelease boolean
31 32 Dolores corporis Album /uploads/covers/coll_32.jpg 2021-11-14 [null] [null]
32 33 Corrupti quis asperiores veritatis Album /uploads/covers/coll_33.jpg 2024-01-10 [null] [null]
33 34 Quidem quis ipsum Single /uploads/covers/coll_34.jpg 2025-07-04 [null] [null]
34 35 Officia voluptates eum Compilation /uploads/covers/coll_35.jpg 2025-09-27 [null] [null]
35 36 Corrupti architecto ipsam Compilation /uploads/covers/coll_36.jpg 2022-05-26 [null] [null]
36 37 Nihil explicabo Compilation /uploads/covers/coll_37.jpg 2021-04-03 [null] [null]
37 38 Iste animi rerum EP /uploads/covers/coll_38.jpg 2021-04-08 [null] [null]
38 39 Nostrum magni Compilation /uploads/covers/coll_39.jpg 2023-03-14 [null] [null]
39 40 Veritatis qui aliquam Compilation /uploads/covers/coll_40.jpg 2025-06-06 [null] [null]
40 1 Inventore suscipit Single /uploads/covers/coll_1.jpg 2022-12-15 4 [null]
Tubes > Postgres16 > Databases > Final_Destination > Schemas > public > Tables > add_songs_playlists
CRLF Ln 3374, Col 1
17:35 21/11/2025
18:02 21/11/2025
18:03 21/11/2025

Tabel 122 PL/SQL - Trigger trg_reorder_playlist_sequence

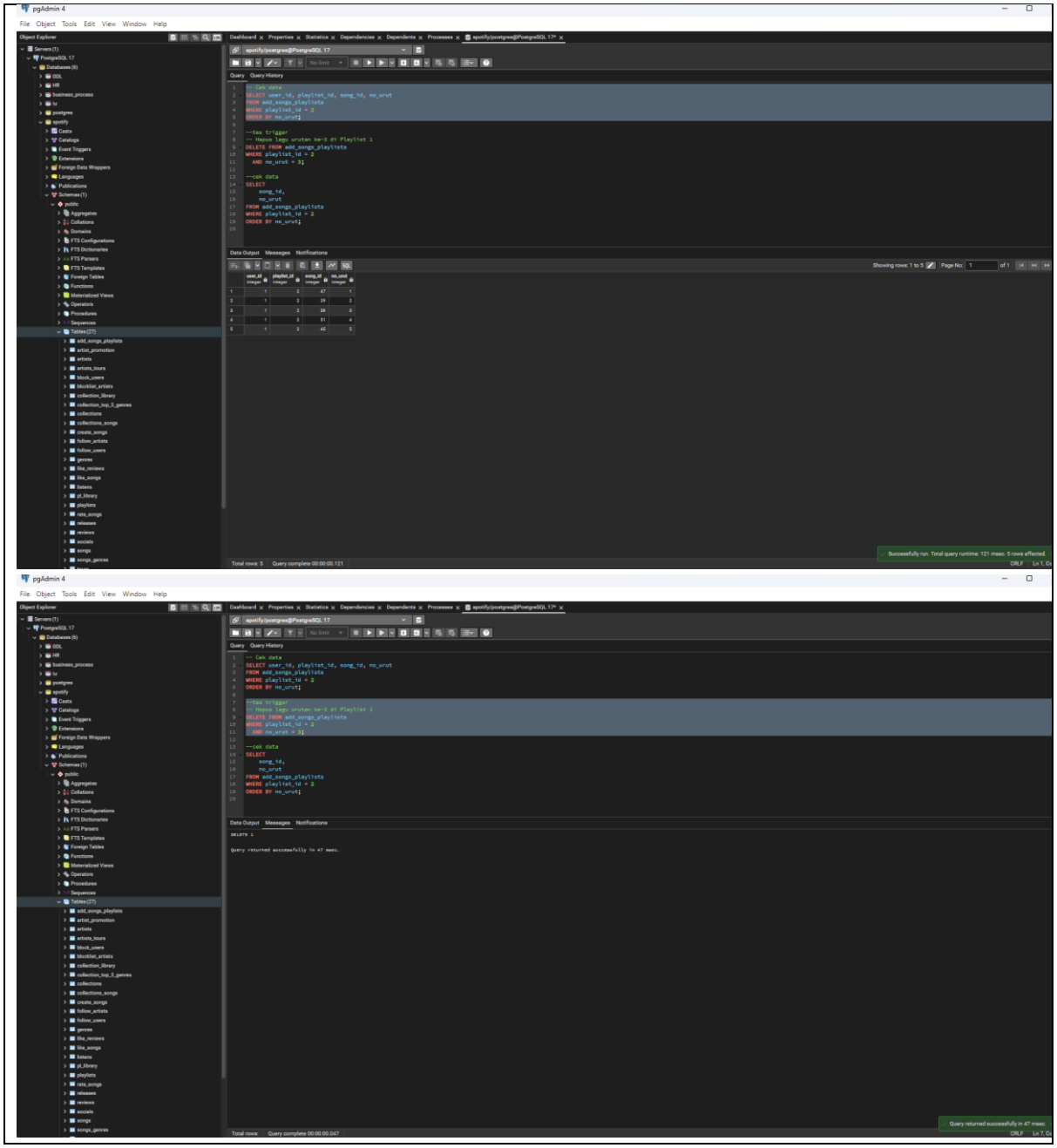
Nama Trigger	reorder_playlist_sequence
Aksi	DELETE
Jenis	AFTER
Deskripsi	Mengubah urutan lagu saat lagu dihapus dari playlist

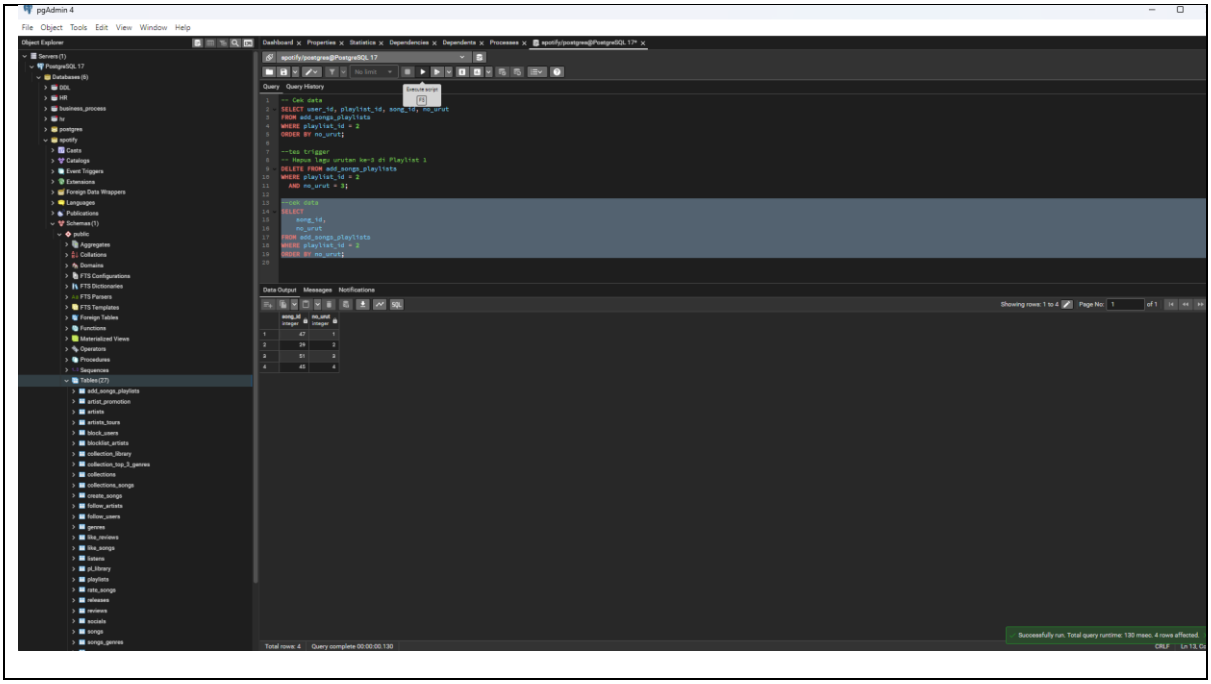
Script SQL

```
-- Cek data

SELECT user_id, playlist_id, song_id, no_urut

FROM add_songs_playlists

WHERE playlist_id = 2

ORDER BY no_urut;



--tes trigger

-- Hapus lagu urutan ke-3 di Playlist 2
```

```
DELETE FROM add_songs_playlists

WHERE playlist_id = 2

  AND no_urut = 3;



--cek data

SELECT

   song_id,

   no_urut

FROM add_songs_playlists

WHERE playlist_id = 2

ORDER BY no_urut;
```

Screenshot Hasil


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Dashboard Properties Statistics Dependencies Dependents Processes spotify/postgres@PostgreSQL 17*
spotify/postgres@PostgreSQL 17
Query Query History
-- Cek data
SELECT user_id, playlist_id, song_id, no_urut
FROM add_songs_playlists
WHERE playlist_id = 2
ORDER BY no_urut;
--tes trigger
-- Hapus lagu urutan ke-3 di Playlist 1
DELETE FROM add_songs_playlists
WHERE playlist_id = 2
AND no_urut = 3;
--cek data
SELECT
song_id,
no_urut
FROM add_songs_playlists
WHERE playlist_id = 2
ORDER BY no_urut;
Data Output Messages Notifications
Showing rows: 1 to 5 Page No: 1 of 1
user_id integer
playlist_id integer
song_id integer
no_urut integer
1 1 2 47 1
2 1 2 29 2
3 1 2 38 3
4 1 2 51 4
5 1 2 45 5
Total rows: 5 Query complete 00:00:00.121
Successfully run. Total query runtime: 121 msec. 5 rows affected.
CRLF Ln 1, Col
pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Dashboard Properties Statistics Dependencies Dependents Processes spotify/postgres@PostgreSQL 17*
spotify/postgres@PostgreSQL 17
Query Query History
-- Cek data
SELECT user_id, playlist_id, song_id, no_urut
FROM add_songs_playlists
WHERE playlist_id = 2
ORDER BY no_urut;
--tes trigger
-- Hapus lagu urutan ke-3 di Playlist 1
DELETE FROM add_songs_playlists
WHERE playlist_id = 2
AND no_urut = 3;
--cek data
SELECT
song_id,
no_urut
FROM add_songs_playlists
WHERE playlist_id = 2
ORDER BY no_urut;
Data Output Messages Notifications
DELETE 1
Query returned successfully in 47 msec.
Total rows: Query complete 00:00:00.047
Query returned successfully in 47 msec.
CRLF Ln 7, Col



Tabel 123 PL/SQL - Trigger trg_enforce_block_logic

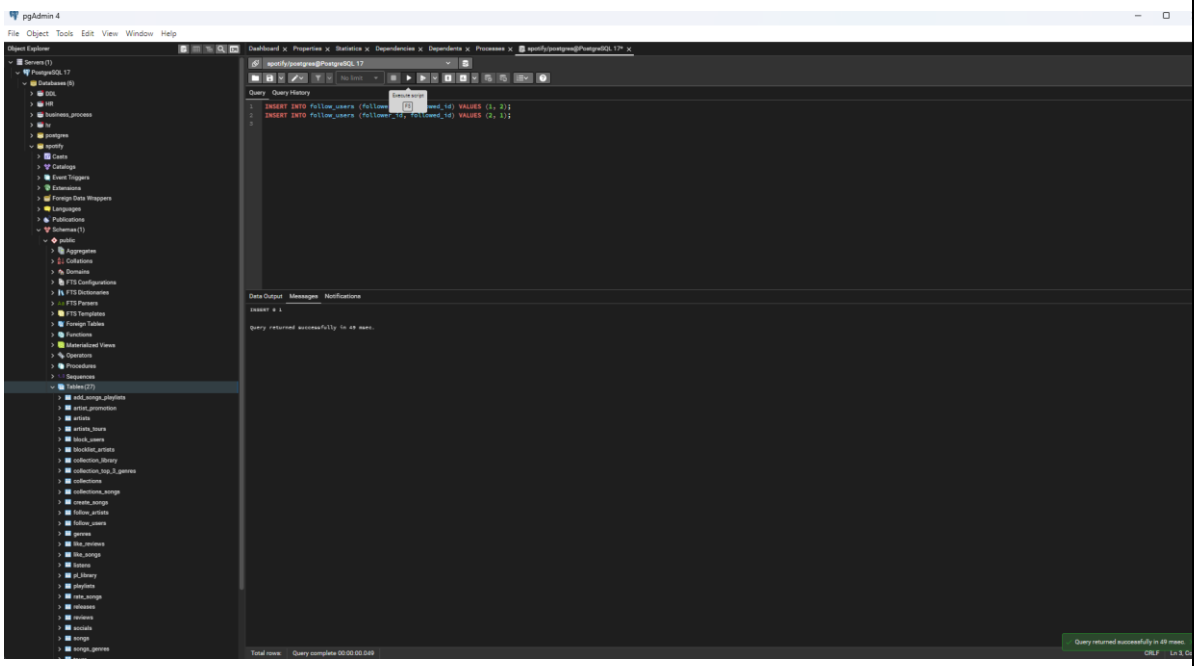
Nama Trigger	enforce_block_logic
Aksi	INSERT
Jenis	AFTER

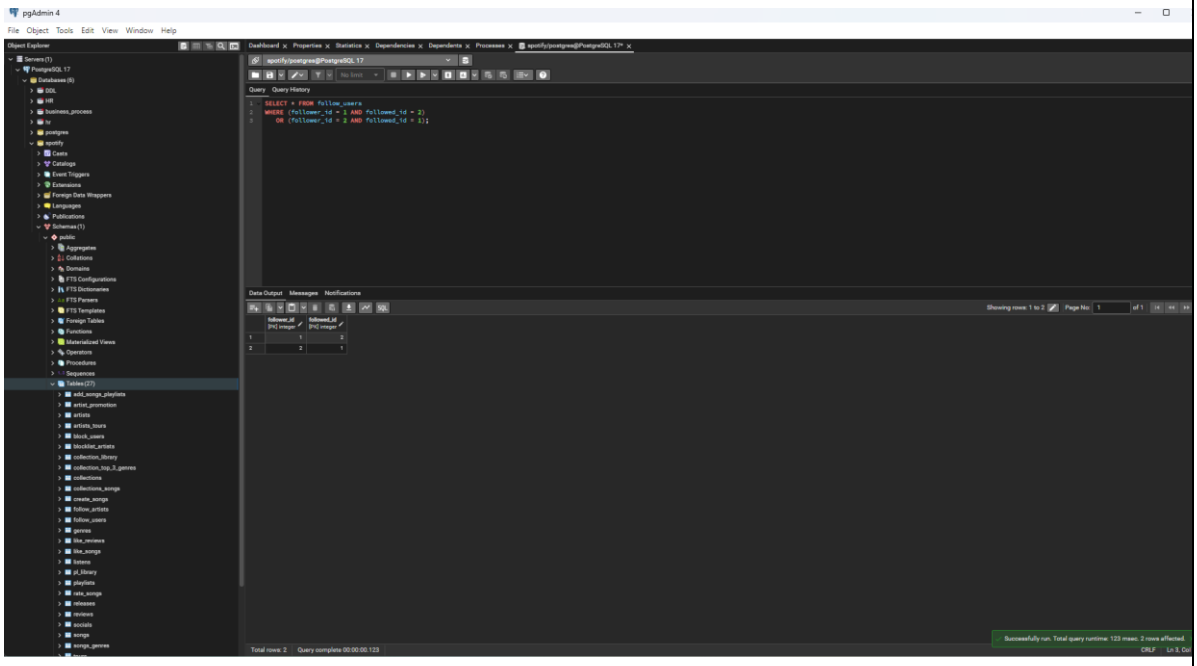
Deskripsi	Otomatis men-unfollow user apabila user mem-block user

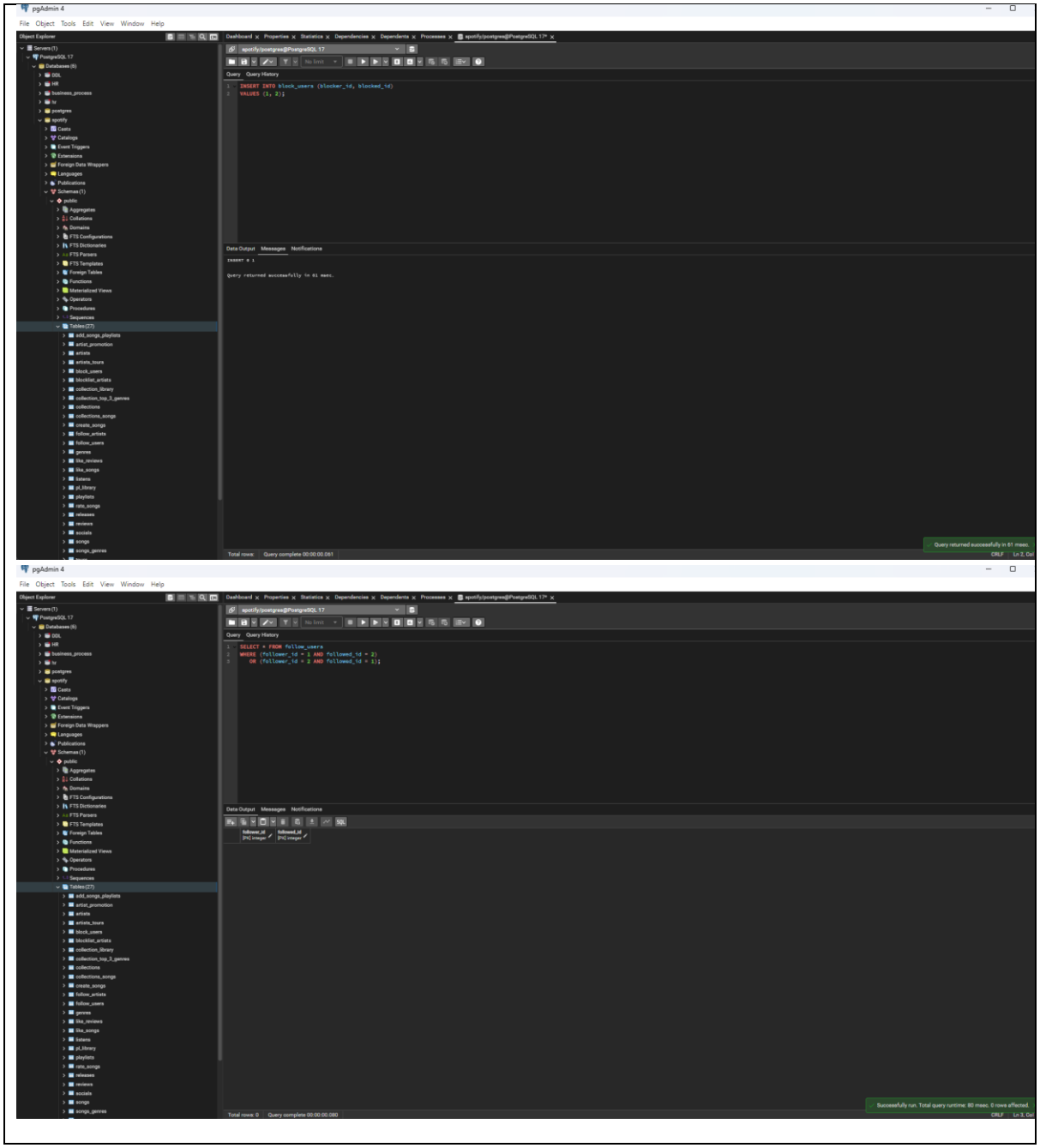
```
--menyiapkan data
INSERT INTO follow_users (follower_id, followed_id) VALUES (1, 2);
INSERT INTO follow_users (follower_id, followed_id) VALUES (2, 1);
--cek
SELECT * FROM follow_users
WHERE (follower_id = 1 AND followed_id = 2) OR (follower_id = 2 AND followed_id = 1);
--trigger
INSERT INTO block_users (blocker_id, blocked_id)
VALUES (1, 2);
--cek
SELECT * FROM follow_users
WHERE (follower_id = 1 AND followed_id = 2)
```

OR (follower_id = 2 AND followed_id = 1);

Screenshot Hasil






pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Dashboard Properties Statistics Dependencies Dependents Processes spotify/postgres@PostgreSQL 17*
spotify/postgres@PostgreSQL 17
Query Query History
INSERT INTO block_users (blocker_id, blocked_id)
VALUES (1, 2);
Data Output Messages Notifications
INSERT 0 1
Query returned successfully in 61 msec.
Total rows: Query complete 00:00:00.061
Query returned successfully in 61 msec.
CRLF Ln 2, Col
pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Dashboard Properties Statistics Dependencies Dependents Processes spotify/postgres@PostgreSQL 17*
spotify/postgres@PostgreSQL 17
Query Query History
SELECT * FROM follow_users
WHERE (follower_id = 1 AND followed_id = 2)
OR (follower_id = 2 AND followed_id = 1);
Data Output Messages Notifications
follower_id [PK] integer
followed_id [PK] integer
Successfully run. Total query runtime: 80 msec. 0 rows affected.
Total rows: 0 Query complete 00:00:00.080
CRLF Ln 3, Col

6. KESIMPULAN

Pada tugas besar ini, kelompok berhasil merancang dan mengimplementasikan sistem basis data untuk layanan *Music Streaming Service* sesuai ruang lingkup perencanaan yang telah ditetapkan. Implementasi meliputi modul inti seperti manajemen pengguna, artis, lagu, koleksi/album, playlist, interaksi sosial antar pengguna, sistem rating–review, promosi artis, serta pencarian terintegrasi. Struktur database yang dibangun telah memenuhi kebutuhan normalisasi, integritas referensial, dan keamanan (termasuk hashing password), serta menerapkan business rule yang ketat untuk menjamin konsistensi.

Fitur-fitur penting yang telah diimplementasikan meliputi:

a. Registrasi dan login pengguna.
b. Sistem follow antar pengguna dan antar artis.
c. Pembuatan review, rating, serta like/unlike review.
d. Pengelolaan playlist, collaborative playlist, dan manajemen track.
e. Manajemen koleksi (album/EP/single/compilation) beserta informasi detail dan tracklist.
f. Manajemen artis, tur artis, dan promosi koleksi.
g. Validasi tanggal prerelease koleksi dan tanggal tur melalui trigger.
h. Fungsi pencarian mencakup lagu, koleksi, playlist, artis, dan pengguna.

Selain progress yang sudah tercapai, masih terdapat beberapa ruang lingkup yang *belum* diimplementasikan dari perencanaan awal, yaitu:

a. Sistem streaming (pencatatan play count, riwayat pemutaran).
b. Sistem rekomendasi berbasis riwayat pengguna atau popularitas.
c. Fitur blocking/blacklist antar pengguna secara penuh (baru dicatat dalam tabel namun belum memiliki prosedur lengkap).
d. Pengelolaan genre secara lanjutan, seperti rekomendasi genre atau kategori trending.
e. Analytics artis, seperti statistik mendalam terkait streaming, tren bulanan, atau data chart otomatis.
f. CRUD admin untuk mengelola konten (lagu, artis, genre, koleksi) secara sistematis.
g. Moderasi konten review (report, hidden review, audit log).

Dalam proses pengerjaan, beberapa kendala utama muncul, seperti penanganan relasi many-to-many yang kompleks, konsistensi tipe data pada fungsi yang memiliki aturan return ketat, desain trigger yang harus mengikuti business rule spesifik, serta penyesuaian query dalam view yang memiliki agregasi yang rumit.

Secara keseluruhan, progress pengerjaan telah memenuhi tujuan utama studi kasus, yaitu menghasilkan rancangan database yang terstruktur, aman, scalable, dan sesuai kebutuhan operasional layanan musik digital. Pekerjaan ini memberikan pembelajaran penting dalam mengintegrasikan desain skema, business rule, implementasi SQL, trigger, view, hingga prosedur/fungsi secara end-to-end.

7. PEMBAGIAN TUGAS

Persentase kontribusi anggota:

Tabel Kontribusi Pengerjaan

Nama	Kontribusi Pengerjaan (%)
Arkan Ramadhan Nugraha	25%
Fauzi Ismail	25%
Nurahma Rahayu	25%
Virli Nasyila Putri	25%

Rincian kontribusi pekerjaan masing-masing anggota:

Tabel Daftar Pengerjaan

Nama	Daftar Pekerjaan
Arkan Ramadhan Nugraha	1. Menentukan Entity-entity dan Relationship-nya dan juga membuat ERD 2. Menetukan Domain Constraint (selain tipe data), seperti unique, not null, check, dll. 3. Membuat semua cronjob 4. Debugging/fixing LDM dan PDM 5. Membuat semua view 6. Ide awal dan kontribusi signifikan di Ruang Lingkup 7. Trigger validate_tour_date()
Fauzi Ismail	1. Membuat, debugging, fixing Logical Data Model. Menentukan seluruh tipe data tiap atribut tabel 2. Membuat, debugging, fixing Physical Data Model(PDM). Menentukan seluruh referential integrity constarint

	3. Membuat trigger user_block dan reorder_playlist 4. Membuat beberapa function/procedure untuk fitur/modul collection top genres 5. Membuat beberapa function/procedure untuk fitur/modul automatic_aggregations 6. Membuat script generate data dummy dan integrasi terhadap API Spotify serta libary faker
Nurahma Rahayu	1. Membuat Conceptual Data Model (CDM) 2. Membuat function dan procedure untuk fitur manajemen user. 3. Membuat function dan procedure untuk fitur collections. 4. Membuat function dan procedure untuk fitur review collection. 5. Membuat function dan procedure untuk fitur manajemen artist. 6. Membuat function dan procedure untuk fitur promotion and tour management. 7. Membuat function dan procedure untuk fitur search system.
Virli Nasyila Putri	1. Sequence (User_id, artist_id, song_id, playlist_id, collection_id, genre_id, review_id, tour_id, social_id, listen_id, add_songs_playlist_id) 2. Membuat function dan procedure untuk fitur Playlist (create_playlist, add_song_to_playlist, remove_song_to_plalist, get_Playlist_detail, get_playlist_tracks) 3. Membuat function dan procedure untuk fitur SONG & LIKE/RATING SYSTEM (get_song_detail_, toggle_like_song, rate_song 4. Membuat funtion dan procedure untuk fitur LISTENING HISTORY (log_listen, dan get_recently_played)

	5. Membuat trigger playlist (fix_playlist_collaborators) 6. Membuat trigger update_artist_follow_count 7. Membuat trigger update_collection_rating 8. Membuat trigger update_collection_top3_genres 9. Membuat trigger update_review_timestamp 10. Membuat trigger update_song_rating 11. Membuat trigger validate_prerelease_date

8. LESSON LEARNED

Arkan Ramadhan Nugraha :

- Proyek ini memberikan pengalaman dalam merancang model data mulai dari ERD sederhana hingga menjadi LDM dan PDM.
- Saya belajar menetukan domain constraint sesuai kebutuhan.
- Saya belajar pentingnya pengorganisasian repository proyek agar pengerjaan database menjadi lebih terstruktur, rapi, dan mudah dikelola.
- Mempelajari penggunaan *view* untuk mempermudah pengambilan data dan mengurangi duplikasi query.
- Mengenal dan mengimplementasikan *cron job* dengan pg_cron untuk penjadwalan tugas otomatis.
- Memahami cara mendefinisikan ruang lingkup proyek agar pengembangan lebih terarah.

Fauzi Ismail :

- Proyek ini memberikan pemahaman bagaimana merancang schema database untuk sebuah aplikasi
- Mempelajari mengenai tipe data dan pentingnya menentukan tipe data yang sesuai bagi schema database/DDL agar optimal dan tidak menghambutkan banyak ruang.
- Mempelajari lebih mendalam mengenai implementasi referential integrity constraints dan hubungannya dengan DML agar menjaga relasi antar tabel serta konsistensi data
- Mempelajari mengenai business rule aplikasi streaming music melalui pembuatan beberapa trigger
- Mempelajari bagaimana view, function/procedure data mengoptimasi query dari sisi keterbacaan

Nurahma Rahayu :

- Proyek ini memberikan pemahaman mendalam tentang perancangan backend berbasis PostgreSQL, mulai dari desain schema, pembuatan function/procedure/trigger, modularisasi fitur hingga error handling.

- Pentingnya menentukan dan memahami struktur data dari tiap tablete, desain output function, dan konsistensi karena bisa berdampak ke pembuatan functionnya.
- Modularisasi fitur mempermudah pengelolaan function dan procedure yang dibutuhkan system, membantu agar kode lebih rapi, mudah dipahami, dan memudahkan pengembangan serta debugging.
- View dapat mempermudah function yang kompleks dan mengurangi duplikasi query.

Virli Nasyila Putri :

- Proyek ini memberikan pengetahuan lebih mendalam mengenai pembangunan backend dengan PostgreSQL, dari penyusunan schema, perancangan function/procedure/trigger.
- Implementasi dari trigger memastikan aturan bisnis berjalan otomatis di level database, menjaga integritas dan konsistensi data .
- Sequence yang konsisten dan jelas penamaannya membantu memastikan primary key unik dan aman saat insert berlangsung secara paralel.
- Perancangan dari sistem rating, like, dan listening history memberikan pemahaman lebih dalam data agregat dan log aktivitas harus diproses secara real-time.
- Pada proyek ini meningkatkan kemampuan dalam membuat query kompleks seperti join, aggregate function, dan looping dalam PL/pgSQL.
- Menambah pemahaman untuk pentingnya melakukan validasi input di dalam procedure, function, maupun trigger untuk menjaga keamanan data dan mencegah inkonsistensi.

Secara keseluruhan, proyek ini memberikan pengalaman komprehensif bagi kelompok dalam merancang dan membangun backend database menggunakan PostgreSQL, dimulai dari penyusunan ERD hingga pengembangan LDM dan PDM yang lengkap. Sepanjang prosesnya, kelompok mempelajari penerapan domain constraint, referential integrity, serta penggunaan function, procedure, trigger, view, dan sequence untuk memastikan aturan bisnis berjalan otomatis, konsisten, dan sesuai kebutuhan sistem. Melalui pengembangan modul seperti playlist, rating, listening history, search system, serta fitur promotion dan tour management, kelompok memperoleh pemahaman menyeluruh tentang bagaimana logika dan alur kerja aplikasi musik dijalankan pada sisi backend. Implementasi pg_cron juga memperkenalkan konsep penjadwalan tugas periodik yang tidak dapat ditangani oleh trigger, sehingga

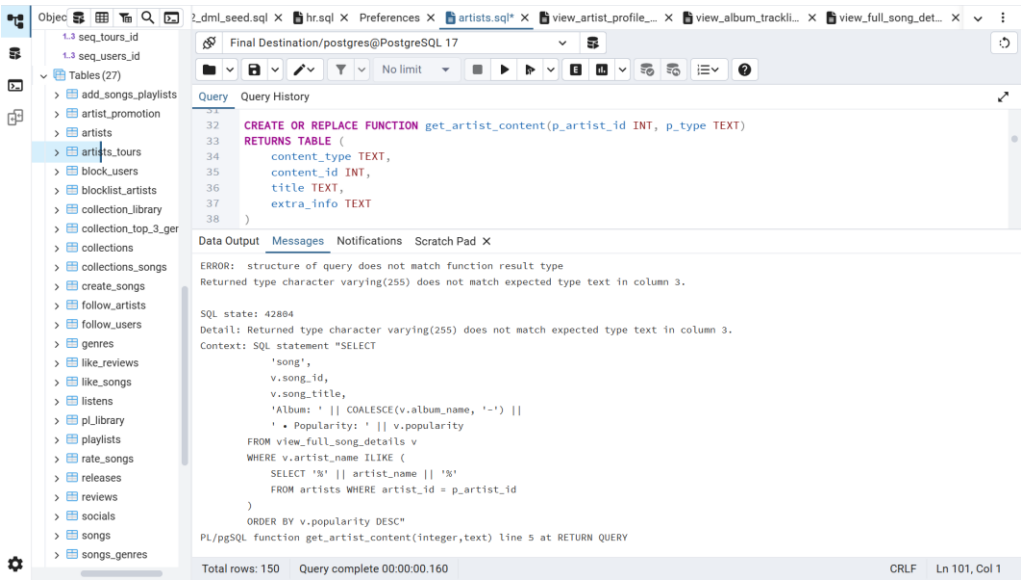
memperluas wawasan mengenai otomasi proses di tingkat database. Selain itu, proyek ini memperkuat kemampuan kelompok dalam menulis query kompleks, melakukan modularisasi fitur, mengelola struktur repository secara terorganisir, serta menyelaraskan ruang lingkup pengembangan agar sistem lebih terstruktur, efisien, dan mudah dikelola di tahap berikutnya.

9. REPOSITORY GITHUB

https://github.com/VirliNasyila/Final_Destination

Dokumentasi Masalah dan Solusi

1. Masalah : tipe data kolom hasil query tidak sama dengan tipe data yang dideklarasikan di RETURN TABLE.

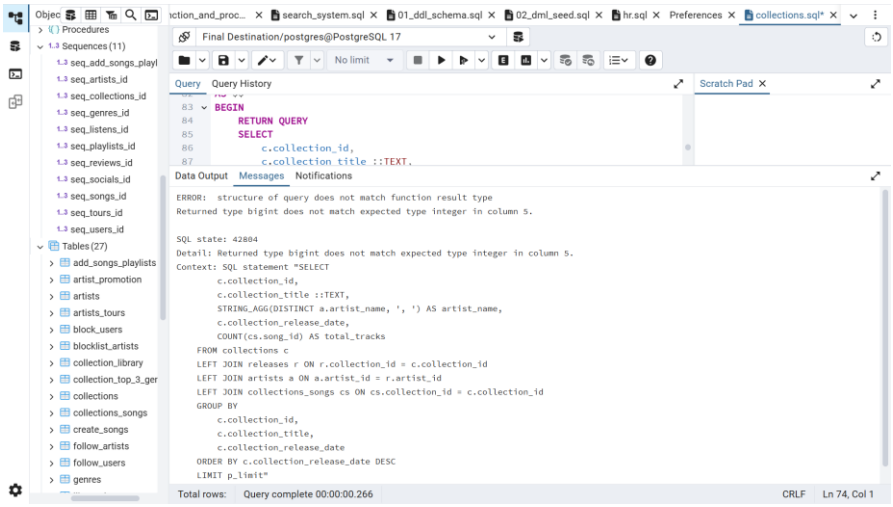


Solusi : Menambahkan cast ::TEXT pada semua kolom yang bukan TEXT

```
-- ========== SONGS ==========
IF p_type = 'songs' THEN
    RETURN QUERY
    SELECT
        'song',
        v.song_id,
        v.song_title ::TEXT,
        'Album: ' || COALESCE(v.album_name, '-') || ' • Popularity: ' || v.popularity
    FROM view_full_song_details v
    WHERE v.artist_name ILIKE (
        SELECT '%' || artist_name || '%'
        FROM artists WHERE artist_id = p_artist_id
    )
    ORDER BY v.popularity DESC;
```

2. Masalah : Function mendefinisikan kolom kelima sebagai INTEGER, tetapi query di dalam RETURN QUERY menghasilkan kolom kelima sebagai BIGINT. Akibatnya

muncul error.

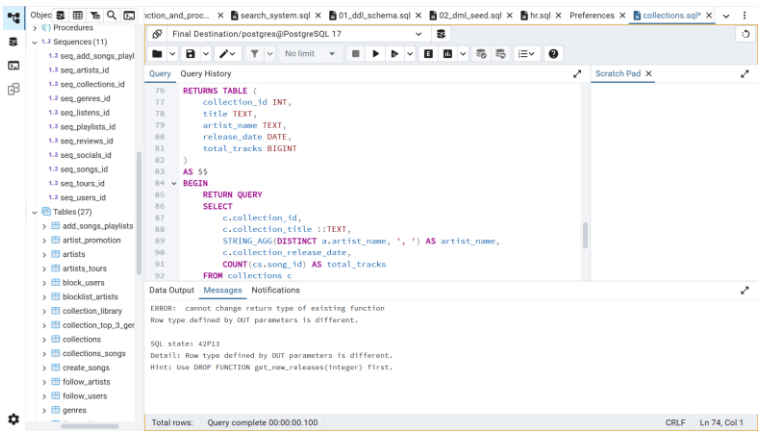


Solusi : Ubah tipe data kolom kelima pada deklarasi RETURNS TABLE menjadi BIGINT

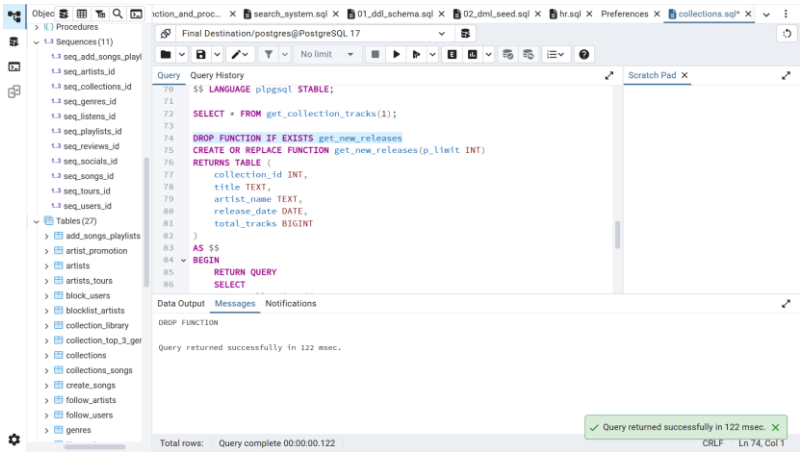
```
CREATE OR REPLACE FUNCTION get_new_releases(p_limit INT)
RETURNS TABLE (
    collection_id INT,
    title TEXT,
    artist_name TEXT,
    release_date DATE,
    total_tracks BIGINT
)
```

3. Masalah : *meng-CREATE OR REPLACE FUNCTION* dengan perubahan tipe data RETURN TABLE

   Namun PostgreSQL tidak mengizinkan mengganti return type function yang sudah ada.



Solusi : Menambahkan DROP FUNCTION/PROCEDURE IF EXISTS lalu CREATE ulang.



4. Masalah: Circular dependency di Artis-Promosi-Collection, Karena promosi promosi butuh artis untuk ada, tetapi artist itu dependent ke promosi. Kesalahan pemilihan dominant entity dalam relationship.
   Solusi: Memilih dominant dalam hubungan 1-to-1 dengan tepat. Berarti, disini Artist yang dominan, karena artis dapat berdiri sendiri, sedangkan promosi butuh artis untuk ada.