Advanced Database Project

Project title

(Spotify)

Project Number

( )

Team Nickname

(رعد)

Team members

Number	ID	Member name	Notes by doctor
1	214022	Ali Mohamed Ali	
2	214029	Mohamed Nasser	
3	214020	Mina Thabet	
4	214021	Sandro Sameh	
5	214065	Mohamed Wael	

Notes by Doctor:

Supervised by

Teacher: Dr. Mohamed Abdelsalam

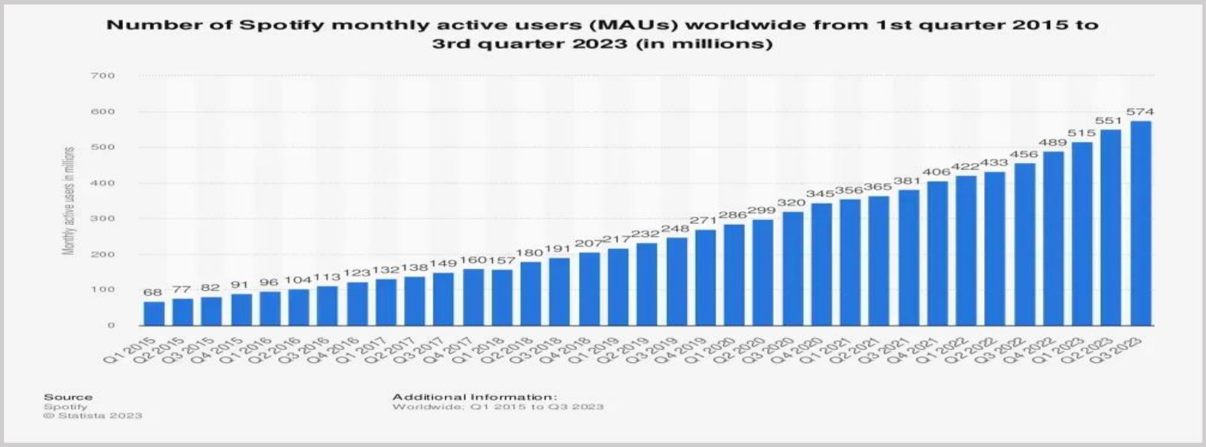
Teacher assistant: Dr. Reham Abdallah

Introduction:

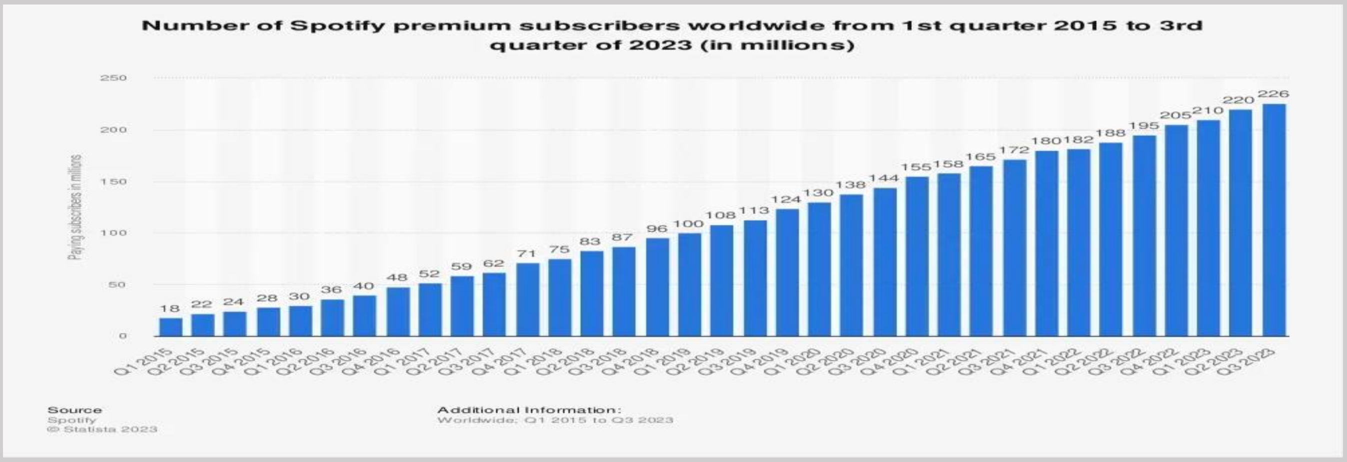
In the digital symphony of modern entertainment, our mission is to compose an unparalleled experience in the form of a groundbreaking music streaming platform, drawing inspiration from the industry luminary, Spotify. In a world where every beat matter, our project aspires to create a harmonious blend of innovation and functionality, offering users a gateway to a rich tapestry of music exploration. At the heart of this endeavor lies a meticulously crafted database system that orchestrates seamless user authentication, dynamic playlist management, advanced music discovery, and an array of features designed to transform the act of listening into an immersive, personalized journey.

Some additional information About Spotify:

1) As of the Third Quarter of 2023, Spotify Has 573 Million Active Users Worldwide.



2) The Service Has 226 Million Premium Subscribers Globally



3)Spotify's 2022 Revenue Was Over 11.7 Billion Euros

4)Spotify is Accessible in over 180 Countries Across the Globe

5)There Are Over 100 Million Tracks on Spotify

6)Over $7 Billion Was Paid to Spotify Artists in 2021

7)There Are Around 4 Billion Playlists on Spotify

8)There Are More Than 5 Million Podcast Titles on Spotify

Business Rules:

The music streaming platform prioritizes user engagement by ensuring a secure and personalized experience through stringent authentication measures and industry-standard encryption. Users enjoy versatile playlist management, including creation, editing, and deletion, with the freedom to add or remove songs at will. The platform enhances music discovery with a comprehensive search system and algorithm-driven personalized recommendations. Seamless music streaming is emphasized, offering intuitive controls for play, pause, skip, and volume adjustment. Users can download selected songs for offline listening, while continuous algorithmic refinement provides personalized recommendations based on listening history. Social integration features foster a vibrant musical community, allowing users to connect, share playlists, and engage in collaborative playlist creation. Detailed artist and album information enriches the overall experience, encouraging user exploration. The platform ensures a unified experience across multiple devices and streamlined account management empowers users to tailor their preferences, from account settings to privacy details.

Our system is designed to be quick, work for lots of people, and not break. It's easy to use, works on different devices and browsers, keeps your data private, and follows all the rules about music copyrights. We also make sure the system is easy to take care of and can handle lots of people using it at the same time. Plus, we use the internet in a smart way to save your data when you're listening to high-quality music. Everything is built to make your music experience the best it can be.

Functional Requirements:

User Authentication:

- Users should be able to create accounts, log in, and securely authenticate themselves.

Playlist Management:

- Users should be able to create, edit, and delete playlists. They should also be able to add or remove songs from playlists.

Music Search and Discovery:

- Users should be able to search for music based on various criteria, such as artist, album, genre, or song title. The system should support music discovery features, such as recommendations and personalized playlists.

Music Streaming:

- Users should be able to stream music seamlessly. The system should support features like play, pause, skip, and adjust volume.

Offline Mode:

- Users should have the option to download music for offline listening.

User Preferences and Recommendations:

- The system should capture user preferences and provide personalized recommendations based on listening history, liked songs, and user-generated playlists.

Social Integration:

- Users should be able to connect with friends, share playlists, and see what others are listening to. Social features may include following users, liking and commenting on playlists, and collaborative playlist creation.

Artist and Album Information:

- Users should have access to detailed information about artists, albums, and songs, including biographies, discographies, and related content.

Multiple Devices Support:

- Users should be able to access their music library and playlists across multiple devices seamlessly.

Account Management:

- Users should be able to manage their account settings, including profile information, subscription details, and privacy settings.

Lyrics Integration:

- Users can access lyrics for songs within the application.

Recommendation Feedback:

- Users can provide feedback on recommended songs to improve the accuracy of future recommendations.

Geolocation-Based Recommendations:

- Utilization of geolocation data to offer music recommendations based on local or regional trends.

Non-Functional Requirements:

Performance:

- The system should provide low-latency response times for music playback, search queries, and other user interactions.

Scalability:

- The database should be scalable to handle a large number of users, songs, and concurrent streaming sessions.

Reliability:

- The system should be highly reliable, with minimal downtime and robust error handling. It should recover gracefully from failures.

Usability:

- The user interface should be intuitive, easy to navigate, and visually appealing. Users should be able to perform common tasks with minimal effort.

Security:

- The system should implement strong security measures to protect user data, including encryption of sensitive information and secure authentication practices.

Compatibility:

- The application should be compatible with a variety of devices, operating systems, and web browsers.

Data Privacy:

- The system should comply with data privacy regulations and ensure that user data is handled with the utmost care.

Maintainability:

- The database and system should be designed for easy maintenance, updates, and enhancements. Documentation should be comprehensive for future development.

Legal Compliance:

- The system should comply with licensing agreements and copyright laws related to music distribution.

Network Bandwidth:

- The system should be optimized to use network bandwidth efficiently, especially for streaming high-quality audio.

Our entities and their attributes

PERSON ENTITY

1) per_id (primary key)

2) first_name

3) last_name

4) gender

5) country

6) age

USER ENTITY

1) user_id (primary key)

2) pref_language

3) e-mail

4) password

5) per_id (foreign key that relates the person table with the user table)

6) sub_id (foreign key that relates the subscription table with the user table)

ARTIST ENTITY

1) artist_id (primary key)

2) nickname

3) no_of_followers

4) per_id (foreign key that relates the person table with the artist table)

SONG ENTITY

1) song_id (primary key)

2) song_name

3) duration

4) date

5) genre

6) BPM (beats per minute)

7) artist_id (foreign key that relates the artist table with the song table)

ALBUM ENTITY

1) album_id (primary_key)

2) no_of_tracks

3) name

4) release_date

5) artist_id (foreign key that relates artist table with the album table)

TRACK ENTITY

1) track_id (primary key)

2) name

3) genre

4) duration

PLAYLIST ENTITY

1) playlist_id (primary key)

2) playlist_name

3) no_of_tracks

SUBSCRIPTION ENTITY

1) sub_id (primary key)

2) status

3) price

4) type

5) start_date

6) end_date

7) renewal_date

PODCAST ENTITY

1) podcast_id (primary key)

2) pd_category

3) pd_title

4) pd_description

5) no_of_episodes

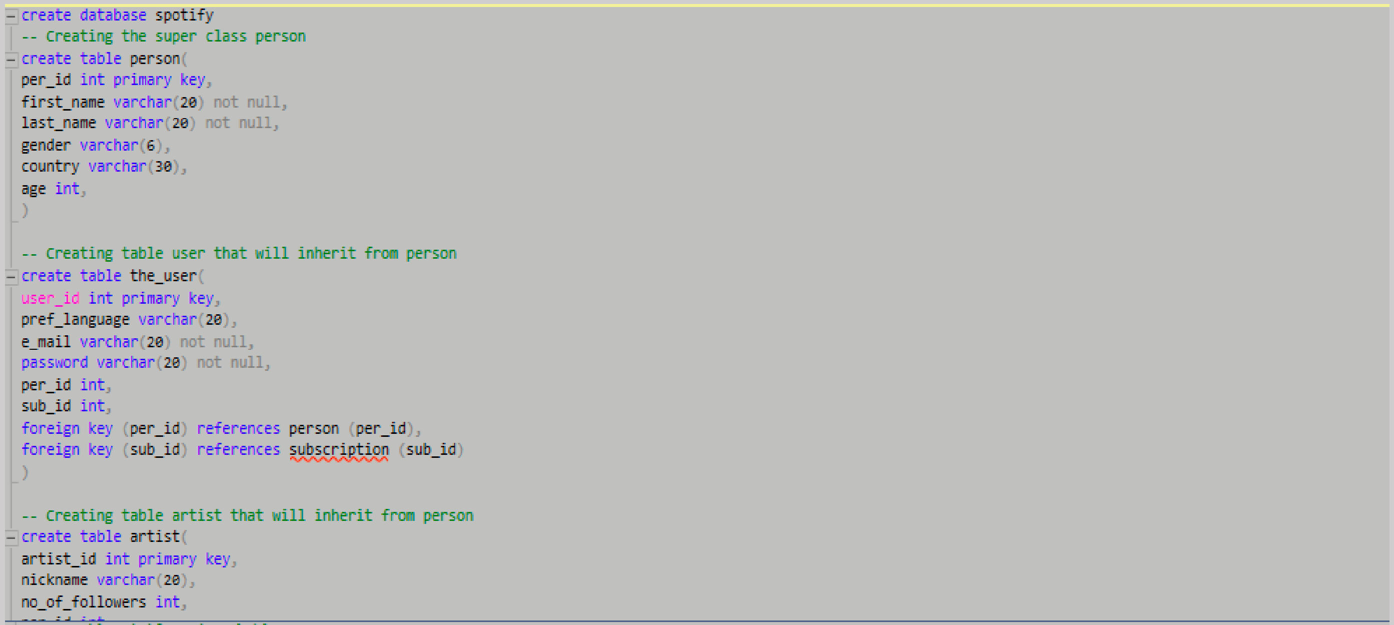
6) pd_duration

7) pd_language

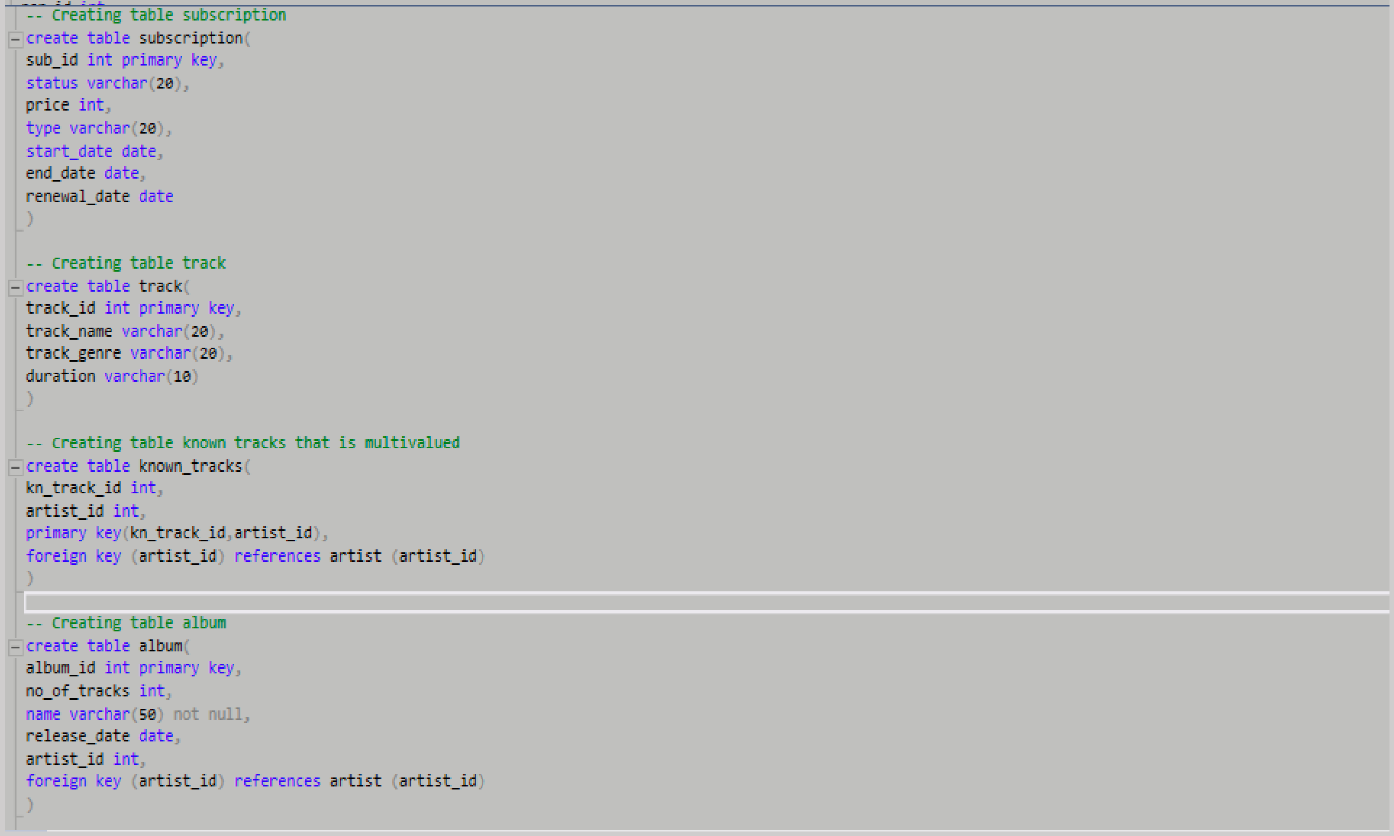
8) pd_host

Implementation:

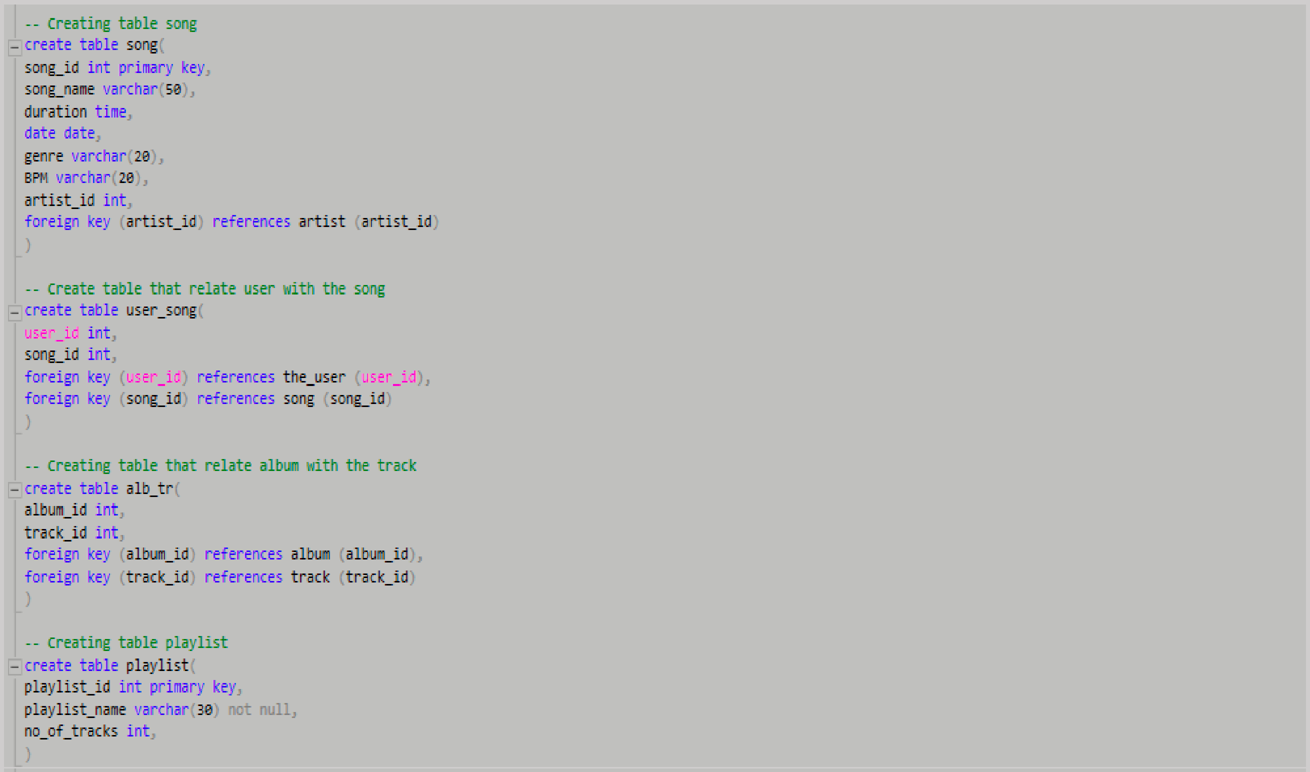
1) Creation of tables:



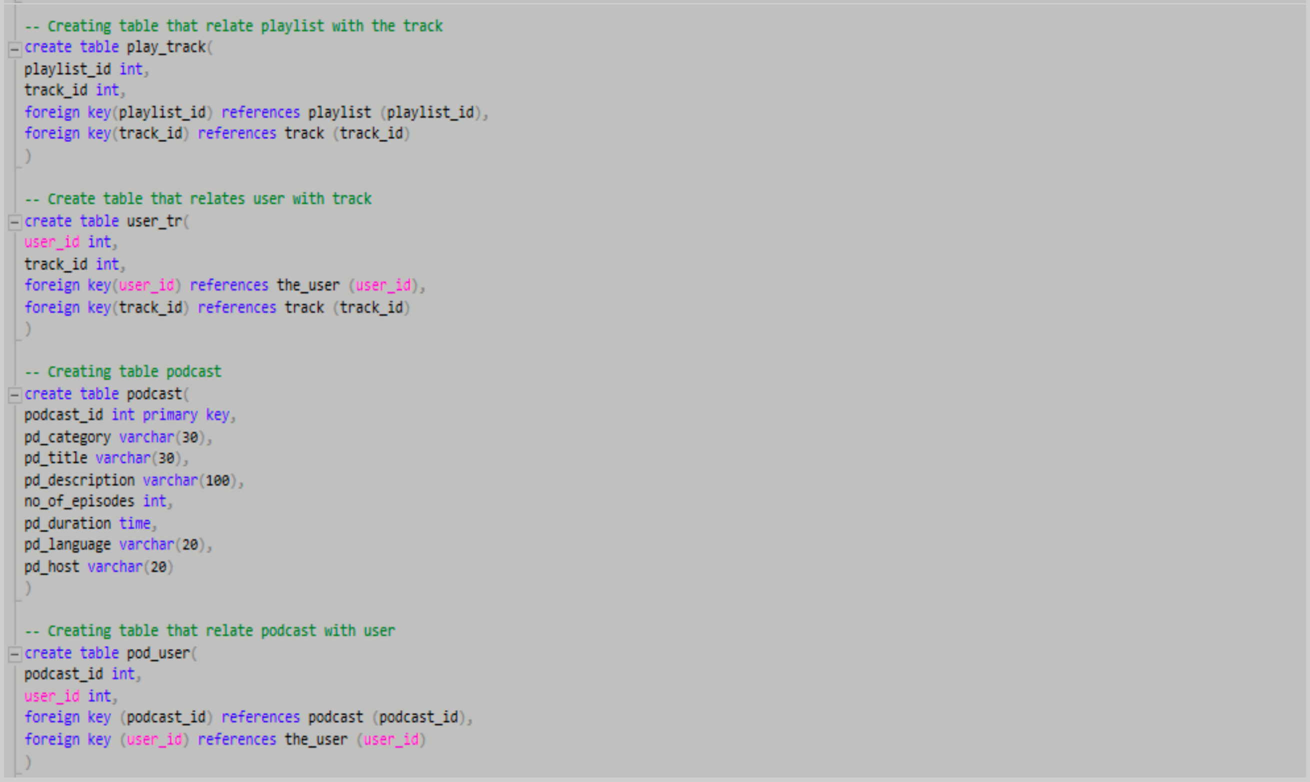
```
create database spotify
-- Creating the super class person
create table person(
per_id int primary key,
first_name varchar(20) not null,
last_name varchar(20) not null,
gender varchar(6),
country varchar(30),
age int,
)

-- Creating table user that will inherit from person
create table the_user(
user_id int primary key,
pref_language varchar(20),
e_mail varchar(20) not null,
password varchar(20) not null,
per_id int,
sub_id int,
foreign key (per_id) references person (per_id),
foreign key (sub_id) references subscription (sub_id)
)

-- Creating table artist that will inherit from person
create table artist(
artist_id int primary key,
nickname varchar(20),
no_of_followers int,
```



```
-- Creating table subscription
create table subscription(
sub_id int primary key,
status varchar(20),
price int,
type varchar(20),
start_date date,
end_date date,
renewal_date date
)

-- Creating table track
create table track(
track_id int primary key,
track_name varchar(20),
track_genre varchar(20),
duration varchar(10)
)

-- Creating table known tracks that is multivalued
create table known_tracks(
kn_track_id int,
artist_id int,
primary key(kn_track_id,artist_id),
foreign key (artist_id) references artist (artist_id)
)

-- Creating table album
create table album(
album_id int primary key,
no_of_tracks int,
name varchar(50) not null,
release_date date,
artist_id int,
foreign key (artist_id) references artist (artist_id)
)
```

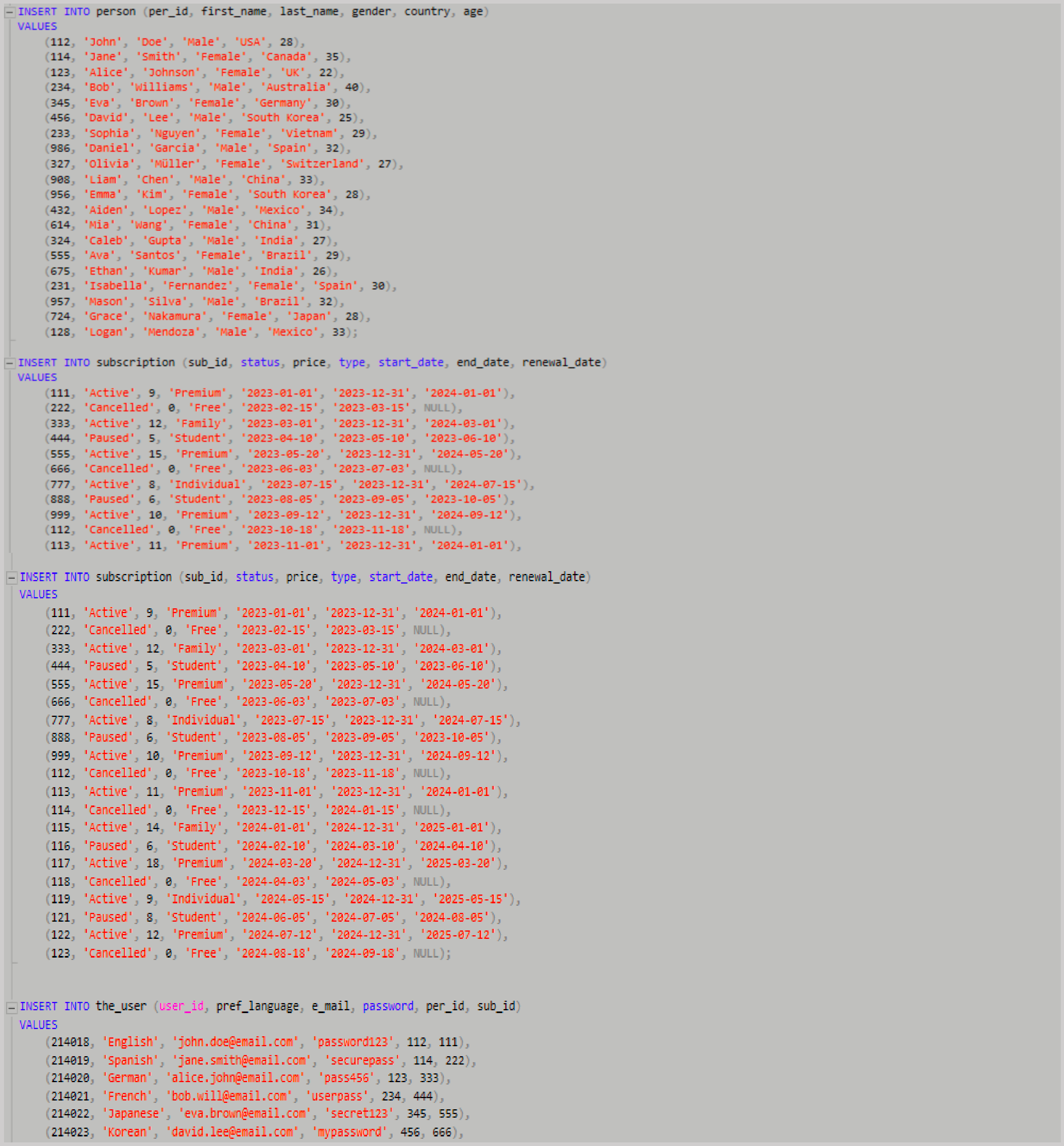


```sql
-- Creating table song
create table song(
song_id int primary key,
song_name varchar(50),
duration time,
date date,
genre varchar(20),
BPM varchar(20),
artist_id int,
foreign key (artist_id) references artist (artist_id)
)

-- Create table that relate user with the song
create table user_song(
user_id int,
song_id int,
foreign key (user_id) references the_user (user_id),
foreign key (song_id) references song (song_id)
)

-- Creating table that relate album with the track
create table alb_tr(
album_id int,
track_id int,
foreign key (album_id) references album (album_id),
foreign key (track_id) references track (track_id)
)

-- Creating table playlist
create table playlist(
playlist_id int primary key,
playlist_name varchar(30) not null,
no_of_tracks int,
)
```

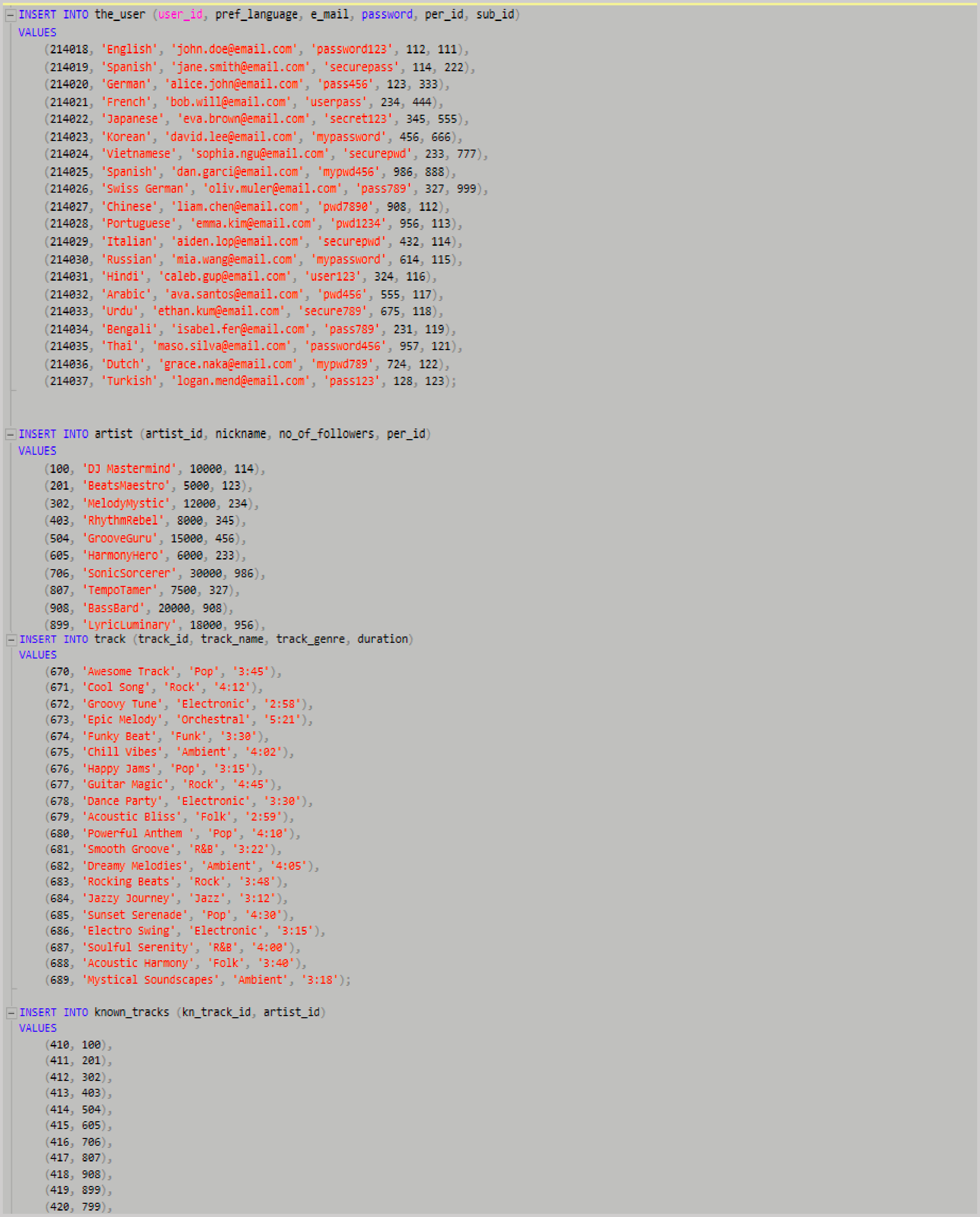


```sql
-- Creating table that relate playlist with the track
create table play_track(
playlist_id int,
track_id int,
foreign key(playlist_id) references playlist (playlist_id),
foreign key(track_id) references track (track_id)
)

-- Create table that relates user with track
create table user_tr(
user_id int,
track_id int,
foreign key(user_id) references the_user (user_id),
foreign key(track_id) references track (track_id)
)

-- Creating table podcast
create table podcast(
podcast_id int primary key,
pd_category varchar(30),
pd_title varchar(30),
pd_description varchar(100),
no_of_episodes int,
pd_duration time,
pd_language varchar(20),
pd_host varchar(20)
)

-- Creating table that relate podcast with user
create table pod_user(
podcast_id int,
user_id int,
foreign key (podcast_id) references podcast (podcast_id),
foreign key (user_id) references the_user (user_id)
)
```

2) Insertion of data to our table:



```sql
INSERT INTO person (per_id, first_name, last_name, gender, country, age)
VALUES
    (112, 'John', 'Doe', 'Male', 'USA', 28),
    (114, 'Jane', 'Smith', 'Female', 'Canada', 35),
    (123, 'Alice', 'Johnson', 'Female', 'UK', 22),
    (234, 'Bob', 'Williams', 'Male', 'Australia', 40),
    (345, 'Eva', 'Brown', 'Female', 'Germany', 30),
    (456, 'David', 'Lee', 'Male', 'South Korea', 25),
    (233, 'Sophia', 'Nguyen', 'Female', 'Vietnam', 29),
    (986, 'Daniel', 'Garcia', 'Male', 'Spain', 32),
    (327, 'Olivia', 'Müller', 'Female', 'Switzerland', 27),
    (908, 'Liam', 'Chen', 'Male', 'China', 33),
    (956, 'Emma', 'Kim', 'Female', 'South Korea', 28),
    (432, 'Aiden', 'Lopez', 'Male', 'Mexico', 34),
    (614, 'Mia', 'Wang', 'Female', 'China', 31),
    (324, 'Caleb', 'Gupta', 'Male', 'India', 27),
    (555, 'Ava', 'Santos', 'Female', 'Brazil', 29),
    (675, 'Ethan', 'Kumar', 'Male', 'India', 26),
    (231, 'Isabella', 'Fernandez', 'Female', 'Spain', 30),
    (957, 'Mason', 'Silva', 'Male', 'Brazil', 32),
    (724, 'Grace', 'Nakamura', 'Female', 'Japan', 28),
    (128, 'Logan', 'Mendoza', 'Male', 'Mexico', 33);

INSERT INTO subscription (sub_id, status, price, type, start_date, end_date, renewal_date)
VALUES
    (111, 'Active', 9, 'Premium', '2023-01-01', '2023-12-31', '2024-01-01'),
    (222, 'Cancelled', 0, 'Free', '2023-02-15', '2023-03-15', NULL),
    (333, 'Active', 12, 'Family', '2023-03-01', '2023-12-31', '2024-03-01'),
    (444, 'Paused', 5, 'Student', '2023-04-10', '2023-05-10', '2023-06-10'),
    (555, 'Active', 15, 'Premium', '2023-05-20', '2023-12-31', '2024-05-20'),
    (666, 'Cancelled', 0, 'Free', '2023-06-03', '2023-07-03', NULL),
    (777, 'Active', 8, 'Individual', '2023-07-15', '2023-12-31', '2024-07-15'),
    (888, 'Paused', 6, 'Student', '2023-08-05', '2023-09-05', '2023-10-05'),
    (999, 'Active', 10, 'Premium', '2023-09-12', '2023-12-31', '2024-09-12'),
    (112, 'Cancelled', 0, 'Free', '2023-10-18', '2023-11-18', NULL),
    (113, 'Active', 11, 'Premium', '2023-11-01', '2023-12-31', '2024-01-01'),

INSERT INTO subscription (sub_id, status, price, type, start_date, end_date, renewal_date)
VALUES
    (111, 'Active', 9, 'Premium', '2023-01-01', '2023-12-31', '2024-01-01'),
    (222, 'Cancelled', 0, 'Free', '2023-02-15', '2023-03-15', NULL),
    (333, 'Active', 12, 'Family', '2023-03-01', '2023-12-31', '2024-03-01'),
    (444, 'Paused', 5, 'Student', '2023-04-10', '2023-05-10', '2023-06-10'),
    (555, 'Active', 15, 'Premium', '2023-05-20', '2023-12-31', '2024-05-20'),
    (666, 'Cancelled', 0, 'Free', '2023-06-03', '2023-07-03', NULL),
    (777, 'Active', 8, 'Individual', '2023-07-15', '2023-12-31', '2024-07-15'),
    (888, 'Paused', 6, 'Student', '2023-08-05', '2023-09-05', '2023-10-05'),
    (999, 'Active', 10, 'Premium', '2023-09-12', '2023-12-31', '2024-09-12'),
    (112, 'Cancelled', 0, 'Free', '2023-10-18', '2023-11-18', NULL),
    (113, 'Active', 11, 'Premium', '2023-11-01', '2023-12-31', '2024-01-01'),
    (114, 'Cancelled', 0, 'Free', '2023-12-15', '2024-01-15', NULL),
    (115, 'Active', 14, 'Family', '2024-01-01', '2024-12-31', '2025-01-01'),
    (116, 'Paused', 6, 'Student', '2024-02-10', '2024-03-10', '2024-04-10'),
    (117, 'Active', 18, 'Premium', '2024-03-20', '2024-12-31', '2025-03-20'),
    (118, 'Cancelled', 0, 'Free', '2024-04-03', '2024-05-03', NULL),
    (119, 'Active', 9, 'Individual', '2024-05-15', '2024-12-31', '2025-05-15'),
    (121, 'Paused', 8, 'Student', '2024-06-05', '2024-07-05', '2024-08-05'),
    (122, 'Active', 12, 'Premium', '2024-07-12', '2024-12-31', '2025-07-12'),
    (123, 'Cancelled', 0, 'Free', '2024-08-18', '2024-09-18', NULL);


INSERT INTO the_user (user_id, pref_language, e_mail, password, per_id, sub_id)
VALUES
    (214018, 'English', 'john.doe@email.com', 'password123', 112, 111),
    (214019, 'Spanish', 'jane.smith@email.com', 'securepass', 114, 222),
    (214020, 'German', 'alice.john@email.com', 'pass456', 123, 333),
    (214021, 'French', 'bob.will@email.com', 'userpass', 234, 444),
    (214022, 'Japanese', 'eva.brown@email.com', 'secret123', 345, 555),
    (214023, 'Korean', 'david.lee@email.com', 'mypassword', 456, 666),
```

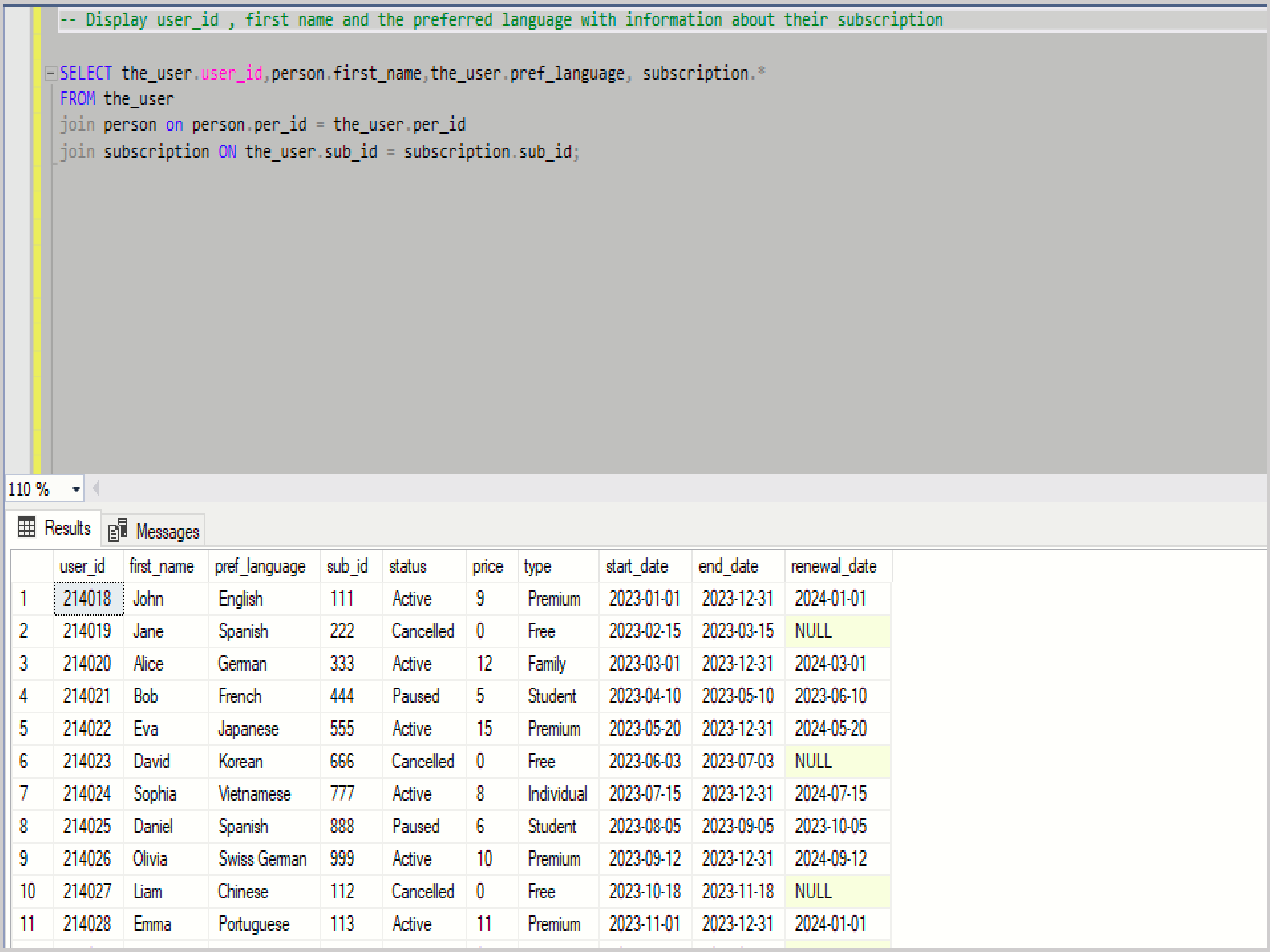


```sql
INSERT INTO the_user (user_id, pref_language, e_mail, password, per_id, sub_id)
VALUES
    (214018, 'English', 'john.doe@email.com', 'password123', 112, 111),
    (214019, 'Spanish', 'jane.smith@email.com', 'securepass', 114, 222),
    (214020, 'German', 'alice.john@email.com', 'pass456', 123, 333),
    (214021, 'French', 'bob.will@email.com', 'userpass', 234, 444),
    (214022, 'Japanese', 'eva.brown@email.com', 'secret123', 345, 555),
    (214023, 'Korean', 'david.lee@email.com', 'mypassword', 456, 666),
    (214024, 'Vietnamese', 'sophia.ngu@email.com', 'securepwd', 233, 777),
    (214025, 'Spanish', 'dan.garci@email.com', 'mypwd456', 986, 888),
    (214026, 'Swiss German', 'oliv.muler@email.com', 'pass789', 327, 999),
    (214027, 'Chinese', 'liam.chen@email.com', 'pwd7890', 908, 112),
    (214028, 'Portuguese', 'emma.kim@email.com', 'pwd1234', 956, 113),
    (214029, 'Italian', 'aiden.lop@email.com', 'securepwd', 432, 114),
    (214030, 'Russian', 'mia.wang@email.com', 'mypassword', 614, 115),
    (214031, 'Hindi', 'caleb.gup@email.com', 'user123', 324, 116),
    (214032, 'Arabic', 'ava.santos@email.com', 'pwd456', 555, 117),
    (214033, 'Urdu', 'ethan.kum@email.com', 'secure789', 675, 118),
    (214034, 'Bengali', 'isabel.fer@email.com', 'pass789', 231, 119),
    (214035, 'Thai', 'maso.silva@email.com', 'password456', 957, 121),
    (214036, 'Dutch', 'grace.naka@email.com', 'mypwd789', 724, 122),
    (214037, 'Turkish', 'logan.mend@email.com', 'pass123', 128, 123);


INSERT INTO artist (artist_id, nickname, no_of_followers, per_id)
VALUES
    (100, 'DJ Mastermind', 10000, 114),
    (201, 'BeatsMaestro', 5000, 123),
    (302, 'MelodyMystic', 12000, 234),
    (403, 'RhythmRebel', 8000, 345),
    (504, 'GrooveGuru', 15000, 456),
    (605, 'HarmonyHero', 6000, 233),
    (706, 'SonicSorcerer', 30000, 986),
    (807, 'TempoTamer', 7500, 327),
    (908, 'BassBard', 20000, 908),
    (899, 'LyricLuminary', 18000, 956),
INSERT INTO track (track_id, track_name, track_genre, duration)
VALUES
    (670, 'Awesome Track', 'Pop', '3:45'),
    (671, 'Cool Song', 'Rock', '4:12'),
    (672, 'Groovy Tune', 'Electronic', '2:58'),
    (673, 'Epic Melody', 'Orchestral', '5:21'),
    (674, 'Funky Beat', 'Funk', '3:30'),
    (675, 'Chill Vibes', 'Ambient', '4:02'),
    (676, 'Happy Jams', 'Pop', '3:15'),
    (677, 'Guitar Magic', 'Rock', '4:45'),
    (678, 'Dance Party', 'Electronic', '3:30'),
    (679, 'Acoustic Bliss', 'Folk', '2:59'),
    (680, 'Powerful Anthem ', 'Pop', '4:10'),
    (681, 'Smooth Groove', 'R&B', '3:22'),
    (682, 'Dreamy Melodies', 'Ambient', '4:05'),
    (683, 'Rocking Beats', 'Rock', '3:48'),
    (684, 'Jazzy Journey', 'Jazz', '3:12'),
    (685, 'Sunset Serenade', 'Pop', '4:30'),
    (686, 'Electro Swing', 'Electronic', '3:15'),
    (687, 'Soulful Serenity', 'R&B', '4:00'),
    (688, 'Acoustic Harmony', 'Folk', '3:40'),
    (689, 'Mystical Soundscapes', 'Ambient', '3:18');

INSERT INTO known_tracks (kn_track_id, artist_id)
VALUES
    (410, 100),
    (411, 201),
    (412, 302),
    (413, 403),
    (414, 504),
    (415, 605),
    (416, 706),
    (417, 807),
    (418, 908),
    (419, 899),
    (420, 799),
```

3) Generating our reports

- **<u>We want to see user's details like user's id , first name and the preferred language with their subscription details.</u>**

- **We selected the user's details we wanted to see and joined between the person and the user table and then we joined the user with the subscription table to display the user's details with the details of the subscription.**



```
-- Display user_id , first name and the preferred language with information about their subscription

SELECT the_user.user_id,person.first_name,the_user.pref_language, subscription.*
FROM the_user
join person on person.per_id = the_user.per_id
join subscription ON the_user.sub_id = subscription.sub_id;
```

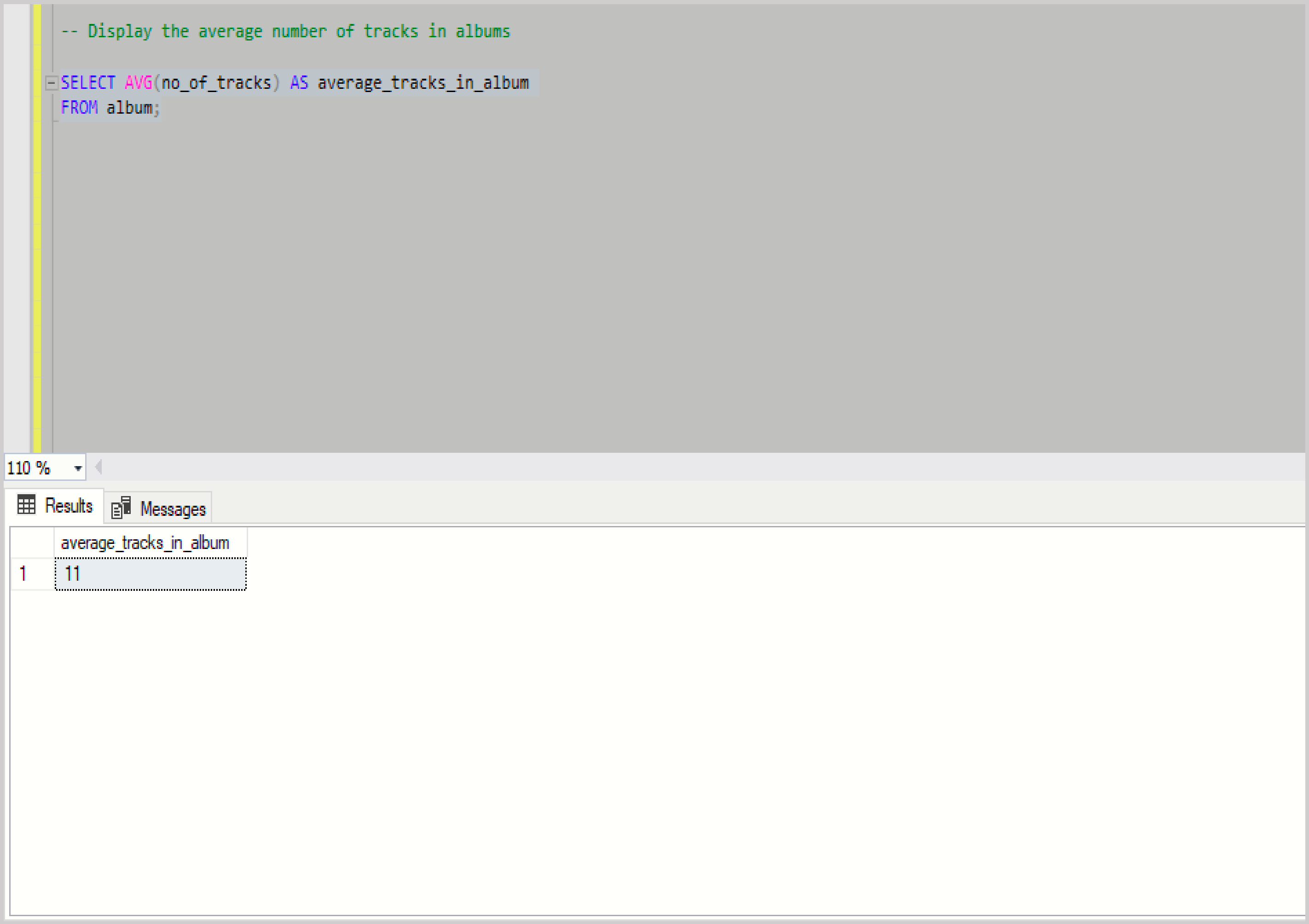
110 %

Results | Messages

	user_id	first_name	pref_language	sub_id	status	price	type	start_date	end_date	renewal_date
1	214018	John	English	111	Active	9	Premium	2023-01-01	2023-12-31	2024-01-01
2	214019	Jane	Spanish	222	Cancelled	0	Free	2023-02-15	2023-03-15	NULL
3	214020	Alice	German	333	Active	12	Family	2023-03-01	2023-12-31	2024-03-01
4	214021	Bob	French	444	Paused	5	Student	2023-04-10	2023-05-10	2023-06-10
5	214022	Eva	Japanese	555	Active	15	Premium	2023-05-20	2023-12-31	2024-05-20
6	214023	David	Korean	666	Cancelled	0	Free	2023-06-03	2023-07-03	NULL
7	214024	Sophia	Vietnamese	777	Active	8	Individual	2023-07-15	2023-12-31	2024-07-15
8	214025	Daniel	Spanish	888	Paused	6	Student	2023-08-05	2023-09-05	2023-10-05
9	214026	Olivia	Swiss German	999	Active	10	Premium	2023-09-12	2023-12-31	2024-09-12
10	214027	Liam	Chinese	112	Cancelled	0	Free	2023-10-18	2023-11-18	NULL
11	214028	Emma	Portuguese	113	Active	11	Premium	2023-11-01	2023-12-31	2024-01-01

- <u>**We want to know the average number of tracks in all albums**</u>

- **We selected the number of tracks from the album table and we calculated the average of them using the aggregate function "AVG" and we named the column of the result "average_tracks_in_albums"**

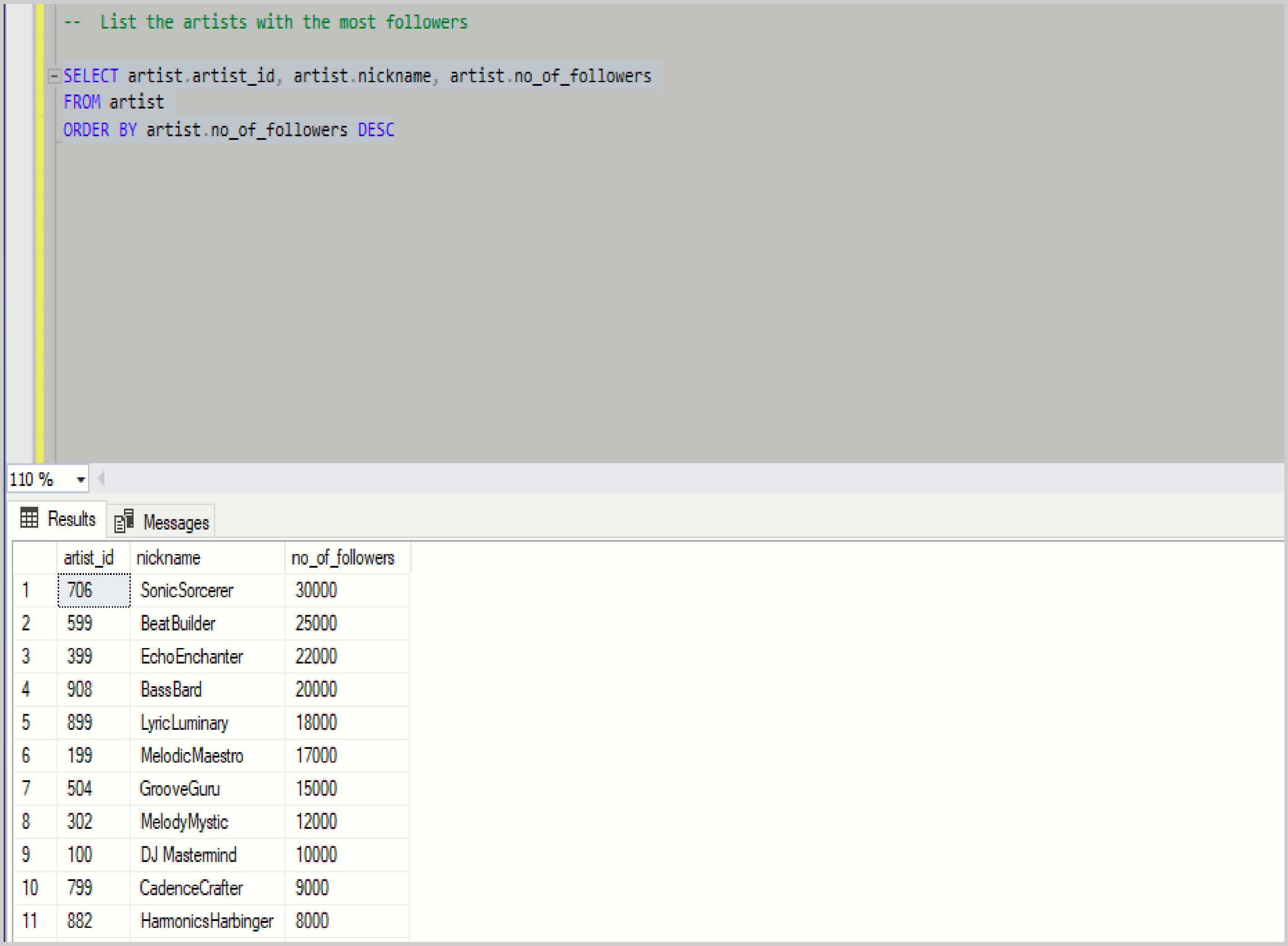


```
-- Display the average number of tracks in albums

SELECT AVG(no_of_tracks) AS average_tracks_in_album
FROM album;
```

```
   average_tracks_in_album
1  11
```

- **<u>We want to know the artists with the most followers</u>**

- **We selected some information about the artist like the artist id, nickname and the number of followers and then we ordered by the number of followers descendingly to see the artists with most followers.**

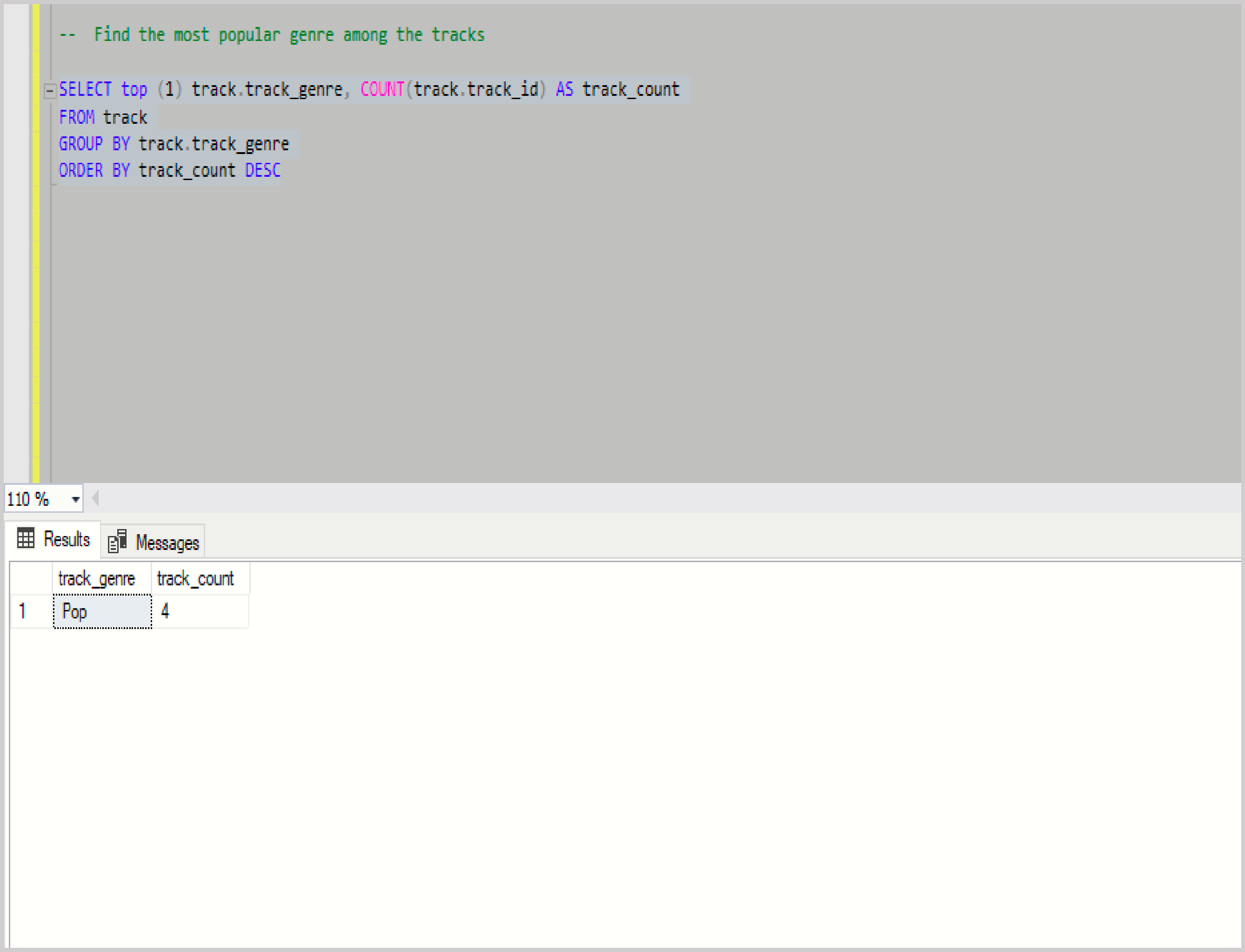


```
-- List the artists with the most followers

SELECT artist.artist_id, artist.nickname, artist.no_of_followers
FROM artist
ORDER BY artist.no_of_followers DESC
```

	artist_id	nickname	no_of_followers
1	706	SonicSorcerer	30000
2	599	BeatBuilder	25000
3	399	EchoEnchanter	22000
4	908	BassBard	20000
5	899	LyricLuminary	18000
6	199	MelodicMaestro	17000
7	504	GrooveGuru	15000
8	302	MelodyMystic	12000
9	100	DJ Mastermind	10000
10	799	CadenceCrafter	9000
11	882	HarmonicsHarbinger	8000

- **<u>We want to know the most popular genre among the tracks</u>**

- **We selected the top 1 genre and we used the aggregate function "COUNT" to know the frequency of the most repeated genre.**

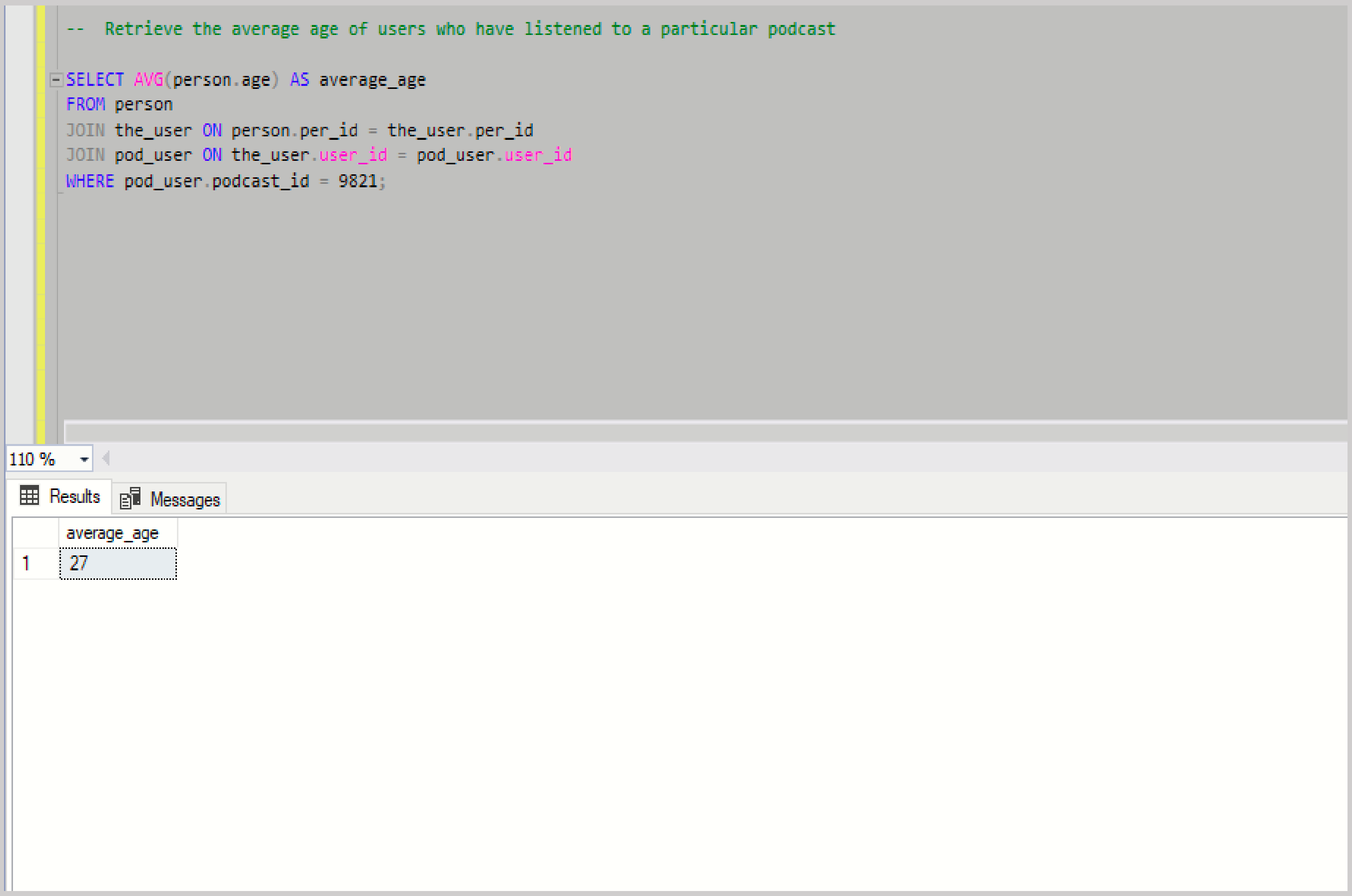


```sql
--  Find the most popular genre among the tracks

SELECT top (1) track.track_genre, COUNT(track.track_id) AS track_count
FROM track
GROUP BY track.track_genre
ORDER BY track_count DESC
```

	track_genre	track_count
1	Pop	4

- **<u>We want to know the average age of the users who have listened the podcast number 9821.</u>**

- **We selected the person's age and calculated the average of it from the table person and then we joined it with the user table and we joined the user with the podcast table. Then we have put a condition to specify which podcast we scope on.**

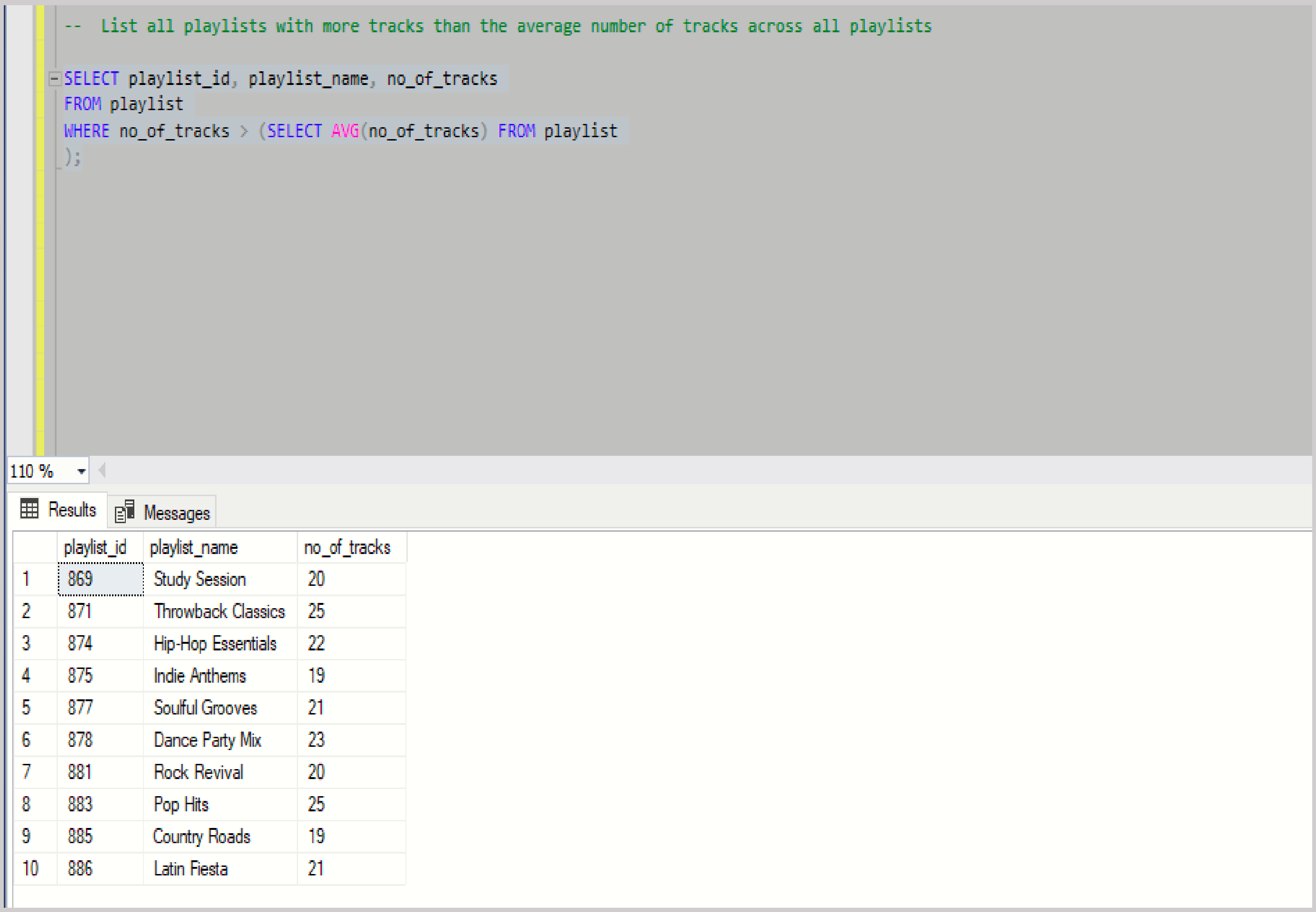


```
--  Retrieve the average age of users who have listened to a particular podcast

SELECT AVG(person.age) AS average_age
FROM person
JOIN the_user ON person.per_id = the_user.per_id
JOIN pod_user ON the_user.user_id = pod_user.user_id
WHERE pod_user.podcast_id = 9821;
```

```
   average_age
1  27
```

- **<u>We want to display all playlists that have tracks more than the average number of tracks in all playlists.</u>**

- **We selected the details of playlists from the playlist table and then we conditioned that number of tracks is more than average number of tracks in all playlists. (We used nested query).**



```
--  List all playlists with more tracks than the average number of tracks across all playlists

SELECT playlist_id, playlist_name, no_of_tracks
FROM playlist
WHERE no_of_tracks > (SELECT AVG(no_of_tracks) FROM playlist
);
```

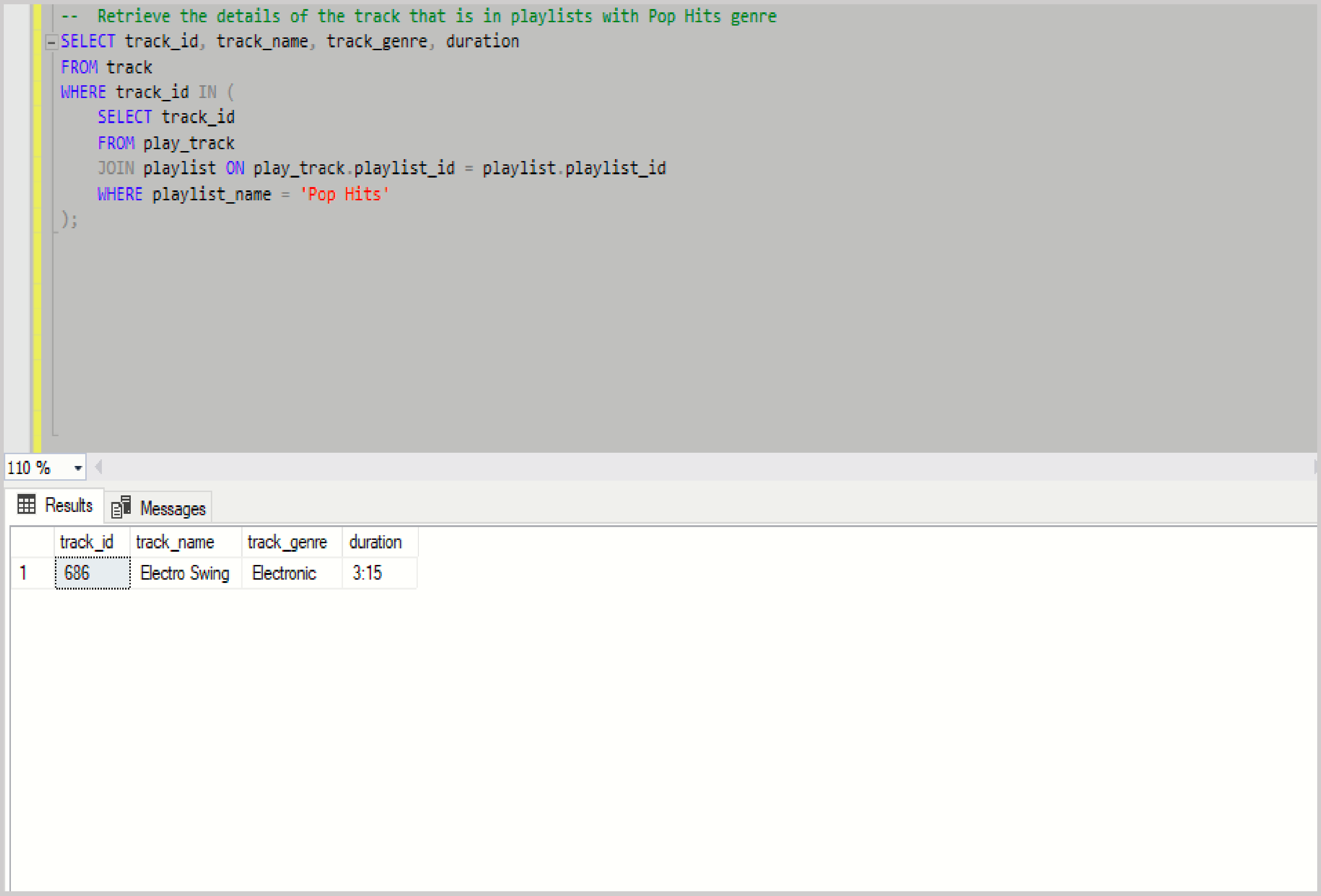
110 %

Results | Messages

	playlist_id	playlist_name	no_of_tracks
1	869	Study Session	20
2	871	Throwback Classics	25
3	874	Hip-Hop Essentials	22
4	875	Indie Anthems	19
5	877	Soulful Grooves	21
6	878	Dance Party Mix	23
7	881	Rock Revival	20
8	883	Pop Hits	25
9	885	Country Roads	19
10	886	Latin Fiesta	21

- <u>**We want to show the details of the track that is in the playlist named Pop Hit**</u>

- **We selected the details of the track from the track table and we conditioned the track id to be in the playlist named Pop Hits. (We used nested query).**

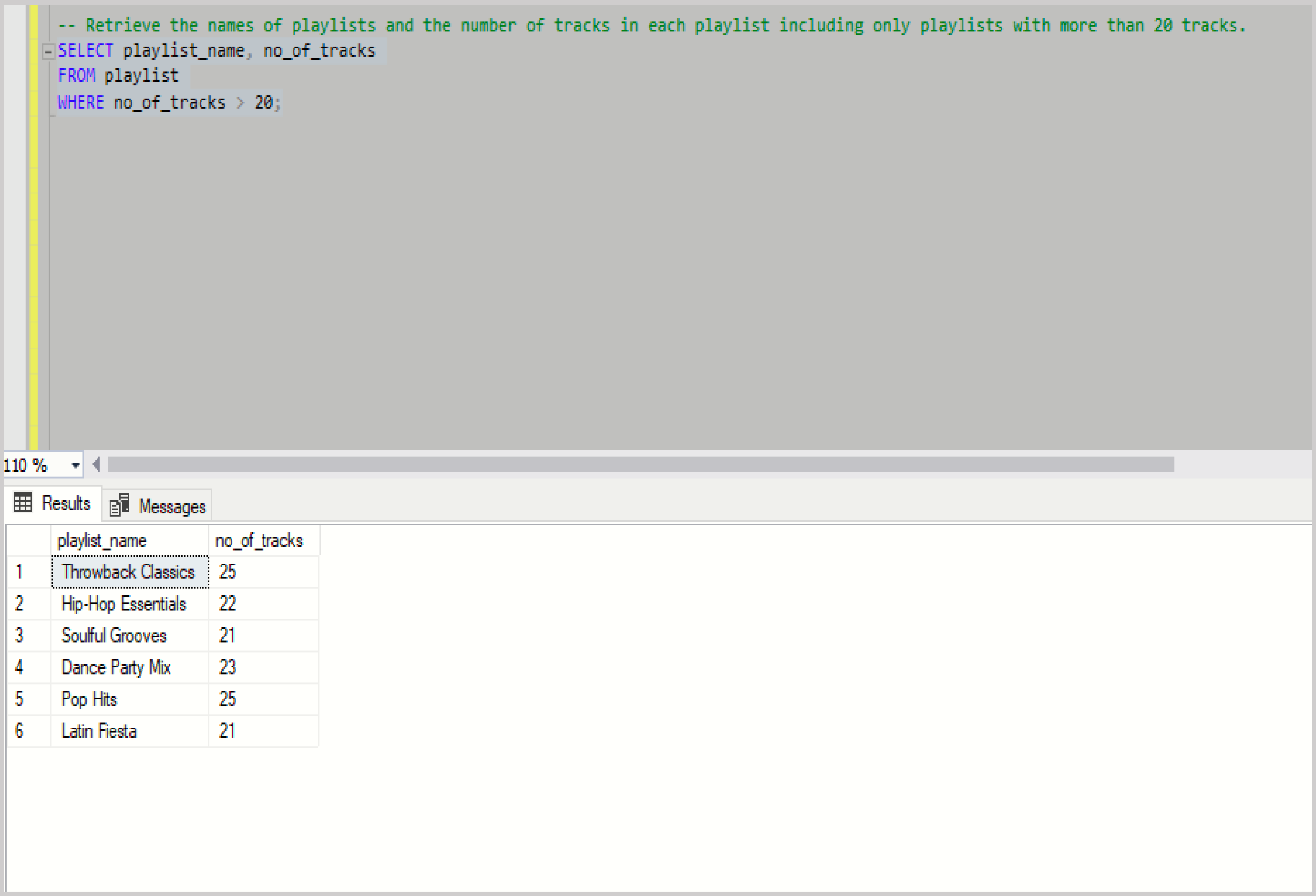


```
-- Retrieve the details of the track that is in playlists with Pop Hits genre
SELECT track_id, track_name, track_genre, duration
FROM track
WHERE track_id IN (
    SELECT track_id
    FROM play_track
    JOIN playlist ON play_track.playlist_id = playlist.playlist_id
    WHERE playlist_name = 'Pop Hits'
);
```

	track_id	track_name	track_genre	duration
1	686	Electro Swing	Electronic	3:15

- <u>We want to retrieve the names of playlists and the number of tracks in each playlist including only playlists more than 20 tracks.</u>

- **We selected the name of the playlist and the number of tracks to check whether the playlist contain more than 20 track or not.**

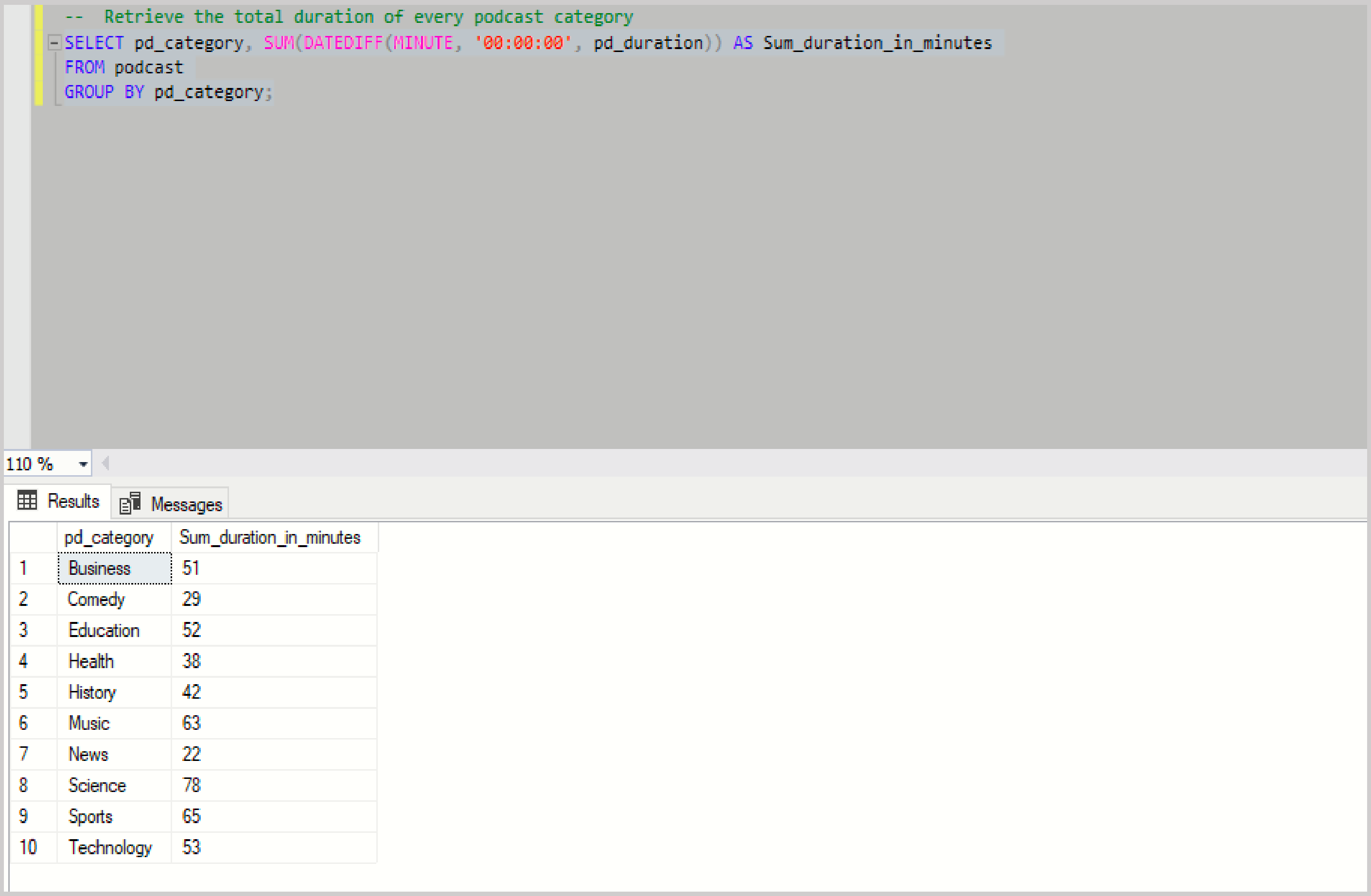


```
-- Retrieve the names of playlists and the number of tracks in each playlist including only playlists with more than 20 tracks.
SELECT playlist_name, no_of_tracks
FROM playlist
WHERE no_of_tracks > 20;
```

	playlist_name	no_of_tracks
1	Throwback Classics	25
2	Hip-Hop Essentials	22
3	Soulful Grooves	21
4	Dance Party Mix	23
5	Pop Hits	25
6	Latin Fiesta	21

- **We want to retrieve the total duration of every podcast category**

- **We selected the podcast category and we used the function "DATEDIFF" that can calculate the total time, from the podcast table and we grouped by the podcast category.**

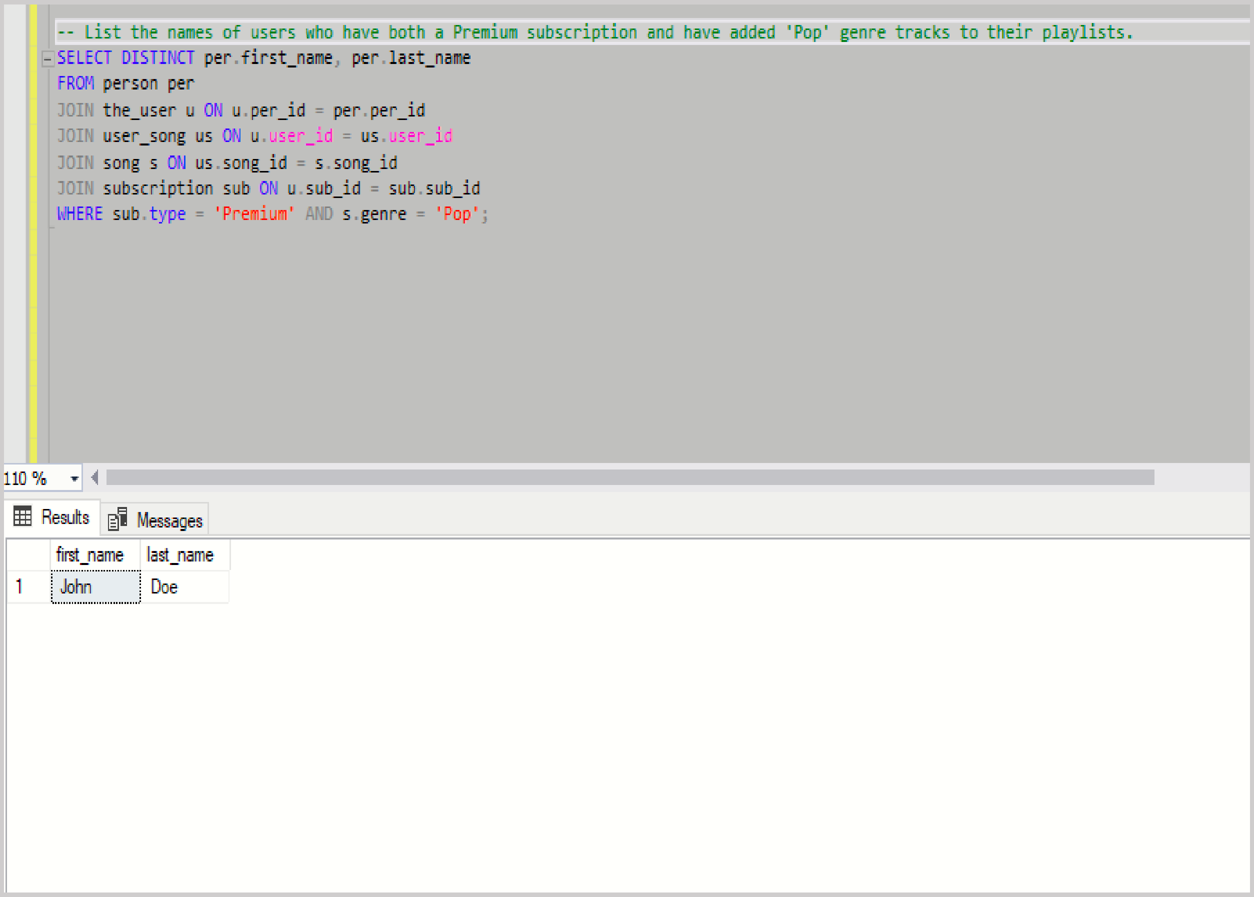


```sql
-- Retrieve the total duration of every podcast category
SELECT pd_category, SUM(DATEDIFF(MINUTE, '00:00:00', pd_duration)) AS Sum_duration_in_minutes
FROM podcast
GROUP BY pd_category;
```

	pd_category	Sum_duration_in_minutes
1	Business	51
2	Comedy	29
3	Education	52
4	Health	38
5	History	42
6	Music	63
7	News	22
8	Science	78
9	Sports	65
10	Technology	53

- **<u>We want to list the names of users who have a premium subscription and in the same time they added “Pop” genre tracks to their playlists.</u>**

- **We selected distinct first name and last name to prevent duplication from the person table and we joined it with the user table, then we joined the user table with the user_song table, then we joined it with the song table and finally we joined the the song with the subscription table. We conditioned that the subscription type must be “premium” and the genre must be “pop”.**



```sql
-- List the names of users who have both a Premium subscription and have added 'Pop' genre tracks to their playlists.
SELECT DISTINCT per.first_name, per.last_name
FROM person per
JOIN the_user u ON u.per_id = per.per_id
JOIN user_song us ON u.user_id = us.user_id
JOIN song s ON us.song_id = s.song_id
JOIN subscription sub ON u.sub_id = sub.sub_id
WHERE sub.type = 'Premium' AND s.genre = 'Pop';
```

	first_name	last_name
1	John	Doe

- **<u>We want to list the users who have listened to tracks from an album released in the last 6 months.</u>**

- **We selected distinct person details to prevent duplication, and we made list of joins between:**

1)Person and user

2)user and user_song

3)user_song and song

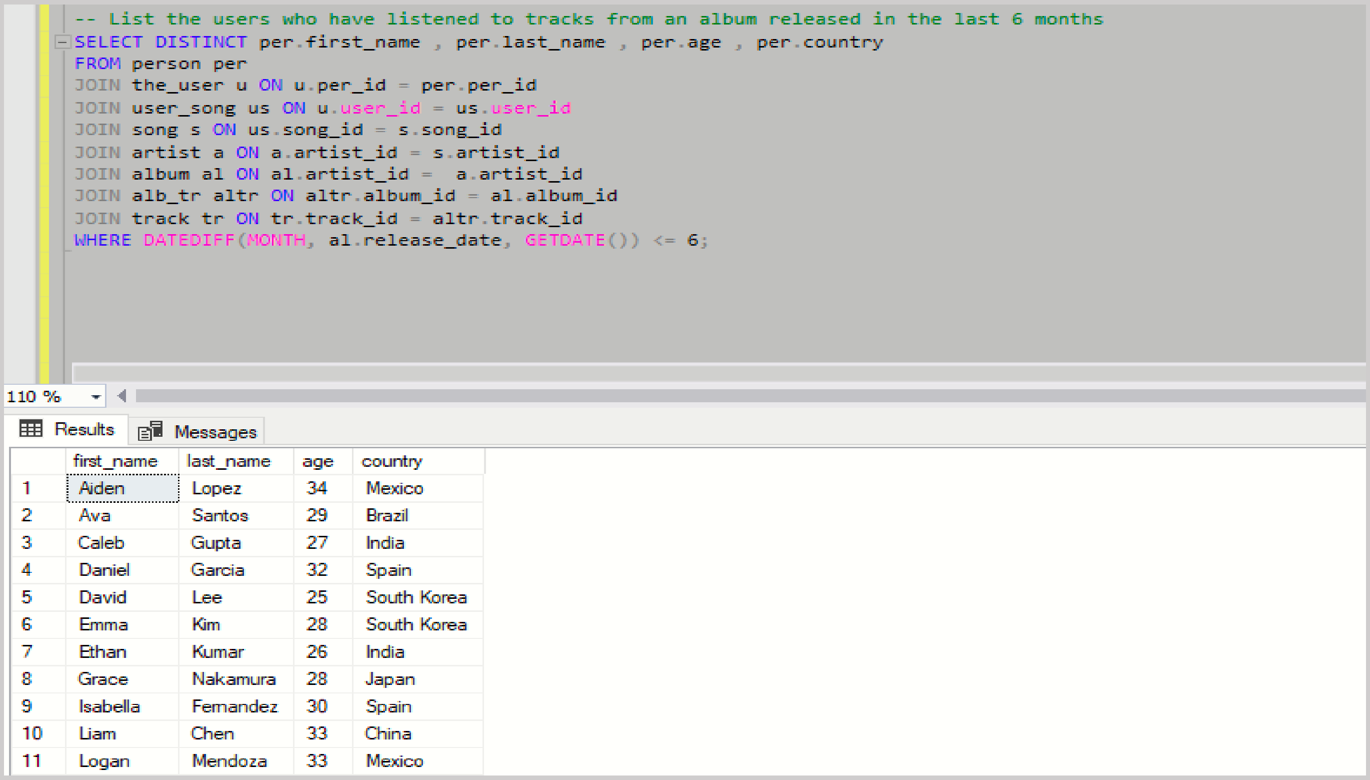
4)song and artist

5)artist and album

6)album and alb_tr

7)alb_tr and track

- **We used the DATEDIFF function and the release date and GETDATE function to call the date of today and calculate if the listened track is in the last 6 months or not.**



```
-- List the users who have listened to tracks from an album released in the last 6 months
SELECT DISTINCT per.first_name , per.last_name , per.age , per.country
FROM person per
JOIN the_user u ON u.per_id = per.per_id
JOIN user_song us ON u.user_id = us.user_id
JOIN song s ON us.song_id = s.song_id
JOIN artist a ON a.artist_id = s.artist_id
JOIN album al ON al.artist_id =  a.artist_id
JOIN alb_tr altr ON altr.album_id = al.album_id
JOIN track tr ON tr.track_id = altr.track_id
WHERE DATEDIFF(MONTH, al.release_date, GETDATE()) <= 6;
```

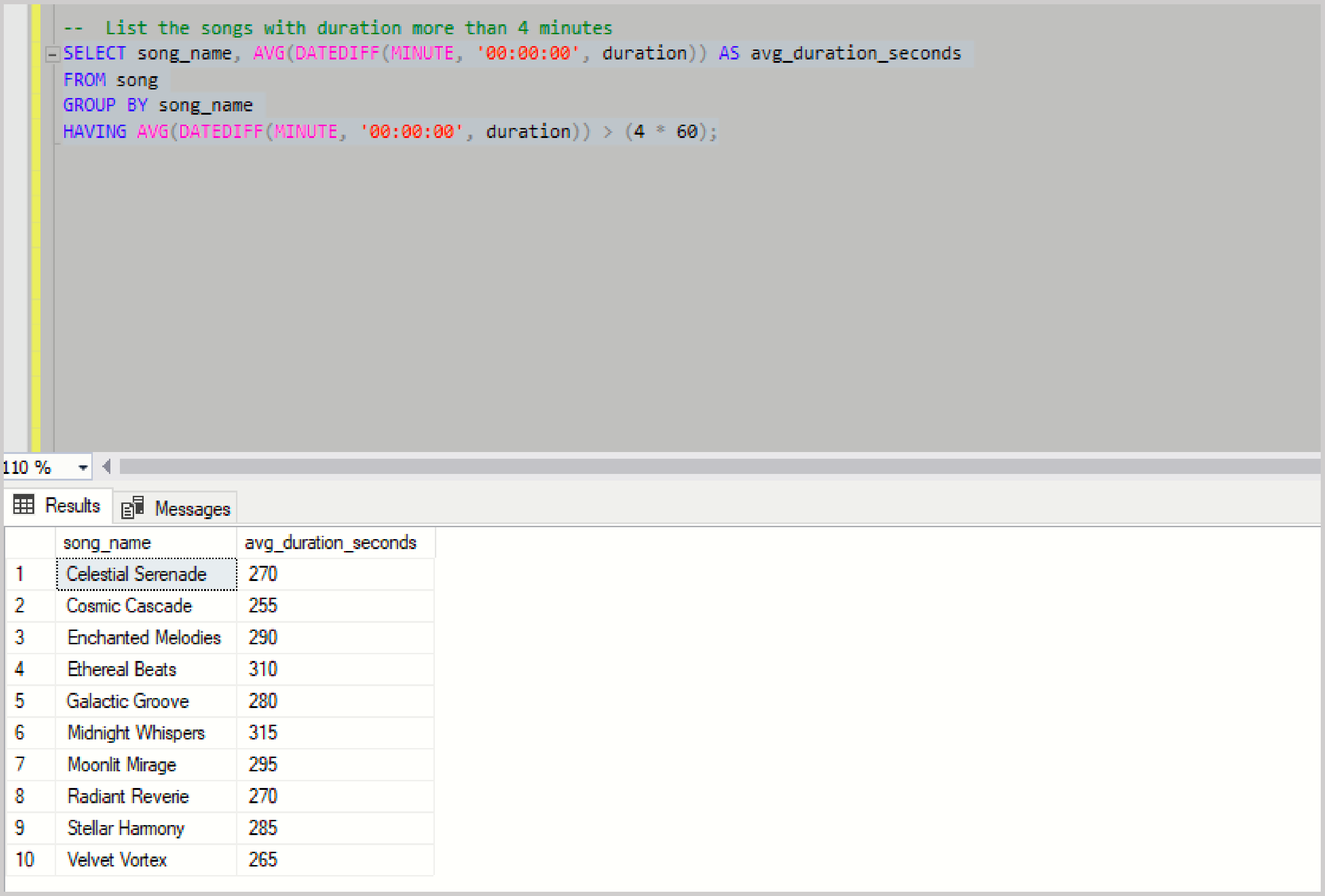
110 %

Results | Messages

	first_name	last_name	age	country
1	Aiden	Lopez	34	Mexico
2	Ava	Santos	29	Brazil
3	Caleb	Gupta	27	India
4	Daniel	Garcia	32	Spain
5	David	Lee	25	South Korea
6	Emma	Kim	28	South Korea
7	Ethan	Kumar	26	India
8	Grace	Nakamura	28	Japan
9	Isabella	Fernandez	30	Spain
10	Liam	Chen	33	China
11	Logan	Mendoza	33	Mexico

- **<u>We want to list all songs with duration more than 4 minutes</u>**

- **We selected the song name and average duration of every song in seconds from the table song grouped by the song name and condition that the average is greater than 4 minutes.**



```sql
--  List the songs with duration more than 4 minutes
SELECT song_name, AVG(DATEDIFF(MINUTE, '00:00:00', duration)) AS avg_duration_seconds
FROM song
GROUP BY song_name
HAVING AVG(DATEDIFF(MINUTE, '00:00:00', duration)) > (4 * 60);
```

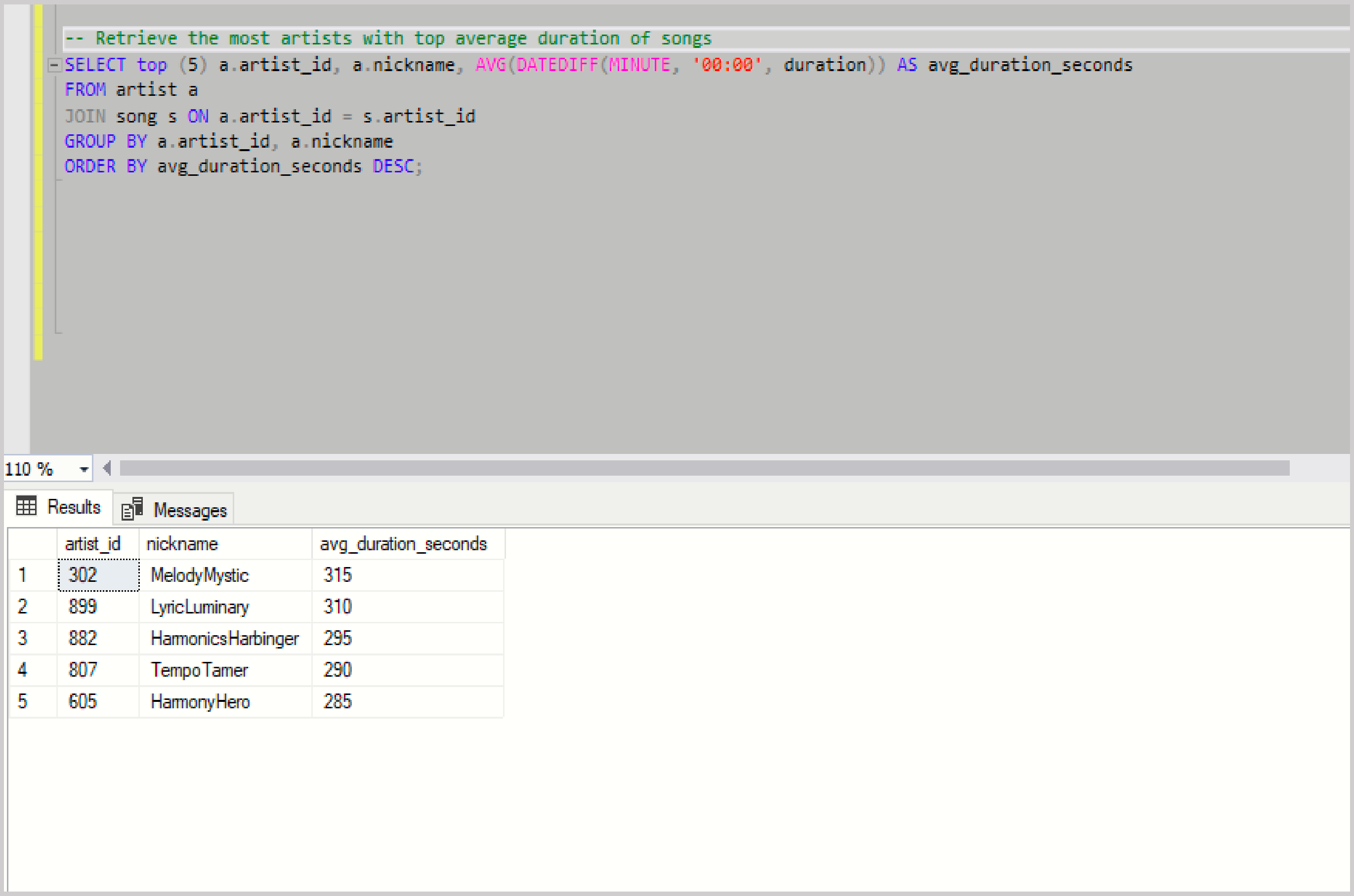
110 %

Results | Messages

	song_name	avg_duration_seconds
1	Celestial Serenade	270
2	Cosmic Cascade	255
3	Enchanted Melodies	290
4	Ethereal Beats	310
5	Galactic Groove	280
6	Midnight Whispers	315
7	Moonlit Mirage	295
8	Radiant Reverie	270
9	Stellar Harmony	285
10	Velvet Vortex	265

- <u>We want to retrieve the most artists with the top average duration of songs</u>

- **We selected the top 5 artists with the average duration in seconds using DATEDIFF function from the artist table joining the song table grouped by the nickname of the artists and ordered descendingly by the average duration in seconds.**

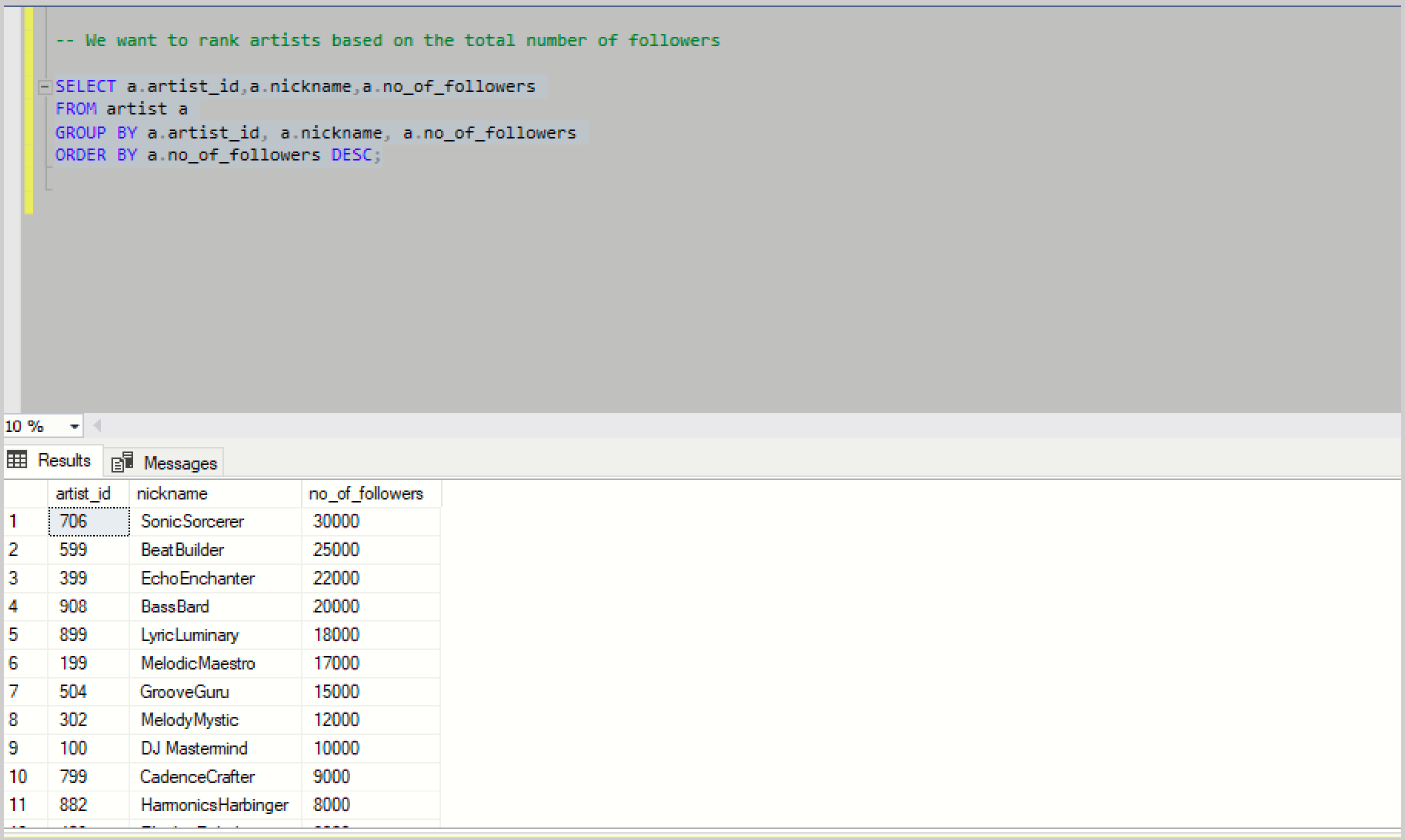


```
-- Retrieve the most artists with top average duration of songs
SELECT top (5) a.artist_id, a.nickname, AVG(DATEDIFF(MINUTE, '00:00', duration)) AS avg_duration_seconds
FROM artist a
JOIN song s ON a.artist_id = s.artist_id
GROUP BY a.artist_id, a.nickname
ORDER BY avg_duration_seconds DESC;
```

	artist_id	nickname	avg_duration_seconds
1	302	MelodyMystic	315
2	899	LyricLuminary	310
3	882	HarmonicsHarbinger	295
4	807	TempoTamer	290
5	605	HarmonyHero	285

- <u>We want to see the rank of the artists based on the total number of followers.</u>

We selected some of the artists details, grouped by the id, nickname and the number of followers and ordered by the no_of_followers descendingly.

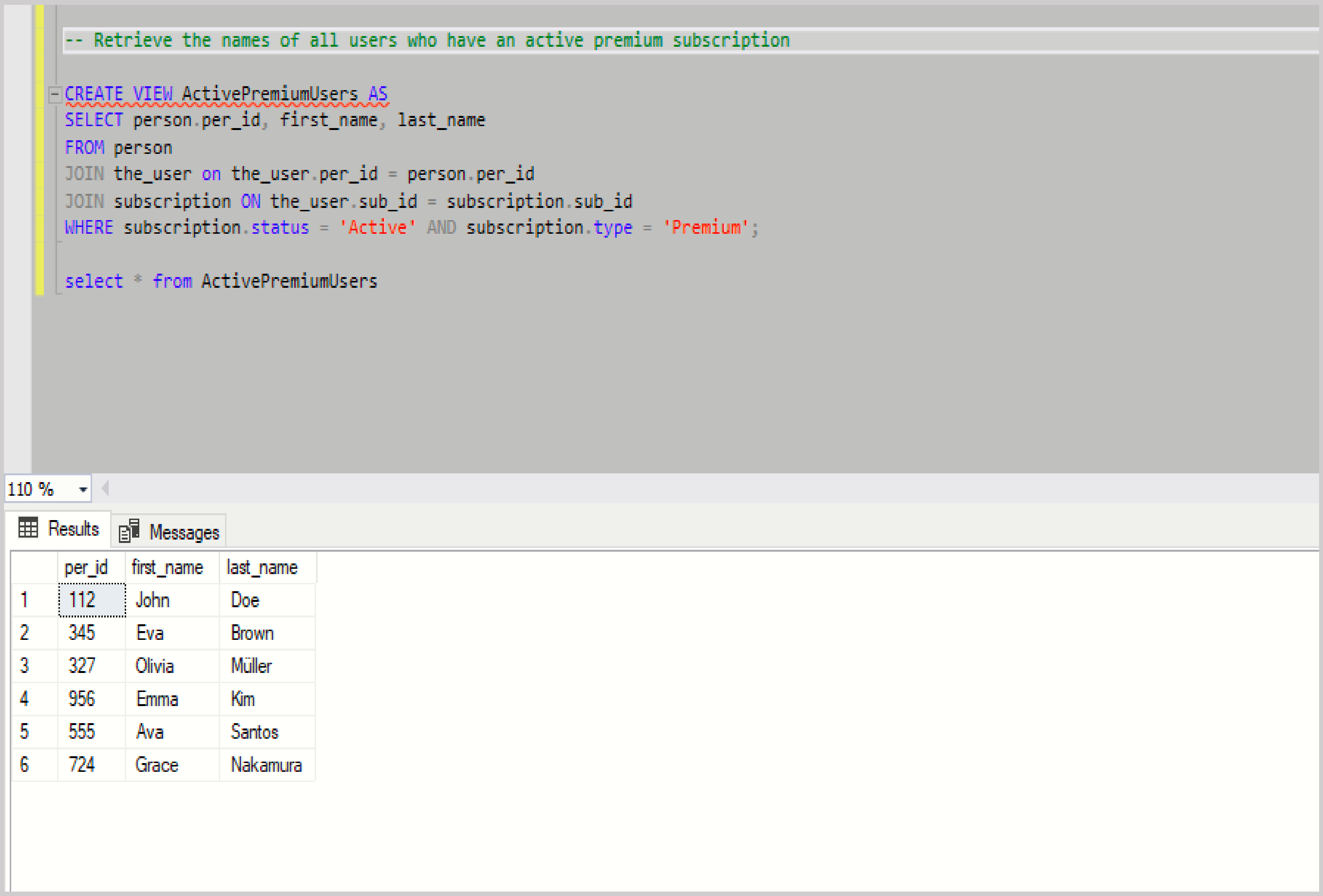


```sql
-- We want to rank artists based on the total number of followers

SELECT a.artist_id,a.nickname,a.no_of_followers
FROM artist a
GROUP BY a.artist_id, a.nickname, a.no_of_followers
ORDER BY a.no_of_followers DESC;
```

	artist_id	nickname	no_of_followers
1	706	SonicSorcerer	30000
2	599	BeatBuilder	25000
3	399	EchoEnchanter	22000
4	908	BassBard	20000
5	899	LyricLuminary	18000
6	199	MelodicMaestro	17000
7	504	GrooveGuru	15000
8	302	MelodyMystic	12000
9	100	DJ Mastermind	10000
10	799	CadenceCrafter	9000
11	882	HarmonicsHarbinger	8000

- **<u>We want to retrieve the names of all users who have an active premium subscription.</u>**

- **Using view named "ActivePremiumUsers" , we selected some person's details from the person table joining the user table, then joining the user with the subscription table to ensure that the subscription status is active and the type is premium.**



```sql
-- Retrieve the names of all users who have an active premium subscription

CREATE VIEW ActivePremiumUsers AS
SELECT person.per_id, first_name, last_name
FROM person
JOIN the_user on the_user.per_id = person.per_id
JOIN subscription ON the_user.sub_id = subscription.sub_id
WHERE subscription.status = 'Active' AND subscription.type = 'Premium';

select * from ActivePremiumUsers
```

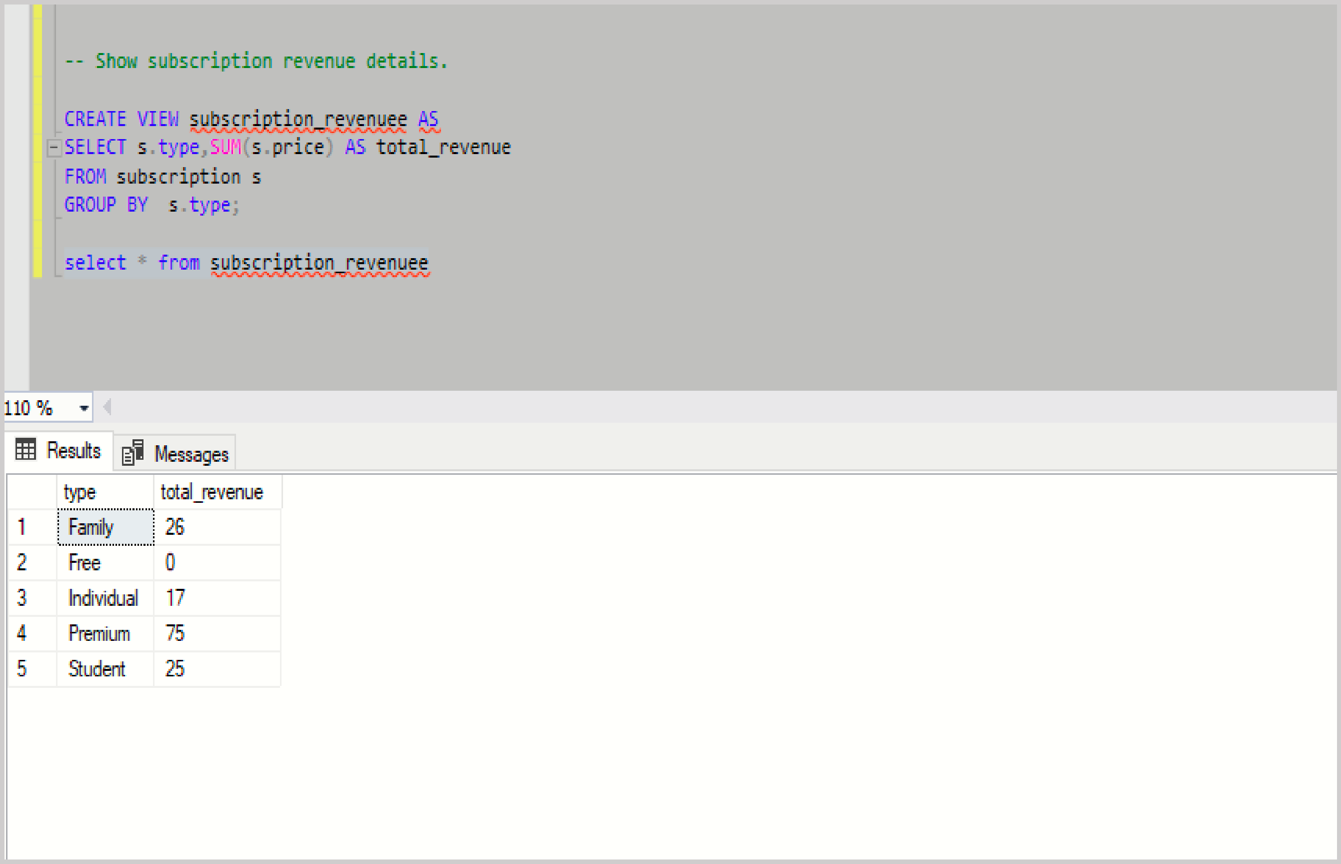
110 %

Results | Messages

	per_id	first_name	last_name
1	112	John	Doe
2	345	Eva	Brown
3	327	Olivia	Müller
4	956	Emma	Kim
5	555	Ava	Santos
6	724	Grace	Nakamura

- **We want to show the total revenue of each subscription type.**

- Using view , we selected the subscription type and the sum of the prices from the subscription table and grouped by the subscription type.



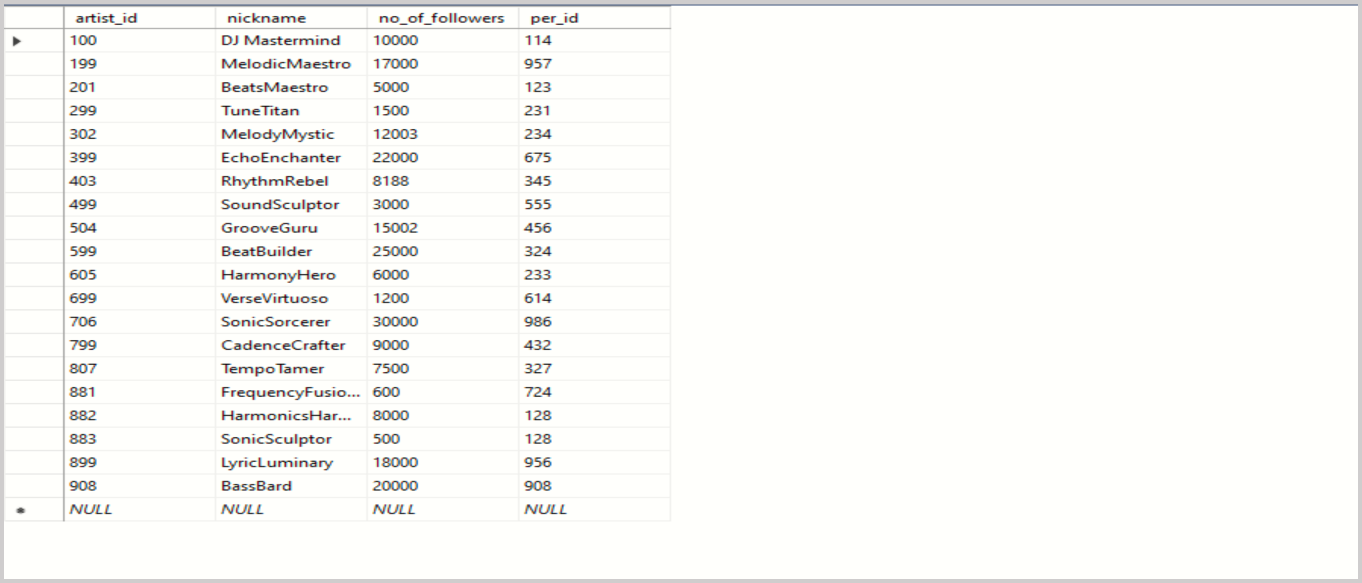
```sql
-- Show subscription revenue details.

CREATE VIEW subscription_revenuee AS
SELECT s.type,SUM(s.price) AS total_revenue
FROM subscription s
GROUP BY  s.type;

select * from subscription_revenuee
```

110 %

Results | Messages

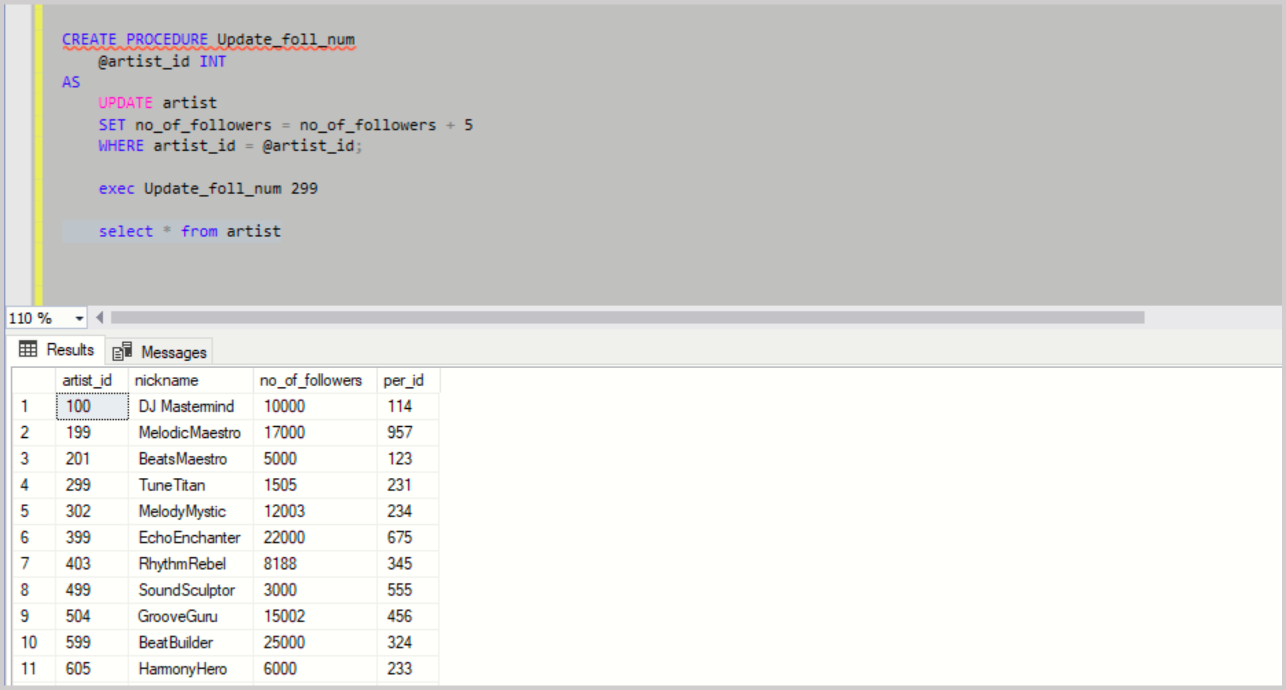
	type	total_revenue
1	Family	26
2	Free	0
3	Individual	17
4	Premium	75
5	Student	25

We want to add the no_of_followers by 5 for a given artist, we will use the procedure, so we will take the artist number 299 as an example. The number of his/her followers was 1500



artist_id	nickname	no_of_followers	per_id
100	DJ Mastermind	10000	114
199	MelodicMaestro	17000	957
201	BeatsMaestro	5000	123
299	TuneTitan	1500	231
302	MelodyMystic	12003	234
399	EchoEnchanter	22000	675
403	RhythmRebel	8188	345
499	SoundSculptor	3000	555
504	GrooveGuru	15002	456
599	BeatBuilder	25000	324
605	HarmonyHero	6000	233
699	VerseVirtuoso	1200	614
706	SonicSorcerer	30000	986
799	CadenceCrafter	9000	432
807	TempoTamer	7500	327
881	FrequencyFusio...	600	724
882	HarmonicsHar...	8000	128
883	SonicSculptor	500	128
899	LyricLuminary	18000	956
908	BassBard	20000	908
NULL	NULL	NULL	NULL

And after completing our query, the number of the followers became 1505



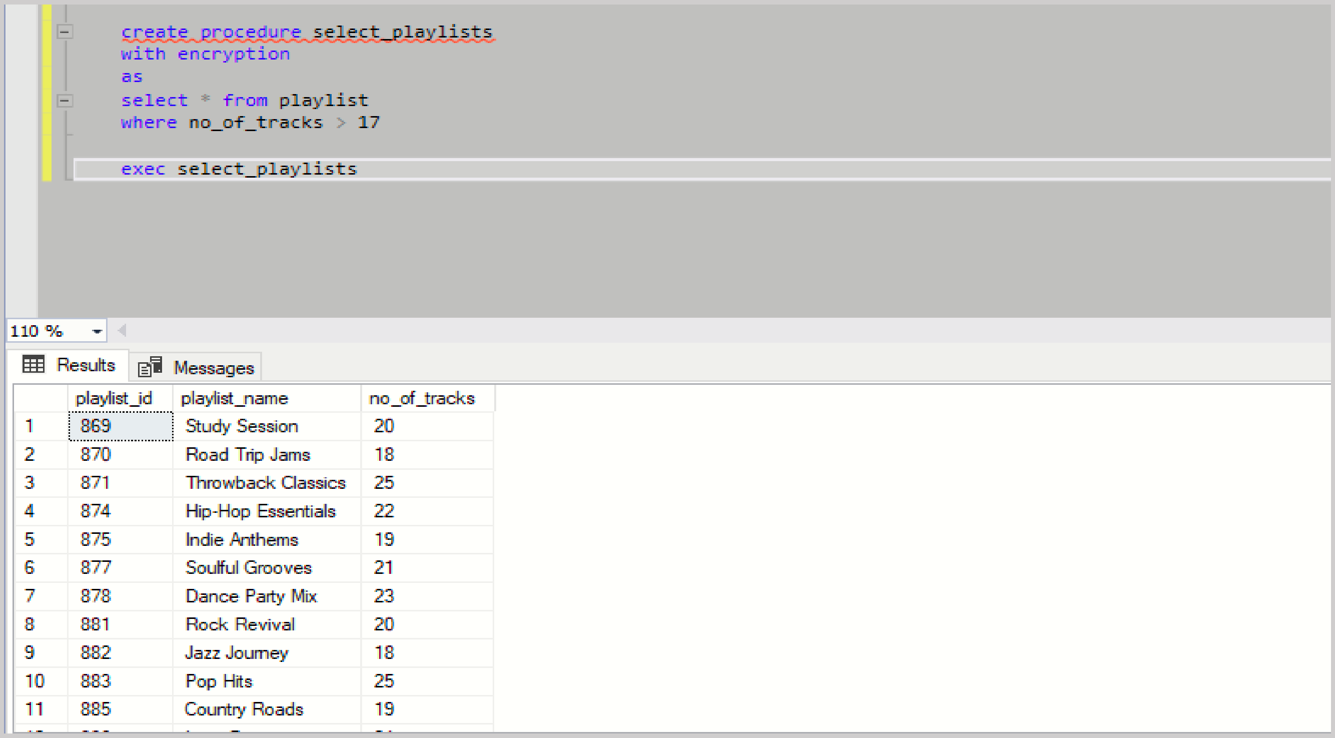
```
CREATE PROCEDURE Update_foll_num
    @artist_id INT
AS
    UPDATE artist
    SET no_of_followers = no_of_followers + 5
    WHERE artist_id = @artist_id;

    exec Update_foll_num 299

    select * from artist
```

110 %

Results | Messages

	artist_id	nickname	no_of_followers	per_id
1	100	DJ Mastermind	10000	114
2	199	MelodicMaestro	17000	957
3	201	BeatsMaestro	5000	123
4	299	TuneTitan	1505	231
5	302	MelodyMystic	12003	234
6	399	EchoEnchanter	22000	675
7	403	RhythmRebel	8188	345
8	499	SoundSculptor	3000	555
9	504	GrooveGuru	15002	456
10	599	BeatBuilder	25000	324
11	605	HarmonyHero	6000	233

We want to retrieve playlists that have more than 17 tracks, so we used the procedure.



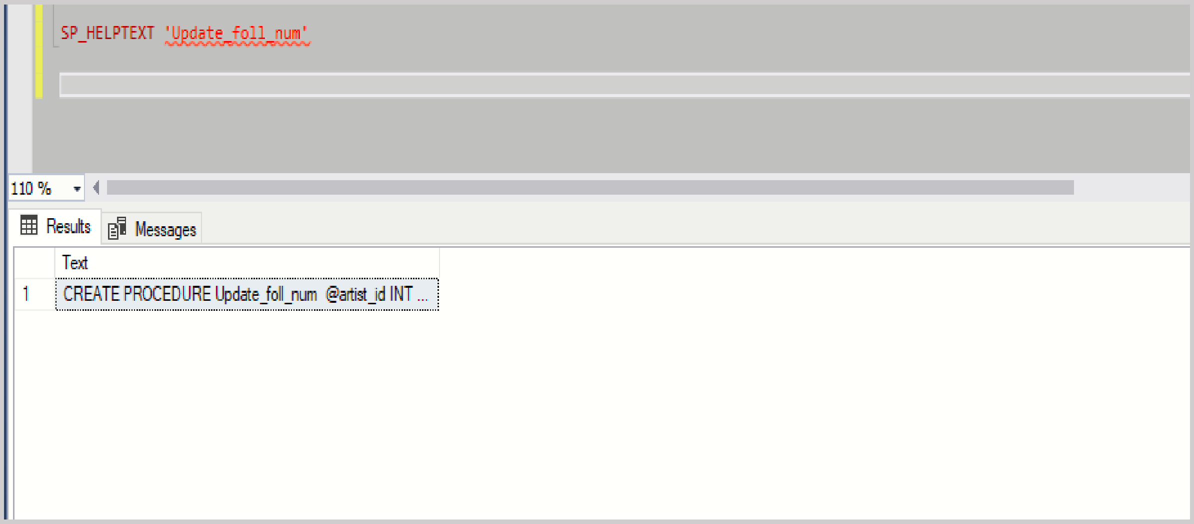
```
create procedure select_playlists
with encryption
as
select * from playlist
where no_of_tracks > 17

exec select_playlists
```

110 %

Results | Messages

	playlist_id	playlist_name	no_of_tracks
1	869	Study Session	20
2	870	Road Trip Jams	18
3	871	Throwback Classics	25
4	874	Hip-Hop Essentials	22
5	875	Indie Anthems	19
6	877	Soulful Grooves	21
7	878	Dance Party Mix	23
8	881	Rock Revival	20
9	882	Jazz Journey	18
10	883	Pop Hits	25
11	885	Country Roads	19

In the first procedure we are able to see our code:

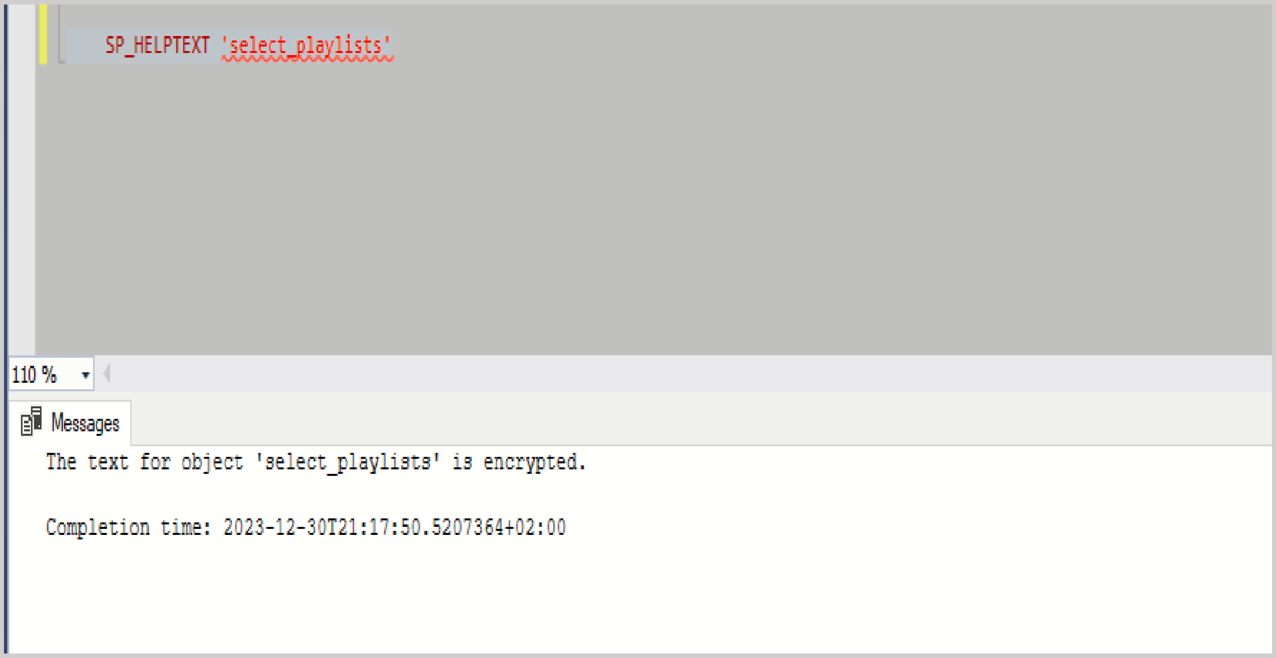


```
SP_HELPTEXT 'Update_foll_num'
```

Results | Messages

	Text
1	CREATE PROCEDURE Update_foll_num @artist_id INT ...

In the second procedure we won’t be able to see the code because we used the “with encryption option”:



```
SP_HELPTEXT 'select_playlists'
```

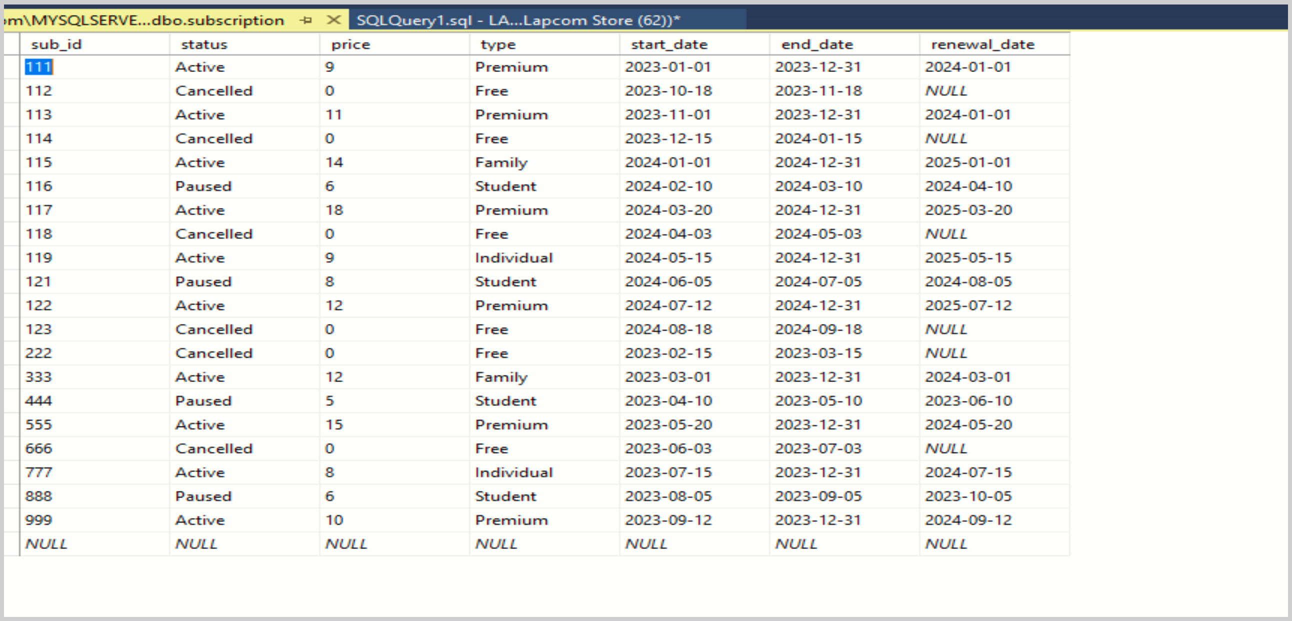
Messages

```
The text for object 'select_playlists' is encrypted.

Completion time: 2023-12-30T21:17:50.5207364+02:00
```

- **<u>We want to increase the price of premium subscription by 5 when any user changes his/her subscription type to premium.</u>**

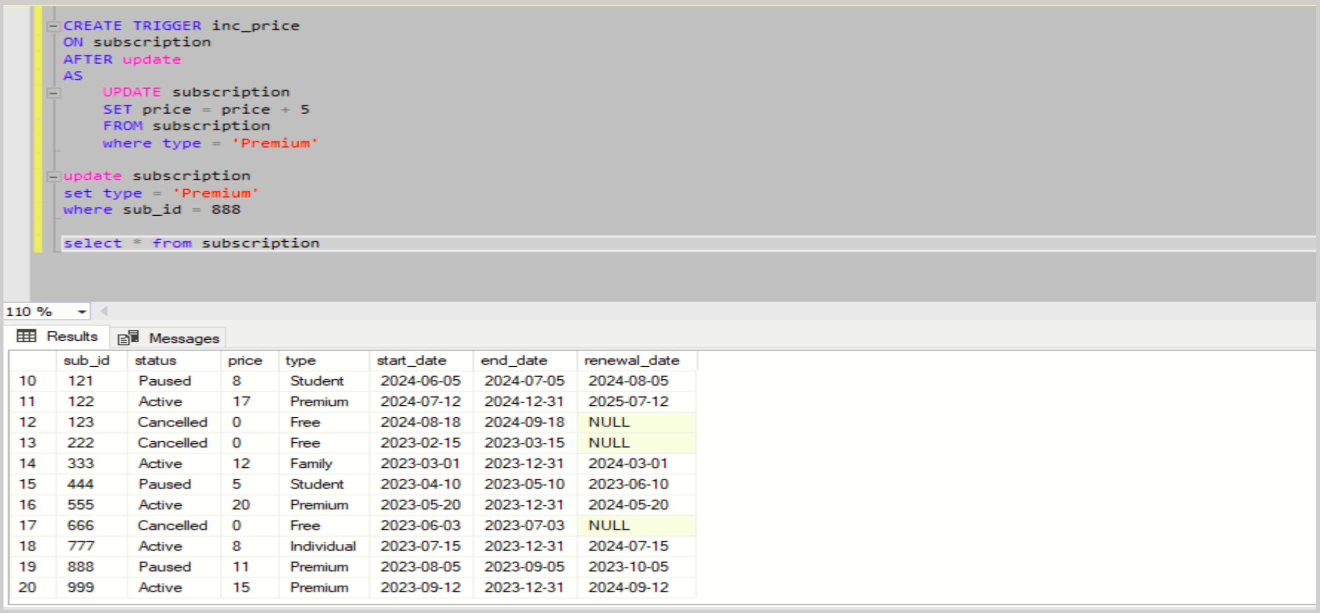
- **We have done this by the trigger**

The prices before was :


om\MYSQLSERVE...dbo.subscription
SQLQuery1.sql - LA...Lapcom Store (62))*

sub_id	status	price	type	start_date	end_date	renewal_date
111	Active	9	Premium	2023-01-01	2023-12-31	2024-01-01
112	Cancelled	0	Free	2023-10-18	2023-11-18	NULL
113	Active	11	Premium	2023-11-01	2023-12-31	2024-01-01
114	Cancelled	0	Free	2023-12-15	2024-01-15	NULL
115	Active	14	Family	2024-01-01	2024-12-31	2025-01-01
116	Paused	6	Student	2024-02-10	2024-03-10	2024-04-10
117	Active	18	Premium	2024-03-20	2024-12-31	2025-03-20
118	Cancelled	0	Free	2024-04-03	2024-05-03	NULL
119	Active	9	Individual	2024-05-15	2024-12-31	2025-05-15
121	Paused	8	Student	2024-06-05	2024-07-05	2024-08-05
122	Active	12	Premium	2024-07-12	2024-12-31	2025-07-12
123	Cancelled	0	Free	2024-08-18	2024-09-18	NULL
222	Cancelled	0	Free	2023-02-15	2023-03-15	NULL
333	Active	12	Family	2023-03-01	2023-12-31	2024-03-01
444	Paused	5	Student	2023-04-10	2023-05-10	2023-06-10
555	Active	15	Premium	2023-05-20	2023-12-31	2024-05-20
666	Cancelled	0	Free	2023-06-03	2023-07-03	NULL
777	Active	8	Individual	2023-07-15	2023-12-31	2024-07-15
888	Paused	6	Student	2023-08-05	2023-09-05	2023-10-05
999	Active	10	Premium	2023-09-12	2023-12-31	2024-09-12
NULL	NULL	NULL	NULL	NULL	NULL	NULL

But when performing our event, the prices has become :



```
CREATE TRIGGER inc_price
ON subscription
AFTER update
AS
    UPDATE subscription
    SET price = price + 5
    FROM subscription
    where type = 'Premium'

update subscription
set type = 'Premium'
where sub_id = 888

select * from subscription
```

	sub_id	status	price	type	start_date	end_date	renewal_date
10	121	Paused	8	Student	2024-06-05	2024-07-05	2024-08-05
11	122	Active	17	Premium	2024-07-12	2024-12-31	2025-07-12
12	123	Cancelled	0	Free	2024-08-18	2024-09-18	NULL
13	222	Cancelled	0	Free	2023-02-15	2023-03-15	NULL
14	333	Active	12	Family	2023-03-01	2023-12-31	2024-03-01
15	444	Paused	5	Student	2023-04-10	2023-05-10	2023-06-10
16	555	Active	20	Premium	2023-05-20	2023-12-31	2024-05-20
17	666	Cancelled	0	Free	2023-06-03	2023-07-03	NULL
18	777	Active	8	Individual	2023-07-15	2023-12-31	2024-07-15
19	888	Paused	11	Premium	2023-08-05	2023-09-05	2023-10-05
20	999	Active	15	Premium	2023-09-12	2023-12-31	2024-09-12

Conclusion: The Soundtrack of Success

In the grand symphony of database design, our journey through the Spotify project has been nothing short of a musical masterpiece. Like a carefully curated playlist, we harmonized tables, danced with joins, and composed queries that hit all the right notes.

As we explored the data, we discovered the rhythm of user preferences, the melody of podcast subscriptions, and the bassline of artist popularity. The SQL queries were our instruments, and the database, our grand concert hall.

Just as a Spotify algorithm fine-tunes recommendations, we fine-tuned our views, procedures, and triggers to create a seamless user experience. We synced our data like a perfectly timed beat drop, ensuring that each piece of information played in harmony with the rest.

And let's not forget the backstage stars – our stored procedures and triggers, working tirelessly behind the scenes, ensuring data integrity and keeping the show running smoothly. It's like having the unsung heroes of a rock band making sure the instruments are perfectly tuned before hitting the stage.

In the world of databases, errors are our unexpected solos. We faced a few unexpected key changes, but with the resilience of a seasoned musician, we handled them with grace and precision.

As we reach the final chord, let's applaud the collaborative effort, the SQL virtuosity, and the data-driven crescendo. Our Spotify database project isn't just a database; it's a symphony of code, creativity, and a touch of SQL magic.

So, whether you're creating playlists or crafting queries, remember: In the database of life, always choose the rhythm that makes your heart sing. Keep on grooving, SQL maestros! 🎶🎸🎤

"The symphony of this project was composed by..."

Ali Mohamed Ali Fahim	*214022*
Mohamed Nasser	*214029*
Mina Thabet	*214020*
Mohamed Wael	*214065*
Sandro Sameh	*214021*