Online Music Management System

DBMS - IT 214

TEAM - S6_T8

Instructor - Prof. Minal Bhise

Mentor TA - Kavan

Team Members -

Karan Solanki	(**201901085**)
Yash Prajapati	(**201901120**)
Milan Vadheri	(**201901121**)
Sachin Prajapati	(**201901125**)

Dhirubhai Ambani Institute of Information and Communication Technology

27 Nov-2021

Table of Contents

Section 1
Final version of SRS

1. Introduction

1.1 Purpose

We want to create Online music system in which any seller, artist, organization can sell Music and Seller can Add music by adding name, artist, genre and duration of the song and he can remove his/her song. Customer can browse from various albums. Artists can personally set their music rates and customers can purchase a song whichever he/she likes.

Taking about Admin privileges: Admin - the owner of the site and the administrator account has full privileges and control over the whole database and has the ability to add/edit/delete any information stored in the database.

1.2 Intended Audience and Reading Suggestions

Developers can read our codes related to add/subtract operations of songs and they can also read our structure of handling different queries. Mainly customers and artists can see how to add and purchase songs.

These features let the customer know that the purchasing process is transparent and hence it would help gain customer trust which in turn will increase customer retention.

1.3 Product Scope

This platform can act as interface between customer and seller and provide them virtual interface that they can use to buy and sell the music tracks. A customer can choose a song that he/she may like based on many things such as > artists, genre etc.

Platform can provide them two things. Firstly, Platform can gather data and suggest music that suits client preferences. Secondly it can collect data of customers location and their preferences that help artist to target customers and help him to improve his ideas.

Our flow of design – all the specified data will be added to the database by creation of new row Then in this specified data from artist we will take song details and from customer we take location and e-Mail and phone number

etc. for Artist we have added two functionalities – Adding track and removing track for customer we want efficient experience so we want to design functionality which provide them correct preferences songs, prices and discount are very important as well so we want to give somewhat that also. Mainly we want customer statistics and sales analysis that In which type of song they are interested mainly.

1.4 Description

In this part, specific system related requirement will be provided briefly. We justify system in terms of its perspective, functionality then user characteristics, project assumption and dependencies after that We also talk about performance requirements.

Majorly we want to design system that if changes necessarily required in future then without much rework system can be maintained smoothly.

Functionalities and their perspectives:

Creating Accounts:

While creating an account, the user is asked to enter some basic information including his name, location, email address, phone number. It also incorporates a column which specifies if the user is an artist, a band or an organization. the user is then requested to choose a username.

This username must be unique and must follow certain constraints such as its length should at least be 2 characters. After choosing a unique user name, the user is requested to enter a password which should be at least 8 characters in length and containing at least one capital letter to ensure password security.

Adding Track:

Sellers can add songs and they need to fulfil some requirements where they are asked to give info about the name of the song, the artist, the genre, duration of the song.

The seller can provide more details about the song such as mode, tempo, liveness, speechiness etc. in order to help the user be more specific about his requirements. This helps the user choose the song better and thereby, there is estimated higher sale of the music. These details shall be requested from the seller itself. However, if the seller is unable to provide these additional sophisticated data, It would leave this field empty in the database.

Removing Track:

If seller wants to add song for limited time and wants to remove without loosing song statistics then that can be implemented and its is like song will be removed from online web and sale statistics associated with it remain in database.

Efficient browsing:

This functionality is very important since because efficient browsing will lead to increase in customer interest in our website. We can provide them filters using which they can display specific titles.

The filters will have criteria such as Genre, Artist, Album, Duration, Price etc. The customer is free to combine multiple criteria together when searching. The customer can also sort the displayed music based on audio features of the song such as duration, tempo, loudness etc.

Through the use of filters and sorting, the customer can find exactly the type of music he prefers.

Managing cart:

This is needed functionality as far as online store is concern. The system provides the buyer with a cart, which would store the songs the buyer is willing to pay for. This cart can store multiple songs, thereby allowing the buyer to purchase all of them at once. Hence, saving him from the trouble of purchasing each music individually. The system allows for the buyer to hop between the web pages without affecting the cart.

User can check out and confirm his payment this payment also can be done via different modes like using ATM card or UPI well these details can’t be stored in database to ensure protection of customer.

Sales statistics:

This thing provides user a clear understanding of his product performance. statistics provide the sellers with an insight and model to implement further strategies in order to enhance their profitability. The seller can view his total number of sales, total revenue, Revenue Breakdown by Genre/Album etc. Popularity of product in countries etc. The information would also be shown in a convenient graphical format to enhance readability.

Seller can use this data to make financial decisions about his future songs or playlists.

Admin privileges:

The Admin is the owner of the site and the administrator account has full privileges and control over the whole database and has the ability to add/edit/delete any information stored in the database.

The admin can help customers/sellers recover their account in the case of a hijacked account or if they misplaced their login information.

1.5 References

1]. https://www.slideshare.net/nileshpadwal456/music-management-system

2]. https://krazytech.com/projects/sample-software-requirements-specificationsrs-report-airline-database

3]. https://www.oreilly.com/library/view/learning-mysql/0596008643/ch04s04.html

2. Fact-Finding Phase

2.1 Background Reading

Today in this busy world every one listen to music when they get tired. Music is one of the greatest so others and healers of an afflicted heart. Customers obtain these music items from music stores. Many sites Uses HTML, CSS, JS as front end for this kind of web designs and they also uses back end well data gathering is also an important part many sites Uses Web Scraper Web scraping is a computer software technique of extracting information from websites. The web scraper would be made with the use of python and its libraries such as Scrapy/BeautifulSoup.

Well it has many uses such as Generate list of tracks. Communicate with Spotify's API to get audio features of a specific track. Gather information related to artists. Spotify and Gaana apps are Examples of online music store they provide variety of albums and genre and customers are highly attracted to them.

Well some references are discussed in terms of today's music system requirements and data handling functionalities.

Reference 1.

https://link.springer.com/content/pdf/10.1007/978-3-319-24800-4_14.pdf

Taking about Recommendation that customers get and some methods to make system efficient in that section.

Online systems help customer finding products according to their taste. In this article a music recommendation system is discussed and this recommendation systems play important role in giving customer what they want.

The way how consumers access music has changed in recent years due to the rise of the web. Nowadays, consumers have the possibility to access a huge amount of different music using various devices and services. An example for such new services is music streaming platforms. The distribution and especially the inventory costs of such platforms are lower than the costs of traditional channels.

Well platforms like Spotify, Gaana provide functionality that users can publish their own creations so music field became more diverse.

People uses different techniques to solidify information in this article it's shown that they use Twitter data and utilize it to give music recommendation. This article mainly talks about two problems

-> The first problem is the recommendation process itself. This problem is addressed by a collaborative filtering (CF) based recommender system.

-> the second problem is addressed, which is a lack of publicly available and recent data appropriate for implementing and evaluating recommender system in academia.

In order to provide artist recommendation system rely on collaborative filtering User-based CF recommends items by finding similar users, herein referred to as the nearest neighbours, and then suggesting items a user didn't interacted with, but his nearest neighbours interacted with. This is based on the assumption that like-minded users prefer the same items and that these user preferences remain stable over time.

Well extracting data from APIs and using them is and approach and this is approach is used by others also well as an example of this -> the generation of appropriate datasets for research in the field of recommender systems.

Hauger, D., Schedl, M., Kosir, A., Tkalcic: published the Million Musical Tweets Dataset (MMTD). The MMTD is a static dataset containing about one million music related tweets. All tweets in this dataset are tagged with a geolocation. Similar to ours, their dataset also suffers from a high data sparsity or rather a low number of tweets per user.

Given system uses a genetic algorithm for optimizing the input parameters to the presented, steadily updated dataset. For computing the actual recommendations, it uses user-based collaborative filtering with a hybrid user-similarity computation.

In this article it's said that performance of this recommendation system is very satisfying although we can use more user centric algorithms.

From this article we took a look at how music recommendation happens and how online music platforms using data for giving their customers best music and best artist according to user taste.

Reference 2.

https://minerva-access.unimelb.edu.au/bitstream/handle/11343/266632/2021-Creating-Authentic-Assessments-for-Online-Music-Courses%20-Mapping-a-Learning-Task.pdf

Taking about specifically streaming and stuff that today's online music systems are providing and looking into this new aspects and related work.

Here we will talk about Music streaming Spotify is a streaming service offering low-latency access to a large library of music. Streaming is performed by a combination of client-server access and a peer-to-peer protocol. The service currently has a user base of over 10 million and is available in seven European countries.

Streaming services such as Spotify, Apple Music, and others become the most popular method to listen to music. In 2017, Music streaming service provider relish 41,1% growth in streaming revenue with 45,5% growth of paid subscription(IFPI, 2018). This means that the music is stored on the consumer's device within the memory of the streaming service app. The user does not own any of the music in the catalogue; in this way, it is similar to how people rent books from a library. Users are also unable to access the individual MP3 files within the catalogue, and therefore they cannot make copies for their own use. Through a monthly subscription, whether paid or free, users are able to stream their favourite artists easily on their own devices.

One of the most popular music streaming services in the world is Spotify. Spotify first launched in Asia about five years ago. Being a global music streaming platform, its most attractive unique selling point is its vast music library, with millions of titles.

Talking about particularly Spotify -> Spotify is a legal streaming service; all music available for streaming is licensed to Spotify. Users can also play any mp3 files they have on their local drive from the Spotify client, as well as include them in playlists together

with streaming tracks. Furthermore, the Spotify client will attempt to “link” local files to tracks in the streaming library by matching the metadata. By default, the client will play the local mp3 file instead of streaming the track when a file has been linked.

Playlists as well as albums, artists, and tracks can be shared with others in the form of URIs. Further encouraging users to share URIs, Spotify clients can directly post URIs and messages to Facebook, Messenger, and Twitter.

Reference 3.

https://ivi.fnwi.uva.nl/fcn/teaching/2007-dbp-ik/Projectmaterial/Team6_final.pdf

A general look at small database design for any online music systems some aspects that we can take reference from as far as our music system is concern.

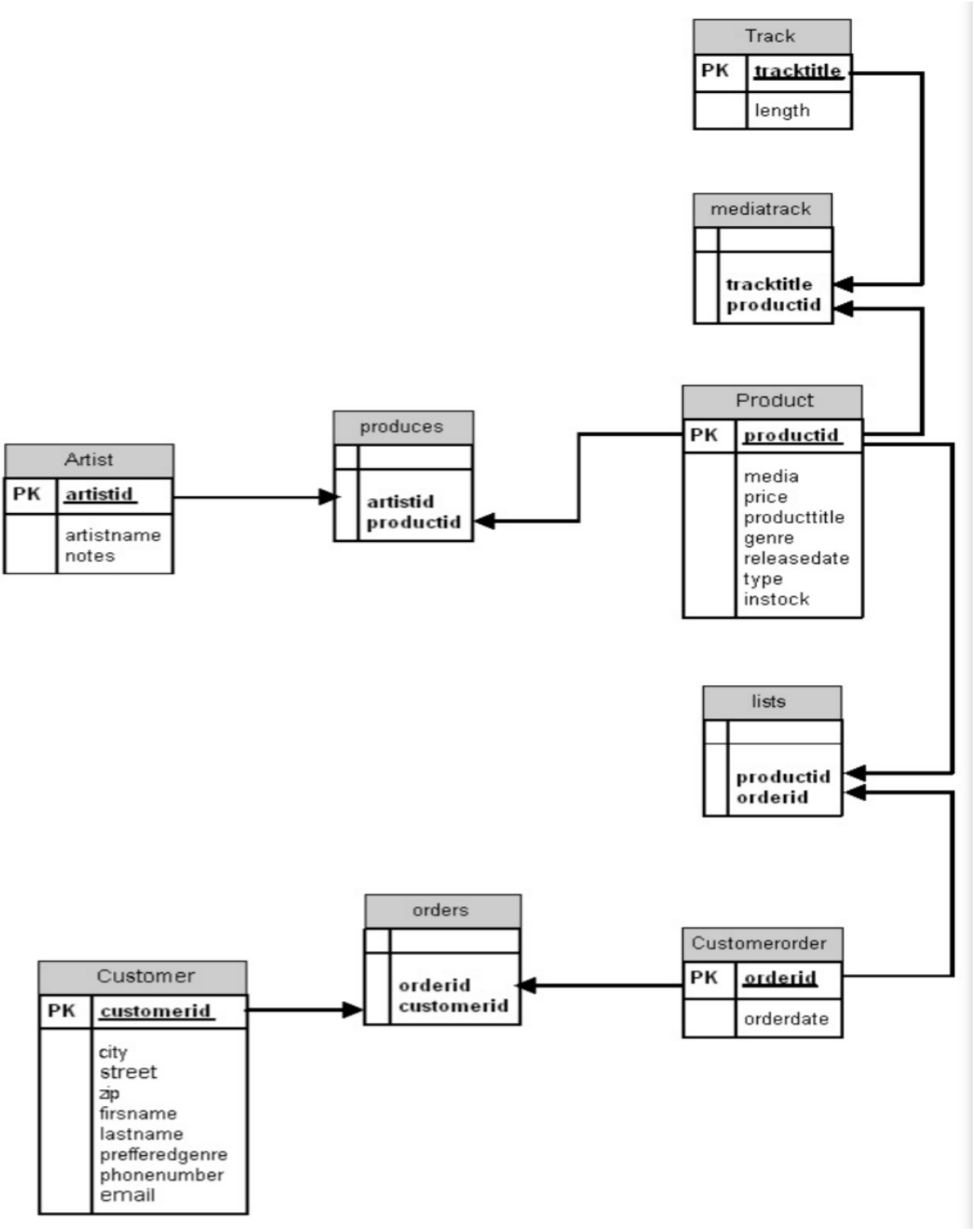
We talk about existing online platforms and their special functionalities but now here we will talk about conventionally database design of this Music stores.

This PDF talks about simple application for customers to order CDs and DVDs. This study gives us an understanding of what we can do for our project although it’s a small application but if we look at hole picture then we will understand that most platforms have their basic stuff on these things only.

Taking about **Technology** they have used:

- MySQL Query browser: This SQL tool enables an easy way to change the database without an application.
- Netbeans IDE: A Java editor
- Microsoft Visio: Designing the structure of the project.

Taking about Database tables particularly they provided a picture well that is given below we can look into that and take a reference that how general database for any music store look like.



Reference data model(Not what we designed)

This are some of the related studies on music management Specific to online and there are many more but now we will look into interviews.

Existing System Problems:

voice control:

In some of the system, serching a music in some of the system is very difficult for some people who are not familier with the system.

so we can add voice control in the system. so that most of people can easily search any music by artist name or music name.

Security

we have to encrypt the information of card number and other details of users so, no one can misuse that password and card details.

it provides more privacy to customer. developer is also not able to see this information.

Streaming

audio streaming is new feature that are available in some of the online music apps.

some of the online music system are not providing the audio streaming.

that is also big problem of some users.

Cold start problem:

Problem definition One of the major problems of recommender systems in general

and music recommender systems in particular is the cold start problem,

i.e., when a new user registers to the system or a new item is added to the catalog

and the system does not have sufficient data associated with these items/users.

In such a case, the system cannot properly recommend existing items to a new user

or recommend a new item to the existing users. This is the most common problem in the online music management system.

Logistic Management

Today music specific systems have statistics but they don’t use it in any other manner and most of it don’t go for extension. For example using this info they can sell goods related to music and move their business in further fields.

2.2 Interviews

I. Artist(Role Play):

Interviewee: 1) Karan Solanki

2) Milan Vadheri

Contact Details: inquiry@whitehatsr.com

Organization Details: www.whitehatsr.com

InterViewer: 1)Arjuna Roy Designation: Artist

Date: 7/10/2021 Time: 4:00

Duration: 45 Min Place: WhitehatSr Office

Purpose of the interview:

First Meeting with An Artist to identify their concern about how their requirements can be fulfilled by this system.

- Copyright Concern about their songs
- Add feature for understanding client behaviour.
- Live Streaming
- Seller to Seller interaction
- Ratting of their songs based on client's behaviour.
- Account Verification
- Add and removing tracks should be easier process

II. Customer(Role Play)

Interviewee: 1) Karan Solanki

2) Milan Vadheri

Contact Details: inquiry@whitehatsr.com

Organization Details: www.whitehatsr.com

Interviewer: 1) Yash sadhu Designation: Music buyer

Date: 7/10/2021 Time: 6:00

Duration: 45 minutes Place: WhitehatSr office

Preliminary Meeting with a Frequent Music Buyer to Understand their perspective:

- Privacy concern about their data history
- Add feature of cart
- Live streaming
- Free Demo Songs
- Functionalities after download
- Provide Original Songs with no Copy case
- Cost for frequent Buyer should be reduced

III. Developer (Role Play)

Interviewee: 1) Karan Solanki

2) Milan Vadheri

Contact Details: inquiry@whitehatsr.com

Organization Details: www.whitehatsr.com

Interviewer: 1) Kishan Thakkar Designation: Technical Associate/ Developer

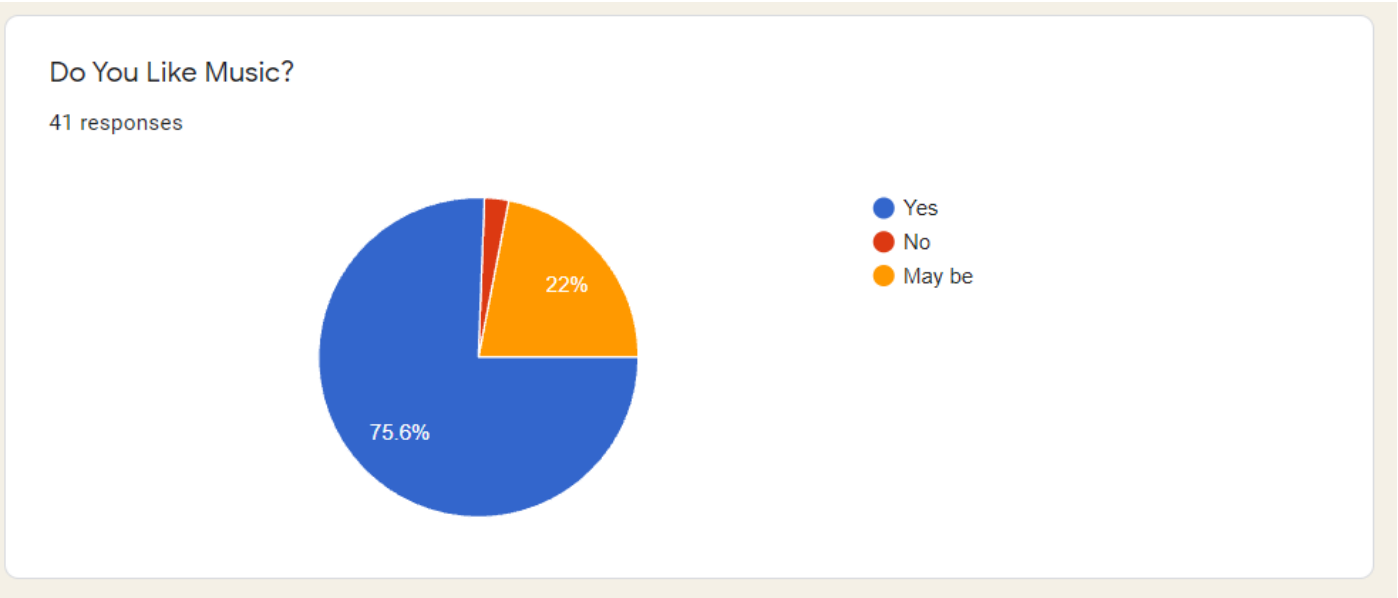
Date: 8/10/2021 Time: 4:00

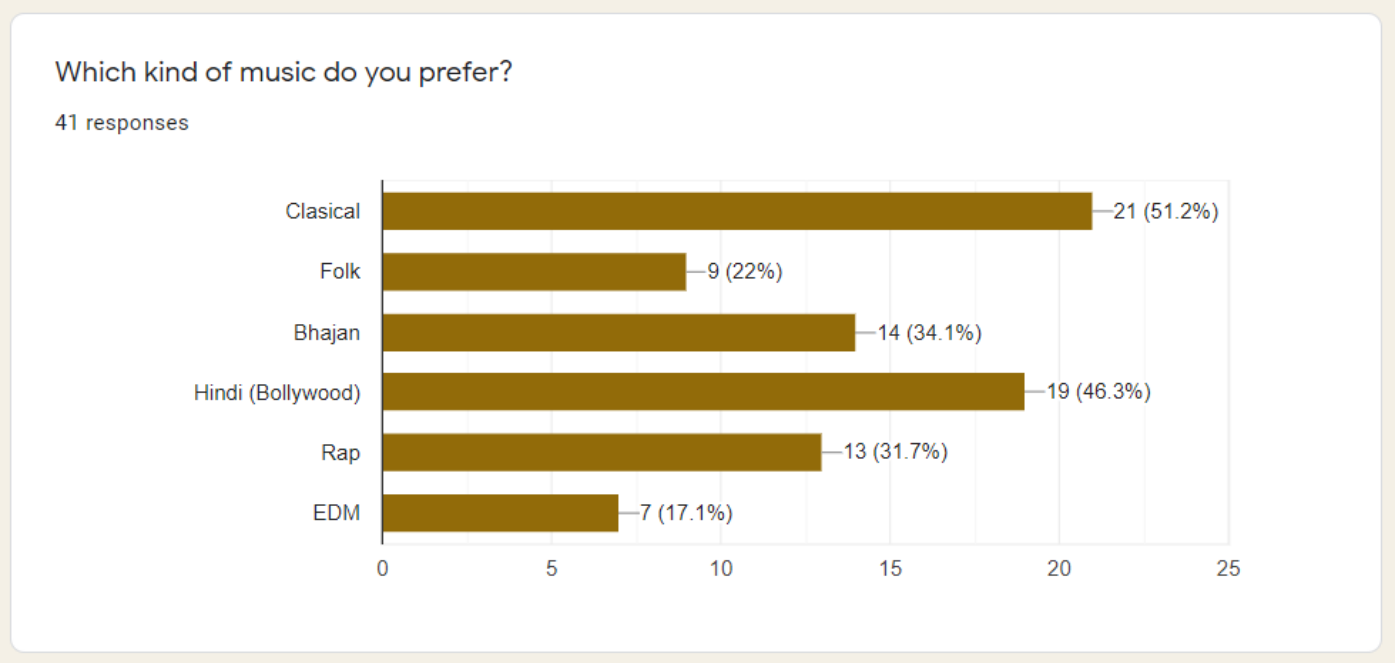
Duration: 45 min Place: WhitehatSr office

Preliminary meeting to discuss features and functionalities of the product:

- Privacy concerns about Artists and Customers
- Add feature of cart, Add/Remove Tracks
- Improve Music recommendation functionality and make it more user centric
- Prevent Dummy songs and give originals
- Give some part of track as demo
- Give discount options on various payment methods
- Improve UI
- Reduce/remove ads as long as possible
- Improve on Auto-play function
- Provide Different Interfaces for Artist and Customers
- Make it available online any time
- Improve on Currently Available functionality
- Handle Large no of audience on live streaming as well as in general
- Anonymous Suggestion

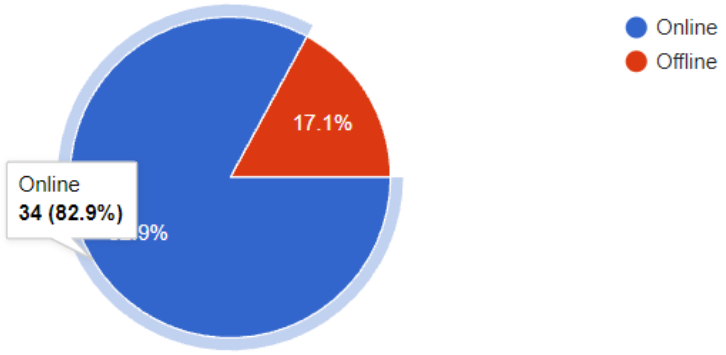
2.3 Questionnaire(summary):





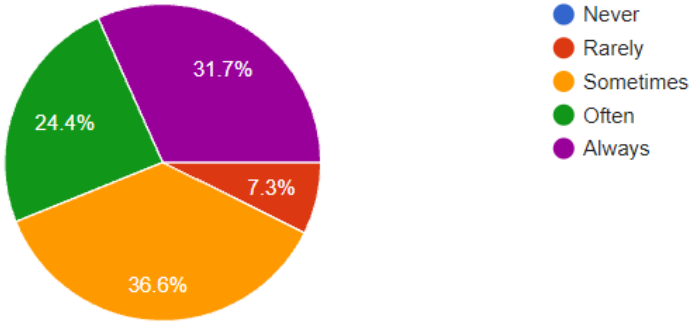
Which mode do you prefer?

41 responses



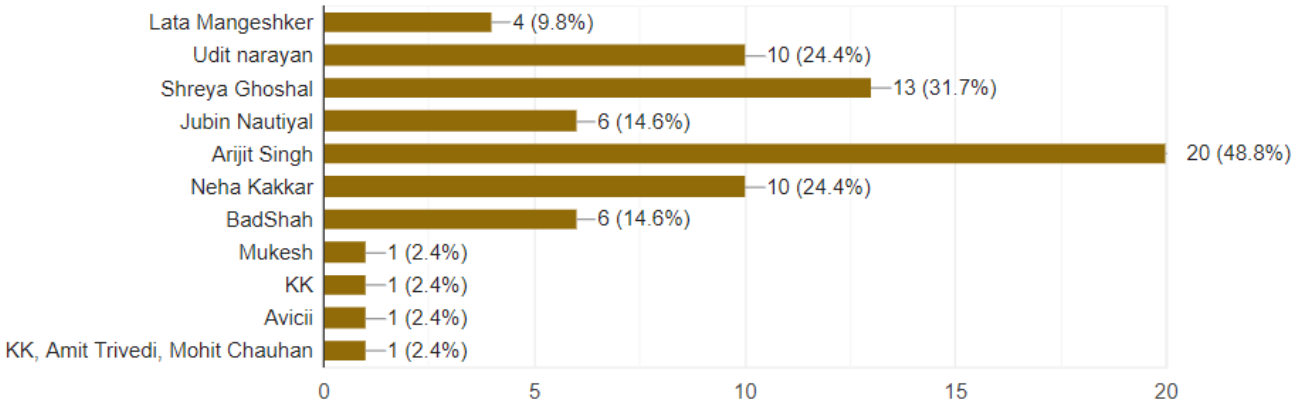
How Frequently you Listen to music?

41 responses



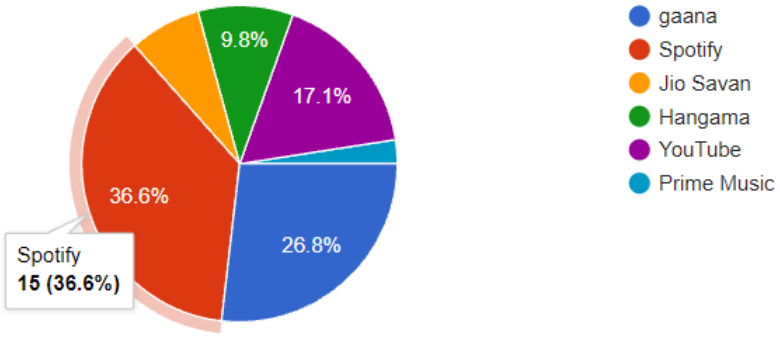
Which Artist You like the most?

41 responses



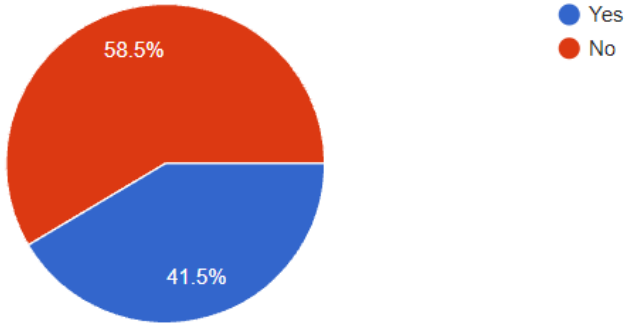
Which online platform you prefer the most?

41 responses



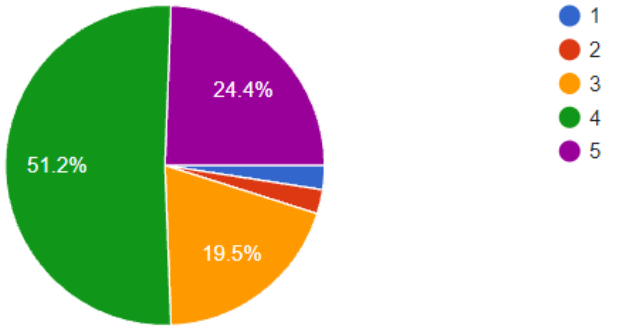
Have you ever purchased any music?

41 responses



What rating you will give to online music system that your favorite platform is providing?

41 responses



2.4 Observations:

- Artist require full security of their songs that copying that would lead to legal consequences

- They want features like client behaviour about there album, live streaming, rating of their songs based on client behaviour

- For Artist – Add/remove process should be smooth
- Artist Accounts should be verified so that consumes can get satisfaction about songs.
- Customers have concern about their data privacy
- general features like cart, free demo songs
- after download functionalities
- want original songs with no copy case
- If customer is frequent buyer then discount and price reduction should be there
- Developers should use artists and consumer's data appropriately
- Develop add/remove, cart, customer/seller statistics
- Recommendation should be more user centric
- Dummy products should be removed and original should be provided to consumers
- Develop free demo parts of music
- Give discount option on various payment methods
- Improve UI
- Reduce/remove ads as long as possible
- Improve on Auto-play function
- Provide Different Interfaces for Artist and Customers
- Make it available online any time
- Improve on Currently Available functionality
- Handle Large no of audience on live streaming as well as in general
- Anonymous Suggestion

2.5 Our Database – Relations and Attributes

- **Track**

Track ID	big int
Artist ID	big int
Album ID	big int
Genre ID	big int
Track Name	varchar
Track Type	varchar
Length	timestamp
Price	big int
Language	varchar
Release Data	timestamp
Genre	varchar

- **Album**

Album ID	big int
Track ID	big int
Album Name	varchar
Release Date	timestamp
No of Songs	big int

- **Artist(Seller)**

Artist ID	big int
Track ID	big int
Artist Name	varchar
Email	varchar
Username	varchar
Password	varchar
Ratting	big int
Gender	varchar

- **Customer**

Customer ID	big int
Customer Name	varchar
Email	varchar
Date Of Birth	varchar
Mobile No	varchar
Gender	varchar
Username	varchar
Password	varchar
Address	varchar

- **Order**

Order ID	big int
Track ID	big int
Artist ID	big int
Date	timestamp
Original Price	varchar
Discount	varchar
Total Amount	varchar

3. Fact Finding Chart

Objective	Technique	Subject	Time Commitment
Preliminary meeting to identify Problems and requirements of Artists	Interview	1 Artist	45 min
Preliminary meeting to identify problems of customers	Interview	1 frequent music buyer	45 min
Preliminary meeting to know problems of developer	Interview	1 Developer	45 min
To get background on design of online music management system	Background Reading	3 Articles	3*0.25 = 0.75 day
To get compact details on Music products	Observation	From articles, Questionnaire, Interview	0.5 day
To get clear understanding of what functionalities the users generally need	Questionnaire	41 Responses	0.5 day

4. List Requirements

- Artists have Copy case Concern about their songs
- Add feature for understanding client behaviour.
- Live Streaming
- Seller to Seller interaction
- Ratting of Artist's songs based on client's behaviour.
- Artist Account Verification
- Add and removing tracks should be easier process
- Customer have privacy concern about their data history
- Add feature of cart
- Free Demo Songs
- Functionalities after download
- Cost for frequent Buyer should be reduced
- Developer have Privacy concerns about Artists and Customers
- Add feature of Add/Remove Tracks
- Improve Music recommendation functionality and make it more user centric
- Prevent Dummy songs and give originals
- Give some part of track as demo
- Give discount options on various payment methods
- Improve UI
- Reduce/remove ads as long as possible
- Improve on Auto-play function
- Provide Different Interfaces for Artist and Customers
- Make it available online any time
- Improve on Currently Available functionality
- Handle Large no of audience on live streaming as well as in general
- Anonymous Suggestion

5. User Classes and Characteristics

Reference: https://www.planetarygroup.com/do-artists-getpaid-every-time-song-played-spotify/

1.Artists:

Description:

Artists can upload their songs on Spotify. They can sell their songs on Spotify for a price that they want. Artists can see how many people have listened to their music, giving them an idea of what needs improvement or what has been a hit with listeners. They have copyright of their own songs. If any film director wants to make remake of particular song, then they have to buy the rights from that song's artist, lyricst and music composer and after that they have to give credit to them also. They get paid from spotify , on the basis of how many times their songs are played. wheather particular artists are independent or one signed to major labels like T-series etc. he/she will use one or more streaming services to get his/her music to out there for everyone to listen and enjoy it. Additionally, it would also help artists for obtain some extra revenue in their musical productions.

If we add more, spotify gives them platform that millions of ears can enjoy and listen their songs and they get more and more streaming.so if they get more and more streaming , the better payout and better recognition they will get. As long as the song is played for thirty seconds, Spotify counts it as a stream and a per stream royalty is added to your grand total, to be paid out later. In early days, spotify doesn't provides sufficient payment to artist but nowdays it is quite ok but still not much sufficient. However, for signed artists this increase still needs to be split up between different parties with a financial interest in the work (publishers, writers, etc.). These amounts are still the subject of intense debate among many artists, from those tied to a contract with a major record label or those managing their own careers. For an independent artist who owns their own masters and the rights to all their songs, splitting the royalties isn't the problem.

It's getting those streams to rack up in a timely manner. For these reasons, the proper promotion and release strategy is of utmost importance. Spotify does not directly pay artists who are on their platform. You also cannot upload your music directly to Spotify. The middle man in this situation is the music distributor, such as TuneCore, DistroKid or CDBaby. These websites are the conduit between the artist and Spotify, and they are the ones who receive your royalties. Artists With their e-mails, can create an account and then can upload their songs for release. Most artists plan their release carefully and upload the songs two or so months in advance so they can take

advantage of pre-order promotion and releasing a single before the whole album comes out. distributors such as TuneCore make their payments to artists using Payoneer, which has four options to payment: PayPal, Direct Bank Transfer, Pre-paid Mastercard, or a paper check for those without a bank account (in the US and Canada only). Payoneer charges a $1 fee for processing your withdrawal and making the payment.

2. Audiences:

Description:

Audiences can download the songs which they want to listen in offline mode or They want to ad-free music by purchasing a premium version of a spotify. Audiences can search any songs from all languages that are available in spotify. they can add their favourite songs in their own made playlists. they have option to make their playlists private or public. Traditionally, other platforms, such as promoting music on the radio, can help you impact a particular segment of the populace. However, streaming services such as Spotify can give people access to the music of their favorite artists any time they wish for as long as they want.They have right to hit like button for the songs which they liked. They can give their reviews and comments about a song in its comment section, so artists can know people's reviews about their songs.

3. administrator:

Description:

administrator is the owner of site and he has full control over spotify database. He can add/edit/delete any information stored in spotify database. They can help to customers/seller incase of their account is hacked or they have misplaced their login details. They can take necessary action on some illegal hackings of customers/sellers account.

6. Operating Environment

For customer/seller

Browser Support:

- Google crome
- Firefox
- Microsoft Edge

Operating System:

- Android
- iOS
- macOS
- Window
- Linux
- PlayStation 4
- PlayStation 5
- Xbox One
- Xbox series X/S

For Desktop, Mobile we need to Vorbis 160 kbit/s audio compression format for standard quality options and Vorbis 320 kbit/s audio compression format for Premium quality options.

For **Administrator:**

- Connectivity: Bandwidth
- Window/Linux/iOS hosting database
- Language: Python, Java, c and c++
- Cloud base services
- Database: Cassandra

7. Product Functions:

We divide this in four parts:

1. Unregistered user
    - Search
    - View new songs
    - Register with site
    - Login to the site
    - Can view Help page
2. User
    - Login
    - Search Items
    - Add items to Shopping Cart
    - Add Items to Wish list
    - Add favourite
    - View Wish list
    - Give feedback
3. Artist
    - Have verified Account
    - Add/remove update songs
    - Create albums
    - Sales Statistics
4. Administrator
    - Login
    - Delete a user
    - Add items to inventory
    - Modify item details
    - View sales Report

- **Create account OR login account:**

In create account on the platform sellers/customers have provide their First name, last name, gender, date of birthdate. They should need to create a password to login. After creating account, customers/sellers have provided a unique id. This id used when customers/sellers have purchased/sold a music.

- **Add/Delete a song:**

Sellers add a song on the platform with artist name, album name, types of song, song name. Customers can see the details of song. So that Customer find particular song or particular artist name easily. Seller also delete song.

He can find the song via searching song name and song's artist name then seller can delete a song.

- **Search and filter:**

Customer can be found by customer id. When seller know about particular customer, seller can be searched by customer. It will be matched to the customer table and seller will be directly redirected to the customer page.

- **Purchase a song**

Customer can buy a song by amount of song and Sellers can sell a song with particular amount. On the platform, payment mode is existed. Customer peek a particular song then in payment mode, customer can fill the amount of the song and other details. So that it can buy a song. So that interaction between customer and seller can be made strong.

- **Managing Cart**

System provides seller a cart well it's very much known functionality where buyers can add several items that they want to buy and system should allow buyers to go on any webpage and cart shouldn't be affected by that.
Well user can checkout his total payment on cart and pay from there. Various payment methods also provided to pay.

- **Sales Statistics**

This functionality is only implemented for Artist and Admin they can look into that and implement their future strategy after seeing that statistics. Functionality provide a graphical format to enhance readability.

Seller(Artist) can view his/her total number of sales across the world and using that they can know their target areas and can also look at areas where their music is not very popular and to improve on that they can look in to major customer statistics.

- **Customer Statistics**

We will provide major customer statistics in Graph format so that Artists can understand much about hot spots and music that major ones are liking.

Also customer specific statistics are there for customer so that customer can look into it and see how much he/she is spending on music and adjust their purchasing frequency. This in turn increase customer trust.

- **Add Wishlist**

We have cart as well but that to be implemented for shopping purpose but as customer want to add some items that he is interested in and in future he can look into it and buy it for that purpose we implement wish list this functionality is useful for unregistered as well as registered users.

- **Feedback**

This function is important because as far as we think that our system is the best but that's can't be right and we always need collective public opinion so this is for users that if any kind of problems they are facing they can go to Feedback and give Feedback about it. This functionality will give administrator or developer a sign that clients are facing issues and they can work on it to resolve it.

- **Admin Privileges**

Admin also have an account on system but that account has privileges to delete any other account, if that account is showing some bad practices. Admin can add new items they can modify system and can look into sales reports.

Admin is owner of the site so he has full privileges and control over database and can edit/remove/add any info in the database. If any user facing issue with his/her account they should go to admin and admin can resolve it and tell what is the issue with user account.

8. Privileges

There are four types of users that can interact with system.

1. **users**
2. **Artists**
3. **Buyers**
4. **Developer**

Users:

users can’t access database directly but they can access this by app or web page. They are not able to change any database information related to the music. They have only permission to listen the music. They can rate the music according to their preference. They are only able to change his account information like (name, password, email) in database.

Artists:

They are also not able to access the whole database. They can upload their songs on that system. They can sell their songs on that platform. They can only insert the tuple that have the information about the song, price of that song, their name. they can change the information in that tuple. They can’t able to change the information of the other music’s information in database.

Buyers:

They don’t have the rights to change the information that currently present in the database. They can only buy the music by taking the permission of artist and paying him money. Buyers post the information of his name, and contact details in the database. So that artist can contact with him.

Developer:

Developer has all rights to related that music system. developer can able to change the information that are currently present in the database. Developer set the issues related that system. Developer can see the background codes that are working behind the app or web site. Developer has rights to remove the copyrighted song. He can perform add/delete/update in information of database. But he can’t use that song other places without taking the permission of the song artists.

9. Assumptions

- Users are able to download spotify app from playstore or app store.
- Audiences have atleast 512Kbps data rate and Artists have atleast 2Mbps data rate.
- Users have good quality of speakers and microphones in builted in their devices and they are well functioning.
- Users devices have required hardware needed to perform the functions efficiently.
- Users are able to create account using phone-number and gmail.

10. Business Constraints

- Artists cannot copy the song that belongs to the author artist. So, if artist or any buyer copy or use that song in film or somewhere else, without taking the permission of Artist of that song, then Artist or this system can take strictly action on them.
- Some of the features of that system are not available to the simple users Only paid users can access that features.
- Content will be available to all the types of uses.
- If someone wants to show their product ads, then they can contact with the developer.

Section 2
Noun Analysis

Final Problem Description

In online music management system we are trying to cover all the needs of people that should be implemented in this system Mainly this system is an interface between Artists, song writers, Music Organizations and user who listen to Music.

By making this system we are trying to solve the problems and requirements of this people For this we have looked at each and every aspect of User. We looked at Unregistered user we looked at Active User who has their login ID and Password.

We looked at Artists and Music Organizations we also looked at Existing systems their new and conventional functions and then we have done interviews and questionnaire and other things to understand their behaviour and to understand their requirements.

Prerequisites Developers should always work on improving system and make system responsive to everybody but still there are some prerequisites that should be there for different users. User should able to access our system and should able to listen to music so for that they should have good internet connectivity and 512kb can be good and For artists uploading new songs and new albums they should have 2Mbps of data rate.

System that we are trying to design will work on mobile, Tab, Laptop, Pc but for some previous version it might not work properly so that's why mobile and tab software should be up to date.

Apart from that user should have Good quality Microphone and Speakers inbuilt in their device and required hardware should be working properly lastly for login and other info they should be able to create their account using Gmail or Phone number.

These prerequisites are very much important because system makers try their level best to improve system but if issue is on user side that system can't do its best at these situations.

Here we will talk about problems and requirements.

- Data Privacy Concern is there for Artist and Customer both How to remove that concern?

- Customer has Fear of not getting original Music from Online system how to remove it?

- Customer want their browsing should be efficient and they should get whatever they want instantly how to deal with it?

- Artist want to trace their Progress and they want to know customer's behaviour to their Music how to provide them that data in efficient manner keeping customer's privacy in mind?

- Overall Improved UI should be there and Customer, Artists and Admin UI must be different what should be design for different users?

- Artists should be provided ADD/Update/Remove, Create Album, Statistics Functionalities how to create database so that these functions can be efficiently implemented?

- Customer have fear and to remove fear of being misguided by online cheaters what to be done at Artists side?

- In general Cart and Add to favourite functionalities should be there.

- Customer want system to be Responsive and Available at any time.
- Online payment options and payment history should be there.
- Like old time Radio new functionality Audio Streaming should be there.

- For frequent Buyers Discounts should be there and in general discounts should be available on online payment.

- Feedback mechanism should be there and admins should be active for customer problems.

- In general Ads Should be reduced or because of Ads system efficiency should not differ much.

- Platform should handle large audience and it should stay efficient in this kind of situations.

- System Recommendations on daily basis should be near to customer's taste.

Now This are requirements from User, Artist(Sellers) so we will try to make database in a way that this problems and requirements should be functioned using that database data.

Now we will talk about different types of users their needs and according to needs they should be provided functionalities also we will talk about specific functionalities inside different User classes.

First of all taking about our general interface which Unregistered user will typically get.

Unregistered User he/she will get sign up option and they can access many things like they can listen to some demo songs can see new songs and search songs well for buying options they need to register with site and free demo songs are limited so that will not be available every time.

Users will be able to login and enter into system then they can search songs that they may like and they are provided best songs at home page only using recommendation algorithm. User can add items to wish List and refer to it in future well For multiple songs deal they can add it to cart and also if user likes song then He can add it to his favourites.

More to it they can purchase and listen to their taste of songs and they can get surety by looking at artists verified accounts. User can give feedback about what is not working properly for him and what can be improved.

So from here we will talk about problems and requirements of users and then we talk about what entities we took to solve these problems.

User can access functionalities one of them will be **Track** playing well inside track we have taken Price, Language, Demo, Length, as Audio features Mode, loudness, Energy, Tempo for as Audio functions. User can quickly look into these things and identify his favourite music by seeing language and hearing demo as well other Audio functions.

As a solution to song choosing from various different songs we make process easy for user by adding these things in Track entity set.

Another requirement is customer Login for that we have implemented **Customer** entity set which contains First_Name, Last_Name, Date of Birth, Gender, Mobile No, Email ID, Address. Security aspects of these details are also be taken care.

Defining Different **Genre** we have took Genre entity and genre ID will be associated with Track ID itself.

For Cart functionality we took **Cart** entity set inside which Track IDs and Amounts are there.

For purchasing Music we have **Order** entity set inside which Order ID, Track ID, Original Price, Discount, Price Paid information is stored in database.

Artists has their requirements and for that we took Artists entity set In this we took Name, City, Ratting, Gender, Date of birth information will be stored in database and we also want that artist should have their accounts as Verified accounts so that customer can get satisfaction about their product.

Ratting and city will give customer a regional idea and popularity Attributes and customer can choose from Artist to Artist Accordingly.

Artist can ADD/Remove his Tracks and also update Track info whenever required so that functionality given to him. For Albums creation and details about that

we have created **Album** entity set inside which Album_name, No of songs and Release date information is stored in database.

For **Wishlist** we take Track ID, Track type information and store it to database this wish list is accessed by user at any time so we don't store much info of Tracks in it as it's only a wish.

Admin Privileges (Developer) in admin accounts all necessary things as well as privileged things should be there so that admin access many things except Artist and user Personal Information like Address and phone number.

Now after implementing this kind of database we have solved major issues and major conventional functions.

Now rest is Recommendation Algorithm, Add Favourite, Feedback mechanism, Sales and customer statistics are can be solved using database that we implemented and this some functionalities are solved without need of database like efficient browsing and giving good UI there. Tech related terms comes in picture so that will be taken care.

After all this proper problems and requirements handling mechanism we can create a good system so now we are going to see its database entity ER diagrams and using which we can infer how to store data in efficient manner.

Table 1.

SR NO	Noun	Verb
1	Music	Listen
2	Spotify	Work
3	Recommendation	Provide
4	UI	Show
5	Ads	Promote
6	Artists	Provide
7	Customer	Purchase
8	Cart	Stores
9	Wishlist	Collects
10	AutoPlay	Auto
11	A better Platform	Sign
12	We	Make
13	A feature	Do
14	It	Give
15	User	Register
16	Account	Login
17	Problem	Handle
18	Streaming	Live
19	Collaborative Filtering	Efficient
20	Security	Safe
21	Management	Manage
22	Control	Regulate
23	Copyright	Copy
24	Frequent	Occur
25	Reduce	Overcome
26	Improve	Grow
27	General	Simple
28	Preference	Up
29	Requirement	Need
30	Publicly	Free
31	Dataset	Store
32	Hybrid	Mix
33	Legal	Correct
34	Link	Direct
35	playlist	Collect
36	Editor	Edit
37	platform	Operate

38	information	provide
39	audio	Play
40	Interface	Part
41	Developer	Develop
42	UI	Design
43	Registration	Sign in
44	Requirement	Need
45	Favourite	Like
46	Implement	Make
47	Privilege	more
48	Privacy concern	Concern
49	Duplicate music	Fear
50	Browsing	Search
51	Customer’s behaviour	Trace
52	Improved UI	Better
53	Misguided	Wrong
54	Database	Store
55	Functionalities	work
56	Responsive	Smooth
57	Available	Open
58	Payment History	Must
59	Options	Choice
60	Radio	Listen
61	Streaming	Live
62	General discount	Open
63	Discount	Reduce
64	Frequent buyer	More
65	Feedback mechanism	Loop
66	Active Admin	Must
67	Advertisement	Show
68	Large Audience	Many
69	Customer’s Taste	like
70	Functioned	Made
71	Specific functionalities	deep
72	Without registration	Unregistered
73	Free demo	Limited
74	Enter in system	login
75	Mail	Flag
76	Ratting	Popular
77	content	Provide
78	Authorization	Must
79	Account	Verify
80	Website	Finish

81	Track name	
82	Track ID	
83	Track Type	
84	Length	
85	Price	
86	Language	
87	Release Date	
88	Genre ID	
89	Genre	
90	Album ID	
91	Album name	
92	No of songs	
93	Artist ID	
94	Artist Name	
95	Email ID	
96	User Name	
97	Password	
98	Ratting	
99	Gender	
100	Customer ID	
101	Customer Name	
102	Date of birth	
103	Mobile No	
104	Address	
105	Order ID	
106	Date	
107	Original Price	
108	Paid price	
109	Transaction ID	
110	Method	
111	Cart	

Table 2.

Sr no	Candidate Entity set	Candidate Attribute set	Candidate Relationship set
1	Track	Track ID, Name, Type, Length, Price, Language, Date	Listening, Make, Container, Categorized by
2	Customer	Customer ID, Customer Name, Email, Date of Birth, Mobile No, Address, Username, Password	Adds products to, Listening
3	Artist	Artist id, Name(Artist Name, Seller Name), Email, Username, Password, Gender, Ratting	Makes
4	Album	Album id, Name, Date, No of songs, ratting	Container
5	Order	Order id, Date, Original Price, Discount, Total Amount	Watch, Purchase
6	Transaction History	Transaction id, Amount, time	Watch
7	Cart	Track ID, Amount	Adds products to
8	Genre	Genre ID, track id	Categorized by
9	Employee	Employee id, Name	
10	Developer	Developer id, Name	
11	Manager	Manager id, Name	

Table 3.

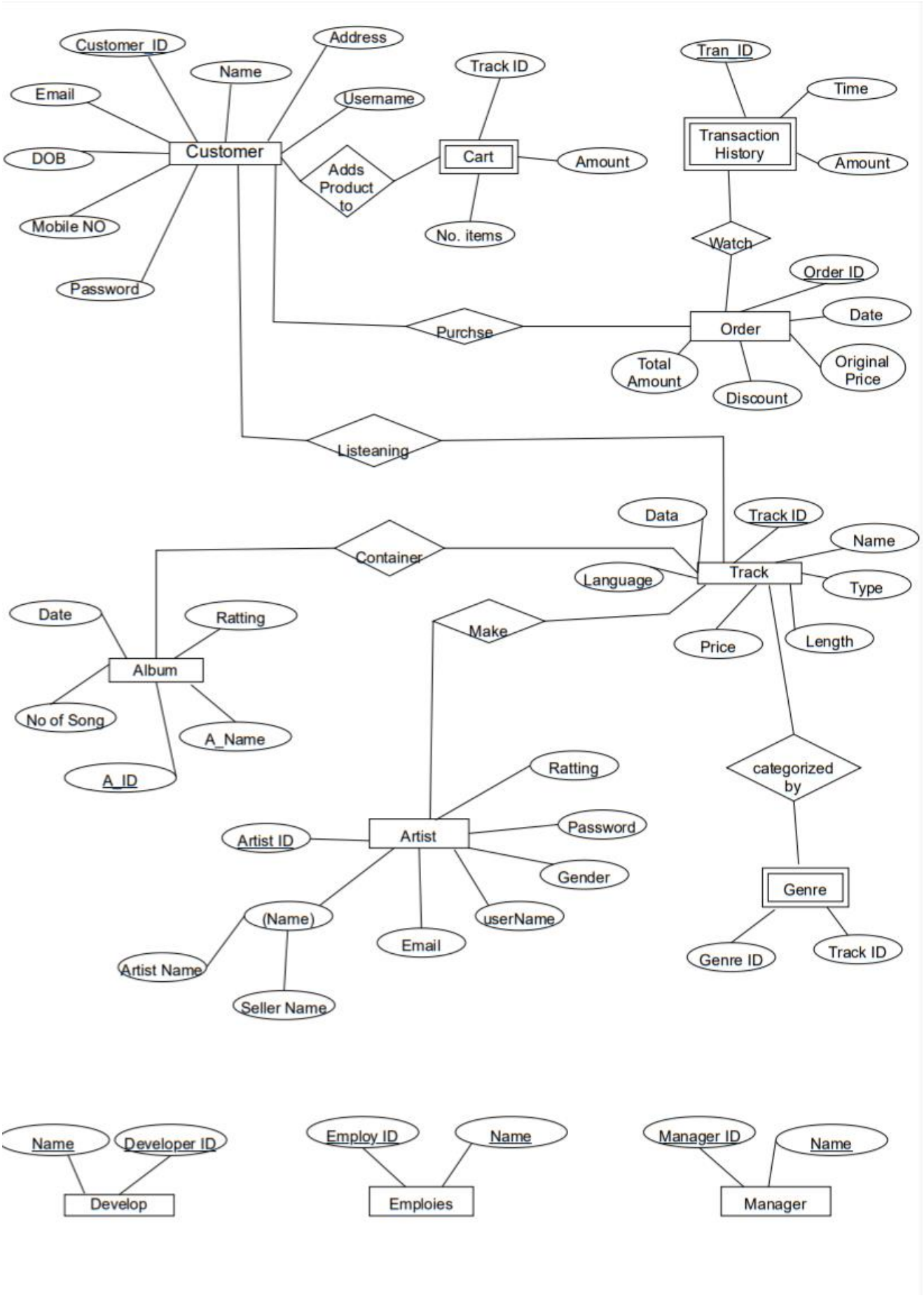
Noun	Reason	Verb	Reason
Artists	Association	Provide	Duplicate
Customers	Association	Listen	Duplicate
Password	Duplicate	Store	Duplicate/Association
Each artist	Duplicate	Access	Duplicate/Attribute
Any song	Duplicate	Upload	Duplicate
Different Privileges	General	Ratings	Duplicate/Attribute
Song	Duplicate	Download	Duplicate/General
This Song	Vague	Play	Duplicate/General
Audio	Duplicate	would	Duplicate/irrelevant
Users	Duplicate	can	Duplicate/irrelevant
Every song	Duplicate	Rhythm	Duplicate/Vague
That song	Duplicate	Satisfy	General
Song	Duplicate	exists	General
Music Label	Duplicate	Contract	General
Content	Duplicate	Visit	irrelevant
That user	Duplicate	Track	Irrelevant
That beat	Duplicate	Change	Vague
That song	Duplicate	Credits	Vague
Recorded Songs	Duplicate	Upload	General
Singer	Duplicate	Want	General
Singer	Duplicate	Earn	General
Song	Duplicate	Put	General
Song	Duplicate	make	Duplicate/General
Song	Duplicate	Should	Duplicate/General
Albums	Duplicate	make	Duplicate
Music	Duplicate	Create	Duplicate/General
Singer Details	Duplicate	Add	Duplicate/General
A song	Duplicate	Update	Duplicate/General
A Discount	Duplicate	Offers	Duplicate/Vague
A song	Duplicate	Drop	General
The specific Song	Duplicate	Search	General
Songs	Duplicate	Organize	General
Records	Duplicate	Keep	Irrelevant
Users	Duplicate	Register	Duplicate/Association
A new song	Duplicate	Notify	Duplicate/Association
Songs	Duplicate	Filtering	Duplicate
Serval Problems	Duplicate	Overcome	Irrelevant

Music Crews	Duplicate	Work	General
Music	Duplicate	Contain	General/Duplicate
Every song	Duplicate	Classify	Duplicate/Association
Total Users	Duplicate	Compute	General
Assistant Singers	Duplicate	Sing	General
Music Producers	Duplicate	Produce	General
This Platform	Duplicate	Provides	General
This Database	Duplicate	Stores	General
New Users	Duplicate	Register	General
Music composer	Duplicate	Compose	General
Singer’s Progress	Duplicate	Track	General
Site Admin	Duplicate	Help	General
Singers	Duplicate	Live Streams	General
A song	Duplicate	Description	General
Singers	Duplicate	Interaction	General/Irrelevant
An Interface	Irrelevant	Displays	General
Singer’s Policy	Irrelevant	Signs	General/Irrelevant
Basic information	General	Provides	General
User’s Expectations	General	Fulfils	General
Playlists	General	Creates	Duplicate/General
Income	General	Provides	General
Genre	General	Filter	Vague/General
Concert	General	Live	Vague
Each song	General	Collects	Vague
Unethical Logins	General	Blocks	General
This song	Duplicate	Flags	Vague/Duplicate
Singers	Duplicate	Hosts	General
Singer’s Progress	Duplicate	Analyse	Vague
Premium Features	Duplicate	Costs	General
Song’s Ratings	Vague	Gives	Duplicate/General
Bank Account Details	Vague	Provides	General
Artists	Duplicate	Verification	General
Song upload	Duplicate	Finish	Vague
Song	Duplicate	Shuffles	Vague
Permissions	Duplicate	Allows	General
Lyrics-Writer	Duplicate	Write	General
Mentor	Duplicate	Guides	General
Recording Engineer	Duplicate	Records	General
Music Editor	Duplicate	Editing	General
Music Publisher	Duplicate	Publishes	General
The Promoter	Duplicate	Promotes	General
Sound technician	Duplicate	Fix	General

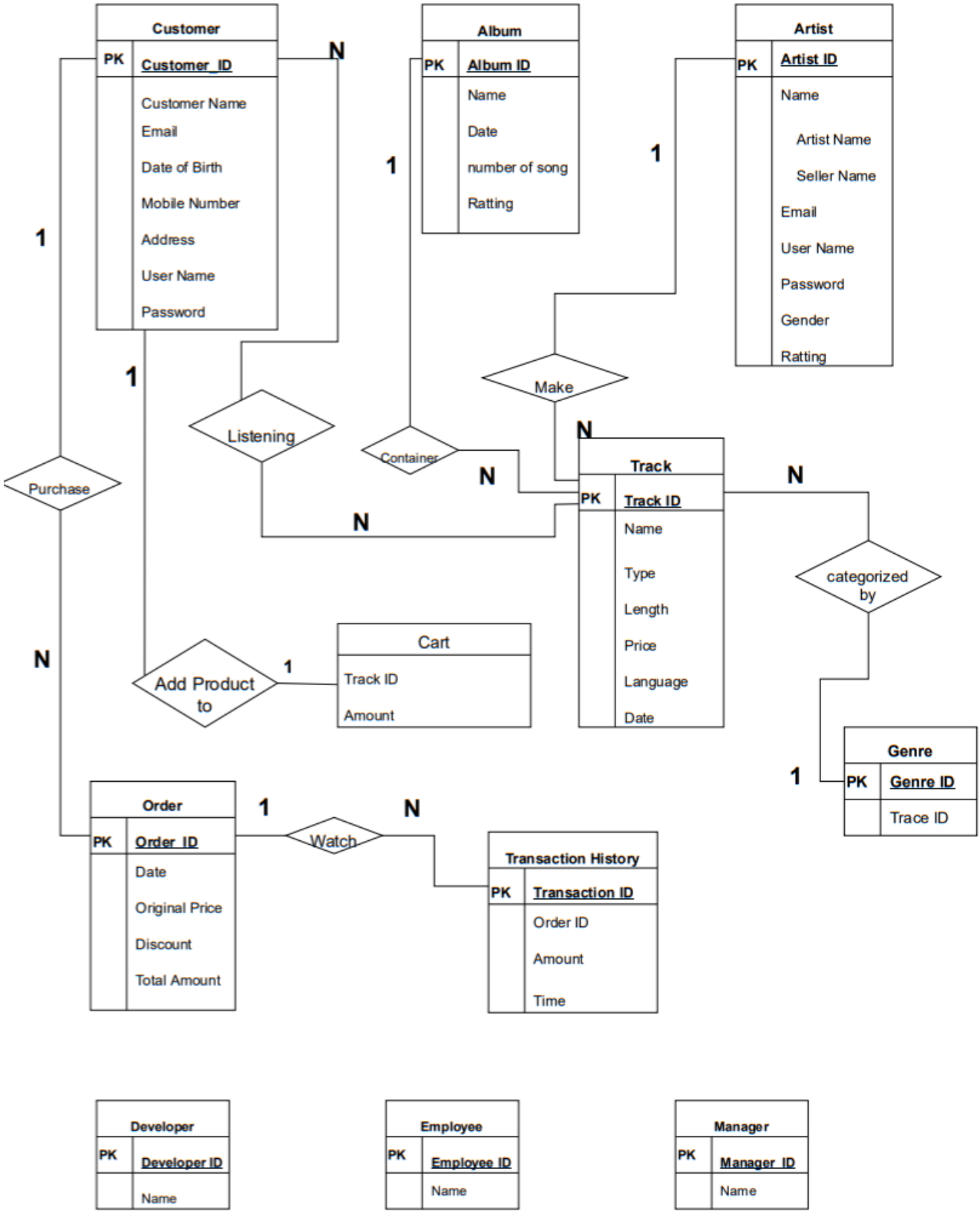
Music Teacher	Duplicate	Teach	General
Recording Mike	Duplicate	Records	General
Musician	Duplicate	Play	Vague
Choreographer	Duplicate	Choreograph	General
Music Video	Duplicate	Shoots	General
Songs Writer	Duplicate	Write	General
Site Server	Duplicate	Stores	General
Data	Duplicate	Stores	General
Credit card	Duplicate	Pay	General
Phone Number	Duplicate	Provide	General
G-Mail	Duplicate	Provide	General
Features	Duplicate	Available	General
Requirement	Duplicate	Fulfils	General
Hardware	Duplicate	Supports	General
Software	Duplicate	Supports	General
Basic information	General		
User's Expectations	General		
Playlists	General		
Income	General		
Genre	General		
Concert	General		
Unethical Logins	General		
This song	Duplicate		
Singers	Duplicate		
Ratings	Duplicate		
Internet	Duplicate		
An Interface	Irrelevant		
New Users	Duplicate		
Concert	General		
Lyrics-Writer	Duplicate		
Audio	Duplicate		
Bank Account Details	Vague		
Artists	Duplicate		
Song upload	Duplicate		

Section 3
ER-Diagram all versions

ER Diagram (Version 1)

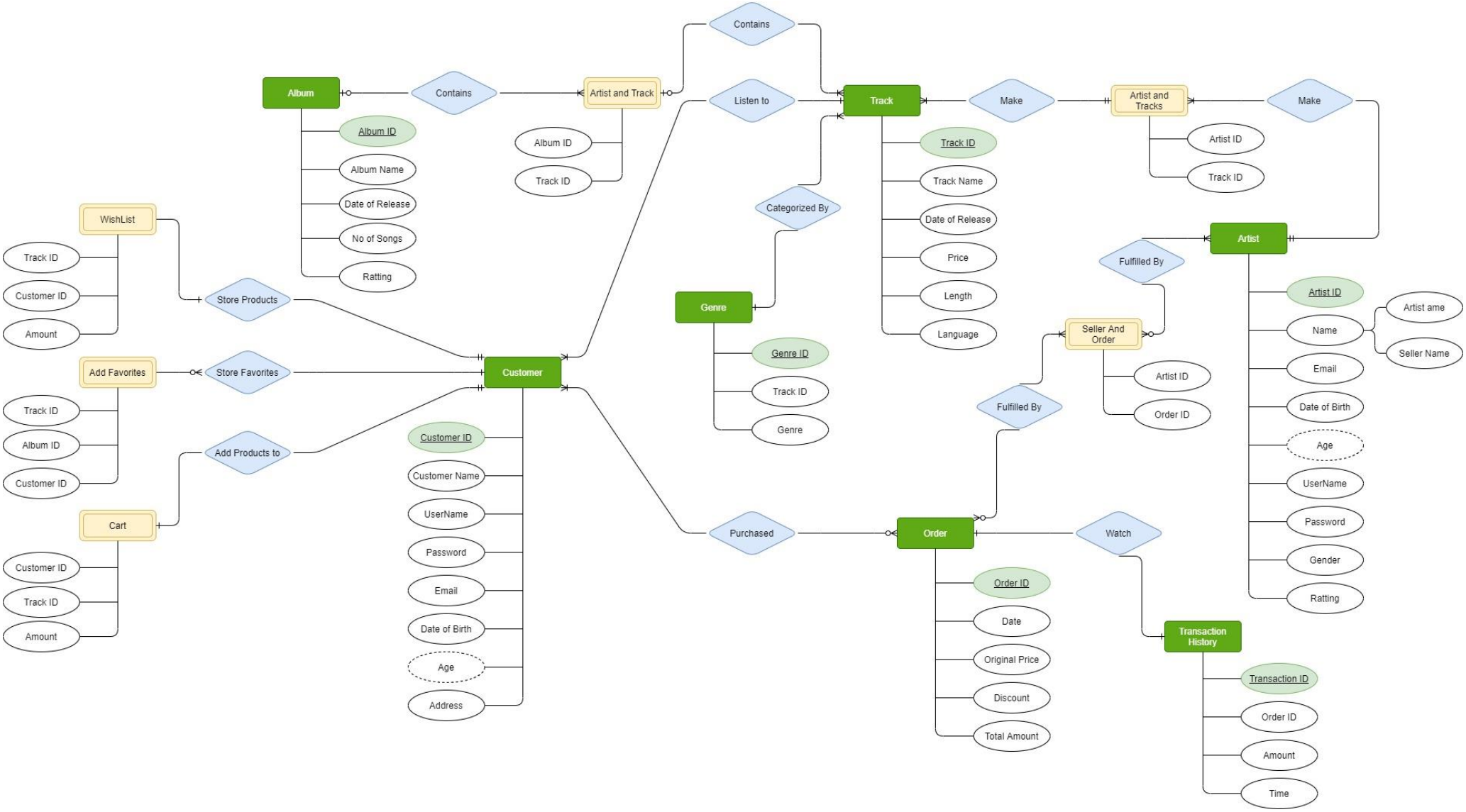


ER Diagram Final (Version 1)

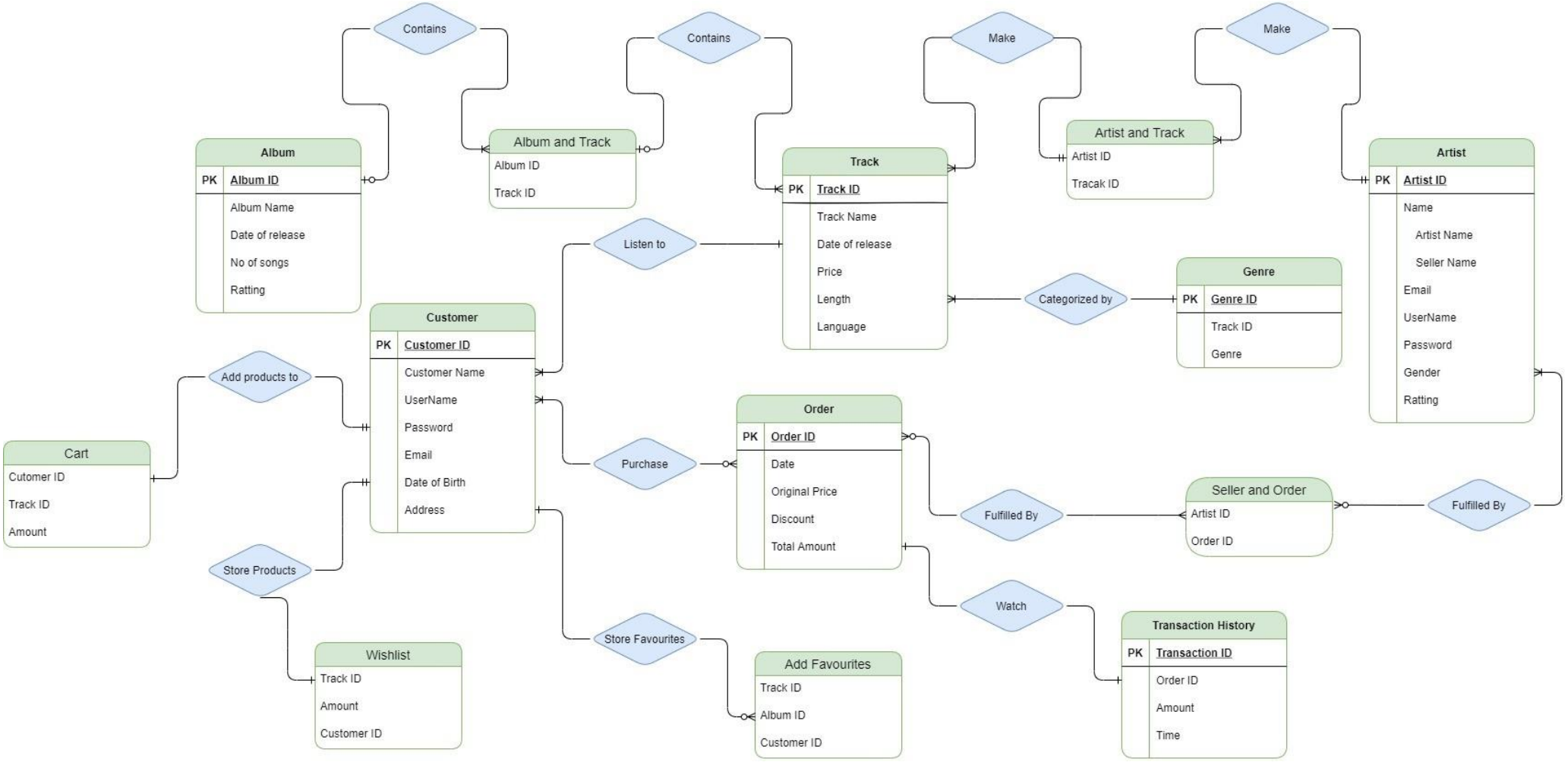


*Note: 1 and N relationship indicates one and many respectively.

ER diagram (Version 2).



ER diagram Final Version



Note: Here week entities are “Cart”, “Wishlist”, “Add Favorites”, “Seller and Order”, “Artist and Track”, “Album and Track”.
Customer and Track are identifying entity sets.
The cardinality of each relationship set is mentioned in the ER diagram itself.

Identify entity Types

As we have said above customer and Track are identifying relationships

Hierarchy:

Album->Track->Genre

Simple Association link:

Customer -> cart

Customer->Wishlist

Customer->Add favorites

Relationship Types

1. **Binary Relationships**
    a. Categorized by
    b. Purchase
    c. Listen to
    d. Add products to
    e. Store Products
    f. Store Favorites
2. **Ternary Relationships**
    a. Make
    b. Contains
    c. Fulfilled By

Section 4

Conversion of Final ER-Diagram to Relational Model

Mapping E-R Model to Relational Model

Write down all the relations with the schema.

(Format: Primary Key: **PK**, Foreign key: *FK*)

Customer: (**Customer ID**, Username, Password, Email ID, Customer name, Date of Birth, Address, Age).

Track: (**Track ID**, *Genre ID*, Track Name, Date of Release, Price, Length, Language).

Foreign Key Genre ID references to Genre.

Album: (**Album ID**, Album Name, Date of Release, No of Songs, Ratting).

Artist: (**Artist ID**, Name (Artist Name, Seller Name), Email ID, Username, Password, Gender, Ratting).

Order: (**Order ID**, Date, Original Price, Discount, Total Amount).

Cart: (*Customer ID*, *Track ID*, Amount).

Foreign Key Customer ID references to Customer.

Foreign Key Track ID references to Track.

Wishlist: (*Track ID*, *Customer ID*, Amount).

Foreign Key Customer ID references to Customer.

Foreign Key Track ID references to Track.

Transaction History: (**Transaction ID**, *Order ID*, Amount, time).

Foreign Key Order ID references to Order.

Add Favorites: (*Track ID*, *Album ID*, *Customer ID*).

Foreign Key Track ID references to Track.

Foreign Key Album ID references to Album.

Foreign Key Customer ID references to Customer.

Genre: (**Genre ID**, Genre).

Listen: (*Track ID, Customer ID*).

Foreign Key Track ID references to Track.

Foreign Key Customer ID references to Customer.

Album and Track: (*Album Id, Track ID*).

Foreign Key Track ID references to Track.

Foreign Key Album ID references to Album.

Artist and Track: (*Artist ID, Track ID*).

Foreign Key Track ID references to Track.

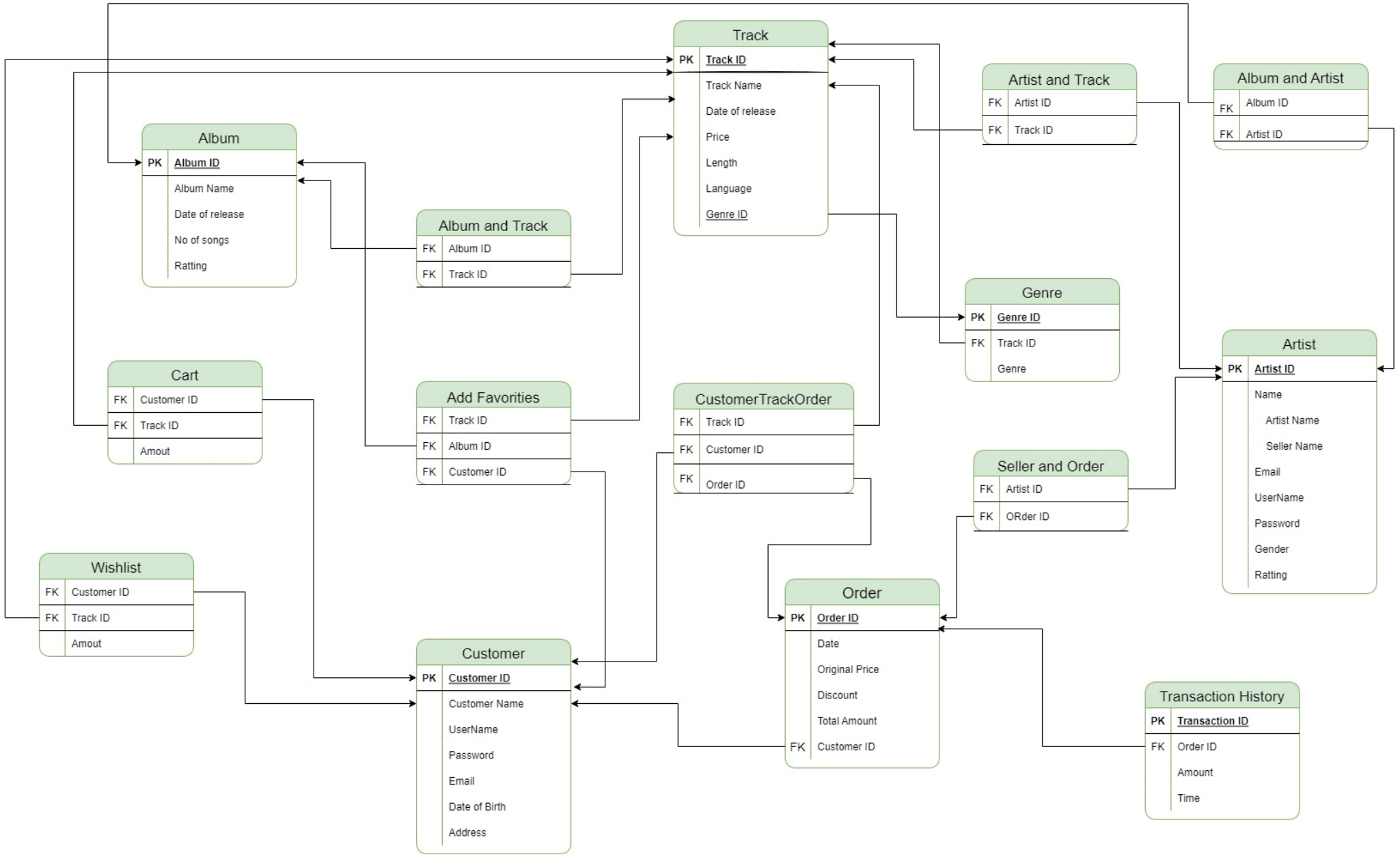
Foreign Key Artist ID references to Artist.

Seller and Order: (*Artist ID, Order ID*).

Foreign Key Artist ID references to Artist.

Foreign Key Order ID references to Order.

Relational Model



Section 5
Normalization and Schema Refinement

Normalization and Schema Refinement

Old Relations with The Schema

(Format: Primary Key: **PK**, Foreign key: *FK*)

Customer: (**Customer ID**, Username, Password, Email ID, Customer name, Date of Birth, Address, Age).

Track: (**Track ID**, *Genre ID*, Track Name, Date of Release, Price, Length, Language).

Foreign Key Genre ID references to Genre.

Album: (**Album ID**, Album Name, Date of Release, No of Songs, Ratting).

Artist: (**Artist ID**, Name (Artist Name, Seller Name), Email ID, Username, Password, Gender, Ratting).

Order: (**Order ID**, Date, Original Price, Discount, Total Amount).

Cart: (*Customer ID*, *Track ID*, Amount).

Foreign Key Customer ID references to Customer.

Foreign Key Track ID references to Track.

Wishlist: (*Track ID*, *Customer ID*, Amount).

Foreign Key Customer ID references to Customer.

Foreign Key Track ID references to Track.

Transaction History: (**Transaction ID**, *Order ID*, Amount, time).

Foreign Key Order ID references to Order.

Add Favorites: (*Track ID, Album ID, Customer ID*).

Foreign Key Track ID references to Track.

Foreign Key Album ID references to Album.

Foreign Key Customer ID references to Customer.

Genre: (**Genre ID**, Genre).

Listen: (*Track ID, Customer ID*).

Foreign Key Track ID references to Track.

Foreign Key Customer ID references to Customer.

Album and Track: (*Album Id, Track ID*).

Foreign Key Track ID references to Track.

Foreign Key Album ID references to Album.

Artist and Track: (*Artist ID, Track ID*).

Foreign Key Track ID references to Track.

Foreign Key Artist ID references to Artist.

Seller and Order: (*Artist ID, Order ID*).

Foreign Key Artist ID references to Artist.

Foreign Key Order ID references to Order.

Artist and Album: (Artist ID, Album ID)

Foreign Key Artist ID references to Artist.

Foreign Key Album ID references to Album.

Final Relations with The Schemes

(Format: Primary Key: **PK**, Foreign key: *FK*)

Customer: (**Customer ID**, Username, Password, Email ID, Customer name, Date of Birth, Address, Age).

- Primary Key: Customer ID
- Foregine Key: Not Exist
- Partial Dependency: Customer ID -> Email ID, Username but we are taking unique Email ID so this will not create any redundant info in our database as well as we are taking unique username also so partial dependancies could be totally removed.
- Redundancy: Here Email ID and Username repeated for some Particular Customer ID but as we have said it and made it unique so it will not create any kind of redundancy.
- Update/Insert or Delete anomalies won't be created.
- Our table is already in 1NF because it doesn't contain multiple attributes and relation with single-attribute primary key is automatically becomes 2 NF.
- No non-primary key attribute is transitively dependent on the primary key so it's in 3 NF.
- we conclude that our table remains as it is.

Track: (**Track ID**, *Genre ID*, Track Name, Date of Release, Price, Length, Language).

Foreign Key Genre ID references to Genre.

- Foregine Key: Genre ID references to Genre.
- Partial Dependency: (Track ID) -> Track ID,Genre ID.
- Redundancy: Here Track ID and Genre ID is repeated for some Particular Track but it will not create redundant information because its primary key for Genre table so we don't define it as strong dependency instead we take it as foreign key so it will not create much redundant info.
- Update/ insert or delete anomalies will not happen here.
- Our table is already in 1NF because it doesn't contain multiple attributes and relation with single-attribute primary key is automatically becomes 2 NF.
- No non-primary key attribute is transitively dependent on the primary key so it's in 3 NF.
- we conclude that our table remains as it is.

Album: (**Album ID**, Album Name, Date of Release, No of Songs, Ratting).

- Primary Key:(Album ID) -> Album Name.
- Foreign Key: Not Exist.
- Partial Dependency:(Album ID) -> Album ID, Album Name.
- Redundancy: Many artists can upload their tracks on same day but we have taken timestamp as datatype so we are having time also and that's why It will not create any redundant information.
- Update/ insert or delete anomalies will not happen here.
- Our table is already in 1NF because it doesn't contain multiple attributes and relation with single-attribute primary key is automatically becomes 2 NF.
- No non-primary key attribute is transitively dependent on the primary key so it's in 3 NF.
- we conclude that our table remains as it is.

Artist: (**Artist ID**, Name (Artist Name, Seller Name), Email ID, Username, Password, Gender, Ratting).

- Primary Key:(Artist ID) -> Artist Name, Email ID, Username.
- Foreign Key: Not Exist.
- Partial Dependency:(Artist ID) -> Email ID, Username this two fields can create dependency but we have taken unique Email ID and Username so we don't have any dependency.
- Redundancy: As we don't have multiple attributes so we will not have any redundant information.
- Update/ insert or delete anomalies will not happen here.
- Our table is already in 1NF because it doesn't contain multiple attributes and relation with single-attribute primary key is automatically becomes 2 NF.
- No non-primary key attribute is transitively dependent on the primary key so it's in 3 NF.
- we conclude that our table remains as it is.

Order: (**Order ID**, Date, Original Price, Discount, Total Amount).

- Primary Key:(Order ID).
- Foreign Key: Not Exist.
- Partial Dependency: There is no dependency here other than primary key itself.
- Redundancy: as we don't have any dependency so redundant info can't be generated.
- Update/ insert or delete anomalies will not happen here.
- Our table is already in 1NF because it doesn't contain multiple attributes and relation with single-attribute primary key is automatically becomes 2 NF.
- No non-primary key attribute is transitively dependent on the primary key so it's in 3 NF.
- we conclude that our table remains as it is.

Cart: (*Customer ID*, *Track ID*, Amount).

Foreign Key Customer ID references to Customer.

Foreign Key Track ID references to Track.

Wishlist: (*Track ID*, *Customer ID*, Amount).

Foreign Key Customer ID references to Customer.

Foreign Key Track ID references to Track.

Transaction History: (**Transaction ID**, *Order ID*, Amount, time).

Foreign Key Order ID references to Order.

1NF - The Schema is already in 1st NF as it has no multi valued attributes

2NF - The schema is already in 2nd NF because it's all attributes are full functional dependent on primary keys c)

3NF - No non-primary key attribute is transitively dependent on the primary key.

Add Favorites: (*Track ID*, *Album ID*, *Customer ID*).

Foreign Key Track ID references to Track.

Foreign Key Album ID references to Album.

Foreign Key Customer ID references to Customer.

Genre: (**Genre ID**, Genre).

1NF - The Schema is already in 1st NF as it has no multi valued attributes

2NF - The schema is already in 2nd NF because it's all attributes are full functional dependent on primary keys c)

3NF - No non-primary key attribute is transitively dependent on the primary key.

Listen: (*Track ID*, *Customer ID*).

Foreign Key Track ID references to Track.

Foreign Key Customer ID references to Customer.

Album and Track: (*Album Id*, *Track ID*).

Foreign Key Track ID references to Track.

Foreign Key Album ID references to Album.

Artist and Track: (*Artist ID*, *Track ID*).

Foreign Key Track ID references to Track.

Foreign Key Artist ID references to Artist.

Seller and Order: (*Artist ID*, *Order ID*).

Foreign Key Artist ID references to Artist.

Foreign Key Order ID references to Order.

Artist and Album: (Artist ID, Album ID)

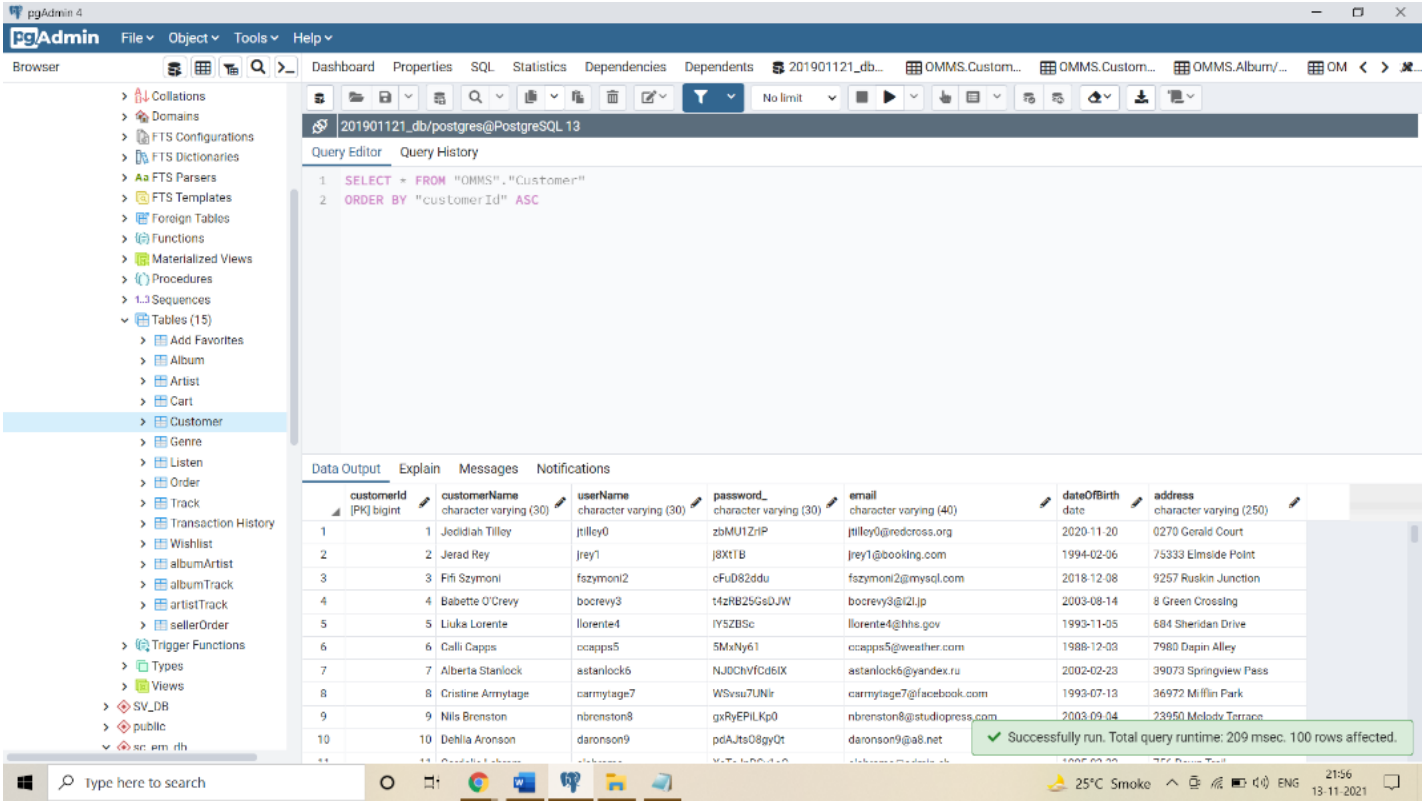
Foreign Key Artist ID references to Artist.

Foreign Key Album ID references to Album.

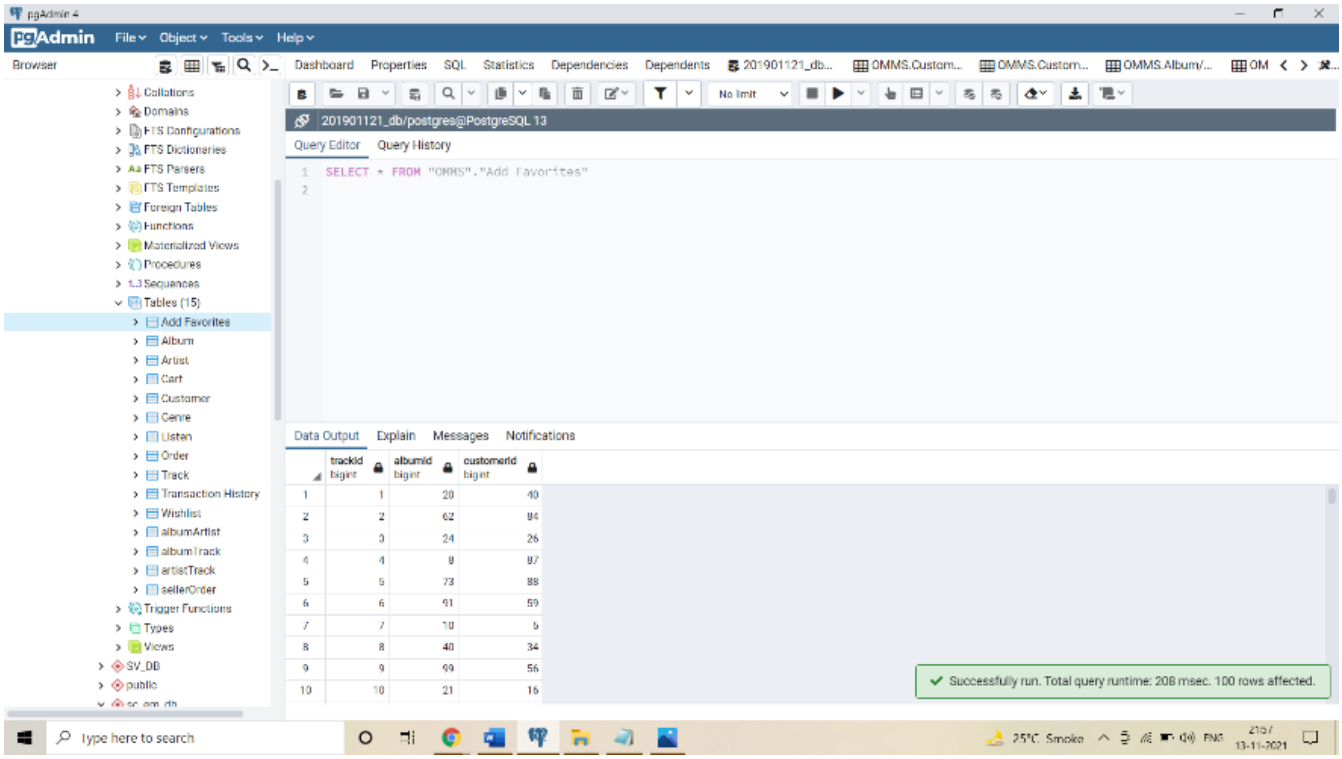
Section6

SQL: Final DDL Scripts, Insert statements, 40 SQL Queries with Snapshots of output of each query.

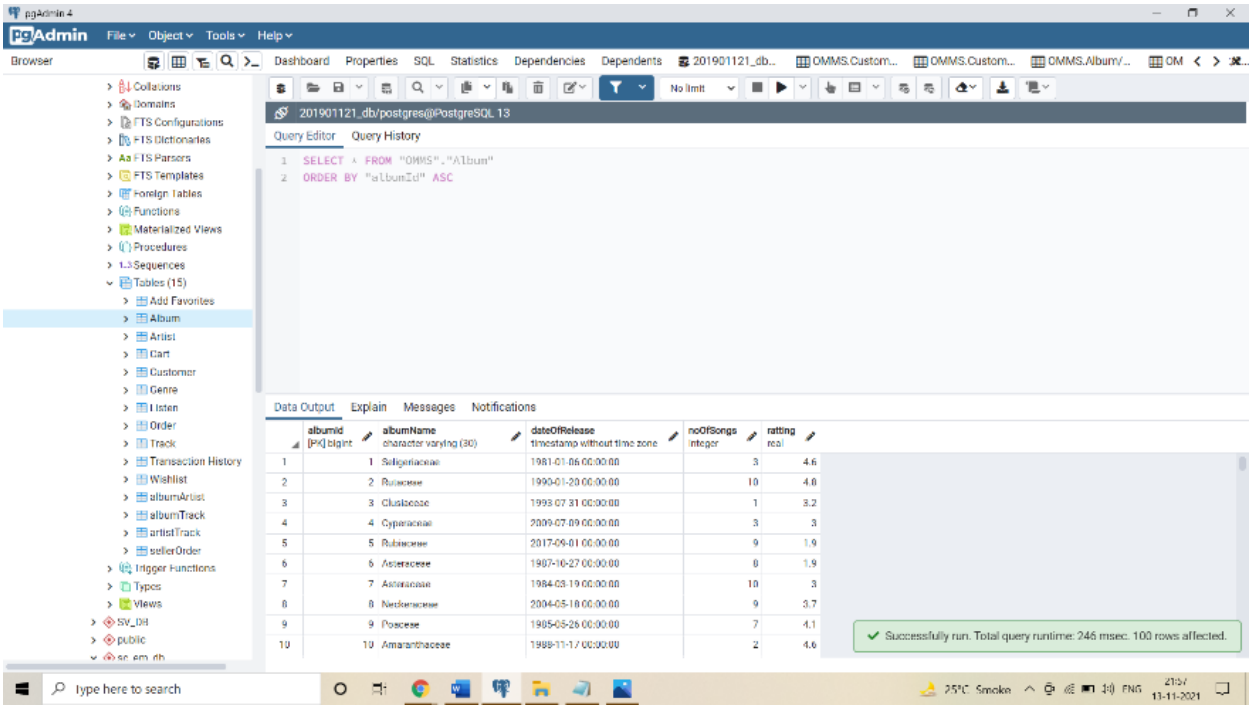
PostgreSQL Inserted Data Screenshots



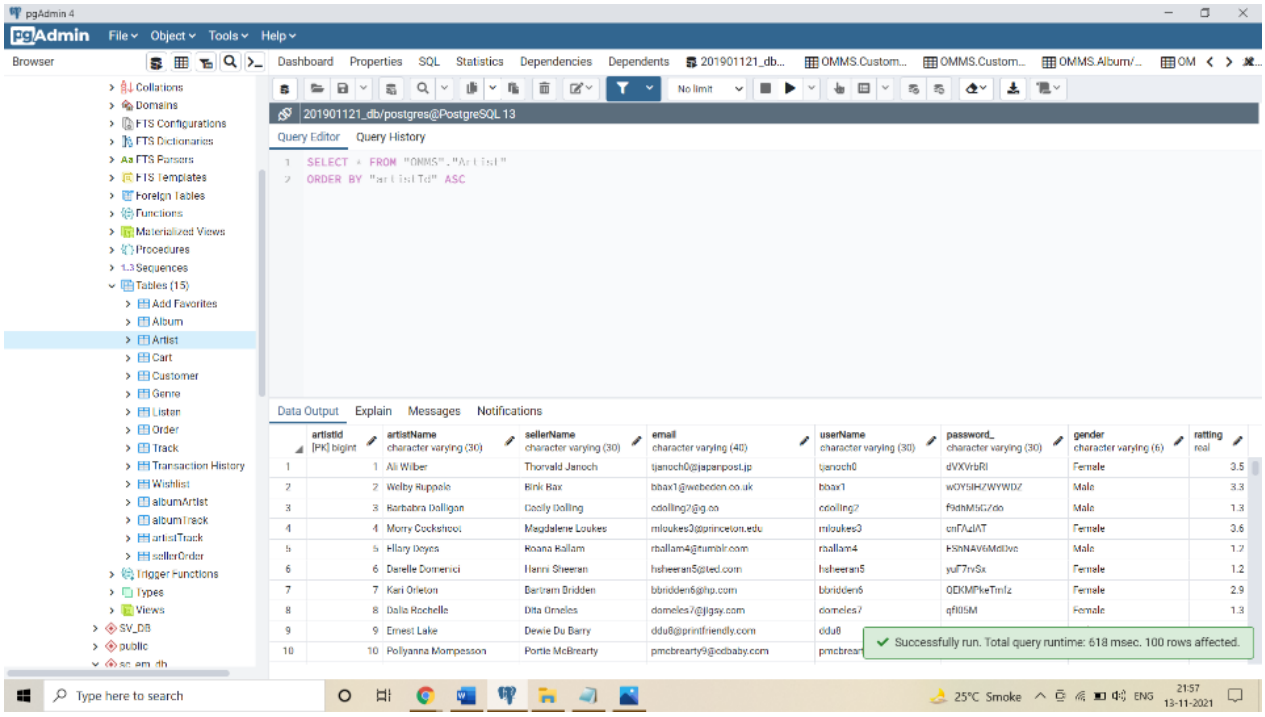
Customer



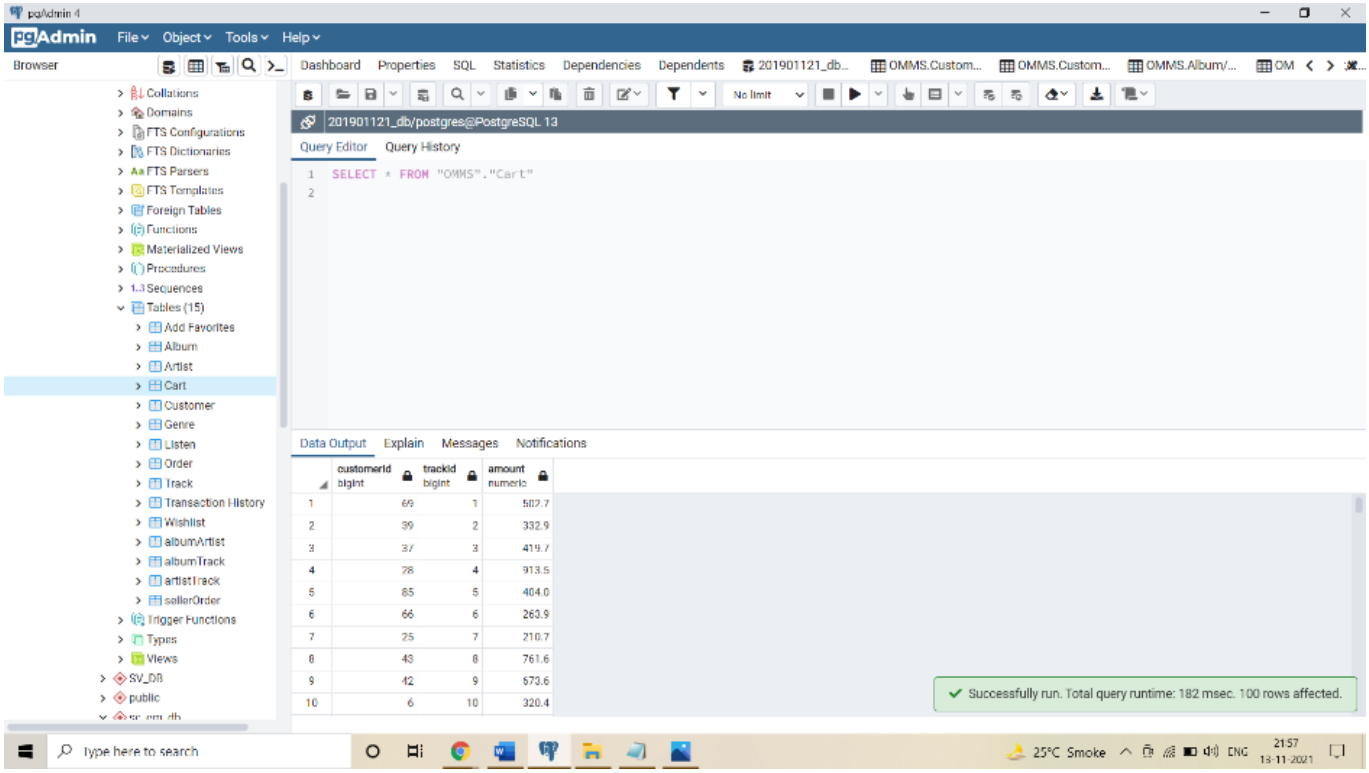
Add Favorites



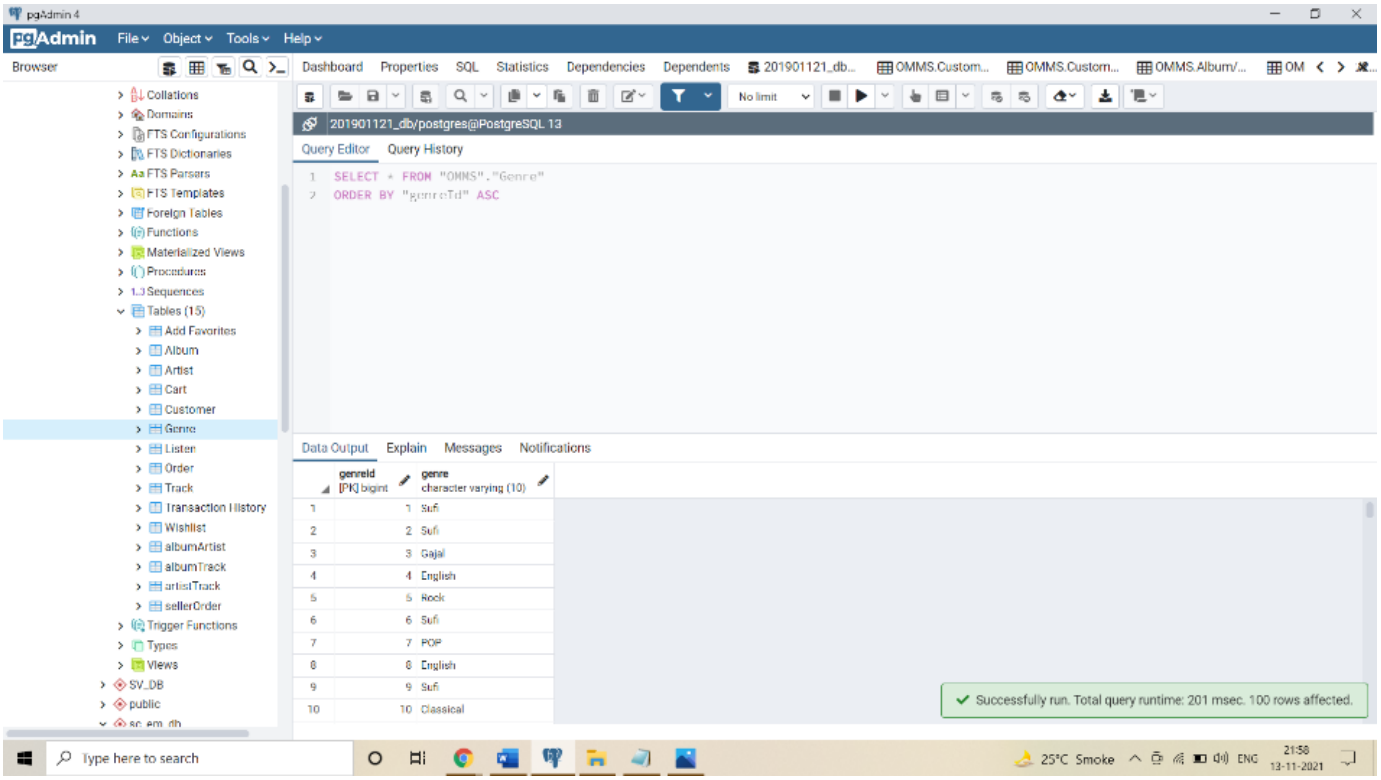
Album


pgAdmin 4
File Object Tools Help
Browser
Dashboard Properties SQL Statistics Dependencies Dependents 201901121_db... OMMS.Custom... OMMS.Custom... OMMS.Album/... OM
201901121_db/postgres@PostgreSQL 13
Query Editor Query History
SELECT * FROM "OMMS"."Artist"
ORDER BY "artistId" ASC
Data Output Explain Messages Notifications
artistId [PK] bigint
artistName character varying (30)
sellerName character varying (30)
email character varying (40)
userName character varying (30)
password_ character varying (30)
gender character varying (6)
ratting real
1 1 Ali Wiber Thorvald Janoch tjanoch0@japanpost.jp tjanoch0 dVXVrbRI Female 3.5
2 2 Welby Ruppele Bink Bax bbax1@webeden.co.uk bbax1 wOY5lHZWYWDZ Male 3.3
3 3 Barbabra Dolligan Cecily Dolling cdolling2@g.co cdolling2 f9dhM5GZdo Male 1.3
4 4 Morry Cockshtoot Magdalene Loukes mloukes3@princeton.edu mloukes3 cnfAzlAT Female 3.6
5 5 Hilary Deyes Roana Rallam rhallam4@tumblr.com rhallam4 F5hNAV6MdIvc Male 1.7
6 6 Darelle Domenici Hanni Sheeran hsheeran5@ted.com hsheeran5 yuf7rvSx Female 1.2
7 7 Kari Orleton Bertram Bridden bbridden6@hp.com bbridden6 QDKMPkeTmfz Female 2.9
8 8 Dalia Rochelle Dita Omeles domeles7@jigsy.com domeles7 qfl05M Female 1.3
9 9 Ernest Lake Dewie Du Barry ddu8@printfriendly.com ddu8 Female
10 10 Pollyanna Mompesson Portie McBrearty pmcbrearty9@cdbaby.com pmcbrear
Successfully run. Total query runtime: 618 msec. 100 rows affected.
Type here to search
25°C Smoke ENG 21:57 13-11-2021

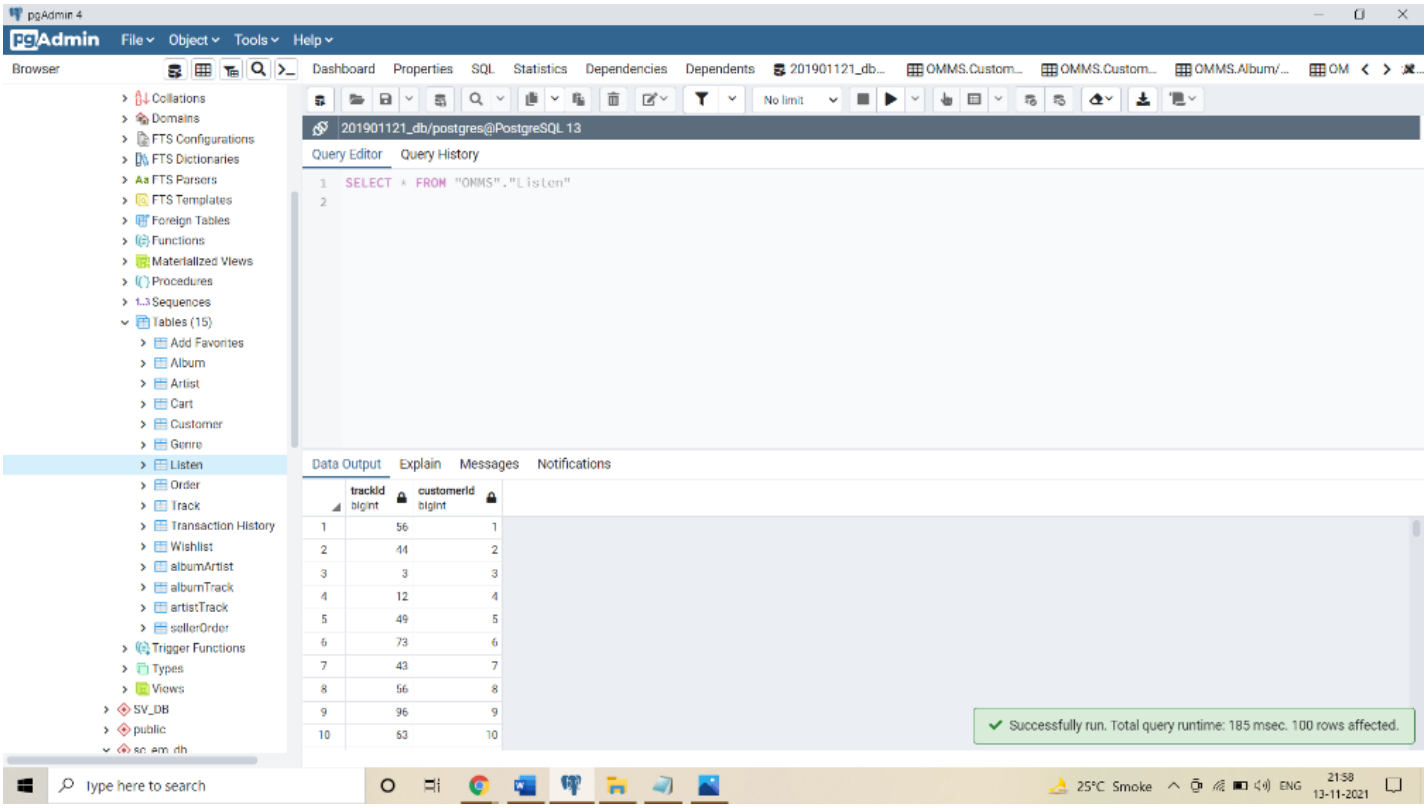
Artist


pgAdmin 4
File Object Tools Help
Browser
Dashboard Properties SQL Statistics Dependencies Dependents 201901121_db... OMMS.Custom... OMMS.Custom... OMMS.Album/... OM
201901121_db/postgres@PostgreSQL 13
Query Editor Query History
SELECT * FROM "OMMS"."Cart"
Data Output Explain Messages Notifications
customerId bigint
trackId bigint
amount numeric
1 69 1 502.7
2 39 2 332.9
3 37 3 419.7
4 28 4 913.5
5 85 5 404.0
6 66 6 263.9
7 25 7 210.7
8 43 8 761.6
9 42 9 573.6
10 6 10 320.4
Successfully run. Total query runtime: 182 msec. 100 rows affected.
Type here to search
25°C Smoke ENG 21:57 13-11-2021

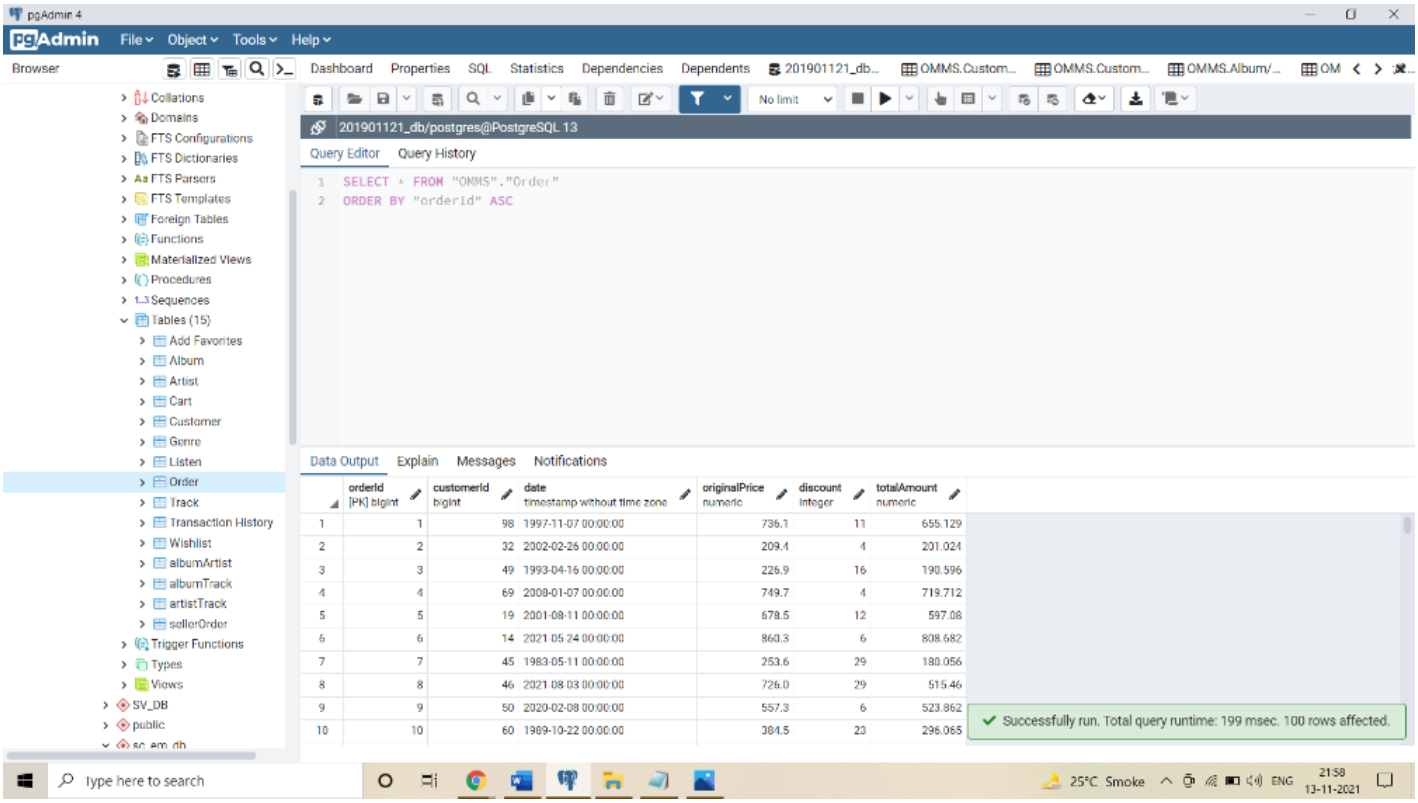
Cart



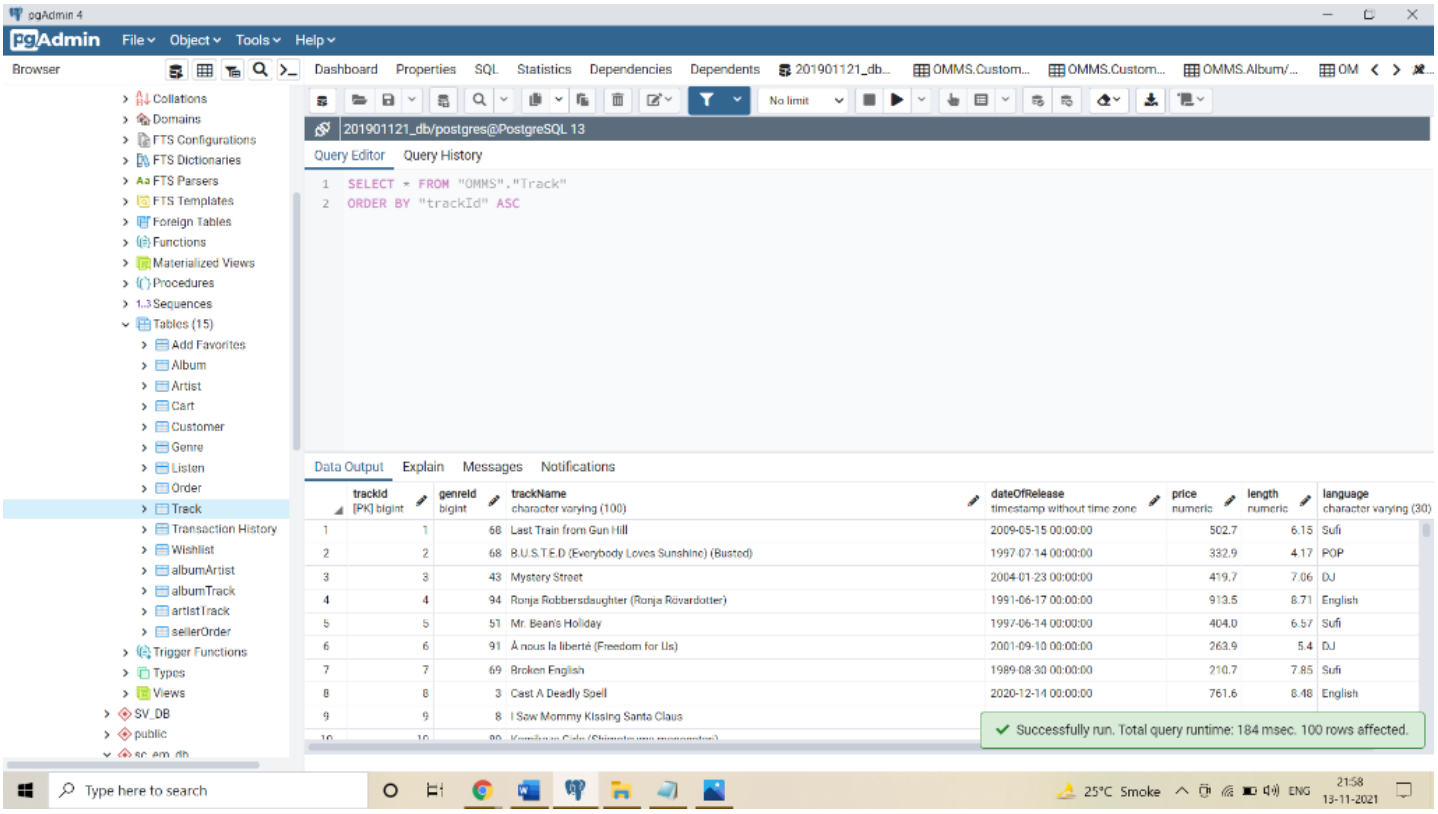
Genre



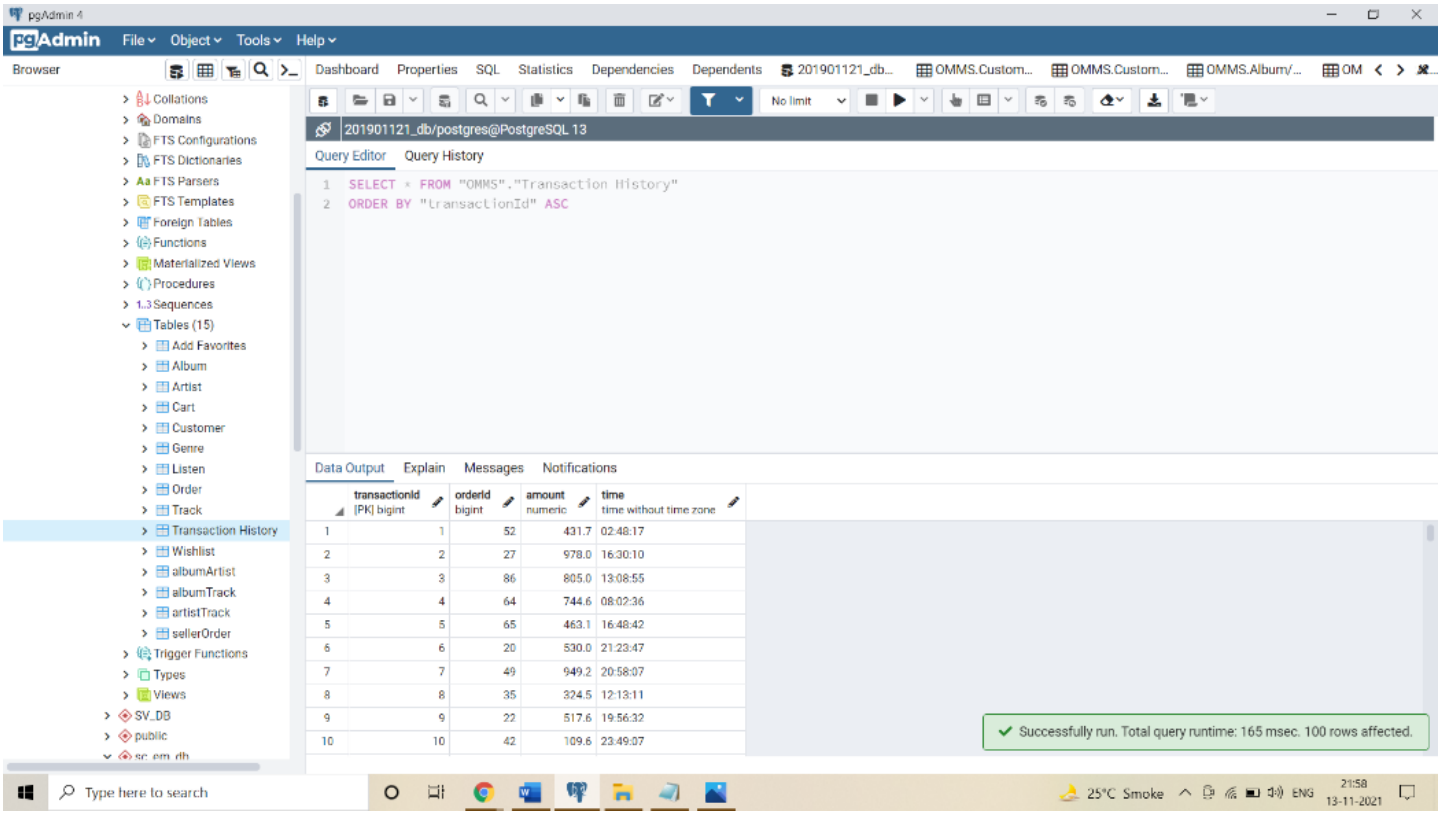
Listen


pgAdmin 4
File Object Tools Help
Browser
Dashboard Properties SQL Statistics Dependencies Dependents 201901121_db... OMMS.Custom... OMMS.Custom... OMMS.Album/... OM
No limit
201901121_db/postgres@PostgreSQL 13
Query Editor Query History
SELECT * FROM "OMMS"."Order"
ORDER BY "orderId" ASC
Data Output Explain Messages Notifications
orderId [PK] bigint
customerId bigint
date timestamp without time zone
originalPrice numeric
discount integer
totalAmount numeric
1 1 98 1997-11-07 00:00:00 736.1 11 655.129
2 2 32 2002-02-26 00:00:00 209.4 4 201.024
3 3 49 1993-04-16 00:00:00 226.9 16 190.596
4 4 69 2008-01-07 00:00:00 749.7 4 719.712
5 5 19 2001-08-11 00:00:00 678.5 12 597.08
6 6 14 2021-05-24 00:00:00 860.3 6 808.682
7 7 45 1983-05-11 00:00:00 253.6 29 180.056
8 8 46 2021-08-03 00:00:00 726.0 29 515.46
9 9 50 2020-02-08 00:00:00 557.3 6 523.862
10 10 60 1989-10-22 00:00:00 384.5 23 296.065
Successfully run. Total query runtime: 199 msec. 100 rows affected.
Type here to search
25°C Smoke
ENG
21:58
13-11-2021

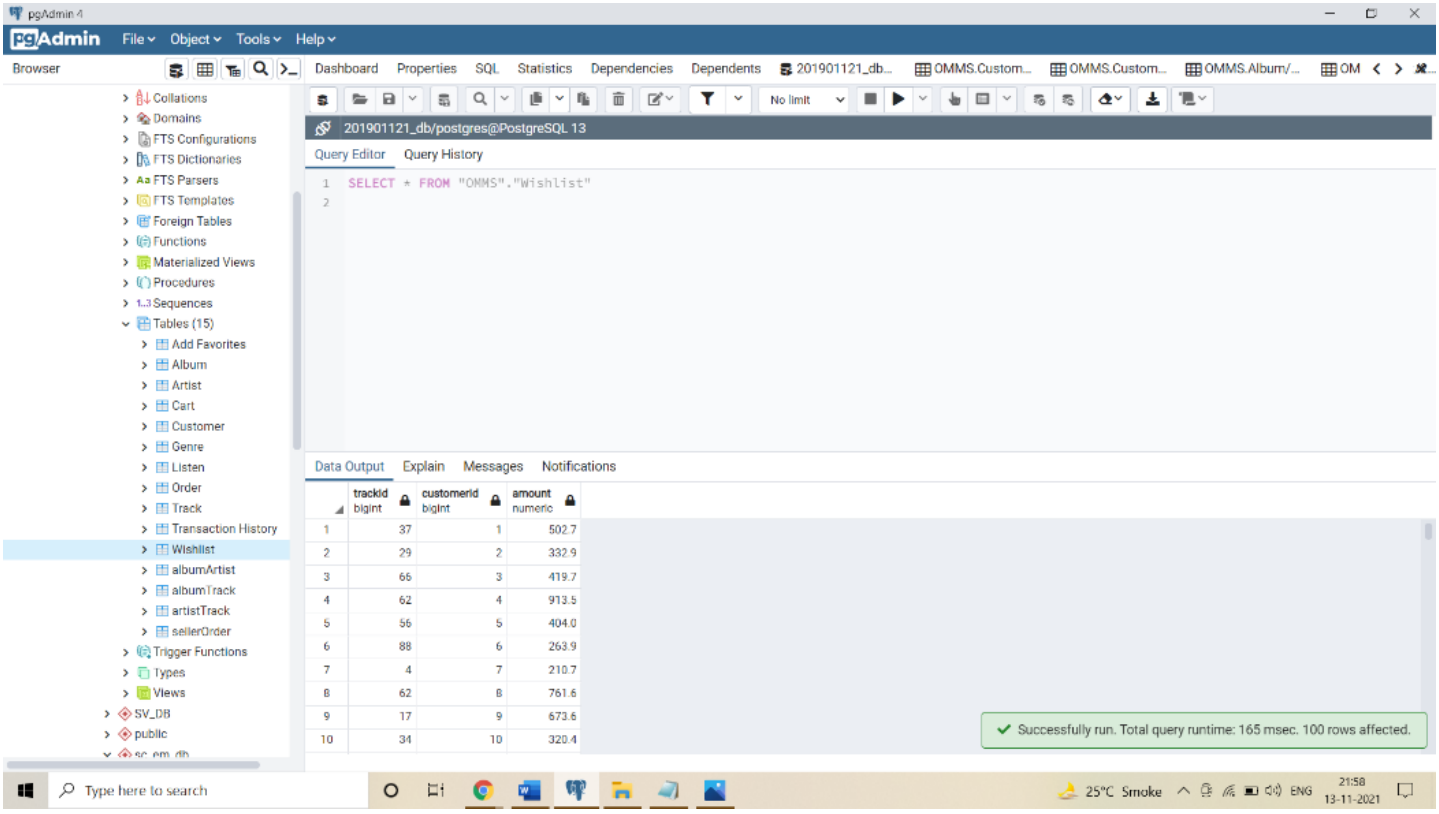
Order


pgAdmin 4
File Object Tools Help
Browser
Dashboard Properties SQL Statistics Dependencies Dependents 201901121_db... OMMS.Custom... OMMS.Custom... OMMS.Album/... OM
No limit
201901121_db/postgres@PostgreSQL 13
Query Editor Query History
SELECT * FROM "OMMS"."Track"
ORDER BY "trackId" ASC
Data Output Explain Messages Notifications
trackId [PK] bigint
genreId bigint
trackName character varying (100)
dateOfRelease timestamp without time zone
price numeric
length numeric
language character varying (30)
1 1 68 Last Train from Gun Hill 2009-05-15 00:00:00 502.7 6.15 Sufi
2 2 68 B.U.S.T.E.D (Everybody Loves Sunshine) (Busted) 1997-07-14 00:00:00 332.9 4.17 POP
3 3 43 Mystery Street 2004-01-23 00:00:00 419.7 7.06 DJ
4 4 94 Ronja Robbersdaughter (Ronja Rövardotter) 1991-06-17 00:00:00 913.5 8.71 English
5 5 51 Mr. Beans Holiday 1997-06-14 00:00:00 404.0 6.57 Sufi
6 6 91 À nous la liberté (Freedom for Us) 2001-09-10 00:00:00 263.9 5.4 DJ
7 7 69 Broken English 1989-08-30 00:00:00 210.7 7.85 Sufi
8 8 3 Cast A Deadly Spell 2020-12-14 00:00:00 761.6 8.48 English
9 9 8 I Saw Mommy Kissing Santa Claus
Successfully run. Total query runtime: 184 msec. 100 rows affected.
Type here to search
25°C Smoke
ENG
21:58
13-11-2021

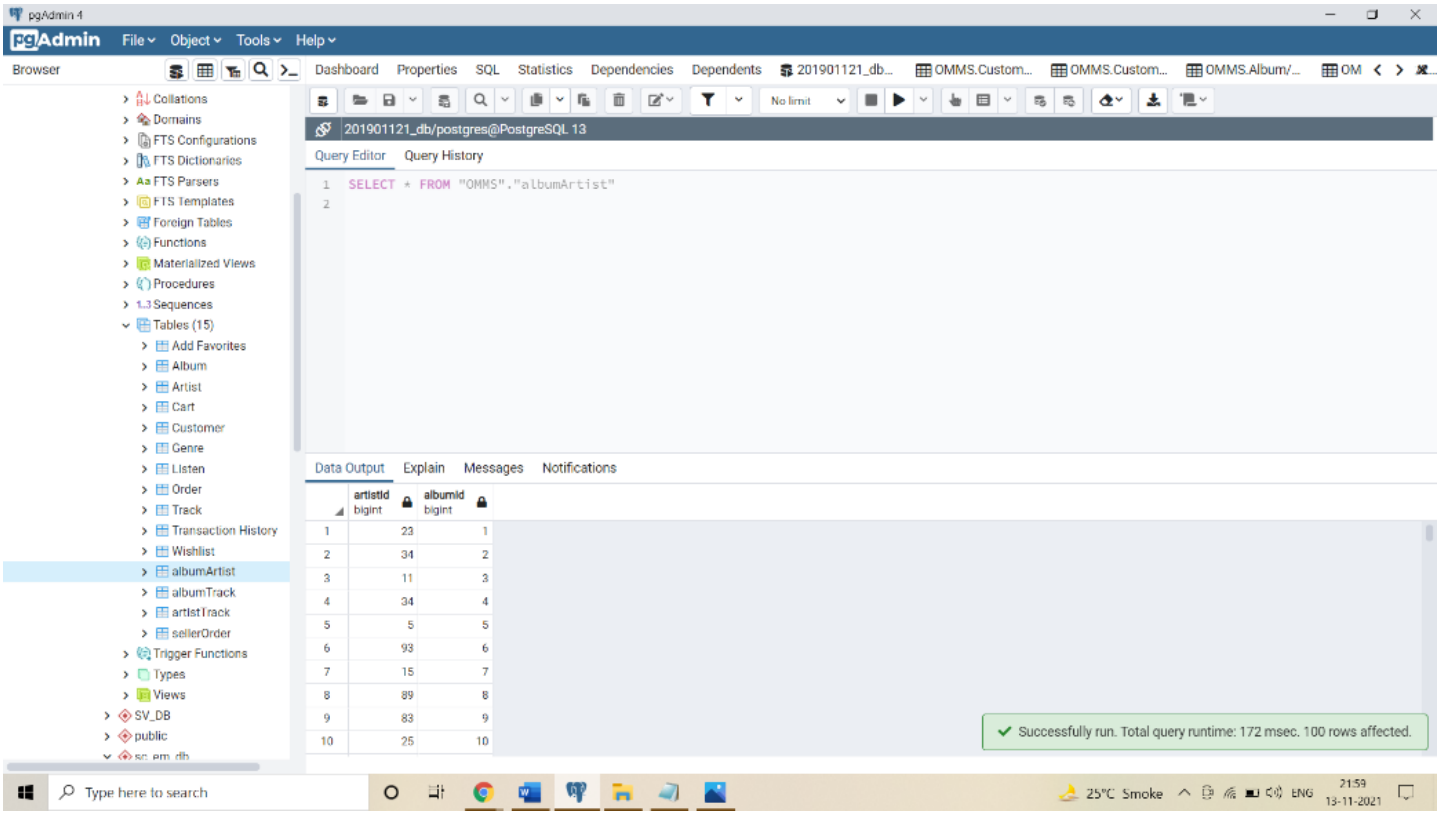
Track



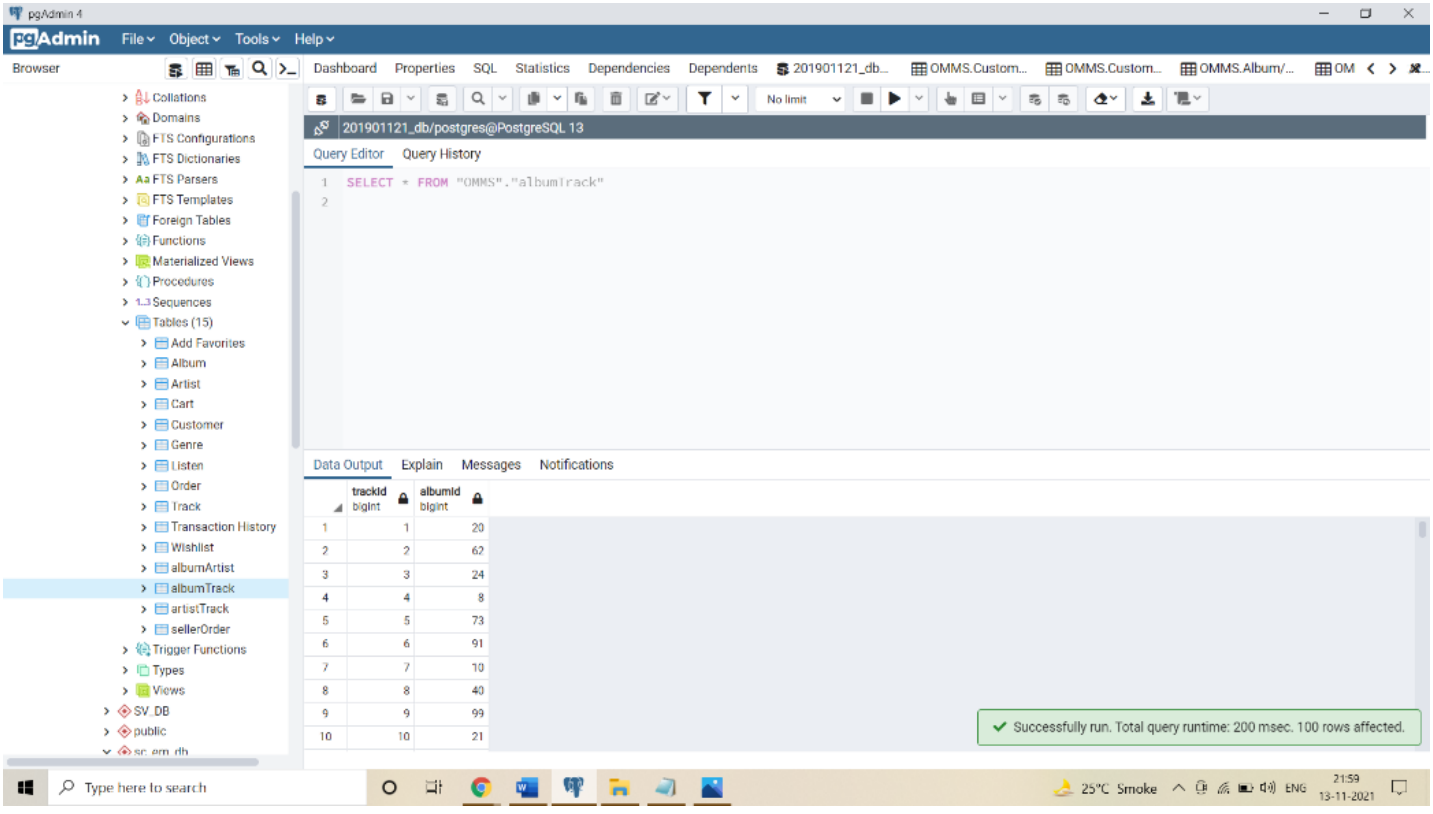
Transaction History



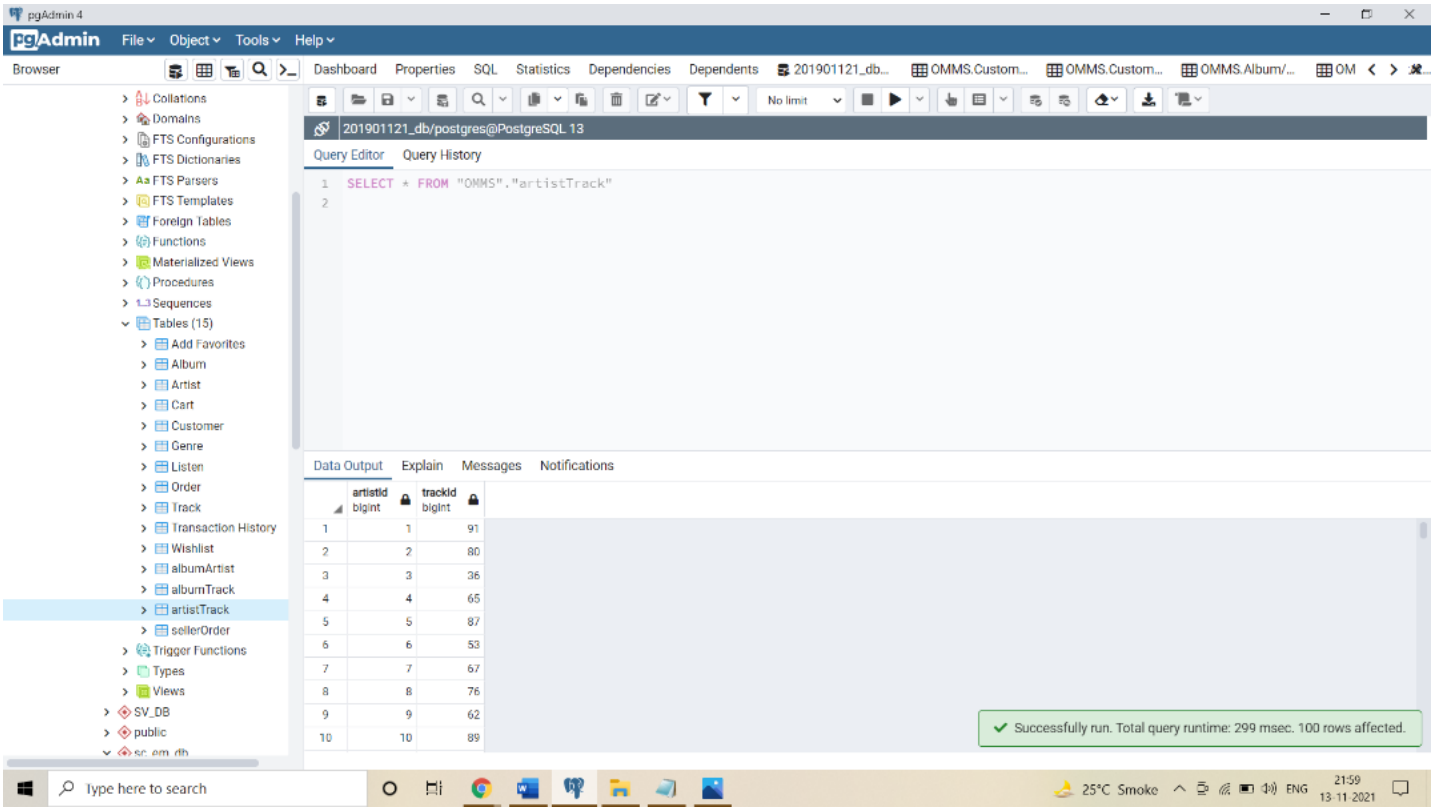
Wishlist



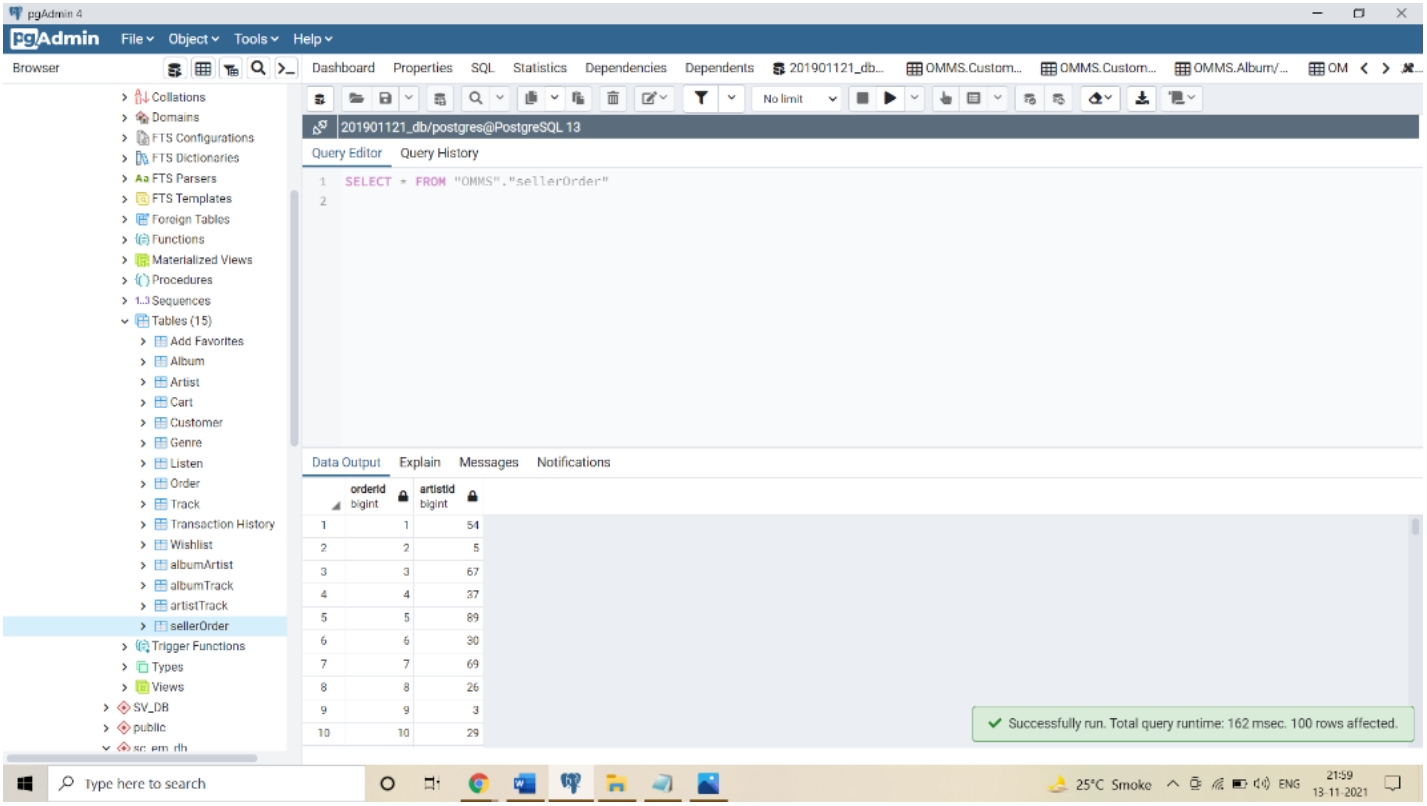
Album and Artist



Album and Track



Artist and Track



Seller and Order

DDL scripts

```
SET SEARCH_PATH TO "OMMS";



CREATE TABLE "Wishlist" (
        "trackId" bigint NOT NULL,
        "customerId" bigint NOT NULL,
        "amount" int NOT NULL,
        FOREIGN KEY ("trackId") REFERENCES "Track"("trackId") ON DELETE CASCADE ON UPDATE CASCADE,
        FOREIGN KEY ("customerId") REFERENCES "Customer"("customerId") ON DELETE CASCADE ON UPDATE CASCADE
);



CREATE TABLE "Cart" (
        "customerId" bigint NOT NULL,
        "trackId" bigint NOT NULL,
        "amount" int NOT NULL,
        FOREIGN KEY ("customerId") REFERENCES "Customer"("customerId") ON DELETE CASCADE ON UPDATE CASCADE,
        FOREIGN KEY ("trackId") REFERENCES "Track"("trackId") ON DELETE CASCADE ON UPDATE CASCADE
);



CREATE TABLE "Add Favorites" (
        "trackId" bigint NOT NULL,
        "albumId" bigint NOT  NULL,
        "customerId" bigint NOT NULL,
        FOREIGN KEY ("trackId") REFERENCES "Track"("trackId") ON DELETE CASCADE ON UPDATE CASCADE,
        FOREIGN KEY ("albumId") REFERENCES "Album"("albumId") ON DELETE CASCADE ON UPDATE CASCADE,
```

```
	FOREIGN KEY ("customerId") REFERENCES "Customer"("customerId") ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE "Listen" (
	"trackId" bigint NOT NULL,
	"customerId" bigint NOT NULL,
	FOREIGN KEY ("trackId") REFERENCES "Track"("trackId") ON DELETE CASCADE ON UPDATE CASCADE,
	FOREIGN KEY ("customerId") REFERENCES "Customer"("customerId") ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE "albumTrack" (
	"trackId" bigint NOT NULL,
	"albumId" bigint NOT NULL,
	FOREIGN KEY ("trackId") REFERENCES "Track"("trackId") ON DELETE CASCADE ON UPDATE CASCADE,
	FOREIGN KEY ("albumId") REFERENCES "Album"("albumId") ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE "sellerOrder" (
	"orderId" bigint NOT NULL,
	"artistId" bigint NOT NULL,
	FOREIGN KEY ("orderId") REFERENCES "Order"("orderId") ON DELETE CASCADE ON UPDATE CASCADE,
	FOREIGN KEY ("artistId") REFERENCES "Artist"("artistId") ON DELETE CASCADE ON UPDATE CASCADE
);
```

```
CREATE TABLE "artistTrack" (
        "artistId" bigint NOT NULL,
        "trackId" bigint NOT NULL,
        FOREIGN KEY ("artistId") REFERENCES "Artist"("artistId") ON DELETE CASCADE ON UPDATE CASCADE,
        FOREIGN KEY ("trackId") REFERENCES "Track"("trackId") ON DELETE CASCADE ON UPDATE CASCADE
);



CREATE TABLE "Album" (
        "albumId" bigint NOT NULL,
        "albumName" character varying(30) NOT NULL,
        "dateOfRelease" timestamp without time zone NOT NULL,
        "noOfSongs" int NOT NULL,
        "ratting" real DEFAULT NULL,
        PRIMARY KEY ("albumId")
);



CREATE TABLE "Artist" (
        "artistId" bigint NOT NULL,
        "artistName" character varying(30) NOT NULL,
        "sellerName" character varying(30) NOT NULL,
        "email" character varying(40) NOT NULL,
        "userName" character varying(30) NOT NULL,
        "password_" character varying(30) NOT NULL,
        "gender" character varying(6) NOT NULL,
        "ratting" real DEFAULT NULL,
        PRIMARY KEY ("artistId")
);
```

```
CREATE TABLE "Track" (
        "trackId" bigint NOT NULL,
        "genreId" bigint NOT NULL,
        "trackName" character varying(40) NOT NULL,
        "dateOfRelease" timestamp without time zone NOT NULL,
        "price" numeric DEFAULT NULL,
        "length" numeric NOT NULL,
        "language" character varying(30) NOT NULL,
        PRIMARY KEY ("trackId"),
        FOREIGN KEY ("genreId") REFERENCES "Genre"("genreId") ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE "Genre" (
        "genreId" bigint NOT NULL,
        "genre" character varying(10) NOT NULL,
        PRIMARY KEY ("genreId")
);

CREATE TABLE "Transaction History" (
        "transactionId" bigint NOT NULL,
        "orderId" bigint NOT NULL,
        "amount" int NOT NULL,
        "time" int NOT NULL,
        PRIMARY KEY ("transactionId"),
        FOREIGN KEY ("orderId") REFERENCES "Order"("orderId") ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE "Order" (
        "orderId" bigint NOT NULL,
        "customerId" bigint NOT NULL,
        "date" timestamp without time zone NOT NULL,
```

```
	"originalPrice" numeric NOT NULL,
	"discount" int DEFAULT NULL,
	"totalAmount" numeric,
	PRIMARY KEY ("orderId"),
	FOREIGN KEY ("customerId") REFERENCES "Customer"("customerId") ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE "Customer" (
	"customerId" bigint NOT NULL,
	"customerName" character varying(30),
	"userName" character varying(30) NOT NULL,
	"password_" character varying(30) NOT NULL,
	"email" character varying(40) NOT NULL,
	"dateOfBirth" date NOT NULL,
	"address" character varying(250) NOT NULL,
	PRIMARY KEY ("customerId")
);

CREATE TABLE "artistAlbum" (
	"artistId" bigint NOT NULL,
	"albumId" bigint NOT NULL,
	FOREIGN KEY ("artistId") REFERENCES "Artist"("artistId") ON DELETE CASCADE ON UPDATE CASCADE,
	FOREIGN KEY ("albumId") REFERENCES "Album"("albumId") ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE "cutomerTrackOrder" (
	"trackId" bigint NOT NULL,
	"customerId" bigint NOT NULL,
	"orderId" bigint NOT NULL,
	FOREIGN KEY ("trackId") REFERENCES "Track"("trackId") ON DELETE CASCADE ON UPDATE CASCADE,
```

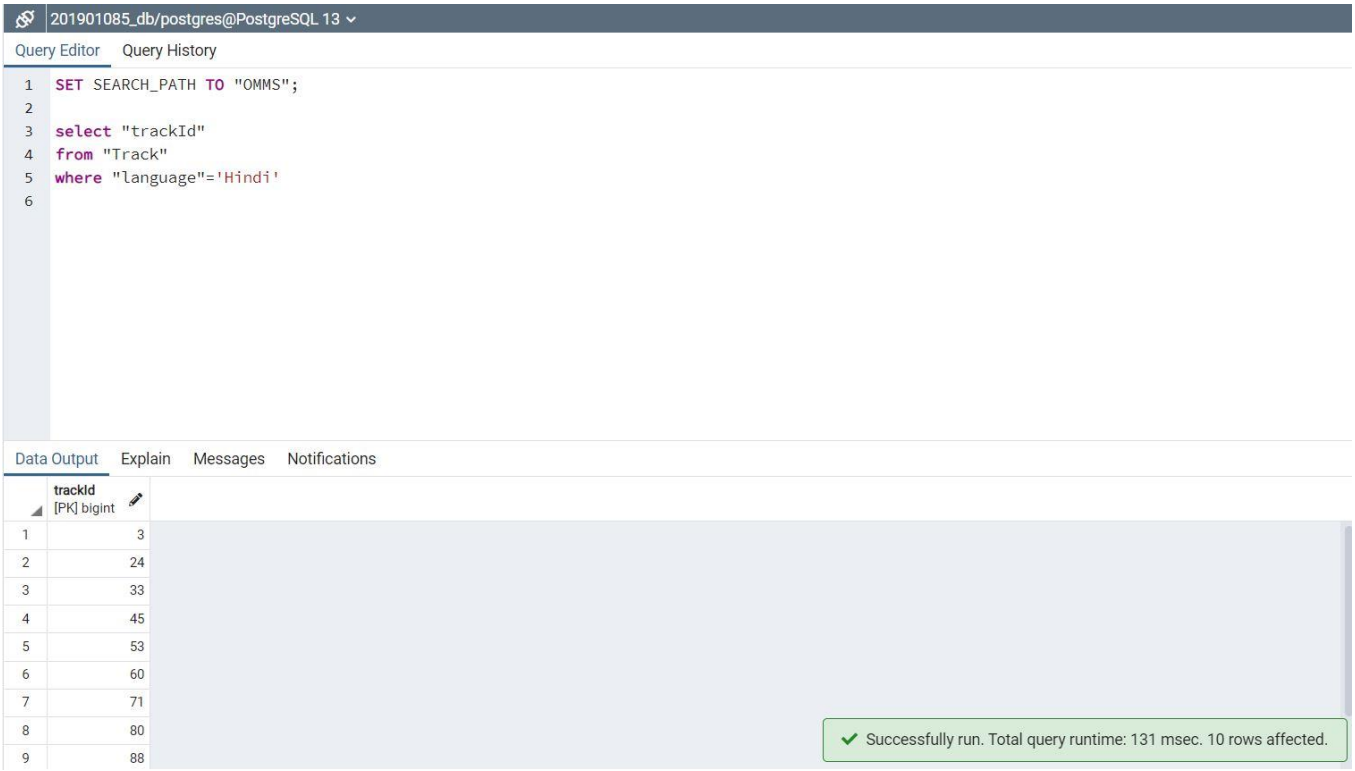
```
    FOREIGN KEY ("customerId") REFERENCES "Customer"("customerId") ON DELETE CASCADE ON UPDATE CASCADE,

    FOREIGN KEY ("orderId") REFERENCES "Order"("orderId") ON DELETE CASCADE ON UPDATE CASCADE

);
```

SQL Queries

1. **List the trackId of Track which is Hindi language.**

```
select "trackId"
from "Track"
where "language"='Hindi'
```

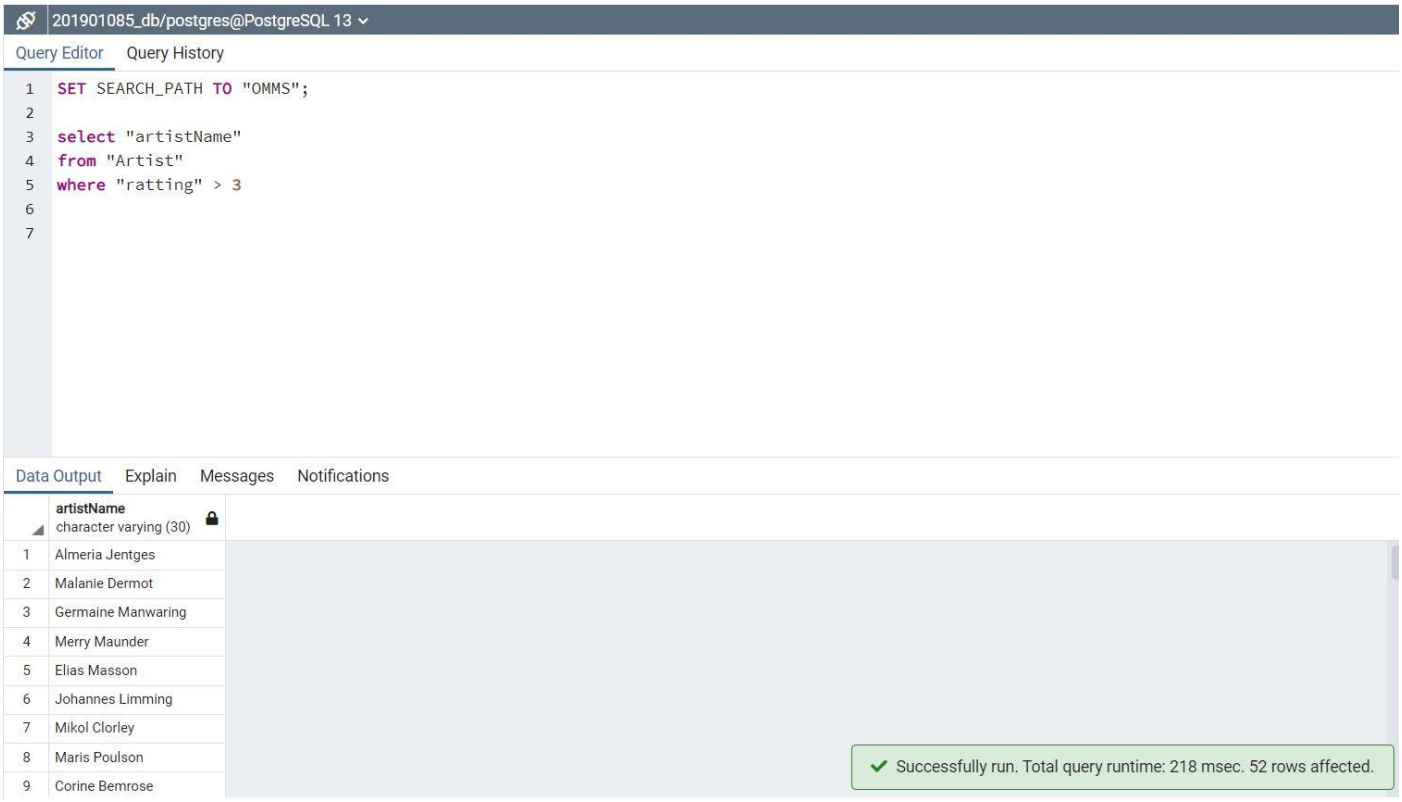
Result Tupples: 10 rows



2. **List the Artist Name which has ratting greater than 3.**

```
select "artistName"
from "Artist"
where "ratting" > 3
```

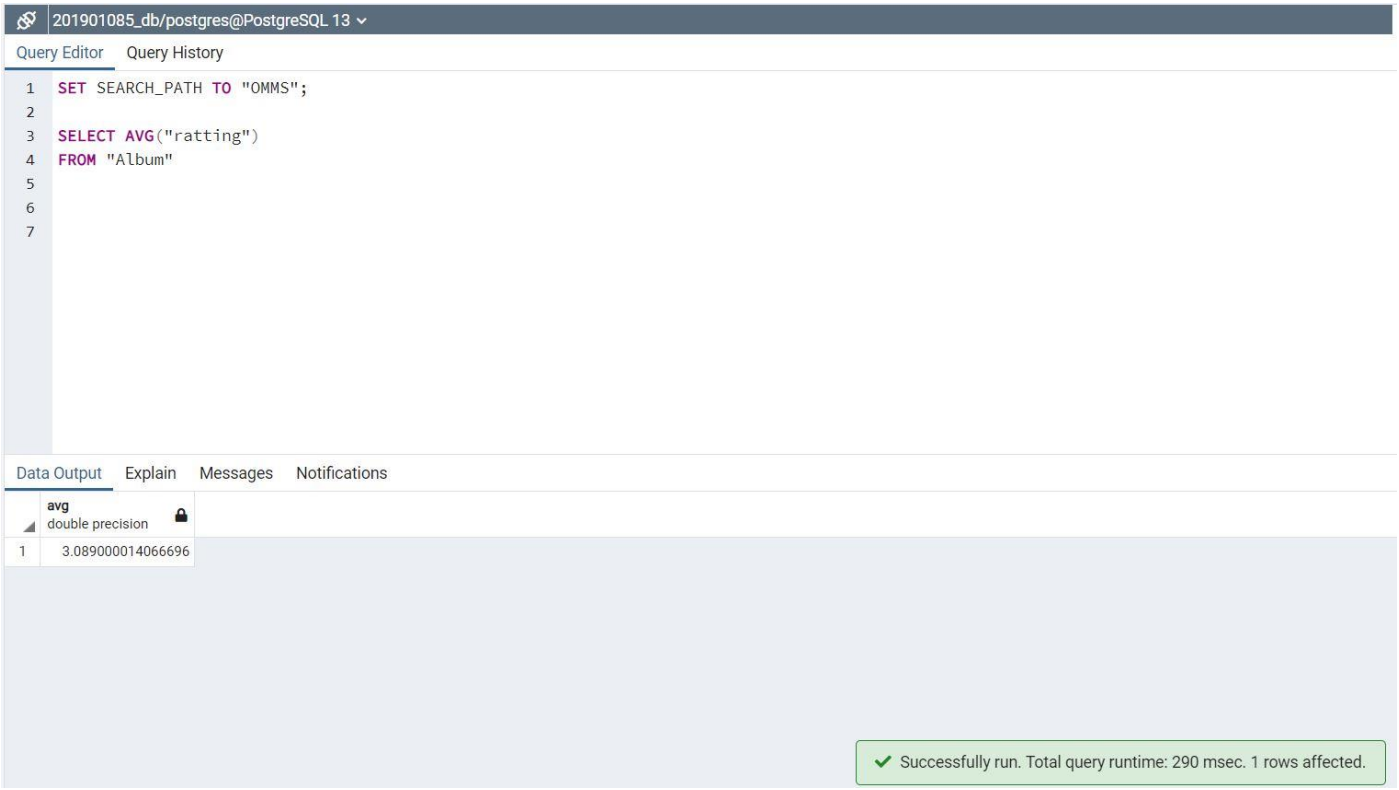
Result Tupples: 52 rows



3. **Find the average ratting in Album**.

```
SELECT AVG("ratting")
FROM "Album"
```

Result Tupples: 1 row



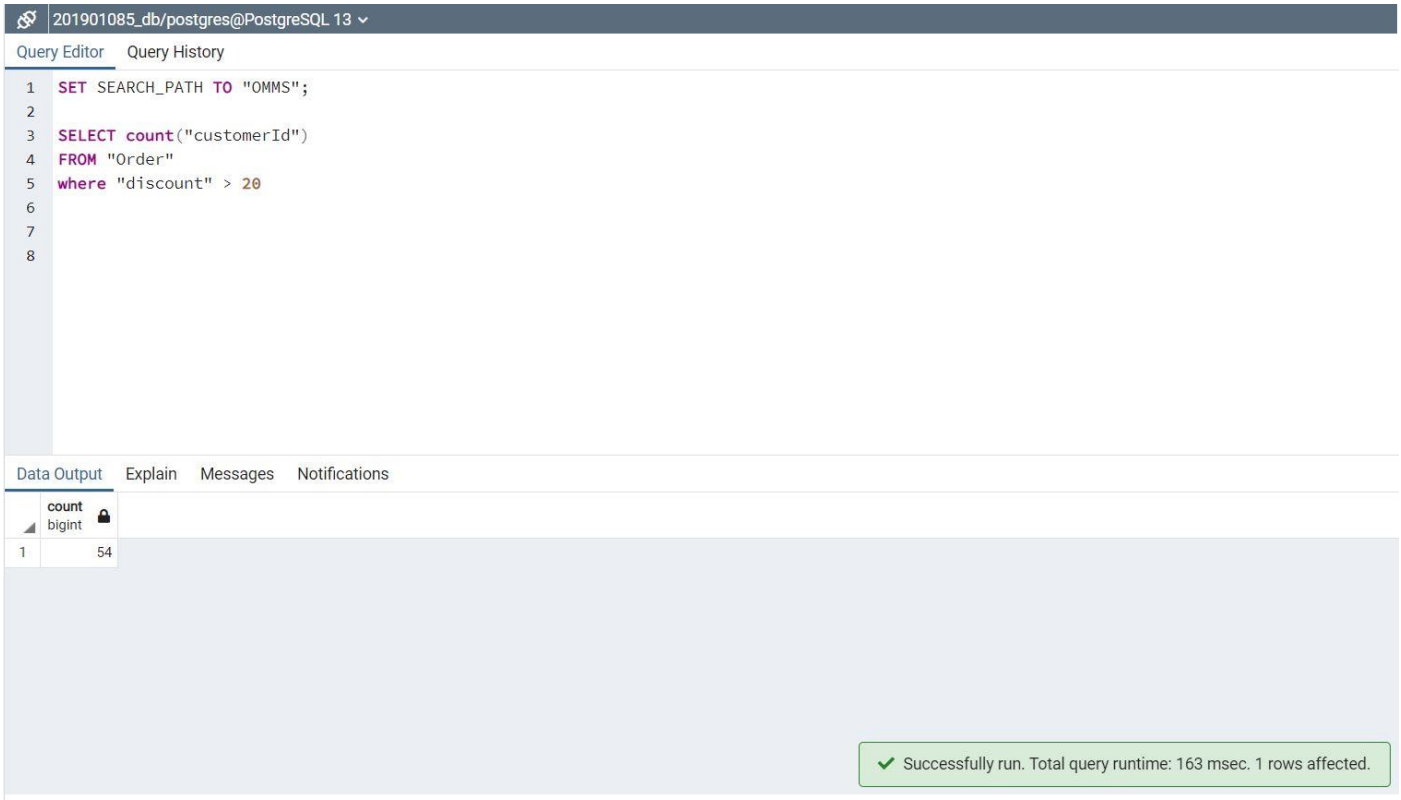
4. **find the total number of customers which have discount greater than 20**.

SELECT count("customerId")

FROM "Order"

where "discount" > 20

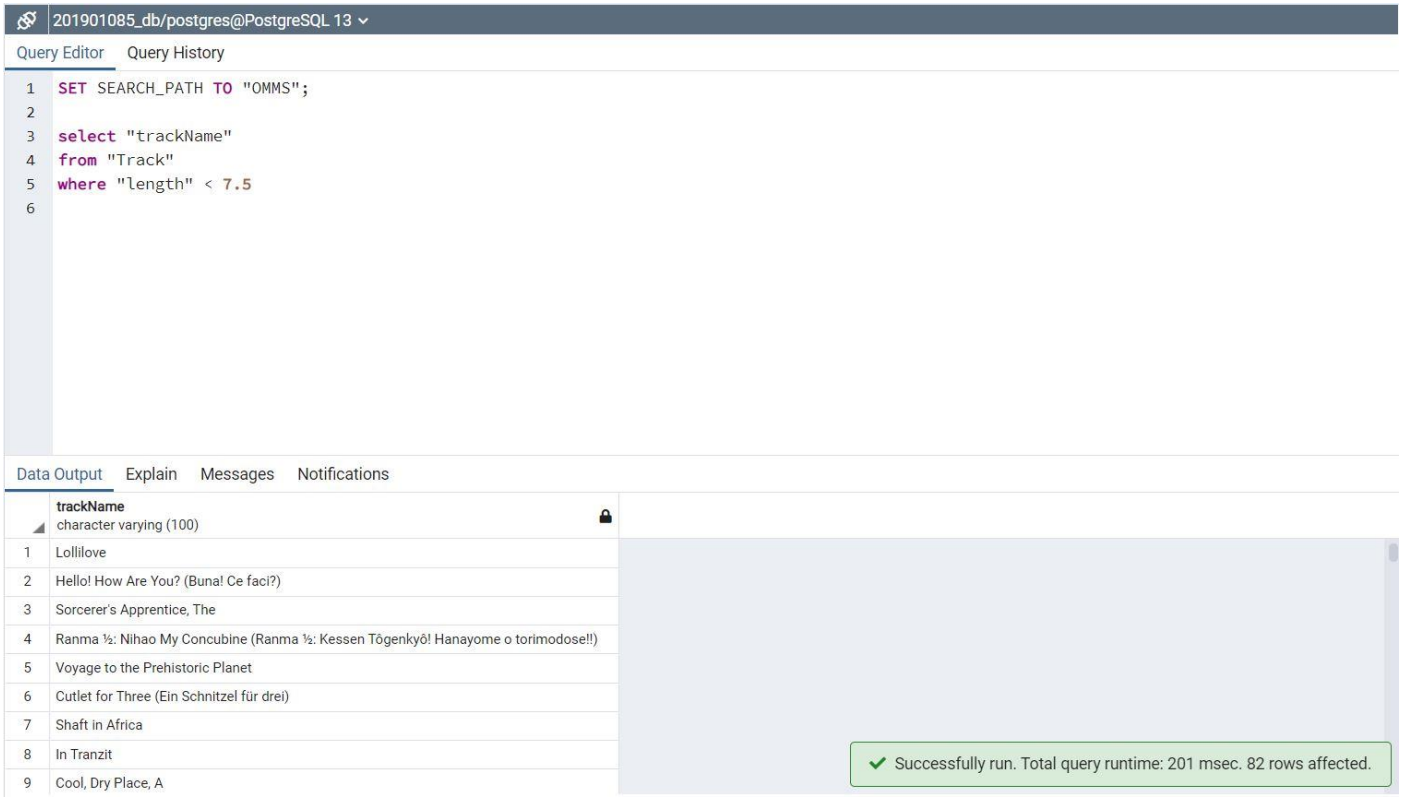
Result Tupples: 1 row



5. **find the Track name which has length less than 7.5**.

```
select "trackName"
from "Track"
where "length" < 7.5
```

Result Tupples: 82 rows



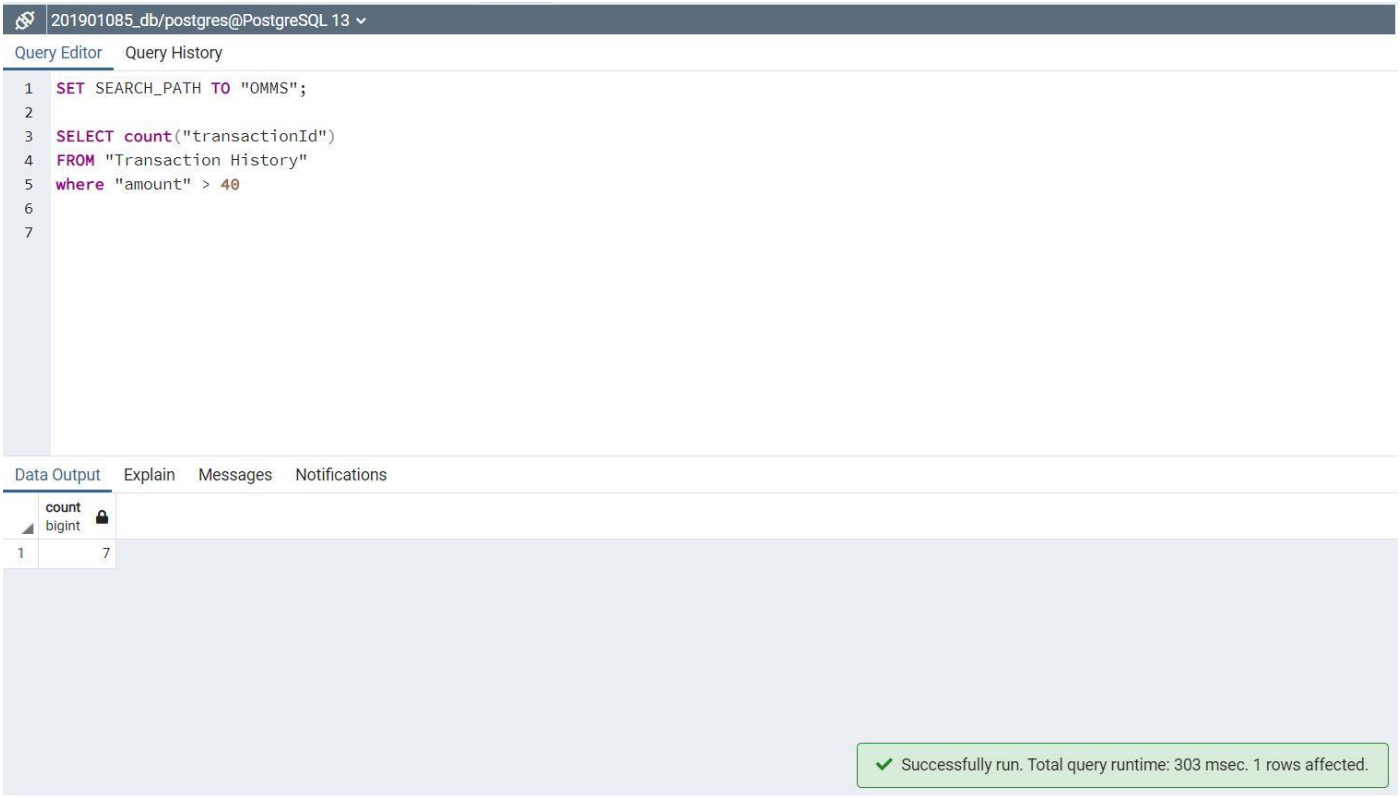
6. **count the transaction ID which has amount greater than 40**.

SELECT count("transactionId")

FROM "Transaction History"

where "amount" > 40

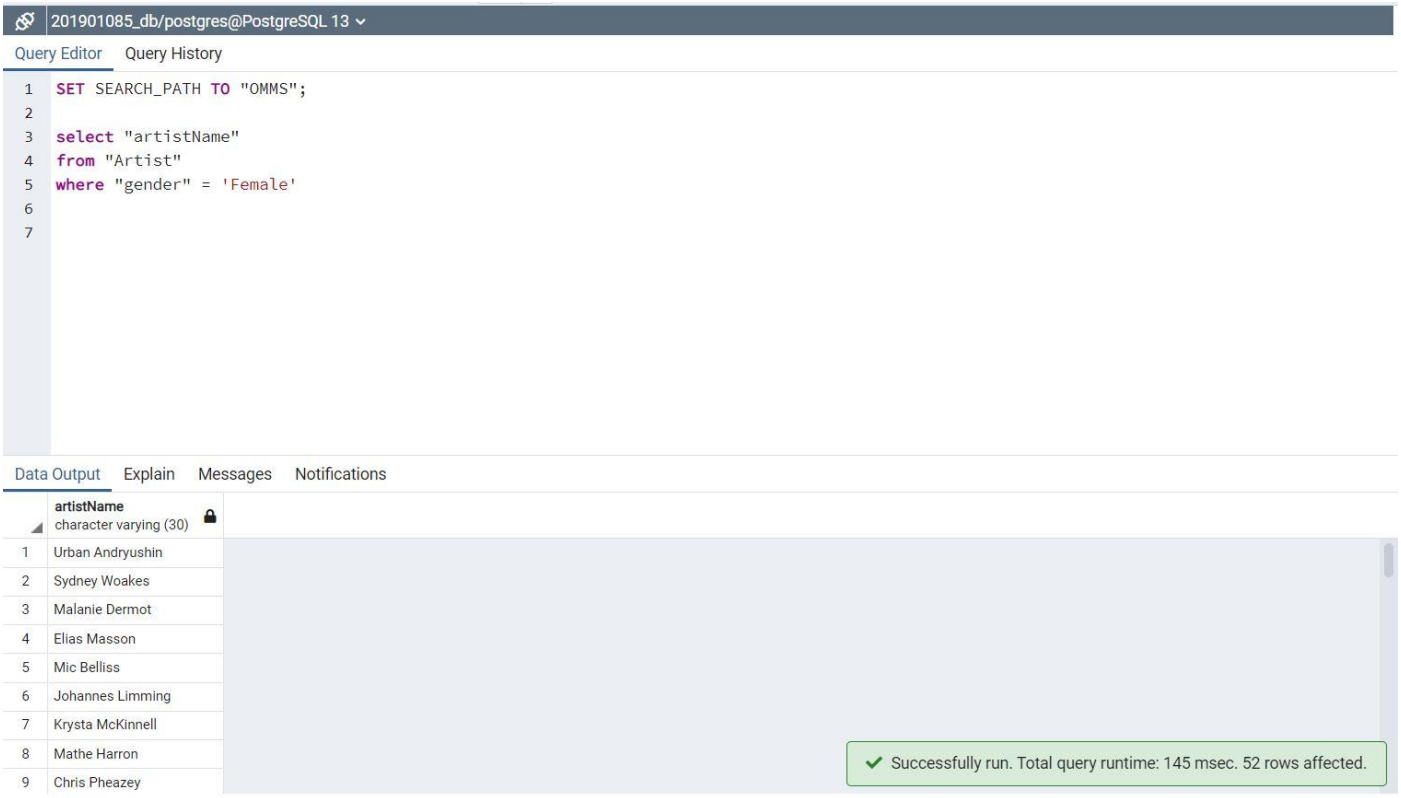
Result Tupples: 1 row



7. **Write a sql query to find the Artist Name who has female Artist**.

```
select "artistName"
from "Artist"
where "gender" = 'Female'
```

Result Tupples: 52 rows

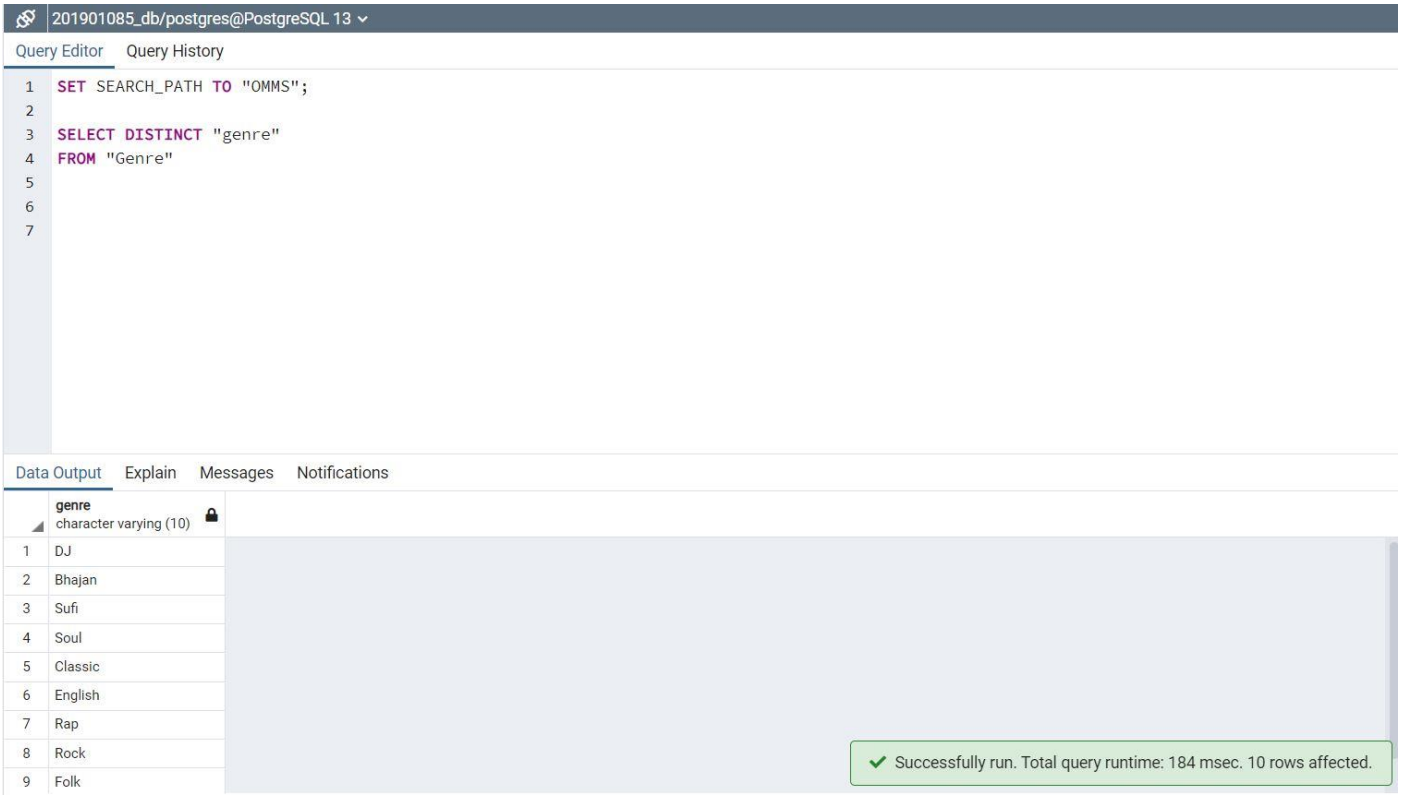


8. **find the different genre**.

SELECT DISTINCT "genre"

FROM "Genre"

Result Tupples: 10 rows

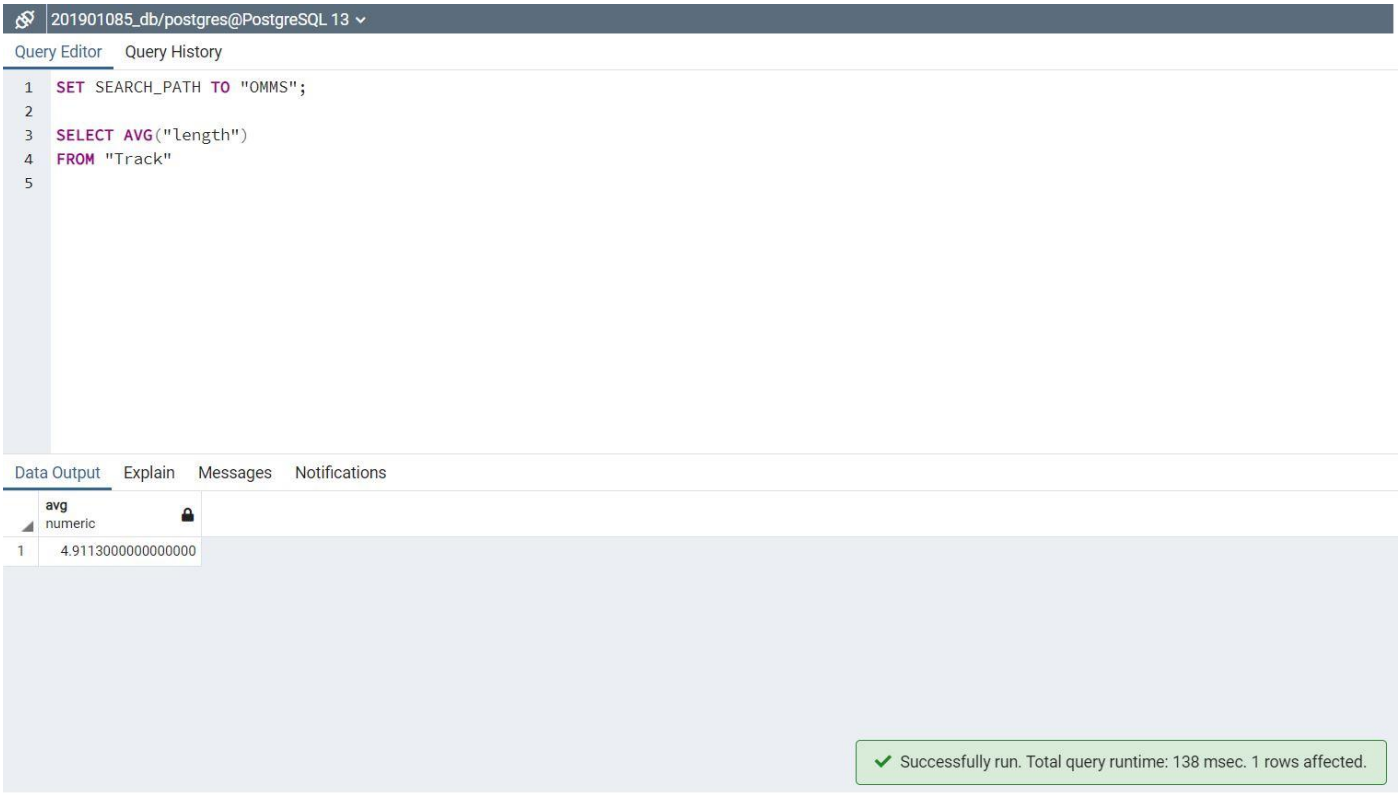


9. **find the average of track length**.

SELECT AVG("length")

FROM "Track"

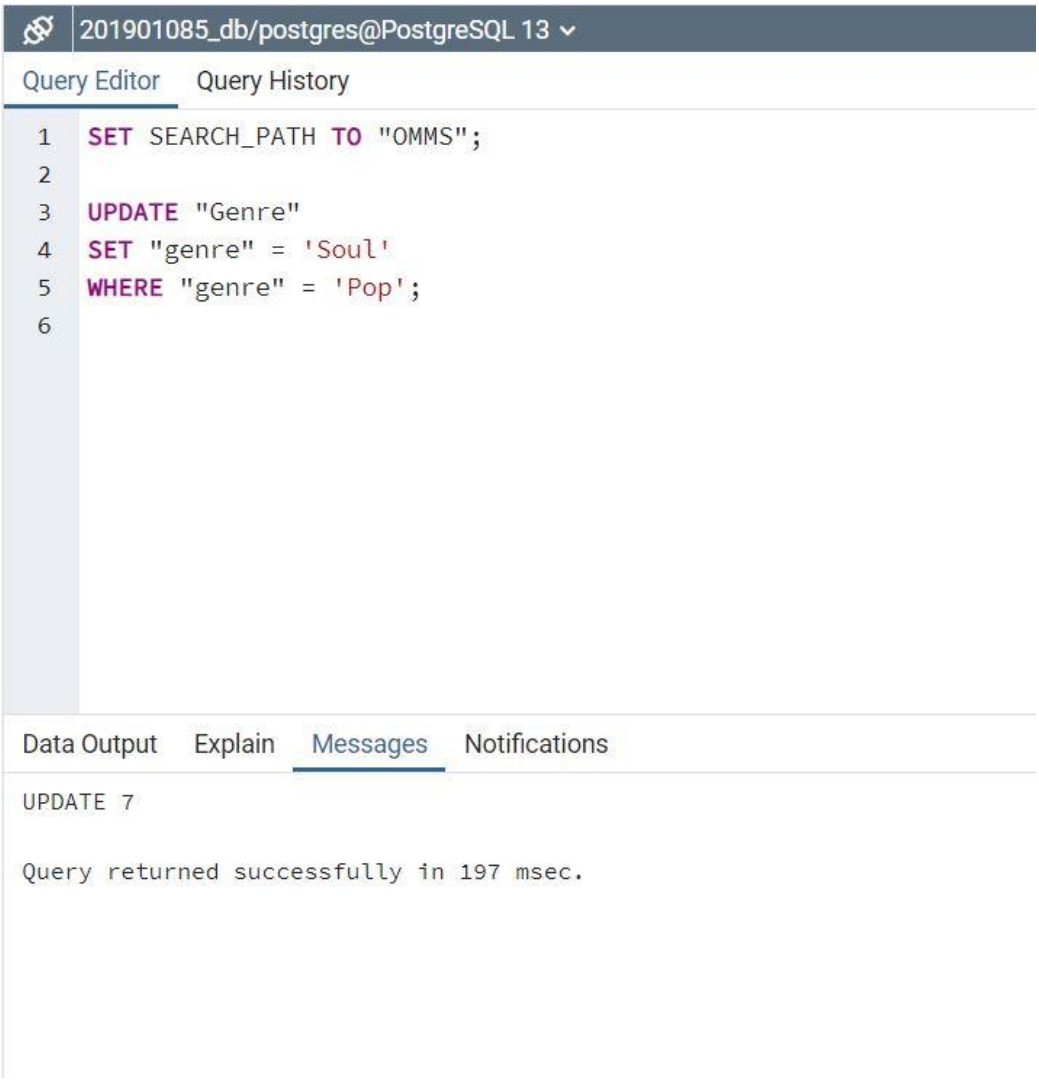
Result Tupples: 1 row



10. **Update the genre to “Soul” for all records where name of genre is written wrongly as “POP” in Genre table.**

```
UPDATE "Genre"
SET "genre" = 'Soul'
WHERE "genre" = 'POP';
```

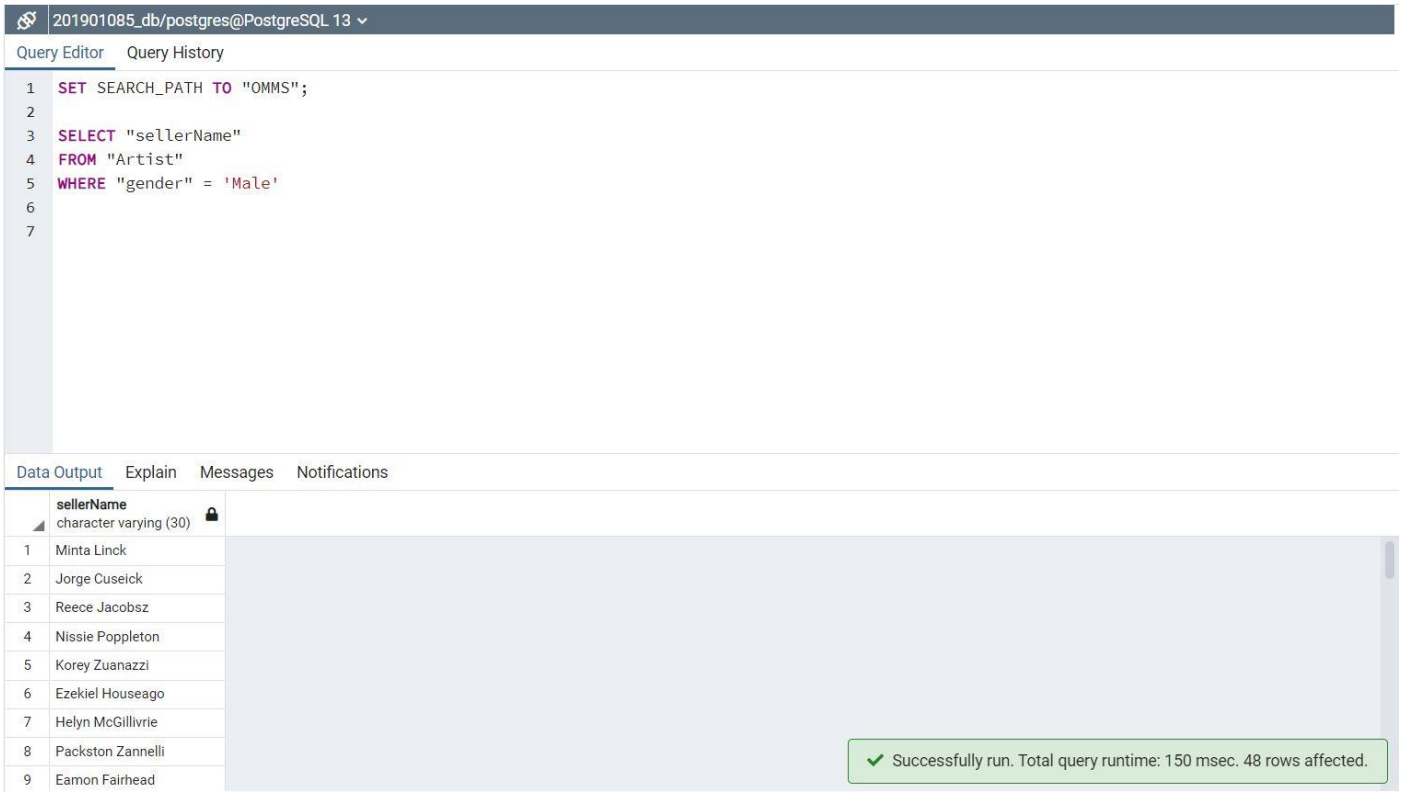
Result Tupples:



11. **Names of sellers whose gender is male**.

```
SET SEARCH_PATH TO "OMMS";

SELECT "sellerName"
FROM "Artist"
WHERE "gender" = 'Male'
```

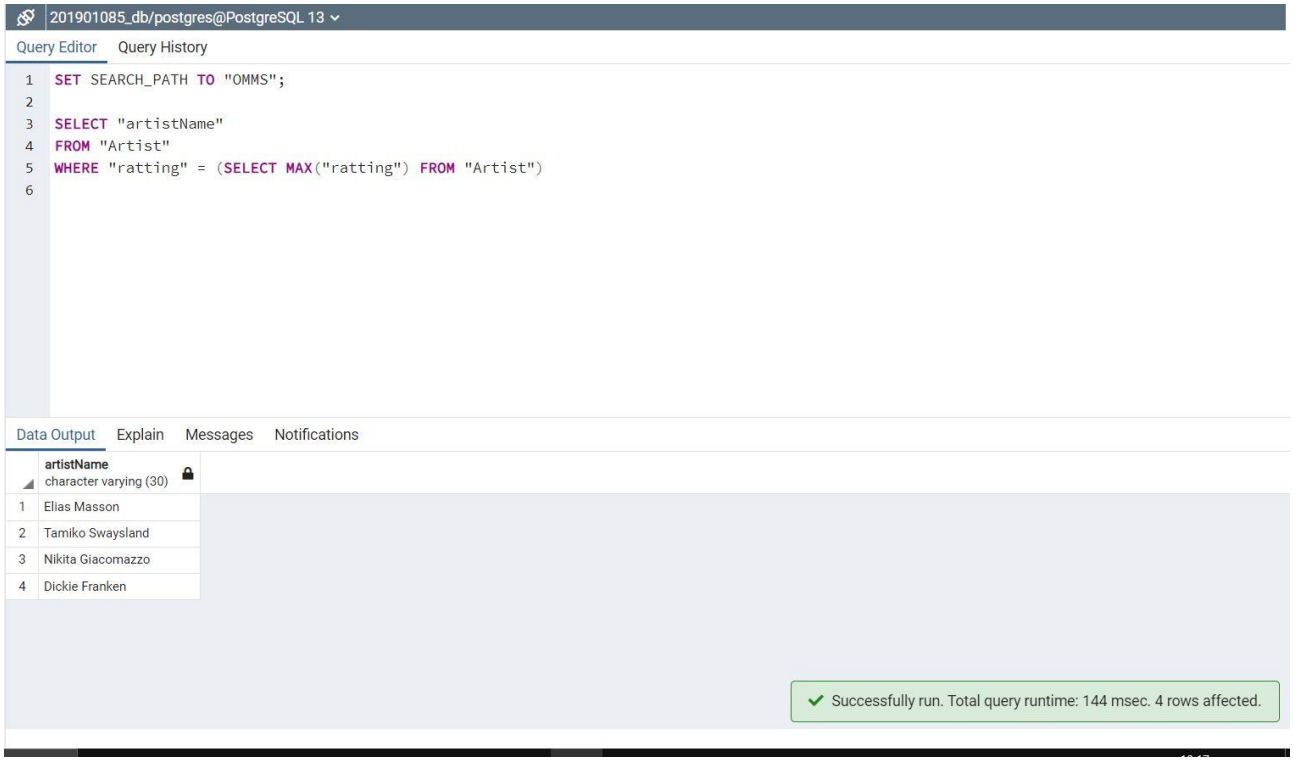
Result Tupples: 48 rows



12. **Names of Artists whose ratting is highest**.

```
SET SEARCH_PATH TO "OMMS";

SELECT "artistName"
FROM "Artist"
WHERE "ratting" = (SELECT MAX("ratting") FROM "Artist")
```

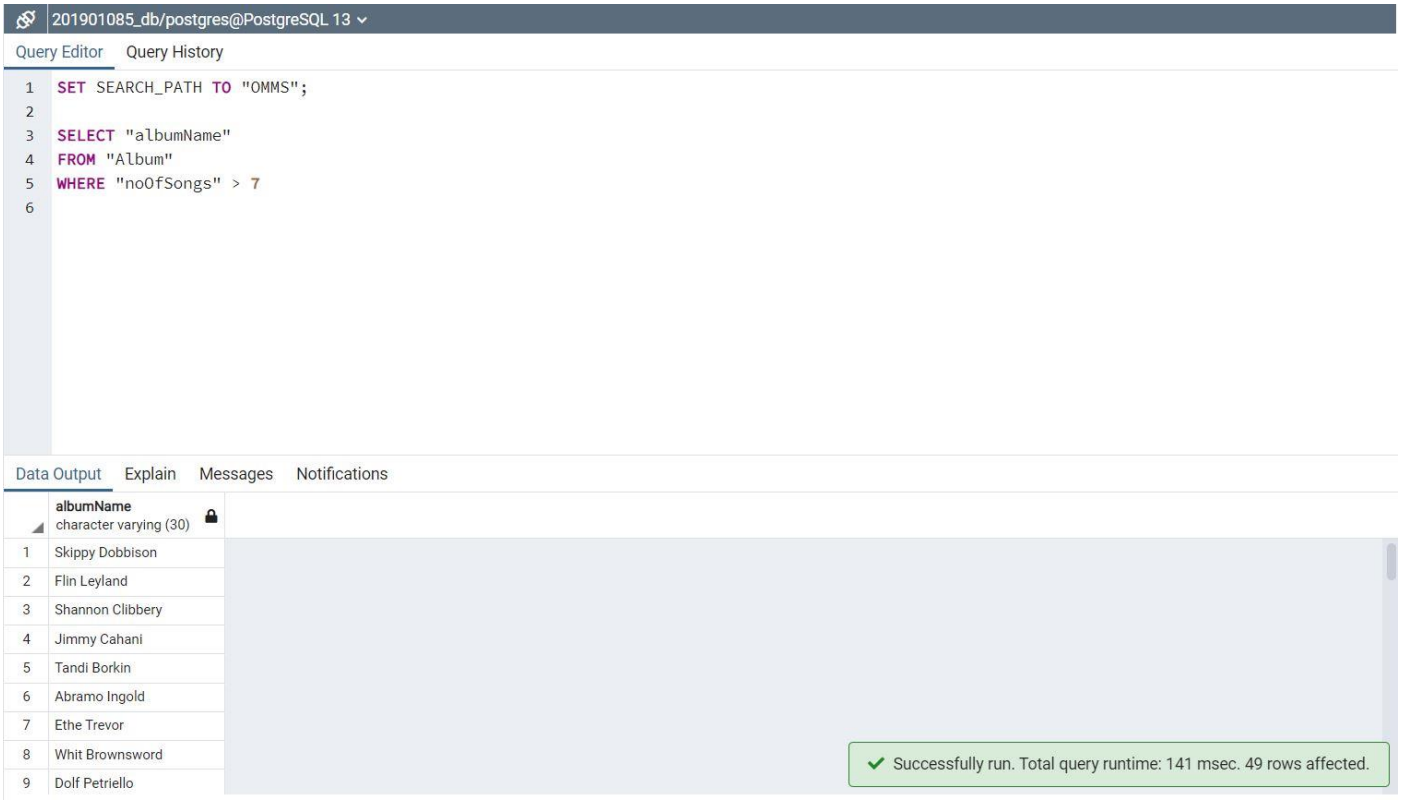
Result Tupples: 4 rows



13. **find album names that contains more than 7 tracks**.

```
SET SEARCH_PATH TO "OMMS";

SELECT "albumName"
FROM "Album"
WHERE "noOfSongs" > 7
```

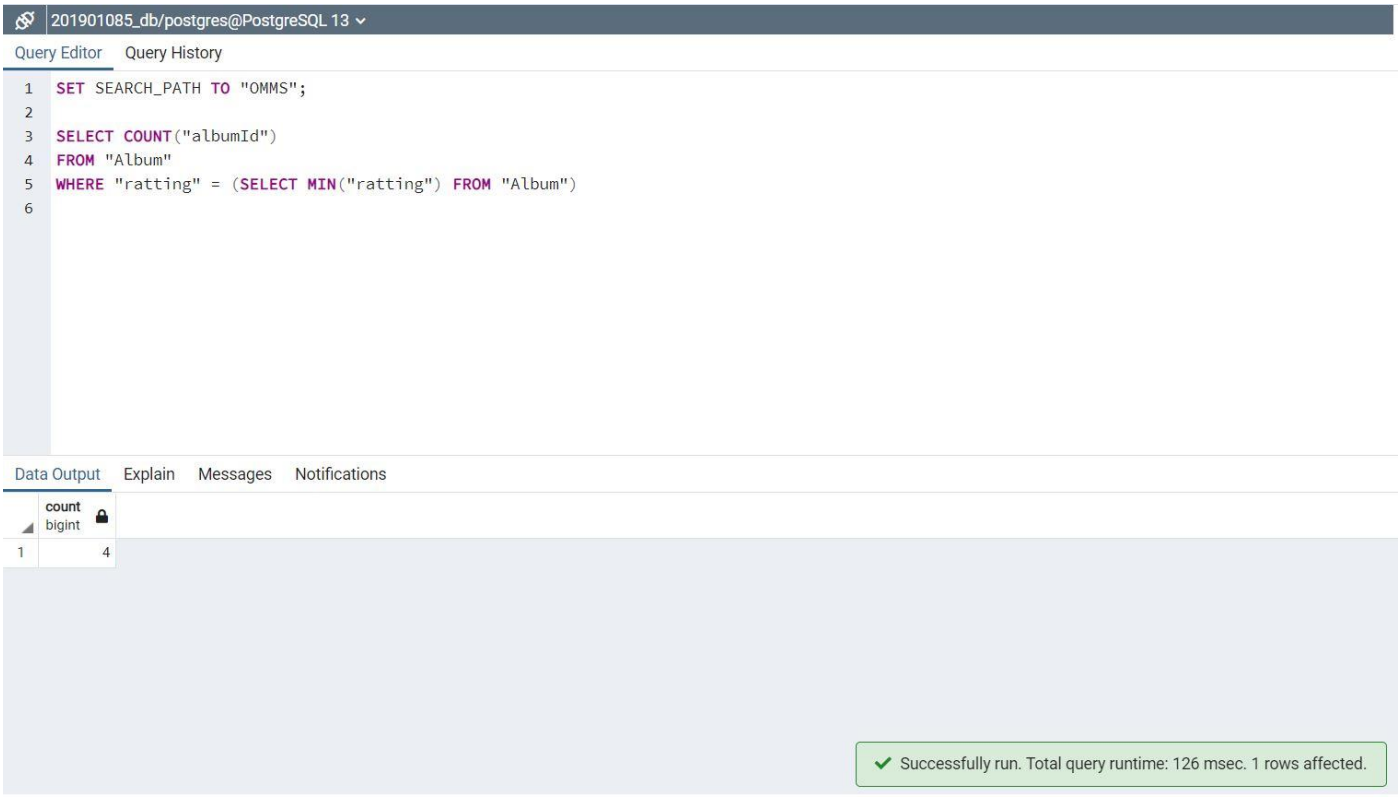
Result Tupples: 49 rows



14. **count albums that have min rating**.

```
SET SEARCH_PATH TO "OMMS";

SELECT COUNT("albumId")
FROM "Album"
WHERE "ratting" = (SELECT MIN("ratting") FROM "Album")
```

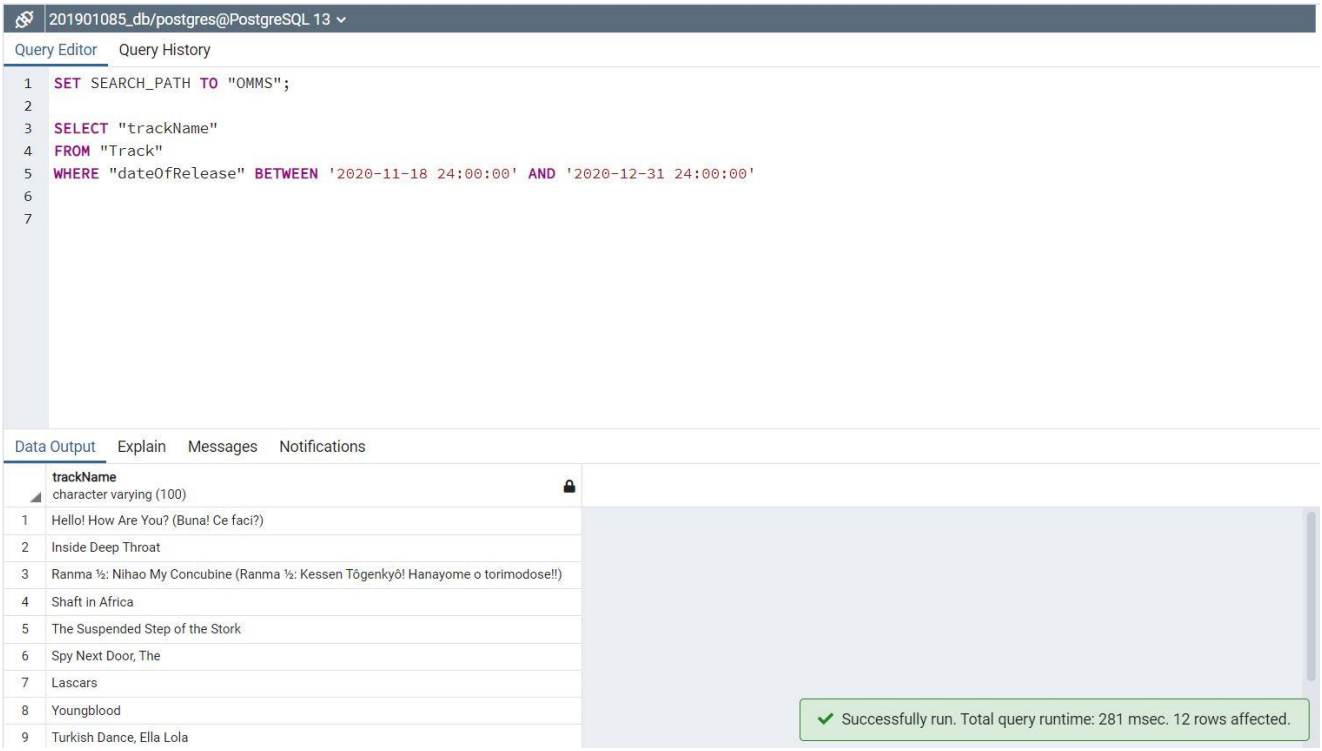
Result Tupples: 1 row



15. **Tracks that have released in 2020**.

```
SET SEARCH_PATH TO "OMMS";

SELECT "trackName"
FROM "Track"
WHERE "dateOfRelease" BETWEEN '2020-11-18 24:00:00' AND '2020-12-31 24:00:00'
```

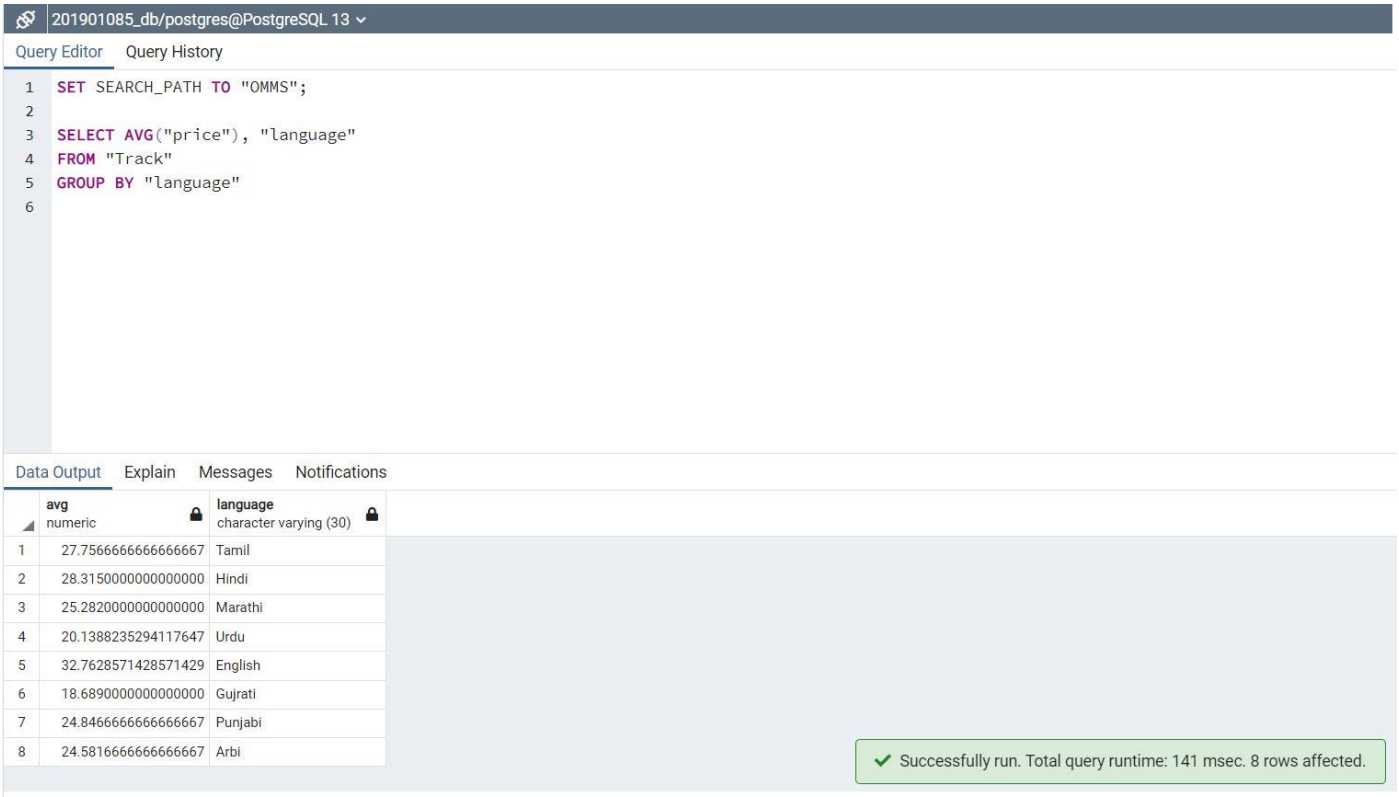
Result Tupples: 12 row



16. **Find Average Price of tracks for every language**.

```
SET SEARCH_PATH TO "OMMS";

SELECT AVG("price"), "language"
FROM "Track"
GROUP BY "language"
```

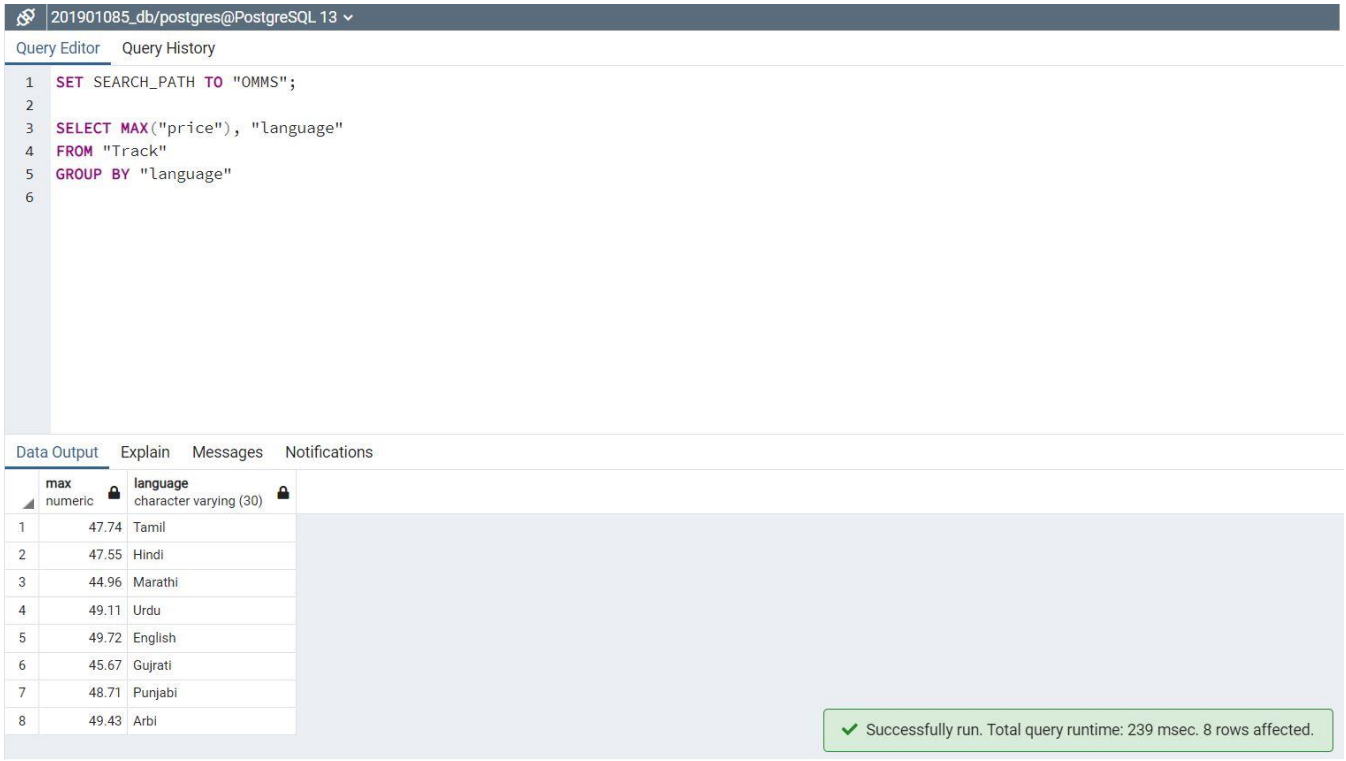
Result Tupples: 8 rows



17. **Find MAX price of tracks for every language**.

```
SET SEARCH_PATH TO "OMMS";

SELECT MAX("price"), "language"
FROM "Track"
GROUP BY "language"
```

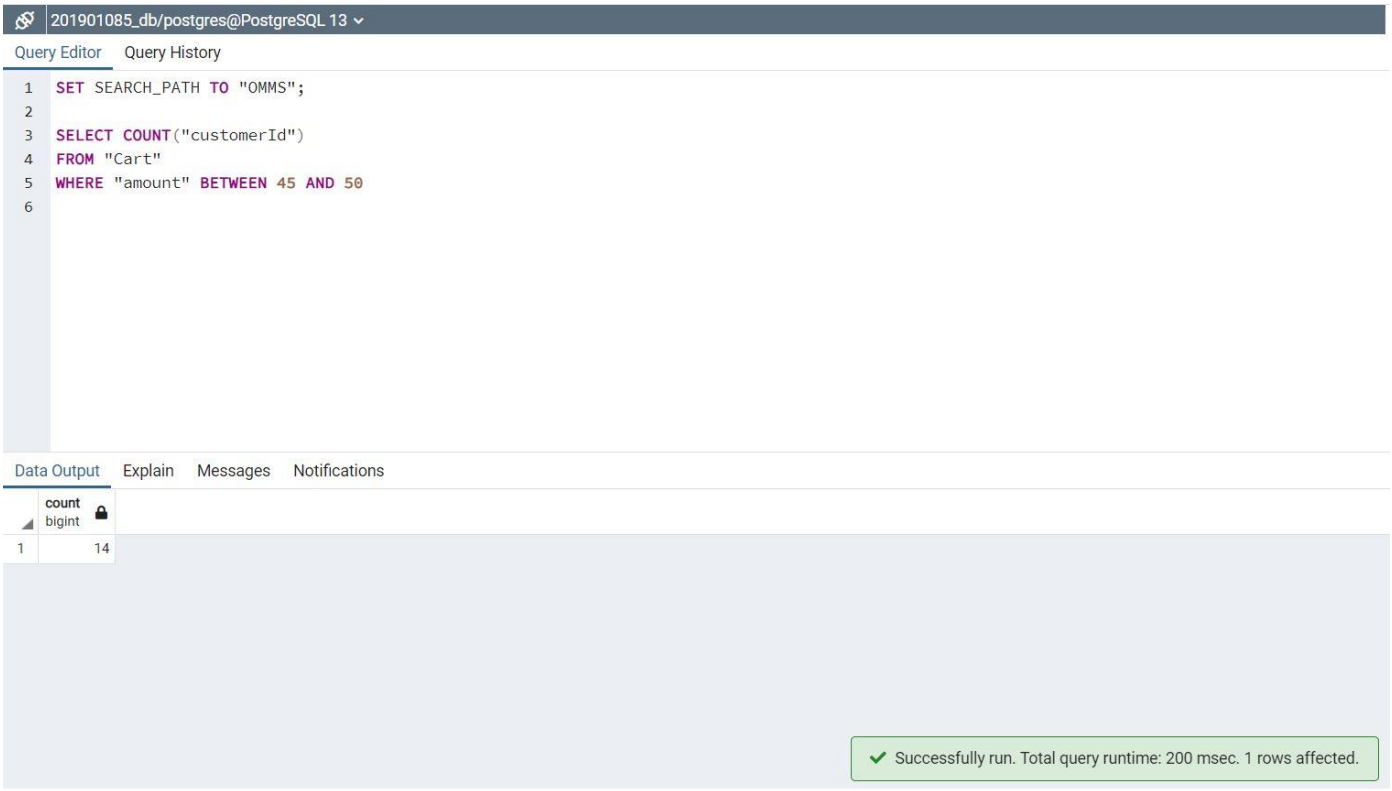
Result Tupples: 8 rows



18. **count Customers whose cart has products with amount between 45 $ to 50 $.**

```
SET SEARCH_PATH TO "OMMS";

SELECT COUNT("customerId")
FROM "Cart"
WHERE "amount" BETWEEN 45 AND 50
```

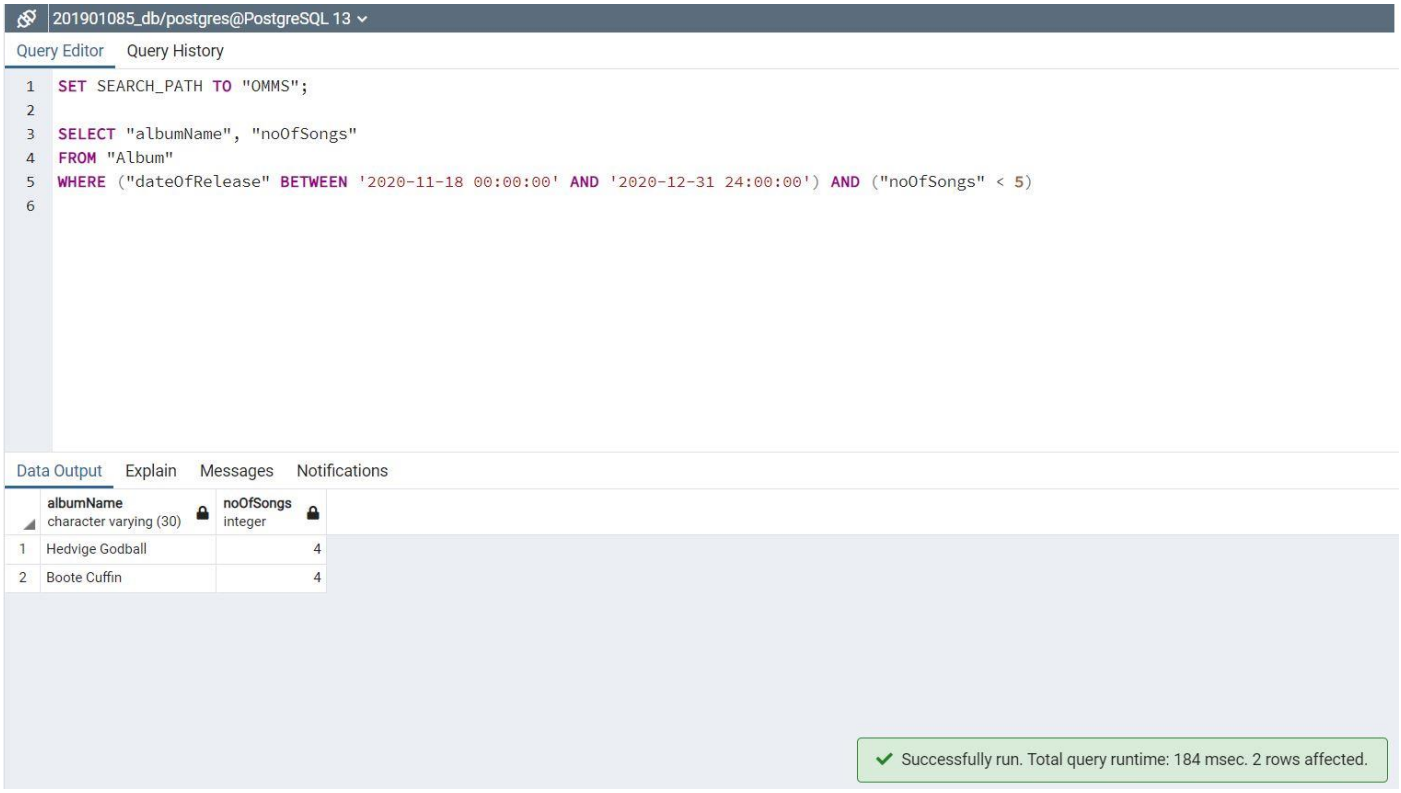
Result Tupples: 1 row



19. **albums that have tracks less than 5 and Released in 2020**.

```
SET SEARCH_PATH TO "OMMS";

SELECT "albumName", "noOfSongs"
FROM "Album"
WHERE ("dateOfRelease" BETWEEN '2020-11-18 00:00:00' AND '2020-12-31 24:00:00') AND ("noOfSongs" < 5)
```

Result Tupples: 2 rows

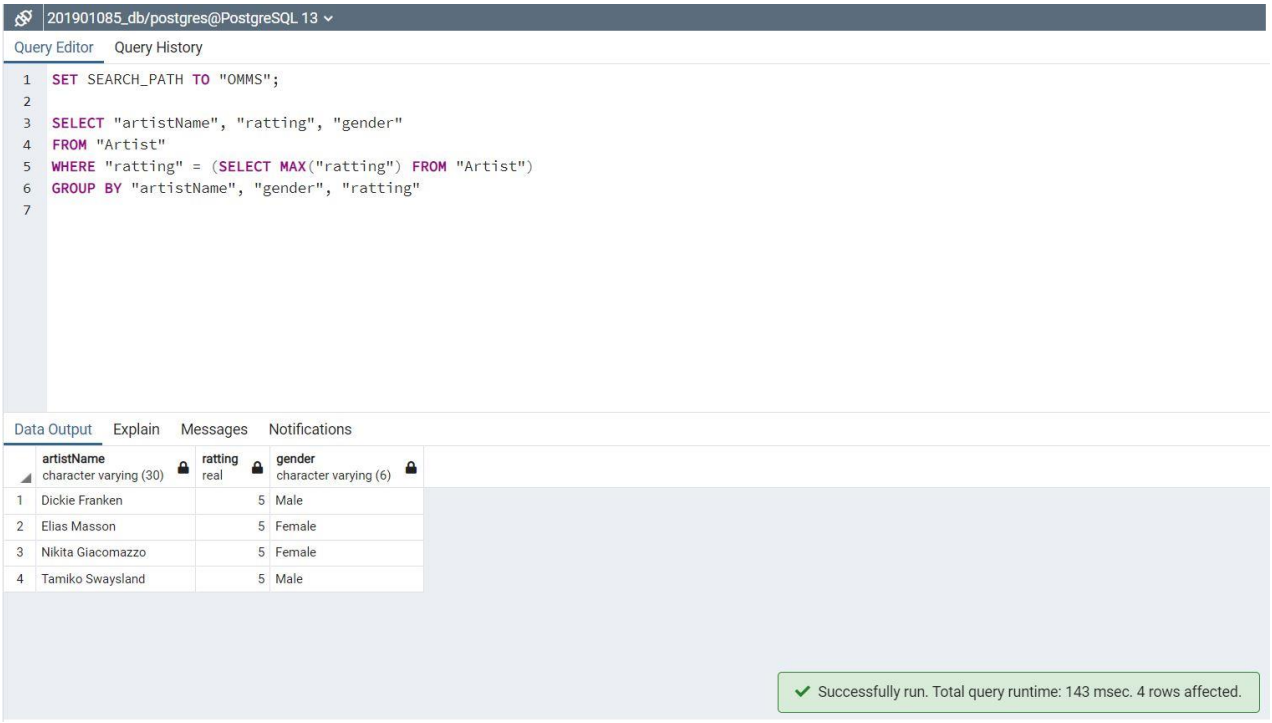


20.**Find Male and Female Artist Names with highest rating**.

SET SEARCH_PATH TO "OMMS";

SELECT "artistName", "ratting", "gender"
FROM "Artist"
WHERE "ratting" = (SELECT MAX("ratting") FROM "Artist")
GROUP BY "artistName", "gender", "ratting"

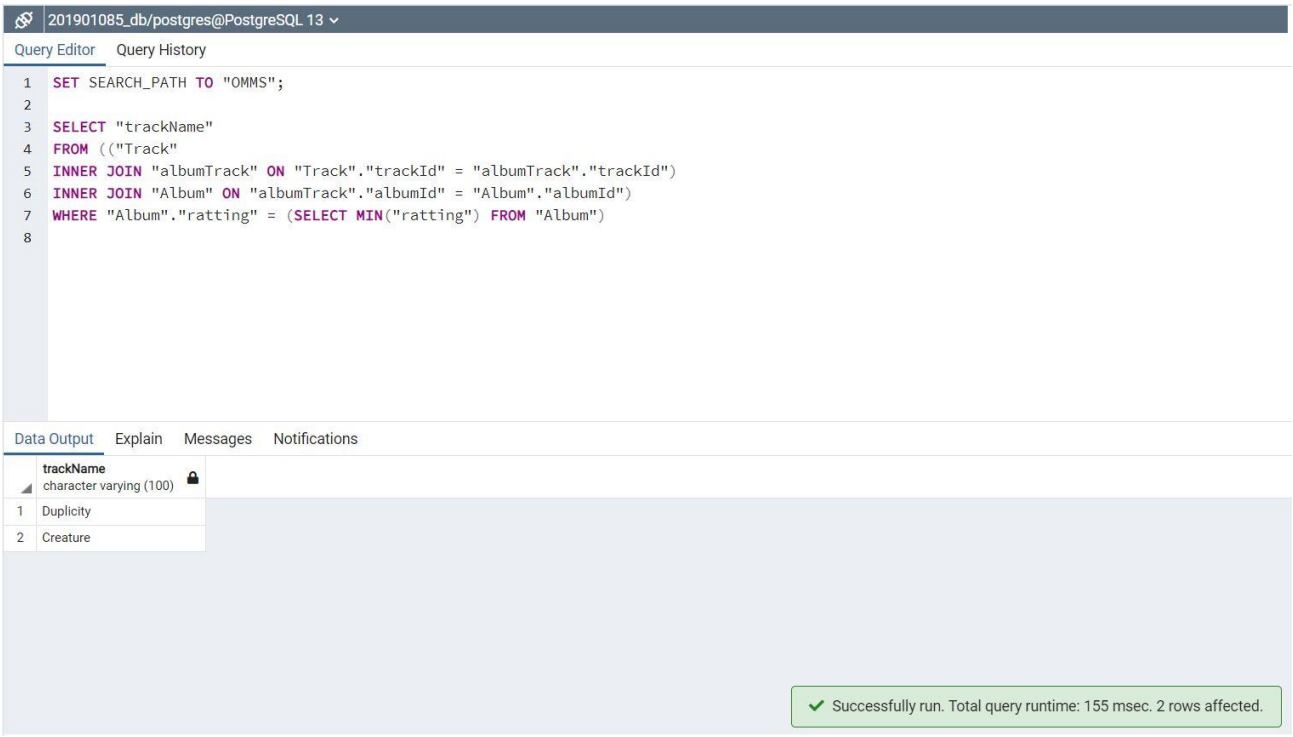
Result Tupples: 4 rows



21. **find Track names which are inside Album that has minimum ratting**.

```
SET SEARCH_PATH TO "OMMS";

SELECT "trackName"
FROM (("Track"
INNER JOIN "albumTrack" ON "Track"."trackId" = "albumTrack"."trackId")
INNER JOIN "Album" ON "albumTrack"."albumId" = "Album"."albumId")

WHERE "Album"."ratting" = (SELECT MIN("ratting") FROM "Album")
```

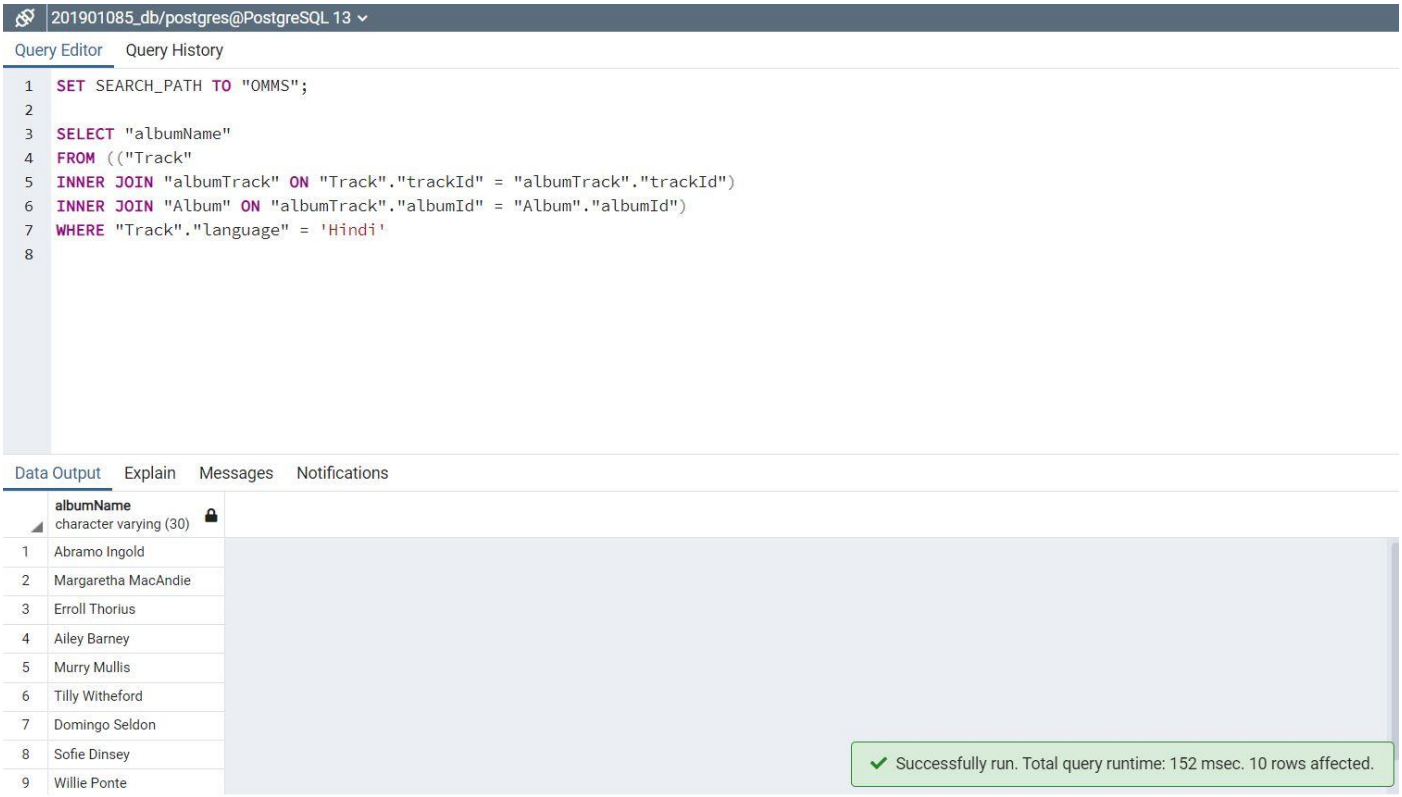
Result Tupples:



22. **Find Album names which has Tracks in Hindi language**.

```
SET SEARCH_PATH TO "OMMS";

SELECT "albumName"
FROM (("Track"
INNER JOIN "albumTrack" ON "Track"."trackId" = "albumTrack"."trackId")
INNER JOIN "Album" ON "albumTrack"."albumId" = "Album"."albumId")
WHERE "Track"."language" = 'Hindi'
```

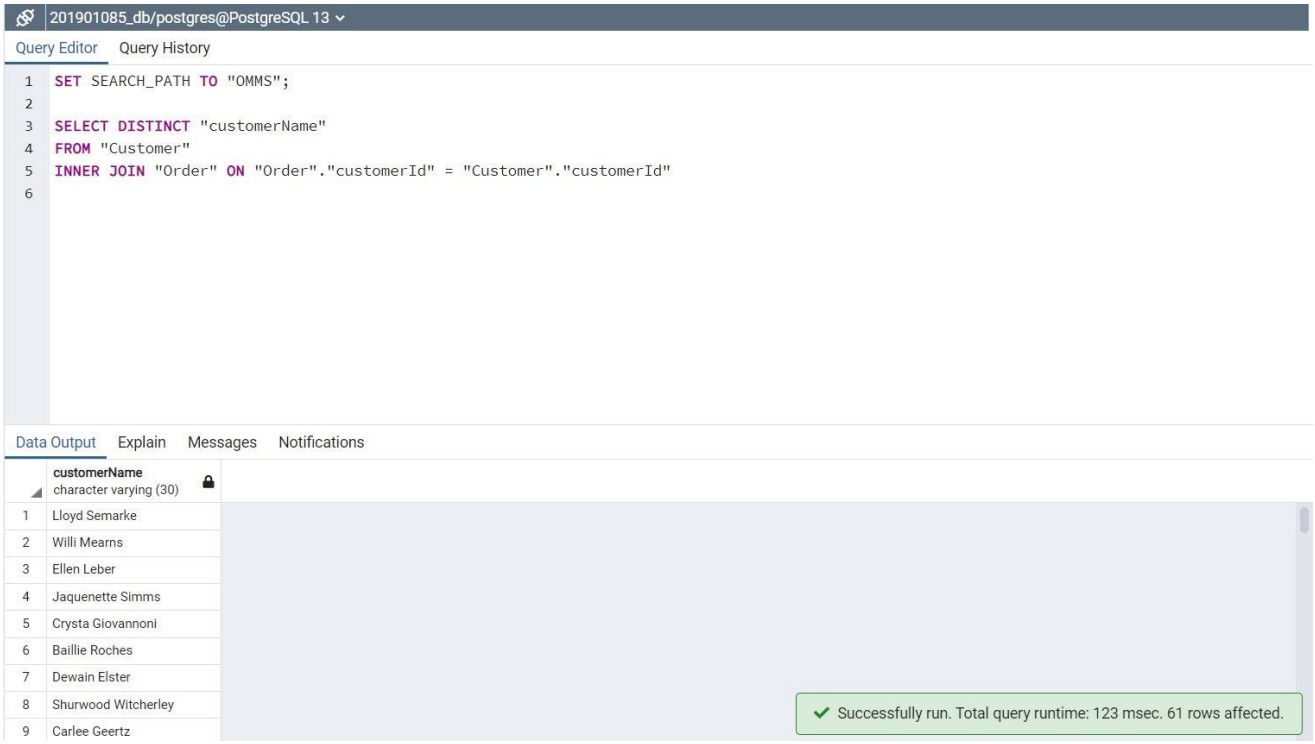
Result Tupples: 10 rows



23. **Customer names who purchased Tracks**.

```
SET SEARCH_PATH TO "OMMS";

SELECT DISTINCT "customerName"
FROM "Customer"
INNER JOIN "Order" ON "Order"."customerId" = "Customer"."customerId"
```

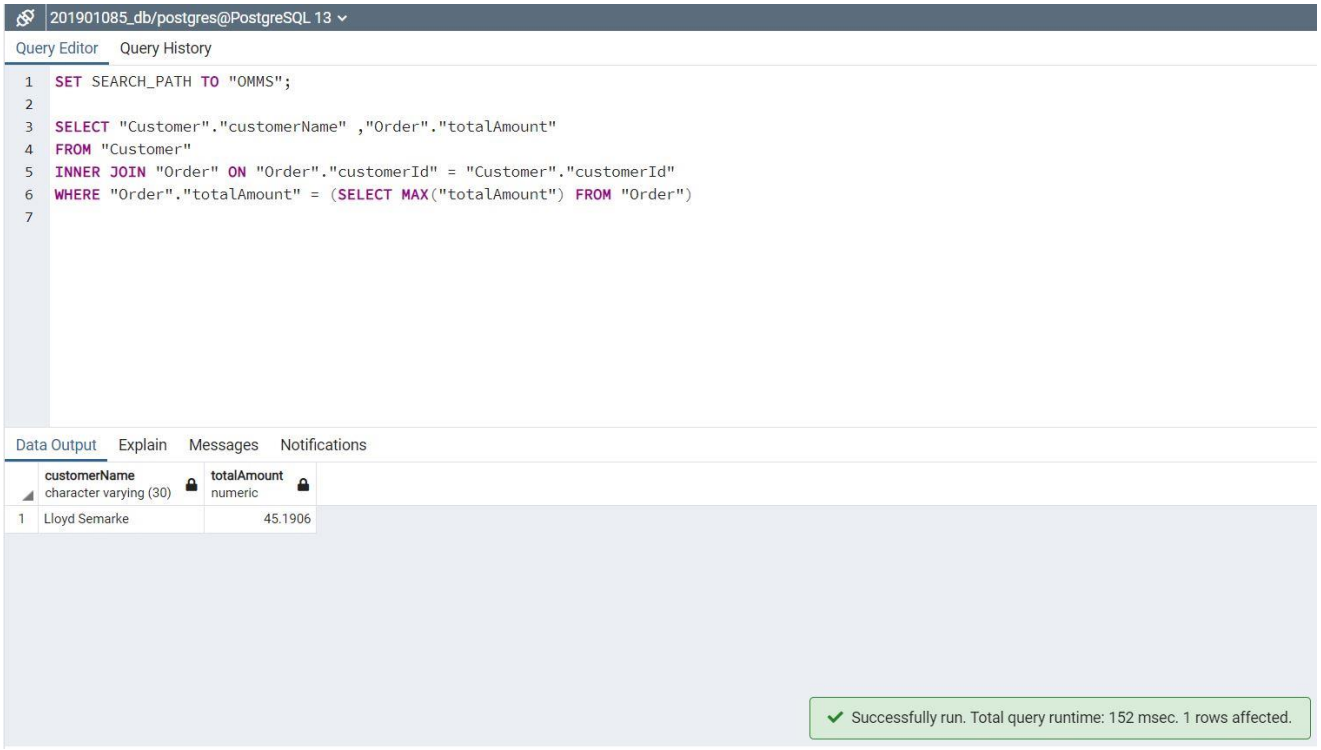
Result Tupples: 61 rows



24. **Customer name who purchased tracks with highest amount**.

```
SET SEARCH_PATH TO "OMMS";

SELECT "Customer"."customerName" ,"Order"."totalAmount"
FROM "Customer"
INNER JOIN "Order" ON "Order"."customerId" = "Customer"."customerId"
WHERE "Order"."totalAmount" = (SELECT MAX("totalAmount") FROM "Order")
```

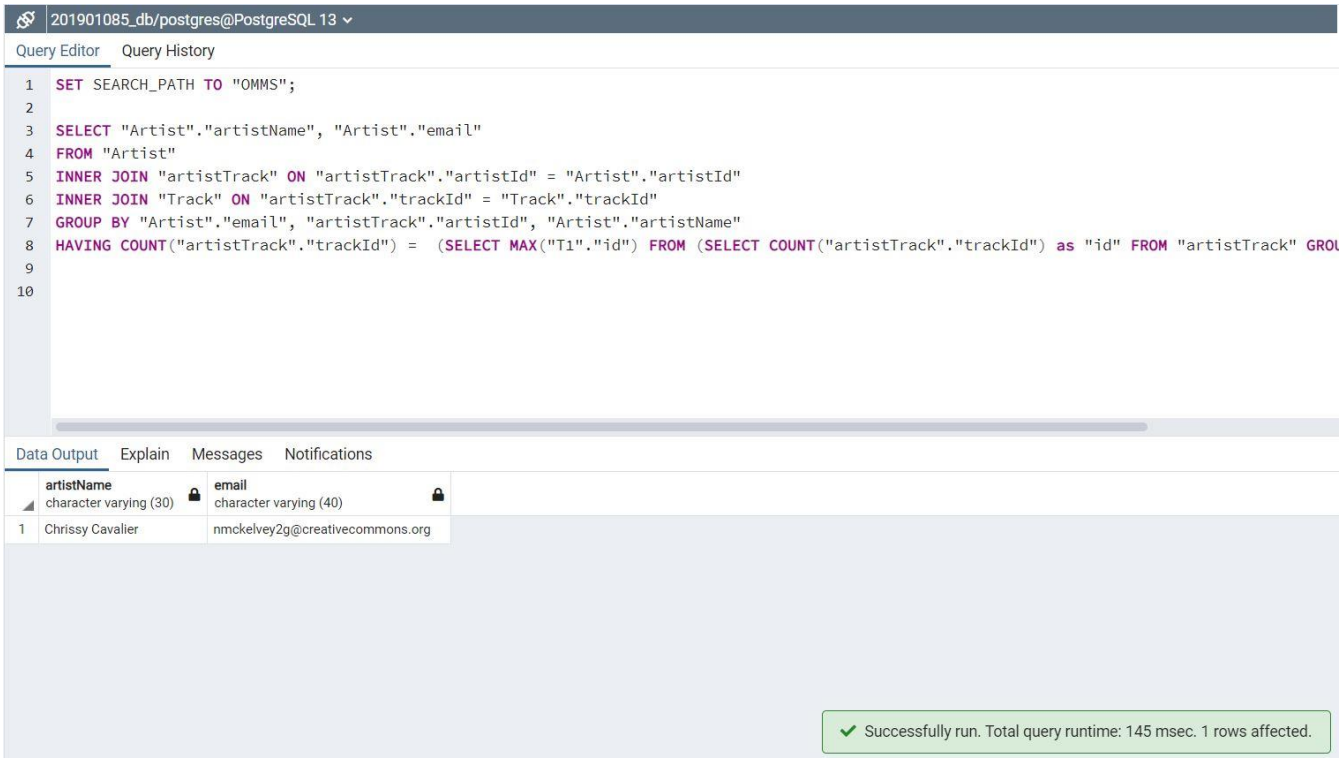
Result Tupples:



25. **Find email of Artist who is creating MAX no of songs**.

```
SET SEARCH_PATH TO "OMMS";

SELECT "Artist"."artistName", "Artist"."email"
FROM "Artist"
INNER JOIN "artistTrack" ON "artistTrack"."artistId" = "Artist"."artistId"
INNER JOIN "Track" ON "artistTrack"."trackId" = "Track"."trackId"
GROUP BY "Artist"."email", "artistTrack"."artistId", "Artist"."artistName"

HAVING COUNT("artistTrack"."trackId") =  (SELECT MAX("T1"."id") FROM
(SELECT COUNT("artistTrack"."trackId") as "id" FROM "artistTrack" GROUP BY
"artistId") AS "T1")
```

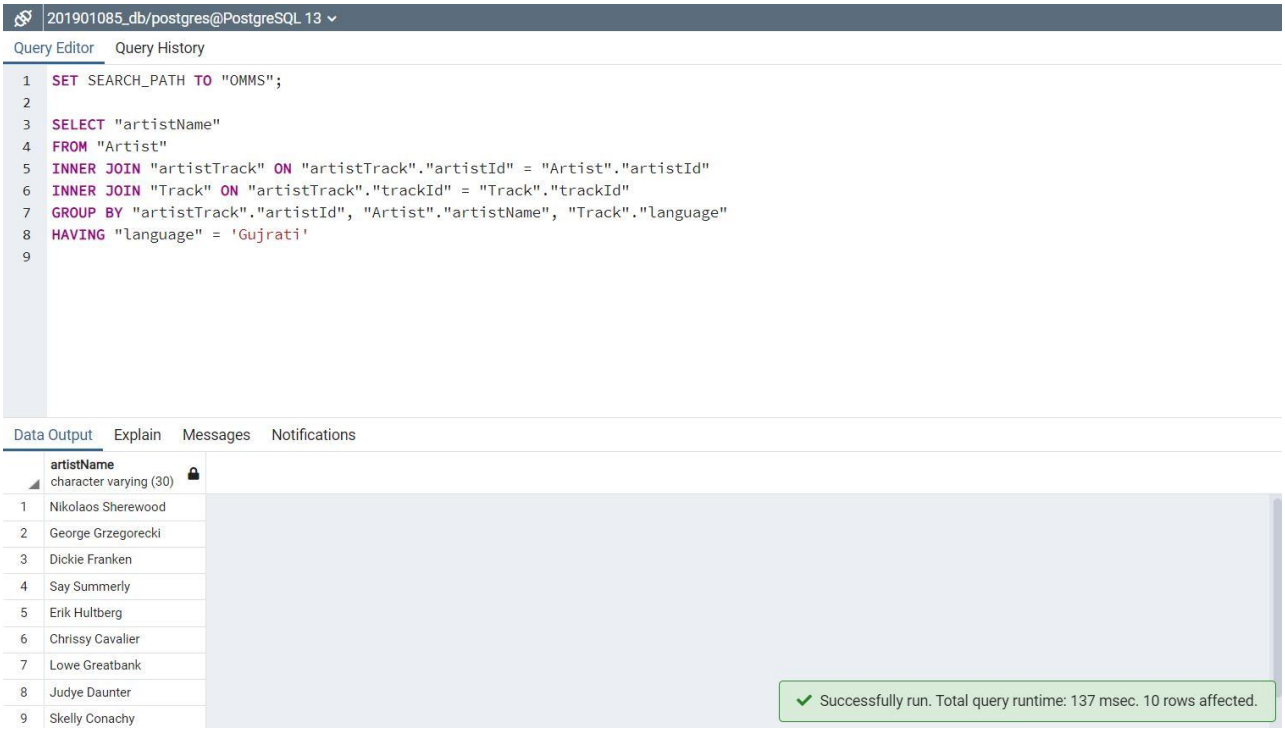
Result Tupples: 1 row



26. **Artist Names who made Tracks in 'Gujrati' language**.

```
SET SEARCH_PATH TO "OMMS";

SELECT "artistName"
FROM "Artist"
INNER JOIN "artistTrack" ON "artistTrack"."artistId" = "Artist"."artistId"
INNER JOIN "Track" ON "artistTrack"."trackId" = "Track"."trackId"
GROUP BY "artistTrack"."artistId", "Artist"."artistName", "Track"."language"
HAVING "language" = 'Gujrati'
```

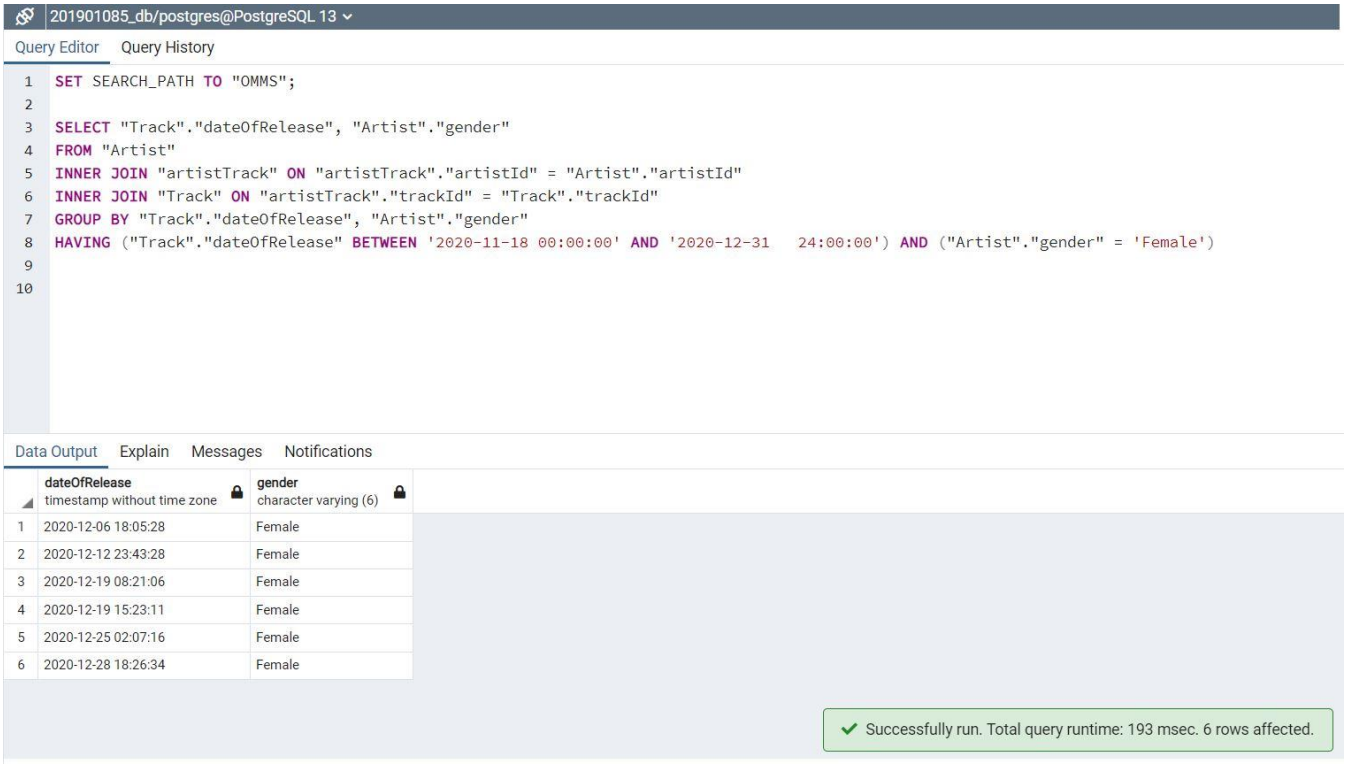
Result Tupples: 10 rows



27. **Female Artist Who Released their songs in 2020**.

```
SET SEARCH_PATH TO "OMMS";

SELECT "Track"."dateOfRelease", "Artist"."gender"
FROM "Artist"
INNER JOIN "artistTrack" ON "artistTrack"."artistId" = "Artist"."artistId"
INNER JOIN "Track" ON "artistTrack"."trackId" = "Track"."trackId"
GROUP BY "Track"."dateOfRelease", "Artist"."gender"
HAVING ("Track"."dateOfRelease" BETWEEN '2020-11-18 00:00:00' AND '2020-
12-31          24:00:00') AND ("Artist"."gender" = 'Female')
```

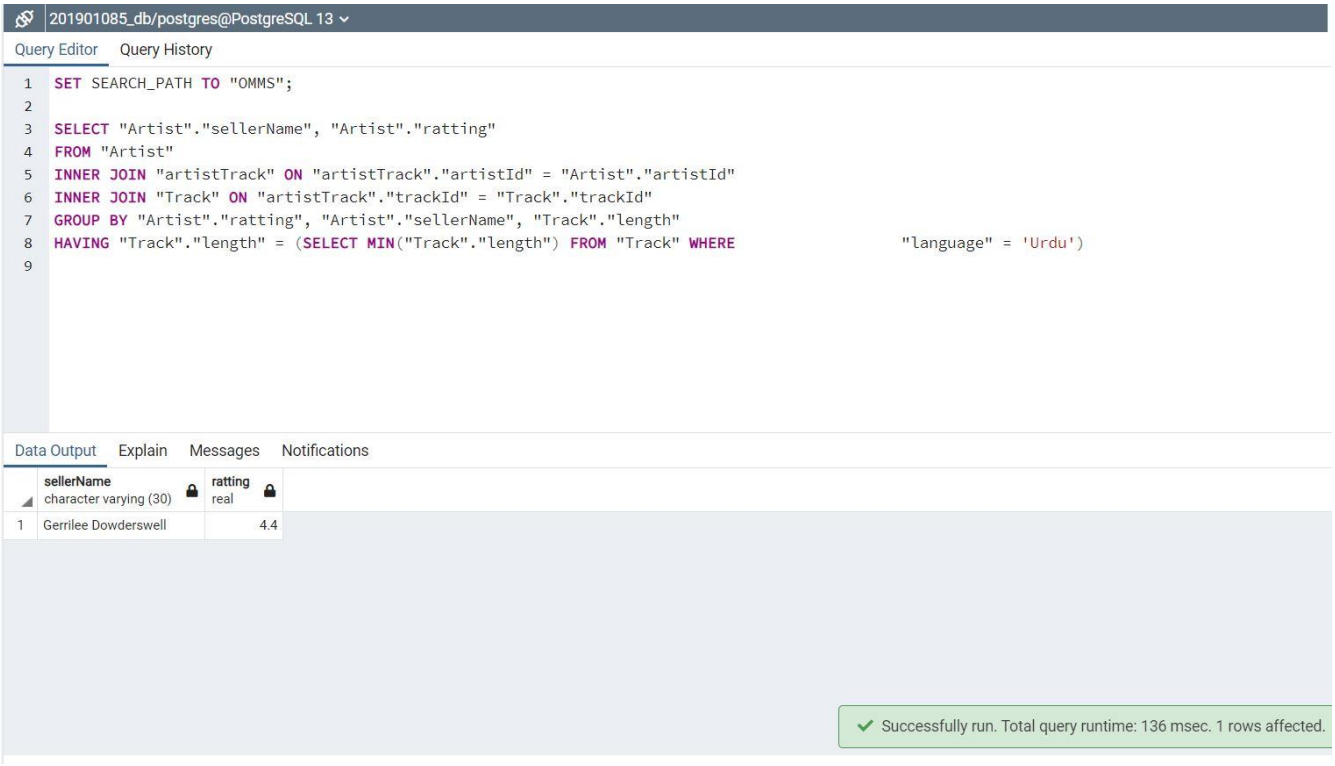
Result Tupples: 6 rows



28. **find Seller ratting who made smallest length song in 'Urdu'.**

```
SET SEARCH_PATH TO "OMMS";


SELECT "Artist"."sellerName", "Artist"."ratting"
FROM "Artist"
INNER JOIN "artistTrack" ON "artistTrack"."artistId" = "Artist"."artistId"
INNER JOIN "Track" ON "artistTrack"."trackId" = "Track"."trackId"
GROUP BY "Artist"."ratting", "Artist"."sellerName", "Track"."length"
HAVING "Track"."length" = (SELECT MIN("Track"."length") FROM "Track"
WHERE                                    "language" = 'Urdu')
```

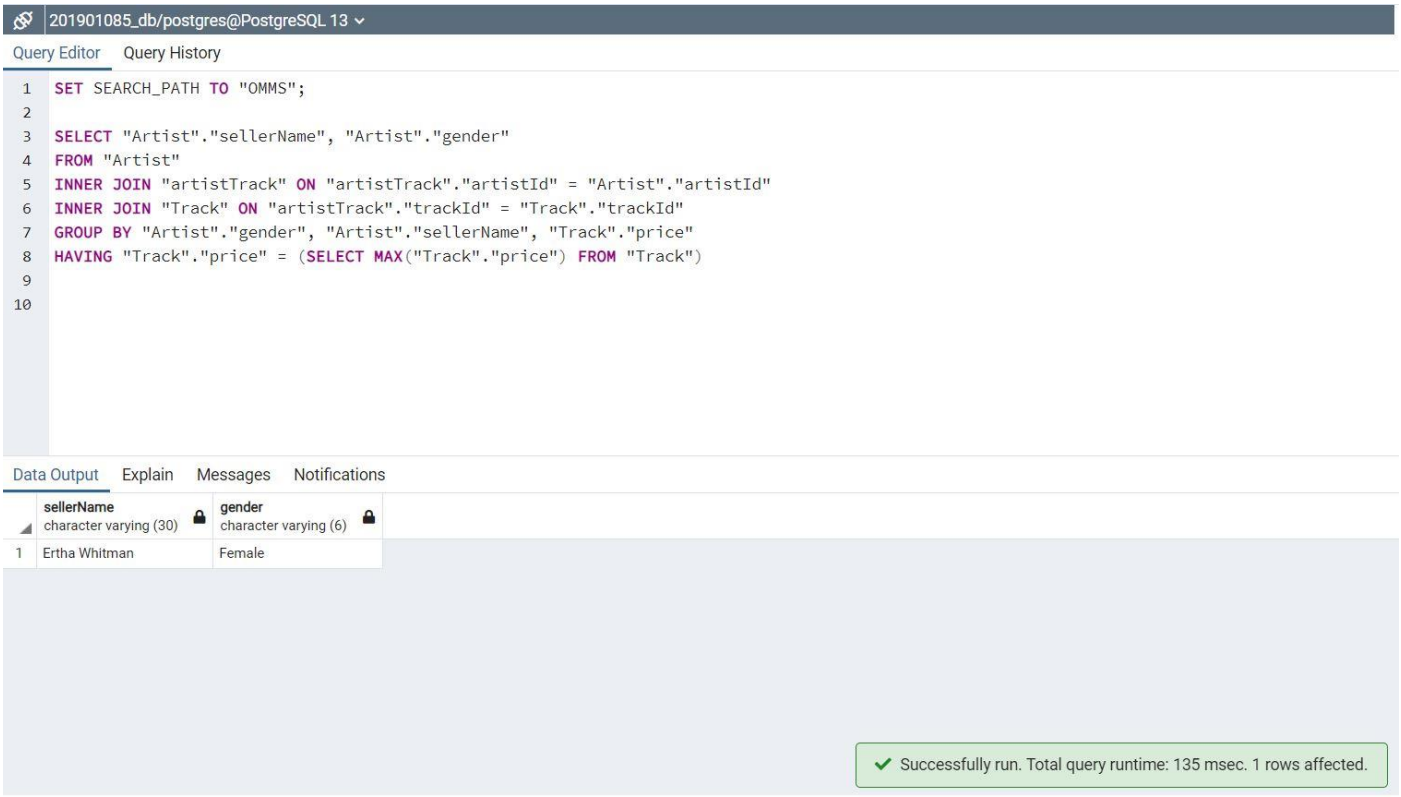
Result Tupples: 1 row



29. **seller Gender who sell Track with highest amount**.

```
SET SEARCH_PATH TO "OMMS";


SELECT "Artist"."sellerName", "Artist"."gender"
FROM "Artist"
INNER JOIN "artistTrack" ON "artistTrack"."artistId" = "Artist"."artistId"
INNER JOIN "Track" ON "artistTrack"."trackId" = "Track"."trackId"
GROUP BY "Artist"."gender", "Artist"."sellerName", "Track"."price"
HAVING "Track"."price" = (SELECT MAX("Track"."price") FROM "Track")
```

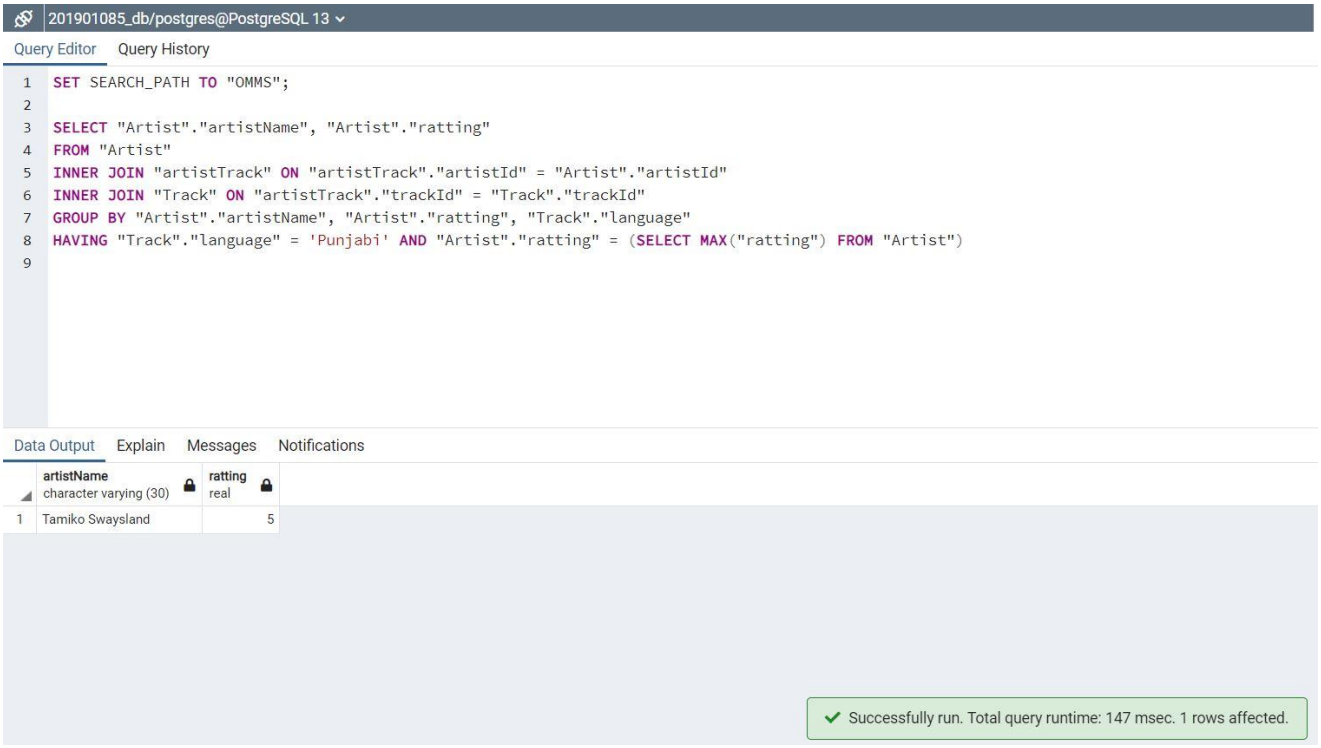
Result Tupples: 1 row



30. **Find Highest rated 'Punjabi' singer**.

```
SET SEARCH_PATH TO "OMMS";


SELECT "Artist"."artistName", "Artist"."ratting"
FROM "Artist"
INNER JOIN "artistTrack" ON "artistTrack"."artistId" = "Artist"."artistId"
INNER JOIN "Track" ON "artistTrack"."trackId" = "Track"."trackId"
GROUP BY "Artist"."artistName", "Artist"."ratting", "Track"."language"
HAVING "Track"."language" = 'Punjabi' AND "Artist"."ratting" = (SELECT
MAX("ratting") FROM "Artist")
```

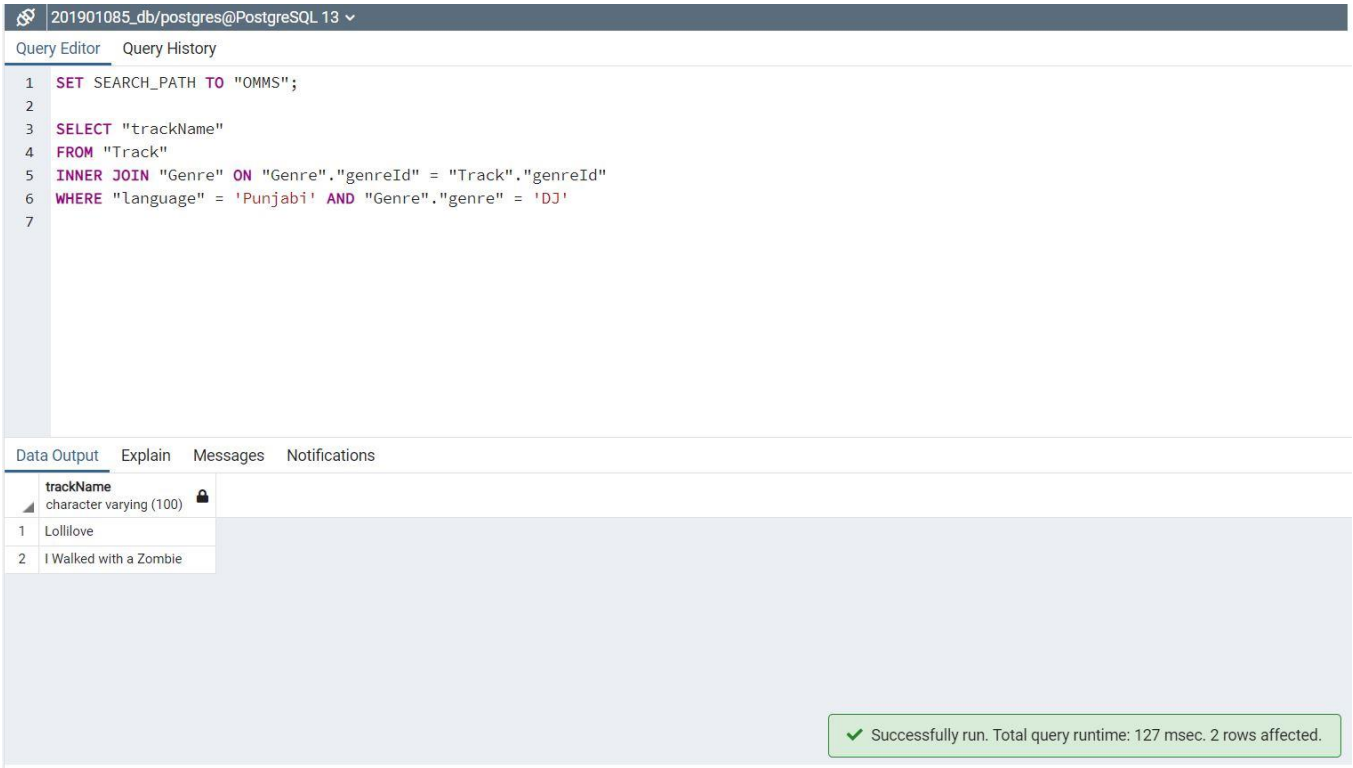
Result Tupples: 1 row



31. **find Punjabi DJ songs**.

```
SET SEARCH_PATH TO "OMMS";

SELECT "trackName"
FROM "Track"
INNER JOIN "Genre" ON "Genre"."genreId" = "Track"."genreId"
WHERE "language" = 'Punjabi' AND "Genre"."genre" = 'DJ'
```

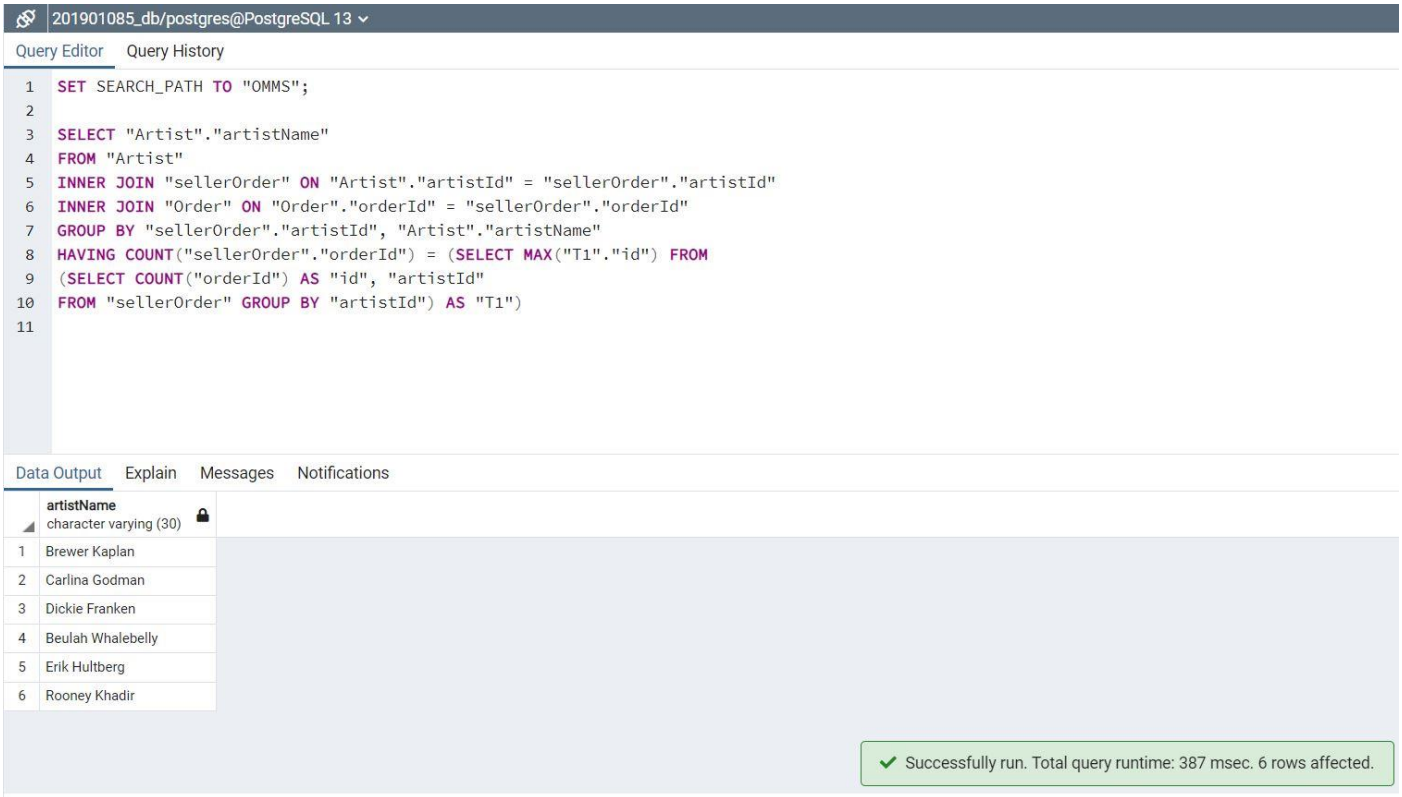
Result Tupples: 2 rows



32. **List artistNames who got MAX orders for their tracks**.

```
SET SEARCH_PATH TO "OMMS";

SELECT "Artist"."artistName"
FROM "Artist"
INNER JOIN "sellerOrder" ON "Artist"."artistId" = "sellerOrder"."artistId"
INNER JOIN "Order" ON "Order"."orderId" = "sellerOrder"."orderId"
GROUP BY "sellerOrder"."artistId", "Artist"."artistName"
HAVING COUNT("sellerOrder"."orderId") = (SELECT MAX("T1"."id") FROM
(SELECT COUNT("orderId") AS "id", "artistId"
FROM "sellerOrder" GROUP BY "artistId") AS "T1")
```

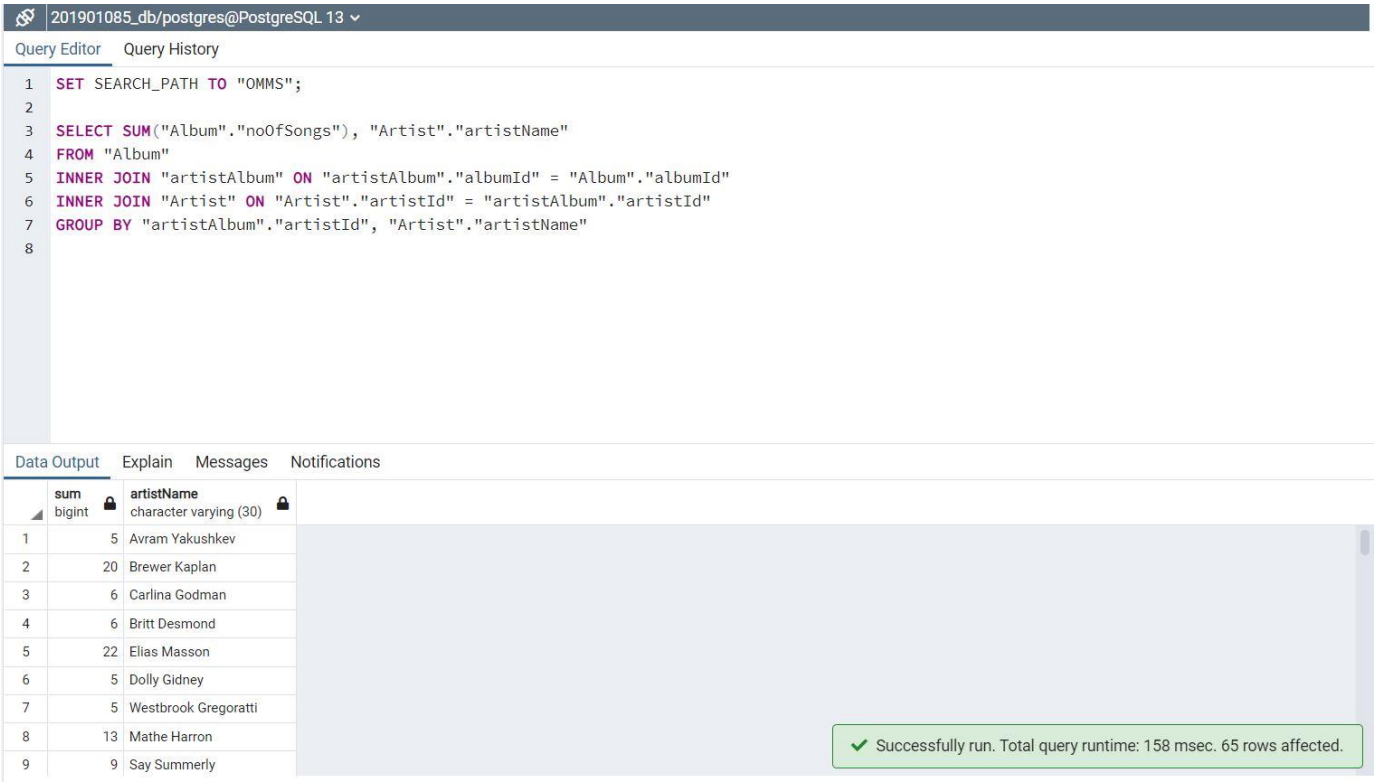
Result Tupples: 6 rows



33. **Total No of album songs created by Artist**.

```
SET SEARCH_PATH TO "OMMS";

SELECT SUM("Album"."noOfSongs"), "Artist"."artistName"
FROM "Album"
INNER JOIN "artistAlbum" ON "artistAlbum"."albumId" = "Album"."albumId"
INNER JOIN "Artist" ON "Artist"."artistId" = "artistAlbum"."artistId"
GROUP BY "artistAlbum"."artistId", "Artist"."artistName"
```

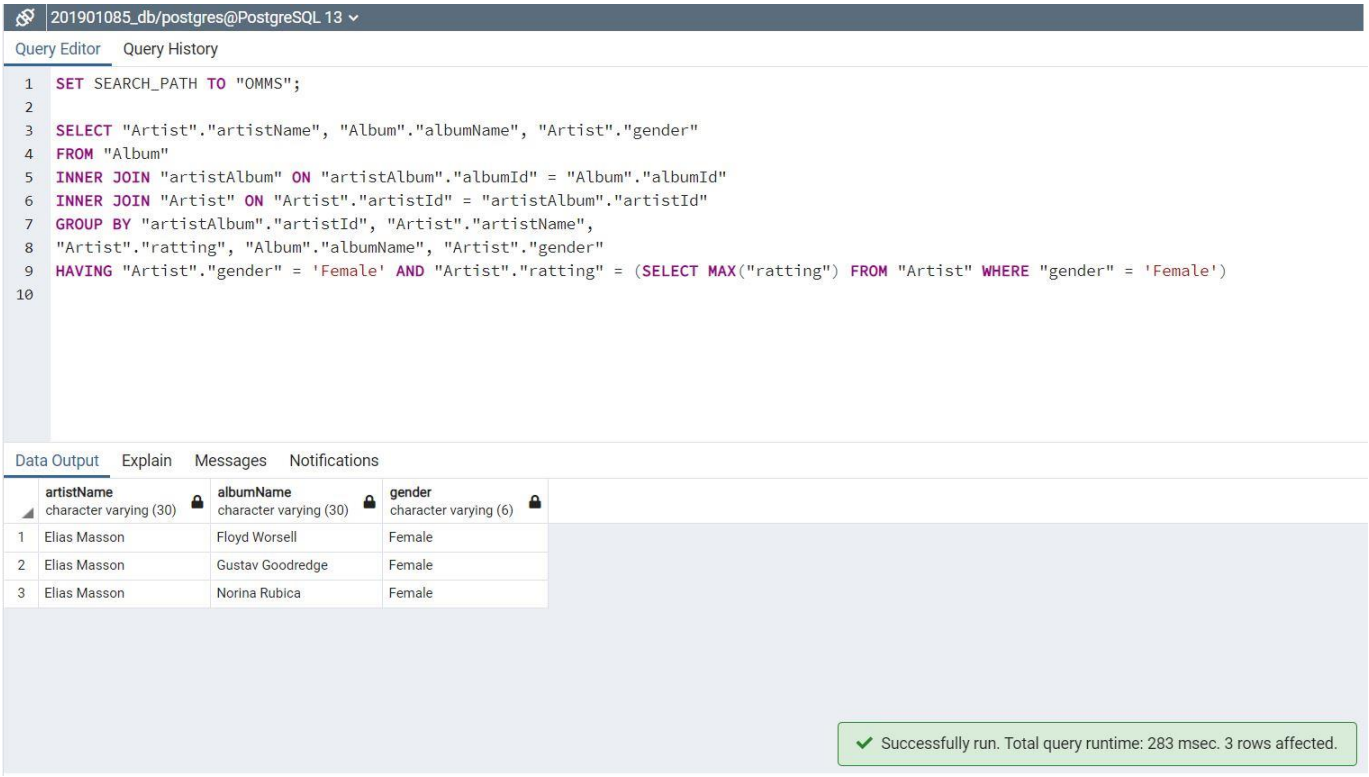
Result Tupples: 65 rows



34. **find Album names created by Female Artist Having MAX ratting**.

```
SET SEARCH_PATH TO "OMMS";


SELECT "Artist"."artistName", "Album"."albumName", "Artist"."gender"
FROM "Album"
INNER JOIN "artistAlbum" ON "artistAlbum"."albumId" = "Album"."albumId"
INNER JOIN "Artist" ON "Artist"."artistId" = "artistAlbum"."artistId"
GROUP BY "artistAlbum"."artistId", "Artist"."artistName",
"Artist"."ratting", "Album"."albumName", "Artist"."gender"
HAVING "Artist"."gender" = 'Female' AND "Artist"."ratting" = (SELECT
MAX("ratting") FROM "Artist" WHERE "gender" = 'Female')
```

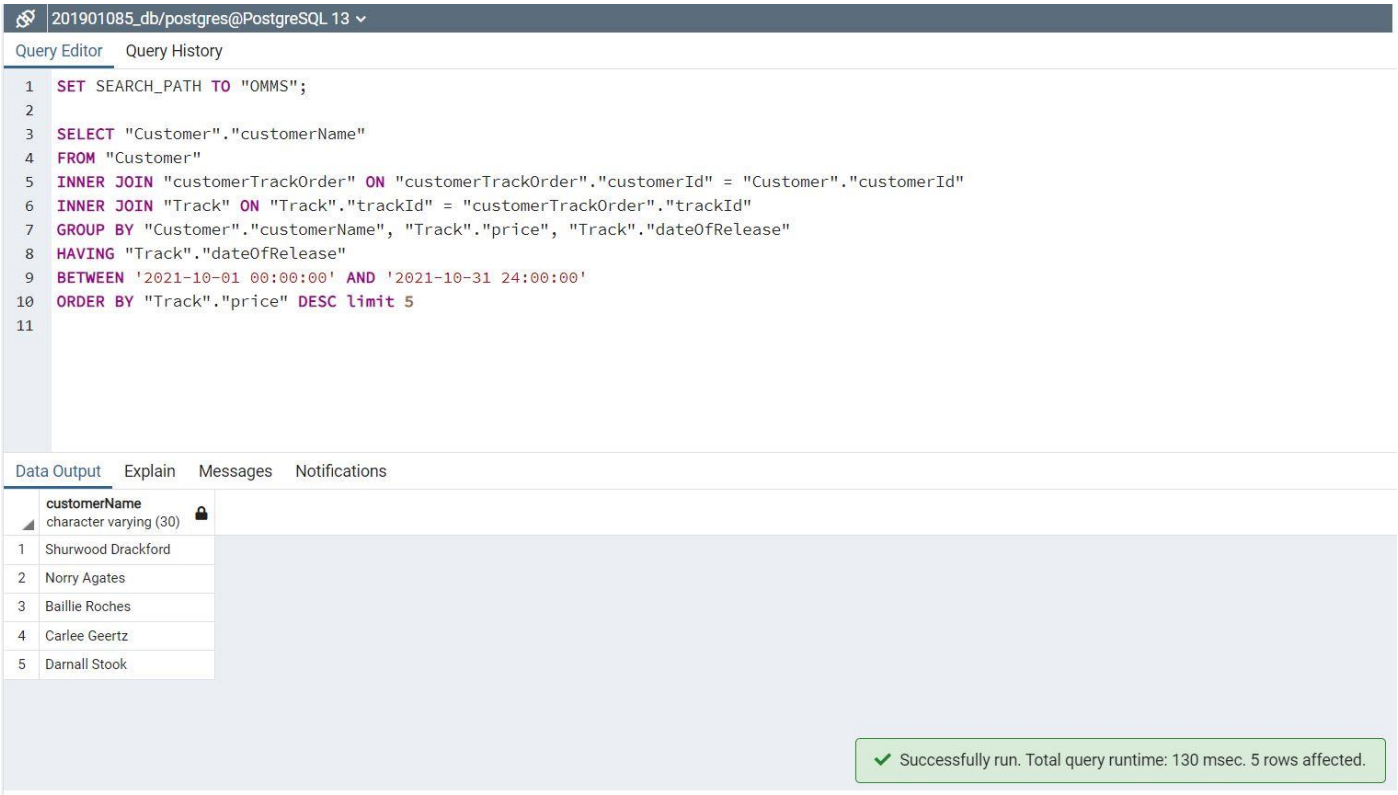
Result Tupples: 3 rows



35. **Who purchased Top 5(in terms of price) Released songs in Navratri month**.

```
SET SEARCH_PATH TO "OMMS";

SELECT "Customer"."customerName"
FROM "Customer"
INNER JOIN "customerTrackOrder" ON "customerTrackOrder"."customerId" =
"Customer"."customerId"
INNER JOIN "Track" ON "Track"."trackId" = "customerTrackOrder"."trackId"
GROUP BY "Customer"."customerName", "Track"."price", "Track"."dateOfRelease"
HAVING "Track"."dateOfRelease"
BETWEEN '2021-10-01 00:00:00' AND '2021-10-31 24:00:00'
ORDER BY "Track"."price" DESC limit 5
```

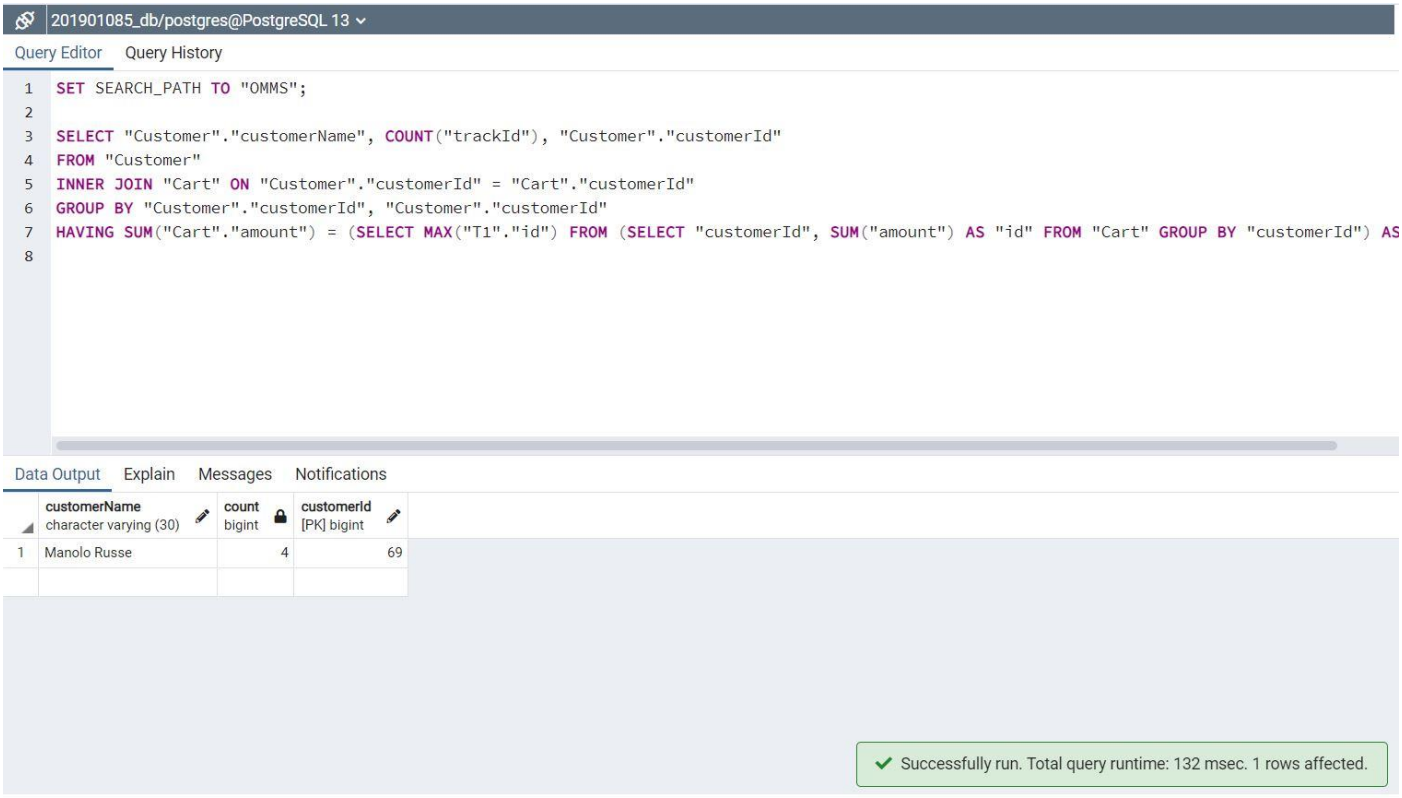
Result Tupples: 5 rows



36. **Find customer whose total amount of products that are included in cart is MAXIMUM.**

```
SET SEARCH_PATH TO "OMMS";

SELECT "Customer"."customerName", COUNT("trackId"), "Customer"."customerId"
FROM "Customer"
INNER JOIN "Cart" ON "Customer"."customerId" = "Cart"."customerId"
GROUP BY "Customer"."customerId", "Customer"."customerId"
HAVING SUM("Cart"."amount") = (SELECT MAX("T1"."id") FROM (SELECT "customerId", SUM("amount") AS "id" FROM "Cart" GROUP BY "customerId") AS "T1")
```

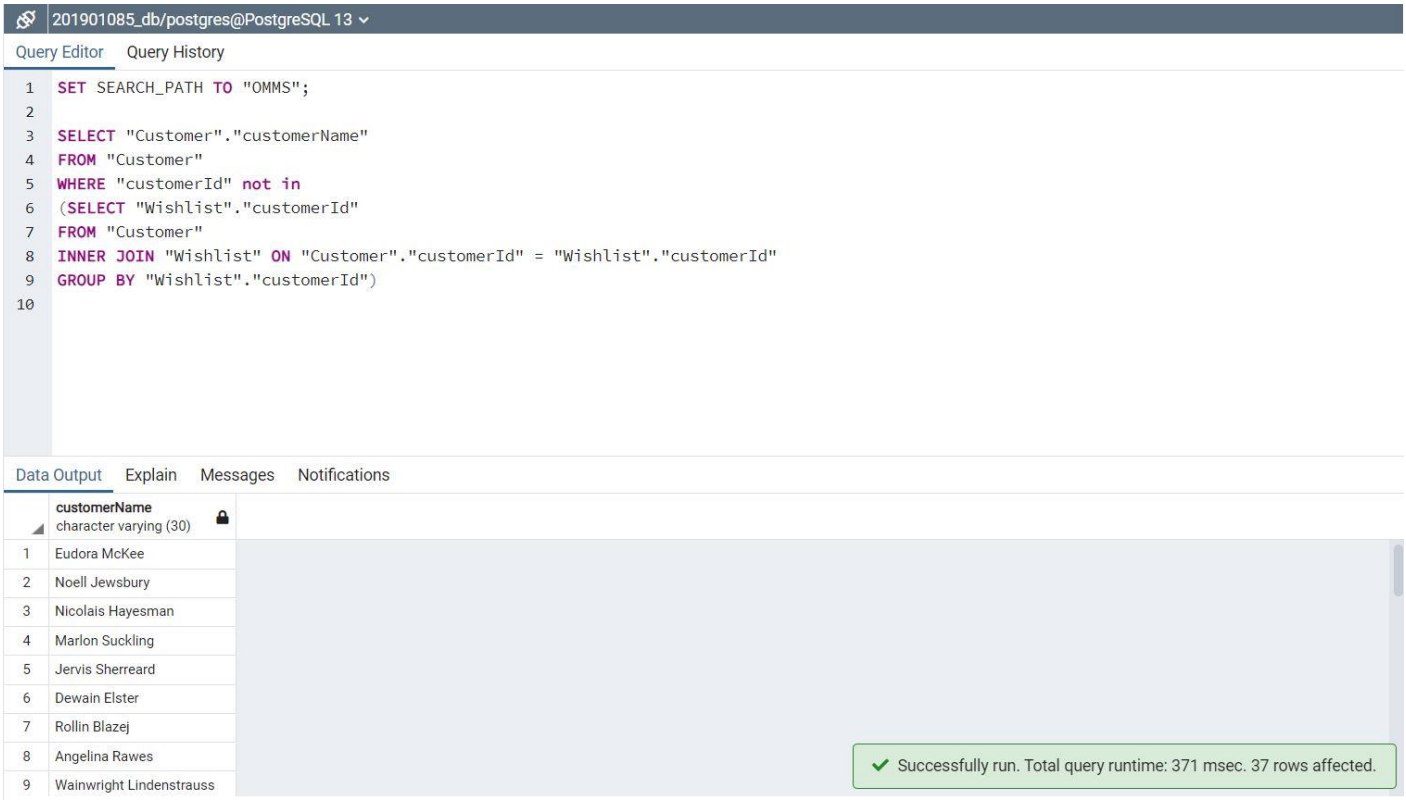
Result Tupples: 1 row



37. **List Customers who doesn't have any items in wish list**.

```
SET SEARCH_PATH TO "OMMS";

SELECT "Customer"."customerName"
FROM "Customer"
WHERE "customerId" not in
(SELECT "Wishlist"."customerId"
FROM "Customer"
INNER JOIN "Wishlist" ON "Customer"."customerId" = "Wishlist"."customerId"
GROUP BY "Wishlist"."customerId")
```

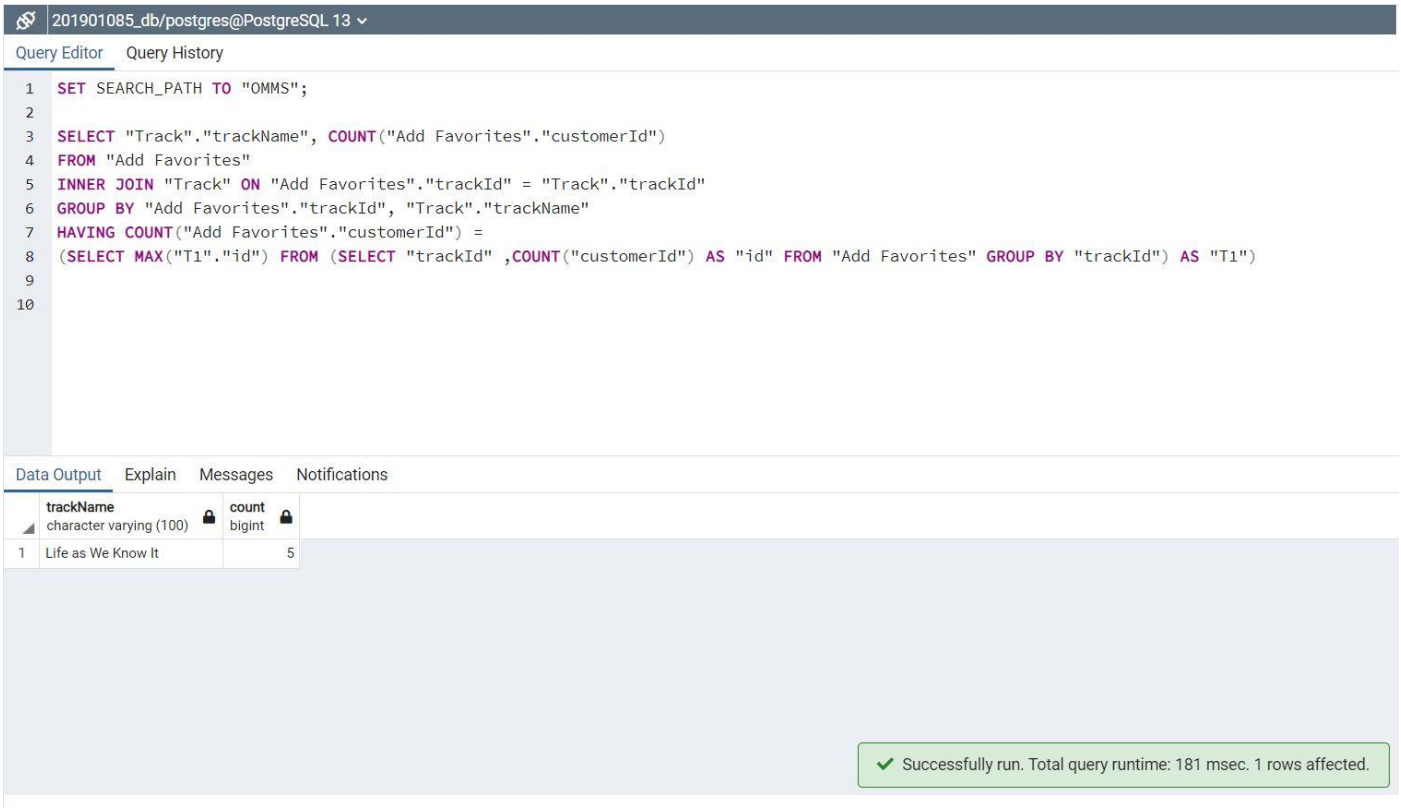
Result Tupples: 37 rows



38. **Find Most Favorite Track on Platform**.

```
SET SEARCH_PATH TO "OMMS";

SELECT "Track"."trackName", COUNT("Add Favorites"."customerId")
FROM "Add Favorites"
INNER JOIN "Track" ON "Add Favorites"."trackId" = "Track"."trackId"
GROUP BY "Add Favorites"."trackId", "Track"."trackName"
HAVING COUNT("Add Favorites"."customerId") =
(SELECT MAX("T1"."id") FROM (SELECT "trackId" ,COUNT("customerId") AS "id"
FROM "Add Favorites" GROUP BY "trackId") AS "T1")
```

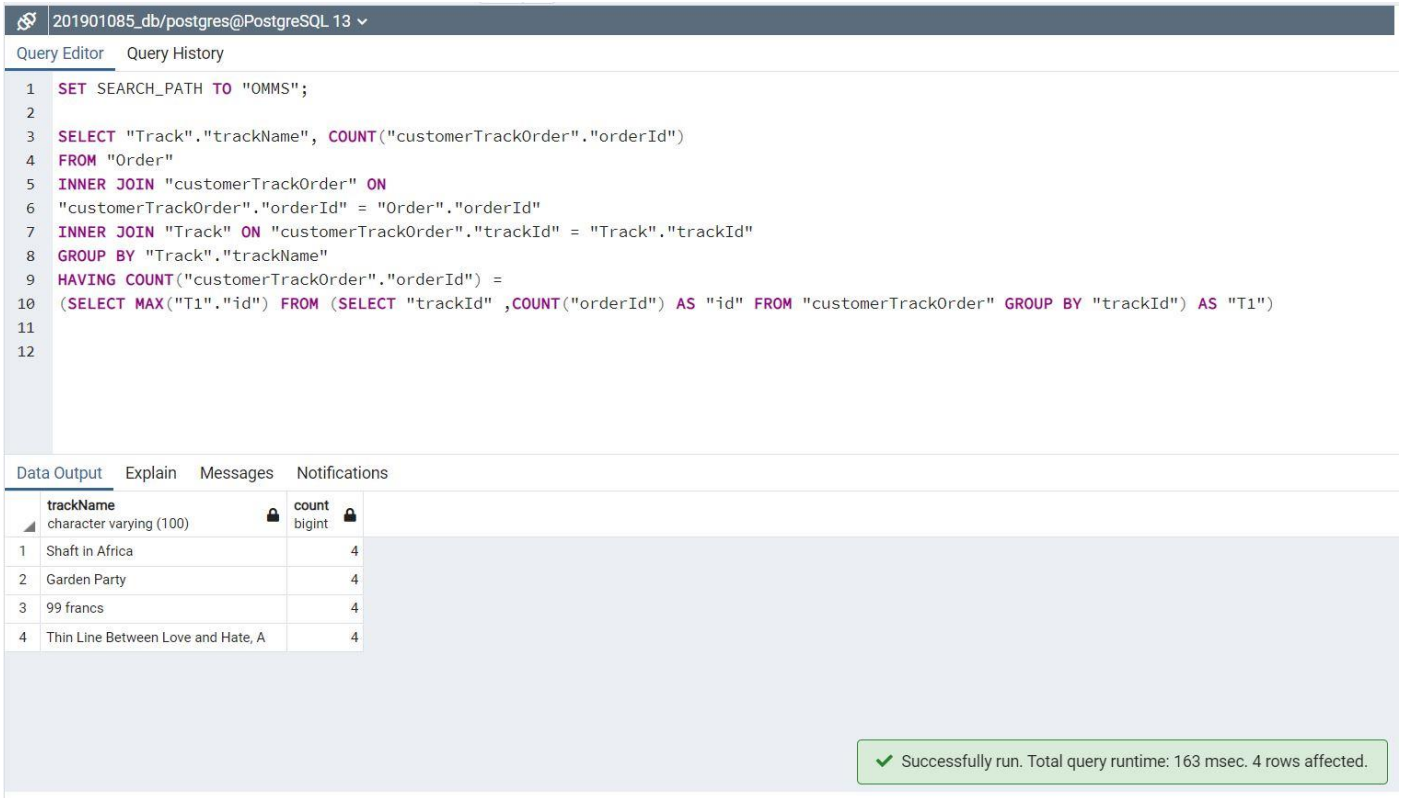
Result Tupples: 1 row



39. **Most Ordered song on Platform**.

```
SET SEARCH_PATH TO "OMMS";

SELECT "Track"."trackName", COUNT("customerTrackOrder"."orderId")
FROM "Order"
INNER JOIN "customerTrackOrder" ON
"customerTrackOrder"."orderId" = "Order"."orderId"
INNER JOIN "Track" ON "customerTrackOrder"."trackId" = "Track"."trackId"
GROUP BY "Track"."trackName"
HAVING COUNT("customerTrackOrder"."orderId") =
(SELECT MAX("T1"."id") FROM (SELECT "trackId" , COUNT("orderId") AS "id"
FROM "customerTrackOrder" GROUP BY "trackId") AS "T1")
```

Result Tupples: 4 rows



40. **Normal Function For Counting No of Songs Genre wise**.

```
CREATE OR REPLACE FUNCTION "noOfSongsForEveryGenre"()
RETURNS TABLE (a bigint, b character varying (10))
LANGUAGE 'plpgsql'
AS
$Body$

BEGIN
RETURN QUERY EXECUTE FORMAT('SELECT COUNT("Track"."trackId"),
"Genre"."genre"
FROM "Track"
INNER JOIN "Genre" ON "Genre"."genreId" = "Track"."genreId"
GROUP BY "Genre"."genre"');
END;


$Body$;
```

201901085_db/postgres@PostgreSQL 13

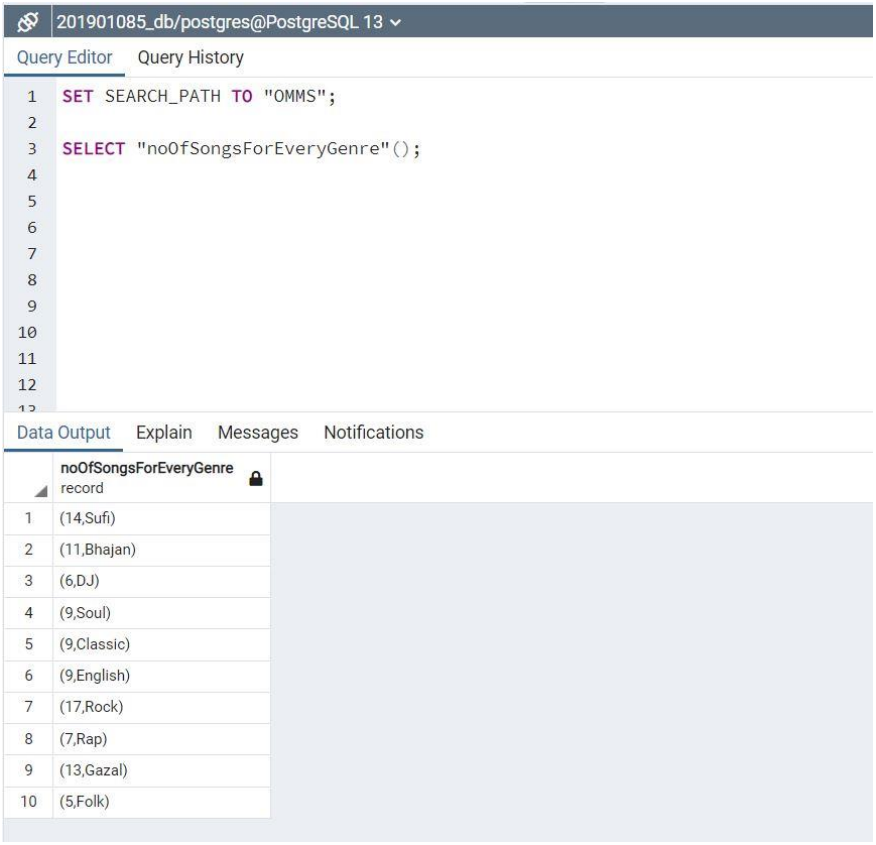
Query Editor | Query History

```
1  SET SEARCH_PATH TO "OMMS";
2
3  CREATE OR REPLACE FUNCTION "noOfSongsForEveryGenre"()
4  RETURNS TABLE (a bigint, b character varying (10))
5  LANGUAGE 'plpgsql'
6  AS
7  $Body$
8
9  BEGIN
10 RETURN QUERY EXECUTE FORMAT('SELECT COUNT("Track"."trackId"), "Genre"."genre"
11 FROM "Track"
12 INNER JOIN "Genre" ON "Genre"."genreId" = "Track"."genreId"
13 GROUP BY "Genre"."genre"');
14 END;
15
16 $Body$;
17
```

Data Output | Explain | Messages | Notifications

```
CREATE FUNCTION

Query returned successfully in 156 msec.
```

FUNCTION CREATED



Function CALL Result

41. **Trigger Function To Update Current Date in Order Table whenever new Order is placed or Updated**.

```
CREATE OR REPLACE FUNCTION "OMMS"."orderTime"()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    VOLATILE
    COST 100
AS $BODY$
DECLARE
xx timestamp without time zone;
yy timestamp without time zone;

BEGIN

xx = now()::timestamp(0);

NEW."date" = now()::timestamp(0);

SELECT NEW."date" into yy FROM "Order";

if(yy = xx) then
RAISE NOTICE 'DateTime is Updated on new Order That is placed Currently.';
else
RAISE NOTICE 'NO you are not that good';
end if;

RETURN NEW;

END
$BODY$;
```

201901085_db/postgres@PostgreSQL 13

Query Editor | Query History

```
1  SET SEARCH_PATH TO "OMMS";
2
3  INSERT INTO "OMMS"."Order"("orderId", "customerId" ,"originalPrice", discount, "totalAmount")
4      VALUES (101, 5, 20, 10, 18);
5
6
7
8
9
10
11
12
13
14
15
16
17
```

Data Output | Explain | Messages | Notifications

```
NOTICE:  DateTime is Updated on new Order That is placed Currently.
INSERT 0 1

Query returned successfully in 101 msec.
```

BEFORE INSERT RAISE NOTICE CAME FROM TRIGGER FUNCTION

201901085_db/postgres@PostgreSQL 13

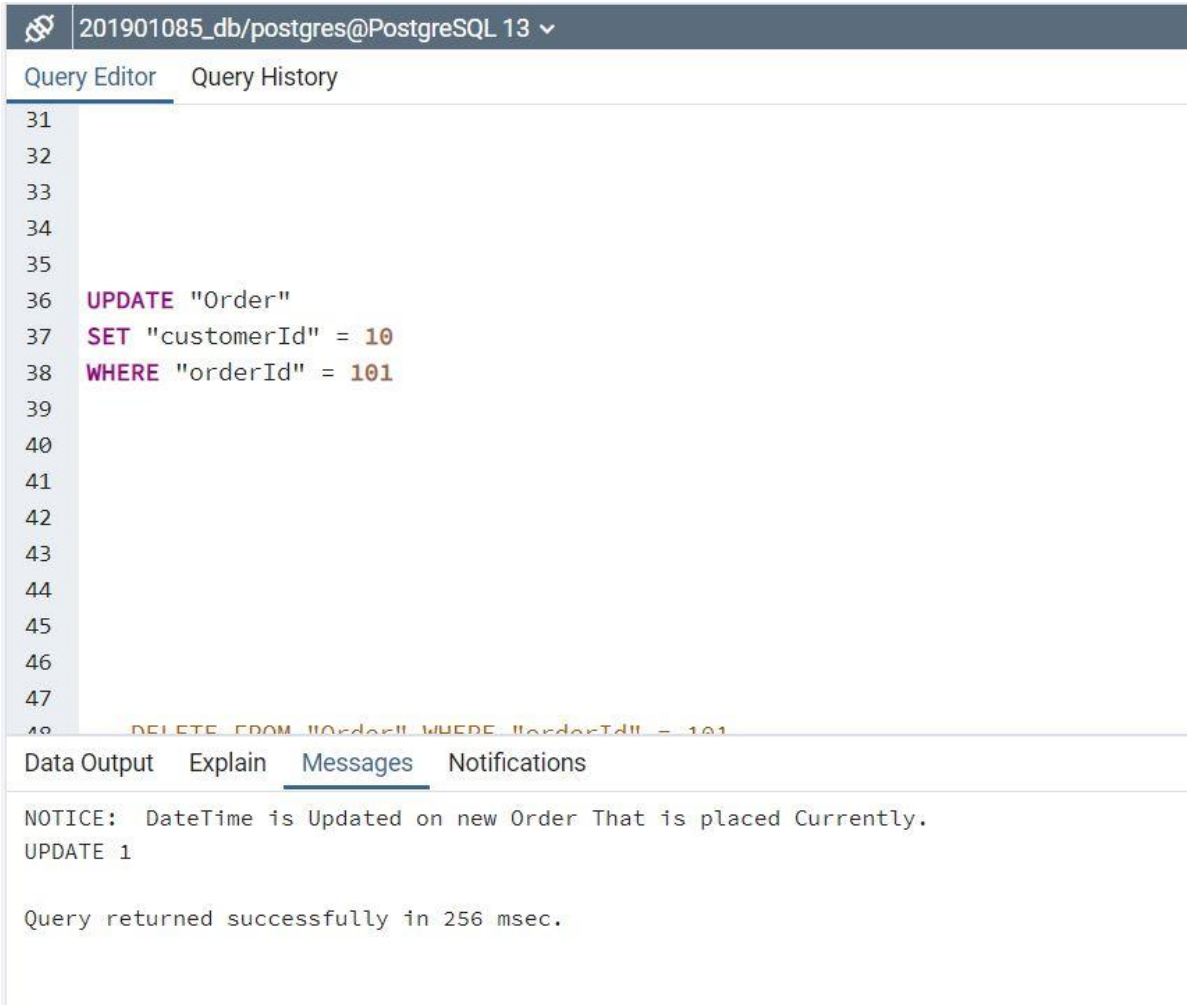
Query Editor | Query History

```
13
14
15
16
17
18
19
20
21 SELECT * FROM "Order"
22
23
24
25
26
27
28
29
30
```

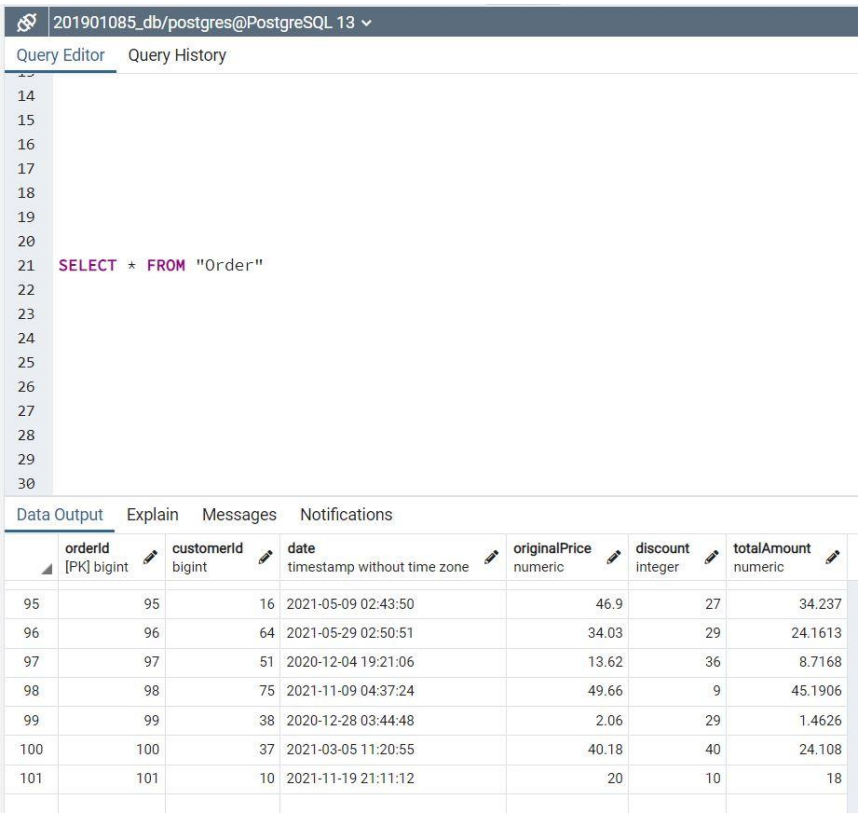
Data Output | Explain | Messages | Notifications

	orderId [PK] bigint	customerId bigint	date timestamp without time zone	originalPrice numeric	discount integer	totalAmount numeric
95	95	16	2021-05-09 02:43:50	46.9	27	34.237
96	96	64	2021-05-29 02:50:51	34.03	29	24.1613
97	97	51	2020-12-04 19:21:06	13.62	36	8.7168
98	98	75	2021-11-09 04:37:24	49.66	9	45.1906
99	99	38	2020-12-28 03:44:48	2.06	29	1.4626
100	100	37	2021-03-05 11:20:55	40.18	40	24.108
101	101	5	2021-11-19 21:08:39	20	10	18

Current Date is Set on “orderId” 101



BEFORE UPDATE TRIGGER RETURNED RAISED NOTICE



	orderId [PK] bigint	customerId bigint	date timestamp without time zone	originalPrice numeric	discount integer	totalAmount numeric
95	95	16	2021-05-09 02:43:50	46.9	27	34.237
96	96	64	2021-05-29 02:50:51	34.03	29	24.1613
97	97	51	2020-12-04 19:21:06	13.62	36	8.7168
98	98	75	2021-11-09 04:37:24	49.66	9	45.1906
99	99	38	2020-12-28 03:44:48	2.06	29	1.4626
100	100	37	2021-03-05 11:20:55	40.18	40	24.108
101	101	10	2021-11-19 21:11:12	20	10	18

Trigger Updated DATE on “orderId” 101

Section7

Project Code with output screenshots

Front-End Development

Node-Postgress Sql code for Front-End

```
const express = require('express');
const bodyParser = require('body-parser');
const ejs = require('ejs');
const _ = require('lodash');
const {
  Pool
} = require('pg');

const storage = require(__dirname + "/storage.js");

const app = express();

app.use(express.urlencoded({
  extended: true
}));

app.set('view engine', 'ejs');
app.use(express.static("public"));

// global vars

var CustomerLoginInfo = {
  customerId: "Null",
  customerName: "Null",
  email: "Null",
  dateOfBirth: "Null",
  address: "Null"
};
```

```
var ArtistLoginInfo = {
  artistName: "",
  sellerName: "",
  email: "",
  ratting: ""
};

var Songs = {};

var Tracks = [
  {
    trackName: "Null"
  }
];



var Albums = [{
  albumName: "Null"
}];



let pool = new Pool({
  user: 'postgres',
  host: 'localhost',
  database: '201901085_db',
  password: 'admin',
  port: 5432
});



app.get("/info/get", (req, res) => {
  try {
    pool.connect(async (err, client, release) => {
      client.query('SET SEARCH_PATH TO "OMMS"');
```

```
    const Query = `SELECT "noOfSongsForEveryGenre"();`;

    let resp = await client.query(Query);
    res.send(resp.rows);
   });
  } catch (err) {
   console.log(err);
  }
});

app.get("/", (req, res) => {
  try {
   pool.connect(async (err, client, release) => {
    client.query('SET SEARCH_PATH TO "OMMS"');
    let Query = `SELECT * FROM "Track"`;

    let resp = await client.query(Query);




    res.render("home", {
      // Artist: resp.rows
      Songs: resp.rows,
      name: "Look at The Tracks",
      visualLogin: "",
      visualCart: "visually-hidden",
      CustomerId: 1
     });
   });
  } catch (err) {
   console.log(err);
  }
```

```
});

app.get("/customerRegister", function(req, res) {
  res.render("customerRegister", {
    visualLogin: "",
    visualCart: "visually-hidden",
    message: "",
    CustomerId: 1
  });
});



app.get("/artistRegister", function(req, res) {
  res.render("artistRegister", {
    visualLogin: "",
    visualCart: "visually-hidden",
    message1: "",
    message2: "",
    CustomerId: 1
  });
});


app.get("/customerLogin", function(req, res) {
  res.render("customerLogin", {
    visualLogin: "",
    visualCart: "visually-hidden",
    CustomerId: 1
  });
});


app.get("/artistLogin", function(req, res) {
  res.render("artistLogin", {
    visualLogin: "",
    visualCart: "visually-hidden",
```

```
    CustomerId: 1
  });
});




app.get("/customerLoggedIn", function(req, res) {

  res.render("customerLoggedIn", {
    customerData: CustomerLoginInfo,
    visualLogin: "visually-hidden",
    visualCart: "",
    CustomerId: CustomerLoginInfo.customerId
  });
});

app.get("/artistLoggedIn", function(req, res) {

  res.render("artistLoggedIn", {
    artistData: ArtistLoginInfo,
    visualLogin: "visually-hidden",
    visualCart: "visually-hidden",
    CustomerId: 1
  });
});

app.post("/customerRegister", function(req, res) {

  const customerName = req.body.CustomerName;
  const email = req.body.Email;
  const DOB = req.body.DOB;
  const Address = req.body.Address;
  const Username = req.body.Username;
  const Password = req.body.Password;
```

```
try {
  pool.connect(async (err, client, release) => {
    client.query('SET SEARCH_PATH TO "OMMS"');
    let Query1 = `SELECT * FROM "Customer"`;

    let resp1 = await client.query(Query1);
    var temp = "false";

    for (let i = 0; i < resp1.rows.length; i++) {
      if (resp1.rows[i].userName === Username) {

        temp = "true";
        res.render("customerRegister", {
          visualLogin: "",
          visualCart: "visually-hidden",
          message: "* Username already exists Please try another one",
          CustomerId: 1
        });
      }
    }

    if(temp === "false") {
      const id = resp1.rows.length + 1;

      const text = `INSERT INTO "Customer"("customerId", "customerName", "userName",
                    "password_", "email", "dateOfBirth", "address")
                VALUES ($1, $2, $3, $4, $5, $6, $7)`;
      const values = [id, customerName, Username, Password, email, DOB, Address];


      client.query('SET SEARCH_PATH TO "OMMS"');

      let resp2 = await client.query(text, values);
```

```
    client.query('SET SEARCH_PATH TO "OMMS"');
    let Query2 = `SELECT * FROM "Track"`;

    let resp3 = await client.query(Query2);



      res.render("home", {
        Songs: resp3.rows,
        name: "Items Inserted!, You can look at the Tracks",
        visualLogin: "",
        visualCart: "visually-hidden",
        CustomerId: 1
      });
    }

  });
 } catch (err) {
  console.log(err);
 }
});


app.post("/artistRegister", function(req, res) {

 const artistName = req.body.ArtistName;
 const sellerName = req.body.SellerName;
 const email = req.body.Email;
 const gender = req.body.Gender;
 const Username = req.body.Username;
 const Password = req.body.Password;

 try {
  pool.connect(async (err, client, release) => {
    client.query('SET SEARCH_PATH TO "OMMS"');
```

```
let Query1 = `SELECT * FROM "Artist"`;


let resp1 = await client.query(Query1);
var temp = "false";


for (let i = 0; i < resp1.rows.length; i++) {
  if (resp1.rows[i].userName === Username) {


    temp = "true";
    res.render("artistRegister", {
      visualLogin: "",
      visualCart: "visually-hidden",
      message1: "* Username already exists Please try another one",
      message2: "",
      CustomerId: 1
    });
  }
}


if(temp === "false" && !(gender == "Male" || gender == "Female")) {
  temp = "true";
  res.render("artistRegister", {
    visualLogin: "",
    visualCart: "visually-hidden",
    message1: "",
    message2: "* Please write gender value as shown in instructions below that
       gender input.",
    CustomerId: 1
  });
}



if(temp === "false") {
  const id = resp1.rows.length + 1;
```

```
      const text = `INSERT INTO "Artist"("artistId", "artistName", "sellerName", "email", "userName", "password_",
                    "gender", "ratting")
                    VALUES ($1, $2, $3, $4, $5, $6, $7, $8);`;
      const values = [id, artistName, sellerName, email, Username, Password, gender, 1];



      client.query('SET SEARCH_PATH TO "OMMS"');


      let resp2 = await client.query(text, values);


    client.query('SET SEARCH_PATH TO "OMMS"');
    let Query2 = `SELECT * FROM "Track"`;


    let resp3 = await client.query(Query2);


      res.render("home", {
        Songs: resp3.rows,
        name: "Items Inserted!, You can look at the Tracks",
        visualLogin: "",
        visualCart: "visually-hidden",
        CustomerId: 1
      });
    }


  });
 } catch (err) {
  console.log(err);
 }
});

app.post("/customerLoggedIn", function(req, res) {


 try {
  pool.connect(async (err, client, release) => {
    await client.query('SET SEARCH_PATH TO "OMMS"');
```

```
    const Query = `SELECT * FROM "Customer"`;
    const resp = await client.query(Query);

    //console.log(req.body);

    for (let i = 0; i < resp.rows.length; i++) {
      //console.log(resp.rows[i]);

      if (req.body.Username === resp.rows[i].userName) {
        if (req.body.Password === resp.rows[i].password_) {
          // console.log(resp1.rows);

          CustomerLoginInfo = resp.rows[i];

          res.render("customerLoggedIn", {
            customerData: resp.rows[i],
            visualLogin: "visually-hidden",
            visualCart: "",
            CustomerId: resp.rows[i].customerId
          });
        }
      }
    }

    // res.redirect("/");

  });
 } catch (err) {
  console.log(err);
  res.redirect("/");
 }

});
```

```
app.post("/artistLoggedIn", function(req, res) {

 try {
   pool.connect(async (err, client, release) => {
    await client.query('SET SEARCH_PATH TO "OMMS"');
    const Query = `SELECT * FROM "Artist"`;
    const resp = await client.query(Query);

    //console.log(req.body);

    for (let i = 0; i < resp.rows.length; i++) {
      //console.log(resp.rows[i]);

      if (req.body.Username === resp.rows[i].userName) {
        if (req.body.Password === resp.rows[i].password_) {
          // console.log(resp1.rows);

          ArtistLoginInfo = resp.rows[i];

          client.query('SET SEARCH_PATH TO "OMMS"');

          const resp1 = await client.query(`SELECT "trackName" FROM "Track"
                    INNER JOIN "artistTrack" ON "Track"."trackId" =
            "artistTrack"."trackId"
                    INNER JOIN "Artist" ON "Artist"."artistId" =
            "artistTrack"."artistId"
                    GROUP BY "Track"."trackName", "Artist"."artistId"
                    HAVING "Artist"."artistId" = $1`, [resp.rows[i].artistId]);

          Tracks = resp1.rows;

          client.query('SET SEARCH_PATH TO "OMMS"');
```

```
        const resp2 = await client.query(`SELECT "albumName" FROM "Album"
                    INNER JOIN "artistAlbum" ON "Album"."albumId" = "artistAlbum"."albumId"
                    INNER JOIN "Artist" ON "Artist"."artistId" =   "artistAlbum"."artistId"
                    GROUP BY "Album"."albumName", "Artist"."artistId"
                    HAVING "Artist"."artistId" = $1`, [resp.rows[i].artistId]);

        Albums = resp2.rows;

        //Tracks = storage.getTracks(resp.rows[i].artistId);
        // console.log(Tracks);

        res.render("artistLoggedIn", {
          artistData: resp.rows[i],
          visualLogin: "visually-hidden",
          visualCart: "visually-hidden",
          ArtistId: resp.rows[i].artistId,
          CustomerId: 1
        });
      }
    }
  }

  // res.redirect("/");

  });
} catch (err) {
  console.log(err);
  res.redirect("/");
}

});
```

```
app.get("/tracks", (req, res) => {
  res.render("tracks", {
    Tracks: Tracks,
    visualLogin: "visually-hidden",
    visualCart: "visually-hidden",
    CustomerId: 1
  });
});


app.get("/albums", (req, res) => {
  res.render("albums", {
    Albums: Albums,
    visualLogin: "visually-hidden",
    visualCart: "visually-hidden",
    CustomerId: 1
  });
});


app.get("/cart/:Customer_Id", (req, res) => {


  try {
    pool.connect(async (err, client, release) => {


      const id = _.lowerCase(req.params.Customer_Id);


      await client.query('SET SEARCH_PATH TO "OMMS"');
      const resp = await client.query(`SELECT "Track"."trackName", "amount" FROM "Cart"
            INNER JOIN "Track" ON "Track"."trackId" = "Cart"."trackId"
            INNER JOIN "Customer" ON "Cart"."customerId" ="Customer"."customerId"
           GROUP BY "Customer"."customerId", "Track"."trackName", "amount"
           HAVING "Customer"."customerId" = $1`, [id]);


      res.render("cart", {
        cartData: resp.rows,
```

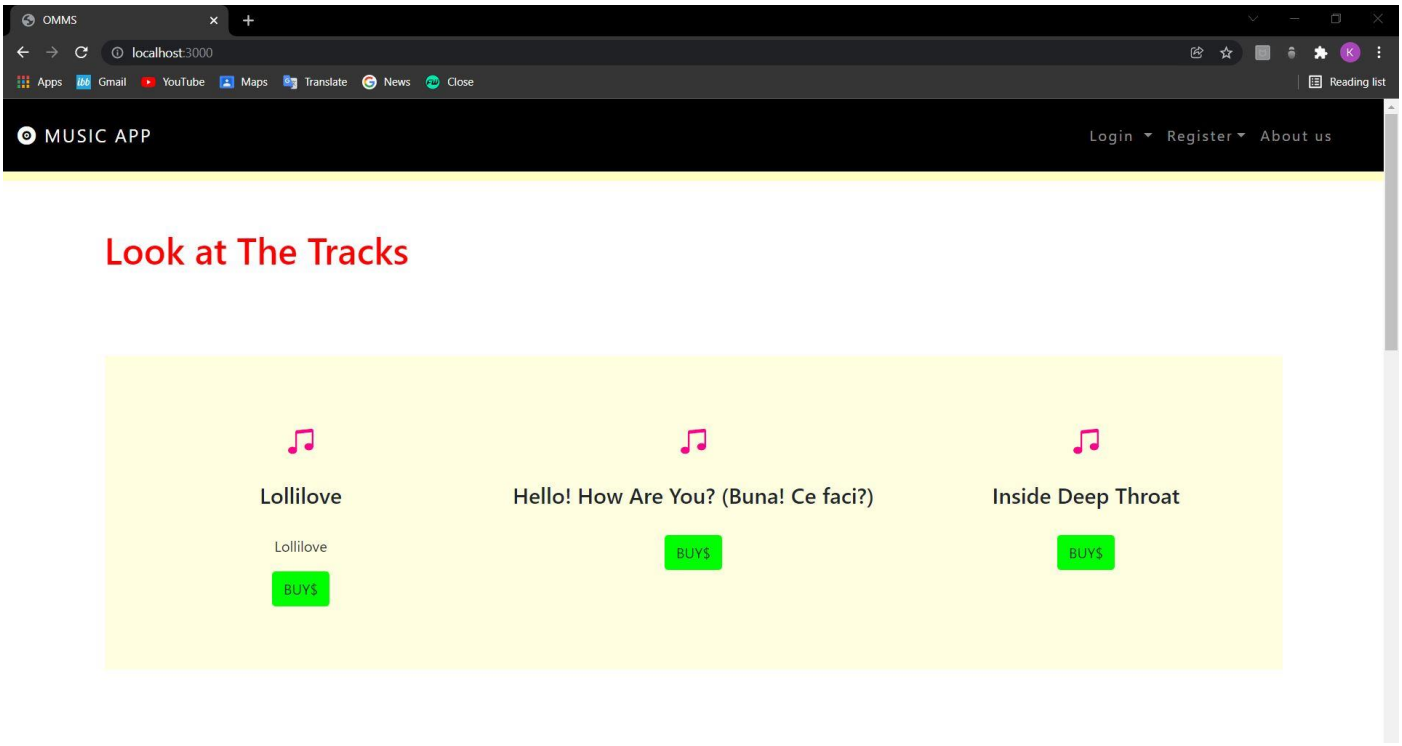
```
      visualLogin: "visually-hidden",
      visualCart: "visually-hidden",
      CustomerId: id
    });
  });
 } catch (err) {
  console.log(err);
  res.redirect("/");
 }

});



app.get("/wishlist/:Customer_Id", (req, res) => {

 try {
  pool.connect(async (err, client, release) => {

    const id = _.lowerCase(req.params.Customer_Id);

    await client.query('SET SEARCH_PATH TO "OMMS"');
    const resp = await client.query(`SELECT "Track"."trackName", "amount" FROM
          "Wishlist" INNER JOIN "Track" ON "Track"."trackId" = "Wishlist"."trackId"
          INNER JOIN "Customer" ON "Wishlist"."customerId" = "Customer"."customerId"
          GROUP BY "Customer"."customerId", "Track"."trackName", "amount"
          HAVING "Customer"."customerId" = $1`, [id]);

    res.render("wishlist", {
      wishlistData: resp.rows,
      visualLogin: "visually-hidden",
      visualCart: "visually-hidden",
      CustomerId: id
    });
  });
 } catch (err) {
```

```
    console.log(err);
    res.redirect("/");
  }


});



app.listen(3000, function() {
  console.log('Server is running on port 3000');
});
```

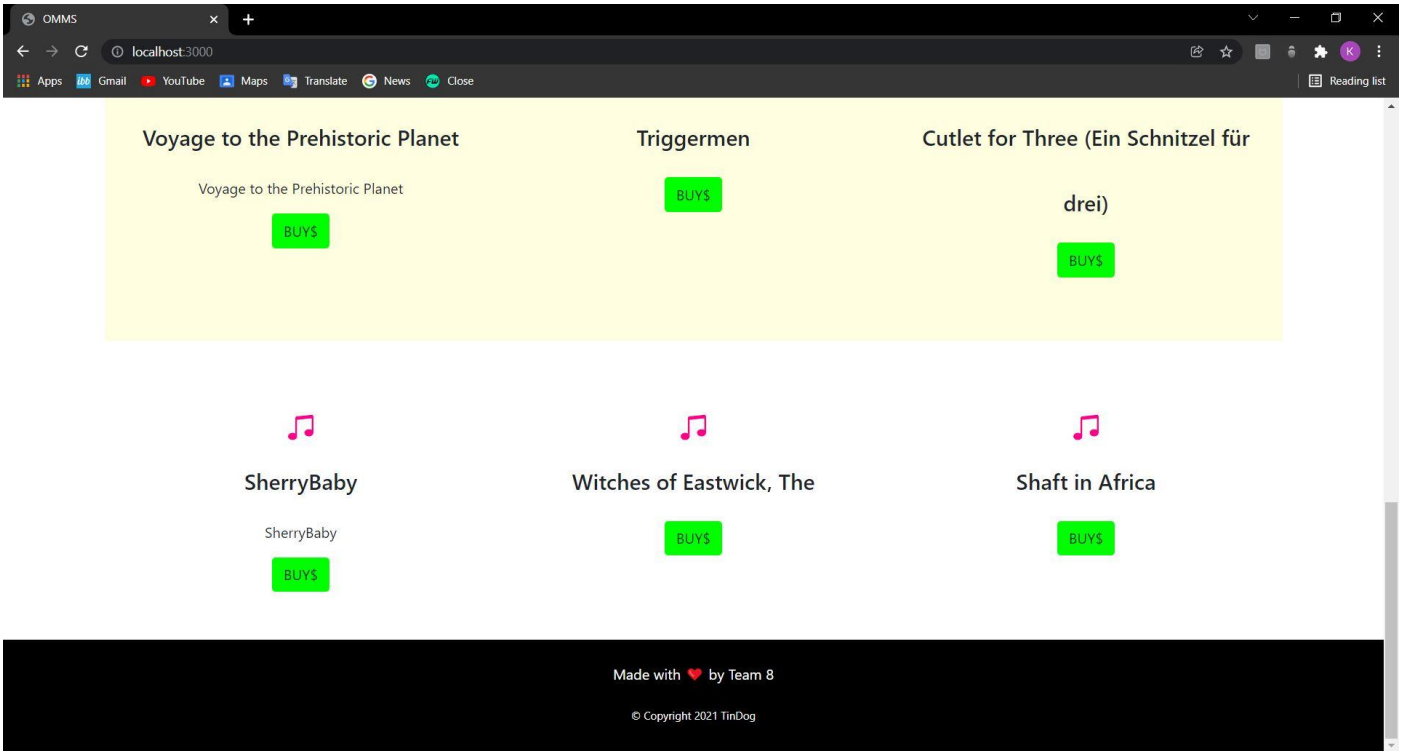
GITHUB link for complete Front-End code.

https://github.com/DesipherMe/Music-System/tree/master

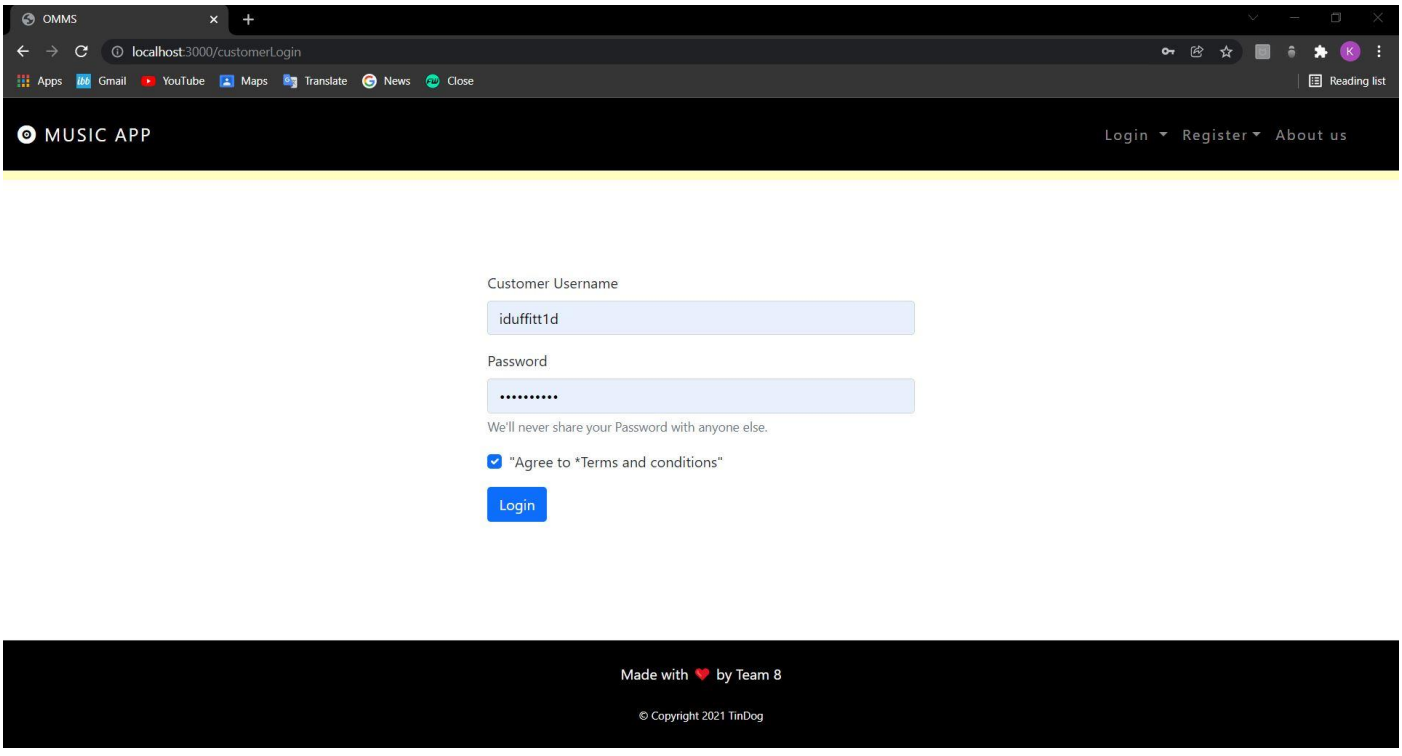
Home Page



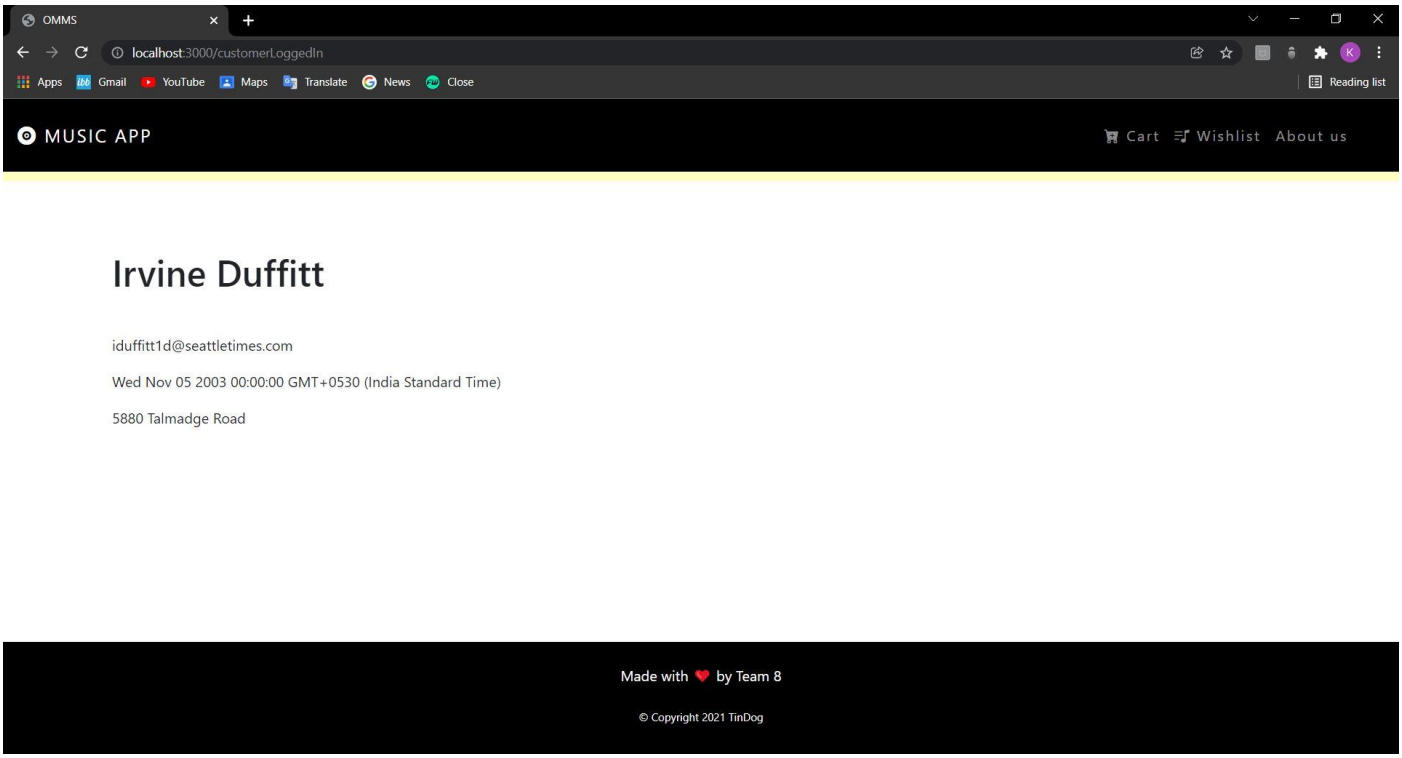
Footer View



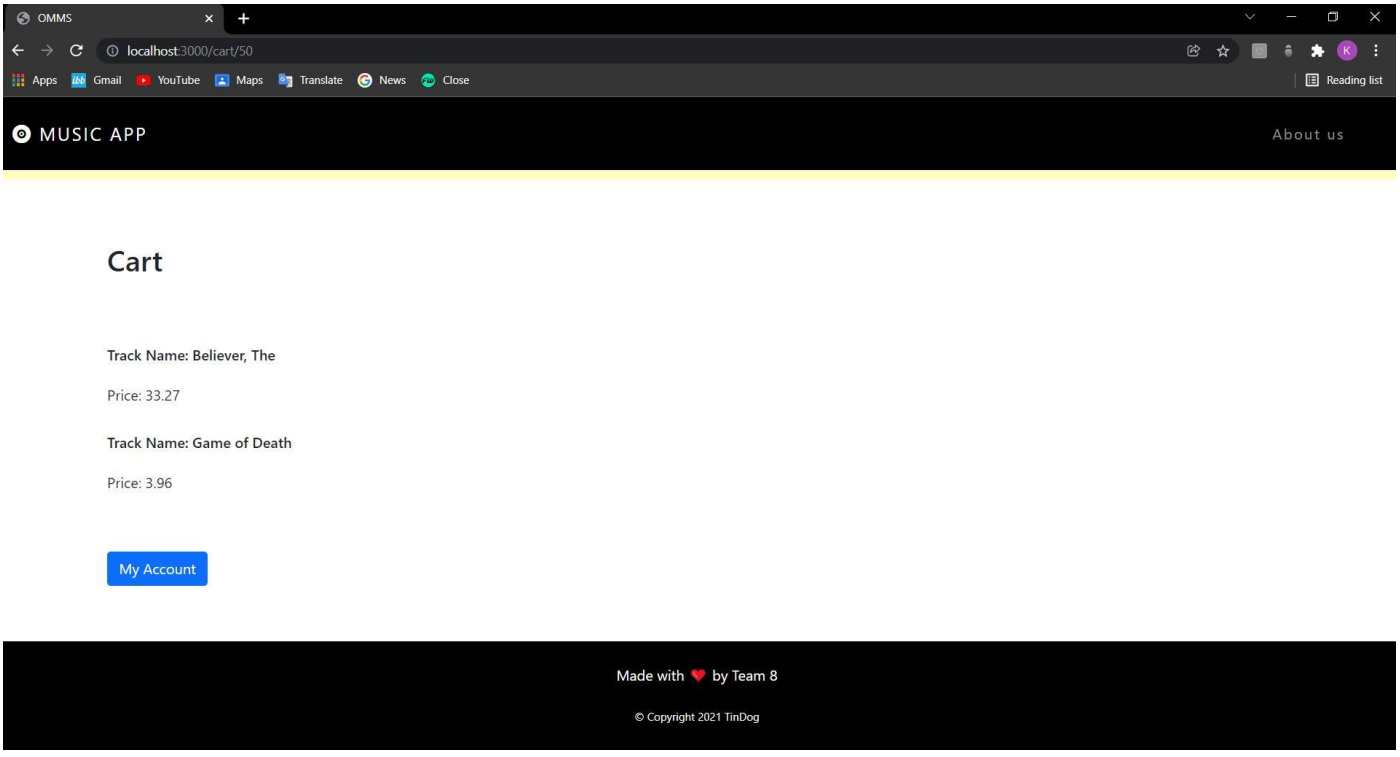
Customer Login



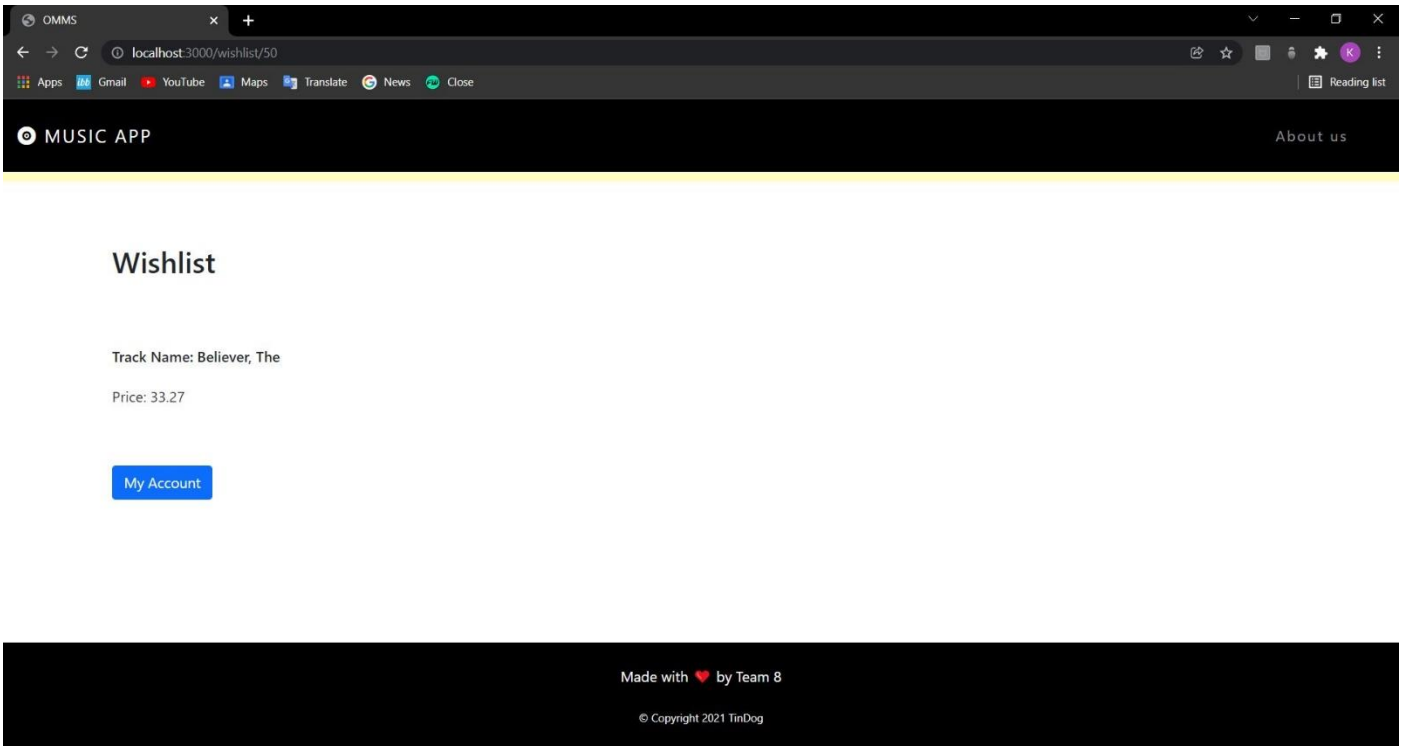
Customer Info



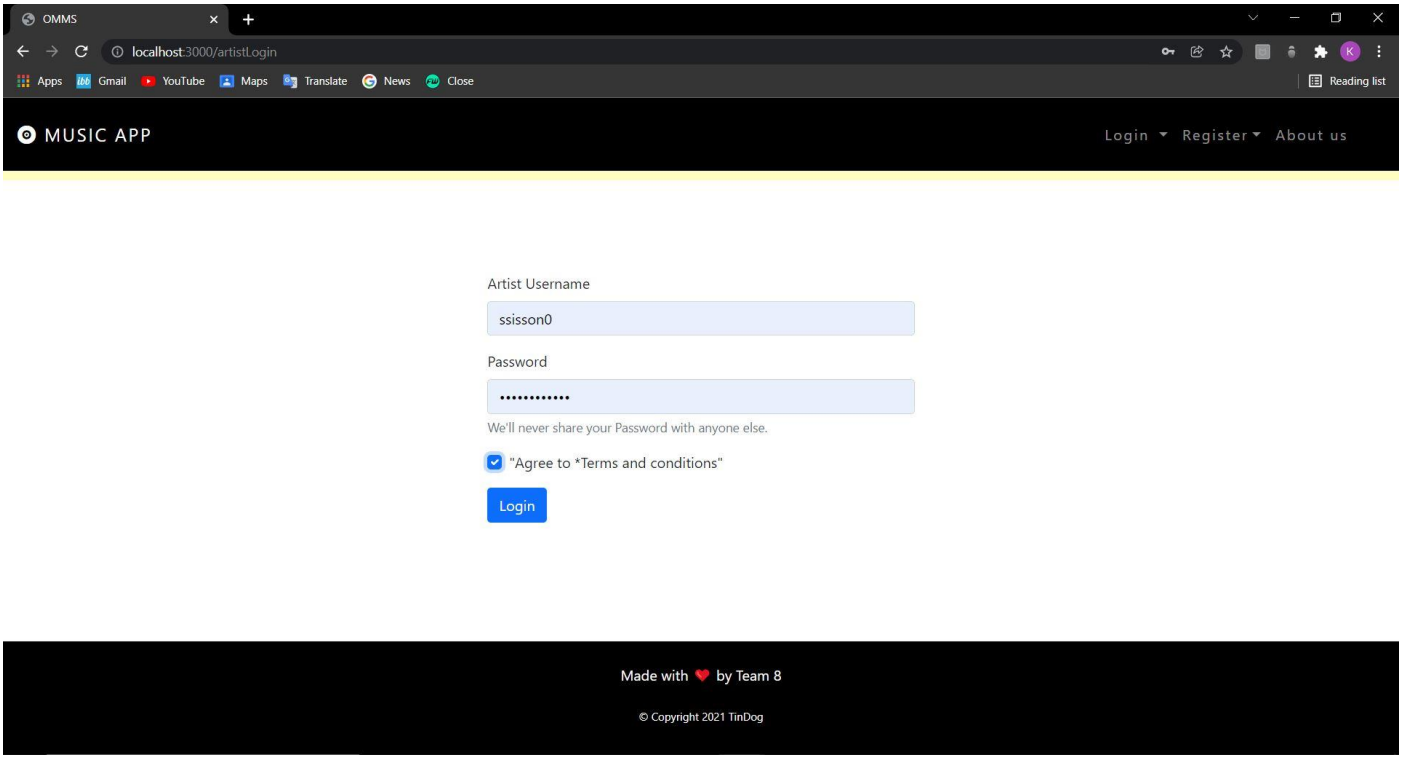
Customer's Cart



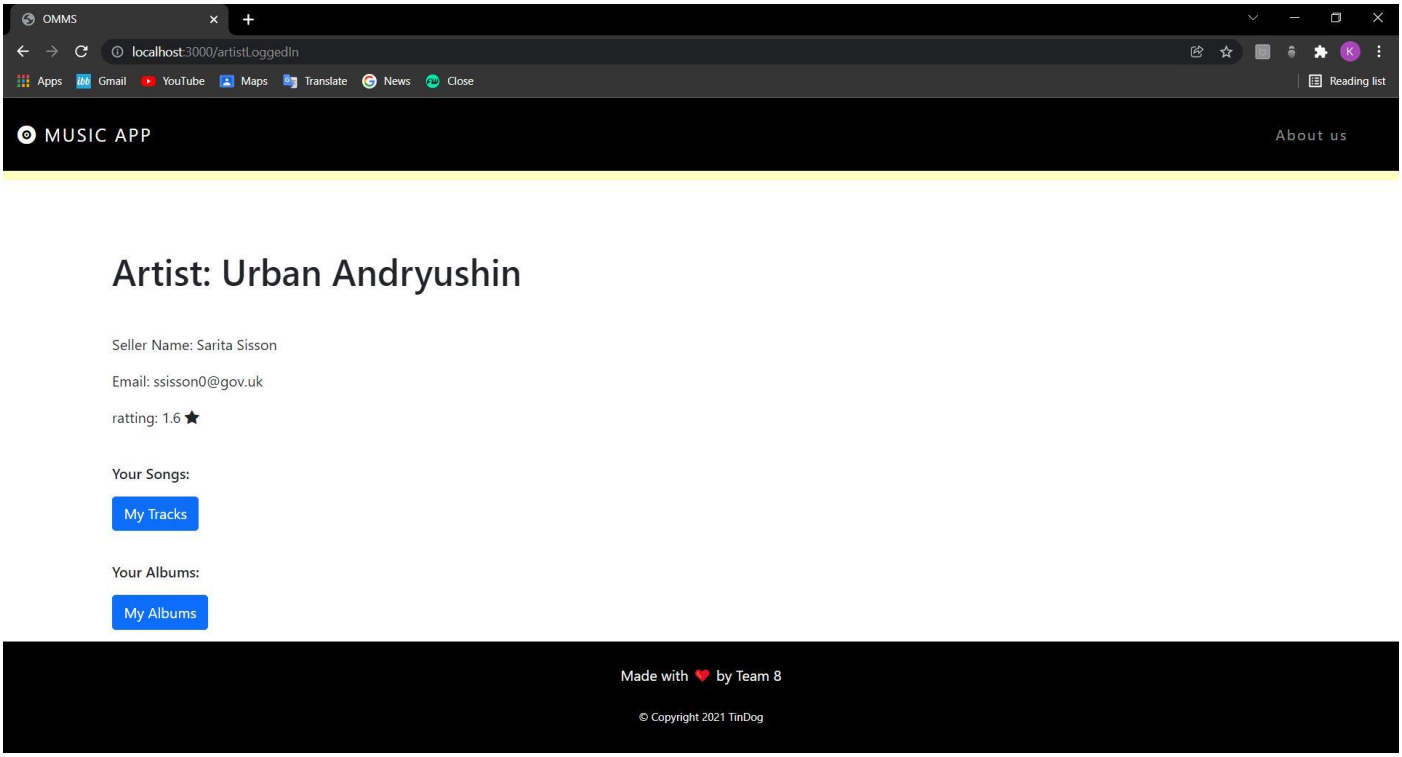
Customer’s Wishlist



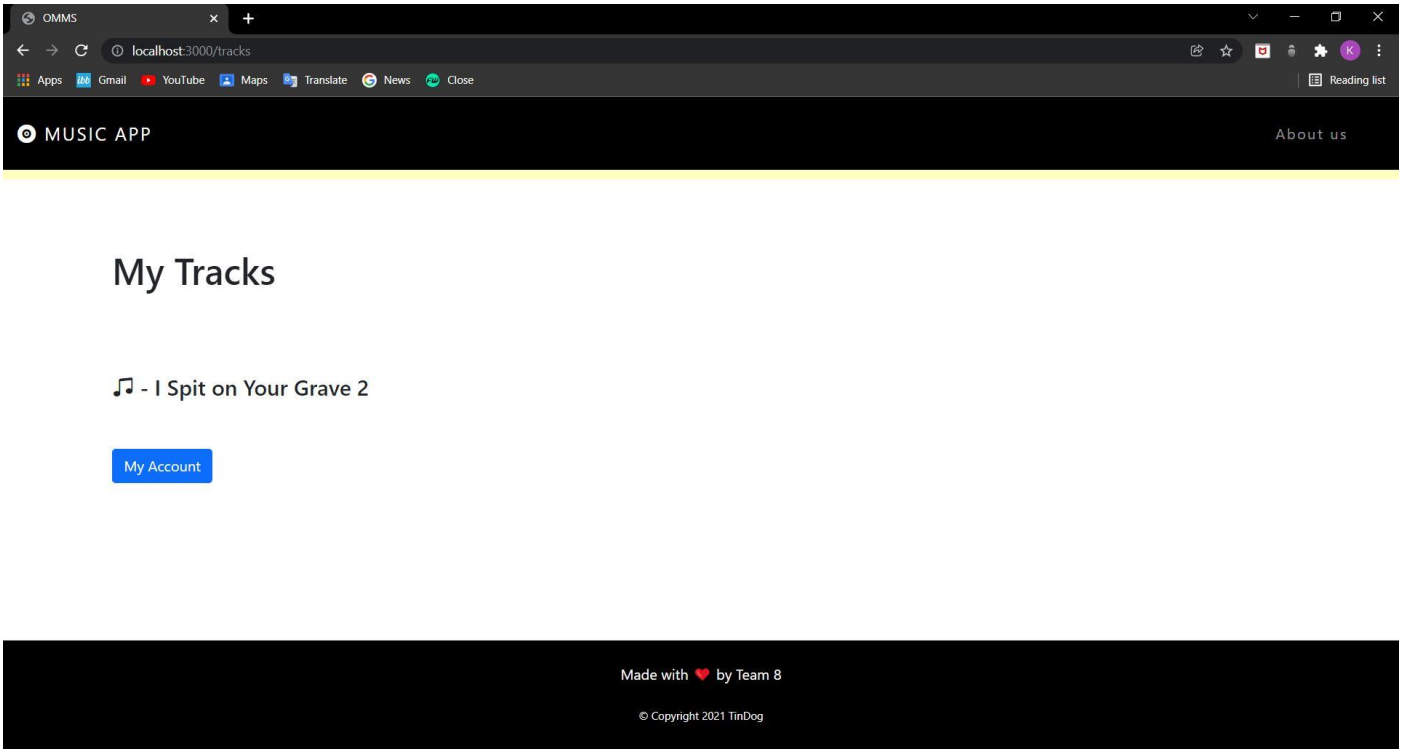
Artist Login



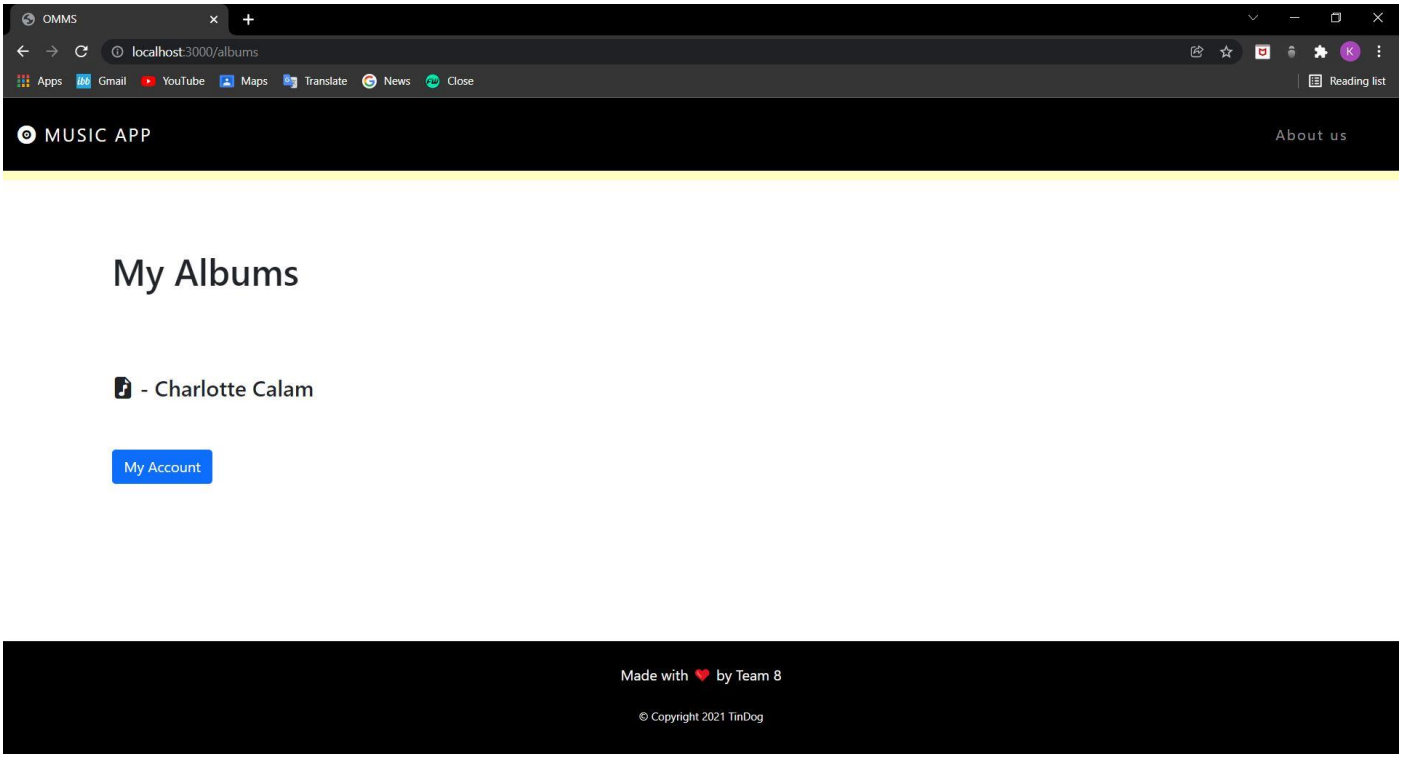
Artist Info



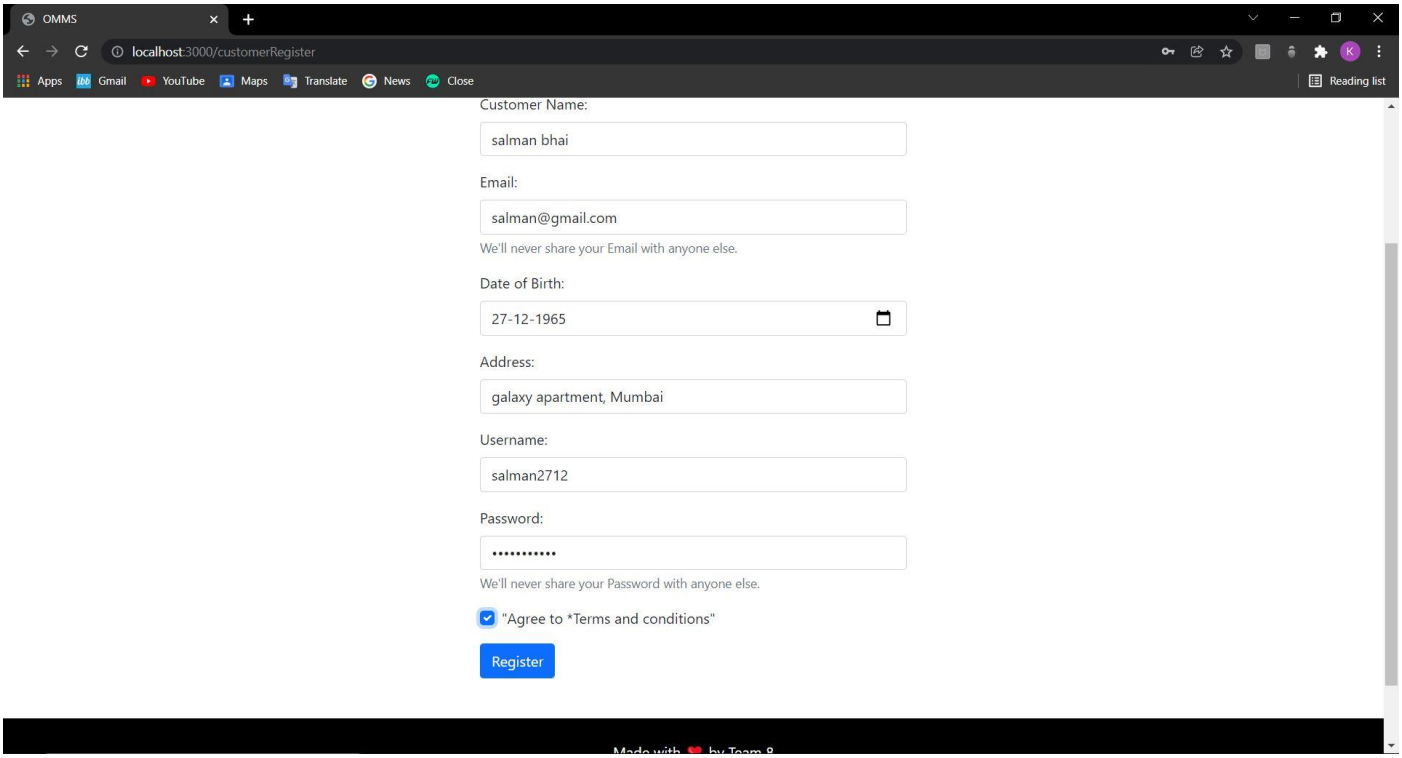
Artist's Tracks



Artist's Albums

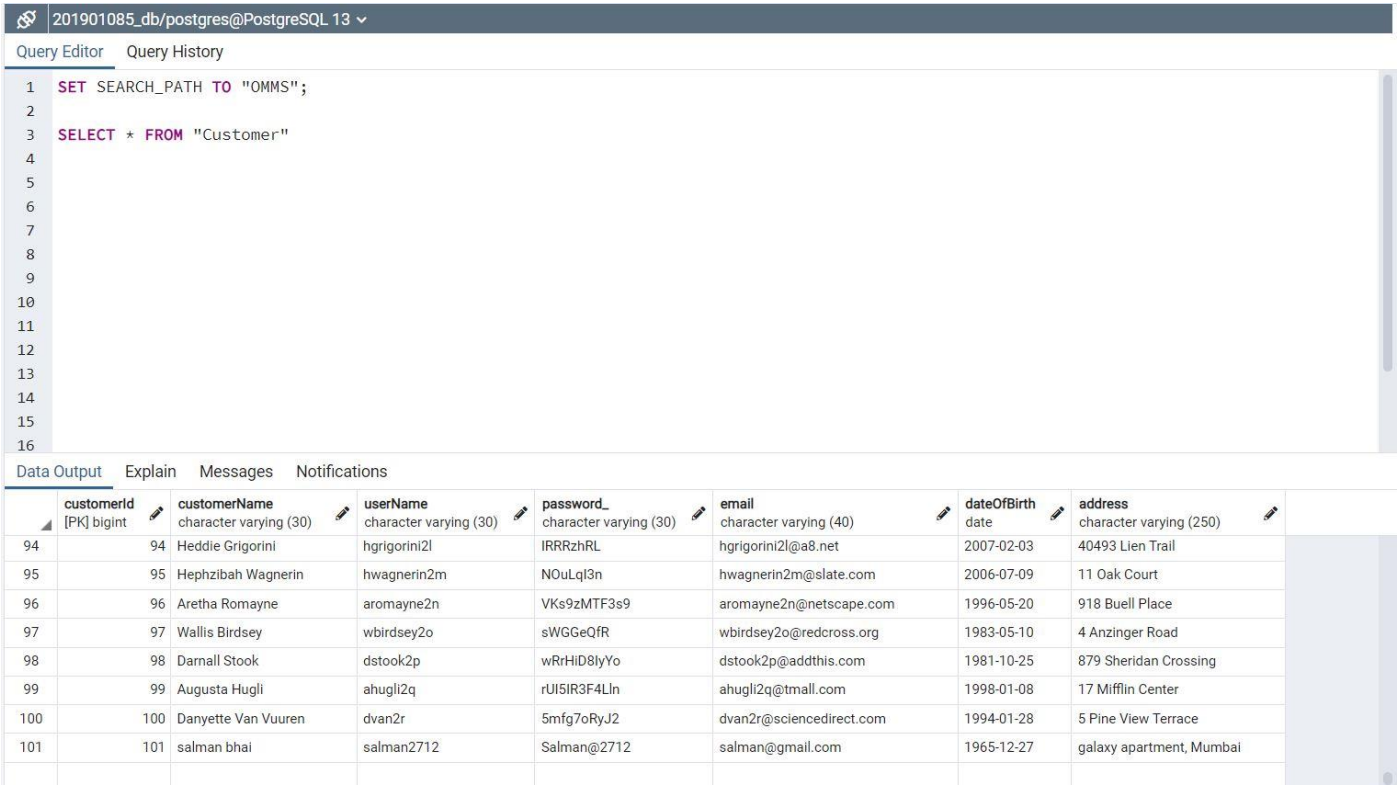


Customer Registration



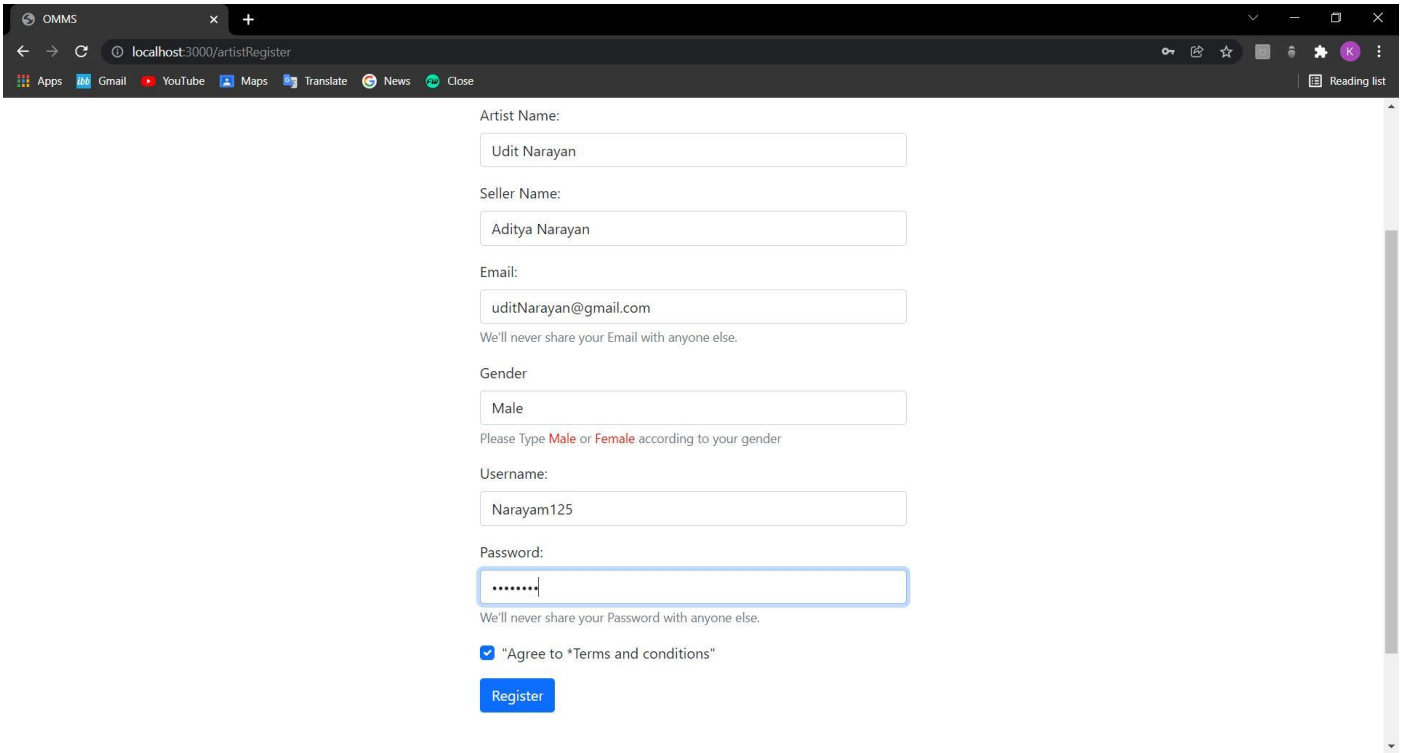
New Customer Inserted into SQL database

you can look at customerID = 101



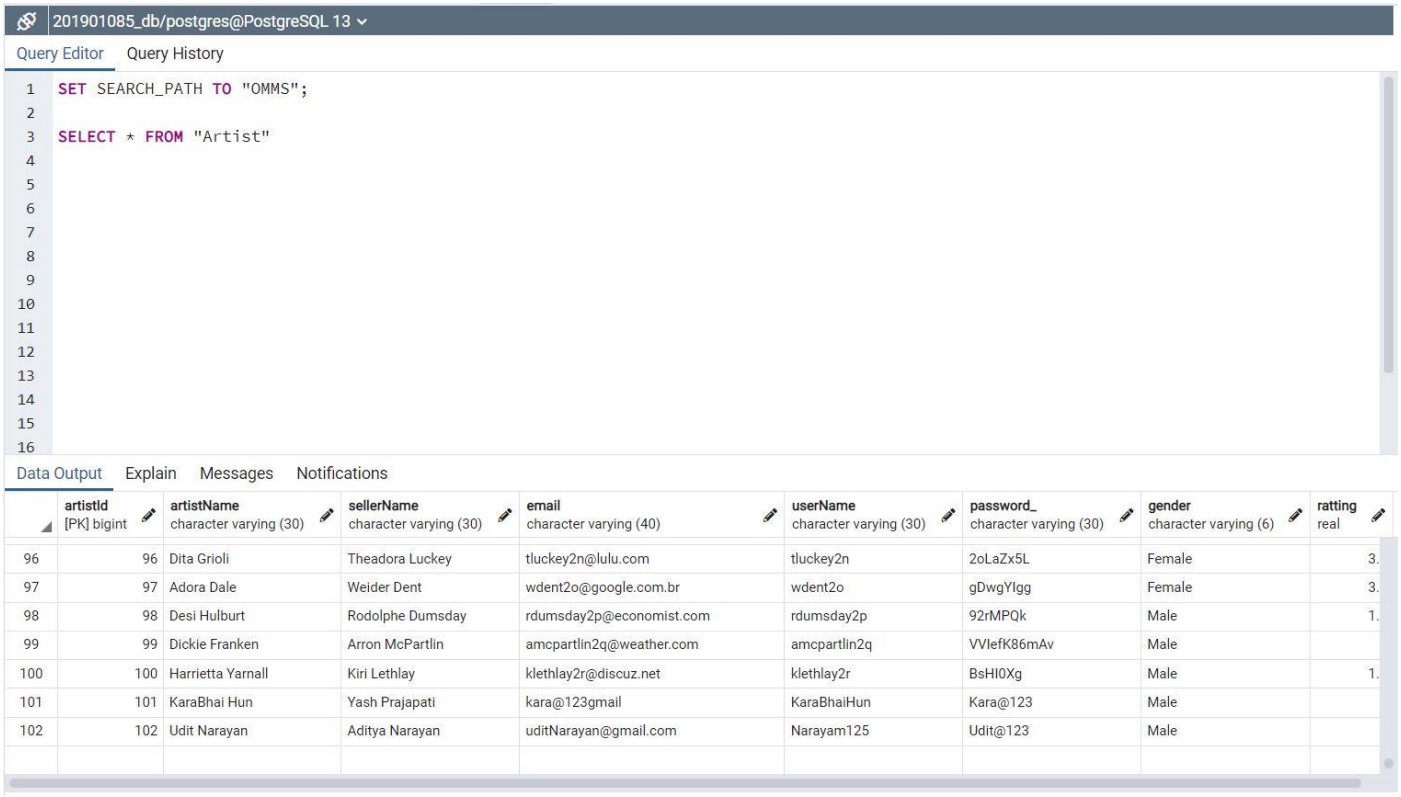
	customerId [PK] bigint	customerName character varying (30)	userName character varying (30)	password_ character varying (30)	email character varying (40)	dateOfBirth date	address character varying (250)
94	94	Heddie Grigorini	hgrigorini2l	IRRRzhRL	hgrigorini2l@a8.net	2007-02-03	40493 Lien Trail
95	95	Hephzibah Wagnerin	hwagnerin2m	NOuLqI3n	hwagnerin2m@slate.com	2006-07-09	11 Oak Court
96	96	Aretha Romayne	aromayne2n	VKs9zMTF3s9	aromayne2n@netscape.com	1996-05-20	918 Buell Place
97	97	Wallis Birdsey	wbirdsey2o	sWGGeQfR	wbirdsey2o@redcross.org	1983-05-10	4 Anzinger Road
98	98	Darnall Stook	dstook2p	wRrHiD8lyYo	dstook2p@addthis.com	1981-10-25	879 Sheridan Crossing
99	99	Augusta Hugli	ahugli2q	rUI5IR3F4Lln	ahugli2q@tmall.com	1998-01-08	17 Mifflin Center
100	100	Danyette Van Vuuren	dvan2r	5mfg7oRyJ2	dvan2r@sciencedirect.com	1994-01-28	5 Pine View Terrace
101	101	salman bhai	salman2712	Salman@2712	salman@gmail.com	1965-12-27	galaxy apartment, Mumbai

Artist Registration



New Artist Inserted into SQL database

you can look at artistID = 102



	artistId [PK] bigint	artistName character varying (30)	sellerName character varying (30)	email character varying (40)	userName character varying (30)	password_ character varying (30)	gender character varying (6)	ratting real
96	96	Dita Grioli	Theadora Luckey	tluckey2n@lulu.com	tluckey2n	2oLaZx5L	Female	3.
97	97	Adora Dale	Weider Dent	wdent2o@google.com.br	wdent2o	gDwgYIgg	Female	3.
98	98	Desi Hulburt	Rodolphe Dumsday	rdumsday2p@economist.com	rdumsday2p	92rMPQk	Male	1.
99	99	Dickie Franken	Arron McPartlin	amcpartlin2q@weather.com	amcpartlin2q	VVIefK86mAv	Male	
100	100	Harrietta Yarnall	Kiri Lethlay	klethlay2r@discuz.net	klethlay2r	BsHI0Xg	Male	1.
101	101	KaraBhai Hun	Yash Prajapati	kara@123gmail	KaraBhaiHun	Kara@123	Male	
102	102	Udit Narayan	Aditya Narayan	uditNarayan@gmail.com	Narayam125	Udit@123	Male	