Chinook Database Sample text to SQL

You are an expert at Generative AI, Python and SQL dialect for PostgreSQL

I am testing a text-to-sql application on the popular Chinook database, which I have ported to PostgresSQL. Can you generate 100 examples of text queries that will generate SQL of progressively increasing complexity. Dont generate similar examples (for example, show me the count of actors and show me the count of albums), examples should progress from text that will generate single table SQL to those that join 2, 3, 4 tables and use aggregation functions, sub-queries, outer joins etc.

Your response should be as follows:

text that user will type in

SQL that will be generated for the same

2 blank lines

1. List all artists.  
   SELECT \* FROM Artist;
2. Show all albums.  
   SELECT \* FROM Album;
3. Display all tracks.  
   SELECT \* FROM Track;
4. List all genres.  
   SELECT \* FROM Genre;
5. Show all media types.  
   SELECT \* FROM MediaType;
6. List all playlists.  
   SELECT \* FROM Playlist;
7. List all customers.  
   SELECT \* FROM Customer;
8. List all employees.  
   SELECT \* FROM Employee;
9. List all invoices.  
   SELECT \* FROM Invoice;
10. List all invoice lines.  
    SELECT \* FROM InvoiceLine;
11. List all playlist tracks.  
    SELECT \* FROM PlaylistTrack;
12. Show artists whose name starts with "A".  
    SELECT \* FROM Artist WHERE Name ILIKE 'A%';
13. List albums with titles containing "Hits".  
    SELECT \* FROM Album WHERE Title ILIKE '%Hits%';
14. List tracks longer than 5 minutes.  
    SELECT \* FROM Track WHERE Milliseconds > (5 \* 60 \* 1000);
15. Display invoices with a total greater than 10 dollars.  
    SELECT \* FROM Invoice WHERE Total > 10;
16. Show customers from Canada.  
    SELECT \* FROM Customer WHERE Country = 'Canada';
17. List employees hired after 2005.  
    SELECT \* FROM Employee WHERE HireDate > '2005-01-01';
18. Display playlists that include the word "Workout".  
    SELECT \* FROM Playlist WHERE Name ILIKE '%Workout%';
19. List media types that are "MPEG audio file".  
    SELECT \* FROM MediaType WHERE Name = 'MPEG audio file';
20. Show genres with non-null names.  
    SELECT \* FROM Genre WHERE Name IS NOT NULL;
21. List artists in alphabetical order.  
    SELECT \* FROM Artist ORDER BY Name;
22. Show albums ordered by title descending.  
    SELECT \* FROM Album ORDER BY Title DESC;
23. Display tracks sorted by unit price.  
    SELECT \* FROM Track ORDER BY UnitPrice;
24. List customers ordered by last name.  
    SELECT \* FROM Customer ORDER BY LastName;
25. Show employees ordered by hire date.  
    SELECT \* FROM Employee ORDER BY HireDate;
26. List the first 5 artists.  
    SELECT \* FROM Artist LIMIT 5;
27. Display the top 10 most expensive tracks.  
    SELECT \* FROM Track ORDER BY UnitPrice DESC LIMIT 10;
28. Show the 3 most recent invoices.  
    SELECT \* FROM Invoice ORDER BY InvoiceDate DESC LIMIT 3;
29. List albums with titles containing "love".  
    SELECT \* FROM Album WHERE Title ILIKE '%love%';
30. Show customers with email ending in "gmail.com".  
    SELECT \* FROM Customer WHERE Email ILIKE '%@gmail.com';
31. Count the number of artists.  
    SELECT COUNT(\*) FROM Artist;
32. Find the maximum invoice total.  
    SELECT MAX(Total) FROM Invoice;
33. Calculate the average track length in milliseconds.  
    SELECT AVG(Milliseconds) FROM Track;
34. Sum the total amount of all invoices.  
    SELECT SUM(Total) FROM Invoice;
35. Find the minimum unit price of tracks.  
    SELECT MIN(UnitPrice) FROM Track;
36. Count tracks for each media type.  
    SELECT MediaTypeId, COUNT(\*) FROM Track GROUP BY MediaTypeId;
37. List the number of invoices per country.  
    SELECT Country, COUNT(\*) FROM Invoice GROUP BY Country;
38. Show average invoice total per customer.  
    SELECT CustomerId, AVG(Total) FROM Invoice GROUP BY CustomerId;
39. Find total sales per customer from invoices.  
    SELECT CustomerId, SUM(Total) FROM Invoice GROUP BY CustomerId;
40. Count albums by artist.  
    SELECT ArtistId, COUNT(\*) FROM Album GROUP BY ArtistId;
41. List album titles with their artist names.  
    SELECT Album.Title, Artist.Name FROM Album JOIN Artist ON Album.ArtistId = Artist.ArtistId;
42. Show track names along with album titles.  
    SELECT Track.Name, Album.Title FROM Track JOIN Album ON Track.AlbumId = Album.AlbumId;
43. List invoice IDs and corresponding customer names.  
    SELECT Invoice.InvoiceId, Customer.FirstName, Customer.LastName FROM Invoice JOIN Customer ON Invoice.CustomerId = Customer.CustomerId;
44. Show invoice line details with track names.  
    SELECT InvoiceLine.InvoiceLineId, Track.Name FROM InvoiceLine JOIN Track ON InvoiceLine.TrackId = Track.TrackId;
45. Show track names with their genre.  
    SELECT Track.Name, Genre.Name AS GenreName FROM Track JOIN Genre ON Track.GenreId = Genre.GenreId;
46. List customer names with their support rep's name.  
    SELECT Customer.FirstName, Customer.LastName, Employee.FirstName AS SupportFirstName, Employee.LastName AS SupportLastName  
    FROM Customer JOIN Employee ON Customer.SupportRepId = Employee.EmployeeId;
47. List albums by the artist "Queen".  
    SELECT Album.Title FROM Album JOIN Artist ON Album.ArtistId = Artist.ArtistId WHERE Artist.Name = 'Queen';
48. Show tracks from the album "For Those About To Rock".  
    SELECT Track.Name FROM Track JOIN Album ON Track.AlbumId = Album.AlbumId  
    WHERE Album.Title ILIKE '%For Those About To Rock%';
49. List invoices with customer country as "USA".  
    SELECT Invoice.InvoiceId, Customer.Country FROM Invoice JOIN Customer ON Invoice.CustomerId = Customer.CustomerId  
    WHERE Customer.Country = 'USA';
50. Display tracks with genre "Rock".  
    SELECT Track.Name FROM Track JOIN Genre ON Track.GenreId = Genre.GenreId WHERE Genre.Name = 'Rock';
51. Show the top 5 most expensive tracks with their media type.  
    SELECT Track.Name, MediaType.Name FROM Track  
    JOIN MediaType ON Track.MediaTypeId = MediaType.MediaTypeId  
    ORDER BY Track.UnitPrice DESC LIMIT 5;
52. Count tracks per album.  
    SELECT Album.Title, COUNT(Track.TrackId) AS TrackCount  
    FROM Album JOIN Track ON Album.AlbumId = Track.AlbumId  
    GROUP BY Album.Title;
53. Find total invoice amount per customer.  
    SELECT Customer.CustomerId, SUM(Invoice.Total) AS TotalSpent  
    FROM Invoice JOIN Customer ON Invoice.CustomerId = Customer.CustomerId  
    GROUP BY Customer.CustomerId;
54. Calculate average track price for each media type.  
    SELECT MediaType.Name, AVG(Track.UnitPrice) AS AvgPrice  
    FROM Track JOIN MediaType ON Track.MediaTypeId = MediaType.MediaTypeId  
    GROUP BY MediaType.Name;
55. List track names with album titles and artist names.  
    SELECT Track.Name AS TrackName, Album.Title AS AlbumTitle, Artist.Name AS ArtistName  
    FROM Track JOIN Album ON Track.AlbumId = Album.AlbumId  
    JOIN Artist ON Album.ArtistId = Artist.ArtistId;
56. Show invoice details with customer name and support rep name.  
    SELECT Invoice.InvoiceId,  
    Customer.FirstName || ' ' || Customer.LastName AS CustomerName,  
    Employee.FirstName || ' ' || Employee.LastName AS SupportRepName  
    FROM Invoice  
    JOIN Customer ON Invoice.CustomerId = Customer.CustomerId  
    JOIN Employee ON Customer.SupportRepId = Employee.EmployeeId;
57. List tracks with album title and genre.  
    SELECT Track.Name, Album.Title, Genre.Name AS GenreName  
    FROM Track  
    JOIN Album ON Track.AlbumId = Album.AlbumId  
    JOIN Genre ON Track.GenreId = Genre.GenreId;
58. Show customer email with invoice total and invoice date.  
    SELECT Customer.Email, Invoice.Total, Invoice.InvoiceDate  
    FROM Invoice JOIN Customer ON Invoice.CustomerId = Customer.CustomerId;
59. Display track name with its media type and genre.  
    SELECT Track.Name, MediaType.Name AS MediaType, Genre.Name AS Genre  
    FROM Track  
    JOIN MediaType ON Track.MediaTypeId = MediaType.MediaTypeId  
    JOIN Genre ON Track.GenreId = Genre.GenreId;
60. List playlist name, track name, album title, and artist name.  
    SELECT Playlist.Name AS PlaylistName, Track.Name AS TrackName, Album.Title AS AlbumTitle, Artist.Name AS ArtistName  
    FROM PlaylistTrack  
    JOIN Playlist ON PlaylistTrack.PlaylistId = Playlist.PlaylistId  
    JOIN Track ON PlaylistTrack.TrackId = Track.TrackId  
    JOIN Album ON Track.AlbumId = Album.AlbumId  
    JOIN Artist ON Album.ArtistId = Artist.ArtistId;
61. Show invoice details including invoice line quantity and track unit price.  
    SELECT Invoice.InvoiceId, InvoiceLine.Quantity, Track.UnitPrice  
    FROM Invoice  
    JOIN InvoiceLine ON Invoice.InvoiceId = InvoiceLine.InvoiceId  
    JOIN Track ON InvoiceLine.TrackId = Track.TrackId;
62. List customers with their invoices and invoice lines.  
    SELECT Customer.CustomerId, Invoice.InvoiceId, InvoiceLine.InvoiceLineId  
    FROM Customer  
    JOIN Invoice ON Customer.CustomerId = Invoice.CustomerId  
    JOIN InvoiceLine ON Invoice.InvoiceId = InvoiceLine.InvoiceId;
63. Display employees with the customers they support and corresponding invoice totals.  
    SELECT Employee.EmployeeId, Employee.FirstName || ' ' || Employee.LastName AS EmployeeName,  
    Customer.CustomerId, Invoice.Total  
    FROM Employee  
    JOIN Customer ON Employee.EmployeeId = Customer.SupportRepId  
    JOIN Invoice ON Customer.CustomerId = Invoice.CustomerId;
64. Show track details with album title, artist name, and genre.  
    SELECT Track.\*, Album.Title AS AlbumTitle, Artist.Name AS ArtistName, Genre.Name AS GenreName  
    FROM Track  
    JOIN Album ON Track.AlbumId = Album.AlbumId  
    JOIN Artist ON Album.ArtistId = Artist.ArtistId  
    JOIN Genre ON Track.GenreId = Genre.GenreId;
65. List customers who have made a purchase over $100 in a single invoice.  
    SELECT \* FROM Customer  
    WHERE CustomerId IN (SELECT CustomerId FROM Invoice WHERE Total > 100);
66. Show albums that have more than 10 tracks.  
    SELECT \* FROM Album  
    WHERE AlbumId IN (SELECT AlbumId FROM Track GROUP BY AlbumId HAVING COUNT(\*) > 10);
67. Display tracks with a unit price above the average unit price.  
    SELECT \* FROM Track  
    WHERE UnitPrice > (SELECT AVG(UnitPrice) FROM Track);
68. List customers whose total spending exceeds the average spending.  
    SELECT \* FROM Customer  
    WHERE CustomerId IN (  
    SELECT CustomerId FROM Invoice  
    GROUP BY CustomerId  
    HAVING SUM(Total) > (SELECT AVG(Total) FROM Invoice)  
    );
69. Show invoices with a total higher than the average invoice total.  
    SELECT \* FROM Invoice  
    WHERE Total > (SELECT AVG(Total) FROM Invoice);
70. For each album, show its title and track count if above the overall average track count.  
    SELECT Title,  
    (SELECT COUNT(*) FROM Track WHERE Track.AlbumId = Album.AlbumId) AS TrackCount  
    FROM Album  
    WHERE (SELECT COUNT(*) FROM Track WHERE Track.AlbumId = Album.AlbumId) >  
    (SELECT AVG(TrackCount) FROM (SELECT COUNT(\*) AS TrackCount FROM Track GROUP BY AlbumId) AS sub);
71. List artists whose albums have an average track unit price above $0.5.  
    SELECT Artist.Name  
    FROM Artist  
    WHERE Artist.ArtistId IN (  
    SELECT Album.ArtistId  
    FROM Album  
    JOIN Track ON Album.AlbumId = Track.AlbumId  
    GROUP BY Album.ArtistId  
    HAVING AVG(Track.UnitPrice) > 0.5  
    );
72. Display the customer whose last invoice total is the highest.  
    SELECT \* FROM Customer  
    WHERE CustomerId = (SELECT CustomerId FROM Invoice ORDER BY Total DESC LIMIT 1);
73. List all artists and their albums, including artists without albums.  
    SELECT Artist.Name, Album.Title  
    FROM Artist  
    LEFT JOIN Album ON Artist.ArtistId = Album.ArtistId;
74. Show all albums and their tracks, including albums with no tracks.  
    SELECT Album.Title, Track.Name  
    FROM Album  
    LEFT JOIN Track ON Album.AlbumId = Track.AlbumId;
75. List all customers and their invoices, including customers with no invoices.  
    SELECT Customer.FirstName, Customer.LastName, Invoice.InvoiceId  
    FROM Customer  
    LEFT JOIN Invoice ON Customer.CustomerId = Invoice.CustomerId;
76. Display all employees and the customers they support, even if some employees support no customers.  
    SELECT Employee.FirstName, Employee.LastName, Customer.CustomerId  
    FROM Employee  
    LEFT JOIN Customer ON Employee.EmployeeId = Customer.SupportRepId;
77. List all tracks and their genres, including tracks without a genre.  
    SELECT Track.Name, Genre.Name  
    FROM Track  
    LEFT JOIN Genre ON Track.GenreId = Genre.GenreId;
78. Show total invoice amount and count of invoices per country.  
    SELECT Country, SUM(Total) AS TotalSales, COUNT(\*) AS InvoiceCount  
    FROM Invoice  
    GROUP BY Country;
79. List album titles with the total number of tracks and average track length.  
    SELECT Album.Title, COUNT(Track.TrackId) AS TrackCount, AVG(Track.Milliseconds) AS AvgLength  
    FROM Album  
    JOIN Track ON Album.AlbumId = Track.AlbumId  
    GROUP BY Album.Title;
80. Display customer spending with total invoices and average invoice total.  
    SELECT Customer.CustomerId, SUM(Invoice.Total) AS TotalSpent,  
    COUNT(Invoice.InvoiceId) AS InvoiceCount, AVG(Invoice.Total) AS AvgInvoice  
    FROM Customer  
    JOIN Invoice ON Customer.CustomerId = Invoice.CustomerId  
    GROUP BY Customer.CustomerId;
81. List tracks with their unit price and the average unit price over all tracks.  
    SELECT Name, UnitPrice, AVG(UnitPrice) OVER () AS AvgUnitPrice  
    FROM Track;
82. Show invoices with their total and a running sum of totals.  
    SELECT InvoiceId, Total, SUM(Total) OVER (ORDER BY InvoiceDate) AS RunningTotal  
    FROM Invoice;
83. Display customers with their invoice totals and rank by total spending.  
    SELECT Customer.CustomerId, SUM(Invoice.Total) AS TotalSpent,  
    RANK() OVER (ORDER BY SUM(Invoice.Total) DESC) AS SpendingRank  
    FROM Invoice  
    JOIN Customer ON Invoice.CustomerId = Customer.CustomerId  
    GROUP BY Customer.CustomerId;
84. List tracks with their unit price and a percentile rank among all tracks.  
    SELECT Name, UnitPrice, PERCENT\_RANK() OVER (ORDER BY UnitPrice) AS PricePercentile  
    FROM Track;
85. Show employee hire dates along with row numbers ordered by hire date.  
    SELECT EmployeeId, HireDate, ROW\_NUMBER() OVER (ORDER BY HireDate) AS RowNum  
    FROM Employee;
86. List top 3 customers by total spending using a CTE.  
    WITH CustomerSpending AS (  
    SELECT CustomerId, SUM(Total) AS TotalSpent  
    FROM Invoice  
    GROUP BY CustomerId  
    )  
    SELECT CustomerId, TotalSpent  
    FROM CustomerSpending  
    ORDER BY TotalSpent DESC LIMIT 3;
87. Show albums with more than 5 tracks using a CTE.  
    WITH AlbumTrackCount AS (  
    SELECT AlbumId, COUNT(\*) AS TrackCount  
    FROM Track  
    GROUP BY AlbumId  
    )  
    SELECT AlbumId  
    FROM AlbumTrackCount  
    WHERE TrackCount > 5;
88. Display artists with average track price above the overall average using a CTE.  
    WITH ArtistAvgPrice AS (  
    SELECT Album.ArtistId, AVG(Track.UnitPrice) AS AvgPrice  
    FROM Album  
    JOIN Track ON Album.AlbumId = Track.AlbumId  
    GROUP BY Album.ArtistId  
    )  
    SELECT ArtistId, AvgPrice  
    FROM ArtistAvgPrice  
    WHERE AvgPrice > (SELECT AVG(AvgPrice) FROM ArtistAvgPrice);
89. List employees and the count of customers they support using a CTE.  
    WITH EmployeeCustomerCount AS (  
    SELECT SupportRepId, COUNT(\*) AS CustomerCount  
    FROM Customer  
    GROUP BY SupportRepId  
    )  
    SELECT SupportRepId, CustomerCount  
    FROM EmployeeCustomerCount;
90. Show invoice totals and their running total using a CTE.  
    WITH InvoiceCTE AS (  
    SELECT InvoiceId, Total, InvoiceDate  
    FROM Invoice  
    )  
    SELECT InvoiceId, Total, SUM(Total) OVER (ORDER BY InvoiceDate) AS RunningTotal  
    FROM InvoiceCTE;
91. Find albums whose track count equals the maximum track count among all albums.  
    SELECT \* FROM Album  
    WHERE AlbumId IN (  
    SELECT AlbumId FROM Track  
    GROUP BY AlbumId  
    HAVING COUNT(*) = (  
    SELECT MAX(TrackCount) FROM (SELECT COUNT(*) AS TrackCount FROM Track GROUP BY AlbumId) AS sub  
    )  
    );
92. List tracks that do not appear in any playlist.  
    SELECT \* FROM Track  
    WHERE TrackId NOT IN (SELECT TrackId FROM PlaylistTrack);
93. Show customers who haven't made any purchase in 2011.  
    SELECT \* FROM Customer  
    WHERE CustomerId NOT IN (  
    SELECT CustomerId FROM Invoice WHERE EXTRACT(YEAR FROM InvoiceDate) = 2011  
    );
94. Display artists whose albums have no tracks below a unit price of 0.99.  
    SELECT DISTINCT Artist.Name  
    FROM Artist  
    JOIN Album ON Artist.ArtistId = Album.ArtistId  
    WHERE Album.AlbumId NOT IN (  
    SELECT AlbumId FROM Track WHERE UnitPrice < 0.99  
    );
95. List genres that are not associated with any track.  
    SELECT \* FROM Genre  
    WHERE GenreId NOT IN (SELECT GenreId FROM Track WHERE GenreId IS NOT NULL);
96. Show top 3 albums by total track unit price sum using a subquery.  
    SELECT AlbumId,  
    (SELECT SUM(UnitPrice) FROM Track WHERE AlbumId = Album.AlbumId) AS TotalPrice  
    FROM Album  
    ORDER BY TotalPrice DESC LIMIT 3;
97. List customers with their total invoice amount and percentage of overall sales.  
    SELECT Customer.CustomerId, SUM(Invoice.Total) AS CustomerTotal,  
    (SUM(Invoice.Total) / (SELECT SUM(Total) FROM Invoice) \* 100) AS SalesPercentage  
    FROM Invoice  
    JOIN Customer ON Invoice.CustomerId = Customer.CustomerId  
    GROUP BY Customer.CustomerId;
98. Display employees and the number of customers they support compared to the overall average.  
    SELECT Employee.EmployeeId, COUNT(Customer.CustomerId) AS CustomerCount,  
    (COUNT(Customer.CustomerId) - (SELECT AVG(CustomerCount) FROM (  
    SELECT COUNT(\*) AS CustomerCount FROM Customer GROUP BY SupportRepId  
    ) AS sub)) AS DifferenceFromAvg  
    FROM Employee  
    LEFT JOIN Customer ON Employee.EmployeeId = Customer.SupportRepId  
    GROUP BY Employee.EmployeeId;
99. List tracks with their rank by unit price within their album.  
    SELECT TrackId, Name, UnitPrice,  
    RANK() OVER (PARTITION BY AlbumId ORDER BY UnitPrice DESC) AS PriceRank  
    FROM Track;
100. Show invoice lines with a discount column calculated as 10% of (unit price × quantity).  
     SELECT InvoiceLineId, InvoiceId, TrackId, UnitPrice, Quantity,  
     (UnitPrice \* Quantity \* 0.1) AS Discount  
     FROM InvoiceLine;
101. List artists along with the number of albums and total tracks.  
     SELECT Artist.Name, COUNT(DISTINCT Album.AlbumId) AS AlbumCount,  
     (SELECT COUNT(\*) FROM Track JOIN Album A ON Track.AlbumId = A.AlbumId WHERE A.ArtistId = Artist.ArtistId) AS TrackCount  
     FROM Artist  
     LEFT JOIN Album ON Artist.ArtistId = Album.ArtistId  
     GROUP BY Artist.Name;
102. Show invoice totals and the number of tracks sold per invoice.  
     SELECT Invoice.InvoiceId, Invoice.Total,  
     (SELECT SUM(InvoiceLine.Quantity) FROM InvoiceLine WHERE InvoiceLine.InvoiceId = Invoice.InvoiceId) AS TotalTracks  
     FROM Invoice;
103. Display playlist names with count of tracks and total duration (in milliseconds).  
     SELECT Playlist.Name, COUNT(PlaylistTrack.TrackId) AS TrackCount,  
     (SELECT SUM(Track.Milliseconds) FROM Track JOIN PlaylistTrack ON Track.TrackId = PlaylistTrack.TrackId  
     WHERE PlaylistTrack.PlaylistId = Playlist.PlaylistId) AS TotalDuration  
     FROM Playlist  
     LEFT JOIN PlaylistTrack ON Playlist.PlaylistId = PlaylistTrack.PlaylistId  
     GROUP BY Playlist.Name;
104. List albums with artist name and the highest track unit price in each album.  
     SELECT Album.Title, Artist.Name,  
     (SELECT MAX(UnitPrice) FROM Track WHERE AlbumId = Album.AlbumId) AS MaxTrackPrice  
     FROM Album  
     JOIN Artist ON Album.ArtistId = Artist.ArtistId;
105. Show customers with the number of invoices and the average invoice total.  
     SELECT Customer.CustomerId, COUNT(Invoice.InvoiceId) AS InvoiceCount, AVG(Invoice.Total) AS AvgInvoice  
     FROM Customer  
     JOIN Invoice ON Customer.CustomerId = Invoice.CustomerId  
     GROUP BY Customer.CustomerId;
106. List all customers and the total amount they've spent, including those with no invoices.  
     SELECT Customer.CustomerId,  
     COALESCE((SELECT SUM(Total) FROM Invoice WHERE Invoice.CustomerId = Customer.CustomerId), 0) AS TotalSpent  
     FROM Customer;
107. Display all albums with the count of tracks, even if an album has no tracks.  
     SELECT Album.AlbumId, Album.Title, COUNT(Track.TrackId) AS TrackCount  
     FROM Album  
     LEFT JOIN Track ON Album.AlbumId = Track.AlbumId  
     GROUP BY Album.AlbumId, Album.Title;
108. Show all employees and the total sales of the customers they support, including employees with no customers.  
     SELECT Employee.EmployeeId, Employee.FirstName, Employee.LastName,  
     COALESCE((SELECT SUM(Total) FROM Invoice JOIN Customer ON Invoice.CustomerId = Customer.CustomerId  
     WHERE Customer.SupportRepId = Employee.EmployeeId), 0) AS TotalSales  
     FROM Employee;
109. List playlists and the average unit price of tracks in each playlist.  
     SELECT Playlist.PlaylistId, Playlist.Name, AVG(Track.UnitPrice) AS AvgUnitPrice  
     FROM Playlist  
     LEFT JOIN PlaylistTrack ON Playlist.PlaylistId = PlaylistTrack.PlaylistId  
     LEFT JOIN Track ON PlaylistTrack.TrackId = Track.TrackId  
     GROUP BY Playlist.PlaylistId, Playlist.Name;
110. Display artists and the number of invoices for their tracks sold, even if zero.  
     SELECT Artist.ArtistId, Artist.Name,  
     COALESCE((SELECT COUNT(InvoiceLine.InvoiceLineId)  
     FROM InvoiceLine  
     JOIN Track ON InvoiceLine.TrackId = Track.TrackId  
     JOIN Album ON Track.AlbumId = Album.AlbumId  
     WHERE Album.ArtistId = Artist.ArtistId), 0) AS InvoiceCount  
     FROM Artist;
111. List the top 5 tracks with unit price above the average, along with their album and artist.  
     SELECT Track.Name, Track.UnitPrice, Album.Title, Artist.Name  
     FROM Track  
     JOIN Album ON Track.AlbumId = Album.AlbumId  
     JOIN Artist ON Album.ArtistId = Artist.ArtistId  
     WHERE Track.UnitPrice > (SELECT AVG(UnitPrice) FROM Track)  
     ORDER BY Track.UnitPrice DESC LIMIT 5;
112. Show customers whose spending is above the 75th percentile.  
     SELECT \* FROM Customer  
     WHERE CustomerId IN (  
     SELECT CustomerId FROM Invoice  
     GROUP BY CustomerId  
     HAVING SUM(Total) > (  
     SELECT PERCENTILE\_CONT(0.75) WITHIN GROUP (ORDER BY SUM(Total))  
     FROM Invoice GROUP BY CustomerId  
     )  
     );
113. Display album titles where all tracks are above $1.00 in unit price.  
     SELECT Album.Title  
     FROM Album  
     WHERE Album.AlbumId NOT IN (SELECT AlbumId FROM Track WHERE UnitPrice <= 1.00);
114. List genres with the highest average track unit price.  
     SELECT Genre.Name, AVG(Track.UnitPrice) AS AvgPrice  
     FROM Track  
     JOIN Genre ON Track.GenreId = Genre.GenreId  
     GROUP BY Genre.Name  
     ORDER BY AvgPrice DESC LIMIT 1;
115. Show invoice details along with the customer’s total spending and the overall average spending.  
     SELECT Invoice.InvoiceId, Invoice.Total,  
     (SELECT SUM(Total) FROM Invoice WHERE CustomerId = Invoice.CustomerId) AS CustomerTotal,  
     (SELECT AVG(Total) FROM Invoice) AS OverallAvg  
     FROM Invoice;
116. List tracks with a label indicating if they are expensive (above 1.0) or cheap.  
     SELECT Name, UnitPrice,  
     CASE WHEN UnitPrice > 1.0 THEN 'Expensive' ELSE 'Cheap' END AS PriceCategory  
     FROM Track;
117. Show invoices with a discount flag if total is over 50 dollars.  
     SELECT InvoiceId, Total,  
     CASE WHEN Total > 50 THEN 'Discount Applicable' ELSE 'No Discount' END AS DiscountFlag  
     FROM Invoice;
118. Display customer spending with a category based on total amount spent.  
     SELECT CustomerId, SUM(Total) AS TotalSpent,  
     CASE WHEN SUM(Total) > 100 THEN 'High'  
     WHEN SUM(Total) BETWEEN 50 AND 100 THEN 'Medium'  
     ELSE 'Low' END AS SpendingCategory  
     FROM Invoice  
     GROUP BY CustomerId;
119. List albums with a classification based on having more than 10 tracks.  
     SELECT Album.Title,  
     CASE WHEN (SELECT COUNT(\*) FROM Track WHERE Track.AlbumId = Album.AlbumId) > 10 THEN 'Full Album' ELSE 'EP' END AS AlbumType  
     FROM Album;
120. Show employees with a status based on the number of customers they support.  
     SELECT Employee.EmployeeId, Employee.FirstName, Employee.LastName,  
     CASE WHEN (SELECT COUNT(\*) FROM Customer WHERE SupportRepId = Employee.EmployeeId) > 5  
     THEN 'Experienced' ELSE 'Newbie' END AS Status  
     FROM Employee;
121. List all artist names and album titles in one column.  
     SELECT Name AS Value FROM Artist  
     UNION  
     SELECT Title FROM Album;
122. Show distinct customer countries from invoices and customers.  
     SELECT Country FROM Invoice  
     UNION  
     SELECT Country FROM Customer;
123. Display track names that are either in the Rock genre or in a playlist named "Rock".  
     SELECT Name FROM Track WHERE GenreId = (SELECT GenreId FROM Genre WHERE Name = 'Rock')  
     UNION  
     SELECT Track.Name FROM Track  
     JOIN PlaylistTrack ON Track.TrackId = PlaylistTrack.TrackId  
     JOIN Playlist ON PlaylistTrack.PlaylistId = Playlist.PlaylistId  
     WHERE Playlist.Name ILIKE '%Rock%';
124. List all media type names and genre names together.  
     SELECT Name FROM MediaType  
     UNION  
     SELECT Name FROM Genre;
125. Show all unique dates from invoice dates and employees' hire dates.  
     SELECT InvoiceDate AS Date FROM Invoice  
     UNION  
     SELECT HireDate AS Date FROM Employee;
126. List employees along with their manager's name.  
     SELECT e.FirstName || ' ' || e.LastName AS EmployeeName,  
     m.FirstName || ' ' || m.LastName AS ManagerName  
     FROM Employee e  
     LEFT JOIN Employee m ON e.ReportsTo = m.EmployeeId;
127. Show customers who share the same support representative.  
     SELECT c1.CustomerId, c1.FirstName, c1.LastName, c2.CustomerId AS OtherCustomer  
     FROM Customer c1  
     JOIN Customer c2 ON c1.SupportRepId = c2.SupportRepId AND c1.CustomerId <> c2.CustomerId;
128. Display albums with similar titles using a self-join.  
     SELECT a1.AlbumId, a1.Title AS Album1, a2.Title AS Album2  
     FROM Album a1  
     JOIN Album a2 ON a1.Title = a2.Title AND a1.AlbumId <> a2.AlbumId;
129. List tracks that are unique by name (no duplicate names).  
     SELECT \* FROM Track t1  
     WHERE NOT EXISTS (SELECT 1 FROM Track t2 WHERE t2.Name = t1.Name AND t2.TrackId <> t1.TrackId);
130. Show customers and compare their invoice totals to the overall average using a self-join.  
     SELECT c1.CustomerId, c1.FirstName, c1.LastName, c2.InvoiceTotal, c1\_total.AvgTotal  
     FROM Customer c1  
     JOIN (SELECT CustomerId, SUM(Total) AS InvoiceTotal FROM Invoice GROUP BY CustomerId) c2  
     ON c1.CustomerId = c2.CustomerId  
     JOIN (SELECT AVG(Total) AS AvgTotal FROM Invoice GROUP BY CustomerId) c1\_total ON true;
131. List tracks ordered by album title and then by unit price descending.  
     SELECT Track.Name, Album.Title, Track.UnitPrice  
     FROM Track  
     JOIN Album ON Track.AlbumId = Album.AlbumId  
     ORDER BY Album.Title, Track.UnitPrice DESC;
132. Show invoices ordered by customer country and invoice date.  
     SELECT Invoice.InvoiceId, Customer.Country, Invoice.InvoiceDate  
     FROM Invoice  
     JOIN Customer ON Invoice.CustomerId = Customer.CustomerId  
     ORDER BY Customer.Country, Invoice.InvoiceDate;
133. Display customers ordered by the total number of invoices they have (descending).  
     SELECT Customer.CustomerId, COUNT(Invoice.InvoiceId) AS InvoiceCount  
     FROM Customer  
     JOIN Invoice ON Customer.CustomerId = Invoice.CustomerId  
     GROUP BY Customer.CustomerId  
     ORDER BY InvoiceCount DESC;
134. List albums ordered by the highest track unit price within each album.  
     SELECT Album.AlbumId, Album.Title,  
     (SELECT MAX(UnitPrice) FROM Track WHERE AlbumId = Album.AlbumId) AS MaxPrice  
     FROM Album  
     ORDER BY MaxPrice DESC;
135. Show playlist tracks ordered by playlist name and then track name.  
     SELECT Playlist.Name AS PlaylistName, Track.Name AS TrackName  
     FROM PlaylistTrack  
     JOIN Playlist ON PlaylistTrack.PlaylistId = Playlist.PlaylistId  
     JOIN Track ON PlaylistTrack.TrackId = Track.TrackId  
     ORDER BY Playlist.Name, Track.Name;
136. List invoices from the year 2010.  
     SELECT \* FROM Invoice  
     WHERE EXTRACT(YEAR FROM InvoiceDate) = 2010;
137. Show customers who registered after 2012 based on invoice dates.  
     SELECT \* FROM Customer  
     WHERE CustomerId IN (SELECT CustomerId FROM Invoice WHERE InvoiceDate > '2012-01-01');
138. Display invoices with the month and year formatted.  
     SELECT InvoiceId, InvoiceDate, TO\_CHAR(InvoiceDate, 'Month YYYY') AS FormattedDate  
     FROM Invoice;
139. Show invoices within the last month.  
     SELECT \* FROM Invoice  
     WHERE InvoiceDate >= CURRENT\_DATE - INTERVAL '1 month';
140. Display employees with their hire year.  
     SELECT EmployeeId, FirstName, LastName, EXTRACT(YEAR FROM HireDate) AS HireYear  
     FROM Employee;
141. List customer names in uppercase.  
     SELECT UPPER(FirstName) || ' ' || UPPER(LastName) AS FullName  
     FROM Customer;
142. Show album titles with the word count.  
     SELECT Title, (LENGTH(Title) - LENGTH(REPLACE(Title, ' ', '')) + 1) AS WordCount  
     FROM Album;
143. Display track names with their first 10 characters.  
     SELECT Name, SUBSTRING(Name FROM 1 FOR 10) AS ShortName  
     FROM Track;
144. List employee full names in title case.  
     SELECT INITCAP(FirstName) || ' ' || INITCAP(LastName) AS FullName  
     FROM Employee;
145. Show customer emails with the domain extracted.  
     SELECT Email, SUBSTRING(Email FROM POSITION('@' IN Email)) AS Domain  
     FROM Customer;
146. Display album titles in reverse order.  
     SELECT Title, REVERSE(Title) AS ReversedTitle  
     FROM Album;
147. List track names with their length in minutes (rounded).  
     SELECT Name, ROUND(Milliseconds / 60000.0, 2) AS LengthMinutes  
     FROM Track;
148. Show invoice dates with the day of the week.  
     SELECT InvoiceId, InvoiceDate, TO\_CHAR(InvoiceDate, 'Day') AS DayOfWeek  
     FROM Invoice;
149. List customers with their name length and email length.  
     SELECT FirstName, LastName, LENGTH(FirstName || ' ' || LastName) AS NameLength, LENGTH(Email) AS EmailLength  
     FROM Customer;
150. Display artist names with a prefix "Artist: ".  
     SELECT 'Artist: ' || Name AS ArtistName  
     FROM Artist;
151. Show running total of invoice amounts by customer.  
     SELECT InvoiceId, CustomerId, Total,  
     SUM(Total) OVER (PARTITION BY CustomerId ORDER BY InvoiceDate) AS RunningTotal  
     FROM Invoice;
152. List tracks with their cumulative sum of unit prices across the album.  
     SELECT TrackId, Name, UnitPrice,  
     SUM(UnitPrice) OVER (PARTITION BY AlbumId ORDER BY TrackId) AS CumulativePrice  
     FROM Track;
153. Display the rank of each album based on total track duration.  
     SELECT Album.AlbumId, Album.Title,  
     (SELECT SUM(Milliseconds) FROM Track WHERE AlbumId = Album.AlbumId) AS TotalDuration,  
     RANK() OVER (ORDER BY (SELECT SUM(Milliseconds) FROM Track WHERE AlbumId = Album.AlbumId) DESC) AS AlbumRank  
     FROM Album;
154. List customers with the percentage contribution of each invoice to their total spending.  
     SELECT Invoice.InvoiceId, Invoice.CustomerId, Invoice.Total,  
     (Invoice.Total / SUM(Invoice.Total) OVER (PARTITION BY Invoice.CustomerId) \* 100) AS ContributionPercentage  
     FROM Invoice;
155. Show playlist tracks with the row number partitioned by playlist.  
     SELECT PlaylistTrack.PlaylistId, Track.Name,  
     ROW\_NUMBER() OVER (PARTITION BY PlaylistTrack.PlaylistId ORDER BY Track.Name) AS RowNum  
     FROM PlaylistTrack  
     JOIN Track ON PlaylistTrack.TrackId = Track.TrackId;
156. Display top 5 artists by total invoice amount using a CTE and window function.  
     WITH ArtistSales AS (  
     SELECT Artist.ArtistId, Artist.Name, SUM(InvoiceLine.UnitPrice \* InvoiceLine.Quantity) AS TotalSales  
     FROM Artist  
     JOIN Album ON Artist.ArtistId = Album.ArtistId  
     JOIN Track ON Album.AlbumId = Track.AlbumId  
     JOIN InvoiceLine ON Track.TrackId = InvoiceLine.TrackId  
     GROUP BY Artist.ArtistId, Artist.Name  
     )  
     SELECT ArtistId, Name, TotalSales, RANK() OVER (ORDER BY TotalSales DESC) AS SalesRank  
     FROM ArtistSales  
     ORDER BY SalesRank LIMIT 5;
157. List customers with their total spending and percentile rank among all customers using a CTE.  
     WITH CustomerSpending AS (  
     SELECT Customer.CustomerId, SUM(Invoice.Total) AS TotalSpent  
     FROM Customer  
     JOIN Invoice ON Customer.CustomerId = Invoice.CustomerId  
     GROUP BY Customer.CustomerId  
     )  
     SELECT CustomerId, TotalSpent, PERCENT\_RANK() OVER (ORDER BY TotalSpent) AS SpendingPercentile  
     FROM CustomerSpending;
158. Show albums along with average track unit price using a CTE and window function.  
     WITH AlbumPrices AS (  
     SELECT Album.AlbumId, Album.Title, AVG(Track.UnitPrice) AS AvgPrice  
     FROM Album  
     JOIN Track ON Album.AlbumId = Track.AlbumId  
     GROUP BY Album.AlbumId, Album.Title  
     )  
     SELECT AlbumId, Title, AvgPrice, ROW\_NUMBER() OVER (ORDER BY AvgPrice DESC) AS Rank  
     FROM AlbumPrices;
159. Display customer invoice details along with a cumulative sum of invoices per customer using a CTE.  
     WITH CustInvoices AS (  
     SELECT InvoiceId, CustomerId, Total, InvoiceDate  
     FROM Invoice  
     )  
     SELECT InvoiceId, CustomerId, Total,  
     SUM(Total) OVER (PARTITION BY CustomerId ORDER BY InvoiceDate) AS CumulativeTotal  
     FROM CustInvoices;
160. List playlist names with total duration and rank them using a CTE and window function.  
     WITH PlaylistDurations AS (  
     SELECT Playlist.PlaylistId, Playlist.Name, SUM(Track.Milliseconds) AS TotalDuration  
     FROM Playlist  
     JOIN PlaylistTrack ON Playlist.PlaylistId = PlaylistTrack.PlaylistId  
     JOIN Track ON PlaylistTrack.TrackId = Track.TrackId  
     GROUP BY Playlist.PlaylistId, Playlist.Name  
     )  
     SELECT PlaylistId, Name, TotalDuration,  
     RANK() OVER (ORDER BY TotalDuration DESC) AS DurationRank  
     FROM PlaylistDurations;
161. Display employee hierarchy using a recursive CTE.  
     WITH RECURSIVE EmployeeHierarchy AS (  
     SELECT EmployeeId, FirstName, LastName, ReportsTo  
     FROM Employee  
     WHERE ReportsTo IS NULL  
     UNION ALL  
     SELECT e.EmployeeId, e.FirstName, e.LastName, e.ReportsTo  
     FROM Employee e  
     JOIN EmployeeHierarchy eh ON e.ReportsTo = eh.EmployeeId  
     )  
     SELECT \* FROM EmployeeHierarchy;
162. Display employee reporting chain using a recursive CTE.  
     WITH RECURSIVE ReportingChain AS (  
     SELECT EmployeeId, FirstName, LastName, ReportsTo, CAST(FirstName || ' ' || LastName AS TEXT) AS Chain  
     FROM Employee  
     WHERE ReportsTo IS NULL  
     UNION ALL  
     SELECT e.EmployeeId, e.FirstName, e.LastName, e.ReportsTo, rc.Chain || ' -> ' || e.FirstName || ' ' || e.LastName  
     FROM Employee e  
     JOIN ReportingChain rc ON e.ReportsTo = rc.EmployeeId  
     )  
     SELECT \* FROM ReportingChain;
163. List all employees with their depth level in the organization using a recursive CTE.  
     WITH RECURSIVE OrgChart AS (  
     SELECT EmployeeId, FirstName, LastName, ReportsTo, 1 AS Level  
     FROM Employee  
     WHERE ReportsTo IS NULL  
     UNION ALL  
     SELECT e.EmployeeId, e.FirstName, e.LastName, e.ReportsTo, oc.Level + 1  
     FROM Employee e  
     JOIN OrgChart oc ON e.ReportsTo = oc.EmployeeId  
     )  
     SELECT \* FROM OrgChart;
164. Show employee hierarchy with a concatenated name chain using a recursive CTE.  
     WITH RECURSIVE EmpChain AS (  
     SELECT EmployeeId, FirstName, LastName, ReportsTo, FirstName AS Chain  
     FROM Employee  
     WHERE ReportsTo IS NULL  
     UNION ALL  
     SELECT e.EmployeeId, e.FirstName, e.LastName, e.ReportsTo, ec.Chain || ' > ' || e.FirstName  
     FROM Employee e  
     JOIN EmpChain ec ON e.ReportsTo = ec.EmployeeId  
     )  
     SELECT \* FROM EmpChain;
165. Display employee reporting hierarchy along with level using a recursive CTE.  
     WITH RECURSIVE Hierarchy AS (  
     SELECT EmployeeId, FirstName, LastName, ReportsTo, 0 AS Level  
     FROM Employee  
     WHERE ReportsTo IS NULL  
     UNION ALL  
     SELECT e.EmployeeId, e.FirstName, e.LastName, e.ReportsTo, h.Level + 1  
     FROM Employee e  
     JOIN Hierarchy h ON e.ReportsTo = h.EmployeeId  
     )  
     SELECT \* FROM Hierarchy;
166. Display invoice totals pivoted by country using CASE statements.  
     SELECT Customer.Country,  
     SUM(CASE WHEN Total > 50 THEN Total ELSE 0 END) AS Over50Sales,  
     SUM(CASE WHEN Total <= 50 THEN Total ELSE 0 END) AS Under50Sales  
     FROM Invoice  
     JOIN Customer ON Invoice.CustomerId = Customer.CustomerId  
     GROUP BY Customer.Country;
167. List customers with invoice counts pivoted by year using CASE.  
     SELECT Customer.CustomerId,  
     SUM(CASE WHEN EXTRACT(YEAR FROM InvoiceDate) = 2010 THEN 1 ELSE 0 END) AS Invoices2010,  
     SUM(CASE WHEN EXTRACT(YEAR FROM InvoiceDate) = 2011 THEN 1 ELSE 0 END) AS Invoices2011  
     FROM Invoice  
     JOIN Customer ON Invoice.CustomerId = Customer.CustomerId  
     GROUP BY Customer.CustomerId;
168. Show track counts per media type and genre pivoted using CASE.  
     SELECT MediaType.Name AS MediaType,  
     SUM(CASE WHEN Genre.Name = 'Rock' THEN 1 ELSE 0 END) AS RockTracks,  
     SUM(CASE WHEN Genre.Name = 'Jazz' THEN 1 ELSE 0 END) AS JazzTracks  
     FROM Track  
     JOIN MediaType ON Track.MediaTypeId = MediaType.MediaTypeId  
     JOIN Genre ON Track.GenreId = Genre.GenreId  
     GROUP BY MediaType.Name;
169. Display album track counts pivoted by genre using CASE.  
     SELECT Album.AlbumId, Album.Title,  
     SUM(CASE WHEN Track.GenreId = (SELECT GenreId FROM Genre WHERE Name = 'Rock') THEN 1 ELSE 0 END) AS RockCount  
     FROM Album  
     JOIN Track ON Album.AlbumId = Track.AlbumId  
     GROUP BY Album.AlbumId, Album.Title;
170. List customers with spending pivoted by invoice year using CASE.  
     SELECT Customer.CustomerId,  
     SUM(CASE WHEN EXTRACT(YEAR FROM InvoiceDate) = 2010 THEN Total ELSE 0 END) AS Spending2010,  
     SUM(CASE WHEN EXTRACT(YEAR FROM InvoiceDate) = 2011 THEN Total ELSE 0 END) AS Spending2011  
     FROM Invoice  
     JOIN Customer ON Invoice.CustomerId = Customer.CustomerId  
     GROUP BY Customer.CustomerId;
171. Show invoices for USA customers with totals above the USA average.  
     SELECT Invoice.InvoiceId, Invoice.Total  
     FROM Invoice  
     JOIN Customer ON Invoice.CustomerId = Customer.CustomerId  
     WHERE Customer.Country = 'USA'  
     AND Invoice.Total > (  
     SELECT AVG(Total) FROM Invoice  
     JOIN Customer ON Invoice.CustomerId = Customer.CustomerId  
     WHERE Customer.Country = 'USA'  
     );
172. List albums whose artist name starts with "The" and have more than 5 tracks.  
     SELECT Album.Title  
     FROM Album  
     JOIN Artist ON Album.ArtistId = Artist.ArtistId  
     WHERE Artist.Name ILIKE 'The%'  
     AND Album.AlbumId IN (  
     SELECT AlbumId FROM Track GROUP BY AlbumId HAVING COUNT(\*) > 5  
     );
173. Display tracks that are in playlists named "Favorites" and have a unit price above the playlist's average.  
     SELECT Track.Name, Track.UnitPrice  
     FROM Track  
     JOIN PlaylistTrack ON Track.TrackId = PlaylistTrack.TrackId  
     JOIN Playlist ON PlaylistTrack.PlaylistId = Playlist.PlaylistId  
     WHERE Playlist.Name = 'Favorites'  
     AND Track.UnitPrice > (  
     SELECT AVG(t.UnitPrice)  
     FROM Track t  
     JOIN PlaylistTrack pt ON t.TrackId = pt.TrackId  
     JOIN Playlist p ON pt.PlaylistId = p.PlaylistId  
     WHERE p.Name = 'Favorites'  
     );
174. List customers along with their most expensive invoice total.  
     SELECT Customer.CustomerId, Customer.FirstName, Customer.LastName,  
     (SELECT MAX(Total) FROM Invoice WHERE CustomerId = Customer.CustomerId) AS MaxInvoiceTotal  
     FROM Customer;
175. Show artists whose albums have an average track length above the overall average.  
     SELECT Artist.Name  
     FROM Artist  
     JOIN Album ON Artist.ArtistId = Album.ArtistId  
     JOIN Track ON Album.AlbumId = Track.AlbumId  
     GROUP BY Artist.Name  
     HAVING AVG(Track.Milliseconds) > (SELECT AVG(Milliseconds) FROM Track);
176. List playlist names with the number of tracks and the average track unit price in each playlist.  
     SELECT Playlist.Name, COUNT(PlaylistTrack.TrackId) AS TrackCount,  
     (SELECT AVG(UnitPrice) FROM Track WHERE Track.TrackId IN (SELECT TrackId FROM PlaylistTrack WHERE PlaylistId = Playlist.PlaylistId)) AS AvgUnitPrice  
     FROM Playlist  
     LEFT JOIN PlaylistTrack ON Playlist.PlaylistId = PlaylistTrack.PlaylistId  
     GROUP BY Playlist.Name;
177. Display customer details along with the count of invoices and the total spent using subqueries.  
     SELECT Customer.CustomerId, Customer.FirstName, Customer.LastName,  
     (SELECT COUNT(\*) FROM Invoice WHERE CustomerId = Customer.CustomerId) AS InvoiceCount,  
     (SELECT SUM(Total) FROM Invoice WHERE CustomerId = Customer.CustomerId) AS TotalSpent  
     FROM Customer;
178. Show employee names with the total sales of their supported customers and rank them.  
     SELECT Employee.FirstName || ' ' || Employee.LastName AS EmployeeName,  
     (SELECT SUM(Total) FROM Invoice JOIN Customer ON Invoice.CustomerId = Customer.CustomerId  
     WHERE Customer.SupportRepId = Employee.EmployeeId) AS TotalSales,  
     RANK() OVER (ORDER BY (SELECT SUM(Total) FROM Invoice JOIN Customer ON Invoice.CustomerId = Customer.CustomerId  
     WHERE Customer.SupportRepId = Employee.EmployeeId) DESC) AS SalesRank  
     FROM Employee;
179. List tracks with their album title, artist name, and genre name, and flag tracks over $1.5 as premium.  
     SELECT Track.Name, Album.Title, Artist.Name, Genre.Name,  
     CASE WHEN Track.UnitPrice > 1.5 THEN 'Premium' ELSE 'Standard' END AS PriceCategory  
     FROM Track  
     JOIN Album ON Track.AlbumId = Album.AlbumId  
     JOIN Artist ON Album.ArtistId = Artist.ArtistId  
     JOIN Genre ON Track.GenreId = Genre.GenreId;
180. Show invoices with customer name, total, and percentage of customer's total spending for each invoice.  
     SELECT Invoice.InvoiceId, Customer.FirstName || ' ' || Customer.LastName AS CustomerName, Invoice.Total,  
     (Invoice.Total / (SELECT SUM(Total) FROM Invoice WHERE CustomerId = Invoice.CustomerId) \* 100) AS SpendingPercentage  
     FROM Invoice  
     JOIN Customer ON Invoice.CustomerId = Customer.CustomerId;
181. List all distinct artist and customer names in alphabetical order.  
     SELECT Name AS PersonName FROM Artist  
     UNION  
     SELECT FirstName || ' ' || LastName FROM Customer  
     ORDER BY PersonName;
182. Show album titles and track names combined from both Album and Track tables.  
     SELECT Title AS Name FROM Album  
     UNION  
     SELECT Name FROM Track  
     ORDER BY Name;
183. Display all media type and genre names together in one list sorted by name.  
     SELECT Name FROM MediaType  
     UNION  
     SELECT Name FROM Genre  
     ORDER BY Name;
184. List all invoice dates from invoices and the invoice count per date using UNION.  
     SELECT InvoiceDate::DATE AS Date FROM Invoice  
     UNION  
     SELECT InvoiceDate::DATE FROM Invoice GROUP BY InvoiceDate::DATE  
     ORDER BY Date;
185. Show all customer emails and artist names combined uniquely.  
     SELECT Email AS Value FROM Customer  
     UNION  
     SELECT Name FROM Artist  
     ORDER BY Value;
186. List tracks with names that match a full-text search for "love".  
     SELECT \* FROM Track  
     WHERE to\_tsvector('english', Name) @@ plainto\_tsquery('english', 'love');
187. Show album titles that match a full-text search for "rock".  
     SELECT \* FROM Album  
     WHERE to\_tsvector('english', Title) @@ plainto\_tsquery('english', 'rock');
188. Display customer names that match a full-text search for "john".  
     SELECT \* FROM Customer  
     WHERE to\_tsvector('english', FirstName || ' ' || LastName) @@ plainto\_tsquery('english', 'john');
189. List artist names that match a full-text search for "band".  
     SELECT \* FROM Artist  
     WHERE to\_tsvector('english', Name) @@ plainto\_tsquery('english', 'band');
190. Show track names that match a full-text search for "guitar".  
     SELECT \* FROM Track  
     WHERE to\_tsvector('english', Name) @@ plainto\_tsquery('english', 'guitar');
191. Display total invoice amount grouped by country and year using GROUPING SETS.  
     SELECT Customer.Country, EXTRACT(YEAR FROM InvoiceDate) AS Year, SUM(Total) AS TotalSales  
     FROM Invoice  
     JOIN Customer ON Invoice.CustomerId = Customer.CustomerId  
     GROUP BY GROUPING SETS ((Customer.Country, EXTRACT(YEAR FROM InvoiceDate)), (Customer.Country), (EXTRACT(YEAR FROM InvoiceDate)))  
     ORDER BY Customer.Country, Year;
192. Show album track counts with overall totals using ROLLUP.  
     SELECT Album.Title, COUNT(Track.TrackId) AS TrackCount  
     FROM Album  
     LEFT JOIN Track ON Album.AlbumId = Track.AlbumId  
     GROUP BY ROLLUP(Album.Title);
193. List customer spending with subtotals and grand total using CUBE.  
     SELECT Customer.Country, SUM(Total) AS TotalSpent  
     FROM Invoice  
     JOIN Customer ON Invoice.CustomerId = Customer.CustomerId  
     GROUP BY CUBE(Customer.Country)  
     ORDER BY Customer.Country;
194. Display track count and total duration per genre with subtotals using GROUPING SETS.  
     SELECT Genre.Name, COUNT(Track.TrackId) AS TrackCount, SUM(Track.Milliseconds) AS TotalDuration  
     FROM Track  
     JOIN Genre ON Track.GenreId = Genre.GenreId  
     GROUP BY GROUPING SETS ((Genre.Name), ());
195. Show the number of invoices and total sales by employee and overall using ROLLUP.  
     SELECT Employee.EmployeeId, Employee.FirstName || ' ' || Employee.LastName AS EmployeeName, SUM(Invoice.Total) AS TotalSales  
     FROM Employee  
     LEFT JOIN Customer ON Employee.EmployeeId = Customer.SupportRepId  
     LEFT JOIN Invoice ON Customer.CustomerId = Invoice.CustomerId  
     GROUP BY ROLLUP(Employee.EmployeeId, Employee.FirstName, Employee.LastName);
196. List tracks with their album and artist, including a column for total tracks in the album using a window function.  
     SELECT Track.Name, Album.Title, Artist.Name,  
     COUNT(Track.TrackId) OVER (PARTITION BY Album.AlbumId) AS AlbumTrackCount  
     FROM Track  
     JOIN Album ON Track.AlbumId = Album.AlbumId  
     JOIN Artist ON Album.ArtistId = Artist.ArtistId;
197. Show the top selling artist based on invoice line sales using subquery and aggregation.  
     SELECT Artist.Name,  
     (SELECT SUM(InvoiceLine.UnitPrice \* InvoiceLine.Quantity)  
     FROM InvoiceLine  
     JOIN Track ON InvoiceLine.TrackId = Track.TrackId  
     JOIN Album ON Track.AlbumId = Album.AlbumId  
     WHERE Album.ArtistId = Artist.ArtistId) AS TotalSales  
     FROM Artist  
     ORDER BY TotalSales DESC LIMIT 1;
198. Display the average track price for each album along with album and artist details.  
     SELECT Album.Title, Artist.Name, AVG(Track.UnitPrice) AS AvgTrackPrice  
     FROM Album  
     JOIN Artist ON Album.ArtistId = Artist.ArtistId  
     JOIN Track ON Album.AlbumId = Track.AlbumId  
     GROUP BY Album.Title, Artist.Name;
199. List customers who have purchased tracks from more than one genre.  
     SELECT Customer.CustomerId, Customer.FirstName, Customer.LastName  
     FROM Customer  
     WHERE Customer.CustomerId IN (  
     SELECT Invoice.CustomerId  
     FROM Invoice  
     JOIN InvoiceLine ON Invoice.InvoiceId = InvoiceLine.InvoiceId  
     JOIN Track ON InvoiceLine.TrackId = Track.TrackId  
     GROUP BY Invoice.CustomerId  
     HAVING COUNT(DISTINCT Track.GenreId) > 1  
     );
200. Show a detailed invoice report with customer, invoice, invoice lines, track, album, and artist information.  
     SELECT Invoice.InvoiceId, Invoice.InvoiceDate,  
     Customer.FirstName || ' ' || Customer.LastName AS CustomerName, Invoice.Total,  
     InvoiceLine.Quantity, Track.Name AS TrackName, Track.UnitPrice,  
     Album.Title AS AlbumTitle, Artist.Name AS ArtistName  
     FROM Invoice  
     JOIN Customer ON Invoice.CustomerId = Customer.CustomerId  
     JOIN InvoiceLine ON Invoice.InvoiceId = InvoiceLine.InvoiceId  
     JOIN Track ON InvoiceLine.TrackId = Track.TrackId  
     JOIN Album ON Track.AlbumId = Album.AlbumId  
     JOIN Artist ON Album.ArtistId = Artist.ArtistId  
     ORDER BY Invoice.InvoiceId;

## Sample text to SQL application

I want to develop a text-to-SQL application using a streamlit frontend, where I want to allow the uset to connect to any supported database (user will specify the connection parameters, such as hostname, userid, password & database) - let's assume it is any PostgreSQL database to begin with. Plan is to extend to PostgreSQL, MySQL, Oracle and SQL Server databases in future using same concept.

When the application connects to this database for the first time, it should create embeddings from the database’s information schema (tables and views definitions, primary & foreign keys) based on which the LLM will be able to generate SQL. The user can (optionally) provide additional training dataset, which will be a PDF (or text file) that list sample text queries (such as "show me the most popular product" and the related SQL) - assume this will be specific to the database we'll be connecting to and it's the user's responsibility to prove the correct samples. If the user provides such a file, it should also be included in the embeddings. The user has the option of taking a 2 step approach too - in the first step, she provided just the database connection info and may choose to provide additional data in a separate step later. The streamlit UI should have a provision to allow user to upload additional training files (could be multiple) at any the time or after the first database connection. Any such embeddings created should be saved to distinct files under local subfolder (let’s call it dbinfo). You should generate code to associate db-connection with the respecting embedding file (or vector db file) stored in this sub-folder. This will help you determine if embedding needs to be created or not. Any training dataset files provided at a later time should be “added” to this vector db only – don’t create multiple vector dbs.

The streamlit UI will have a sidebar, where the user will enter database connection information & connect to the database. It will also have a file upload facility, which user can use to upload additional training dataset files. The main section should have an expandable multiline text field, where the user can enter a text prompt (such as “show me the product that has maximum sales”). Once user enters this text and pressed ENTER, the code should use the embedding and ask LLM to generate SQL from the embeddings – SQL should be displayed to the user in a collapsible “div” with appropriate syntax highlighting. After SQL is generated, the SQL will be executed and results displayed in a table below the generated SQL – table should fit width of the main section. If a data plot is appropriate for the kind of SQL generated, then a plotly graph should also be displayed below the table.

Using the above requirements spec, generate appropriate Python code – I’ll be starting with PostgreSQL, but you should generate (within comments) appropriate code for MySQL, Oracle and SQL Server databases too – prefer native database interface libraries (e.g. psycopg for PostgreSQL) where available and SQLAlchemy otherwise.

Here is the Streamlit-based Python application for your text-to-SQL system. The code is structured to:

* **Connect to PostgreSQL (with placeholders for MySQL, Oracle, and SQL Server)**
* **Generate embeddings from the database schema** (tables, columns, relationships)
* **Incorporate user-provided training datasets** (PDF or text files)
* **Allow users to input text queries** and convert them to SQL using an LLM
* **Execute the generated SQL** and display results in a table
* **Visualize results using Plotly** (if applicable)

This version starts with **PostgreSQL** using psycopg and SQLAlchemy. Comments indicate how to extend it for MySQL, Oracle, and SQL Server.

**Features:**

* **Database Schema Embeddings:** Extracts schema metadata and stores it in a vector database.
* **User Training Files:** Supports additional training via uploaded PDFs or text files.
* **Text-to-SQL via LLM:** Converts natural language queries to SQL.
* **SQL Execution & Result Display:** Runs the generated SQL and presents results.
* **Plotly Visualization:** If applicable, generates a relevant chart.

Here is the Streamlit-based Python application for your text-to-SQL system. The code is structured to:

* **Connect to PostgreSQL (with placeholders for MySQL, Oracle, and SQL Server)**
* **Generate embeddings from the database schema** (tables, columns, relationships)
* **Incorporate user-provided training datasets** (PDF or text files)
* **Allow users to input text queries** and convert them to SQL using an LLM
* **Execute the generated SQL** and display results in a table
* **Visualize results using Plotly** (if applicable)

This version starts with **PostgreSQL** using psycopg and SQLAlchemy. Comments indicate how to extend it for MySQL, Oracle, and SQL Server.

**Features:**

* **Database Schema Embeddings:** Extracts schema metadata and stores it in a vector database.
* **User Training Files:** Supports additional training via uploaded PDFs or text files.
* **Text-to-SQL via LLM:** Converts natural language queries to SQL.
* **SQL Execution & Result Display:** Runs the generated SQL and presents results.
* **Plotly Visualization:** If applicable, generates a relevant chart.

Here is the **Streamlit application code**:

The Streamlit app is now ready to handle text-to-SQL queries using PostgreSQL, with plans to extend to MySQL, Oracle, and SQL Server. Let me know if you need modifications or further enhancements! 🚀

import os  
import json  
import psycopg2 *# PostgreSQL*  
import pandas as pd  
import streamlit as st  
import plotly.express as px  
from langchain.embeddings import OpenAIEmbeddings  
from langchain.vectorstores import FAISS  
from langchain.llms import OpenAI  
from langchain.chains import RetrievalQA  
from langchain.document\_loaders import PyPDFLoader, TextLoader  
from sqlalchemy import create\_engine, text  
from typing import Optional  
  
*# Directory to store database embeddings*  
EMBEDDINGS\_DIR = "dbinfo"  
os.makedirs(EMBEDDINGS\_DIR, exist\_ok=True)  
  
*# Load environment variables (for OpenAI API key)*  
OPENAI\_API\_KEY = os.getenv("OPENAI\_API\_KEY")  
  
*# Streamlit UI*  
def main():  
 st.set\_page\_config(page\_title="Text-to-SQL Query Generator", layout="wide")  
 st.sidebar.header("Database Connection")  
   
 *# Database connection inputs*  
 db\_host = st.sidebar.text\_input("Host", "localhost")  
 db\_port = st.sidebar.text\_input("Port", "5432")  
 db\_name = st.sidebar.text\_input("Database Name")  
 db\_user = st.sidebar.text\_input("Username")  
 db\_password = st.sidebar.text\_input("Password", type="password")  
   
 connect\_btn = st.sidebar.button("Connect")  
   
 *# File upload for additional training data*  
 st.sidebar.header("Upload Training Data")  
 uploaded\_files = st.sidebar.file\_uploader("Upload training files (PDF/Text)", type=["pdf", "txt"], accept\_multiple\_files=True)  
   
 if connect\_btn and db\_name and db\_user and db\_password:  
 connection\_string = f"postgresql://{db\_user}:{db\_password}@{db\_host}:{db\_port}/{db\_name}"  
   
 with st.spinner("Connecting to database..."):  
 engine = create\_engine(connection\_string)  
 connection = engine.connect()  
 st.session\_state["engine"] = engine  
 st.success("Connected successfully!")  
   
 embedding\_file = os.path.join(EMBEDDINGS\_DIR, f"{db\_name}.faiss")  
 if not os.path.exists(embedding\_file):  
 with st.spinner("Generating database embeddings..."):  
 generate\_schema\_embeddings(engine, db\_name)  
 st.success("Embeddings created and saved.")  
   
 st.header("Text-to-SQL Query")  
 user\_query = st.text\_area("Enter your question:", height=100)  
 submit\_btn = st.button("Generate SQL & Execute")  
   
 if submit\_btn and user\_query:  
 with st.spinner("Generating SQL query..."):  
 generated\_sql = generate\_sql\_from\_text(user\_query, db\_name)  
 st.subheader("Generated SQL")  
 st.code(generated\_sql, language='sql')  
   
 with st.spinner("Executing query..."):  
 results\_df = execute\_sql\_query(engine, generated\_sql)  
 if results\_df is not None:  
 st.subheader("Query Results")  
 st.dataframe(results\_df)  
   
 *# Check if visualization is possible*  
 if len(results\_df.columns) >= 2:  
 st.subheader("Data Visualization")  
 fig = px.bar(results\_df, x=results\_df.columns[0], y=results\_df.columns[1])  
 st.plotly\_chart(fig)  
  
*# Function to generate schema embeddings*  
def generate\_schema\_embeddings(engine, db\_name):  
 metadata = {}  
 with engine.connect() as conn:  
 result = conn.execute(text("""  
 SELECT table\_name, column\_name, data\_type  
 FROM information\_schema.columns  
 WHERE table\_schema = 'public';  
 """))  
 metadata["tables"] = [dict(row) for row in result]  
   
 result = conn.execute(text("""  
 SELECT conname, conrelid::regclass, confrelid::regclass  
 FROM pg\_constraint WHERE contype = 'f';  
 """))  
 metadata["foreign\_keys"] = [dict(row) for row in result]  
   
 *# Save embeddings*  
 embeddings = OpenAIEmbeddings()  
 vectorstore = FAISS.from\_texts([json.dumps(metadata)], embeddings)  
 vectorstore.save\_local(os.path.join(EMBEDDINGS\_DIR, db\_name))  
  
*# Function to generate SQL from text*  
def generate\_sql\_from\_text(query: str, db\_name: str) -> str:  
 embedding\_file = os.path.join(EMBEDDINGS\_DIR, f"{db\_name}.faiss")  
 embeddings = OpenAIEmbeddings()  
 vectorstore = FAISS.load\_local(embedding\_file, embeddings)  
   
 retriever = vectorstore.as\_retriever()  
 qa\_chain = RetrievalQA.from\_chain\_type(OpenAI(temperature=0), retriever=retriever)  
 response = qa\_chain.run(query)  
 return response  
  
*# Function to execute SQL query*  
def execute\_sql\_query(engine, sql\_query: str) -> Optional[pd.DataFrame]:  
 try:  
 with engine.connect() as conn:  
 df = pd.read\_sql(sql\_query, conn)  
 return df  
 except Exception as e:  
 st.error(f"Error executing query: {e}")  
 return None  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()  
  
"""  
# Adapting to Other Databases:  
# - MySQL: Use `import pymysql` and `mysql+pymysql://{user}:{password}@{host}:{port}/{database}`  
# - Oracle: Use `import cx\_Oracle` and `oracle+cx\_oracle://{user}:{password}@{host}:{port}/{service\_name}`  
# - SQL Server: Use `import pyodbc` and `mssql+pyodbc://{user}:{password}@{server}/{database}?driver=ODBC+Driver+17+for+SQL+Server`  
"""