Chinook Music Store — Sales Analysis

Intermediate SQL: subqueries, CASE expressions, and multi-table analysis over the Chinook store database.

Instructions

Answer each of the 8 questions below with a single, complete SQL query. Every table reference must be schema-qualified using the chinook prefix (e.g., chinook.Track), and every column reference must be table-qualified (e.g., Track.Name) — this applies in all clauses including SELECT, WHERE, JOIN ON, GROUP BY, HAVING, and ORDER BY. Do not use table aliases. Submit a single .sql file named Lastname_Firstname_Assignment.sql with each query clearly numbered using a comment (e.g., -- Question 1) corresponding to the question numbers below. Queries that are not runnable or that omit schema and table qualification will receive partial credit at best.

Scenario

Chinook Digital Music Store needs to better understand its catalog, customer base, and sales performance across global markets. As junior data analysts, you have been tasked with running a series of SQL queries against the Chinook database to surface key business insights. Your findings will inform decisions about marketing campaigns, employee performance reviews, and catalog expansion.

Database Structure

The Chinook database models a digital music shop. Artists produce Albums, which contain Tracks. Tracks belong to Genres and MediaTypes. Customers place purchases recorded as Invoices, each with InvoiceLines linking back to Tracks. Employees support customers in a hierarchy. Tracks can also be grouped into Playlists via PlaylistTrack.

Questions

1. Retrieve the first name, last name, and email address of all customers who are located in the United States. Sort the results alphabetically by last name, then by first name. *(5 pts)*

Solution:

```
SELECT Customer.FirstName, Customer.LastName, Customer.Email FROM chinook.Customer WHERE Custom
```

2. List the names of all tracks that have a unit price greater than $0.99. Return only the track name and its unit price, ordered by unit price descending and then track name ascending. *(5 pts)*

Solution:

```
SELECT Track.Name, Track.UnitPrice FROM chinook.Track WHERE Track.UnitPrice > 0.99 ORDER BY Tra
```

3. Retrieve a list of all tracks along with the name of the genre each track belongs to and the name of its media type. Return the track name, genre name, and media type name, ordered by genre name and then track name, both ascending. *(10 pts)*

> *Hint: You will need to join Track to both Genre and MediaType using the appropriate foreign keys.*

Solution:

```
SELECT Track.Name, Genre.Name, MediaType.Name FROM chinook.Track JOIN chinook.Genre ON Track.Ge
```

4. For each artist, show the artist's name and the total number of albums they have in the catalog. Include artists who have no albums (show 0 for those). Order the results by album count descending, then by artist name ascending. *(10 pts)*

> *Hint: Use a LEFT JOIN so that artists with no albums still appear in the results, and use COUNT on the album side of the join.*

Solution:

```
SELECT Artist.Name, COUNT(Album.AlbumId) AS AlbumCount FROM chinook.Artist LEFT JOIN chinook.Al
```

5. Find all genres that contain more than 50 tracks in the catalog. Return the genre name and the total track count for each qualifying genre, ordered by track count descending. *(15 pts)*

> *Hint: After grouping by genre, use HAVING to filter groups based on the count of tracks.*

Solution:

```
SELECT Genre.Name, COUNT(Track.TrackId) AS TrackCount FROM chinook.Genre JOIN chinook.Track ON
```

6. For each support employee, report the employee's full name (first and last), their title, and the total revenue (sum of invoice totals) generated from the customers they support. Only include employees who actually have the title 'Sales Support Agent'. Order the results by total revenue descending. *(15 pts)*

> *Hint: Join Employee to Customer using SupportRepId, then join Customer to Invoice to reach the invoice totals.*

Solution:

```
SELECT Employee.FirstName, Employee.LastName, Employee.Title, SUM(Invoice.Total) AS TotalRevenu
```

7. Retrieve the full name (first and last) and email of all customers who have never made a purchase (i.e., have no invoices recorded in the system). Return the results ordered by last name, then first name, both ascending. *(20 pts)*

> *Hint: Consider using a subquery in a NOT IN clause to identify customers whose IDs do not appear in the Invoice table.*

Solution:

```
SELECT Customer.FirstName, Customer.LastName, Customer.Email FROM chinook.Customer WHERE Custom
```

8. For each billing country found in the Invoice table, calculate the total number of invoices and the total revenue (sum of invoice totals). Additionally, classify each country into a revenue tier using the following rules: 'High' if total revenue is greater than $100, 'Medium' if total revenue is between $50 and $100 inclusive, and 'Low' if total revenue is less than $50. Return the billing country, invoice count, total revenue (rounded to 2 decimal places), and revenue tier. Order the results by total revenue descending. *(20 pts)*

> *Hint: Group by billing country first, then apply a CASE WHEN expression to the aggregated SUM to assign each country its revenue tier.*

Solution:

```
SELECT Invoice.BillingCountry, COUNT(Invoice.InvoiceId) AS InvoiceCount, ROUND(SUM(Invoice.Tota
```

Answer Key

Q1. (5 pts) — *select, where, order_by*

Retrieve the first name, last name, and email address of all customers who are located in the United States. Sort the results alphabetically by last name, then by first name.

```
SELECT Customer.FirstName, Customer.LastName, Customer.Email FROM chinook.Customer WHERE Custom
```

Tables: chinook.Customer
Fields: Customer.FirstName, Customer.LastName, Customer.Email, Customer.Country

Q2. (5 pts) — *select, where, order_by*

List the names of all tracks that have a unit price greater than $0.99. Return only the track name and its unit price, ordered by unit price descending and then track name ascending.

```
SELECT Track.Name, Track.UnitPrice FROM chinook.Track WHERE Track.UnitPrice > 0.99 ORDER BY Tra
```

Tables: chinook.Track
Fields: Track.Name, Track.UnitPrice

Q3. (10 pts) — *join, order_by*

Retrieve a list of all tracks along with the name of the genre each track belongs to and the name of its media type. Return the track name, genre name, and media type name, ordered by genre name and then track name, both ascending.

```
SELECT Track.Name, Genre.Name, MediaType.Name FROM chinook.Track JOIN chinook.Genre ON Track.Ge
```

Tables: chinook.Track, chinook.Genre, chinook.MediaType
Fields: Track.Name, Genre.Name, MediaType.Name

Q4. (10 pts) — *left_join, aggregate, group_by*

For each artist, show the artist's name and the total number of albums they have in the catalog. Include artists who have no albums (show 0 for those). Order the results by album count descending, then by artist name ascending.

```
SELECT Artist.Name, COUNT(Album.AlbumId) AS AlbumCount FROM chinook.Artist LEFT JOIN chinook.Al
```

Tables: chinook.Artist, chinook.Album
Fields: Artist.Name, Album.AlbumId

Q5. (15 pts) — *join, aggregate, group_by, having*

Find all genres that contain more than 50 tracks in the catalog. Return the genre name and the total track count for each qualifying genre, ordered by track count descending.

```
SELECT Genre.Name, COUNT(Track.TrackId) AS TrackCount FROM chinook.Genre JOIN chinook.Track ON
```

Tables: chinook.Genre, chinook.Track

Fields: Genre.Name, Track.TrackId

Q6. (15 pts) — *join, aggregate, group_by, order_by*

For each support employee, report the employee's full name (first and last), their title, and the total revenue (sum of invoice totals) generated from the customers they support. Only include employees who actually have the title 'Sales Support Agent'. Order the results by total revenue descending.

```
SELECT Employee.FirstName, Employee.LastName, Employee.Title, SUM(Invoice.Total) AS TotalRevenu
```

Tables: chinook.Employee, chinook.Customer, chinook.Invoice

Fields: Employee.FirstName, Employee.LastName, Employee.Title, Invoice.Total, Customer.SupportRepId

Q7. (20 pts) — *subquery, in_clause, join*

Retrieve the full name (first and last) and email of all customers who have never made a purchase (i.e., have no invoices recorded in the system). Return the results ordered by last name, then first name, both ascending.

```
SELECT Customer.FirstName, Customer.LastName, Customer.Email FROM chinook.Customer WHERE Custom
```

Tables: chinook.Customer, chinook.Invoice

Fields: Customer.FirstName, Customer.LastName, Customer.Email, Customer.CustomerId, Invoice.CustomerId

Q8. (20 pts) — *join, aggregate, group_by, case_when, order_by*

For each billing country found in the Invoice table, calculate the total number of invoices and the total revenue (sum of invoice totals). Additionally, classify each country into a revenue tier using the following rules: 'High' if total revenue is greater than $100, 'Medium' if total revenue is between $50 and $100 inclusive, and 'Low' if total revenue is less than $50. Return the billing country, invoice count, total revenue (rounded to 2 decimal places), and revenue tier. Order the results by total revenue descending.

```
SELECT Invoice.BillingCountry, COUNT(Invoice.InvoiceId) AS InvoiceCount, ROUND(SUM(Invoice.Tota
```

Tables: chinook.Invoice

Fields: Invoice.BillingCountry, Invoice.InvoiceId, Invoice.Total