Final Project Report

GAP

Step 1. Scenario and Database Requirements

What is the imaginary company or organization you are working for?

The Global Audio Platform (GAP) shines a light on the problems that independent writers and artists around the world face in the vast world of digital music distribution. GAP's multidimensional structure allows music distribution and enables artists to efficiently market and monetize their work. In an era where visibility and accessibility are critical to success, GAP acts as an essential conduit, providing the tools and resources required for indie musicians, record companies, podcasters, and other audio content providers to prosper.

At its foundation, GAP is motivated by the fundamental goal of broadening the music industry, creating a dynamic ecosystem in which artistic expression has no boundaries. GAP provides a comprehensive range of services customized to the needs of independent producers, letting them engage with audiences worldwide. Whether it's releasing music on major streaming platforms, developing focused marketing campaigns, or implementing creative monetization models, GAP gives artists the confidence and agility they need to navigate the digital landscape with ease.

However, GAP is more than just a distribution platform; it represents community values, functioning as a supportive network where artists can find solidarity, collaboration chances, and crucial insights from their peers. GAP fosters a culture of collaboration and progress by hosting a varied range of forums, workshops, and networking events, educating artists through the ever-changing landscape of the music industry. In essence, GAP is more than just a service provider; it is a driving force in the democratization of music, paving the way for a more inclusive, diverse, and vibrant audio ecosystem.

In a world where entry hurdles often appear insurmountable, GAP emerges as a transformational force, leveling the playing field and amplifying the voices of independent producers all around the world. As artists negotiate the difficulties of the digital age, GAP remains committed to empowering their path, ushering in a time of limitless creativity, cooperation, and opportunity. GAP transforms the once-daunting environment of the music industry into passable territory, where talent and creativity reign supreme and every artist can thrive and achieve on their own terms.

What is the purpose of the database?

The GAP database is the backbone of the music industry, built to handle the massive amounts of data that are inherent to this dynamic field. This huge library contains everything from individual tracks to full albums, as well as artist bios and curated playlists, covering every aspect of the musical landscape. Its powerful architecture provides users with advanced management capabilities that keep data organized, searchable, and easily accessible.

However, the database serves more than just as a storage facility; it is an essential component of the marketing process. It serves as the conduit for music distribution to online stores and streaming services globally, allowing artists and record companies to easily interact with a global audience. Furthermore, it allows for real-time tracking of sales, royalties, and audience engagement analytics, giving artists crucial insights into how their work is received and how far their popularity reaches.

From the user's perspective, the GAP database transforms into a dynamic discovery platform, utilizing advanced algorithms to provide personalized recommendations based on individual tastes and preferences. Whether consumers are looking for the latest releases or exploring obscure genres, they rely on the database to find hidden gems and undiscovered talents, enriching their musical journey with each discovery. Stakeholders also benefit from extensive analytics, which provide insights into music trends, artist popularity, and user behavior, so driving strategic decisions and improving awareness of the changing music industry landscape.

Every part of the musical ecosystem comes together in the GAP database, creating an environment in which creators, consumers, and industry professionals can thrive. Its broad reach and comprehensive skills enable stakeholders to manage the challenges of today's music industry with confidence and agility. As evidence of its value, the GAP database serves not just as a repository of musical knowledge, but also as a catalyst for creativity, cooperation, and discovery throughout the worldwide music community.

Furthermore, the database acts as a hub for collaboration and creation, promoting partnerships between artists, producers, and industry participants. It promotes synergies and mutual support through sophisticated networking capabilities and collaboration tools, resulting in a lively ecosystem of artistic expression and creativity.

In conclusion, the GAP database is more than just a collection of music-related data; it is a critical hub that propels the modern music industry forward. It enables stakeholders to traverse the digital ecosystem and realize their creative vision by offering necessary tools, insights, and chances. As the music industry evolves, the GAP database remains a steadfast ally, promoting innovation, encouraging collaboration, and determining the future of music around the world.

Who are the intended users?

1. Independent Artists and Bands

Independent musicians and bands are important to the GAP community, using the platform to post, manage, and distribute their music to a global audience. The GAP database provides artists with a centralized platform for showcasing their work, engaging with fans, and tracking progress. The platform provides critical tools for promotion, revenue, and audience engagement, allowing artists to confidently negotiate the complexity of the music industry.

2. Music Consumers

The GAP database allows music enthusiasts to explore music in many genres and styles. With its user-friendly design and personalized recommendations, users may discover new releases, niche genres, and hidden gems based on their likes. The platform improves the music discovery experience by establishing a stronger connection between artists and their audiences.

3. Record Labels

The GAP database helps record labels manage and distribute various artists and releases more efficiently. With sophisticated analytics and real-time tracking tools, labels obtain vital insights about their roster's performance, allowing for more informed decision-making and strategic planning. The platform encourages collaboration between labels and artists, resulting in a dynamic ecosystem of invention and innovation.

4. GAP Marketing Teams

Marketing teams use the platform's data to analyze trends, select target consumers, and execute campaigns. Marketing teams can maximize reach and impact by employing detailed analytics and user interaction data in their promotional tactics. The platform enables marketers to maximize their efforts by increasing visibility and engagement for artists and content across digital channels.

5. Administrators

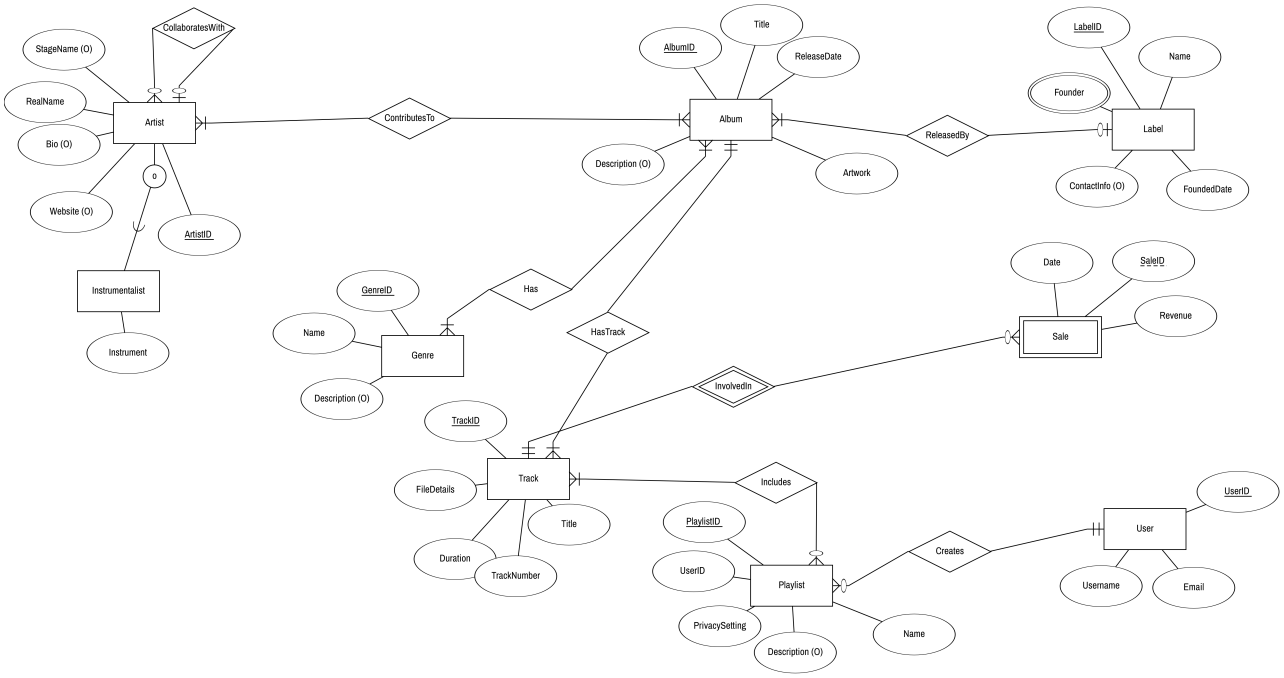
Administrators ensure the GAP platform's integrity and availability. Administrators are responsible for managing user accounts, maintaining data integrity, and assuring platform stability. They maintain the highest levels of performance and security. Their tireless efforts guarantee that all users have a flawless experience, allowing the platform to run smoothly and maintaining community trust.

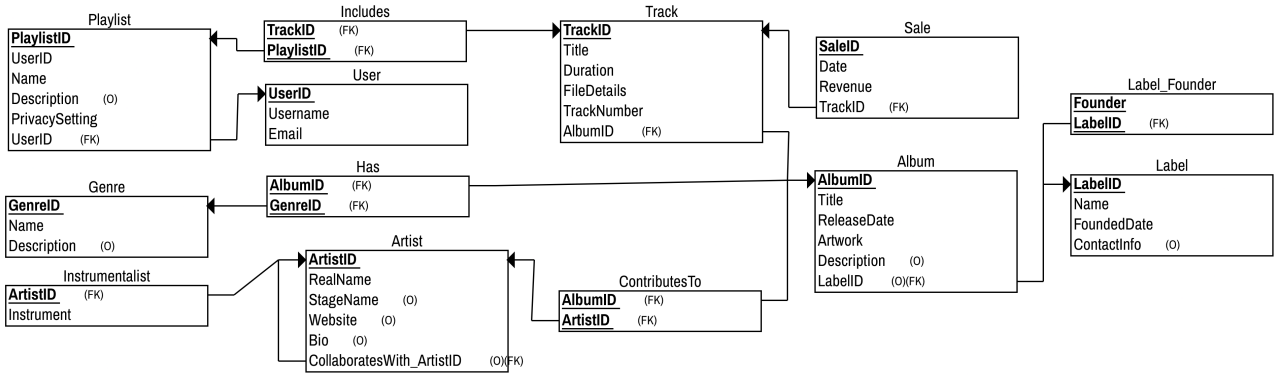
In essence, the GAP platform serves a wide range of users, from independent artists seeking exposure to music consumers looking for new discoveries, from record labels managing their rosters to marketing teams driving promotional efforts, and from administrators ensuring platform reliability to all stakeholders navigating the music industry's digital landscape.

Step 2. EERD

You've created a many-to-many relationship, recursive relationship, weak entity, and set of a superclass and subclasses.

For each case, what is your reason behind the design decision?





Many-to-Many

One example of a many-to-many relationship in our database structure is the Artist>ContributesTo>Album relationship. We used this structure because it is possible for an artist to contribute to multiple albums and, at the same time, for a single album to be the product of contributions by many artists. In order to accurately represent this state of affairs, we used a many-to-many relationship.

Recursive Relationship

A recursive relationship in our database design is the Artist>CollaboratesWith>Artist relationship. This points between artists which have worked together on an album. We used this structure to describe links which may occasionally work together, but don't do so under a collective name.

Weak Entity

A weak entity relationship in our database structure is the Track>InvolvedIn>Sale relationship. This exists because it is not possible to have a sale without a track, and it is only useful to record sale information in the context of tracks; there is no independent purpose to it.

Subclass-Superclass

A subclass-superclass relationship in our database structure is the Instrumentalist>Artist relationship. This exists because an artist can be an instrumentalist, which requires further information about the instrument they play, but some artists are not instrumentalists. Furthermore, by creating a subclass instead of an optional property, it becomes very simple to subdivide artists between instrumentalists and noninstrumentalists.

Step 4. Normalization/Denormalization

If you decide to denormalize any table, what are the rationales for the decision?

If all of your tables are normalized, why didn't you denormalize any table?

When exploring denormalization for our database, we considered the possible benefits of improved query performance and simplified data retrieval methods. Most importantly, we entertained the idea of denormalizing the "Sale" table and merging it with the "Track" table via the "TrackID" foreign key. We suggested that doing so might enhance query speed, simplify data retrieval and database complexity, and improve backend response times.

However, in the end we decided not to denormalize the table. It was felt that, while it might be convenient to see all the sales data for a track from the track table without having to construct special queries, the high upper limit to the number of times that a track was played—and therefore involved in a sale transaction—could produce results that interfered with the use of the Track table for other purposes. For example, if a track had 5,000 listens, it would show up in the Track table 5,000 times, as each sale transaction would require its own row. Scale up to hundreds of thousands, or even millions, of possible transactions per track, times thousands of tracks, and it quickly becomes clear that denormalizing the relationship would create many more difficulties than affordances. The amount of duplicate rows for each track would make any track-level queries require large amounts of omitted data, which could easily be avoided by fully normalizing the relationship.

Step 5. Create and populate DB

Where do your data come from if you didn't create the entire data alone?

Some data for our database was manually extracted from Gareth's personal music collection. The rest was created originally.

For a few attributes, why did you select the data types?

1. TrackID: CHAR(t)

The TrackID is a unique identifier for each track in the database. It has a set format, beginning with a letter signifying the type of entity (e.g., 'T' for tracks) and followed by a numerical value. CHAR(t) was chosen because it allows for a fixed-length string of characters, which ensures that TrackIDs are formatted consistently.

2. Album ID: CHAR(5)

AlbumID, like TrackID, is a unique identifier that is particular to albums. It follows a consistent format, beginning with a letter designating the entity type ('A' for albums) and ending with a numerical value. Using CHAR(5) ensures that AlbumIDs have a set length, allowing for efficient storage and retrieval.

3. Title: VARCHAR(255)

Album titles vary in length and may include a substantial amount of text. VARCHAR(255) was chosen because it can store variable-length strings of up to 255 characters, allowing for a wide range of album titles without imposing unwanted constraints.

4. Duration: TIME

The duration of each recording in hours, minutes, and seconds. The TIME data type is chosen because it accurately records time information without requiring further formatting. It makes it easier to store and manipulate track durations, as well as do computations and comparisons.

5. FileDetails: VARCHAR(5)

FileDetails contains information about the file type associated with each track, such as MP3, AAC, or FLAC. Because file type abbreviations might vary in length, VARCHAR is used to handle various name lengths. Using VARCHAR(5) offers you greater versatility while assuring efficient file storage.

6. TrackNumber: INT(5)

TrackNumber identifies each track's location inside an album. It is written as an integer with up to five digits to allow for huge albums with many tracks. Using INT(5) means that TrackNumber can efficiently store and handle numeric values ranging from 1 to 99999, allowing for scalability across albums of various sizes.

Step 6. SQL Statements

Among your 10 queries, pick at least four queries and state how they are closely related to the purpose and intended users of your database.

If your queries are for generating reports, answer the following questions.

Who will see the report?

What decision will be made based on it?

If your queries are for direct interactions with end-users via an application, answer the following questions.

What is the application?

Who are the users?

1.
```
SELECT LABEL.Name, COUNT(*) AS TotalAlbums
FROM ALBUM
JOIN LABEL ON ALBUM.LabelID = LABEL.LabelID
GROUP BY LABEL.Name;
```

Input

Run SQL

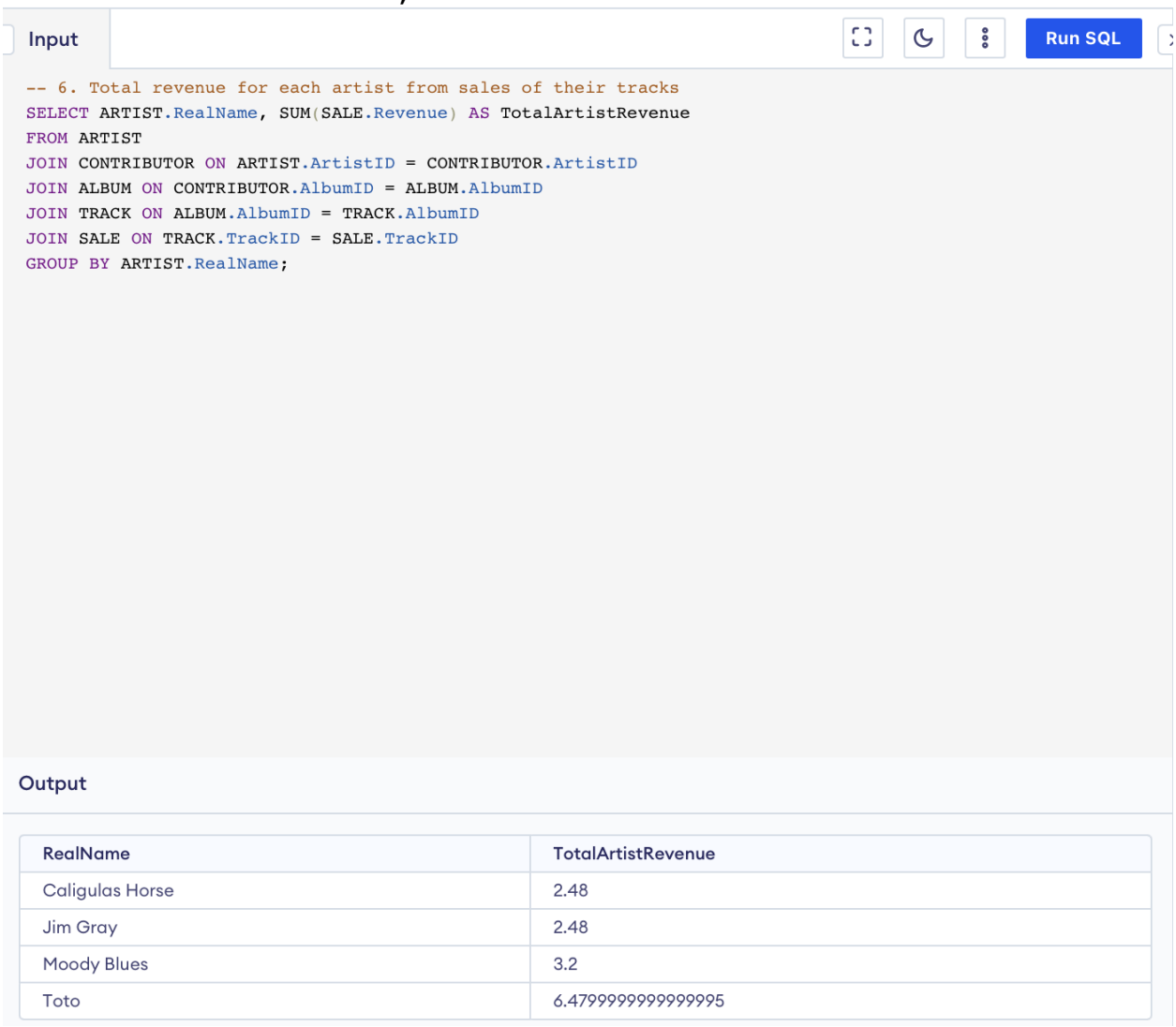
```
-- 1. Total number of albums per label
SELECT LABEL.Name, COUNT(*) AS TotalAlbums
FROM ALBUM
JOIN LABEL ON ALBUM.LabelID = LABEL.LabelID
GROUP BY LABEL.Name;
```

Output

Name	TotalAlbums
BIS Records	1
Columbia	1
Decca	1
InsideOutMusic	1
Universal	1

This query retrieves a count of the total albums in the database published by each label. This is useful for generating reports, as it can show us which of our customers are most heavily represented on the platform. Marketing and sales teams can then use this information to target their efforts on those more impactful customers, or focus away from them in order to broaden our customer base.

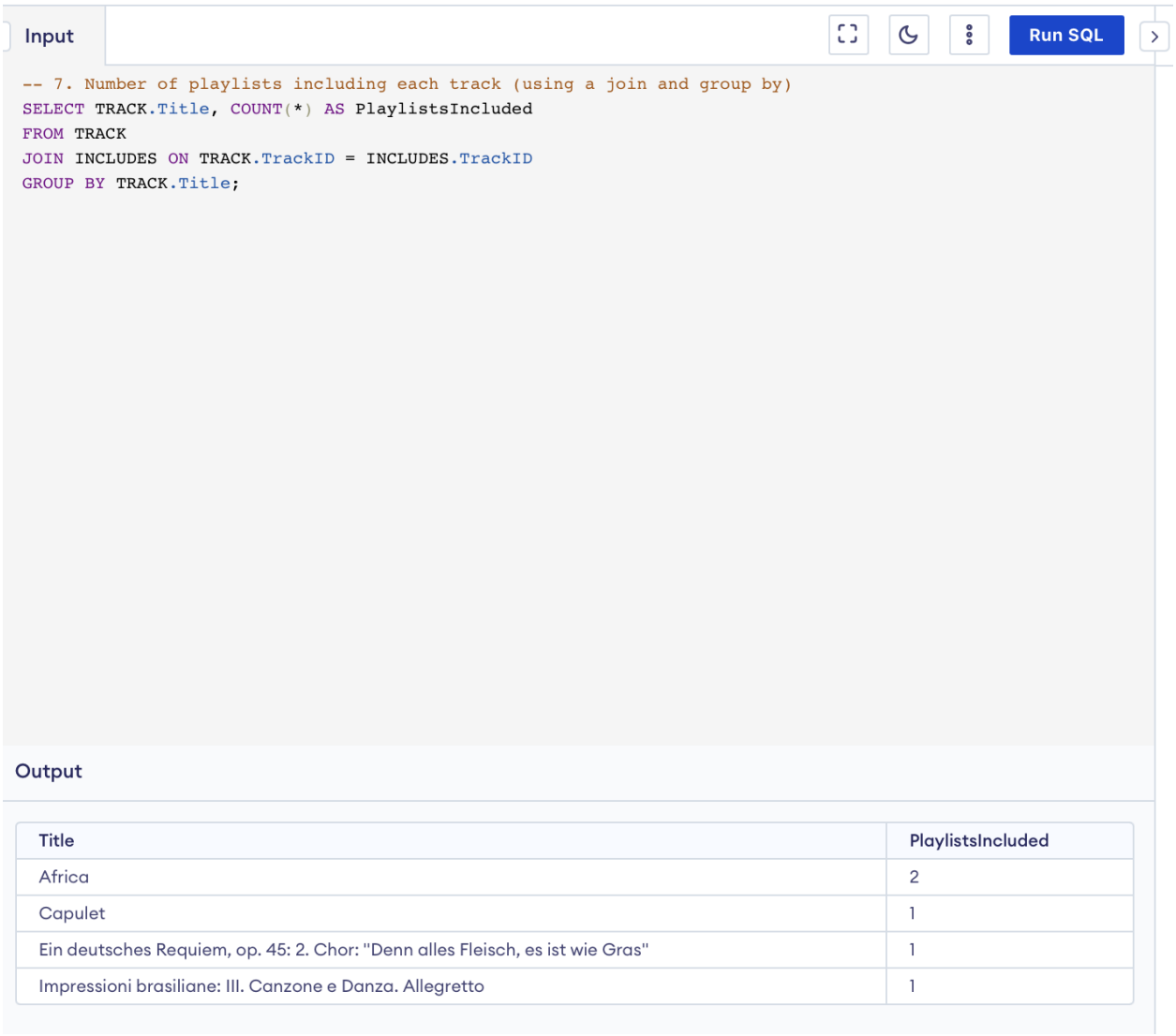
2.
```
SELECT ARTIST.RealName, SUM (SALE.Revenue) AS TotalArtistRevenue
FROM ARTIST
JOIN CONTRIBUTOR ON ARTIST.ArtistID = CONTRIBUTOR.ArtistID
JOIN ALBUM ON CONTRIBUTOR.AlbumID = ALBUM.AlbumID
JOIN TRACK ON ALBUM.AlbumID = TRACK.AlbumID
JOIN SALE ON TRACK.TrackID = SALE.TrackID
GROUP BY ARTIST.RealName;
```



RealName	TotalArtistRevenue
Caligulas Horse	2.48
Jim Gray	2.48
Moody Blues	3.2
Toto	6.4799999999999995

This query retrieves the total revenue generated for each artist from sales of their tracks. This is useful for generating reports, as it allows us to demonstrate, in marketing and sales materials, the advantages for artists to becoming our customers. In a dense, murky world of complicated and varied contractual employees, which can often make it difficult to “follow the money” from listener to label to artist, being able to conclusively point to a number which tells an artist how much money they have made is a big sell.

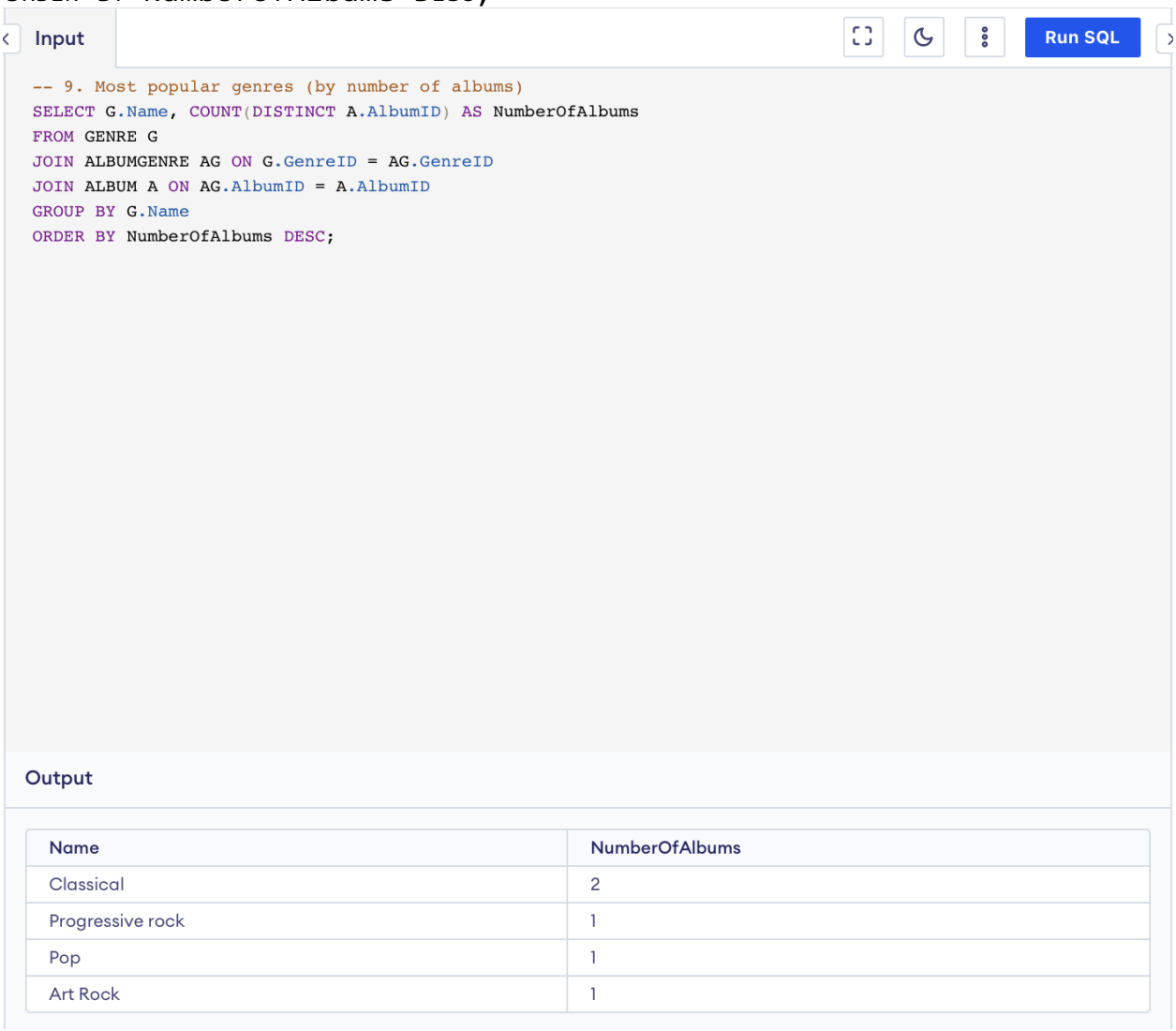
3.
```
SELECT Track.Title, Count(*) as PlaylistsIncluded
From Track
Join Includes on Track:TrackID = Includes.TrackID
Group by Track.Title
```



Title	PlaylistsIncluded
Africa	2
Capulet	1
Ein deutsches Requiem, op. 45: 2. Chor: "Denn alles Fleisch, es ist wie Gras"	1
Impressioni brasiliane: III. Canzone e Danza. Allegretto	1

This query retrieves the total number of playlists each track is included on. This report serves a similar impact-demonstration purpose as the previous queries, allowing us to pinpoint top-performing tracks and leverage them to demonstrate the impact of our product. However, it can also serve an internal use by being factored into suggestion algorithms; tracks which have a higher number of playlist additions are more popular.

4.
```
SELECT G.Name, COUNT(DISTINCT A.AlbumID) AS NumberOfAlbums
FROM GENRE G
JOIN AlbumGenre AG ON G.GenreID = AG.GenreID
JOIN ALBUM A ON AG.AlbumID = A.AlbumID
GROUP BY G.Name
ORDER BY NumberOfAlbums DESC;
```



Name	NumberOfAlbums
Classical	2
Progressive rock	1
Pop	1
Art Rock	1

This query enumerates the number of albums in each genre. This report allows us to identify areas in which our database collection is underdeveloped; if we have only a very few albums in a genre which is reasonably popular, that is a signal that we should target artists and labels active in that genre space in order to continue to have a dominant and diverse position in the industry. Once again, this is data which should be directed to marketing or sales personnel for use in their activities.

Process

The production process for the database and report was straightforward. The team collaborated closely in the planning stage, creating a cohesive, unified vision for the structure of the database through the use of ERDs and relational schemas. This was expressed in the project proposal, produced in March of this year. In the initial phase of the preparation of the final product, the team divided responsibilities: Gareth formulated the SQL statements for table creation and data insertion based on previously agreed-upon standards for data types and structures; Pranav created insightful sample queries in order to

demonstrate the abilities of the database and the manifold uses it affords to the organization and our customers; and Achu documented our work thus far and prepared the final report. After our individual parts were completed, we reconvened, revised, and reflected on our own and the others' contributions, making any changes that we collectively felt were appropriate.