Exploring Local and Cloud Database Implementation: A Comparative Analysis of MySQL vs. PostgreSQL in Developing a Predictive Analytics Tool for Music Agents

By

Jonathan Berman Sax

Thesis Manuscript
Submitted in partial fulfillment of the
Requirements for the degree of

MASTER OF SCIENCE IN DATA SCIENCE

March 2024

Abid Ali, Ph.D, Thesis Advisor
Sharon Dill, Ph.D, Final Reader

ABSTRACT

Exploring Local and Cloud Database Implementation: A Comparative Analysis of MySQL vs. PostgreSQL in Developing a Predictive Analytics Tool for Music Agents

Jonathan Berman Sax

The role of a music agent demands proficiency across various domains. Logistical expertise, music knowledge, and interpersonal finesse are a few key components. In addition, as the speed and scalability of local and cloud database management systems improve, agents who add a nimble and adaptive data strategy to their repertoire will enjoy the most rapid advancement in work efficiency. To that extent, touring data is constantly being generated. At any given moment, there are measurable insights from ticket sales, target audience behavior, travel considerations, and seasonal factors. This manuscript pioneers the development of a novel and bespoke booking tool for music agents that integrates three central components of data science: database design and testing, analytics, and data visualization. The database will be implemented in two of the most popular open-source commercial engines, MySQL and PostgreSQL. The performance of business queries across varying sizes and complexity will be benchmark tested locally and in Google Cloud. The output of these queries will be assessed and translated into actionable analytics. To enhance analytical capabilities, interactive geospatial data visualizations will be created, aiding agents in deciding where, when, and how often to tour their artists. Lastly, an optimal path forward to utilizing local and cloud SQL capabilities will be configured, and recommendations for future updates and scalability will be established.

Table of Contents

List of Tables, Illustrations, Figures and Graphs

Chapter 1: Introduction

In the music industry, agents are similar to sales professionals, in that they bear the responsibility of ensuring the consistent, sustainable sale of their product in a competitive marketplace. In the electronic music realm, agents' customers are promoters or promotional entities, and agents' clients are DJs and live acts. The product agents sell are their artists' musical performances. The multifaceted nature of this operation emphasizes the potential for agents to gain a technological edge by harnessing the power of data science in ways that have not yet been explored by their competitors.

A music agent's principal role is to negotiate, confirm, and contract events for touring artists based on a myriad of considerations. From the artist's side, a music agent factors exact specifications like the artist's style of music, travel preferences, visa requirements within the agent's jurisdiction, and the artist's tour schedule in other territories. But there are also non-exact criteria to consider like buzzworthiness (or in other words, an artist's recent popularity level), which is, in turn, a projection of their future ticket sales.

Facilitating the alignment of artists' performances with suitable promoters is of utmost importance. The promoter-side negotiation is normally led by a talent buyer, whose role is to "buy" the performance in a business-savvy way that takes into account the artist's buzz, the venue's budget, capacity, and ticket price scaling, all while factoring various other potential revenue streams like bar sales, ticketing fees, and advertising.

Much of the strategy and decision making music agents pursue is based on years of relationship building, empirical observations, and trial and error. Still, within the last decade, there have been great advancements in database management systems for music agencies that have allowed for significant improvements in archiving and organization. This shift has allowed agents to increasingly rely on quantitative metrics, and less on soft skills.

The music industry is currently experiencing a significant AI revolution, and the way fans find, purchase, and enjoy music events is evolving at an extremely rapid pace (Habib and Wilde, 2023). But like in so many other industries, change can be hard to implement swiftly on an enterprise level, and in today's digital age, predictive technologies are evolving at an unprecedented clip. Therefore, remaining stubborn, entrenched, and unwavering in a booking agency's analytics strategy is unwise, if not impossible, even in the short term.

An overarching purpose of this manuscript is to synthesize three main components of data science: database design and testing, analytics, and visualization of insights. The first step will be delving into the capabilities of relational database management in a live music touring context, and exploring database architecture and implementation in order to most efficiently deliver actionable insights, and drive better decision making for future touring. The second step will be defining a relational database schema for artists' past booking data utilizing a one-of-a-kind dataset consisting of thousands of event bookings over the last five years. The database will be populated in both MySQL and PostgreSQL, and the performance of useful, substantive queries in both engines will be measured via benchmark testing on a local machine, and again in Google Cloud. Benchmark testing will be run again in each environment without indexing. Comparisons between all eight scenarios will be closely documented. In summary, the eight database environments to be evaluated are:

1. PostgreSQL local (with primary and foreign key indexing)
2. MySQL local (with PK and FK indexing)
3. PostgreSQL cloud (with PK and FK indexing)
4. MySQL cloud (with PK and FK indexing)
5. PostgreSQL local (without indexing)
6. MySQL local (without indexing)

7. PostgreSQL cloud (without indexing)
8. MySQL cloud (without indexing)

By querying this unique dataset, there lies an opportunity to identify trends previously undetectable, acknowledge bottlenecks and other unfavorable business practices, and then accelerate toward areas with growth potential. The content assembled in this manuscript aims to facilitate professionals involved in touring negotiations to work smarter, not harder. Through data, music agents and talent buyers can work harmoniously to bridge the gap between profitability and risk (Habib and Wilde, 2023).

Chapter 2: Business Problem and Justification

Successful tours involve several months of lead time to properly route (navigate), negotiate, discuss, and confirm. However, unforeseen circumstances – while sudden and unexpected – do occur frequently in touring, and with very little to no lead time. Macro examples of these events include inclement weather, pandemics, political elections, and war. Micro examples include flight delays, artist illnesses, promoter loss of festival and/or event permitting, venue loss of liquor licensing, and abrupt venue closures. With all of these possibilities in mind, an agile approach to troubleshooting is a vital component of a music agent's skill set.

When a problem occurs in real time, one of the most important items to factor is where the artist and their touring party currently are on the map. That is one of the many reasons why, even in today's digital age, most agents have a globe on their desk, or at minimum, a map on their bulletin board. A central part of this manuscript is harnessing the power of data to enhance that static map or desk globe through the digital exploration of geospatial analysis and geographic information systems (GIS). By harmonizing Python scripts and SQL queries, interactive maps will be generated with mouseover capabilities. This empowers agents to personalize their troubleshooting efforts with real-time, contextual results tailored to their unique needs.

This powerful tool is just as useful for tour planning during periods that are neither urgent nor problematic. In this dataset, each artist and promoter possesses three musical genre classifications mapped to their profiles based on the music they produce, and the styles of music they book. These genre classifications are based on new music released by each artist within the last two years. This segmentation helps agents place artists who are currently DJing and producing certain styles of music with corresponding promoters booking those genres.

By assessing data from an artist's recent tour(s), agents can sequentially:

- Approximate future performance fees and venue capacities through linear regression.
- Utilize findings from these predictive analytics to identify future performance locations that align closely with predetermined criteria.
- Engage in highly precise targeted outreach to the most compatible promoters, determined by the alignment of artist genre and promoter genre.
- Enhance the potency of each reach-out, getting tours booked quicker and more efficiently.

Another compelling rationale justifying the development of this sales tool is its indirect benefit to artists. The improved efficiency in agents' work enables artists to not only secure more income through touring, but also allow them to book travel earlier as well. Booking tours well in advance broadens travel options, generally reducing overall expenses, which is particularly important nowadays, as artists increasingly depend on touring for their income.

Chapter 3: Literature Review

Impacts of Digitalization on the Music Industry

Due to various technological disruptions and reconfigurations of capital, income streams in the music industry have evolved over the last half century (Zendel 2021). The rapid transitions from records to CDs, CDs to mp3s, mp3s to streaming, and so forth have occurred without consideration for individuals whose livelihoods relied upon these media formats. There is much less money to be made by the sale of tangible media today, and revenue from streaming has not made up for that loss (Zendel 2021). Therefore, nowadays touring is a central, primary form of income for musicians, and this is especially true in the electronic music scene.

This is why the geography of the Americas – with significantly further distances between major markets than European counterparts – presents sizeable challenges to electronic DJs and their teams. When an agent is planning a tour, it needs to be routed in a way that maximizes exposure, minimizes expenses, and ensures that artist(s) and their touring parties can consistently deliver their best performances.

To further complicate this challenge, electronic music genres, venues, neighborhoods, and local electronic music scenes of various markets in the Americas are constantly changing, and it is up to the artist's agent to stay up-to-date and ahead of the curve. "Music scenes are the transitory, ever-evolving culture zones of popular music tied to geographic locations and aesthetic concerns" (Rogers and Whiting 2020, 452). This is particularly true in electronic music, where every facet of style is constantly changing, and whatever sound is coming next is extremely challenging to predict.

Although the digitalization of the music industry has facilitated remote work, there is still a need for music agents to engage with artists, managers, promoters, and other professionals face-to-face. This is why agents who travel to clubs and festivals, meet with potential new clients and customers, and constantly interface with other industry professionals will have a leg up over their competitors who stay confined to their email inboxes and day-to-day dealings. This separation has been further exacerbated by the growth of work-from-home setups after the recent COVID-19 pandemic.

The advent of a versatile sales tool is particularly advantageous in this context. Agents will have the ability to utilize this booking tool worldwide, whether it be via the cloud, or connecting “locally” to a hosted database via a virtual private network (VPN). And for agents isolated by any social, political, or economic factors, the tool helps bridge the gap by enabling rapid and independent analytical access to their unique historical dataset to guide future decision making. Regardless of whether users work at boutique or corporate agencies, on the road or at home, this booking tool minimizes a significant portion of uncertainties in the tour planning process.

Scalability and Performance Optimization in SQL

Several factors contribute to why the preferred method for database administration in this manuscript is structured query language (SQL) and the two relational database management systems (RDBMS) under comparison are MySQL and PostgreSQL.

Firstly, SQL implements properties of ACID (atomicity, consistency, isolation, and durability) on the dataset. As described by Ganesh Chandra Deka (2015, 15), the ACID principles include: “Atomicity – All or nothing of the N actions (commit or rollback). Consistency – Transactions never observe or cause inconsistent data. Isolation – Transactions are not aware of concurrent transactions. Durability – Acknowledged transactions persist in all events”. Opting for an alternative database implementation, such as NoSQL (not only SQL), which tolerates temporary database inconsistencies, introduces the possibility that only a portion of a database update or modification may execute. This scenario can result in inconsistency and potentially jeopardize data integrity. Some data experts, akin to Deka (2015), may prefer the alternative BASE properties of NoSQL – optimistic behavior, basically available, soft state, eventually consistent data. However, within the scope of electronic music touring data, the prevailing choice is guided by ACID properties for the aforementioned reasons, in addition to the ability to use indexing. The data in this manuscript has been organized primarily in the third normal form (3NF). This normalization, complementing ACID principles, serves to prevent data anomalies and establishes referential integrity (Ravikiran 2023).

Secondly, as of September 2023, PostgreSQL and MySQL are two of the top four most popular relational database management systems worldwide (Taylor 2023). By utilizing two of the most widely adopted RDBMSs, this ensures accessibility for new users, and reduces vulnerability to roadblocks during personnel changes in data management.

Throughout this manuscript, five queries are examined, three with great detail. The differences in query syntax between PostgreSQL and MySQL are nominal, and this allows for quick adjustments where necessary. Although the syntax differences are minor, significant distinctions between PostgreSQL and MySQL exist in areas such as error logging, ease of data importing, and query speeds, especially when handling certain complexities of queries, and varying data volumes (Klimek and Skublewska-Paszkowska, 2021). A thorough comparison will be conducted to assess how each SQL engine manages these aspects.

Thirdly, implementing relational databases in the cloud offers scalability.

Conventionally, a large portion of a data scientist's workload is conducted locally (Wang 2023). If a booking agency utilizes a local analytics suite confined to a specific office network in one city, it limits the ability to manage, update, or modify the dataset remotely or by multiple users in different locations. Local configurations necessitate more planning, and pose greater challenges than cloud counterparts. With cloud capabilities, cloud-based data management tools leverage the collective computing capabilities of networked machines to facilitate rapid and collaborative analytics around the globe (Wang 2023).

As a booking agency's dataset expands after each successive weekend of shows, cloud database management facilitates the seamless input of an extensive volume of future data. Beyond scalability, cloud tools empower individual users to amplify their impact within an organization by democratizing access to both data and data science tools (Wang 2023). The sales tool developed in this manuscript will eventually be accessible to non-technical booking agents in any location. Cloud computing makes this possible.

Keeping on the topic of scalability, Google Cloud Platform was selected as the cloud computing service to host PostgreSQL and MySQL instances due to its capacity for seamless expansion at an affordable cost for both small and large firms. Google Cloud offers flexible pricing models, including on-demand plans that allow users to pay solely for the services they require during their usage period (Chouliaras and Sotiriadis, 2023).

At present, this booking tool incorporates useful features such as data storage, geospatial data analysis, and analytics-motivated queries on a normalized database. The database, conforming to the third normal form (3NF), necessitates a specific extract, transform and load (ETL) strategy for the importing of new data. This technique involves the extraction of new data from a dynamic staging document (Google Sheet), followed by the transformation of the unnormalized data into the appropriate 3NF dimensions for cloud storage and analysis.

Once this tool is deployed, non-technical agents will copy-and-paste their new data into this unnormalized Google Sheet, after which a data engineer will oversee the transformation process into 3NF and the subsequent loading into Google Cloud. However, in the future, parameters can be established to automate this ETL process within Google Cloud, tailored to the unique scaling requirements of each organization (Chouliaras and Sotiriadis, 2023). The prospect of removing this staging document through the utilization of Google Cloud infrastructure underscores the distinctive capabilities offered by this platform. A detailed, sequential explanation of the current data import and update strategy is given below.

Chapter 4: Methodology

Data Collection and Preparation

Currently, as every negotiation becomes finalized, and each event is confirmed and contracted, it becomes a record in a large Google Sheets document, which this manuscript will refer to moving forward as the Annual Bookings Sheet (ABS). This aggregation of all bookings begins as one concentrated table in unnormalized form (UNF). The ABS is separated by tabs of each year, with each tab containing the following columns: ArtistName, Performance Fee, Commission, VenueName, City, ShowDate, DepositSchedule, and PromoterName. Here is a brief explanation of each feature:

ArtistName – the stage name of the musician.

(Performance) Fee – this is the dollar amount (all shows, regardless of country, are always contracted in United States Dollars or USD) the artist has earned for the show. In electronic music bookings, most shows have a set fee, or "flat guarantee," that is decided during the negotiation. Other events have bonus structures if the event goes well, or "versus deals" upon which the artist's final fee is either a flat guarantee or (versus) a percentage of the net box office receipts (NBOR) of the venue and promoter.

Commission – the percentage of the performance fee that the agent and agency collect on each booking. This percentage normally ranges from 10-15% based upon whether or not the agency is organizing and booking the artist's travel, logistics, and creating their itinerary.

VenueName – the name of the club, festival, or event space in which the event is taking place.

Capacity – the maximum amount of people permitted to attend the venue's events. This number is determined by local health and safety regulations.

City – the name of the city where the event is taking place. An important consideration for this feature is that some records display the approximate city/metropolitan area of the event, even if the event is outside of the city limits. Events that are within approximately 50 miles of a major metropolitan city are, for the intent and purpose of this manuscript, considered events within the city's "market."

ShowDate – the day or night the event begins. An important consideration for this feature is that a common start time of an electronic music artist's performance is midnight, 12:00am. This is technically the following day. It is common practice to contract an event on the evening the event begins. For example, if an artist performed at midnight on the evening of Saturday, August 7, 2021 (technically 12:00am on Sunday, August 8, 2021), the event contract and the record in the ABS is Saturday, August 7, 2021 (08-07-2021).

DepositSchedule – the only feature in varying sentence form, this denotes all payment due dates for each event.

PromoterName – The name of the promotional company producing and promoting the event. Additional details like talent buyer names, email addresses, and phone numbers are redundant in the scope of this manuscript, so they have been omitted.

Until now, the ABS has intentionally remained unnormalized, mainly due to the relatively small size of the dataset; each year consists of approximately 400-600 bookings (records). The dataset this manuscript will consider consists of 2,601 bookings from 2018 to 2023.

Omitting Financials From the Dataset

The following financial features of each booking have been omitted to protect confidentiality and non-disclosure agreements in place among artists, promoters, and agency: (Performance) Fee, Commission, and DepositSchedule. With these features omitted, this manuscript is now suitable for public dissemination.

While financials are a critical part of any touring negotiation, the predictive analysis this manuscript strives to achieve can be done without disclosing the artists' performance fees.

When an artist is experiencing a meteoric rise, their ticket sales and performance fee from their previous play in a specific market is less relevant in booking their next tour date than it is for artists with a more consistent ticket draw. This is due to several other factors, including but not limited to: Spotify/Soundcloud/Apple Music streams by region, social media followers by region, and virality of recent social media content. While those factors go beyond the scope of this manuscript, the features this manuscript *does* take into great account – promoter, venue, capacity – become significantly more relevant in deciding an appropriate performance fee.

In Chapter 5, this manuscript will examine a case study of an artist who currently has buzz, and is playing larger venues as his profile continues to grow. The growth of the average venue capacity where an artist plays is directly related to the growth of the artist's performance fee. Therefore, for analyzing artist growth while making future touring decisions, this manuscript favors a publicly available parameter like venue capacity over a confidential one like performance fee.

Converting the ABS to Third Normal Form (3NF)

Until the advent of this application, the ABS remained in unnormalized form, separated by year only, with all features in one table. In order to create the most optimal relational database schema design, the dataset will be converted into third normal form (3NF). This conversion will ensure referential integrity, eliminate data anomalies, and reduce data duplication (Ravikiran 2023).

Abiding by the principles of 3NF led the ABS to quite naturally convert into four distinct tables, listed in the entity relationship diagram (ERD) below:

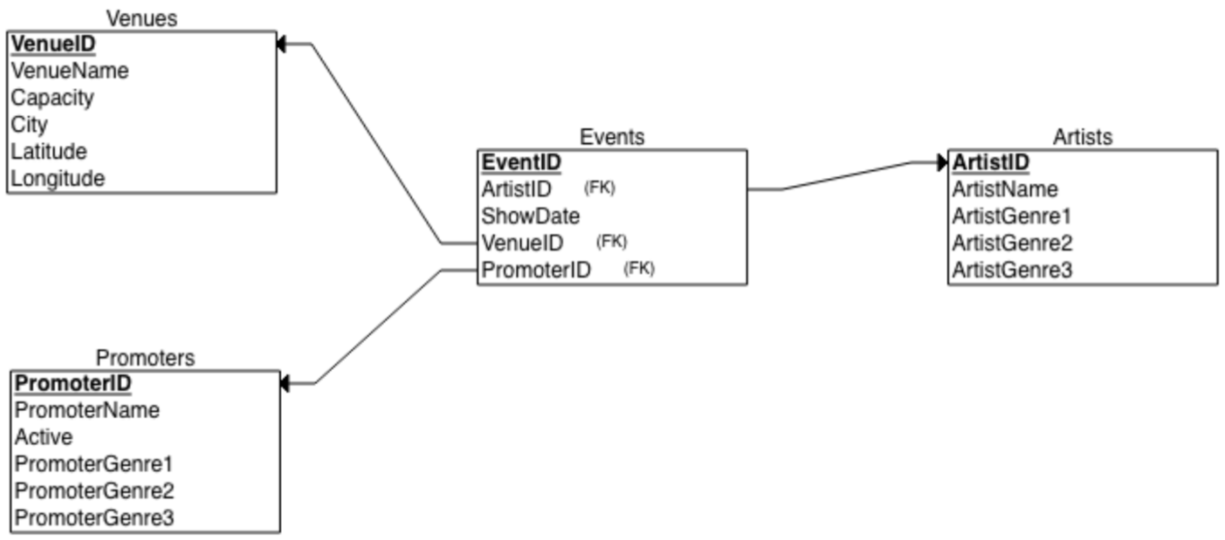


Figure A. Entity Relationship Diagram (ERD) of the Bookings Database.

The Events table in the above ERD acts as the base table, or base relation, as it is the foundation for building relationships with all other tables in the database. By centralizing the Events table with the 'EventID' primary key and three foreign keys – 'ArtistID,' 'VenueID,' and 'PromoterID,' all 2601 event records can efficiently link additional information about each event record to other tables.

The dataset encounters a minor caveat that prevents it from achieving true third normal form: there is no cities table. This is intentional for two reasons.

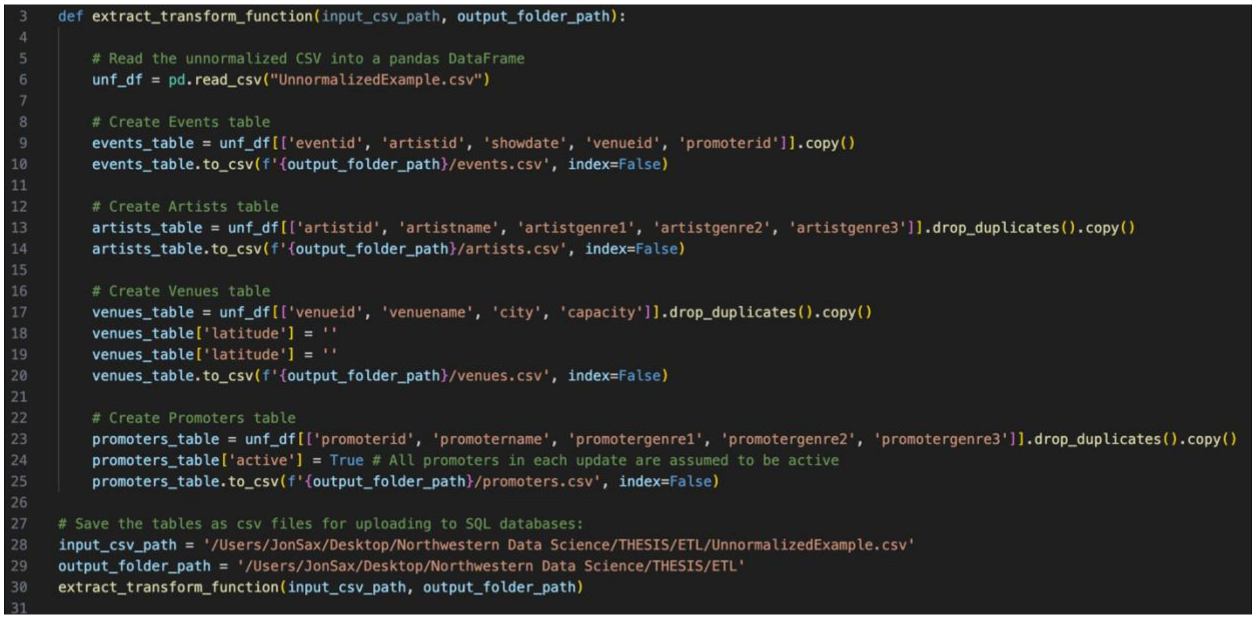
Firstly, there are only two situations in which the absence of a unique primary key 'cityid' could affect the data's integrity: there are events in San Jose, California, and San Jose, Costa Rica, as well as Santa Cruz, California, and Santa Cruz, Bolivia. To remedy these extremely rare occurrences, the author hard-coded the correct city coordinates. Secondly, if a fifth table was created within this relatively small dataset, there would need to be an additional JOIN for nearly every query involving venue and city data. Therefore, the decision to make the Venues table slightly denormalized strikes an ideal balance between normalization and performance, based on this specific use case.

Extract, Transform, Load (ETL)

To extract, transform, and load the unnormalized ABS Google Sheet to a 3NF database in SQL, a careful series of sequential steps are implemented.

Firstly, all relevant columns of the ABS are copied and pasted into a separate interim Google Sheet. This first step is taken to prevent any disruption to the original ABS. The interim Google Sheet does not contain columns that are irrelevant to the SQL analysis like ‘DepositSchedule,’ ‘Fee,’ and ‘Commission.’ Additionally, the original ABS does not possess columns that will eventually become relevant to the SQL analysis like ‘active’, ‘latitude’, and ‘longitude.’ A full explanation for the addition of all of these columns, and how they improve the overall analysis, is provided immediately after this ETL commentary.

In order to make the data import into SQL as smooth as possible, each table’s primary key data is extracted and transformed into .csv files. Additional empty columns are added where necessary in order to replicate the exact column dimensions of the final 3NF table as it exists in the SQL database. To achieve this, the below Python function is written to execute the extract and transform stages of the data migration:

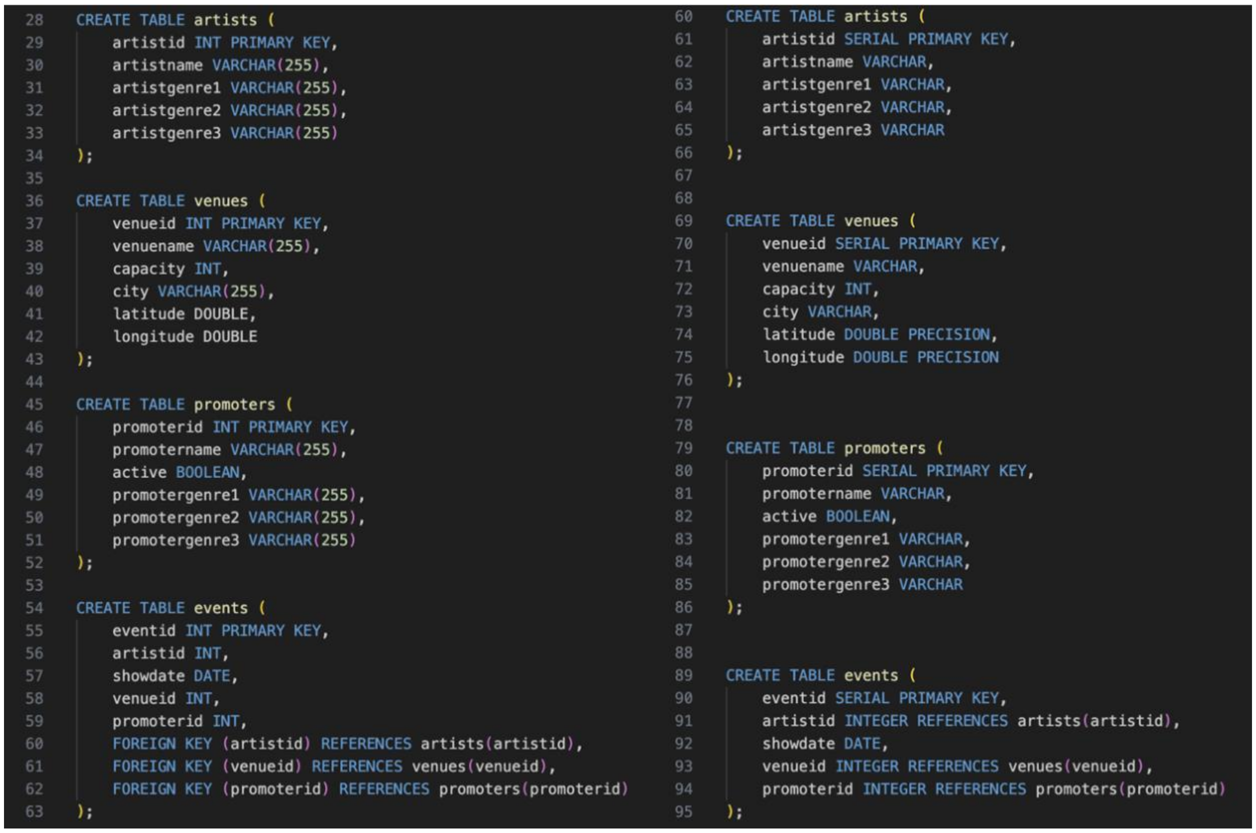


```python
def extract_transform_function(input_csv_path, output_folder_path):

    # Read the unnormalized CSV into a pandas DataFrame
    unf_df = pd.read_csv("UnnormalizedExample.csv")

    # Create Events table
    events_table = unf_df[['eventid', 'artistid', 'showdate', 'venueid', 'promoterid']].copy()
    events_table.to_csv(f'{output_folder_path}/events.csv', index=False)

    # Create Artists table
    artists_table = unf_df[['artistid', 'artistname', 'artistgenre1', 'artistgenre2', 'artistgenre3']].drop_duplicates().copy()
    artists_table.to_csv(f'{output_folder_path}/artists.csv', index=False)

    # Create Venues table
    venues_table = unf_df[['venueid', 'venuename', 'city', 'capacity']].drop_duplicates().copy()
    venues_table['latitude'] = ''
    venues_table['latitude'] = ''
    venues_table.to_csv(f'{output_folder_path}/venues.csv', index=False)

    # Create Promoters table
    promoters_table = unf_df[['promoterid', 'promotername', 'promotergenre1', 'promotergenre2', 'promotergenre3']].drop_duplicates().copy()
    promoters_table['active'] = True # All promoters in each update are assumed to be active
    promoters_table.to_csv(f'{output_folder_path}/promoters.csv', index=False)

# Save the tables as csv files for uploading to SQL databases:
input_csv_path = '/Users/JonSax/Desktop/Northwestern Data Science/THESIS/ETL/UnnormalizedExample.csv'
output_folder_path = '/Users/JonSax/Desktop/Northwestern Data Science/THESIS/ETL'
extract_transform_function(input_csv_path, output_folder_path)
```

Figure B. In Python, the extract_transform_function converts the unnormalized Google Sheet document into four .csv files ahead of the data import into each SQL database.

Note that, unlike the other two tables, the events_table and artists_table are already in third normal form after the extract_transform_function is executed; no new columns within those tables are added in the body of the function. Meanwhile, the geocode columns for the venues_table and 'active' column in the promoters_table are added during the extract_transform_function.

With four distinct .csv tables created, the loading phase of the data migration can begin. In both PostgreSQL and MySQL, CREATE TABLE functions were written with slightly different syntaxes to denote primary and foreign keys. The code for both write operations is displayed below:



```
CREATE TABLE artists (
    artistid INT PRIMARY KEY,
    artistname VARCHAR(255),
    artistgenre1 VARCHAR(255),
    artistgenre2 VARCHAR(255),
    artistgenre3 VARCHAR(255)
);

CREATE TABLE venues (
    venueid INT PRIMARY KEY,
    venuename VARCHAR(255),
    capacity INT,
    city VARCHAR(255),
    latitude DOUBLE,
    longitude DOUBLE
);

CREATE TABLE promoters (
    promoterid INT PRIMARY KEY,
    promotername VARCHAR(255),
    active BOOLEAN,
    promotergenre1 VARCHAR(255),
    promotergenre2 VARCHAR(255),
    promotergenre3 VARCHAR(255)
);

CREATE TABLE events (
    eventid INT PRIMARY KEY,
    artistid INT,
    showdate DATE,
    venueid INT,
    promoterid INT,
    FOREIGN KEY (artistid) REFERENCES artists(artistid),
    FOREIGN KEY (venueid) REFERENCES venues(venueid),
    FOREIGN KEY (promoterid) REFERENCES promoters(promoterid)
);
```

```
CREATE TABLE artists (
    artistid SERIAL PRIMARY KEY,
    artistname VARCHAR,
    artistgenre1 VARCHAR,
    artistgenre2 VARCHAR,
    artistgenre3 VARCHAR
);


CREATE TABLE venues (
    venueid SERIAL PRIMARY KEY,
    venuename VARCHAR,
    capacity INT,
    city VARCHAR,
    latitude DOUBLE PRECISION,
    longitude DOUBLE PRECISION
);


CREATE TABLE promoters (
    promoterid SERIAL PRIMARY KEY,
    promotername VARCHAR,
    active BOOLEAN,
    promotergenre1 VARCHAR,
    promotergenre2 VARCHAR,
    promotergenre3 VARCHAR
);


CREATE TABLE events (
    eventid SERIAL PRIMARY KEY,
    artistid INTEGER REFERENCES artists(artistid),
    showdate DATE,
    venueid INTEGER REFERENCES venues(venueid),
    promoterid INTEGER REFERENCES promoters(promoterid)
);
```

Figure C. PostgreSQL import code (left) and MySQL import code (right).

Several data imports were administered in local and cloud environments. Additionally, imports were administered with and without indexing. There were a great deal of items to keep organized during the import process. In order to maintain atomicity, all CREATE TABLE statements were written in a single query for each database environment. While achieving readability, maintainability, and avoiding data duplication, this one-query strategy during the load stage of ETL also ensured data consistency and integrity.

With all eight databases now containing all necessary data, dependencies, and column dimensions, the reasoning behind the addition of each new feature will now be described in detail.

Adding Genre Column

The first additional feature is adding the genres of music that artists release, and genres of music that promoters book, based on the artists' style. Deciding which genres to earmark for which artists can be an extremely controversial and subjective task. To make it as objective as possible, this manuscript employs genres that Beatport.com has designated for each artist's music. Beatport.com is widely considered a de facto governing body of electronic music genres. According to iMusician.com, "Beatport holds the largest repertoire of downloadable dance music and is considered the go-to marketplace for DJs to purchase tracks for their set" (Mill 2021). In the context of genre selection, "there are Beatport-only curated genres, that means no distributor can deliver content to those genres, it's up to Beatport's curation team to tag the content for those genres. That happens to ensure the right music falls under those genres and to avoid wrong-tagging" (Mill 2021).

It is important to note that there were no genre columns originally included in the unnormalized ABS. To ensure accuracy during the population of genre columns, the addition of these features was implemented within the staging Google Sheet, before the ETL phase.

With the arrival of genre classifications, deeper analytical capabilities within SQL are now possible. By zeroing in on multiple genres that each artist plays, there is considerable improvement in the precision of future touring recommendations and strategy. Three new features were added to the ABS to eventually be included in the Artists table: 'ArtistGenre1,' 'ArtistGenre2,' and 'ArtistGenre3,' which are the artist's first, second, and third most frequently-released genres on Beatport within the last two years[1]. For artists who have not released at least

[1] Given the propensity for artists to change their styles, only music released within the last two years was factored into the population and ranking of genre data.

three genres of music on Beatport, the remaining genre values were imputed based upon their style of DJing.

Similar to artist genres, 'PromoterGenre1,' 'PromoterGenre2,' and 'PromoterGenre3' are also methodically selected. The first, second, and third promoter genre rankings are a product of the artists they book, and the genres upon which those artists release music. These genres were extracted in Google Sheets by using a combination of 'INDEX' and 'MATCH' functions. Here is an example of the code to display 'PromoterGenre1' of the first promoter alphabetically in the Promoters table, 1015 Folsom in San Francisco:

```
=INDEX(Artists!$C$2:$E$47,MATCH(MAX(COUNTIFS(Events!$H$2:$H$2602,$B2,
Events!$B$2:$B$2602,
Artists!$A$2:$A$47)),COUNTIFS(Events!$H$2:$H$2602,$B2,Events!$B$2:$B$2602,Art
ists!$A$2:$A$47),0))
```

The formula is searching for genre values in the Artists table based on the maximum count of occurrences in the Events table. In situations where a promoter has only booked one artist in the database, these promoter genres will display that particular artist's top three genres. For promoters who have booked multiple artists in the database, the top three genres corresponding to their profile are a ranking of their top three most-booked artist genres. From here, there is now a robust dataset of 19 different genres.

Adding Active Column

At any given moment, promoters can come into existence, or shut down for good. Venues are the same. In fact, generally speaking, venues tend to have higher turnover than promoters. The purpose of creating this database is to make recommendations for future touring. There is an important reason the Active column exists is in the Promoters table, and not the Venues table. For one, it would be redundant and time consuming to have both. But in selecting between which table to add this parameter to, it is important to think like an agent. When the early plans of a tour commence, the first discussion between client(s) (artist and artist manager) and agent is to decide which critical markets (cities) need to be on the tour. And while city is a feature in the Venues table, a more important determinant contributing to the success of an event is not the venue, but rather, the promoter producing the event. There have been times when great artists play at the right venues, but for the wrong promoter. Promoters bring subsets of crowds. Putting artists in front of the correct subset is something promoters know and do best. Therefore, a very influential judgment call an agent makes in sending artists to key markets is which promoters to play for. That positioning is, in most cases, more important than the venue itself. To this degree, sometimes it is favorable for underground artists to play completely unmarked warehouse spaces. Access and permitting to these spaces are responsibilities of the promoter. Therefore, in the context of querying this database, whether or not a promoter is active is more important than whether or not a venue is. That said, there are times when agents urge promoters to seek out particular venues, but generally speaking, it is less common.

Adding Latitude and Longitude Columns

Many geospatial libraries for Python rely upon the geography data type, which requires latitude and longitude coordinates (Debarros 2022, 280). These two additional columns were added to the Venues table using convert tasks provided by the Mapping Sheets add-on in Google Sheets. The latitude and longitude columns were then imported as double-precision floating point data types into each SQL database.

The addition of geospatial data is important for two reasons. Firstly, it unlocks the ability to create queries using geospatial parameters to display results with heightened geographical precision, and thus, drive better decision making through optimized tabular results. Secondly, using these tabular results, an agent can then create compelling and interactive data visualizations that not only improve workflows, but convey work in a clear and easily understandable way to stakeholders (artists and managers).

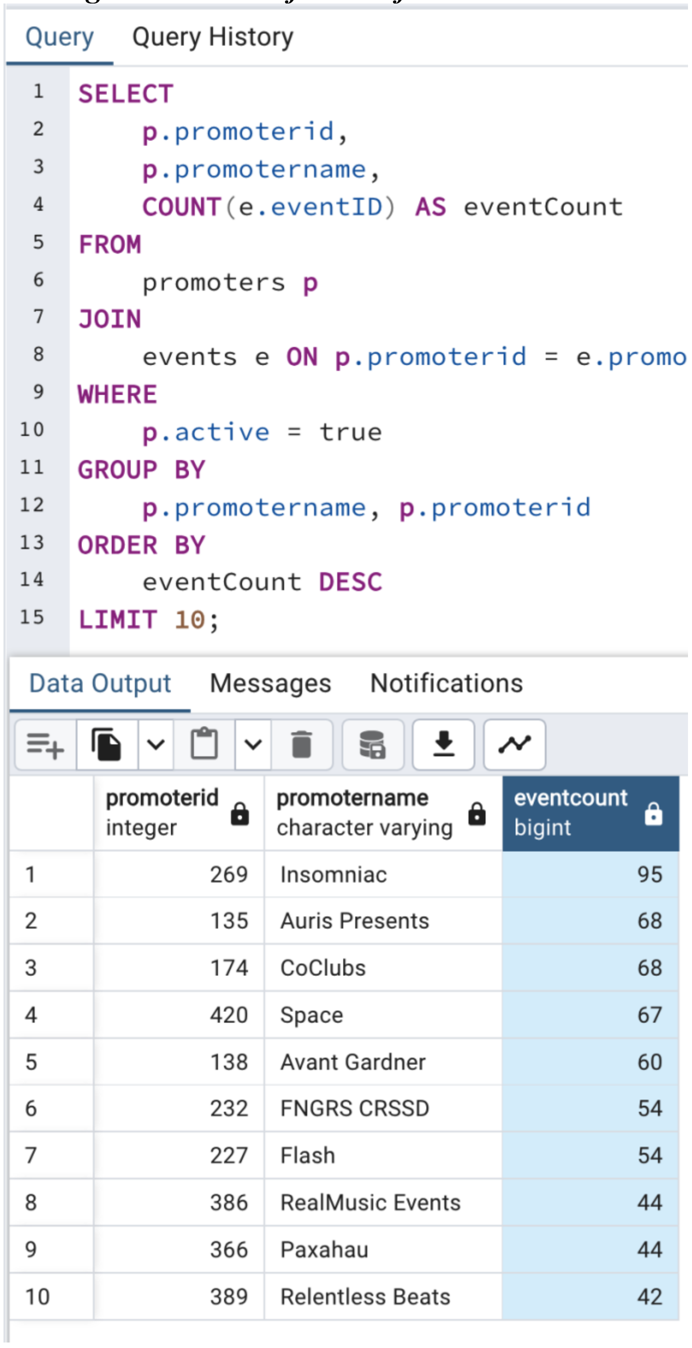
Having concluded the explanation of methodology, the following section will delve into exploratory data analysis (EDA), data visualizations, and valuable insights extracted from this database architecture.

Chapter 5: Analytics

Exploratory Data Analysis (EDA) and Useful Queries

The entire data set has been cleaned, converted to 3NF, and is now prepared for querying and eventual analysis. Below are four business questions that are useful to agents, and answered by querying this custom dataset.

Query 1: ***Who are the Top 10 active promoters with the highest amount of events from 2018-2023?***



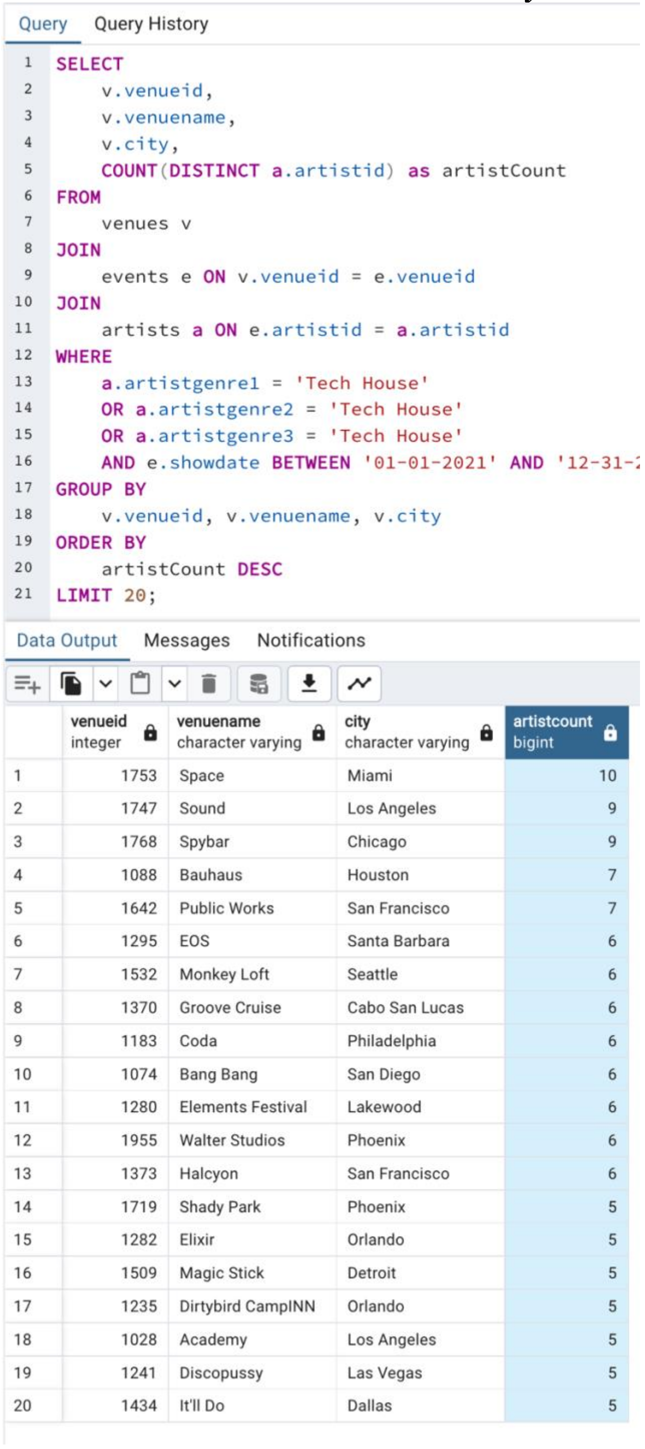
```sql
SELECT
    p.promoterid,
    p.promotername,
    COUNT(e.eventID) AS eventCount
FROM
    promoters p
JOIN
    events e ON p.promoterid = e.promo
WHERE
    p.active = true
GROUP BY
    p.promotername, p.promoterid
ORDER BY
    eventCount DESC
LIMIT 10;
```

	promoterid integer	promotername character varying	eventcount bigint
1	269	Insomniac	95
2	135	Auris Presents	68
3	174	CoClubs	68
4	420	Space	67
5	138	Avant Gardner	60
6	232	FNGRS CRSSD	54
7	227	Flash	54
8	386	RealMusic Events	44
9	366	Paxahau	44
10	389	Relentless Beats	42

Figure D. SQL code and data output for Query 1.

This first query is important because, from an analytics perspective, it displays a list of active talent buyers over the entire duration of the dataset with whom the agent most frequently does business. For guiding future business strategy, this query is helpful because it shows the agent's accounts (promoters who are currently active) with the highest buying frequency and in turn, are highly probable to create future business.

Query 2: ***Which are the Top 20 venues hosting the highest amount of Tech House artists within the last two years?***



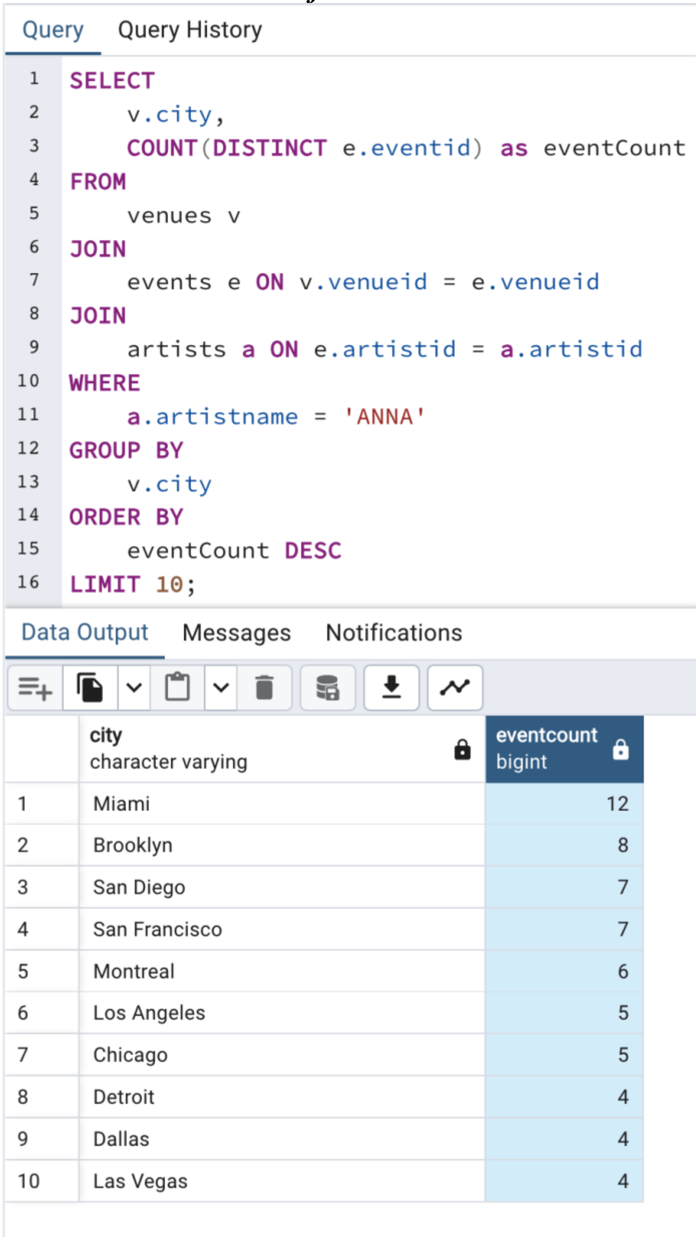
```
SELECT
    v.venueid,
    v.venuename,
    v.city,
    COUNT(DISTINCT a.artistid) as artistCount
FROM
    venues v
JOIN
    events e ON v.venueid = e.venueid
JOIN
    artists a ON e.artistid = a.artistid
WHERE
    a.artistgenre1 = 'Tech House'
    OR a.artistgenre2 = 'Tech House'
    OR a.artistgenre3 = 'Tech House'
    AND e.showdate BETWEEN '01-01-2021' AND '12-31-
GROUP BY
    v.venueid, v.venuename, v.city
ORDER BY
    artistCount DESC
LIMIT 20;
```

	venueid integer	venuename character varying	city character varying	artistcount bigint
1	1753	Space	Miami	10
2	1747	Sound	Los Angeles	9
3	1768	Spybar	Chicago	9
4	1088	Bauhaus	Houston	7
5	1642	Public Works	San Francisco	7
6	1295	EOS	Santa Barbara	6
7	1532	Monkey Loft	Seattle	6
8	1370	Groove Cruise	Cabo San Lucas	6
9	1183	Coda	Philadelphia	6
10	1074	Bang Bang	San Diego	6
11	1280	Elements Festival	Lakewood	6
12	1955	Walter Studios	Phoenix	6
13	1373	Halcyon	San Francisco	6
14	1719	Shady Park	Phoenix	5
15	1282	Elixir	Orlando	5
16	1509	Magic Stick	Detroit	5
17	1235	Dirtybird CampINN	Orlando	5
18	1028	Academy	Los Angeles	5
19	1241	Discopussy	Las Vegas	5
20	1434	It'll Do	Dallas	5

Figure E. SQL code and data output for Query 2.

By zeroing in on locations that book a certain electronic music genre, an agent can group certain subsets of artists, and exclusively reach out to promoters who book at these top venues for that genre. As previously discussed, venues tend to come and go more frequently than promoters. Limiting this query to only the last two years of data, enhances the results by weeding out older bookings at venues that may have either lost relevance or closed recently.

Query 3 (small query): ***Which Top 10 cities has ANNA performed in the most from 2018-2023?***



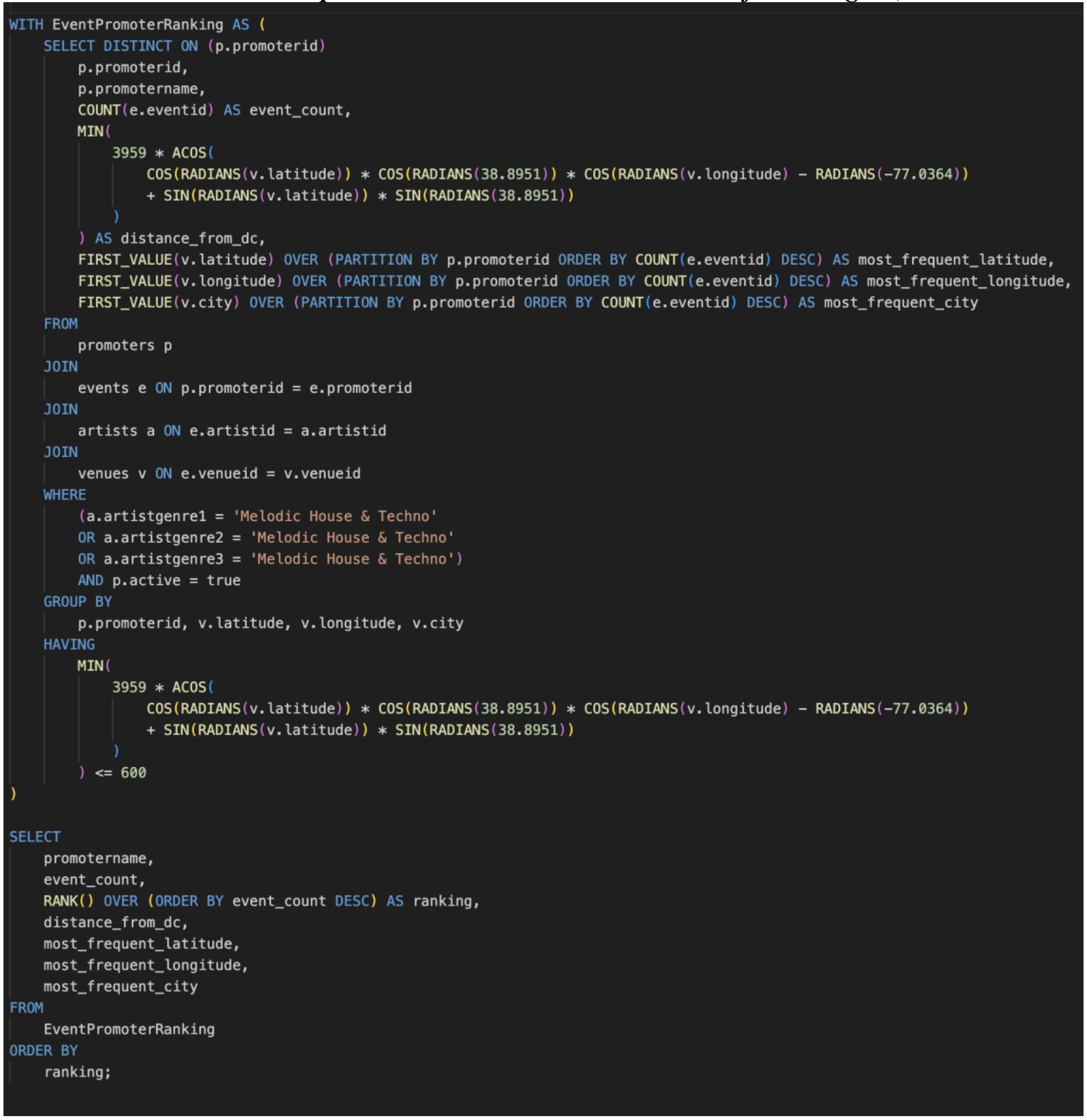
```
SELECT
    v.city,
    COUNT(DISTINCT e.eventid) as eventCount
FROM
    venues v
JOIN
    events e ON v.venueid = e.venueid
JOIN
    artists a ON e.artistid = a.artistid
WHERE
    a.artistname = 'ANNA'
GROUP BY
    v.city
ORDER BY
    eventCount DESC
LIMIT 10;
```

	city character varying	eventcount bigint
1	Miami	12
2	Brooklyn	8
3	San Diego	7
4	San Francisco	7
5	Montreal	6
6	Los Angeles	5
7	Chicago	5
8	Detroit	4
9	Dallas	4
10	Las Vegas	4

Figure F. SQL code and data output for Query 3

Every artist has a custom-tailored tour schedule based on which markets (cities) are the most important to frequent for profile implications. Just as important is the skillful restraint of not playing these key markets too often. Query 3 is useful because it shows which markets have the highest demand for ANNA performances, while also demonstrating that Miami – as the busiest market in the entire dataset – is an exception to typical tour strategy. The reason Miami has 50% more events than any other market in ANNA's list is due to two paramount week-long festivals/conferences in the region in March and December respectively: Miami Music Week and Art Basel. During these times of year, artists tend to play several times throughout the week, making Miami unlike any other region in the dataset. This manuscript will refer to this query as Query 3 or the small query interchangeably, as it is used in later analysis.

Query 4 (large query): ***Locate, identify, and rank by booking frequency, all active Melodic House & Techno promoters within a 600-mile radius of Washington, DC?***



```sql
WITH EventPromoterRanking AS (
    SELECT DISTINCT ON (p.promoterid)
        p.promoterid,
        p.promotername,
        COUNT(e.eventid) AS event_count,
        MIN(
            3959 * ACOS(
                COS(RADIANS(v.latitude)) * COS(RADIANS(38.8951)) * COS(RADIANS(v.longitude) - RADIANS(-77.0364))
                + SIN(RADIANS(v.latitude)) * SIN(RADIANS(38.8951))
            )
        ) AS distance_from_dc,
        FIRST_VALUE(v.latitude) OVER (PARTITION BY p.promoterid ORDER BY COUNT(e.eventid) DESC) AS most_frequent_latitude,
        FIRST_VALUE(v.longitude) OVER (PARTITION BY p.promoterid ORDER BY COUNT(e.eventid) DESC) AS most_frequent_longitude,
        FIRST_VALUE(v.city) OVER (PARTITION BY p.promoterid ORDER BY COUNT(e.eventid) DESC) AS most_frequent_city
    FROM
        promoters p
    JOIN
        events e ON p.promoterid = e.promoterid
    JOIN
        artists a ON e.artistid = a.artistid
    JOIN
        venues v ON e.venueid = v.venueid
    WHERE
        (a.artistgenre1 = 'Melodic House & Techno'
        OR a.artistgenre2 = 'Melodic House & Techno'
        OR a.artistgenre3 = 'Melodic House & Techno')
        AND p.active = true
    GROUP BY
        p.promoterid, v.latitude, v.longitude, v.city
    HAVING
        MIN(
            3959 * ACOS(
                COS(RADIANS(v.latitude)) * COS(RADIANS(38.8951)) * COS(RADIANS(v.longitude) - RADIANS(-77.0364))
                + SIN(RADIANS(v.latitude)) * SIN(RADIANS(38.8951))
            )
        ) <= 600
)

SELECT
    promotername,
    event_count,
    RANK() OVER (ORDER BY event_count DESC) AS ranking,
    distance_from_dc,
    most_frequent_latitude,
    most_frequent_longitude,
    most_frequent_city
FROM
    EventPromoterRanking
ORDER BY
    ranking;
```

Figure G. SQL code for Query 4 (large query).

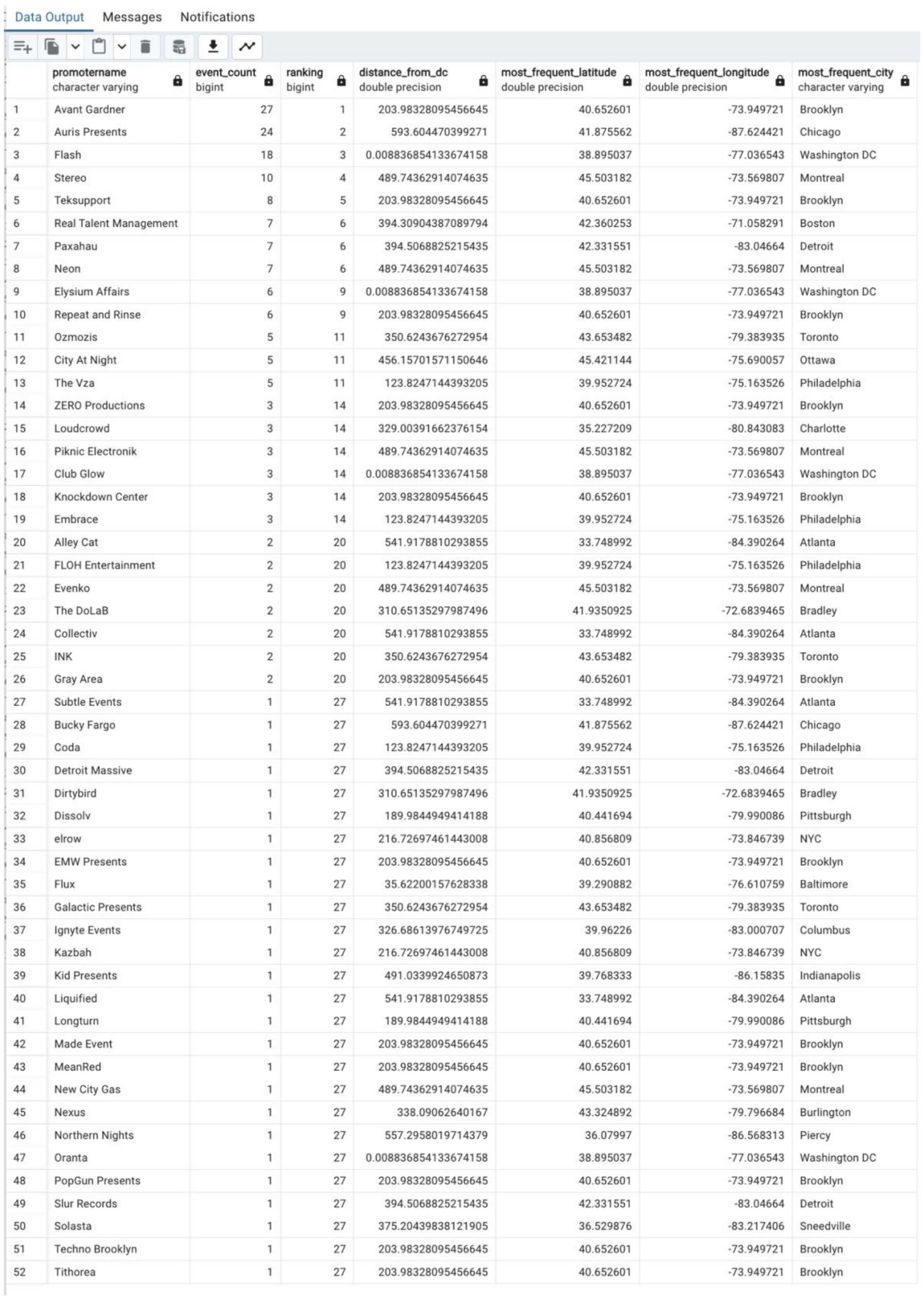
Query 4, which will also be referred to as the large query in later analysis, is the most powerful and important query of the manuscript. It serves as the primary focal point of two pivotal aspects of this thesis: the derivation and comparison of database systems, and the answering of the business question and justification. From a business perspective, this large query allows agents to group promoters by booking frequency of a certain genre, specifying their geographic distance from a particular location. In order to measure where a promoter generally books (recall, the promoters table does not have any location data), three additional criteria were created: ‘most_frequent_latitude’, ‘most_frequent_longitude’, and from this data, a ‘most_frequent_city parameter’ is created in order to communicate to the agent where each of these promoters books these particular artists the most. This large query joins all four tables of the dataset in a way that not only returns an ‘event_count’ ranking among other similar promoters, and most-frequented city by promoters, but it also displays the distance each of these most-frequented cities is from Washington, DC. Here is an example business situation that highlights the importance and usefulness of this particular query.

If one month from today, a Europe-based artist who plays mostly Melodic House & Techno has a three-event weekend taking place over a Friday, Saturday, and Sunday, the events are almost certainly announced and on sale now. In the context of this query, it is assumed that one of the events is canceled for unforeseen circumstances, and one of the two remaining events is in Washington, DC. Query 4 displays a highly customized list of active and nearby promoters whom the agent can quickly reach out to to fill the newly vacant date. This manuscript uses this particular example because this has happened to the author in real life on several occasions, especially recently, as the Melodic House & Techno sound has been very popular on the eastern seaboard of the United States.

Now that the business justification of Query 4 has been laid out, this manuscript will dive deep into the query’s syntax, and all of the geospatial data visualizations the query facilitates.

Breaking Down Query 4 (Large Query)

To start building this query, the creation of a nested query was necessary using a common table expression (CTE) called ‘RankedEventPromoter’. ‘RankedEventPromoter’ is a temporary result that the query references to organize and rank promoters based on how frequently they book Melodic House & Techno artists. Following the creation of this common table expression is a nested query that retrieves each promoter’s event count (‘event_count’), distance from Washington DC (‘distance_from_dc’), and ensures the results only display active promoters. The main query (lines 39-51 of Figure G) at the bottom then selects seven columns from the newly-created common table expression to display the final results, ordered by the ranking (aka booking frequency) of each promoter. In MySQL syntax, ROW_NUMBER() is the window function that assigns the ranking of each promoter ordered by their event count. In PostgreSQL syntax, there are three FIRST_VALUE() window functions. The first one returns the promoter rankings ordered by ‘event_count’. In other words, it ranks the promoters in order of which ones book the highest amount of Melodic House & Techno bookings. The second and third FIRST_VALUE() window functions rank the latitude and longitude of the most frequent locations each promoter books Melodic House & Techno artists. In other words, the primary city in which each promoter books events. Here is the output of the query:



	promotername character varying	event_count bigint	ranking bigint	distance_from_dc double precision	most_frequent_latitude double precision	most_frequent_longitude double precision	most_frequent_city character varying
1	Avant Gardner	27	1	203.98328095456645	40.652601	-73.949721	Brooklyn
2	Auris Presents	24	2	593.604470399271	41.875562	-87.624421	Chicago
3	Flash	18	3	0.008836854133674158	38.895037	-77.036543	Washington DC
4	Stereo	10	4	489.74362914074635	45.503182	-73.569807	Montreal
5	Teksupport	8	5	203.98328095456645	40.652601	-73.949721	Brooklyn
6	Real Talent Management	7	6	394.30904387089794	42.360253	-71.058291	Boston
7	Paxahau	7	6	394.5068825215435	42.331551	-83.04664	Detroit
8	Neon	7	6	489.74362914074635	45.503182	-73.569807	Montreal
9	Elysium Affairs	6	9	0.008836854133674158	38.895037	-77.036543	Washington DC
10	Repeat and Rinse	6	9	203.98328095456645	40.652601	-73.949721	Brooklyn
11	Ozmozis	5	11	350.6243676272954	43.653482	-79.383935	Toronto
12	City At Night	5	11	456.15701571150646	45.421144	-75.690057	Ottawa
13	The Vza	5	11	123.8247144393205	39.952724	-75.163526	Philadelphia
14	ZERO Productions	3	14	203.98328095456645	40.652601	-73.949721	Brooklyn
15	Loudcrowd	3	14	329.00391662376154	35.227209	-80.843083	Charlotte
16	Piknic Electronik	3	14	489.74362914074635	45.503182	-73.569807	Montreal
17	Club Glow	3	14	0.008836854133674158	38.895037	-77.036543	Washington DC
18	Knockdown Center	3	14	203.98328095456645	40.652601	-73.949721	Brooklyn
19	Embrace	3	14	123.8247144393205	39.952724	-75.163526	Philadelphia
20	Alley Cat	2	20	541.9178810293855	33.748992	-84.390264	Atlanta
21	FLOH Entertainment	2	20	123.8247144393205	39.952724	-75.163526	Philadelphia
22	Evenko	2	20	489.74362914074635	45.503182	-73.569807	Montreal
23	The DoLaB	2	20	310.65135297987496	41.9350925	-72.6839465	Bradley
24	Collectiv	2	20	541.9178810293855	33.748992	-84.390264	Atlanta
25	INK	2	20	350.6243676272954	43.653482	-79.383935	Toronto
26	Gray Area	2	20	203.98328095456645	40.652601	-73.949721	Brooklyn
27	Subtle Events	1	27	541.9178810293855	33.748992	-84.390264	Atlanta
28	Bucky Fargo	1	27	593.604470399271	41.875562	-87.624421	Chicago
29	Coda	1	27	123.8247144393205	39.952724	-75.163526	Philadelphia
30	Detroit Massive	1	27	394.5068825215435	42.331551	-83.04664	Detroit
31	Dirtybird	1	27	310.65135297987496	41.9350925	-72.6839465	Bradley
32	Dissolv	1	27	189.9844949414188	40.441694	-79.990086	Pittsburgh
33	elrow	1	27	216.72697461443008	40.856809	-73.846739	NYC
34	EMW Presents	1	27	203.98328095456645	40.652601	-73.949721	Brooklyn
35	Flux	1	27	35.62200157628338	39.290882	-76.610759	Baltimore
36	Galactic Presents	1	27	350.6243676272954	43.653482	-79.383935	Toronto
37	Ignyte Events	1	27	326.68613976749725	39.96226	-83.000707	Columbus
38	Kazbah	1	27	216.72697461443008	40.856809	-73.846739	NYC
39	Kid Presents	1	27	491.0339924650873	39.768333	-86.15835	Indianapolis
40	Liquified	1	27	541.9178810293855	33.748992	-84.390264	Atlanta
41	Longturn	1	27	189.9844949414188	40.441694	-79.990086	Pittsburgh
42	Made Event	1	27	203.98328095456645	40.652601	-73.949721	Brooklyn
43	MeanRed	1	27	203.98328095456645	40.652601	-73.949721	Brooklyn
44	New City Gas	1	27	489.74362914074635	45.503182	-73.569807	Montreal
45	Nexus	1	27	338.09062640167	43.324892	-79.796684	Burlington
46	Northern Nights	1	27	557.2958019714379	36.07997	-86.568313	Piercy
47	Oranta	1	27	0.008836854133674158	38.895037	-77.036543	Washington DC
48	PopGun Presents	1	27	203.98328095456645	40.652601	-73.949721	Brooklyn
49	Slur Records	1	27	394.5068825215435	42.331551	-83.04664	Detroit
50	Solasta	1	27	375.20439838121905	36.529876	-83.217406	Sneedville
51	Techno Brooklyn	1	27	203.98328095456645	40.652601	-73.949721	Brooklyn
52	Tithorea	1	27	203.98328095456645	40.652601	-73.949721	Brooklyn

Figure H. Output of the large query.

The query is dynamic in that the origin location, distance from origin, and genre of the artist are all customizable to help solve virtually any agent conundrum like the example above. A particular point of intrigue in Query 4 is how the results from the query allow for interactive map functionality.

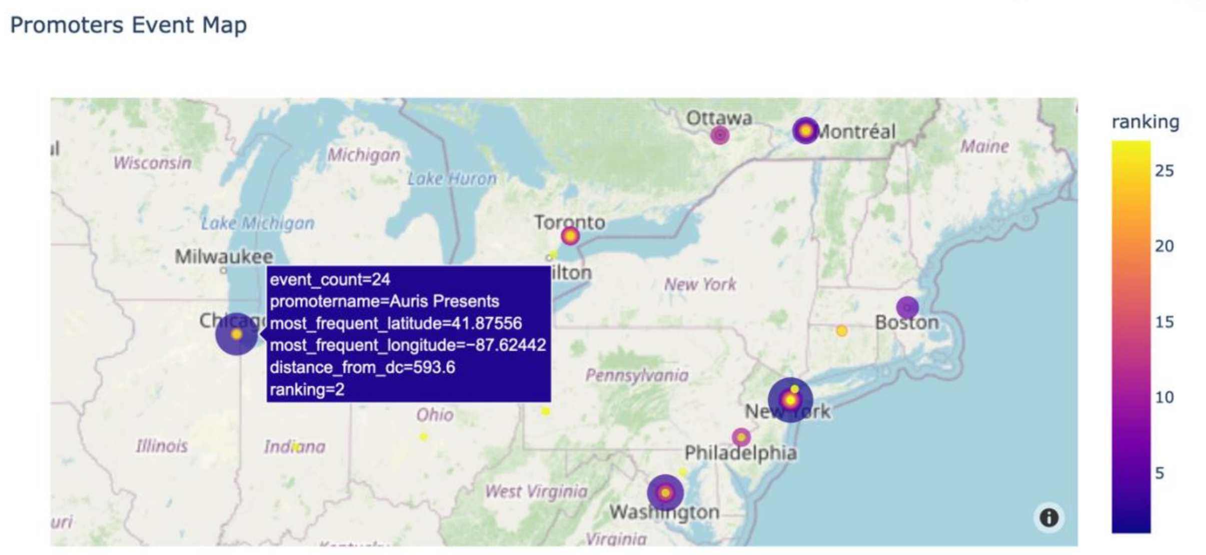
Geospatial Analysis

This extensive list of 52 promoters hand-selected with extreme precision via the large query has already proven to be valuable. However, by leveraging the capabilities of geospatial analysis, this list can come to life digitally in a way that an agent's static map or globe never could. For this, the Python graphic library Plotly is used.

"One of the most deceptively powerful features of interactive [mapping] visualization using Plotly is the ability for the user to reveal more information about a data point by moving their mouse cursor over the point and having a hover label appear" (Plotly.com 2024).

By utilizing Plotly alongside the psycopg2, sqlalchemy, plotly.express and pandas librarires in Python, this manuscript effectively creates an interactive map displaying all 52 active Melodic House & Techno promoters within a 600-mile radius of Washington DC.

The large query is converted from SQL output to a pandas dataframe. The dataframe is then refined to only include parameters relevant to the visualization: 'promotername,' 'event_count,' 'ranking,' 'distance_from_dc,' 'most_frequent_latitude,' 'most_frequent_longitude,' and 'most_frequent_city.' Then the interactive mapping tool is created using the scatter_mapbox function which overlays a scatterplot of promoters on a map using the Mapbox mapping service (Plotly.com 2024). Each hover label displays the above features of each promoter, and the extent of promoters in each market is color-coded on a ranking scale located on the right side of Figure I.



event_count=24
promotername=Auris Presents
most_frequent_latitude=41.87556
most_frequent_longitude=−87.62442
distance_from_dc=593.6
ranking=2

Figure I. Interactive Map Displaying the Results of Query 4

The practicality of this tool cannot be overstated. Nearly every day, agents are tasked with optimizing each client's tour routing based on many of the criteria portrayed in Query 4. With an agent and agency's comprehensive archive of booking data cleaned and imported into this database schema, mapping functionality can bring business queries to life based on highly customizable, individualized parameters to suit an agent's needs. To the author's knowledge, this personalized interactive geospatial tool for agents is the first of its kind in electronic music bookings.

Three additional map visualizations are explored in this manuscript. The first is a general heatmap of all 2,601 events from 2018-2023. A dominant observation of this visual is the density of bookings in the United States as opposed to other countries; nearly ¾ of the dataset is within the United States. In fact, there are more USA bookings than there are bookings in all other countries combined.

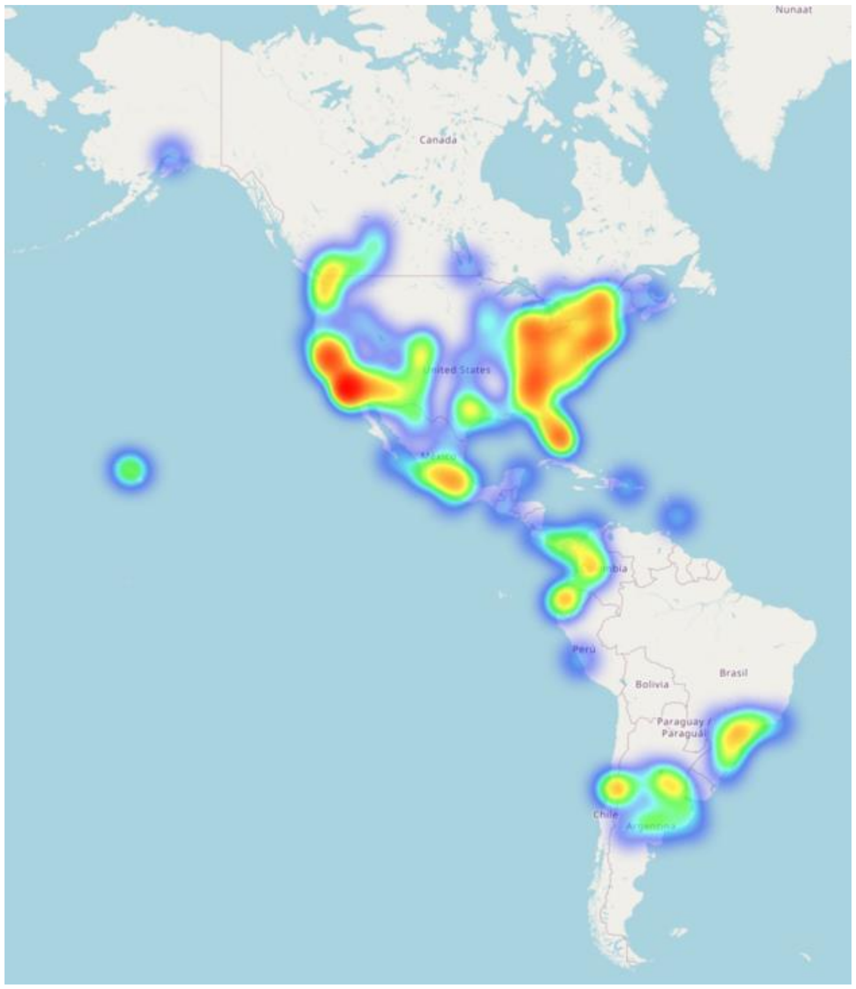


Figure J. Heatmap of North and South America bookings 2018-2023.

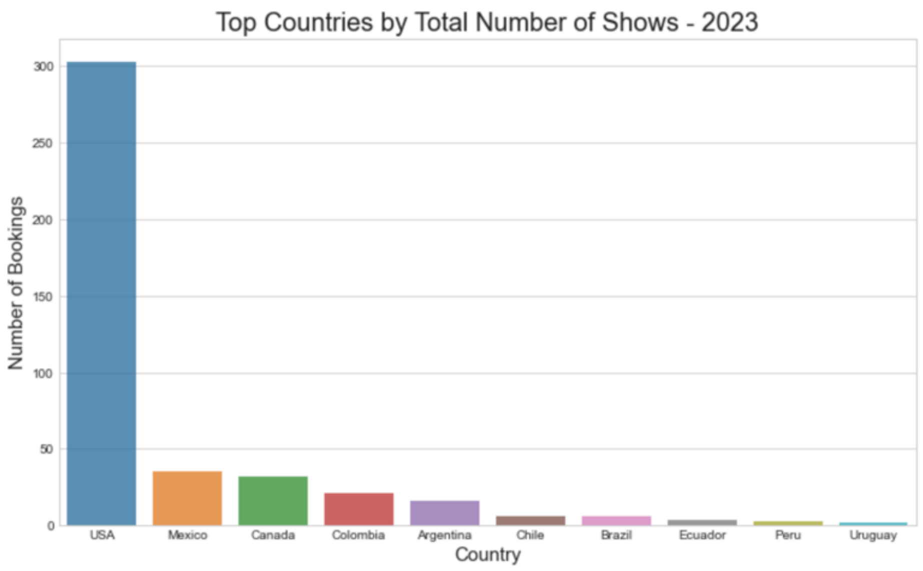


Figure K. Top Countries by Total Number of Shows - 2023.

This is due to a multitude of factors, including but not limited to: the location of the agent and booking agency (San Francisco, CA), the number of domestic artists on the agent's roster, the relatively fewer amount of flights, promoters, and dance music venues in other countries, and the economic conditions in the United States versus other countries in the Americas. Above all, to have global success in electronic music, it is very important to have a strong presence in the United States.

The third geographic visualization is an interactive time series heatmap. Figure L below displays all 2,601 bookings on a geographic basis over time, incrementing by the days in which events took place. The user of this interactive time series heatmap can speed up or slow down the visualization by toggling the frames per second (FPS) dial in the lower right. The user may also navigate forward and backward within the five-year span by clicking and dragging the center button on the time dial.

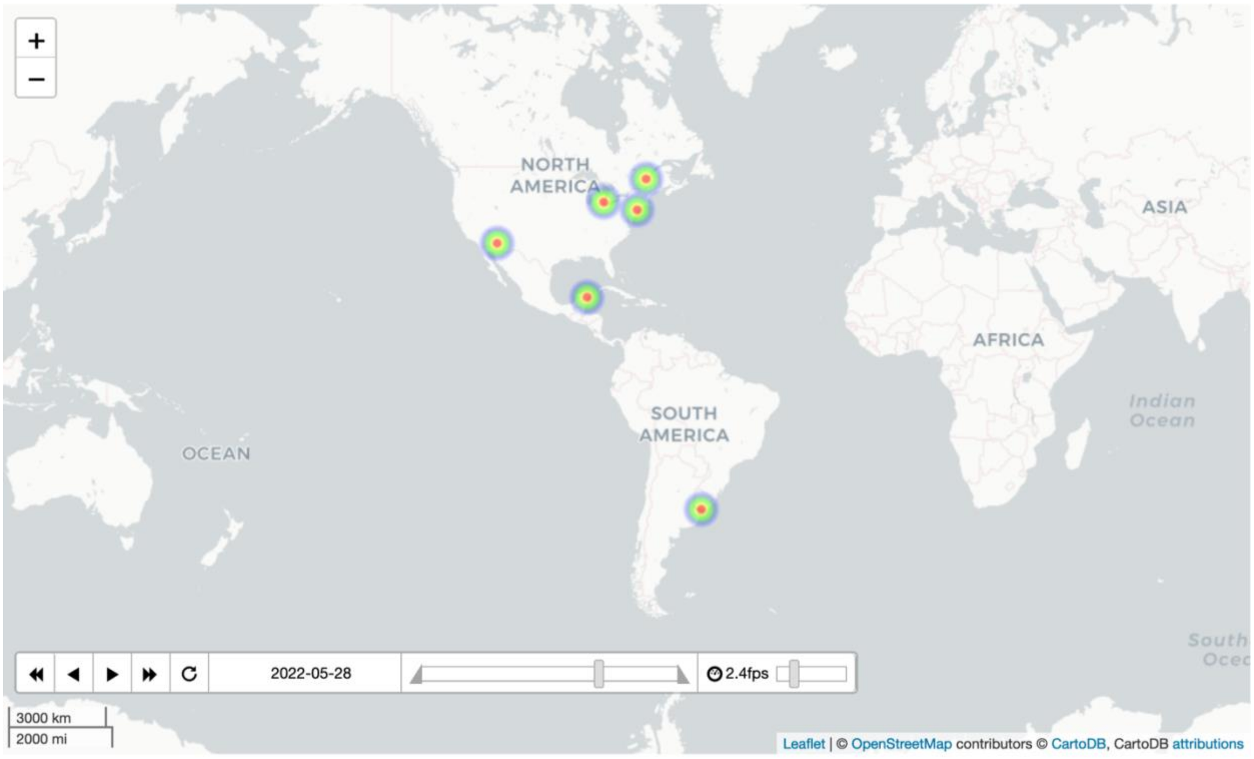


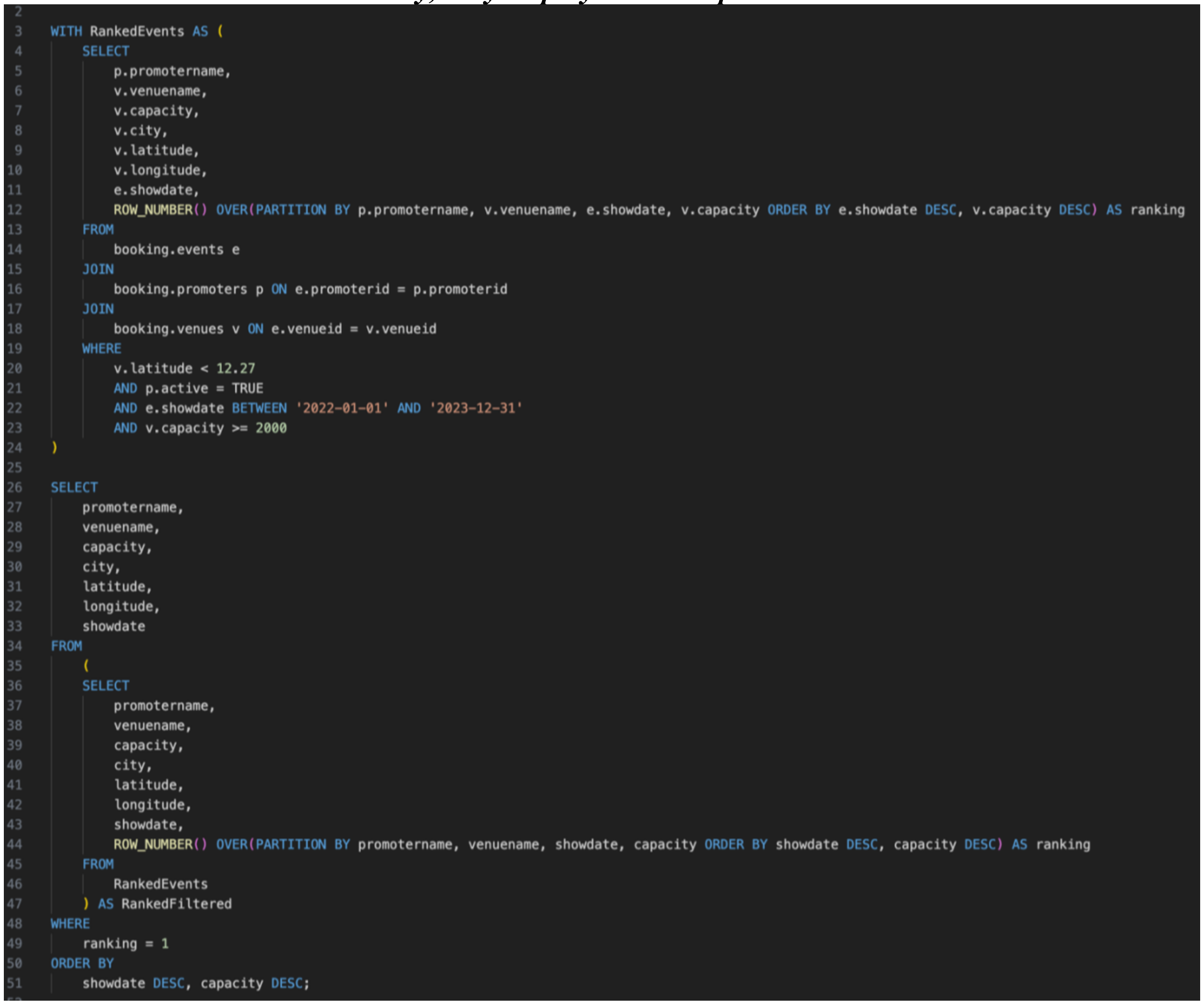
Figure L. Time series heatmap with interactive date and time speed functionality.

To create this visualization, the pandas DataFrame from the static total bookings heatmap (Figure J) was augmented to only display columns ‘showdate,’ ‘latitude,’ and ‘longitude.’ From there, a time index was created as a sorted, chronological list of dates from 2018-01-01 to 2023-12-31 in which at least one event occurred. Next, a `for` loop was written, grouping the DataFrame by ‘showdate’ column and iterating over each group. Within each group, a list of lists was created, with the inner lists containing ‘latitude’ and ‘longitude’ values for events occurring within the corresponding date. Since the majority of dates in the dataset are Fridays and Saturdays, there are normally several events displayed on any given date. This visualization is particularly useful in illustrating the seasonal significance of specific regions in the Americas. Here are some examples of insights this time series heatmap provides:

- Mexico and Colombia have considerably more events around Halloween and Día de los Muertos (Day of the Dead).
- South America has more events during the Southern Hemisphere’s summer months (December, January, and February).
- Central America has the highest amount of events January through April.
- November is a particularly strong month in the United States, and Thanksgiving Eve is a popular night for booking events in the Midwest (eg Chicago and Detroit).

A fifth query was created to (1) highlight the convenience and practicality of interactive geospatial visualizations, and (2) compare query speeds when joining only three of the four tables (versus the large query, which joins all four tables). This time, the query is searching for large venues in South America with recent bookings. Similar to the large query, this SQL statement contains a nested query and window functions.

Query 5 (South America query): ***Display a list of active promoters in South America who have done events within the last two years of 2,000 or larger capacity. List the venue names and dates where and when the events occurred. If multiple artists played the same event on the same day, only display each unique row once.***



```
WITH RankedEvents AS (
    SELECT
        p.promotername,
        v.venuename,
        v.capacity,
        v.city,
        v.latitude,
        v.longitude,
        e.showdate,
        ROW_NUMBER() OVER(PARTITION BY p.promotername, v.venuename, e.showdate, v.capacity ORDER BY e.showdate DESC, v.capacity DESC) AS ranking
    FROM
        booking.events e
    JOIN
        booking.promoters p ON e.promoterid = p.promoterid
    JOIN
        booking.venues v ON e.venueid = v.venueid
    WHERE
        v.latitude < 12.27
        AND p.active = TRUE
        AND e.showdate BETWEEN '2022-01-01' AND '2023-12-31'
        AND v.capacity >= 2000
)

SELECT
    promotername,
    venuename,
    capacity,
    city,
    latitude,
    longitude,
    showdate
FROM
    (
    SELECT
        promotername,
        venuename,
        capacity,
        city,
        latitude,
        longitude,
        showdate,
        ROW_NUMBER() OVER(PARTITION BY promotername, venuename, showdate, capacity ORDER BY showdate DESC, capacity DESC) AS ranking
    FROM
        RankedEvents
    ) AS RankedFiltered
WHERE
    ranking = 1
ORDER BY
    showdate DESC, capacity DESC;
```

Figure M. PostgreSQL code for Query 5 (the South America query).

This query returns a list of 54 events over 2,000 capacity booked by active promoters in South America between January 1, 2022, and December 31, 2023. Like Query 4, the results obtained from Query 5 become particularly valuable when represented geographically. This visualization allows an agent to swiftly access a unique map of large-scale promoters and events and mouseover each location for further details:

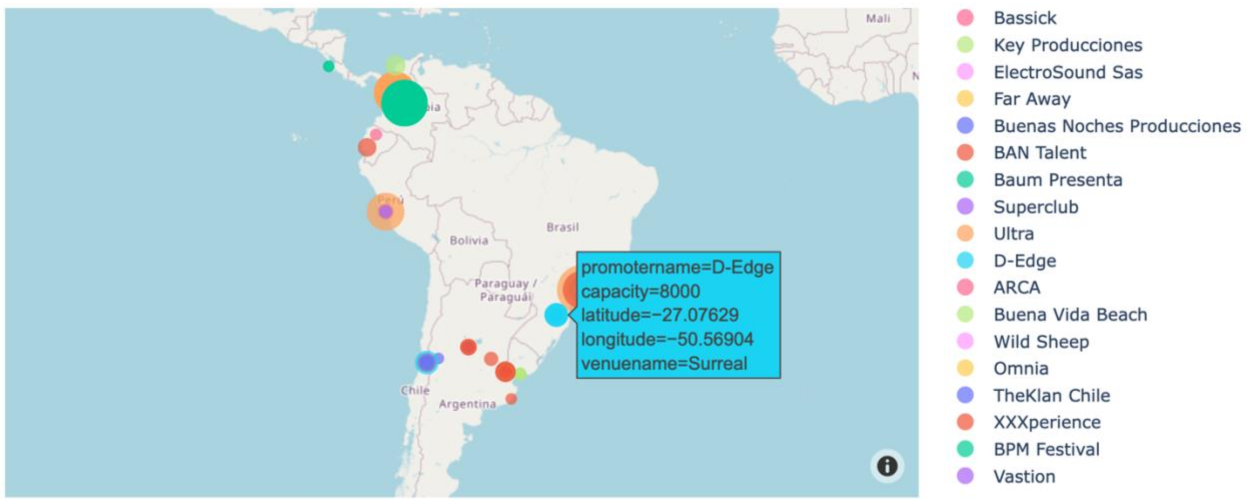


Figure N. Interactive geospatial visualization of the results from Query 5 (South America query).

The relative size of each event circle depicted above is proportional to the event's capacity. Events with greater capacities are represented by larger circles. Being able to narrow down the data by focusing solely on large-scale events enables agents representing high-profile artists to streamline the event search process when making future tour plans. The practical significance remains when seeking more intimate club events, as a quick adjustment of the capacity size and inequality sign in the query (for example, events <= 300 instead of >= 2000).

These four geographic displays are the product of years of analytical curiosity in the context of event bookings. Utilizing statistical programming and domain expertise, these visualizations went from abstract ideas to interactive data tools.

While the above figures graphed event patterns over time, the focus will now shift to a singular analysis, centering on individual artist touring trends.

Measuring Artist Profile Growth via Venue Capacity

As mentioned in Chapter 1, when an artist is enjoying a meteoric rise, it can be challenging to calculate the most beneficial tour routing and performance fee. Leaving the performance fee out of the analysis for reasons aforementioned, there are other effective ways to use quantitative analysis to make predictions and recommendations about where the artist should perform.

Enter the parameter of venue capacity, located in the Venues table of the database.

By running a SQL query to isolate one artist's 'artistname,' 'capacity', and 'showdate' within a specific growth period, an agent can measure the growth or contraction of the average venue size the artist plays. In turn, an agent can then create a regression line to predict the average size venue the artist should play in the months or years ahead. This manuscript will focus on the performing artist Innellea's 51 bookings over the most recent two-year span 2022-2023.

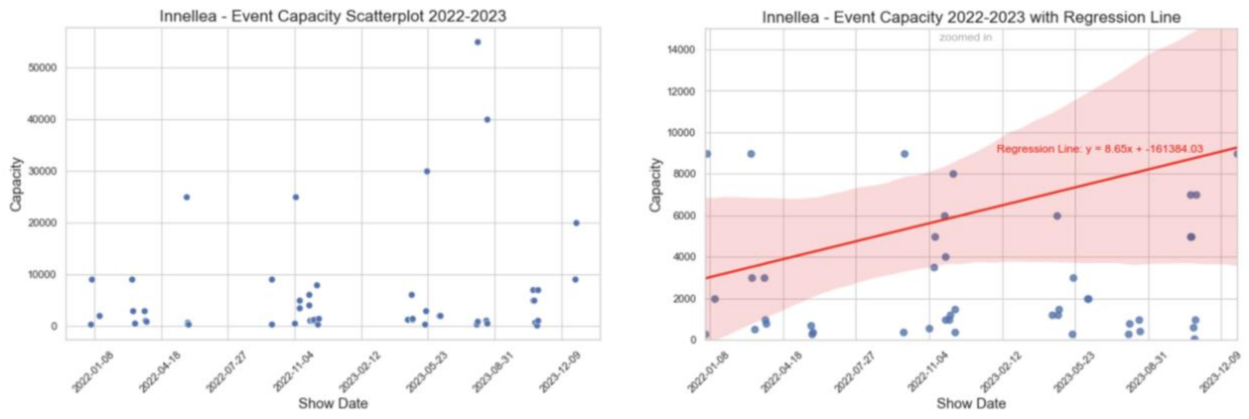


Figure O. Scatterplots of Innellea event capacities from 2022-2023. The second plot contains the same data as the first, but is zoomed in and contains the regression line and equation.

Venue capacity is a central consideration when it comes to pricing an artist's performance fee. Therefore, conducting a regression analysis of an artist's event capacities over time can serve as an effective approach to determine which promoters to play for, which venues to perform in, the appropriate ticket price for fans, and, subsequently, the performance fees that the artists should earn.

In the context of Innellea's bookings, a prediction can now be made to forecast the average capacity venue he should be playing in halfway through 2024. When the regression equation is evaluated at 'showdate' 2024-07-01, the y output is 10,847. Meaning, to remain on the same growth pace as today, the average capacity of the venues Innellea should be playing in by halfway through 2024 is 10,847. It is important to note that the majority of events this artist is playing are clubs and smaller venues; the median venue capacity of all Innellea shows between 2022 and 2023 is only 2,000. The average is pulled upward by premier large-scale festivals like Coachella, Proper, Zamna, Baum, and Ritvales. This manuscript does not remove these outliers, as their presence in the dataset is an important indicator of profile growth.

A thorough list of analyses has been assembled, illuminating the pathways for future touring decisions. With a clear business justification for this booking tool established, the manuscript will now measure the performance of each database. Benchmark tests will be conducted to assess the query speeds of each SQL engine on Queries 3, 4, and 5, both with and without indexing.

Chapter 6: Performance

Benchmark Testing

Ahead of discussing benchmark testing, two important acknowledgments must be addressed. Firstly, benchmark testing on all four databases was performed under controlled conditions: all of the results were captured using only one iMac desktop, no other irrelevant applications were open during testing, and all tests were run on identical data. This is the same hardware, software, and production environment where all the data was originally collected and formatted.

Secondly, several different database metrics can be measured by benchmark testing (IBM 2023). Examples include the amount of CPU utilization, speed of write operations, and the tuning of table space configurations. Based on the unique needs of music agents when booking tours, the query performance speed of each database is the variable this manuscript will test. The performance speed will be tested on the final three queries: the small query (Query 3), the large query (Query 4), and the South America query (Query 5). The selection of these queries for benchmark testing was deliberate, taking into consideration their range of distinct SQL operations, their yielding of varying amounts of data output, and their practical use for agents.

Upon reviewing the performance results of all eight databases across the three query types, a decision will be made regarding the specific database engine to be implemented in all future operations.

Small Query (Query 3)

Recall the small query displays the top 10 cities the artist ANNA performs in the most. Despite retrieving a relatively small output, it is quick and efficient, and the criteria are very commonly analyzed. Owing to the relatively modest scale of this dataset, the difference in performance times across each benchmark test is only fractions of a second. To amplify these differences, the results of all benchmark tests (local, cloud, with, and without indexing) have been multiplied by 100.

The small query was tested 1,000 times in each environment. MySQL performed quicker on the small query in every scenario (cloud, local, with, without indexing). The resulting table below displays the average query time of each of the eight environments, with corresponding standard deviations indicating the general distribution of times centered around the mean:

Environment (small query)	**Avg. Query Speed (with indexes)**	**Standard Deviation (σ) with indexes**	**Avg. Query Speed (without indexes)**	**Standard Deviation (σ) without indexes**
PostgreSQL local	0.351 seconds	0.038 seconds	**0.327 seconds**	0. 034 seconds

MySQL local	0.126 seconds	0.019 seconds	0.276 seconds	0.029 seconds
PostgreSQL cloud	5.844 seconds	1.25 seconds	**5.747 seconds**	1.386 seconds
MySQL cloud	1.995 seconds	0.304 seconds	2.176 seconds	0.301 seconds

Figure P. Table of results of average performance speed on the small query, and variation of all eight database environments.

Since the small query only returns a very small amount of data, it is not surprising that the small query possesses the most nominal difference in average performance results across all eight database environments. What is quite surprising, however, is that both PostgreSQL environments demonstrated a faster average execution time without any indexes, as opposed to with indexes (denoted in green in Figure P). One plausible explanation for this may be that the smaller query size exhibits improved performance when executed without the complicating factor of indexes.

During each table's import, indexes were automatically created on each table's primary key. Foreign keys were also added during the loading phase (refer to Figure C for corresponding PostgreSQL and MySQL table import code in Chapter 4). Without any primary or foreign keys, a small code addition was necessary in order to run the large query: 'promotername' needed to be included in the GROUP BY clause. Without this addition, and without foreign keys, a DatabaseError occurred since 'promotername' was nonaggregated, and not functionally dependent on 'promoterid.' Therefore, the non-indexed databases also had more code to execute than the indexed ones. Therefore, in theory, more complex queries should perform quicker with indexes than without. Subsequently, the query speed of the larger query will now undergo evaluation.

LARGE QUERY (QUERY 4)

The large query, comprised of three window functions, a Common Table Expression (CTE), a nested query, and 52 total lines of code, exhibits significantly more complexity. This sophistication is expected to create a larger distinction between non-indexed and indexed, as well as local versus cloud environment speeds.

The large query was tested 1,000 times in each environment. Similar to the small query, the difference in performance speed of local databases was also nominal:

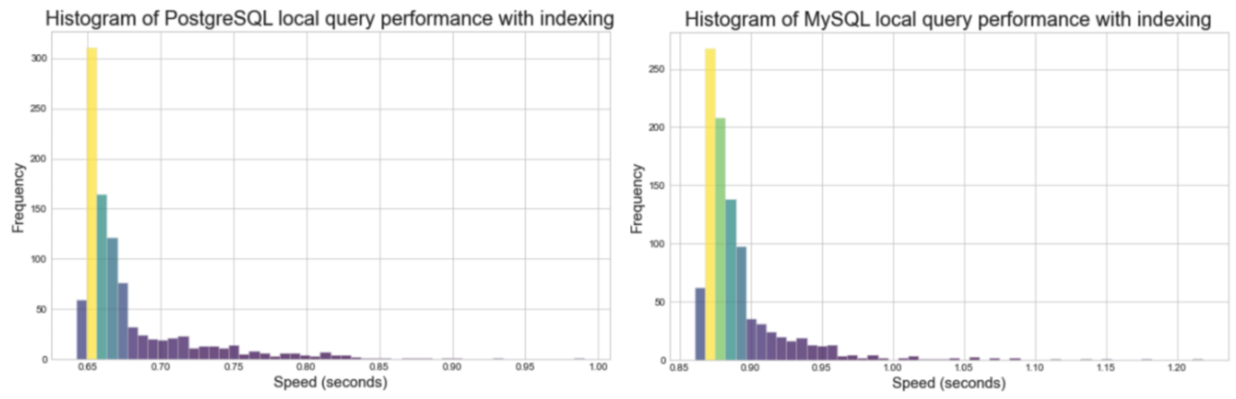
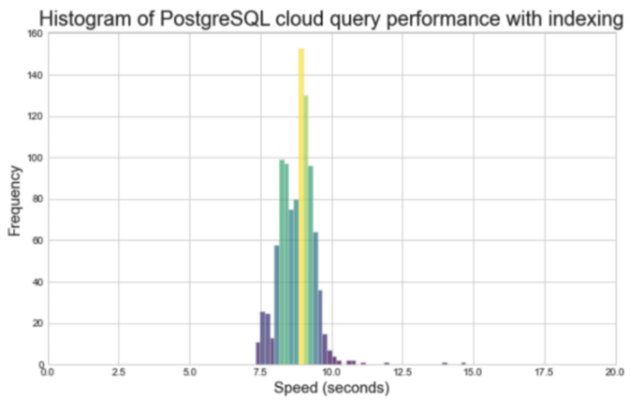


Figure Q. Benchmark testing PostgreSQL local and MySQL local performances on the large query, with indexing.

Query speed of both local environments with indexing averaged under one second, with PostgreSQL narrowly faster at 0.678 seconds, to MySQL's 0.892 seconds.

The results of the benchmark tests in cloud environments were drastically different. A notable distinction in performance speeds was evident when executing the larger query locally as opposed to on a cloud server:



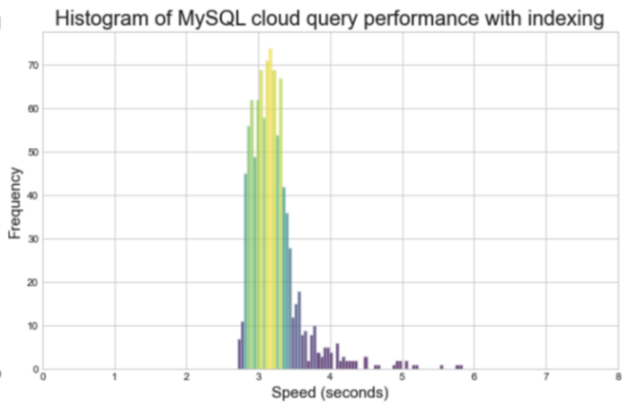


Figure R. Benchmark testing PostgreSQL cloud and MySQL cloud performances on the large query, with indexing.

Several factors can contribute to cloud query performance taking longer than local query performance. Public IP internet routing and network latency are two potential reasons for the large discrepancy. An additional likely cause is the amount of operations this large query is undergoing.

Since PostgreSQL cloud query performance took the longest relative to its local engine performance, a comparison of the planning time and execution time of each Postgres environment is listed below, after running an EXPLAIN ANALYZE query:

Environment	Planning Time (milliseconds)	Execution Time (milliseconds)
PostgreSQL Local	13.677	12.037
PostgreSQL Cloud	246.461	48.537

Figure S. The planning and execution times of PostgreSQL local and cloud environments on the large query.

The planning time took 19x longer in cloud vs. local, while the execution time was only 4x longer. This implies that optimizing the planning phase in the cloud environment will be the best way to improve query speed.

Complete results of the benchmark tests on the large query are below:

Environment (large query)	**Avg. Query Speed (with indexes)**	**Standard Deviation (σ) with indexes**	**Avg. Query Speed**	**Standard Deviation (σ)**

			(without indexes)	without indexes
PostgreSQL local	0.678 seconds	0.043 seconds	**0.545 seconds**	0.028 seconds
MySQL local	0.892 seconds	0.036 seconds	1.019 seconds	0.048 seconds
PostgreSQL cloud	8.806 seconds	1.192 seconds	**8.785 seconds**	1.645 seconds
MySQL cloud	3.222 seconds	0.454 seconds	3.226 seconds	0.399 seconds

Figure T. Table of results of average performance speed on the large query, and variation of all eight database environments.

There are several interesting observations from the large query results of these eight database benchmark tests. Firstly, it is not unexpected that the query speeds of all SQL engines on the local machine surpassed those executed in the cloud. What is quite surprising, however, is that both PostgreSQL environments – once again – demonstrated a faster average execution time without any indexes, as opposed to with indexes (denoted in green in Figure T above). While only fractions of a second quicker on average, the absence of any primary or foreign keys during the execution of this extensive and complex query should theoretically have resulted in a slowdown.

It is most likely that the relatively small size of the dataset is contributing to the query speed without indexes outperforming that with indexes in PostgreSQL. However, some believe that when a query is joining together many (in this case, all) tables of the database, then a full table scan may be more appropriate than indexing on a large portion of the database (Qi 2016).

For enhanced visualization of both performance time and frequency distribution in each environment, Figures U and V below displays all eight histograms of large query performances, separated by indexed and non-indexed scenarios.

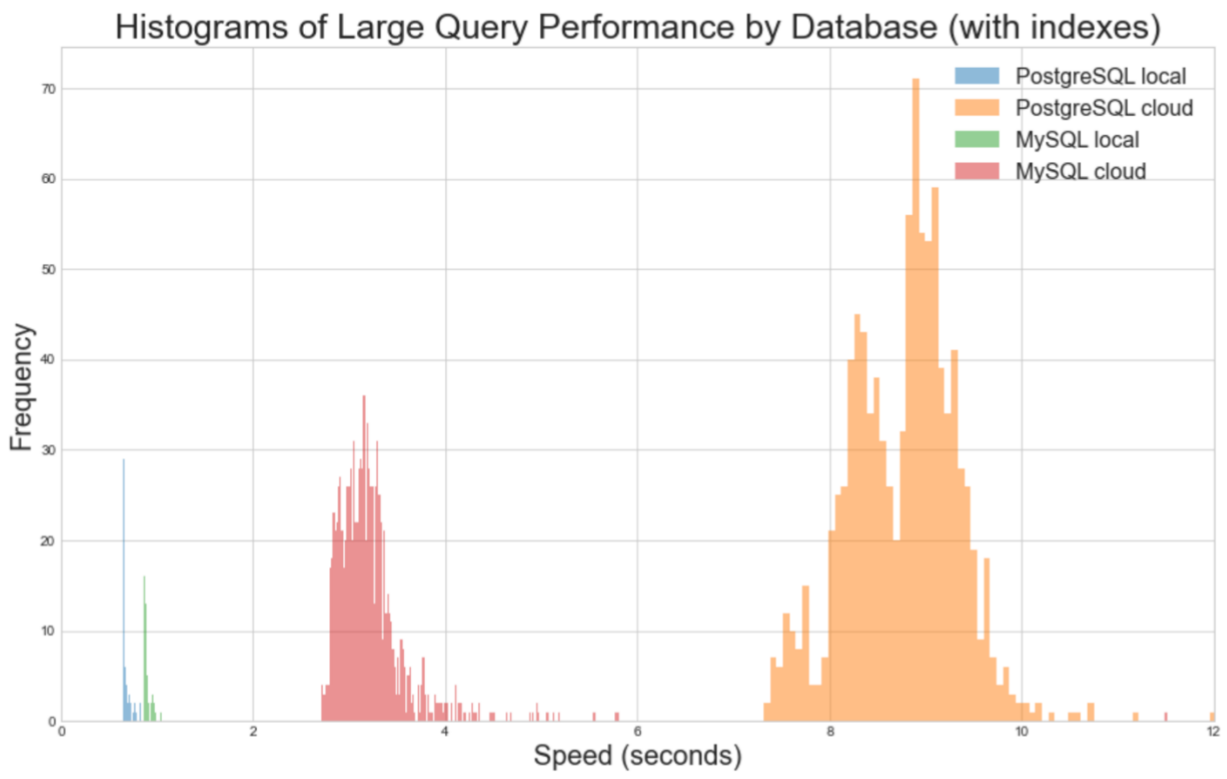


Figure U. Histgrams of large query performance by database (with indexes).

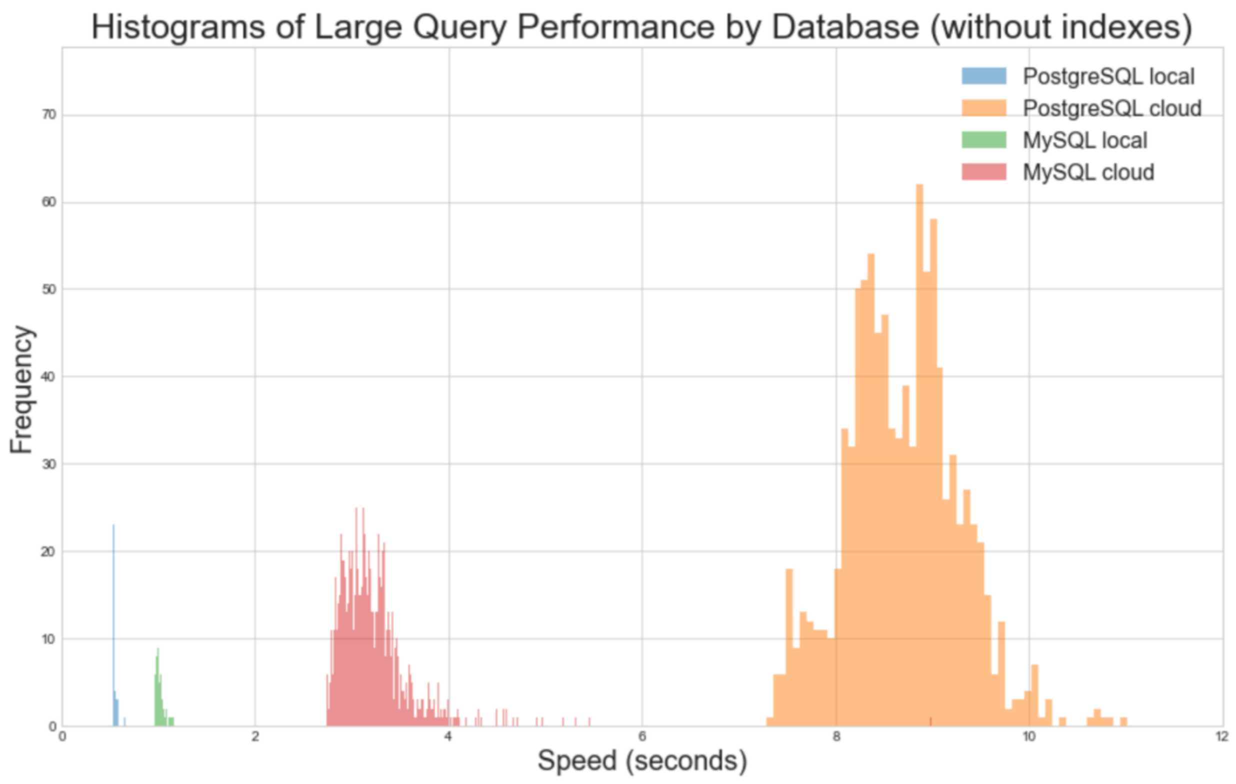


Figure V. Histgrams of large query performance by database (without indexes).

Comparing these two graphics on top of one another demonstrates more interesting observations. Removing indexes did not create significant changes in any of the eight databases' standard deviations. Meaning, once primary and foreign key dependencies were removed from the database, the volatility of query speeds did not change.

Returning to Qi's scenarios for administering a full table scan instead of indexing, he asserted that, depending on the overall size of the database, full table scans can be better than index scans when at least half of the tables are used in the query: "in this case the index is somewhat similar to a smaller copy of the table, [so] querying data actually means query the target data twice (once in the index tree, the other in the actual data page), [and therefore] the query efficiency is greatly reduced" (Qi 2016). This could be a viable explanation for why both PostgreSQL environments performed better on the large query without any indexes. Therefore, this manuscript will benchmark test Query 5 (the South America query), noting that this query only joins three of the four tables.

South America Query (Query 5)

While similar in code length to Query 4, there are some important differences that make the South America query particularly attractive for evaluation. Query 5 is the first query to take every single data type in the database into account (integers, variable characters, dates, booleans, and double-precision floating points). This is significant, as the import of the date column type proved to be particularly difficult in MySQL. Figure W below uncovers whether or not MySQL's difficulty in importing date columns had any bearing on the query performance related to date data. Another appealing aspect of this query for assessment is that, as previously mentioned, it only joins three of the four tables. Below are the results of the benchmark test on Query 5:

Environment (South America query)	**Avg. Query Speed (with indexes)**	**Standard Deviation (σ) with indexes**	**Avg. Query Speed (without indexes)**	**Standard Deviation (σ) without indexes**
PostgreSQL local	0.431 seconds	0.050 seconds	**0.398 seconds**	0.034 seconds
MySQL local	0.572 seconds	0.040 seconds	**0.473 seconds**	0.030 seconds
PostgreSQL cloud	5.857 seconds	0.444 seconds	5.888 seconds	1.274 seconds
MySQL cloud	2.378 seconds	0.447 seconds	**2.158 seconds**	0.426 seconds

Figure W. Table of results of average performance speed on the South America query, and variation of all eight database environments.

This time, three of the four database environments performed the query faster on average without indexes (denoted in green in Figure W above). Only PostgreSQL cloud performed faster with indexes, and only by a couple thousandths of a second, on average.

Another important observation is that the large query took longer than the South America query in *all* eight scenarios. This suggests that the extent of a four-table join plays a more important role in prolonging execution times than the diversity of data types.

Chapter 7: Results and Future Application

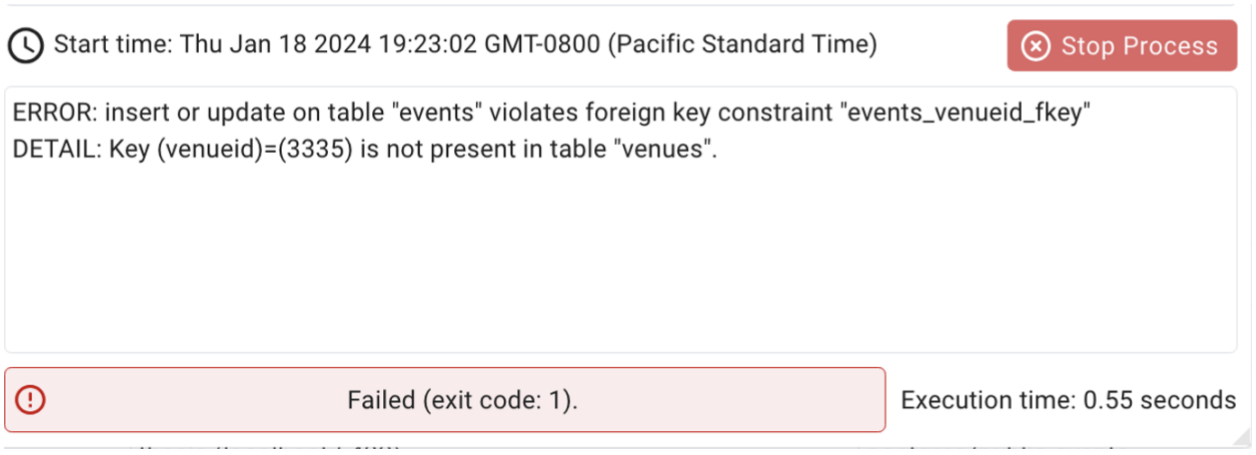
Optimal Database Implementation

Executing an abundance of read and write operations, along with rigorous benchmark testing across all eight databases, led the author of this manuscript to develop a deep familiarity with PostgreSQL and MySQL functionalities. The process involved considerable trial and error, and required meticulous scrutiny to refine queries until they achieved their eventual effectiveness and accuracy. As tables and features were incrementally added in the early stages of the project, a gradual inclination toward one SQL engine over the other became evident.

PostgreSQL (and its management tool pgAdmin) cloud with indexes became the preferred configuration for building this database and agent booking tool.

A primary reason for choosing PostgreSQL over MySQL is due to the ease of data importing. With pgAdmin, the import was executed in one query without any problems. Meanwhile, in MySQL Workbench, there were some complications, particuarly with adding a column with date data. For all four MySQL databases, the Events table in MySQL Workbench needed to be imported as a variable character (text) column and then changed to date column after the import. Otherwise, the following error message would appear: "Row import failed with error: ("Incorrect datetime value: '11-14-2021' for column 'ShowDate' at row 1", 1292)", however, the entire import would still execute, and then once the table was queried, it would display a completely empty table (so nothing was imported). All due to this date column. No amount of reformatting would solve this error; only converting the column to variable character (text) during import would remedy this.

A second problem with data importing in MySQL Workbench was the automatic skipping of rows with null cells. The MySQL error log did not disclose enough information to identify any inconsistencies. When importing indexed tables, it simply skipped rows that did not have readable foreign key constraints. This violates atomicity. Meanwhile, PostgreSQL had a more granular error message, going as specific as the cell value so that any data abnormalities were a quick find:



Start time: Thu Jan 18 2024 19:23:02 GMT-0800 (Pacific Standard Time)

```
ERROR: insert or update on table "events" violates foreign key constraint "events_venueid_fkey"
DETAIL: Key (venueid)=(3335) is not present in table "venues".
```

Failed (exit code: 1).

Execution time: 0.55 seconds

Figure X. PostgreSQL error notification.

A third issue with MySQL Workbench was its inability to import special characters. From the beginning, the use of UTF-8 encoding was clearly denoted prior to all data imports. However, for MySQL, characters like ñ, é, í, á all had to be updated to n, e, i, and a respectively in order for the Venues table to import properly. Otherwise, the import would stop 20-30 rows before the special character. This was an extremely challenging issue to identify and resolve. The author had to display each .csv file in Visual Studio Code and comb through the .csv to identify where and when these issues arose. Meanwhile, pgAdmin did not have any trouble importing special characters.

Finally, the use of indexes is strongly preferred, because ensuring referential integrity is a vital pillar of database management. pgAdmin highlighted this importance by including reminders in error messages that unique identifiers for every table were necessary. This was a valuable indication to make sure each database followed certain data conventions. Due to the guidance of pgAdmin error messages, the author rewrote primary key values so that they were not only unique for every primary key column, but they also were written in a way that allowed for the database to be updated for years to come:

ArtistID: 1 – 46 (room for 54 more artists before reconfiguration)
PromoterID: 101 – 503 (room for ~500 more promoters before reconfiguration)
VenueID: 1001 – 1992 (room for 8000 more venues before reconfiguration)

EventID: 10001 – 12601 (room for an infinite amount of more bookings)

With the above primary key values selected, it is anticipated that the database may undergo updates without any interruption for at least another five years before larger multi-digit primary keys would need to be implemented.

Real World Application

As this database grows, the ability to update all tables in 3NF with the code written in Figure B in Chapter 4 will be critical to its upkeep. A drawback of this design is that someone with comprehensive knowledge of SQL would need to manage this. But as discussed in Chapter 3, in the future, parameters can be established to automate the ETL process within Google Cloud (Chouliaras and Sotiriadis, 2023). Additionally, after ETL and during the querying phase, front-end software could be developed so that non-technical colleagues do not have to write SQL code or Python scripts to retrieve valuable insights from the database. Or even better, a generative AI tool could be created so that other non-technical agents can simply ask the AI tool their questions. This would require an extensive natural language processing model capable of comprehending the various ways a non-technical agent might informally query the database.

Chapter 8: Conclusion

With this comprehensive tool at an agent's disposal, incredibly precise queries can be implemented to subset previous booking data and allow agents to make predictions for future tour strategy.

This analytics tool is also a platform for interactive data visualizations, generating state-of-the-art digital maps to facilitate bookings for both immediate and future touring scenarios. Issues stemming from routine tour challenges, which once demanded laborious efforts, are now just a few lines of SQL code away from resolution. This software eliminates the necessity of searching for promoter leads in physical rolodexes or sorting through digital inboxes and spreadsheets. Now an agent can specify their artist's location, musical genre, and other exact parameters to extract a highly-customized list of potential customers and proceed confidently and quickly toward their business solutions.

Additionally, the database architecture was cleverly designed to allow for public dissemination while preserving the critical and valuable analytical components of each booking. The creation of four separate tables in third normal form ensures that the data is accurate, easily retrievable, and stored in an efficient manner.

After thorough benchmark testing of eight distinct database scenarios, the decision was reached to proceed with a PostgreSQL cloud environment incorporating indexing for all upcoming database interactions. Importing, updating, reading, and writing booking data in this environment proved to be the best pathway forward.

In leveraging the capabilities of data science within the music agent world, the question is not whether the industry will adopt these technological advancements, but rather a question of when they will do so. This powerful database application has successfully transformed an archive of formerly static information into dynamic 21st-century data, and is a massive step forward for booking agents responsible for representing and touring electronic music artists.

Bibliography

Chouliaras, Spyridon, and Stelios Sotiriadis. 2023. "An Adaptive Auto-Scaling Framework for Cloud Resource Provisioning." *Future Generation Computer Systems* 148 (2023): 173–83. <https://doi.org/10.1016/j.future.2023.05.017>

DeBarros, Anthony. 2022. *Practical SQL : A Beginner's Guide to Storytelling with Data*. 2nd edition. San Francisco: No Starch Press, pp. 275-303.

Deka, Ganesh Chandra. 2015. "BASE Analysis of NoSQL Database." *Future Generation Computer Systems*, 52 (November 2015): 13–21. <https://doi.org/10.1016/j.future.2015.05.003>

Habib, Jeremie, and Heather Wilde. 2023. "The Future of Live Events with AI." Advance, by Gigwell. 15 June 2023. < https://advance.gigwell.com/posts/future-of-live-events-with-ai>

IBM Documentation. "Benchmark testing." 17 March 2023. <https://www.ibm.com/docs/en/db2/11.5?topic=methodology-benchmark-testing>

Klimek, Bartłomiej, and Maria Skublewska-Paszkowska. 2021. "Comparison of the Performance of Relational Databases PostgreSQL and MySQL for Desktop Application." *Journal of Computer Sciences Institute* 18 (2021): 61–66. Last modified March 17, 2023. <https://doi.org/10.35784/jcsi.2314>

Mill, Steve. 2021. "What is Beatport and how does it work?" iMusician. 31 August 2021. < https://imusician.pro/en/resources/guides/what-is-beatport-and-how-does-it-work>

Plotly Hover Text and Formatting in Python < https://plotly.com/python/hover-text-and-formatting/> and scatter_box explanation < https://plotly.com/python/figure-structure/>

Qi, Chunxia. 2016. "On Index-Based Query in SQL Server Database." In *2016 35th Chinese Control Conference (CCC)*, 9519–23. TCCT, 2016. <https://doi.org/10.1109/ChiCC.2016.7554868>

Ravikiran, A S. 2023. "What is Normalization in SQL? 1NF, 2NF, 3NF and BCNF in DBMS." Simplilearn. 11 October 2023. < https://www.simplilearn.com/tutorials/sql-tutorial/what-is-normalization-in-sql>

Rogers, Ian, and Samuel Whiting. 2020. "'If There Isn't Skyscrapers, Don't Play There!' Rock Music Scenes, Regional Touring, and Music Policy in Australia." *Popular Music and Society* 43, no. 4 (2020): 450–60. <https://doi.org/10.1080/03007766.2020.1730654>

Taylor, Petroc. 2023. "Ranking of the most popular relational database management systems worldwide, as of September 2023." Statista. 14 September 2023.

<https://www.statista.com/statistics/1131568/worldwide-popularity-ranking-relational-database-management-systems/>

Wang, Peter. 2023. “How the Cloud is Changing Data Science.” Harvard Business Review. 2 November 2023. Accessed December 2023-January 2024. <https://hbr.org/2023/11/how-the-cloud-is-changing-data-science>

Zendel, Adam. 2021. “‘There Are No Days off, Just Days without Shows’: Precarious Mobilities in the Touring Music Industry.” *Applied Mobilities* 6, no. 2 (2021): 184–201. <https://doi.org/10.1080/23800127.2020.1827516>