CONCERT FINDER

1st Shailesh Mahto
ES - Data Science
University at Buffalo
smahto@buffalo.edu

2nd Shivam Bhalla
ES - Data Science
University at Buffalo
sbhalla4@buffalo.edu

3rd Abhiroop Ghosh
ES - Data Science
University at Buffalo
aghosh4@buffalo.edu

***Abstract*—Concert finder is a platform where music enthusiasts can come in and find events related to their favorite artists or in their area or other relevant criteria. For this task, we modelled a database with eight tables handling various many-to-many and one-to-many relationships. Indexing was used for efficient retrieval and a web-app is used to enable users to interact with the database.**

***Index Terms*—database modeling, web app development, indexing**

I. Introduction

We present Concert Finder, a one-stop solution for music enthusiasts to find relevant events. This includes a BCNF normalized data model with eight tables namely - Event, Artist, Genre, Venue, State, Event-Venue Bridge, Event-Genre Bridge and Event-Artist Bridge. The bridge tables are used to efficiently convert many-to-many relationships into a couple of one-to-many relationships. Furthermore, indexing is used to enable efficient retrieval of information that users might want to use for filtering the database. A web-app is used to package all of this in a user-friendly UI to enable seamless interaction.

II. Data Collection

The initial phase of the project involved collecting music event data from the Ticketmaster API [**?**], specifically focusing on events along the US coast. This required obtaining an API key from Ticketmaster and accessing their REST API services to retrieve event data. The Ticketmaster API endpoints provided comprehensive data covering event details, attractions (artists), venues, genres, and other relevant information crucial for music event management along the US coast.

- Obtaining API Key: To access Ticketmaster's REST API services, an API key is required. This key is obtained through registration on the Ticketmaster Developer Portal, where developers generate unique keys for API access.
- Utilizing Ticketmaster API Endpoints: Ticketmaster's API offers various endpoints to fetch event-related data comprehensively. These endpoints encompass event details, attractions (artists), venues, and other relevant information crucial for event management.
- Development of HTTP Request Scripts: In the project's development phase, bespoke Python scripts were crafted to interact with the Ticketmaster API. Leveraging the requests library, these scripts facilitate HTTP requests to designated API endpoints, retrieving event data in JSON format for subsequent processing.
- A focus was given on not bombarding the Ticketmaster server with many requests by using a 20 seconds gap between subsequent API calls.
- The Ticketmaster API Explorer offers users a user-friendly interface to set up the parameters of API URI relevant for any use case and search for events to access relevant information efficiently as observed in (Fig.1). With extensive event listings and detailed event entries, it serves as a valuable tool for event exploration and ticket purchasing. Further exploration and utilization of the Ticketmaster API can provide users with enhanced insights into the world of events and entertainment.
- The results were saved locally to allow easy experimentation and development.

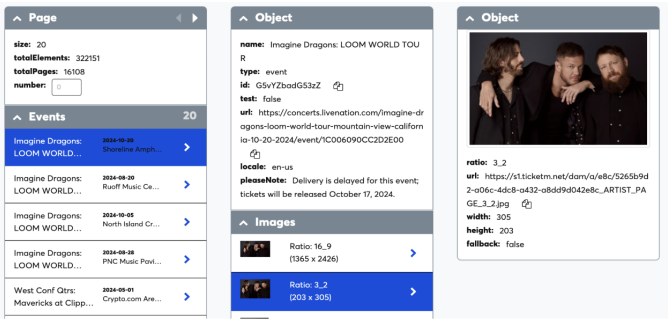

Fig. 1. Ticketmaster API results on Ticketmaster Platform

III. Data Model

A. Overview

Schema Design for Event-related Data: Prior to data import, meticulous attention was devoted to designing a robust data model tailored to music event data. The database schema was meticulously crafted to encompass tables dedicated to event, venue, artist, genre, state, and associations or bridge tables between event and their corresponding venues, artists, and genres. Entity relationships were carefully defined within the schema to maintain data integrity and facilitate efficient querying, ensuring that music event data could be effectively organized and queried based on various criteria.

Entity Relationship Definition: Entity relationships were meticulously defined within the database schema to uphold

data integrity and enable streamlined querying. Noteworthy relationships include those between events and venues, events and artists, and events and genres, enforced through foreign key constraints.

B. ER Diagram

The ER diagram shown in Fig 2 drove all of the development effort. The ER Diagram was created keeping many-to-many relationships in mind. The idea is that one event could have various artists performing, but one artist could be performing in various events. The bridge table allows us to convert such relationship into two one-to-many relationships.

C. BCNF form

The data model has the following functional dependpencies:

- $event\_id \rightarrow event\_name$
- $event\_id \rightarrow event\_start\_date$
- $event\_id \rightarrow event\_start\_time$
- $event\_id \rightarrow event\_price\_min$
- $event\_id \rightarrow event\_price\_max$
- $event\_id \rightarrow event\_age\_restrictions$
- $event\_id \rightarrow event\_seatmap\_url$
- $artist\_id \rightarrow artist\_name$
- $genre\_id \rightarrow genre\_name$
- $genre\_id \rightarrow segment$
- $venue\_id \rightarrow venue\_name$
- $venue\_id \rightarrow venue\_city$
- $venue\_id \rightarrow venue\_postal\_code$
- $venue\_id \rightarrow venue\_latitude$
- $venue\_id \rightarrow venue\_longitude$
- $venue\_id \rightarrow venue\_address$
- $state\_code \rightarrow state\_name$
- $state\_code \rightarrow country\_name$
- $state\_code \rightarrow country\_name$
- Additionally, the bridge tables contain only the keys of other tables. In other words, all the keys get their own table, ensuring that the tables in the data model are indeed in BCNF form.

IV. Data Tranformation and Loading

A. Table Creation

SQL scripts are executed to create database tables based on the generated SQL statements. For instance, the ”create_venues_table” script is executed to create the ”venues” table with appropriate column definitions.

B. Transformation high-level flow

The API was parsed to collect relevant data for each table. The code is available in the notebooks within etl folder. First, a smaller subset of data was extracted from the API for development purpose. Then, the data was parsed carefully and loaded to Pandas dataframe. Then, efficent and reusable code was written to convert the pandas dataframe into a list of insert statements, which were saved locally. Then, another function saved these scripts for loading. Finally, a third function loaded the script and executed them in PostgreSQL database to complete the bulk load of data. Once, this code was ready, these stable functions were used to automate data loading for all the tables.

Later, larger data was extracted from the API for final bulk loading. Various unforeseen issues were faced at this stage.

C. Some of the challenges faced were

- Sub-genre missing values: For the genre table, initially another column called sub-genre was used as the primary key. However it was later realized while loading the large extract that sub-genre value was missing for some of the events. Hence, we made the decision to drop sub-genre and use genre as the primary key
- Special character in name-type columns: We realised that certain special characters like ’ existed in the name of certain events, venues or other text fields. Since this is the same character used to separate field values in the INSERT statements, the code was failing during insertion. We handled this situation by adding one extra sigle quote which lets SQL know that the character needs to be escaped.
- Missing values: Certain events did not have certain column values like event_price, etc.. NULL value was used to automatically handle such instances.

D. Order of loading tables

Since primary-key, foreign-key relationship existed in the tables, the tables on the primary key side had to be run before the tables using the same column as foreign key in another table. Moreover, on delete, we decided to CASCADE deletion from table with primary key to the table with foreign key to ensure data integrity is maintained at all times. Hence, the order of loading tables was as follows:

1) event
2) artist
3) genre
4) state
5) venue
6) event_artist_bridge
7) event_genre_bridge
8) event_venue_bridge
9) venue_state_bridge

V. Data Loading

A. Order of data loading matters

Data Loading Process:

- Table Creation: SQL scripts are executed to create database tables based on the generated SQL statements. For instance, the ”create_venues_table” script is executed to create the ”venues” table with appropriate column definitions.
- Data Insertion: SQL scripts containing insert statements are executed to insert values into the corresponding tables. Each insert statement is generated based on the data present in the Pandas DataFrame. For example, the

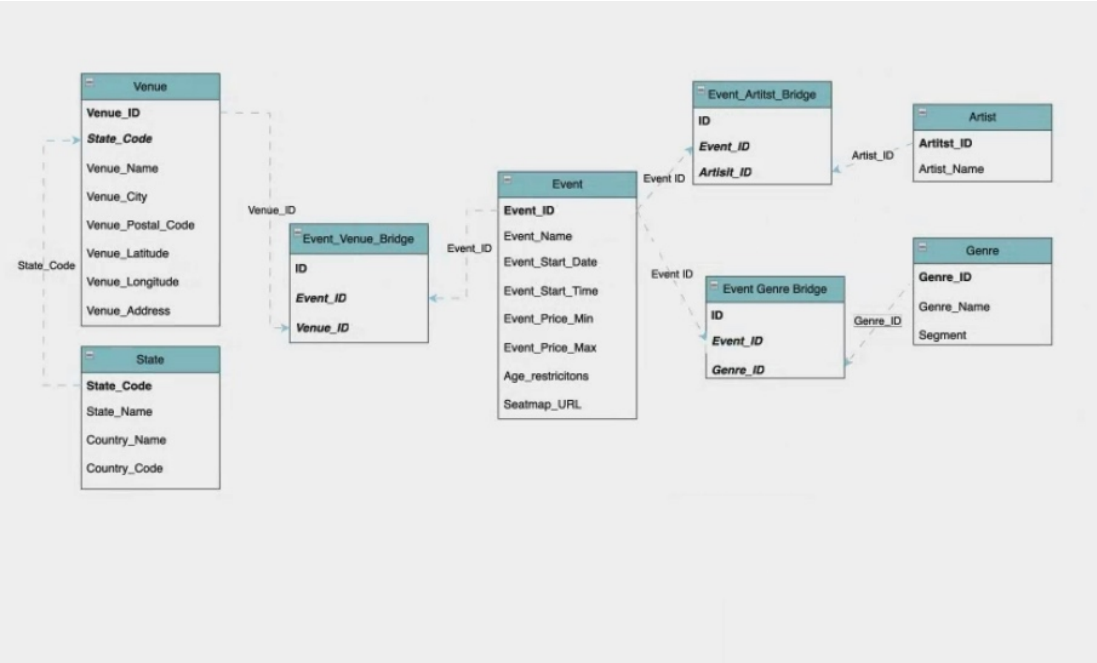

Fig. 2. ER Diagram

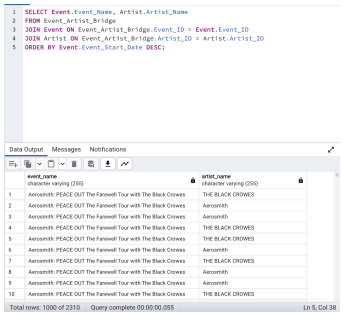

Fig. 3. Query 1 without indexing

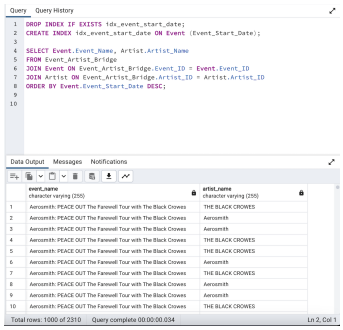

Fig. 4. Query 1 with indexing

”insert_venues” script inserts venue information extracted from the DataFrame into the ”venues” table.

VI. Performance Analysis

Performance Analysis was done based on indexing of attributes on different queries. Scripts for each query were first executed without indexing, and subsequently with addition of an index.

- For Query 1, the complete execution time (cost) was found to be 55ms. With indexing of the ”event_start_date” attribute, the cost significantly reduced to 34ms. This results in speeding up the data retrieval and faster look-ups.
- For Query 2, the cost was found to be 48ms. With indexing of the ”venue_name” attribute, the cost significantly reduced to 41ms.
- For Query 4, the cost was found to be 26ms. With indexing of the ”artist_name” attribute, the cost significantly reduced to 19ms.

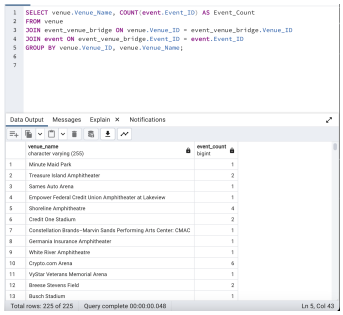

Fig. 5. Query 2 without indexing

- For Query 5, the cost was found to be 63ms. With indexing of the ”artist_name” attribute, the cost significantly reduced to 39ms. From the figure, we can see that indexing results in reduced table scans and faster joins. With proper indexes, one can quickly locate data using

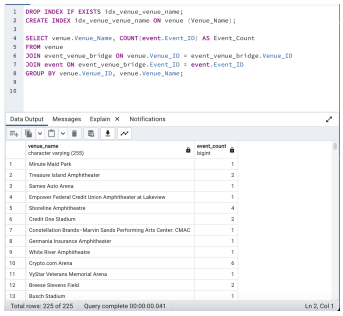

Fig. 6. Query 2 with indexing

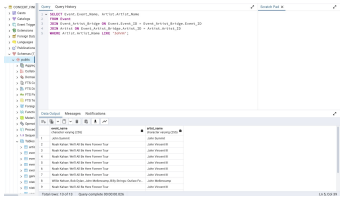

Fig. 7. Query 5 without indexing

the index structure, significantly reducing the scanned number of rows. Hence, one can deduce that the data flow in the diagram is compact, which evidently showcases the impact of indexing.

- For Query 6, the cost was found to be 56ms. With indexing of the ”event_start_date” and ”event_name” attributes, the cost significantly reduced to 34ms. In this query, we have used multi-indexing, which involves creating an index on multiple columns in a table. This improves query performance when filtering or sorting.

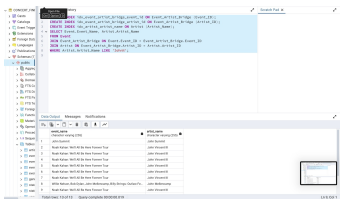

Fig. 8. Query 4 with indexing

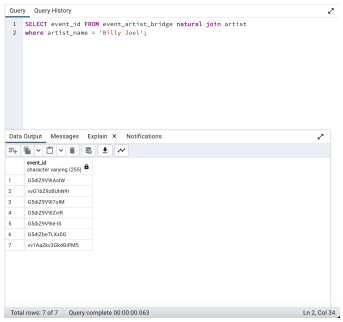

Fig. 9. Query 5 without indexing

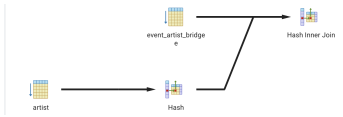

Fig. 10. Data flow without indexing

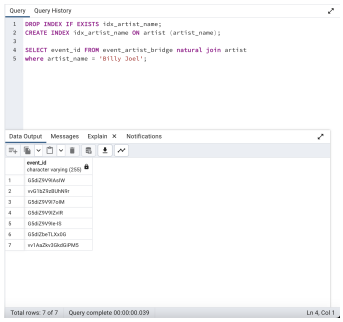

Fig. 11. Query 5 with indexing

VII. Web App Development

A web app was developed using Python-Flask. This app allows users to filter the events based on their chosen artists, etc.

VIII. Contribution

Abhiroop Ghosh: Abhiroop has made significant contributions to the data extraction process from the Ticketmaster API. His expertise in Python scripting and API integration has facilitated the retrieval of event-related data efficiently. Shivam Bhalla: Shivam’s contributions have been invaluable in database design and schema modeling. His meticulous approach to defining relationships between entities and ensuring data integrity has laid a solid foundation for the project’s database structure. Shailesh Mahto: Shailesh has played a key role in the data loading phase, particularly in utilizing pandas and SQL Alchemy for data transformation and seamless integration into the PostgreSQL database. His expertise in data manipulation and SQL scripting has streamlined the data import process.

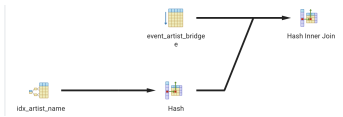

Fig. 12. Data flow with indexing

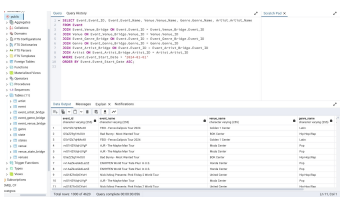

Fig. 13. Query 6 without indexing

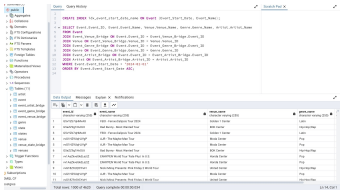

Fig. 14. Query 6 with indexing

Figure Labels: Use 8 point Times New Roman for Figure l

ACKNOWLEDGMENT

We would like to express our sincere gratitude to everyone who contributed to the success of this project.

First and foremost, we extend our deepest appreciation to the team at Ticketmaster for providing access to their API services and invaluable support throughout the development process.

We are grateful to our project mentors and advisors for their guidance and encouragement. Their expertise and insights have been instrumental in shaping the direction of our work.

Special thanks to Prof.Shreyasee Das and the Ta's, whose dedication and expertise significantly contributed in guiding us to achieve this project.

We would also like to thank our TA's for their assistance and feedback during various stages of the project.

Last but not least, we express our heartfelt appreciation to our friends and family for their unwavering support and understanding during the course of this endeavor.

This project would not have been possible without the collective effort and support of all those involved, and for that, we are truly grateful.

REFERENCES

[1]Ticketmaster Developer Portal: Official documentation and resources provided by Ticketmaster for developers accessing their API services. [2]Python Requests Library Documentation: Documentation for the Python Requests library, which is used for making HTTP requests to the Ticketmaster API. [3]Pandas Documentation: Official documentation for the pandas library, which is used for data manipulation and structuring in Python. [4]PostgreSQL Documentation: Documentation for PostgreSQL, the open-source relational database management system used for storing event data in your project. [5]SQLAlchemy Documentation: Documentation for SQLAlchemy, a Python SQL toolkit and Object-Relational Mapping (ORM) library used for interacting with databases. IEEE Conference Paper Template: If you're formatting your report according to IEEE standards, you can reference their conference paper template for guidance on formatting and structure. [6]Stack Overflow: While not a formal reference, Stack Overflow can be a valuable resource for troubleshooting specific coding issues or finding solutions to common problems encountered during development. [7]Python Software Foundation: Official documentation and resources provided by the Python Software Foundation for Python programming language.