CONCERT COMPASS

CSE 560 - Data Models and Query Languages
Team Name: X Æ A-Xii

Shailesh Mahto - smahto - 50540379
Shivam Bhalla - sbhalla4 - 50546187
Abhiroop Ghosh - aghosh4 - 50541142

Problem Statement

In today's bustling event scene across the United States, music lovers often find themselves lost in a maze of scattered information when trying to discover upcoming concerts and performances. With details about artists, genres, and venues spread across various platforms, the task of finding the perfect event becomes a daunting challenge. What's missing is a central hub—a place where enthusiasts can easily access comprehensive artist-centric data to tailor their event experiences. That's why we're embarking on a mission to create a nationwide database that brings together detailed information about artists, their genres, event schedules, and venues. By putting the power of choice back into the hands of users, our solution aims to revolutionize the way people explore and engage with events, making every musical experience unforgettable.

Reasons for Choosing a Database over an Excel File -

1. **Scalability:** A database can efficiently handle large volumes of event data, including event details, artist information, and venue details, without the limitations of Excel files, which may become slow and cumbersome as data volume increases.

2. **Data Integrity:** Databases enforce data integrity constraints, reducing the risk of errors and inconsistencies in event listings and related information compared to Excel files, where data validation and integrity checks are limited.

3. **Real-time Updates:** A database enables real-time updates to event listings, ensuring that users have access to the latest information about upcoming events, whereas Excel files require manual updates and may not reflect changes promptly.

4. **Advanced Search Functionality:** Databases offer advanced search and filtering options, allowing users to narrow down their search based on criteria such as event genre, date, location, and artist, which is challenging to achieve efficiently in Excel files.

5. **Collaborative Access:** A database allows multiple users to simultaneously access and update event data, facilitating collaboration among event organizers, promoters, and users. Changes made to the database are immediately visible to all users, facilitating real-time collaboration and decision-making.

Target User

- **Event Attendees:** Music lovers seeking diverse live performances and concerts nationwide, eager to discover new artists and genres conveniently

- **Event Organizers:** Individuals or groups hosting music events with the aim to improve their audience reach, and enhance attendee experience

- **Venue Owners/ Managers:** Establishments hosting music events, seeking to attract organizers and attendees by effectively showcasing artists and their venues

- **Promoters/Marketers:** Specialists focused on driving ticket sales and event visibility across the United States

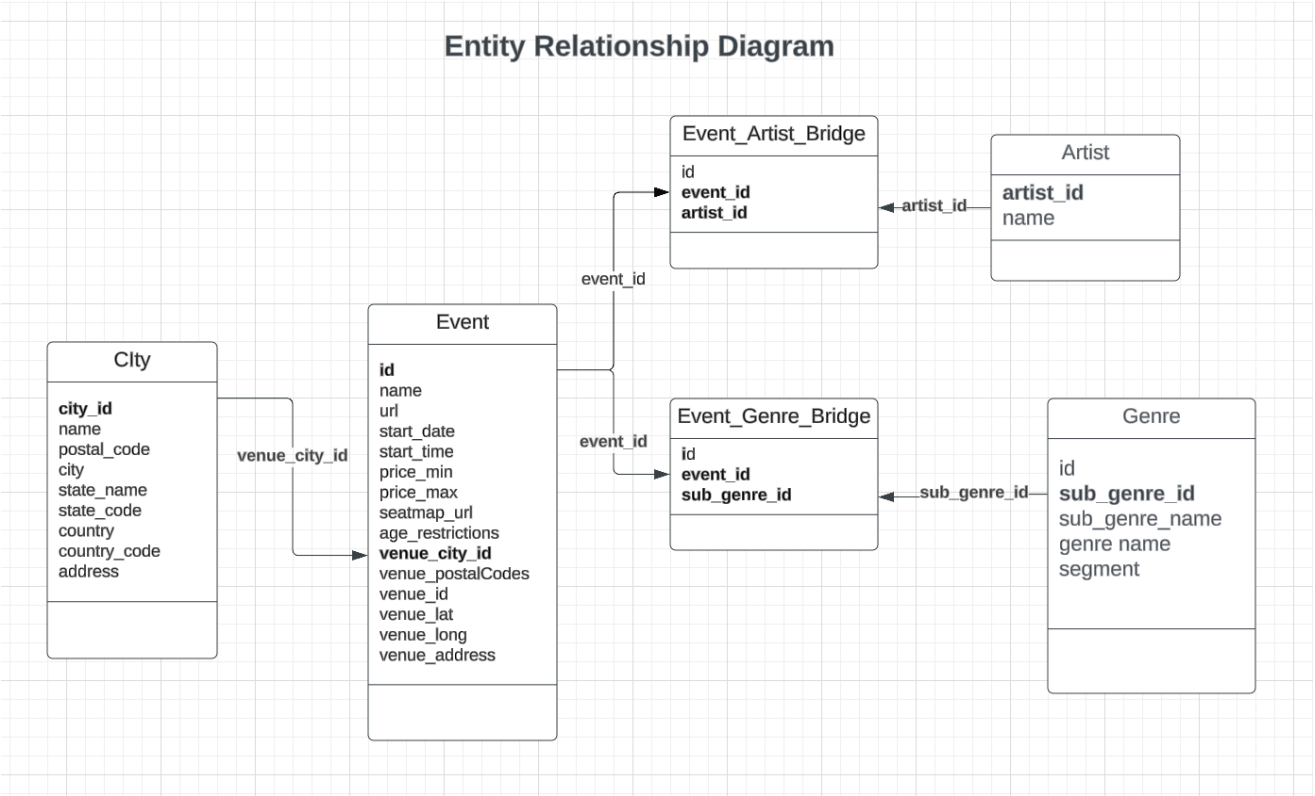
Database Administration:

- The database will be administered by a dedicated team within the organization responsible for managing the NYC Events Platform (NEP).
This team will include:

- Database Administrators (DBAs): Responsible for database setup, configuration, security management, performance optimization, data maintenance, and backup and recovery.

- System Administrators: Responsible for managing the infrastructure and servers hosting the database platform, ensuring uptime, scalability, and overall system reliability.

Data Model



Event Table:

Attributes:

- event_id (Primary Key): Unique identifier for each event. (datatype: VARCHAR)(NOT NULL)
- name: Name of the event.(datatype: VARCHAR)(NOT NULL)
- url: URL link to the event.(datatype: VARCHAR)
- start_date: Date when the event starts (datatype: DATE).
- start_time: Time when the event starts (datatype: TIME).
- price_min: Minimum price for tickets (datatype: DECIMAL).
- price_max: Maximum price for tickets (datatype: DECIMAL).
- seatmap_url: URL link to the seatmap of the venue.(datatype: VARCHAR)
- age_restrictions: Age restrictions for the event (datatype: BOOLEAN).
- venue_id: ID associated with the event venue received from Ticketmaster API response (datatype: VARCHAR)
- venue_name: venue name for the event (datatype: VARCHAR)
- venue_postalCodes: Postal code of the venue of the event (datatype: VARCHAR)
- venue_city_id: Foreign Key references the City table (datatype: VARCHAR)(NOT NULL)
  - Action: ON DELETE SET NULL - If a city is deleted, all events associated with that city should have a null value for the event. This way we can ensure no useful data gets deleted).
- venue_lat: Latitude coordinate of the venue's location.(datatype: DECIMAL)
- venue_long: Longitude coordinate of the venue's location.(datatype: DECIMAL)
- venue_address: Full address of the venue. (datatype: VARCHAR)

City Table:

Attributes:

- city_id (Primary Key): Unique identifier for each city. (datatype: VARCHAR)(NOT NULL)
- city: City where the venue is located. (datatype: VARCHAR)(NOT NULL)
- state_name: Name of the state where the venue is located. (datatype: VARCHAR)(NOT NULL)
- state_code: Code of the state where the venue is located. (datatype: VARCHAR)(NOT NULL)
- country: Country where the venue is located. (datatype: VARCHAR)(NOT NULL)
- country_code: Code of the country where the venue is located. (datatype: VARCHAR)(NOT NULL)

Genre Table:

Attributes:

- subgenre_id (Primary Key): Unique identifier for each subgenre.(datatype: VARCHAR)(NOT NULL)
- subgenre: Name of the subgenre. (datatype: VARCHAR)(NOT NULL)
- genre: Name of the genre. (datatype: VARCHAR)(NOT NULL)
- segment: Higher level grouping than genre. (datatype: VARCHAR)(NOT NULL)

Event_Genre_Bridge Table:

Purpose: Since the relationship between events and genres is many to many, this bridge table converts the relationship into two one-to-many relationship connections.

Attributes:

- id (Primary Key) : Unique identifier of each row. (datatype: VARCHAR)(NOT NULL)
- event_id (Foreign Key): Reference to the Event table indicating which event is associated with a genre. (datatype: VARCHAR)(NOT NULL)
- subgenre_id (Foreign Key): Reference to the Subgenre table indicating the subgenre of the event. (datatype: VARCHAR)(NOT NULL)

Artist Table:

Attributes:

- artist_id (Primary Key): Unique identifier for each artist. (datatype: VARCHAR)(NOT NULL)
- name: Name of the artist. (datatype: VARCHAR)(NOT NULL)

Event_Artist_Bridge_Table:

Purpose: Since the relationship between events and artists is many to many, this bridge table is used to convert the relationship to two one-to-many relationship connections.

Attributes:

- id (Primary Key) : Unique identifier of each row (datatype: VARCHAR)(NOT NULL)
- event_id (Foreign Key): Reference to the Event table indicating which event is associated with an artist. (Action: ON DELETE CASCADE - If an event is deleted, all artist associations related to that event are also deleted.) (datatype: VARCHAR)(NOT NULL)
- artist_id (Foreign Key): Reference to the Artist table indicating the artist performing at the event. (Action: ON DELETE CASCADE - If an artist is deleted, all event associations related to that artist are also deleted.) (datatype: VARCHAR)(NOT NULL)

Summary of the above design:

- Primary keys are indicated for each table to identify records within the table uniquely.
- Foreign keys are specified where there are relationships between tables to maintain referential integrity.
- Detailed descriptions are provided for each attribute, including purpose and datatype.
- Default values are not specified unless explicitly required.
- Attributes that can be set to 'null' are indicated as such in the descriptions.

Implementation Steps:

API Integration:

- Obtain an API key from Ticketmaster to access their REST API services.

- Utilize Ticketmaster API endpoints to fetch event data, including event details, attractions, venues, and artist information.

- Develop scripts to make HTTP requests to the Ticketmaster API, retrieve event data in JSON format, and parse the data for database insertion.

Database Design:

- Design the database schema to accommodate event-related data, including tables for events, location, artists, and genre associations.

- Define relationships between entities to maintain data integrity and facilitate efficient querying.

Data Import:

- Python scripts were employed to extract data from the Ticketmaster API responses via REST API calls.

- The requests library facilitated the fetching of API responses, storing the data in Python dictionaries.

- The response is parsed to gather all data points in JSON format, which is cleaned and transformed into separate fact and dimension tables.
- Data was structured into DataFrames format using the pandas library, enabling easy manipulation and organization.

- The parsed data was transformed into a structured DataFrame, encapsulating event details such as name, start date, venue information, and artist details.

- Utilizing the to_sql() function provided by pandas, the event data was seamlessly transferred to a PostgreSQL database.

- The data import process ensured consistency and integrity by inserting the parsed event data into corresponding tables within the database.

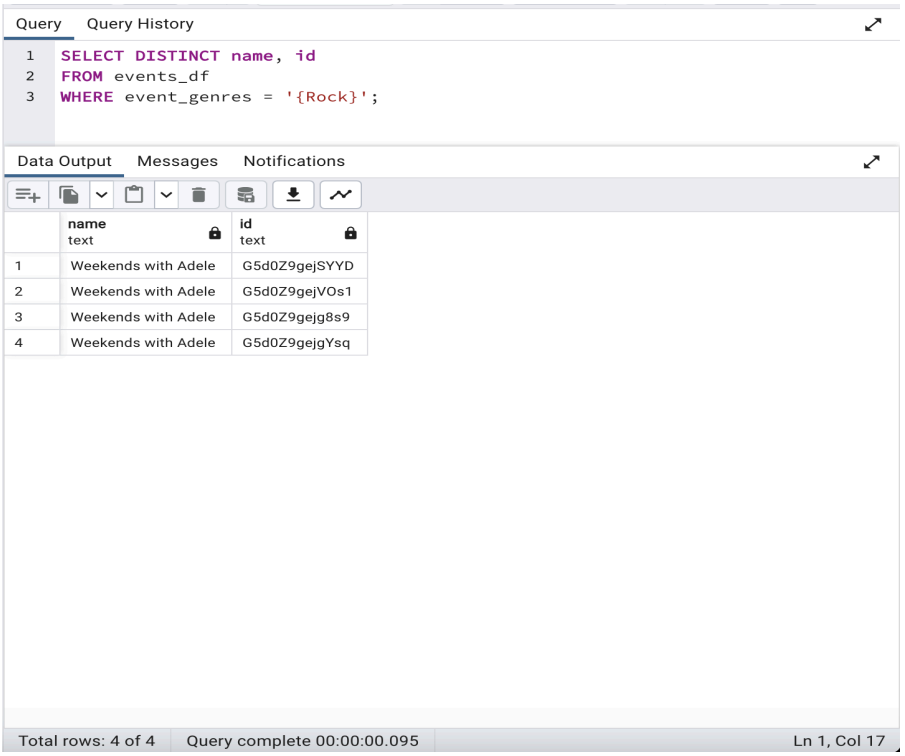
- The combination of Python, pandas, and SQL Alchemy streamlined the data import process, enabling efficient integration of event data from the Ticketmaster API into the database system.

Designing Query:

- Design SQL queries to perform various operations on the database, such as retrieving event details, and searching for events by location, date, genre, etc.

- Optimize queries for performance and efficiency to handle large volumes of event data effectively.

SQL QUERIES


Query
Query History
1 SELECT DISTINCT name, id
2 FROM events_df
3 WHERE event_genres = '{Rock}';
Data Output
Messages
Notifications
name text
id text
1 Weekends with Adele G5d0Z9gejSYYD
2 Weekends with Adele G5d0Z9gejVOs1
3 Weekends with Adele G5d0Z9gejg8s9
4 Weekends with Adele G5d0Z9gejgYsq
Total rows: 4 of 4
Query complete 00:00:00.095
Ln 1, Col 17

Query | Query History

```
1  SELECT DISTINCT name, start_time, venue_citys
2  FROM events_df
3  WHERE venue_state_names IN ('{California}', '{Nevada}');
```

Data Output | Messages | Notifications

	name text	start_time text	venue_citys text
1	Offset - Set It Off Tour	19:00:00	{Anaheim}
2	Offset - Set It Off Tour	19:00:00	{Hollywood}
3	Offset - Set It Off Tour	20:00:00	{"San Francisco"}
4	Weekends with Adele	20:00:00	{"Las Vegas"}

Total rows: 4 of 4 | Query complete 00:00:00.045 | Ln 1, Col 46

Query | Query History

```
1  SELECT COUNT (*)
2  FROM events_df
3  WHERE event_segments = '{Music}' AND event_subgenres = '{Pop}';
```

Data Output | Messages | Notifications

	count bigint
1	8

Total rows: 1 of 1 | Query complete 00:00:00.059 | Ln 1, Col 17

Query Query History

```
1  SELECT COUNT(id), name
2  FROM events_df
3  GROUP BY name;
```

Data Output Messages Notifications

	count bigint	name text
1	8	Weekends with Adele
2	2	Offset - Set It Off Tour After Party
3	30	Offset - Set It Off Tour

Total rows: 3 of 3 Query complete 00:00:00.048 Ln 1, Col 7

Query Query History

```
1  SELECT DISTINCT name, id, start_date
2  FROM events_df
3  WHERE event_genres IN (SELECT event_genres
4                         FROM events_df
5                         WHERE event_subgenres = '{"French Rap"}');
6
```

Data Output Messages Notifications

	name text	id text	start_date text
1	Offset - Set It Off Tour	1A_ZkZwGkeUwxuk	2024-04-03
2	Offset - Set It Off Tour	1avbZbd89k4ZdGv1	2024-03-22
3	Offset - Set It Off Tour	G5dlZbd2M-4PB	2024-04-07
4	Offset - Set It Off Tour	G5vYZbdC9Ed-8	2024-03-30
5	Offset - Set It Off Tour	G5vzZbd2hhfbx	2024-03-27
6	Offset - Set It Off Tour	k7vGFbd5I4AHR	2024-03-14
7	Offset - Set It Off Tour	vv1A7ZkZwGkd2ws_Q	2024-03-23
8	Offset - Set It Off Tour	vv1AFZkZwGke0Pe54	2024-03-20
9	Offset - Set It Off Tour	vv1AFZkZwGketOwxC	2024-03-19
10	Offset - Set It Off Tour	vv1AaZkZSGkeiOg8_	2024-03-29
11	Offset - Set It Off Tour	vv1AaZkZwGkeEXe8_	2024-04-01
12	Offset - Set It Off Tour	vv1AeZkZSGkeXw4Pr	2024-03-10
13	Offset - Set It Off Tour	vv1AvZkZwGkeY9NrU	2024-03-15
14	Offset - Set It Off Tour	vvG1YZbdGCsA06	2024-04-05
15	Offset - Set It Off Tour	vvG1zZbd2170TS	2024-04-10
16	Offset - Set It Off Tour After Party	vv177Zb7GkT4YpFz	2024-03-15

Total rows: 16 of 16 Query complete 00:00:00.041 Ln 1, Col 17

Concert Finder

In [ ]:
```
import pandas as pd
import requests
from sqlalchemy import create_engine
# sqlalchemy provides a set of tools for interacting with SQL databases usir
```

```
/Users/shaileshmahto/Documents/UB/Academics/CSE_560_Data_Models_and_Query_La
nguage/Databases/concert-finder/venv/lib/python3.9/site-packages/urllib3/__i
nit__.py:35: NotOpenSSLWarning: urllib3 v2 only supports OpenSSL 1.1.1+, cur
rently the 'ssl' module is compiled with 'LibreSSL 2.8.3'. See: https://gith
ub.com/urllib3/urllib3/issues/3020
  warnings.warn(
```

Data Collection using API

In [ ]:
```
# making the rest API call
url = "https://app.ticketmaster.com/discovery/v2/events?apikey=QOXJm5N0BfIVG

payload = {}
headers = {}

response = requests.request("GET", url, headers=headers, data=payload)
response_json = response.json()
```

In [ ]:
```
# parsing the json response
events_json = response_json['_embedded']['events']
events = [event_json for event_json in events_json]
```

In [ ]:
```
events_list = []
for event in events:

    name = event["name"]
    id = event["id"]
    url = event["url"]
    event_start_date = event["dates"]["start"]["localDate"]
    event_start_time = event["dates"]["start"]["localTime"]
    # TODO: Check price range for multiple values
    priceRanges_min = [priceRange["min"] for priceRange in event["priceRange
    priceRanges_max = [priceRange["max"] for priceRange in event["priceRange
    seatmap = event["seatmap"]["staticUrl"]
    ageRestrictions = event["ageRestrictions"]["legalAgeEnforced"]
    # subgenre and genre
    event_subgenres, event_genres, event_segments = [], [], []
    event_subgenres_ids, event_genres_ids, event_segments_ids = [], [], []
    for classification in event["classifications"]:
        event_subgenres.append(classification["subGenre"]["name"])
        event_genres.append(classification["genre"]["name"])
        event_segments.append(classification["segment"]["name"])
        event_subgenres_ids.append(classification["subGenre"]["id"])
        event_genres_ids.append(classification["genre"]["id"])
```

```
        event_segments_ids.append(classification["segment"]["id"])

    venues = event["_embedded"]["venues"]
    venue_ids, venue_names, venue_postalCodes, venue_citys = [], [], [], []
    venue_state_names, venue_state_codes, venue_countrys, venue_country_code
    venue_addresses, venue_lats, venue_longs = [], [], []
    for venue in venues:
        venue_names.append(venue["name"])
        venue_ids.append(venue["id"])
        venue_postalCodes.append(venue["postalCode"])
        venue_citys.append(venue["city"]["name"])
        venue_state_names.append(venue["state"]["name"])
        venue_state_codes.append(venue["state"]["stateCode"])
        venue_countrys.append(venue["country"]["name"])
        venue_country_codes.append(venue["country"]["countryCode"])

        venue_addresses.append(venue["address"]["line1"])

        venue_lats.append(venue["location"]["latitude"])
        venue_longs.append(venue["location"]["longitude"])

    attractions = event["_embedded"]["attractions"]
    artists, artist_ids = [], []
    for attraction in attractions:
        artists.append(attraction["name"])
        artist_ids.append(attraction["id"])

    rec = {
        # event table
        "name" : name,
        "url" : url,
        "id" : id ,
        "start_date": event_start_date,
        "start_time": event_start_time,
        "price_min": priceRanges_min,
        "price_max": priceRanges_max,
        "seatmap_url": seatmap,
        "age_restrictions": ageRestrictions,

        # genre table
        "event_subgenres": event_subgenres,
        "event_genres": event_genres,
        "event_segments": event_segments,
        "event_subgenres_ids": event_subgenres_ids,
        "event_genres_ids": event_genres_ids,
        "event_segments_ids": event_segments_ids,


        # event table
        "venue_ids": venue_ids,
        "venue_names": venue_names,
        "venue_postalCodes": venue_postalCodes,

        # city table
        "venue_citys": venue_citys,
        "venue_state_names": venue_state_names,
```

```
        "venue_state_codes": venue_state_codes,
        "venue_countrys": venue_countrys,
        "venue_country_codes": venue_country_codes,

        # event table
        "venue_addresses": venue_addresses,
        "venue_lats": venue_lats,
        "venue_longs": venue_longs,

        # artist table columns
        "artists": artists,
        "artist_ids": artist_ids
    }
    events_list.append(rec)
```

In [ ]:
```
# getting a list of column names using the dictionary of 1 record
columns = list(rec.keys())
columns
```

Out[ ]:
```
['name',
 'url',
 'id',
 'start_date',
 'start_time',
 'price_min',
 'price_max',
 'seatmap_url',
 'age_restrictions',
 'event_subgenres',
 'event_genres',
 'event_segments',
 'event_subgenres_ids',
 'event_genres_ids',
 'event_segments_ids',
 'venue_ids',
 'venue_names',
 'venue_postalCodes',
 'venue_citys',
 'venue_state_names',
 'venue_state_codes',
 'venue_countrys',
 'venue_country_codes',
 'venue_addresses',
 'venue_lats',
 'venue_longs',
 'artists',
 'artist_ids']
```

In [ ]:
```
event_v1_df = pd.DataFrame(data = events_list , columns = columns)
```

In [ ]:
```
event_v1_df
```

Out[ ]:

	name	url	id	start_date
0	Offset - Set It Off Tour	https://concerts.livenation.com/offset-set-it-...	vvG1zZbd2170TS	2024-04-10
1	Offset - Set It Off Tour	https://www.ticketmaster.com/offset-set-it-off...	k7vGFbd5I4AHR	2024-03-14
2	Offset - Set It Off Tour	https://concerts.livenation.com/offset-set-it-...	vv1AvZkZwGkeY9NrU	2024-03-15
3	Offset - Set It Off Tour	https://concerts.livenation.com/offset-set-it-...	G5vYZbdC9Ed-8	2024-03-30
4	Offset - Set It Off Tour	https://concerts.livenation.com/offset-set-it-...	vv1A7ZkZwGkd2ws_Q	2024-03-23
5	Offset - Set It Off Tour	https://concerts.livenation.com/offset-set-it-...	vv1AeZkZSGkeXw4Pr	2024-03-10
6	Offset - Set It Off Tour	https://concerts.livenation.com/offset-set-it-...	vv1AFZkZwGketOwxC	2024-03-19
7	Offset - Set It Off Tour	https://concerts.livenation.com/offset-set-it-...	1A4ZkZwGkdJYwWn	2024-03-12
8	Offset - Set It Off Tour	https://concerts.livenation.com/offset-set-it-...	vv1AaZkZSGkeiOg8_	2024-03-29
9	Offset - Set It Off Tour	https://concerts.livenation.com/offset-set-it-...	1avbZbd89k4ZdGv1	2024-03-22

	name	url	id	start_date
10	Offset - Set It Off Tour	https://www.ticketmaster.com/offset-set-it-off...	vvG1YZbdGCsA06	2024-04-05
11	Offset - Set It Off Tour	https://concerts.livenation.com/offset-set-it-...	1A_ZkZwGkeUwxuk	2024-04-03
12	Offset - Set It Off Tour	https://concerts.livenation.com/offset-set-it-...	G5dIZbd2M-4PB	2024-04-07
13	Offset - Set It Off Tour	https://concerts.livenation.com/offset-set-it-...	G5vzZbd2hhfbx	2024-03-27
14	Offset - Set It Off Tour	https://concerts.livenation.com/offset-set-it-...	vv1AaZkZwGkeEXe8_	2024-04-01
15	Offset - Set It Off Tour	https://concerts.livenation.com/offset-set-it-...	vv1AFZkZwGke0Pe54	2024-03-20
16	Offset - Set It Off Tour After Party	https://www.ticketmaster.com/offset-set-it-off...	vv177Zb7GkT4YpFz	2024-03-15
17	Weekends with Adele	https://www.ticketmaster.com/weekends-with-ade...	G5d0Z9gejSYYD	2024-03-01
18	Weekends with Adele	https://www.ticketmaster.com/weekends-with-ade...	G5d0Z9gejVOs1	2024-03-02
19	Weekends with Adele	https://www.ticketmaster.com/weekends-with-ade...	G5d0Z9gejg8s9	2024-03-08

20 rows × 28 columns

In [ ]:
```
# creating the dataframe
event_df = pd.DataFrame(columns=["id", "name", "url", "start_date", "start_t
```

```
genre_df = pd.DataFrame(columns=["id", "sub_genre", "genre", "segment"])

genre_event_bridge_df = pd.DataFrame(columns=["id", "event_id", "sub_genre_i

attraction_df = pd.DataFrame(columns=["id", "name"])

event_attraction_df = pd.DataFrame(columns=["id", "attraction_id", "event_ic

city_df = pd.DataFrame(columns=["id", "city", "state", "state_code", "countr
```

Working with database

In [ ]:
```
#flask_sqlalchemy integrates SQLAlchemy into youout Flask application to per
pip install flask_sqlalchemy
```

```
Defaulting to user installation because normal site-packages is not writeabl
e
Requirement already satisfied: flask_sqlalchemy in /Users/abhiroopghosh/Libr
ary/Python/3.9/lib/python/site-packages (3.1.1)
Requirement already satisfied: flask>=2.2.5 in /Users/abhiroopghosh/Library/
Python/3.9/lib/python/site-packages (from flask_sqlalchemy) (3.0.2)
Requirement already satisfied: sqlalchemy>=2.0.16 in /Users/abhiroopghosh/Li
brary/Python/3.9/lib/python/site-packages (from flask_sqlalchemy) (2.0.27)
Requirement already satisfied: Werkzeug>=3.0.0 in /Users/abhiroopghosh/Libra
ry/Python/3.9/lib/python/site-packages (from flask>=2.2.5->flask_sqlalchemy)
(3.0.1)
Requirement already satisfied: blinker>=1.6.2 in /Users/abhiroopghosh/Librar
y/Python/3.9/lib/python/site-packages (from flask>=2.2.5->flask_sqlalchemy)
(1.7.0)
Requirement already satisfied: Jinja2>=3.1.2 in /Users/abhiroopghosh/Librar
y/Python/3.9/lib/python/site-packages (from flask>=2.2.5->flask_sqlalchemy)
(3.1.3)
Requirement already satisfied: click>=8.1.3 in /Users/abhiroopghosh/Library/
Python/3.9/lib/python/site-packages (from flask>=2.2.5->flask_sqlalchemy)
(8.1.7)
Requirement already satisfied: itsdangerous>=2.1.2 in /Users/abhiroopghosh/L
ibrary/Python/3.9/lib/python/site-packages (from flask>=2.2.5->flask_sqlalch
emy) (2.1.2)
Requirement already satisfied: importlib-metadata>=3.6.0 in /Users/abhiroopg
hosh/Library/Python/3.9/lib/python/site-packages (from flask>=2.2.5->flask_s
qlalchemy) (7.0.1)
Requirement already satisfied: zipp>=0.5 in /Users/abhiroopghosh/Library/Pyth
on/3.9/lib/python/site-packages (from importlib-metadata>=3.6.0->flask>=2.
2.5->flask_sqlalchemy) (3.17.0)
Requirement already satisfied: MarkupSafe>=2.0 in /Users/abhiroopghosh/Libra
ry/Python/3.9/lib/python/site-packages (from Jinja2>=3.1.2->flask>=2.2.5->fl
ask_sqlalchemy) (2.1.5)
Requirement already satisfied: typing-extensions>=4.6.0 in /Users/abhiroopgh
osh/Library/Python/3.9/lib/python/site-packages (from sqlalchemy>=2.0.16->fl
ask_sqlalchemy) (4.9.0)
WARNING: You are using pip version 21.2.4; however, version 24.0 is availabl
e.
You should consider upgrading via the '/Library/Developer/CommandLineTools/u
sr/bin/python3 -m pip install --upgrade pip' command.
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:
```
pip install psycopg2-binary
```

```
Defaulting to user installation because normal site-packages is not writeabl
e
Collecting psycopg2-binary
  Downloading psycopg2_binary-2.9.9-cp39-cp39-macosx_11_0_arm64.whl (2.6 MB)
     |████████████████████████████████| 2.6 MB 2.3 MB/s eta 0:00:01
Installing collected packages: psycopg2-binary
Successfully installed psycopg2-binary-2.9.9
WARNING: You are using pip version 21.2.4; however, version 24.0 is availabl
e.
You should consider upgrading via the '/Library/Developer/CommandLineTools/u
sr/bin/python3 -m pip install --upgrade pip' command.
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:
```
# credentials only included in the current version of file, final versio  wi
# Specify your PostgreSQL database connection parameters
db_username = 'postgres' # same dn_username for all of us
db_password = '9820619960' #wahtever your server password is for access
db_host = 'localhost'  # hostname(localhost in our case)
db_port = '5432'  # Default PostgreSQL port is 5432 (should be same for all
db_name = 'DMQL NYSYNC'

# Construct the database connection string
connection_string = f'postgresql://{db_username}:{db_password}@{db_host}:{db

# Create the engine
engine = create_engine(connection_string)
```

In [ ]:
```
# Load data into SQL tables
event_df.to_sql('events', con=engine, index=False, if_exists='append')
genre_df.to_sql('genres', con=engine, index=False, if_exists='append')
genre_event_bridge_df.to_sql('genre_event_bridge', con=engine, index=False,
attraction_df.to_sql('attractions', con=engine, index=False, if_exists='appe
event_attraction_df.to_sql('event_attractions', con=engine, index=False, if_
city_df.to_sql('cities', con=engine, index=False, if_exists='append')
```

Out[ ]:
```
0
```

In [ ]:
```
event_v1_df.to_sql('events_df', con=engine, index=False, if_exists='append')
```

Out[ ]:
```
20
```