**Game Analytics: Unlocking Tennis Data with SportRadar API**

**Business Objectives**

1. **Event Exploration**: Enable users to navigate through competition hierarchies, such as ATP Vienna events, to explore and understand the structure of tennis tournaments.
2. **Trend Analysis**: Visualize the distribution of events by type, gender, and competition level to identify trends and patterns in tennis competitions.
3. **Performance Insights**: Analyze player participation across singles and doubles events to gain insights into player performance and engagement.
4. **Decision Support**: Offer data-driven insights to event organizers or sports bodies for resource allocation and strategic planning.

**Skills Takeaway**

* **Python Scripting**: Developing scripts to interact with APIs and manage data.
* **Data Collection**: Using API integration to extract and transform JSON data into a structured format.
* **Data Management**: Utilizing SQL for database management, including designing schemas and executing queries.
* **Streamlit Application Development**: Building interactive dashboards for data visualization and analysis.

**Domain**

**Sports/Data Analytics**

**Approach**

**Data Extraction**

* **API Integration**: Parse and extract data from SportRadar JSON responses.
* **Data Transformation**: Convert nested JSON structures into a flat relational schema for analysis.

**Data Storage**

* **SQL Database**: Create a well-structured SQL database with appropriate data types and primary keys to store the extracted data.

**Data Analysis**

* **SQL Queries**: Execute various SQL queries to derive insights, such as:
  + Listing all competitions along with their category names.
  + Counting the number of competitions in each category.
  + Finding all competitions of a specific type (e.g., doubles).
  + Identifying parent competitions and their sub-competitions.
  + Analyzing the distribution of competition types by category.
  + Listing all top-level competitions with no parent.
  + Retrieving details of venues and their associated complexes.
  + Counting the number of venues in each complex.
  + Getting details of venues in specific countries.
  + Identifying complexes with multiple venues.
  + Listing venues grouped by country.
  + Finding all venues for a specific complex.
  + Getting all competitors with their rank and points.
  + Finding competitors ranked in the top 5.
  + Listing competitors with no rank movement.
  + Getting the total points of competitors from specific countries.
  + Counting the number of competitors per country.
  + Finding competitors with the highest points in the current week.

**Project Deliverables**

1. **SQL Database**: Populated with structured sports event data from the SportRadar API.
2. **API Scripts**: Automate data extraction and transform JSON into a relational format.
3. **Streamlit App**: Interactive tool for exploring and visualizing competition data.
4. **Documentation**: Detailed report on workflow, schema design, challenges, and insights.

**Project Guidelines**

**Coding Standards**

* Use meaningful names for variables, functions, and database tables.
* Follow PEP 8 guidelines for Python code.
* Modularize code into functions or classes for better readability and reusability.
* Implement error handling for API errors and SQL exceptions.
* Document code with docstrings and comments.

**SQL Database Practices**

* Normalize tables to avoid redundancy and ensure efficient data storage.
* Use indexes to optimize query performance.
* Follow consistent naming conventions for tables and fields.

**Streamlit Application Development**

* Ensure the UI is responsive with interactive widgets for filters.
* Maintain a minimalist design for a smooth user experience.
* Optimize performance by avoiding loading all data at once; use pagination or batch processing.

**General Best Practices**

* Test each component (API requests, SQL queries, Streamlit app) frequently during development.
* Maintain backups of the SQL database and code.
* Provide a README file with setup instructions, project objectives, and a demo walkthrough.

**Technical Tags**

* **Languages**: Python
* **Database**: MySQL/PostgreSQL
* **Application**: Streamlit
* **API Integration**: SportRadar API

**Python Code For Extracting Data from API**

**1) fetch\_complexes.ipynb**

pip install pymysql

import pymysql

import requests

# API details

api\_key="9vwTpZnV8T3jHXHKfp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

# Fetch data from API

url = f"https://api.sportradar.com/tennis/trial/v3/en/complexes.json?api\_key={api\_key}"

data = requests.get(url).json()

# Connect using PyMySQL

conn = pymysql.connect(

host="host\_name",

user="root",

password="Password",

database="tennis"

)

cursor = conn.cursor()

# Insert complexes and venues

for comp in data.get("complexes", []):

cursor.execute("""

INSERT INTO Complexes (complex\_id, complex\_name)

VALUES (%s, %s)

ON DUPLICATE KEY UPDATE complex\_name = %s

""", (comp["id"], comp["name"], comp["name"]))

for v in comp.get("venues", []):

cursor.execute("""

INSERT INTO Venues (venue\_id, venue\_name, city\_name, country\_name, country\_code, timezone, complex\_id)

VALUES (%s, %s, %s, %s, %s, %s, %s)

ON DUPLICATE KEY UPDATE

venue\_name = VALUES(venue\_name),

city\_name = VALUES(city\_name),

country\_name = VALUES(country\_name),

country\_code = VALUES(country\_code),

timezone = VALUES(timezone),

complex\_id = VALUES(complex\_id)

""", (

v["id"],

v["name"],

v["city\_name"],

v["country\_name"],

v["country\_code"],

v["timezone"],

comp["id"]

))

# Finalize and close

conn.commit()

cursor.close()

conn.close()

**2) fetch\_rankings.ipynb**

pip install pymysql

import requests

import pymysql

# API details

api\_key = "9vwTpZnV8T3jHXHK\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

url = f"https://api.sportradar.com/tennis/trial/v3/en/double\_competitors\_rankings.json?api\_key={api\_key}"

# Fetch data from API

data = requests.get(url).json()

# Connect to MySQL using PyMySQL

conn = pymysql.connect(

host="hsot\_name",

user="root",

password="password",

database="tennis"

)

cursor = conn.cursor()

# Loop through the rankings and then through competitor\_rankings

for ranking in data.get("rankings", []):

for competitor\_ranking in ranking.get("competitor\_rankings", []):

c = competitor\_ranking["competitor"]

# Insert or update Competitors table

cursor.execute("""

INSERT INTO Competitor (competitor\_id, name, country, country\_code, abbreviation)

VALUES (%s, %s, %s, %s, %s)

ON DUPLICATE KEY UPDATE

name = VALUES(name),

country = VALUES(country),

country\_code = VALUES(country\_code),

abbreviation = VALUES(abbreviation)

""", (c["id"], c["name"], c.get("country", ""), c.get("country\_code", ""), c.get("abbreviation", "")))

# Insert into Competitor\_Rankings table

cursor.execute("""

INSERT INTO Competitor\_Rankings (`rank`, movement, points, competitions\_played, competitor\_id)

VALUES (%s, %s, %s, %s, %s)

""", (competitor\_ranking["rank"], competitor\_ranking.get("movement", 0), competitor\_ranking.get("points", 0), competitor\_ranking.get("competitions\_played", 0), c["id"]))

# Commit and close

conn.commit()

cursor.close()

conn.close()

**3) fetch\_competitions.ipynb**

pip install pymysql

import requests

import pymysql

# API details

api\_key = "9vwTpZnV8T3jHXHKfpAU\*\*\*\*\*\*\*\*\*\*\*\*"

url = f"https://api.sportradar.com/tennis/trial/v3/en/competitions.json?api\_key={api\_key}"

# Fetch data from API

data = requests.get(url).json()

# Connect to MySQL using PyMySQL

conn = pymysql.connect(

host="host\_name",

user="root",

password="Password",

database="tennis"

)

cursor = conn.cursor()

# Insert data into MySQL tables

for comp in data.get("competitions", []):

cat = comp["category"]

# Insert or update into Categories table

cursor.execute("""

INSERT INTO Categories (category\_id, category\_name)

VALUES (%s, %s)

ON DUPLICATE KEY UPDATE category\_name = VALUES(category\_name)

""", (cat["id"], cat["name"]))

# Insert or update into Competitions table

cursor.execute("""

INSERT INTO Competitions (competition\_id, competition\_name, parent\_id, type, gender, category\_id)

VALUES (%s, %s, %s, %s, %s, %s)

ON DUPLICATE KEY UPDATE

competition\_name = VALUES(competition\_name),

parent\_id = VALUES(parent\_id),

type = VALUES(type),

gender = VALUES(gender),

category\_id = VALUES(category\_id)

""", (

comp["id"],

comp["name"],

comp.get("parent\_id"),

comp.get("type"),

comp.get("gender"),

cat["id"]

))

conn.commit()

cursor.close()

conn.close()

**MYSQL Results**

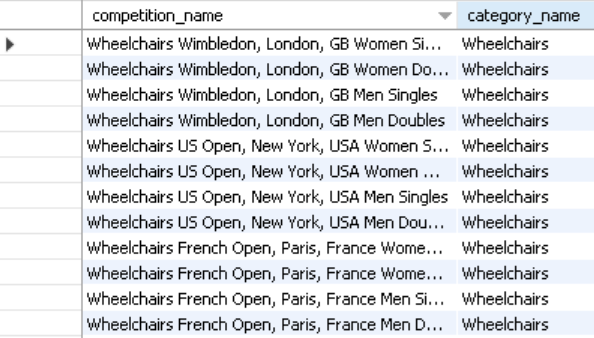
**Competitions and Category Query Results**

1) List all competitions along with their category name:

SELECT c.competition\_name, cat.category\_name

FROM Competitions c

JOIN Categories cat ON c.category\_id = cat.category\_id;



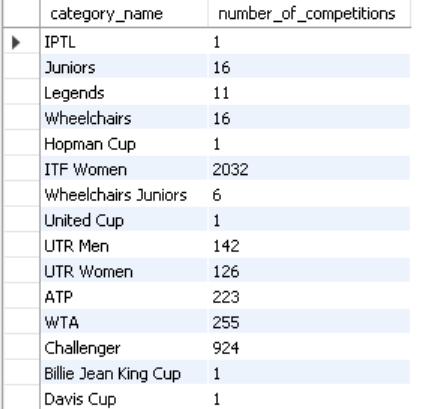
2) Count the number of competitions in each category:

SELECT cat.category\_name, COUNT(c.competition\_id) AS number\_of\_competitions

FROM Categories cat

LEFT JOIN Competitions c ON cat.category\_id = c.category\_id

GROUP BY cat.category\_name;

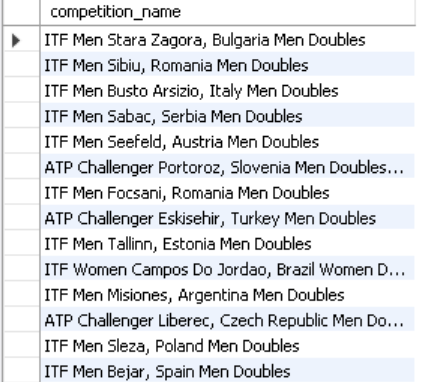


3) Find all competitions of type 'doubles':

SELECT competition\_name

FROM Competitions

WHERE type = 'doubles';



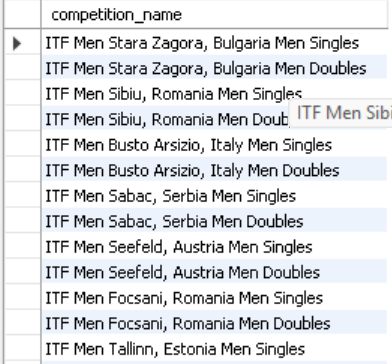
4) Get competitions that belong to a specific category (e.g., ITF Men):

SELECT c.competition\_name

FROM Competitions c

JOIN Categories cat ON c.category\_id = cat.category\_id

WHERE cat.category\_name = 'ITF Men';



5) Identify parent competitions and their sub-competitions:

SELECT p.competition\_name AS parent\_competition, s.competition\_name AS sub\_competition

FROM Competitions p

LEFT JOIN Competitions s ON p.competition\_id = s.parent\_id

WHERE p.parent\_id IS NULL;



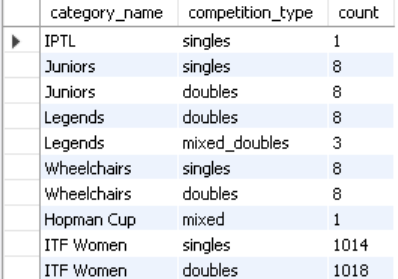
6) Analyze the distribution of competition types by category:

SELECT cat.category\_name, c.type AS competition\_type, COUNT(\*) AS count

FROM Competitions c

JOIN Categories cat ON c.category\_id = cat.category\_id

GROUP BY cat.category\_name, c.type;

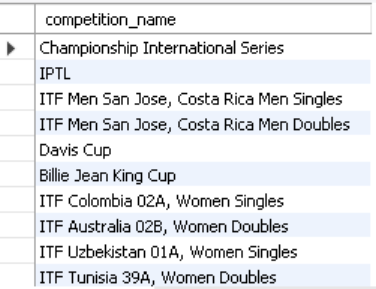


7) List all competitions with no parent (top-level competitions):

SELECT competition\_name

FROM Competitions

WHERE parent\_id IS NULL;



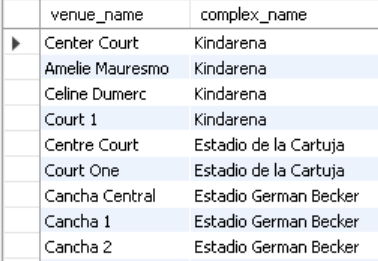
**Queries for Complexes and Venues**

1) List all venues along with their associated complex name:

SELECT v.venue\_name, c.complex\_name

FROM Venues v

JOIN Complexes c ON v.complex\_id = c.complex\_id;



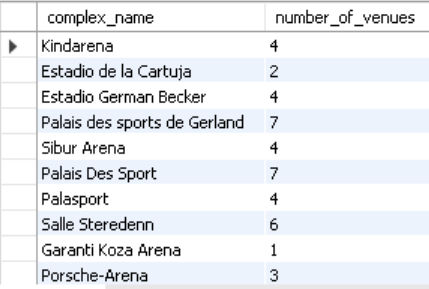
2) Count the number of venues in each complex:

SELECT c.complex\_name, COUNT(v.venue\_id) AS number\_of\_venues

FROM Complexes c

LEFT JOIN Venues v ON c.complex\_id = v.complex\_id

GROUP BY c.complex\_name;

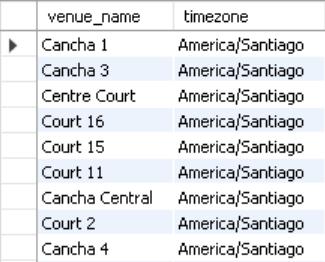


3) Get details of venues in a specific country (e.g., Chile):

SELECT v.venue\_name, v.timezone

FROM Venues v

WHERE v.country\_name = 'Chile';



4) Identify all venues and their timezones:

SELECT venue\_name, timezone

FROM Venues;



5) Find complexes that have more than one venue:

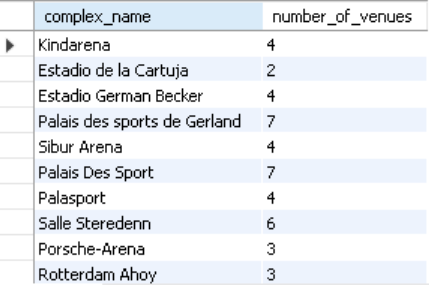
SELECT c.complex\_name, COUNT(v.venue\_id) AS number\_of\_venues

FROM Complexes c

LEFT JOIN Venues v ON c.complex\_id = v.complex\_id

GROUP BY c.complex\_name

HAVING COUNT(v.venue\_id) > 1;

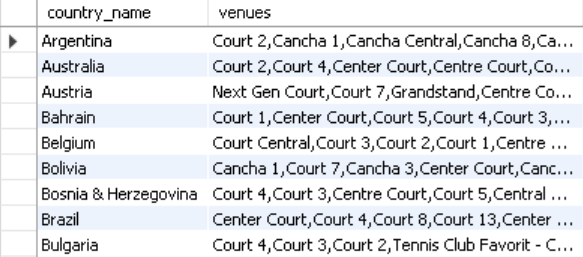


6) List venues grouped by country:

SELECT country\_name, GROUP\_CONCAT(venue\_name) AS venues

FROM Venues

GROUP BY country\_name;



7) Find all venues for a specific complex (e.g., Nacional):

SELECT v.venue\_name

FROM Venues v

JOIN Complexes c ON v.complex\_id = c.complex\_id

WHERE c.complex\_name = 'Nacional';



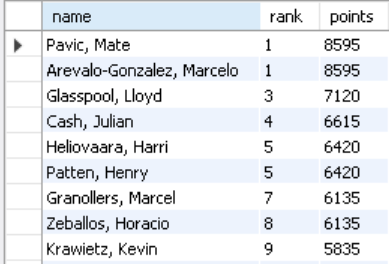
Queries for Rankings and Competitors

1) Get all competitors with their rank and points:

SELECT c.name, cr.rank, cr.points

FROM Competitor c

JOIN Competitor\_Rankings cr ON c.competitor\_id = cr.competitor\_id;



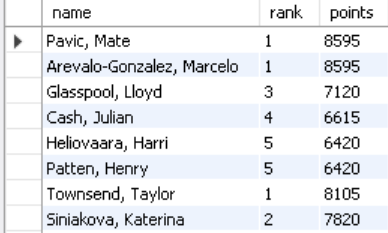
2) Find competitors ranked in the top 5:

SELECT c.name, cr.rank, cr.points

FROM Competitor c

JOIN Competitor\_Rankings cr ON c.competitor\_id = cr.competitor\_id

WHERE cr.rank <= 5;



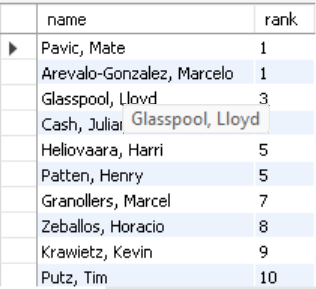
3) List competitors with no rank movement (stable rank):

SELECT c.name, cr.rank

FROM Competitor c

JOIN Competitor\_Rankings cr ON c.competitor\_id = cr.competitor\_id

WHERE cr.movement = 0;



4) Get the total points of competitors from a specific country (e.g., Croatia):

SELECT SUM(cr.points) AS total\_points

FROM Competitor c

JOIN Competitor\_Rankings cr ON c.competitor\_id = cr.competitor\_id

WHERE c.country = 'Croatia';

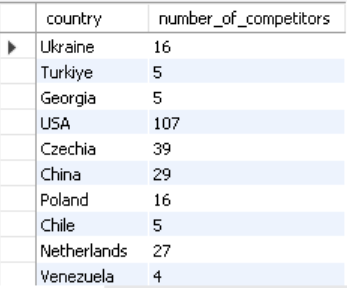
***ANS total\_points -16742***

5) Count the number of competitors per country:

SELECT c.country, COUNT(\*) AS number\_of\_competitors

FROM Competitor c

GROUP BY c.country;



6) Find competitors with the highest points in the current week:

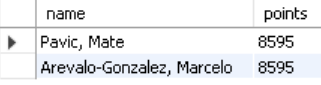
#To find competitors with the highest points, you can use a subquery to determine the maximum points and then find competitors with those points:

SELECT c.name, cr.points

FROM Competitor c

JOIN Competitor\_Rankings cr ON c.competitor\_id = cr.competitor\_id

WHERE cr.points = (SELECT MAX(points) FROM Competitor\_Rankings);



**Insights from the SQL queries**

**Competitions and Category Query Results**

1. **List all competitions along with their category name:**
   * **Query:** This query retrieves the names of all competitions along with their respective category names.
   * **Insight:** This provides a comprehensive list of all competitions categorized by their type, which can help in understanding the distribution and variety of competitions available.
2. **Count the number of competitions in each category:**
   * **Query:** This query counts the number of competitions in each category.
   * **Insight:** This helps in identifying which categories have the most competitions. For example, "ITF Women" has the highest number of competitions, indicating a strong focus or popularity in that category.
3. **Find all competitions of type 'doubles':**
   * **Query:** This query lists all competitions that are of the type 'doubles'.
   * **Insight:** This can help in analyzing the distribution and frequency of doubles competitions across different locations and categories.
4. **Get competitions that belong to a specific category (e.g., ITF Men):**
   * **Query:** This query retrieves all competitions within a specific category, in this case, "ITF Men".
   * **Insight:** This provides a focused view of competitions within a specific category, useful for targeted analysis or reporting.
5. **Identify parent competitions and their sub-competitions:**
   * **Query:** This query identifies parent competitions and their sub-competitions.
   * **Insight:** This helps in understanding the hierarchical structure of competitions, showing which competitions are part of larger events or series.
6. **Analyze the distribution of competition types by category:**
   * **Query:** This query analyzes the distribution of competition types (e.g., singles, doubles) within each category.
   * **Insight:** This provides insights into the variety of competition types within each category, helping to understand the diversity and focus areas.
7. **List all competitions with no parent (top-level competitions):**
   * **Query:** This query lists all top-level competitions that do not have a parent competition.
   * **Insight:** This helps in identifying the main or primary competitions that are not part of any sub-series or events.

**Queries for Complexes and Venues**

1. **List all venues along with their associated complex name:**
   * **Query:** This query lists all venues along with the names of their associated complexes.
   * **Insight:** This provides a comprehensive list of venues and their associated complexes, useful for logistical planning and understanding venue distribution.
2. **Count the number of venues in each complex:**
   * **Query:** This query counts the number of venues in each complex.
   * **Insight:** This helps in identifying which complexes have the most venues, indicating larger or more significant complexes.
3. **Get details of venues in a specific country (e.g., Chile):**
   * **Query:** This query retrieves details of venues located in a specific country.
   * **Insight:** This provides a focused view of venues within a specific country, useful for regional analysis or planning.
4. **Identify all venues and their time zones:**
   * **Query:** This query lists all venues along with their respective time zones.
   * **Insight:** This helps in understanding the geographical distribution of venues and can aid in scheduling and time management for events.
5. **Find complexes that have more than one venue:**
   * **Query:** This query identifies complexes that have more than one venue.
   * **Insight:** This helps in identifying larger complexes with multiple venues, which might be key locations for hosting multiple events.
6. **List venues grouped by country:**
   * **Query:** This query lists venues grouped by their country.
   * **Insight:** This provides a regional view of venue distribution, useful for understanding the global spread of venues.
7. **Find all venues for a specific complex (e.g., Nacional):**
   * **Query:** This query retrieves all venues associated with a specific complex.
   * **Insight:** This provides a focused view of venues within a specific complex, useful for detailed logistical planning.

**Queries for Rankings and Competitors**

1. **Get all competitors with their rank and points:**
   * **Query:** This query retrieves all competitors along with their rank and points.
   * **Insight:** This provides a comprehensive list of competitors and their standings, useful for overall performance analysis.
2. **Find competitors ranked in the top 5:**
   * **Query:** This query lists competitors who are ranked in the top 5.
   * **Insight:** This helps in identifying the top-performing competitors, useful for highlighting elite players.
3. **List competitors with no rank movement (stable rank):**
   * **Query:** This query lists competitors whose rank has not changed.
   * **Insight:** This helps in identifying competitors with stable performance, indicating consistency.
4. **Get the total points of competitors from a specific country (e.g., Croatia):**
   * **Query:** This query calculates the total points of competitors from a specific country.
   * **Insight:** This provides a regional performance metric, useful for understanding the strength of competitors from specific countries.
5. **Count the number of competitors per country:**
   * **Query:** This query counts the number of competitors from each country.
   * **Insight:** This helps in understanding the global distribution of competitors, highlighting countries with a strong presence in the sport.
6. **Find competitors with the highest points in the current week:**
   * **Query:** This query identifies competitors with the highest points in the current week.
   * **Insight:** This helps in identifying the top performers in the current week, useful for recent performance analysis.

**Streamlitapp.py**

import streamlit as st

import pymysql

import pandas as pd

import plotly.express as px

# Set page configuration

st.set\_page\_config(page\_title="Tennis Rankings Explorer", layout="wide")

# Custom CSS for styling

st.markdown("""

<style>

.main {

background: linear-gradient(135deg, #8bc34a, #4caf50);

color: #2c3e50;

font-family: 'Arial', sans-serif;

}

.sidebar .sidebar-content {

background: linear-gradient(135deg, #2c3e50, #34495e);

color: white;

padding: 20px;

}

.sidebar .sidebar-content .stSelectbox,

.sidebar .sidebar-content .stSlider,

.sidebar .sidebar-content .stTextInput {

background-color: rgba(255, 255, 255, 0.2);

color: white;

border-radius: 5px;

padding: 10px;

margin-bottom: 20px;

border: 1px solid rgba(255, 255, 255, 0.5);

}

.sidebar .sidebar-content label {

color: white;

font-size: 16px;

margin-bottom: 10px;

display: block;

}

.heading, .subheading {

color: white;

background-color: rgba(44, 62, 80, 0.7);

padding: 10px;

border-radius: 5px;

box-shadow: 0 2px 4px rgba(0, 0, 0, 0.2);

margin: 10px 0;

text-align: center;

}

.heading {

font-size: 2.5em;

}

.subheading {

font-size: 1.5em;

}

.content-box {

background-color: rgba(255, 255, 255, 0.9);

border-radius: 10px;

padding: 20px;

margin: 10px auto;

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);

}

</style>

""", unsafe\_allow\_html=True)

# Function to create a database connection using pymysql

def get\_connection():

try:

return pymysql.connect(

host="localhost",

user="root",

password="Laptop@321",

database="tennis"

)

except pymysql.Error as err:

st.error(f"Error connecting to MySQL: {err}")

return None

# Function to load data from the database

@st.cache\_data

def load\_data(query):

conn = get\_connection()

if conn is not None:

try:

df = pd.read\_sql(query, conn)

conn.close()

return df

except pymysql.Error as err:

st.error(f"Error loading data: {err}")

return pd.DataFrame()

# Load datasets

df\_rankings = load\_data("""

SELECT r.rank, r.movement, r.points, r.competitions\_played, c.name AS competitor\_name, c.country, c.abbreviation

FROM Competitor\_Rankings r

JOIN Competitor c ON r.competitor\_id = c.competitor\_id

""")

df\_venues = load\_data("""

SELECT v.venue\_id, v.venue\_name, v.city\_name, v.country\_name, v.country\_code, v.timezone, c.complex\_name

FROM Venues v

JOIN Complexes c ON v.complex\_id = c.complex\_id

""")

# Sidebar filters

with st.sidebar:

st.markdown('<div class="heading">Filters</div>', unsafe\_allow\_html=True)

year = st.selectbox("Year", [2024], key="year\_select")

week = st.selectbox("Week", list(range(1, 53)), key="week\_select")

rank\_range = st.slider("Rank Range", 1, 100, (1, 24), key="rank\_slider")

# Apply filters

df\_rankings = df\_rankings[(df\_rankings['rank'] >= rank\_range[0]) & (df\_rankings['rank'] <= rank\_range[1])]

# Main content

st.markdown('<div class="heading">🎾 Tennis Rankings Explorer</div>', unsafe\_allow\_html=True)

# Rankings Section

st.markdown('<div class="subheading">Rankings</div>', unsafe\_allow\_html=True)

st.markdown('<div class="content-box">', unsafe\_allow\_html=True)

if not df\_rankings.empty:

st.dataframe(df\_rankings)

else:

st.warning("No rankings data available.")

st.markdown('</div>', unsafe\_allow\_html=True)

# Visualizations

col1, col2 = st.columns(2)

with col1:

st.markdown('<div class="subheading">Rankings Distribution</div>', unsafe\_allow\_html=True)

st.markdown('<div class="content-box">', unsafe\_allow\_html=True)

if not df\_rankings.empty:

fig = px.histogram(df\_rankings, x='rank', title='Distribution of Rankings', color\_discrete\_sequence=['#3498db'])

st.plotly\_chart(fig, use\_container\_width=True)

else:

st.warning("No data available for rankings distribution.")

st.markdown('</div>', unsafe\_allow\_html=True)

with col2:

st.markdown('<div class="subheading">Points Distribution</div>', unsafe\_allow\_html=True)

st.markdown('<div class="content-box">', unsafe\_allow\_html=True)

if not df\_rankings.empty:

fig = px.histogram(df\_rankings, x='points', title='Distribution of Points', color\_discrete\_sequence=['#2ecc71'])

st.plotly\_chart(fig, use\_container\_width=True)

else:

st.warning("No data available for points distribution.")

st.markdown('</div>', unsafe\_allow\_html=True)

# Country-wise Stats

st.markdown('<div class="subheading">Country-wise Statistics</div>', unsafe\_allow\_html=True)

st.markdown('<div class="content-box">', unsafe\_allow\_html=True)

if not df\_rankings.empty:

country\_df = df\_rankings.groupby("country").agg(

Competitors=('competitor\_name', 'count'),

Avg\_Points=('points', 'mean')

).reset\_index().sort\_values(by='Competitors', ascending=False)

fig = px.bar(country\_df, x='country', y='Competitors', color='Avg\_Points',

title='Competitors by Country', labels={'country': 'Country', 'Competitors': 'Number of Competitors'},

color\_continuous\_scale=px.colors.sequential.Viridis)

st.plotly\_chart(fig, use\_container\_width=True)

else:

st.warning("No data available for country-wise stats.")

st.markdown('</div>', unsafe\_allow\_html=True)

# Additional Insights

st.markdown('<div class="subheading">Competitions Played vs Points</div>', unsafe\_allow\_html=True)

st.markdown('<div class="content-box">', unsafe\_allow\_html=True)

if not df\_rankings.empty:

fig = px.scatter(df\_rankings, x='competitions\_played', y='points', color='country',

title='Competitions Played vs Points', hover\_name='competitor\_name')

st.plotly\_chart(fig, use\_container\_width=True)

else:

st.warning("No data available for competitions played vs points.")

st.markdown('</div>', unsafe\_allow\_html=True)

# Venues and Complexes Section

st.markdown('<div class="subheading">Venues and Complexes</div>', unsafe\_allow\_html=True)

st.markdown('<div class="content-box">', unsafe\_allow\_html=True)

if not df\_venues.empty:

st.dataframe(df\_venues)

else:

st.warning("No venues data available.")

st.markdown('</div>', unsafe\_allow\_html=True)

# Search Competitors Layout

left\_column, right\_column = st.columns([1, 2])

with left\_column:

st.markdown('<div class="subheading">Search Competitors</div>', unsafe\_allow\_html=True)

competitor\_name = st.text\_input("Competitor Name", key="competitor\_search")

with right\_column:

if competitor\_name:

competitor\_data = df\_rankings[df\_rankings['competitor\_name'].str.contains(competitor\_name, case=False)]

st.markdown('<div class="content-box">', unsafe\_allow\_html=True)

if not competitor\_data.empty:

st.markdown(f'<div class="subheading">Details for {competitor\_name}</div>', unsafe\_allow\_html=True)

st.dataframe(competitor\_data)

else:

st.warning("No data found for the specified competitor.")

st.markdown('</div>', unsafe\_allow\_html=True)