import pymysql

import pandas as pd

import streamlit as st

import plotly.express as px

# --- DATABASE CONNECTION ---

@st.cache\_resource # Keep using cache\_resource for the connection

def get\_connection():

    # Ensure you handle potential connection errors in a real app

    try:

        conn = pymysql.connect(

            host='localhost',

            user='root',

            password='',        # Use environment variables for sensitive info in production

            database='Tennis\_Game',

            cursorclass=pymysql.cursors.DictCursor # Optional: Easier to work with results

        )

        return conn

    except pymysql.Error as e:

        st.error(f"Database connection error: {e}")

        return None # Return None if connection fails

# --- FETCH DATA FROM SQL ---

@st.cache\_data # Keep using cache\_data for query results

def fetch\_table(query):

    conn = get\_connection()

    if conn: # Check if connection was successful

        try:

            # Using 'with' ensures the cursor is closed automatically

            with conn.cursor() as cursor:

                cursor.execute(query)

                result = cursor.fetchall()

            # Create DataFrame from the list of dictionaries (if using DictCursor)

            # If not using DictCursor, pd.read\_sql might be simpler,

            # but manual fetching gives more control over errors.

            df = pd.DataFrame(result)

            # conn.close() # Don't close here if using @st.cache\_resource

            return df

        except pymysql.Error as e:

            st.error(f"Error fetching data: {e}")

            return pd.DataFrame() # Return empty DataFrame on error

        # Note: pd.read\_sql might implicitly handle cursor/connection closure differently

        # return pd.read\_sql(query, conn) # Original way also works

    else:

        return pd.DataFrame() # Return empty DataFrame if no connection

# --- Initialize Session State ---

# This ensures these variables exist across script reruns for the user session

if 'filters\_applied' not in st.session\_state:

    st.session\_state.filters\_applied = False

    st.session\_state.submitted\_country = None

    st.session\_state.submitted\_category = None

# --- Load Data ---

# Put data loading inside a function or check to avoid reloading constantly if possible

# For simplicity here, we load it once. Use caching effectively.

conn = get\_connection()

if conn: # Only proceed if connection is established

    df\_competitors = fetch\_table("SELECT \* FROM competitors")

    df\_rankings = fetch\_table("SELECT \* FROM competitor\_rankings")

    df\_complexes = fetch\_table("SELECT \* FROM complexes")

    df\_venues = fetch\_table("SELECT \* FROM venues")

    df\_category = fetch\_table("SELECT \* FROM categories")

    df\_competition = fetch\_table("SELECT \* FROM competitions")

    # Check if DataFrames are empty before proceeding

    if not df\_competitors.empty and not df\_rankings.empty:

        df\_merged = pd.merge(df\_rankings, df\_competitors, on="competitor\_id", how="left")

    else:

        df\_merged = pd.DataFrame() # Create empty if data missing

    if not df\_venues.empty and not df\_complexes.empty:

        venue\_info = pd.merge(df\_venues, df\_complexes, on="complex\_id", how="left")

    else:

        venue\_info = pd.DataFrame()

    if not df\_competition.empty and not df\_category.empty:

        comp\_info = pd.merge(df\_competition, df\_category, on="category\_id", how="left")

    else:

        comp\_info = pd.DataFrame()

else: # Handle case where initial connection failed

    st.error("Failed to connect to the database. Cannot load data.")

    # Assign empty dataframes to prevent errors later

    df\_competitors = df\_rankings = df\_complexes = df\_venues = df\_category = df\_competition = pd.DataFrame()

    df\_merged = venue\_info = comp\_info = pd.DataFrame()

# --- SIDEBAR FILTERS ---

st.sidebar.header("🔍 Filters")

# Use st.form to group inputs and submit button

with st.sidebar.form("filter\_form"):

    # Get unique values, handle potential errors if df is empty

    country\_list = sorted(df\_competitors['country'].dropna().unique()) if not df\_competitors.empty else []

    category\_list = sorted(df\_category['category\_name'].dropna().unique()) if not df\_category.empty else []

    # Set default index based on previous submission if available

    default\_country\_index = 0

    if st.session\_state.filters\_applied and st.session\_state.submitted\_country in country\_list:

       try:

           default\_country\_index = country\_list.index(st.session\_state.submitted\_country)

       except ValueError:

           default\_country\_index = 0 # Fallback if value not found

    default\_category\_index = 0

    if st.session\_state.filters\_applied and st.session\_state.submitted\_category in category\_list:

        try:

            default\_category\_index = category\_list.index(st.session\_state.submitted\_category)

        except ValueError:

            default\_category\_index = 0 # Fallback

    # Selectboxes inside the form

    selected\_country = st.selectbox(

        "Select Country",

        options=country\_list,

        index=default\_country\_index,

        disabled=not country\_list # Disable if list is empty

        )

    selected\_category = st.selectbox(

        "Select Category",

        options=category\_list,

        index=default\_category\_index,

        disabled=not category\_list # Disable if list is empty

        )

    # Submit button for the form

    submitted = st.form\_submit\_button("Apply Filters")

    if submitted:

        # Store the selected values in session state when form is submitted

        st.session\_state.filters\_applied = True

        st.session\_state.submitted\_country = selected\_country

        st.session\_state.submitted\_category = selected\_category

        st.sidebar.success("Filters Applied!") # User feedback

# Cancel/Clear button OUTSIDE the form

clear\_filters = st.sidebar.button("Clear Filters / Show All")

if clear\_filters:

    st.session\_state.filters\_applied = False

    st.session\_state.submitted\_country = None

    st.session\_state.submitted\_category = None

    st.sidebar.info("Filters Cleared. Showing all data.")

    # st.experimental\_rerun() # Optional: Force rerun to immediately reset selectbox index

# --- MAIN LAYOUT ---

st.title("🎾Tennis Game Dashboard")

# Determine current filter status for display

display\_country = st.session\_state.submitted\_country if st.session\_state.filters\_applied else "All"

display\_category = st.session\_state.submitted\_category if st.session\_state.filters\_applied else "All"

st.info(f"Displaying data for Country: \*\*{display\_country}\*\*, Category: \*\*{display\_category}\*\*")

# --- Apply Filters based on Session State ---

# Only filter if 'filters\_applied' is True and the corresponding state variable has a value

filtered\_df = df\_merged.copy() # Start with a copy of the full data

if st.session\_state.filters\_applied and st.session\_state.submitted\_country and not filtered\_df.empty:

    filtered\_df = filtered\_df[filtered\_df['country'] == st.session\_state.submitted\_country]

filtered\_comp = comp\_info.copy() # Start with a copy

if st.session\_state.filters\_applied and st.session\_state.submitted\_category and not filtered\_comp.empty:

    filtered\_comp = filtered\_comp[filtered\_comp['category\_name'] == st.session\_state.submitted\_category]

# --- Display Data ---

col1, col2 = st.columns(2)

with col1:

    st.subheader(f"📊 Competitor Rankings")

    st.dataframe(filtered\_df) # Display the potentially filtered df

with col2:

    st.subheader("🏆 Top 10 Players (Overall)")

    # Usually Top 10 is based on the overall data, not the filtered one, unless specified otherwise

    if not df\_merged.empty:

        top10 = df\_merged.sort\_values("rank").head(10)

        st.dataframe(top10)

    else:

        st.write("No ranking data available.")

st.markdown("---")

st.header("📍 Venues & Complexes")

st.dataframe(venue\_info) # Venue info usually isn't filtered by player country/category

st.markdown("---")

st.header(f"📅 Competitions")

st.dataframe(filtered\_comp) # Display the potentially filtered competition info

# --- VISUALIZATION ---

st.markdown("---")

st.header("📈 Analysis")

# Histogram - Show overall distribution unless filters are applied

if not df\_merged.empty:

    # Decide if histogram should be filtered or always show all countries

    # Option 1: Always show all countries

    hist\_data = df\_merged

    hist\_title = "Number of Players per Country (Overall)"

    # # Option 2: Show only filtered country in histogram (might not be very useful for histogram)

    # if st.session\_state.filters\_applied and st.session\_state.submitted\_country:

    #     hist\_data = filtered\_df

    #     hist\_title = f"Players in {st.session\_state.submitted\_country}"

    # else:

    #     hist\_data = df\_merged

    #     hist\_title = "Number of Players per Country (Overall)"

    fig = px.histogram(hist\_data, x="country", title=hist\_title, color\_discrete\_sequence=['#EF553B'])

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

else:

    st.write("No data for player distribution plot.")

# Bar Chart - Show Overall Top 10

if 'top10' in locals() and not top10.empty: # Check if top10 exists and is not empty

    fig2 = px.bar(top10, x='name', y='points', color='points', title='Top 10 Players by Points (Overall)')

    st.plotly\_chart(fig2, use\_container\_width=True)

else:

    st.write("No data for top 10 players plot.")

# Add a placeholder if connection failed initially

if not conn:

     st.warning("Application running with limited functionality due to database connection failure.")

     # --- FOOTER ---

st.markdown("---")

st.markdown("Done by [DEVAKUMAR SUGUMARAN]")