A database in MySQL to track the data associated with the current NCAA basketball season is created. Data for this project can be accessed from a service provider using API calls but however, the API was locked out after a few iterations of the project, hence I copied the data into .json files attached.

The database is designed to track game and season performance for each of the included teams. However, you should also be able to display a data model to include player information for the season as well. The database should be designed to hold all of the data, and to include instructions for updates as new data is available.

1. A data model containing all of the required fields.

A picture containing diagram

Description automatically generated

1. SQL commands to create the tables in the data model, including all primary and foreign keys.

Used Python to create tables including primary and foreign keys. Please refer to DataWrangling.py.

1. SQL commands to insert data from *all of the data for the games, teams, stadiums, and players* into the tables.

import numpy as np

import pandas as pd

import sqlalchemy

from sqlalchemy import create\_engine

# Set up Python to enable MySQL access to the 'root' account

engine = create\_engine('mysql+mysqlconnector://root:xxxxx@127.0.0.1:3306/mydb', echo=False)

# Stadium Data:

print("Importing Stadium Data ...")

stadium = pd.read\_json('Stadium.json')

# # Select six specific fields for the teams table to be added to MySQL

stadium2 = stadium[['StadiumID', 'Active', 'Name', 'Address', 'City', 'State', 'Zip', 'Country', 'Capacity', 'GeoLat', 'GeoLong']]

# Players Data being transferred to SQL

stadium2.to\_sql(name='stadium', con=engine, if\_exists = 'append', index=False)

# Importing Teams Data:

print("Importing Teams Data ...")

# Reading data from JSON file.

teams = pd.read\_json('Teams.json')

# # Delete the Active columm from teams before converting the stadium dict

del teams['Active']

# # Rename the Name column to Mascot

teams.rename(columns={'Name':'Mascot'}, inplace=True)

# # Convert the dict information in the Stadium field to individual fields

teams = teams.join(pd.DataFrame(teams["Stadium"].to\_dict()).T)

# # Observing what data is there, to import later to SQL

print(teams.columns)

# Dropping rows having fields empty within the StadiumID column

teams = teams.dropna(subset=["StadiumID"])

# # Select six specific fields for the teams table to be added to MySQL

teams2 = teams[['TeamID','School','Mascot','Wins','Losses','Conference','ConferenceWins','ConferenceLosses','ApRank','StadiumID']]

# # Teams Data being transferred to SQL

teams2.to\_sql(name='teams', con=engine, if\_exists = 'append', index=False)

# Player data:

print("Importing Player Data ...")

# Reading data from JSON file

players = pd.read\_json('PlayerDetailsbyActive.json')

# Creating copy of players data

players\_copy = players.copy()

# Removing data that do not match original set of TeamIDs imported from Teams Data

for index, row in players\_copy.iterrows():

    if row['TeamID'] not in list(teams['TeamID']):

       players.drop(index, inplace = True)

# # Selecting specific fields for the teams table to be added to MySQL

players2 = players[['PlayerID', 'FirstName', 'LastName', 'TeamID', 'Team', 'Jersey', 'Position', 'Class', 'Height', 'Weight']]

# Players Data being transferred to SQL

players2.to\_sql(name='player', con=engine, if\_exists = 'append', index=False)

# Importing Game Data:

print("Importing Games Data ...")

# Reading Data from JSON file.

games = pd.read\_json('GameData.json')

# # Converting DateTime column to datetime format:

pd.to\_datetime(games['DateTime'])

# #Convert the dict information in the Stadium field to individual fields

games = games.join(pd.DataFrame(games["Stadium"].to\_dict()).T)

# Deleting duplicates

games = games.drop\_duplicates(subset=["GameID"], keep=False)

# # Selecting specific fields for the teams table to be added to MySQL

games2 = games[['GameID', 'Season', 'SeasonType', 'Status', 'DateTime', 'AwayTeam', 'HomeTeam', 'AwayTeamID', 'HomeTeamID', 'AwayTeamScore', 'HomeTeamScore', 'PointSpread', 'OverUnder', 'AwayPointSpreadPayout', 'HomePointSpreadPayout', 'OverPayout', 'UnderPayout', 'StadiumID', 'AwayTeamMoneyLine', 'HomeTeamMoneyLine']]

# Converting periods to dataframe

periods = pd.DataFrame(columns=['PeriodID', 'GameID', 'Number', 'Name', 'Type', 'AwayScore', 'HomeScore'])

for i in games['Periods']:

    for j in range(len(i)):

        periods = periods.append(i[j], ignore\_index = True)

# # Selecting specific fields for the teams table to be added to MySQL

periods2 = periods[['PeriodID', 'GameID', 'Number', 'Name', 'Type', 'AwayScore', 'HomeScore']]

# Games and Periods Data being transferred to SQL

games2.to\_sql(name='games', con=engine, if\_exists = 'append', index=False)

periods2.to\_sql(name='periods', con=engine, if\_exists = 'append', index=False)

# Player and TeamSeasonStats.json

print("Importing Player Season Stats and Team Season Stats Data ...")

#Reading Player Season Stats from JSON File

pstats = pd.read\_json('PlayerSeasonStats.json')

# Only taking rows that match with TeamIDs and PlayerIDs imported earlier

pstats = pstats[pstats.TeamID.isin(teams.TeamID) & pstats.PlayerID.isin(players.PlayerID)]

# # Selecting  specific fields for the teams table to be added to MySQL

pstats2 = pstats[['StatID', 'TeamID', 'PlayerID', 'SeasonType', 'Season', 'Team', 'Games', 'Minutes', 'FieldGoalsMade', 'FieldGoalsAttempted', 'FieldGoalsPercentage', 'TwoPointersMade', 'TwoPointersAttempted', 'TwoPointersPercentage', 'ThreePointersMade', 'ThreePointersAttempted', 'ThreePointersPercentage', 'FreeThrowsMade', 'FreeThrowsAttempted', 'FreeThrowsPercentage', 'OffensiveRebounds', 'DefensiveRebounds', 'Rebounds', 'Assists', 'Steals', 'BlockedShots','Turnovers', 'PersonalFouls', 'Points']]

# Player Season Stats being transferred to SQL

pstats2.to\_sql(name='seasonstats', con=engine, if\_exists = 'append', index=False)

#Reading Team Season Stats from JSON File

tstats = pd.read\_json('TeamSeasonStats.json')

# Only taking rows that match with TeamIDs imported earlier

tstats = tstats[tstats.TeamID.isin(teams.TeamID)]

# # Selecting  specific fields for the teams table to be added to MySQL

tstats2 = tstats[['StatID', 'TeamID', 'SeasonType', 'Season', 'Team', 'Wins','Losses', 'ConferenceWins', 'ConferenceLosses', 'Games','Minutes', 'FieldGoalsMade', 'FieldGoalsAttempted','FieldGoalsPercentage', 'TwoPointersMade', 'TwoPointersAttempted','TwoPointersPercentage', 'ThreePointersMade', 'ThreePointersAttempted','ThreePointersPercentage', 'FreeThrowsMade', 'FreeThrowsAttempted','FreeThrowsPercentage', 'OffensiveRebounds', 'DefensiveRebounds','Rebounds', 'Assists', 'Steals', 'BlockedShots', 'Turnovers','PersonalFouls', 'Points']]

# Team Season Stats Data being transferred to SQL

tstats2.to\_sql(name='seasonstats', con=engine, if\_exists = 'append', index=False)

1. SQL commands to answer the following questions (but add at least 5 more queries to answer relevant and interesting questions about the data):
   1. What team had the highest score in a single game this season?

SELECT max(Points), Team

FROM seasonstats

GROUP BY Team

ORDER BY max(Points) DESC;

* 1. What teams (and mascots) are in the Atlantic 10, SEC, and Big 10 conferences?

SELECT School, Mascot, Conference

FROM teams

WHERE Conference = 'Atlantic 10' OR Conference = 'SEC' OR Conference = 'Big 10';

* 1. What are the 10 highest scoring players (points per game) so far this season?

SELECT ss.PlayerID, FirstName, LastName, max(Points)

FROM seasonstats ss, player

WHERE ss.PlayerID = player.PlayerID

GROUP BY ss.PlayerID

ORDER BY max(points) desc;

* 1. How many games have been played so far this season?

SELECT sum(Games)

FROM seasonstats

WHERE Season = 2023 and SeasonType = 1;

* 1. What games were played in the last 3 days, and what were the results?

SELECT AwayTeam, AwayTeamScore, HomeTeam, HomeTeamScore, DateTime

FROM games

WHERE DateTime > '2023-03-07 22:00:00';

* 1. What was the score of the game held in the smallest stadium this season?

SELECT min(stadium.Capacity), stadium.Name, AwayTeamScore, HomeTeamScore

FROM stadium, games

WHERE games.StadiumID = stadium.StadiumID;

* 1. Which teams have scored more than 60 points and had more than 15 offensive rebounds in a single game?

SELECT Team, Points, OffensiveRebounds

FROM seasonstats

WHERE Points > 60 AND OffensiveRebounds > 15;

* 1. Extra 5 queries:
     1. Best player of all time

SELECT player.PlayerID, FirstName, LastName, Wins, Losses, seasonstats.Team, max(Points)

FROM seasonstats, player

WHERE seasonstats.PlayerID = player.PlayerID;

* + 1. Best Team of all time

SELECT teams.TeamID, teams.School, teams.Mascot, seasonstats.Wins, seasonstats.Losses, max(Points)

FROM seasonstats, teams

WHERE seasonstats.TeamID = teams.TeamID;

* + 1. Teams that have more losses than wins:

SELECT seasonstats.TeamID, teams.School, teams.Mascot, seasonstats.Wins, seasonstats.Losses  
FROM seasonstats, teams  
WHERE seasonstats.TeamID = teams.TeamID  
AND seasonstats.Losses > seasonstats.Wins;

* + 1. Players in the 'F' position for Florida State School team

SELECT player.PlayerID, player.FirstName, player.LastName, player.TeamID, teams.School, teams.Mascot, player.Position

FROM player, teams

WHERE player.TeamID = teams.TeamID

AND School = "Florida State"

HAVING player.Position = 'F';

* + 1. Players with less than 15 points

SELECT seasonstats.PlayerID, player.FirstName, player.LastName, seasonstats.Team, seasonstats.Points

FROM player, seasonstats

WHERE player.PlayerID = seasonstats.PlayerID

HAVING seasonstats.Points < 15;

1. Instructions for the ETL process, including commands, order of operations, frequency, etc.

For the ETL process, in case of new data arrival and in the JSON Format:

1. Please ensure the data file names are correct – GameData.json, PlayerDetailsbyActive.json, PlayerSeasonStats.json, Schedule.json, Stadium.json, Teams.json, TeamSeasonStats.json.
2. Once the data files are named correctly and are of the right format, just run the DataWrangling.py file.
3. However, the python program may not work properly if any of the names of existing columns are changed.

Order of operations:

The python program ingests the data, from locally stored JSON files, parses through them, transforms a couple of rows into columns of dataframe, cleans the data of missing values, selects a couple of columns that are to be transferred as per the defined data model and then then transfers the data to SQL to the respective tables for further analysis.

Frequency:

The ETL process can be done seasonally – an incremental load is more than enough as the python program is designed to append the data it’s given.

--------------------------------------------------------------------------------------------------------------------------------