Project to convert SQL to NoSQL

Database: ICC Cricket World Cup 2019

ABSTRACT:

In this assignment, we are converting a SQL relational database into a non-relational NoSQL database. We have used MongoDB to convert from relational to a non-relational database. This dataset contains all the tables related to the ICC Cricket world cup. Some tables have been previously scraped using Beautiful soup in Python. Additional tables have been web scraped which contain various tweets corresponding to the trending hashtags during the world cup.

Datasets:

We had 3 tables that were normalized in the previous assignment. These tables were

1. Players table
2. Boards table
3. Teams table

There are a total of 152 players in the player's table, 10 Cricket boards and their respective teams. We scrapped additional tables that were related to twitter hashtags that were trending during the world cup. Some of those hashtags were #kohli, #cwc19, etc. to name a few.

MongoDB

1. What is MongoDB?

MongoDB is a document database, which means it stores data in JSON – like documents. Unlike a relational SQL database where data is stored in tables, a non-relational NOSQL database stores data in the form of collections. MongoDB stores documents in collections within databases.

In a SQL database where we have structured database schema, we can use join function which can join two tables. In NoSQL database having unstructured schema, we can write object in object.

2. Libraries used to convert a SQL Database into NOSQL database

We have used pymongo library to convert tables from SQL to NOSQL. Pymongo is Python distribution containing tools for working with MongoDB.

When we login into Pymongo, localhost: 27017 is the default port for connecting to MongoDB server.

Code to Web Scrape from Twitter

Twitter needs the user who is web scrapping to have valid permissions from twitter. These permissions are granted in the form of authorization codes.

We have made use of the Tweepy library available for python to web scrape.

A glimpse of the code is given below:

```
#Using authorisation keys provided by twitter and assigning to twitter_api variable
CONSUMER_KEY = 'auTIVP4hHiJA7Snb4XYWWEofR'
CONSUMER_SECRET = 'VeoKqZ1twMgYaxTSNB6MRO6dvg112BYuTajEZJMw8gzutjpDKR'
ACCESS_TOKEN = '1121864797-GyTUHsApFDjZGoP4ovCDXudSq48wXUXGiBwVczo'
```

```
ACCESS_TOKEN_SECRET = 'I5FK5vwn70xskQTgHema1VzAZLoHZIR1NDlTXHfvuTSVS'

auth = tweepy.OAuthHandler(CONSUMER_KEY, CONSUMER_SECRET)
auth.set_access_token(ACCESS_TOKEN, ACCESS_TOKEN_SECRET)

api = tweepy.API(auth)

#Create a new csv file
csv_file = open('benstokes.csv', 'w+')

#Write rows in those csv files
csv_writer = csv.writer(csv_file)
csv_writer.writerow(['Timestamp', 'Tweet', 'username', 'hashtags' ,"followers_count"])

#Search for the mentioned hashtag
for tweet in tweepy.Cursor(api.search, q="#benstokes",
                           lang="en",
                           since="2019-01-01").items(100):
    csv_writer.writerow([tweet.created_at, tweet.text.encode('utf-8'), tweet.user.screen_name.encode('utf-8'),
                         [e['text'] for e in tweet._json['entities']
                          ['hashtags']] ,tweet.user.followers_count]
```

In this code, after all the authorizations are done, code creates a new .csv file where scraped data is stored. It then updates the column headers and cursor functions gathers information related to these hashtags from Twitter.

We have also used Twitter API to web scrape twitter details of some players and created a table with those details

Code for that is:

```
twitter_api = twitter.Twitter(auth=auth)
print(twitter_api)

Top_ten_Test_Batsmen = ["@imVkohli", "@stevesmith49", "@marnus3cricket", "@NotNossy",
"@davidwarner31",
                        "@cheteshwar1", "@babarazam258", "@ajinkyarahane88", "@root66", "@benstokes38"]

player_name = []
twitter_id = []
followers = []
friends = []
verify = []

#Using for loop to iterate and print all values
for i in Top_ten_Test_Batsmen:
    twitter_id.append(i)
    player_name.append(twitter_api.statuses.user_timeline(screen_name=i)[1]['user']['name'])
    followers.append(twitter_api.statuses.user_timeline(screen_name=i)[1]['user']['followers_count'])
    friends.append(twitter_api.statuses.user_timeline(screen_name=i)[1]['user']['friends_count'])
    verify.append(twitter_api.statuses.user_timeline(screen_name=i)[1]['user']['verified'])

Twitter_Specifications = pd.DataFrame({
        'Player Name': player_name,
        'Twitter ID': twitter_id,
```

```
        'Followers Count': followers,
        'Friends Count': friends,
        'Verified': verify})
```

Script to convert Structured database into MongoDB

This script reads the data in python and writes it in MongoDB using pymongo library. Nested for loops are used to create sub documents under a document from the SQL tables. Iterrows function has been used to create object for every row of the tables with matching IDs. First it iterates on the players dataframe, next iterates on the teams dataframe, then iterates on boards dataframe and lastly combines all the respective data into one object/document and creates sub-objects/sub-documents for MongoDB. As MongoDB can contain sub documents under the documents, the following script has been written to convert from SQL schema to NoSQL schema.

```
for index,row in df_players.iterrows():
    t_id = row['Team_Id']
    team = ""
    team_object = {}
    board_object = {}
    for index1,row1 in df_teams.iterrows():
        if row1['Team_ID'] == t_id:

            # team
            team = row1
            team_object = {
            "t_id": team.Team_ID,
            "teamname" : team.Team_name
            }

            # board
            b_id = team.Board_Id
            board = ""
            for index2,row3 in df_boards.iterrows():
                if row3['Board_Id'] == b_id:
                    board = row3

            board_object = {
                "Board Id": board.Board_Id,
                "Board Name": board['Board Name'],
                "Coach Name": board['Coach Name']
            }

    ICC = {
        "PlayerID" : row['Player_Id'],
        "Player" : row['Player_Name'],

        "team" : team_object,
        "cricket_board": board_object
    }
    mycol.insert_one(ICC)
```

Import .csv files in MongoDB

First we read all csv files and with the help of below code, we import in MongoDB

```
df_kohli_tweets = pd.read_csv("kohli.csv")
df_benstokes_tweets = pd.read_csv("benstokes.csv")
df_cwc19_tweets = pd.read_csv("cwc19.csv")
df_cwc19final_tweets = pd.read_csv("cwc19final.csv")
df_davidwarner_tweets = pd.read_csv("davidwarner.csv")
df_rohitsharma_tweets = pd.read_csv("rohitsharma.csv")
df_teamindia_tweets = pd.read_csv("teamindia.csv")
df_teamaustralia_tweets = pd.read_csv("teamaustralia.csv")
df_worldcup_tweets = pd.read_csv("worldcup.csv")
df_most_runs = pd.read_csv("most_runs.csv", encoding = "ISO-8859-1")
df_most_wickets = pd.read_csv("most_wickets.csv", encoding = "ISO-8859-1")

#converting csv files to a data frame and inserting those data frame into mongodb
records1 = df_cwc19_tweets.to_dict(orient = 'records')
result = mydb.cwc19_tweets.insert_many(records1 )
records2 = df_cwc19final_tweets.to_dict(orient = 'records')
result = mydb.cwc29final_tweets.insert_many(records2 )
records3 = df_worldcup_tweets.to_dict(orient = 'records')
result = mydb.worldcup_tweets.insert_many(records3 )
records4 = df_teamindia_tweets.to_dict(orient = 'records')
result = mydb.teamindia_tweets.insert_many(records4 )
records5 = df_teamaustralia_tweets.to_dict(orient = 'records')
result = mydb.teamaustralia_tweets.insert_many(records5 )
records6 = df_kohli_tweets.to_dict(orient = 'records')
result = mydb.kohli_tweets.insert_many(records6 )
records7 = df_benstokes_tweets.to_dict(orient = 'records')
result = mydb.benstokes_tweets.insert_many(records7 )
records8 = df_rohitsharma_tweets.to_dict(orient = 'records')
result = mydb.rohitsharma_tweets.insert_many(records8 )
records9 = df_davidwarner_tweets.to_dict(orient = 'records')
result = mydb.davidwarner_tweets.insert_many(records9 )
records10 = Twitter_Specifications.to_dict(orient = 'records')
result = mydb.Twitter_Specifications.insert_many(records10)
records11 = df_most_runs.to_dict(orient = 'records')
result = mydb.most_runs.insert_many(records11)
records12 = df_most_wickets.to_dict(orient = 'records')
result = mydb.most_wickets.insert_many(records12)
records13 = df_teams.to_dict(orient = 'records')
result = mydb.teams_dataset.insert_many(records13)
```

Functions used in MongoDB

1. Aggregate

Aggregation operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.

How have we used Aggregate in our database?

⇒ $match takes a document that specifies the query conditions. The query syntax is identical to the read operation query syntax

⇒ $project Passes along the documents with the requested fields to the next stage in the pipeline. The specified fields can be existing fields from the input documents or newly computed fields.

⇒ $lookup function
$lookup function in MongoDB allows us to join documents on collections that reside in the same database. The $lookup function will return the documents as if they came from this 'joined' collection, in the form of a sub-array of the target original collection.

⇒ .Pretty()
Configures the cursor to display results in an easy-to-read format.