Emulation Based system for distributed file storage and parallel computation - Final Report

Team members:
1. Lahari Palem
2. Rohan Chaudhari

Team Number:
30 (Google Sheets)
29 (USC Den)

Links:
Code: Project CodeBase
Video: Youtube Video Link
Dataset: Match & Ball Datasets

The particular link listed above is a cricket dataset that consists of 2 CSV files:
1. Ball by Ball data
2. Complete Match data

Here, we plan on using match_id from both the datasets to join them and carry out various analytical operations.

Technologies:
1. Python
2. Jupyter Notebook
3. Flask & REST APIs
4. MySQL

Project Progress:

Task 1:
Implemented all the EDFS file system commands using Flask and REST API calls.

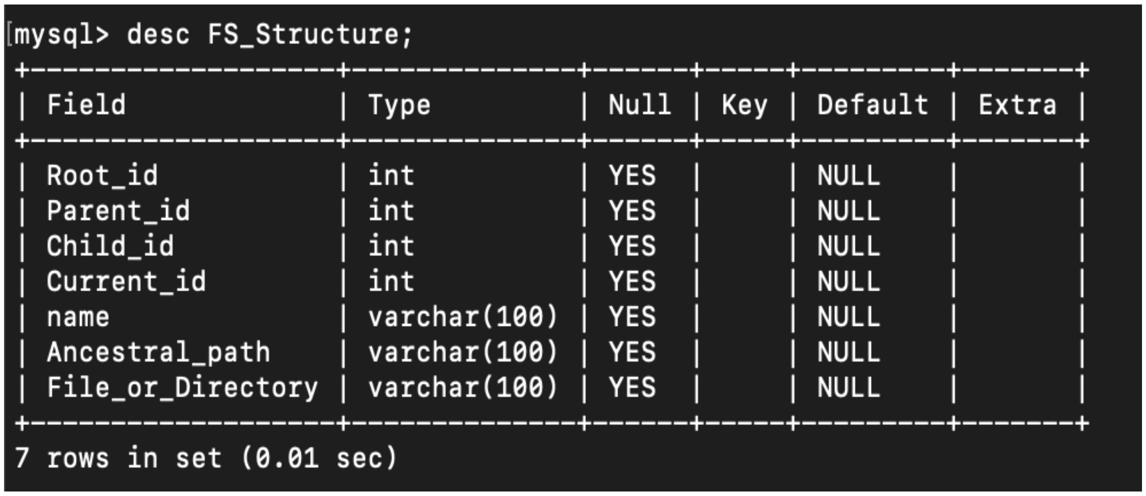
a. **mkdir** - Here, we maintain a table called FS_Structure where we store the file system structure such as the file/directory name, ancestral path, id for the file/directory, child id's for directories etc.

   In the MYSQL implementation of EDFS to add a directory at multiple hierarchy levels we have considered the following cases for the EDFS development.
   Parent directory, current directory and child directory are given ID as primary key to maintain the EDFS.

   All the directories to be added at root level, it should be as one level hierarchy (ex:; /user) where parent id will be updated, current directory id would be the highest id from the EDFS table plus one and child id will be updated when some other directory with already present as parent id is inserted.

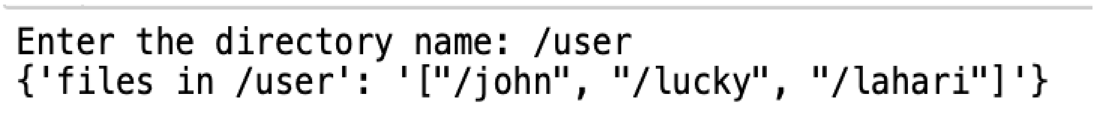
There is a flag identifier which is used to make the difference between an already existing child of a parent directory and to update the newly inserted tuple of parent id with multiple children.

There is also a flag which can identify the type of folders at given path,like if they are “Directory” or “File”



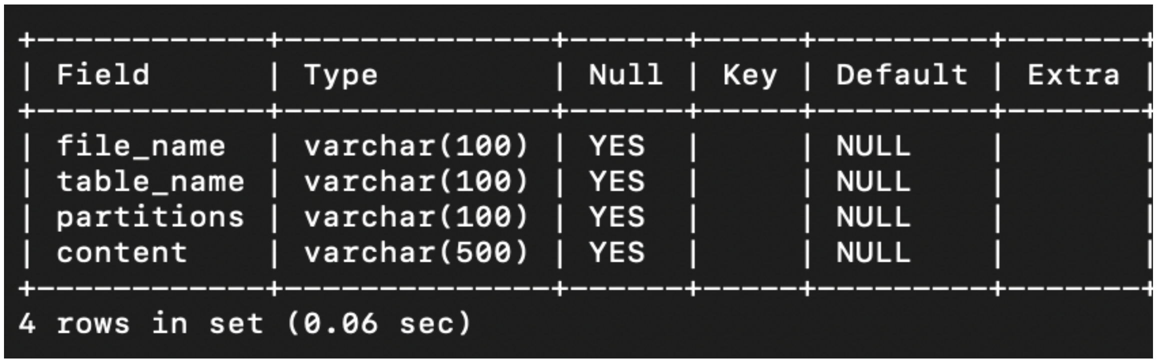
```
[mysql> desc FS_Structure;
+-------------------+--------------+------+-----+---------+-------+
| Field             | Type         | Null | Key | Default | Extra |
+-------------------+--------------+------+-----+---------+-------+
| Root_id           | int          | YES  |     | NULL    |       |
| Parent_id         | int          | YES  |     | NULL    |       |
| Child_id          | int          | YES  |     | NULL    |       |
| Current_id        | int          | YES  |     | NULL    |       |
| name              | varchar(100) | YES  |     | NULL    |       |
| Ancestral_path    | varchar(100) | YES  |     | NULL    |       |
| File_or_Directory | varchar(100) | YES  |     | NULL    |       |
+-------------------+--------------+------+-----+---------+-------+
7 rows in set (0.01 sec)
```

b. **ls -** For the ls command we first run an sql command to find all the child id’s of the directory we are trying to list the files and directories. On obtaining the said child id’s we then find the file/directory names of the children and return them as a dictionary.



```
Enter the directory name: /user
{'files in /user': '["/john", "/lucky", "/lahari"]'}
```

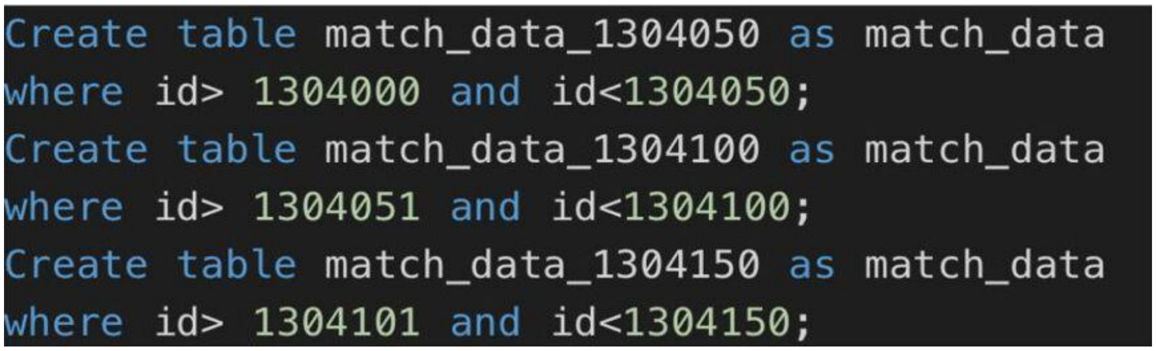
c. **cat -** The cat command collects all the data for that particular file from the table partition file table and displays the content for text files.



```
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| file_name  | varchar(100) | YES  |     | NULL    |       |
| table_name | varchar(100) | YES  |     | NULL    |       |
| partitions | varchar(100) | YES  |     | NULL    |       |
| content    | varchar(500) | YES  |     | NULL    |       |
+------------+--------------+------+-----+---------+-------+
4 rows in set (0.06 sec)
```

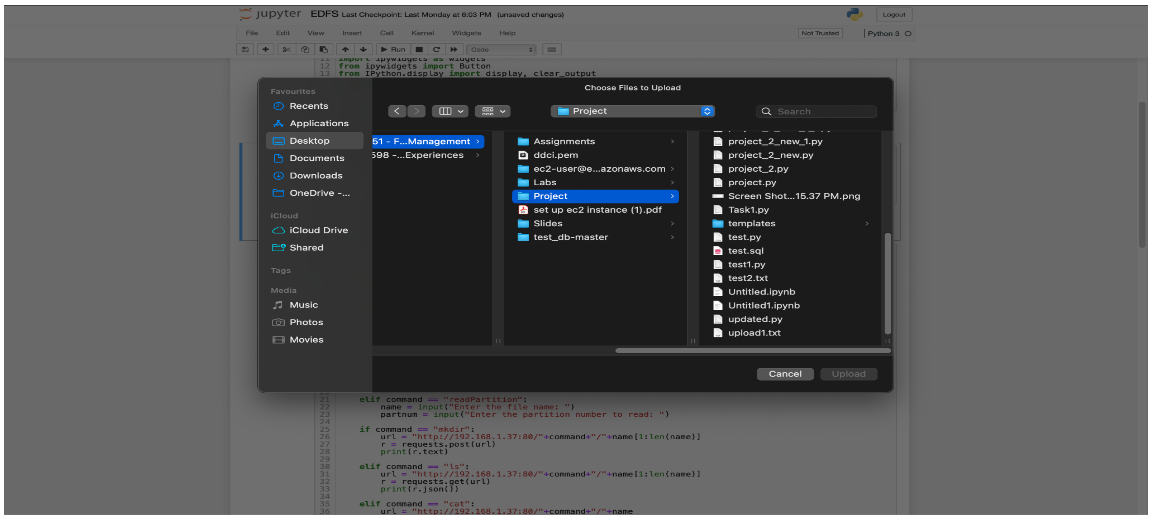
d. **rm** - On running the rm command we edit the FS_Structure table as shown in the screenshot above. Once we find the file to remove, we remove its entry from the FS_Structure table and additionally, also remove its id from the child id column of the directory it resided in. Additionally, we also have to remove the entry of the file from the partition_file table to maintain consistency.

e. **put** - The put command here is used to insert '.csv' and '.txt' files into the EDFS. All text files are stored in the parititon_file table where the csv files are first stored in the master data table and then based on the id of the row, we split the data into 'k' different tables, where 'k' is the number of partitions. Additionally, we also update the child id column in the FS_Structure table with for the directory into which the particular file is being updates and also update the partition_file table with what part of the data is stored in which partition with the partition id, such as all data with id's between 1304000 to 1304050 is stored in partition 1 and so on.



```
Create table match_data_1304050 as match_data
where id> 1304000 and id<1304050;
Create table match_data_1304100 as match_data
where id> 1304051 and id<1304100;
Create table match_data_1304150 as match_data
where id> 1304101 and id<1304150;
```

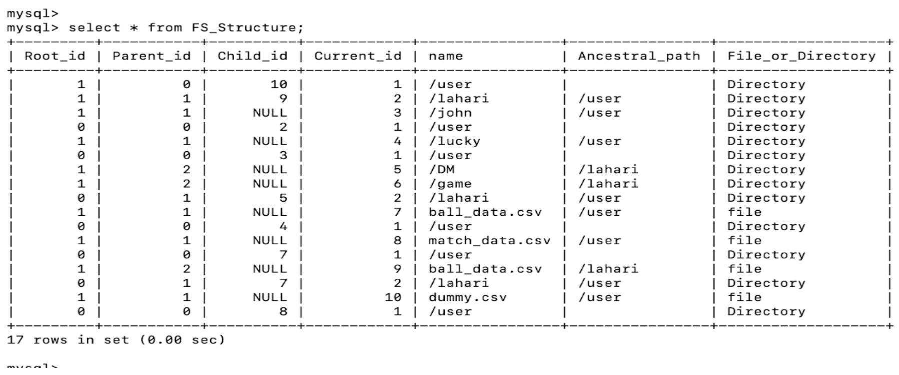
Additionally, we also have an UI for the put command that browses the local file system to select the file we want to upload to the file system as shown below.



```
import ipywidgets as widgets
from ipywidgets import Button
from IPython.display import display, clear_output
```

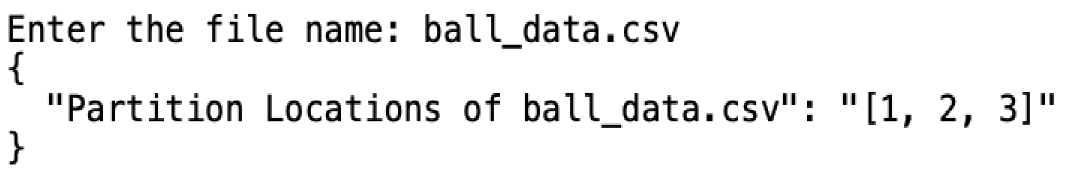
```
    elif command == "readPartition":
        name = input("Enter the file name: ")
        partnum = input("Enter the partition number to read: ")

    if command == "mkdir":
        url = "http://192.168.1.37:80/"+command+"/"+name[1:len(name)]
        r = requests.post(url)
        print(r.text)

    elif command == "ls":
        url = "http://192.168.1.37:80/"+command+"/"+name[1:len(name)]
        r = requests.get(url)
        print(r.json())

    elif command == "cat":
        url = "http://192.168.1.37:80/"+command+"/"+name
```

The file structure after placing the file at given Path



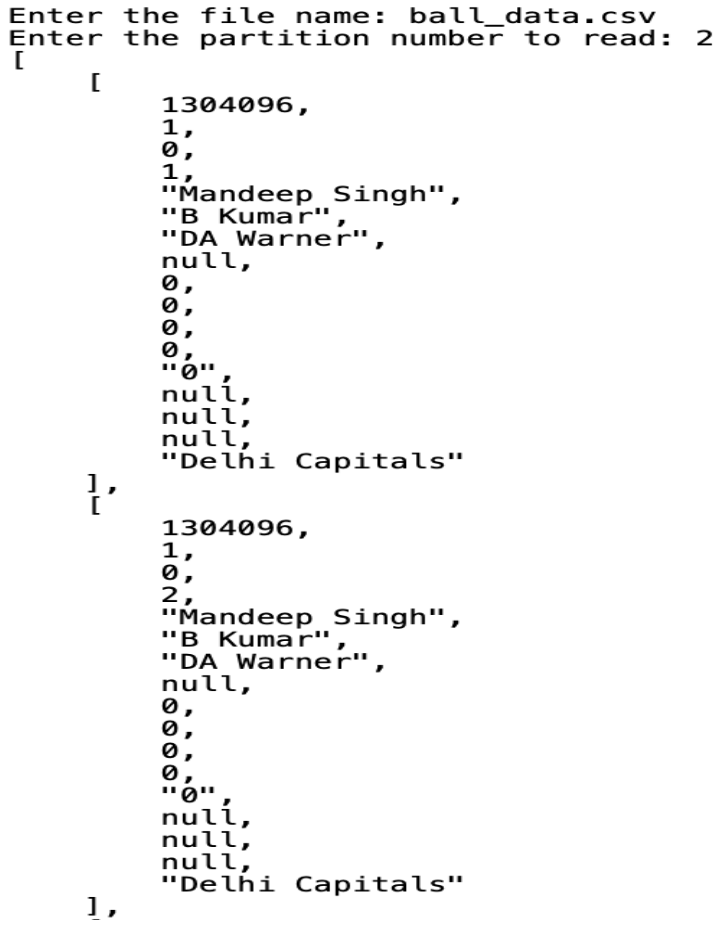
```
mysql>
mysql> select * from FS_Structure;
+---------+-----------+----------+------------+----------------+----------------+-------------------+
| Root_id | Parent_id | Child_id | Current_id | name           | Ancestral_path | File_or_Directory |
+---------+-----------+----------+------------+----------------+----------------+-------------------+
|       1 |         0 |       10 |          1 | /user          |                | Directory         |
|       1 |         1 |        9 |          2 | /lahari        | /user          | Directory         |
|       1 |         1 |     NULL |          3 | /john          | /user          | Directory         |
|       0 |         0 |        2 |          1 | /user          |                | Directory         |
|       1 |         1 |     NULL |          4 | /lucky         | /user          | Directory         |
|       0 |         0 |        3 |          1 | /user          |                | Directory         |
|       1 |         2 |     NULL |          5 | /DM            | /lahari        | Directory         |
|       1 |         2 |     NULL |          6 | /game          | /lahari        | Directory         |
|       0 |         1 |        5 |          2 | /lahari        | /user          | Directory         |
|       1 |         1 |     NULL |          7 | ball_data.csv  | /user          | file              |
|       0 |         0 |        4 |          1 | /user          |                | Directory         |
|       1 |         1 |     NULL |          8 | match_data.csv | /user          | file              |
|       0 |         0 |        7 |          1 | /user          |                | Directory         |
|       1 |         2 |     NULL |          9 | ball_data.csv  | /lahari        | file              |
|       0 |         1 |        7 |          2 | /lahari        | /user          | Directory         |
|       1 |         1 |     NULL |         10 | dummy.csv      | /user          | file              |
|       0 |         0 |        8 |          1 | /user          |                | Directory         |
+---------+-----------+----------+------------+----------------+----------------+-------------------+
17 rows in set (0.00 sec)

mysql>
```

f. **getPartitionLocation** - This function reads the partition_file table to find all the partition id's where the particular file is stored.



```
Enter the file name: ball_data.csv
{
  "Partition Locations of ball_data.csv": "[1, 2, 3]"
}
```

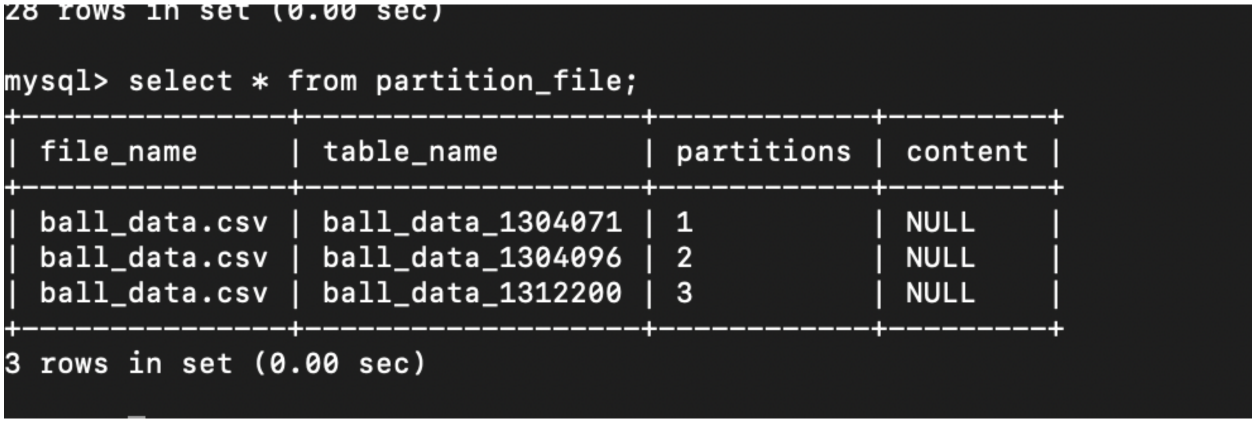
g. **readPartition** - This EDFS command is used to read only a part of the data for the given file. Taking the filename and partition number as input, it reads the partition_file table to find which partition id and correlates the partition id to the table which stores the data for partition 'x'. Ex: If we pass readPartition(match_data,2) it first reads the table which stored the 2nd partition of match_data and then retrieves all the data from that particular table. This function along with getPartitionLocation will be used to carry out the various analytical and search queries.



```
Enter the file name: ball_data.csv
Enter the partition number to read: 2
[
    [
        1304096,
        1,
        0,
        1,
        "Mandeep Singh",
        "B Kumar",
        "DA Warner",
        null,
        0,
        0,
        0,
        0,
        "0",
        null,
        null,
        null,
        "Delhi Capitals"
    ],
    [
        1304096,
        1,
        0,
        2,
        "Mandeep Singh",
        "B Kumar",
        "DA Warner",
        null,
        0,
        0,
        0,
        0,
        "0",
        null,
        null,
        null,
        "Delhi Capitals"
    ],
```

Task 2:
The data sets we selected are “match_data.csv” and “ball_data.csv”. We have stored both Match and Ball data in 3 partitions as 3 different SQL tables based on match ids.

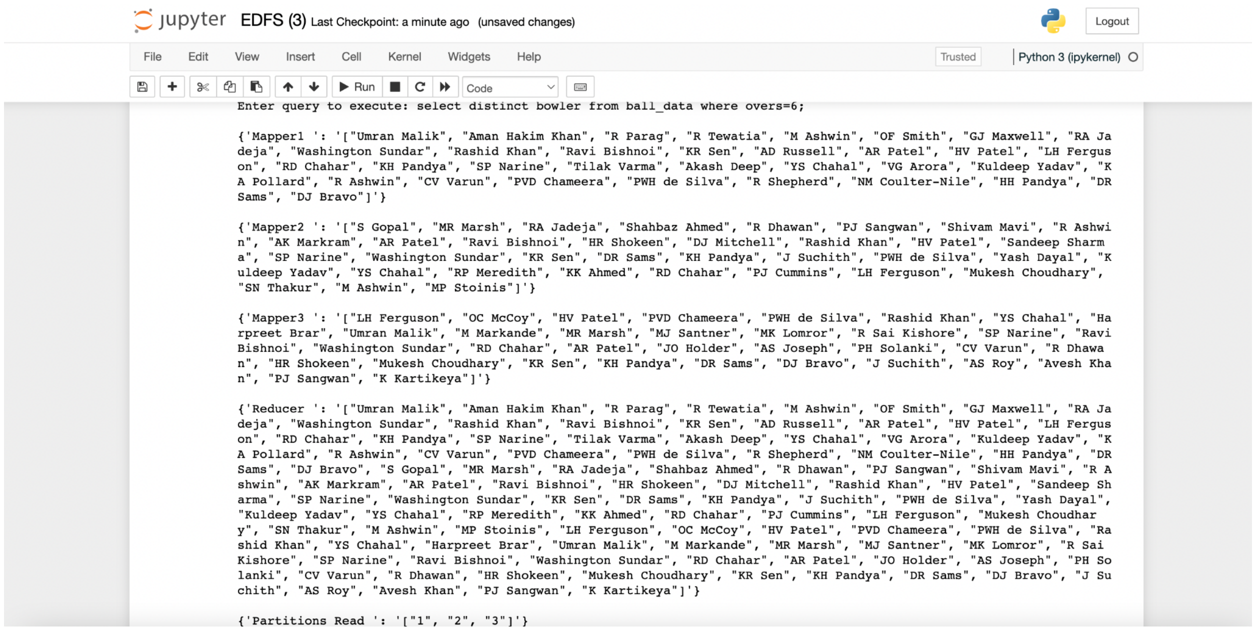
We store them in partition_file with file_name,table_name and number of partitions for the file.
The analytical and search query would perform search operation on the partitioned tables to perform map reduce function to fetch the result by querying the related partitioned tables



```
28 rows in set (0.00 sec)

mysql> select * from partition_file;
+---------------+-------------------+------------+---------+
| file_name     | table_name        | partitions | content |
+---------------+-------------------+------------+---------+
| ball_data.csv | ball_data_1304071 | 1          | NULL    |
| ball_data.csv | ball_data_1304096 | 2          | NULL    |
| ball_data.csv | ball_data_1312200 | 3          | NULL    |
+---------------+-------------------+------------+---------+
3 rows in set (0.00 sec)
```

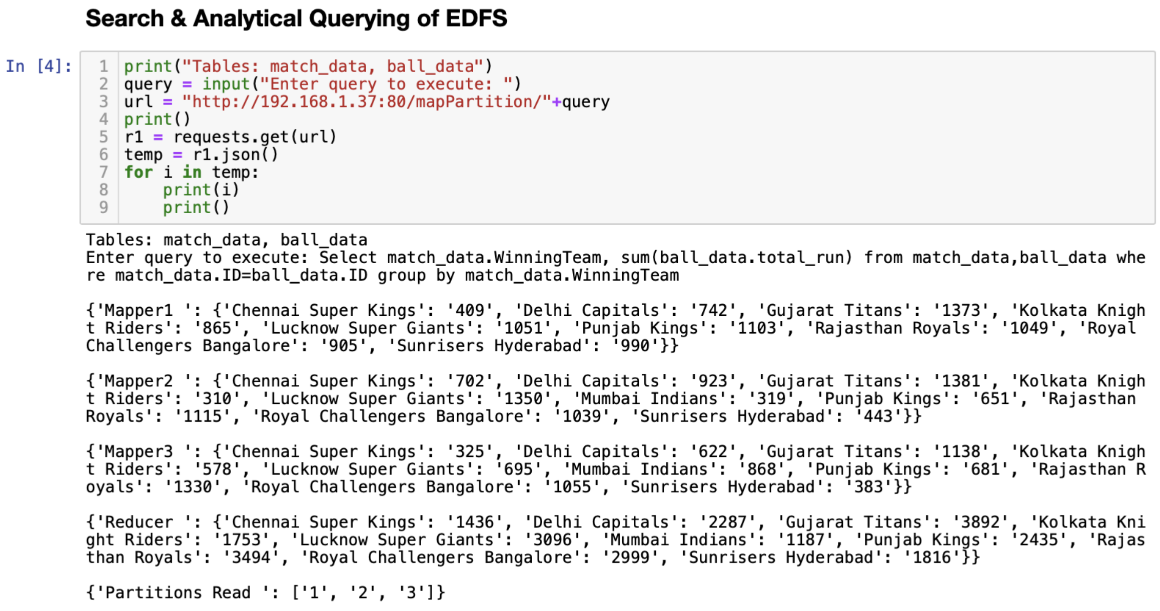
SEARCH AND ANALYTICAL QUERY:
Initially, the difference between Search and Analytical query is implemented based on the analytical functions present in the query passed.
If the query contains analytical functions like min, max, avg, sum and so on, the query is considered as analytical with a reduce function to aggregate mappers else the query is considered as a search and reduce function is used to combine mappers to the final result.

Search query:
For Search query, execution which maps the result from multiple partitions is passed as input from jupyter notebook and executed on all the partition tables to map the required result and then shuffled and reduced to return the reduced result from the search query.



```
Enter query to execute: select distinct bowler from ball_data where overs=6;

{'Mapper1 ': '["Umran Malik", "Aman Hakim Khan", "R Parag", "R Tewatia", "M Ashwin", "OF Smith", "GJ Maxwell", "RA Ja
deja", "Washington Sundar", "Rashid Khan", "Ravi Bishnoi", "KR Sen", "AD Russell", "AR Patel", "HV Patel", "LH Fergus
on", "RD Chahar", "KH Pandya", "SP Narine", "Tilak Varma", "Akash Deep", "YS Chahal", "VG Arora", "Kuldeep Yadav", "K
A Pollard", "R Ashwin", "CV Varun", "PVD Chameera", "PWH de Silva", "R Shepherd", "NM Coulter-Nile", "HH Pandya", "DR
Sams", "DJ Bravo"]'}

{'Mapper2 ': '["S Gopal", "MR Marsh", "RA Jadeja", "Shahbaz Ahmed", "R Dhawan", "PJ Sangwan", "Shivam Mavi", "R Ashwi
n", "AK Markram", "AR Patel", "Ravi Bishnoi", "HR Shokeen", "DJ Mitchell", "Rashid Khan", "HV Patel", "Sandeep Sharm
a", "SP Narine", "Washington Sundar", "KR Sen", "DR Sams", "KH Pandya", "J Suchith", "PWH de Silva", "Yash Dayal", "K
uldeep Yadav", "YS Chahal", "RP Meredith", "KK Ahmed", "RD Chahar", "PJ Cummins", "LH Ferguson", "Mukesh Choudhary",
"SN Thakur", "M Ashwin", "MP Stoinis"]'}

{'Mapper3 ': '["LH Ferguson", "OC McCoy", "HV Patel", "PVD Chameera", "PWH de Silva", "Rashid Khan", "YS Chahal", "Ha
rpreet Brar", "Umran Malik", "M Markande", "MR Marsh", "MJ Santner", "MK Lomror", "R Sai Kishore", "SP Narine", "Ravi
Bishnoi", "Washington Sundar", "RD Chahar", "AR Patel", "JO Holder", "AS Joseph", "PH Solanki", "CV Varun", "R Dhawa
n", "HR Shokeen", "Mukesh Choudhary", "KR Sen", "KH Pandya", "DR Sams", "DJ Bravo", "J Suchith", "AS Roy", "Avesh Kha
n", "PJ Sangwan", "K Kartikeya"]'}

{'Reducer ': '["Umran Malik", "Aman Hakim Khan", "R Parag", "R Tewatia", "M Ashwin", "OF Smith", "GJ Maxwell", "RA Ja
deja", "Washington Sundar", "Rashid Khan", "Ravi Bishnoi", "KR Sen", "AD Russell", "AR Patel", "HV Patel", "LH Fergus
on", "RD Chahar", "KH Pandya", "SP Narine", "Tilak Varma", "Akash Deep", "YS Chahal", "VG Arora", "Kuldeep Yadav", "K
A Pollard", "R Ashwin", "CV Varun", "PVD Chameera", "PWH de Silva", "R Shepherd", "NM Coulter-Nile", "HH Pandya", "DR
Sams", "DJ Bravo", "S Gopal", "MR Marsh", "RA Jadeja", "Shahbaz Ahmed", "R Dhawan", "PJ Sangwan", "Shivam Mavi", "R A
shwin", "AK Markram", "AR Patel", "Ravi Bishnoi", "HR Shokeen", "DJ Mitchell", "Rashid Khan", "HV Patel", "Sandeep Sh
arma", "SP Narine", "Washington Sundar", "KR Sen", "DR Sams", "KH Pandya", "J Suchith", "PWH de Silva", "Yash Dayal",
"Kuldeep Yadav", "YS Chahal", "RP Meredith", "KK Ahmed", "RD Chahar", "PJ Cummins", "LH Ferguson", "Mukesh Choudhar
y", "SN Thakur", "M Ashwin", "MP Stoinis", "LH Ferguson", "OC McCoy", "HV Patel", "PVD Chameera", "PWH de Silva", "Ra
shid Khan", "YS Chahal", "Harpreet Brar", "Umran Malik", "M Markande", "MR Marsh", "MJ Santner", "MK Lomror", "R Sai
Kishore", "SP Narine", "Ravi Bishnoi", "Washington Sundar", "RD Chahar", "AR Patel", "JO Holder", "AS Joseph", "PH So
lanki", "CV Varun", "R Dhawan", "HR Shokeen", "Mukesh Choudhary", "KR Sen", "KH Pandya", "DR Sams", "DJ Bravo", "J Su
chith", "AS Roy", "Avesh Khan", "PJ Sangwan", "K Kartikeya"]'}

{'Partitions Read ': '["1", "2", "3"]'}
```

For analytical query:
For analytical query, execution which gets the result from multiple partitions is passed as input from jupyter notebook and executed on all the partition tables to map the required result and then aggregated based on the analytical function and reduced to return the reduced result from the analytical query

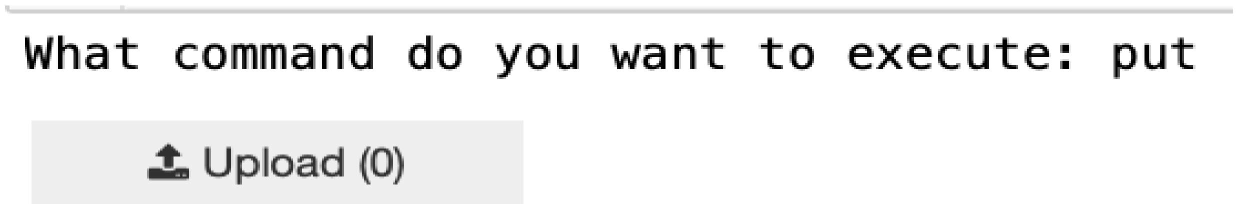


Search & Analytical Querying of EDFS

```
In [4]:
print("Tables: match_data, ball_data")
query = input("Enter query to execute: ")
url = "http://192.168.1.37:80/mapPartition/"+query
print()
r1 = requests.get(url)
temp = r1.json()
for i in temp:
    print(i)
    print()
```

```
Tables: match_data, ball_data
Enter query to execute: Select match_data.WinningTeam, sum(ball_data.total_run) from match_data,ball_data whe
re match_data.ID=ball_data.ID group by match_data.WinningTeam

{'Mapper1 ': {'Chennai Super Kings': '409', 'Delhi Capitals': '742', 'Gujarat Titans': '1373', 'Kolkata Knigh
t Riders': '865', 'Lucknow Super Giants': '1051', 'Punjab Kings': '1103', 'Rajasthan Royals': '1049', 'Royal
Challengers Bangalore': '905', 'Sunrisers Hyderabad': '990'}}

{'Mapper2 ': {'Chennai Super Kings': '702', 'Delhi Capitals': '923', 'Gujarat Titans': '1381', 'Kolkata Knigh
t Riders': '310', 'Lucknow Super Giants': '1350', 'Mumbai Indians': '319', 'Punjab Kings': '651', 'Rajasthan
Royals': '1115', 'Royal Challengers Bangalore': '1039', 'Sunrisers Hyderabad': '443'}}

{'Mapper3 ': {'Chennai Super Kings': '325', 'Delhi Capitals': '622', 'Gujarat Titans': '1138', 'Kolkata Knigh
t Riders': '578', 'Lucknow Super Giants': '695', 'Mumbai Indians': '868', 'Punjab Kings': '681', 'Rajasthan R
oyals': '1330', 'Royal Challengers Bangalore': '1055', 'Sunrisers Hyderabad': '383'}}

{'Reducer ': {'Chennai Super Kings': '1436', 'Delhi Capitals': '2287', 'Gujarat Titans': '3892', 'Kolkata Kni
ght Riders': '1753', 'Lucknow Super Giants': '3096', 'Mumbai Indians': '1187', 'Punjab Kings': '2435', 'Rajas
than Royals': '3494', 'Royal Challengers Bangalore': '2999', 'Sunrisers Hyderabad': '1816'}}

{'Partitions Read ': ['1', '2', '3']}
```

Task 3:
The UI we are developing is based on the Jupyter Notebook. We have built a basic UI using ipywidgets to take input such as command to execute and input parameters required to execute said command. Specifically for “put” we have also built a UI mechanism to browse the local file system directory to select the file we want to upload to the EDFS.

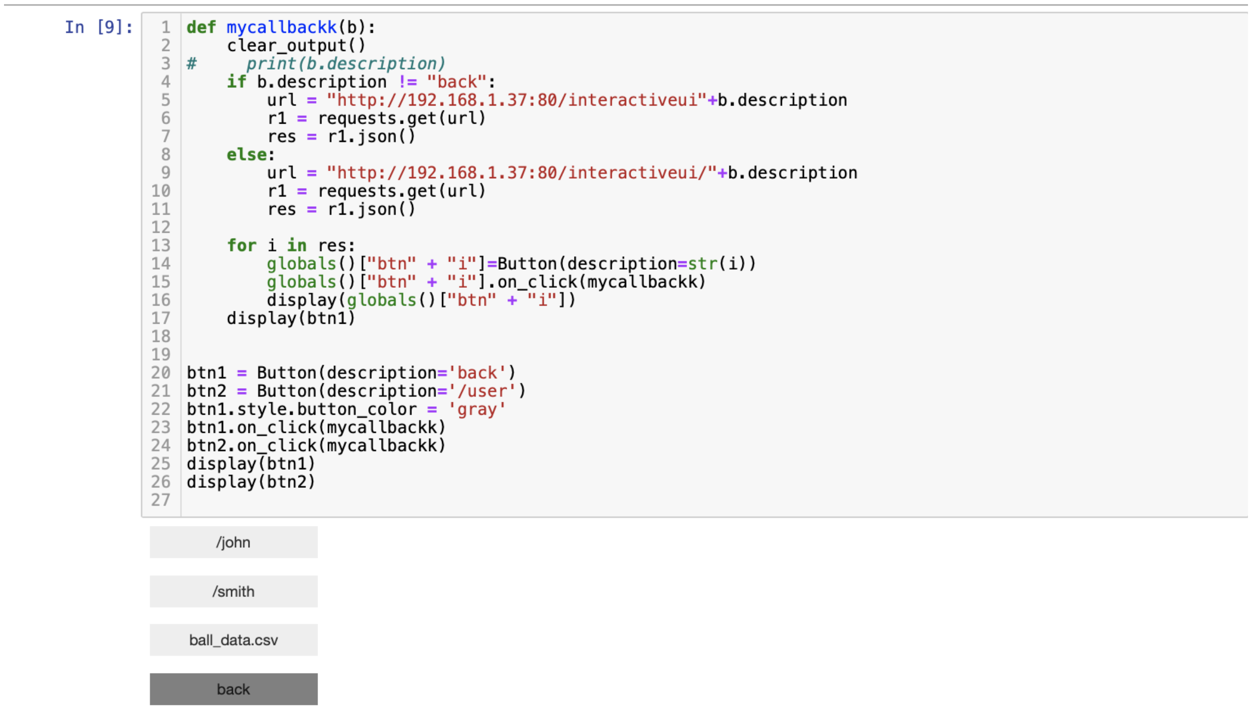


```
What command do you want to execute: put
```

On clicking the Upload button as shown above, a file browser window opens that allow the end user to select the file he wants to upload. As per the project requirements, the put command allows only the upload of “.csv” and “.txt” and if any other file format is being uploaded an error message is returned.

Additionally, we have also created an Interactive UI that the end user can use to explore through our EDFS.

Interactive UI



In [9]:
```python
def mycallbackk(b):
    clear_output()
#     print(b.description)
    if b.description != "back":
        url = "http://192.168.1.37:80/interactiveui"+b.description
        r1 = requests.get(url)
        res = r1.json()
    else:
        url = "http://192.168.1.37:80/interactiveui/"+b.description
        r1 = requests.get(url)
        res = r1.json()

    for i in res:
        globals()["btn" + "i"]=Button(description=str(i))
        globals()["btn" + "i"].on_click(mycallbackk)
        display(globals()["btn" + "i"])
    display(btn1)


btn1 = Button(description='back')
btn2 = Button(description='/user')
btn1.style.button_color = 'gray'
btn1.on_click(mycallbackk)
btn2.on_click(mycallbackk)
display(btn1)
display(btn2)

```

We make use of Buttons to select any directory and move forward to view all the folders and files of the selected directory. Additionally, we also have the “back” button to move back 1 directory just as you can do in an actual file system, moving forward and backward in the file system hierarchy.

Challenges:
One of the hardest challenges we faced was to understand how to implement the partition function. As there was a lot of implementation design required we had to interpret from our understanding of the project guideline document and come out with an intuitive solution of how we can emulate the partitioning of data in SQL which we have now implemented by splitting the data across different tables based on the match id of the row.

Learning Experience:
Learning how to handle database tables with multiple overwrites as the EDFS commands execute on the File Structure table.
Learning to work on a jupyter notebook and the interactive UI using it (ipywidgets).
Understanding the concept of how the partitions for a file and dataset are to be handled and implementing the conceptual understanding of it.
Additionally, we also obtained a great deal of knowledge in Map Partitions and how Hadoop Map Reduce exactly works.