Title : Emulating Firebase

Topic : Los Angeles Traffic Collision Data Analysis

Introduction :

This is an Application that emulates the functioning of Firebase database system. I have used MySQL in the backend as the database server to emulate Firebase. The application runs for CURL commands GET, PUT, POST, PATCH and DELETE like Firebase. It provides support for CURL commands with orderBy, limitToFirst, limitToLast, equalTo, startAt, endAt.

Data :

Structure of the data :

The data is Los Angeles Traffic Collision Data for last 10 years. The data contains details of victims of different traffic collisions occurring in LA in the last decade. The data is in a CSV file that contains columns DR Number, Data Occurred, Victim Age, Victim Descent, Area Name, Address and Location. The value of column DR Number is unique for each record of accident.

All the data, codes along with documentation/README file to run the commands has been added to a Google Drive Folder named 'Project-551'. The link for the folder is:

https://drive.google.com/drive/folders/1IP84DR6xiGHqpNbOrnKWaB6f2AHReyPy

*** Please **download all the necessary files first to a folder** and then run Python from inside that folder in terminal. Otherwise, the CSV file **might show a 'FileNotFound' error.**

Requirements :

1. Python3
2. Flask (pip install Flask)
3. MySQL connector (pip install mysql-connector-python)

All the pip commands to install the necessary packages have been provided in file **requirements.txt**. The commands to be typed in the terminal are :

1. **pip install -r requiments.txt**
2. **pip3 install -r requirements.txt**

For further safety, as I am not sure which pip version is present in the grader's system, please also try **pip3 install -r requirements.txt**

** **All commands have been included after Implementation and Points to be Noted with expected output and explanation. Please scroll down to check the commands. I have also added some invalid commands at the end and the expected output for each of them.**

Implementation :

1. **Transferring Data in MySQL** : The data used for this project is present in a CSV file named **p_data.csv**. The file has been added to the Google folder. The data has been inserted to a table named 'data' in MySQL using a .py file named data_transfer.py. **To be noted, the values of user and root are to be changed in the file at line 5 in data_transfer.py file:**

    cnx = mysql.connector.connect(user='**provide username'**, password='**provide password**', host='localhost', database='**provide database name**')

The grader needs to **change these values** based on **his/her system's MySQL username and passwords. Please make sure to change these values before starting the server file.**

*** The file **p_data.csv needs to be present in the same directory**. Here I have provided the relative path of the file at **line 18 'with open…..'.** If the file stays somewhere else on the grader's system, please **provide that path at this line in data_transfer.py file.**

To run this data_transfer.py, please type:
python3 data_transfer.py/python data_transfer.py

2. **Starting the server .py file**: The server_try.py file is to be started typing the following command from a terminal.

python3 server_try.py/python server_try.py

The above command will start running the server Python file. This Python file contains methods to handle all **GET, POST, PUT, PATCH** and **DELETE** requests coming with the **CURL** commands. Below is the mapping for different requests and the methods to be triggered :

GET : catch_all_get(myPath)
PUT : catch_all_put(myPath)
PATCH : catch_all_patch(myPath)
POST : catch_all_post(myPath)
DELETE : catch_all_delete(myPath)

Based on the request type, the specific method will be triggered, and the equivalent queries will run in the backend to return the results on the terminal.

3. **Using Flask to hear and intercept requests :**

I have used Flask to hear at **localhost**(**port 8000**) and catch the requests coming in. By default, localhost in my system is at port 8000. **Based on the grader's system, he/she can change this to specify the port number of localhost.** The change to the port is to be made at line **app.run(port=8000, debug=True)** at the end of server_try.py file. **Please make sure you specify the port number for localhost correctly.**

4. **<u>CURL commands entered from terminal</u>** : CURL commands are to entered from terminal with methods like GET, POST, PUT, PATCH and DELETE.

   Example :

   ```
   curl -X 'http://localhost:8000/data.json?orderBy="Area%20Name"&limitToFirst=50'
   ```

5. **<u>Running the SQL queries</u>**: In the backend, the equivalent SQL queries run like for a GET method, it runs an SQL query with '**SELECT * from data where**….'. More details about running commands and their respective outputs **are provided later**.

<u>Points to be noted :</u>

1. While working repeatedly with the CURL commands and repeatedly retrieving/updating records, I noticed some times the **server gets stuck** and nothing happens (no error message/exception, just idle screen). Under this circumstance, please **shutdown the server, wait for 3 seconds and then restart it.** All operations will work smoothly.

2. For some GET methods, there are too many rows to fetch and show so the server might take long to return a value. **I have added adequate, user-friendly print debugging statements to the server_try.py console so that the graders can know which step is getting executed.** Ideally, the commands shouldn't stay stuck or take too long so if this happens, request the grader to please **restart the server Python file(shut it down and restart after 3 seconds).**

3. The results **take longer to be displayed on ec2** instance than on my personal system. Please ensure **no other service like Hadoop/Spark or any such BigData service is running** on the instance before starting the server Python file. Please be patient as the results take little bit more time to load on ec2 than on normal device.

Commands and expected outputs :

**** Please type the exact commands as listed below for seeing output. To be noted, here %20 in column name stands for a blank space( ) character because in URL, blank space isn't allowed.**

GET commands :

The GET commands are used to fetch/retrieve data from Firebase. It can have parameters like **orderBy, limitToFirst, limitToLast, startAt, endAt, equalTo, print=pretty.**

Examples :

Command :

curl -X GET 'http://localhost:8000/data.json?orderBy="Area%20Name"&startAt="Southeast"&endAt="Southwest"'

Output :

The output of the command will have 455 rows printed so I will just show a snippet of what should be printed.

{"Date Occurred": "2022-05-23", "Area Name": "Southwest", "Victim Age": 47.0, "Victim Sex": "M", "Victim Descent": "B", "Address": "ARLINGTON", "Location": "(34.0, -118.3182)"}

{"Date Occurred": "2022-05-27", "Area Name": "Southwest", "Victim Age": 52.0, "Victim Sex": "M", "Victim Descent": "H", "Address": "OBAMA BL", "Location": "(34.0215, -118.3556)"}

{"Date Occurred": "2022-05-27", "Area Name": "Southwest", "Victim Age": 56.0, "Victim Sex": "M", "Victim Descent": "H", "Address": "COLISEUM", "Location": "(34.0183, -118.3521)"}

{"Date Occurred": "2019-09-21", "Area Name": "Southwest", "Victim Age": 52.0, "Victim Sex": "M", "Victim Descent": "B", "Address": "MARTIN LUTHER KING JR BL", "Location": "(34.0124, -118.3351)"}

Backend SQL query :

SELECT * from data WHERE `Area Name`>="Southeast" AND `Area Name`<="Southwest" ORDER BY `Area Name`ASC;

Explanation :

This command will first sort the table data based on column name 'Area Name'. Then it starts at Area Name Southeast and ends at Southwest and returns all the rows that have area names between this range.

Command :

curl -X GET 'http://localhost:8000/data.json?orderBy="Area%20Name"&limitToLast=50'

Output :

The output will print 50 rows so I will share a snippet of how the output should look like :

{"Date Occurred": "2019-07-09", "Area Name": "Wilshire", "Victim Age": 49.0, "Victim Sex": "M", "Victim Descent": "O", "Address": "WILSHIRE BL", "Location": "(34.062, -118.3291)"}

{"Date Occurred": "2019-07-12", "Area Name": "Wilshire", "Victim Age": 56.0, "Victim Sex": "M", "Victim Descent": "O", "Address": "3RD ST", "Location": "(34.0689, -118.3386)"}

{"Date Occurred": "2019-07-12", "Area Name": "Wilshire", "Victim Age": 25.0, "Victim Sex": "F", "Victim Descent": "W", "Address": "BEVERLY BL", "Location": "(34.0762, -118.3532)"}

{"Date Occurred": "2019-07-12", "Area Name": "Wilshire", "Victim Age": 23.0, "Victim Sex": "M", "Victim Descent": "B", "Address": "HIGHLAND", "Location": "(34.0835, -118.3386)"}

{"Date Occurred": "2019-07-13", "Area Name": "Wilshire", "Victim Age": 63.0, "Victim Sex": "F", "Victim Descent": "W", "Address": "SAN VICENTE BL", "Location": "(34.0537, -118.3619)"}

Backend SQL query :

```
SELECT * from data ORDER BY `Area Name` DESC LIMIT 50
```

Explanation :

This command will first sort the data in table based on column ‘Area name’ in descendinh order. Then it will return the first 50 rows from the result.

Command :

```
curl -X GET 'http://localhost:8000/data.json?orderBy="Area%20Name"&equalTo="West%20LA"'
```

Output :

This would return all the rows where the area name is equal to West LA. As per the data there are 268 rows so I will share just a snippet of the results.

{"Date Occurred": "2022-03-28", "Area Name": "West LA", "Victim Age": 50.0, "Victim Sex": "F", "Victim Descent": "W", "Address": "WILSHIRE BL", "Location": "(34.0643, -118.4321)"}

{"Date Occurred": "2022-04-04", "Area Name": "West LA", "Victim Age": 46.0, "Victim Sex": "F", "Victim Descent": "B", "Address": "SHENANDOAH ST", "Location": "(34.0418, -118.3848)"}

{"Date Occurred": "2022-05-18", "Area Name": "West LA", "Victim Age": 21.0, "Victim Sex": "X", "Victim Descent": "O", "Address": "CENTINELA", "Location": "(34.0313, -118.4593)"}

{"Date Occurred": "2022-05-13", "Area Name": "West LA", "Victim Age": 49.0, "Victim Sex": "M", "Victim Descent": "H", "Address": "ANGELO DR", "Location": "(34.0632, -118.4237)"}

{"Date Occurred": "2022-05-25", "Area Name": "West LA", "Victim Age": 39.0, "Victim Sex": "M", "Victim Descent": "B", "Address": "CASIANO RD", "Location": "(34.0819, -118.4719)"}

Backend SQL query :

```
SELECT * from data WHERE `Area Name`="West LA";
```

Explanation : This command will fetch all the rows from the database having area name as West LA.

Command :

```
curl -X GET
'http://localhost:8000/data.json?orderBy="Victim%20Age"&startAt=90&endAt=96
'
```

Output :

{"Date Occurred": "2019-04-11", "Area Name": "West Valley", "Victim Age": 90.0, "Victim Sex": "M", "Victim Descent": "W", "Address": "RESEDA BL", "Location": "(34.18, -118.536)"}

{"Date Occurred": "2019-06-18", "Area Name": "N Hollywood", "Victim Age": 90.0, "Victim Sex": "F", "Victim Descent": "W", "Address": "RADFORD AV", "Location": "(34.1437, -118.3929)"}

{"Date Occurred": "2019-09-05", "Area Name": "N Hollywood", "Victim Age": 90.0, "Victim Sex": "M", "Victim Descent": "O", "Address": "COLDWATER CANYON AV", "Location": "(34.1794, -118.4137)"}

{"Date Occurred": "2022-05-25", "Area Name": "West Valley", "Victim Age": 91.0, "Victim Sex": "M", "Victim Descent": "W", "Address": "LOUISE AV", "Location": "(34.1894, -118.5098)"}

{"Date Occurred": "2019-09-04", "Area Name": "N Hollywood", "Victim Age": 96.0, "Victim Sex": "F", "Victim Descent": "W", "Address": "WEDDINGTON ST", "Location": "(34.167, -118.3752)"}

Backend SQL query :

SELECT * from data WHERE `Victim Age`>=90 AND `Victim Age`<=96 ORDER BY `Victim Age`ASC;

Explanation : This command will fetch all the records from the database that have their ages between 90 and 96 inclusive. There are 5 such records in the table as shown.

Command :

curl -X GET 'http://localhost:8000/data/Victim%20Age.json?print=pretty'

Output : The command will show all the ages of victims from table data in proper format so I will show a snippet of how the output will look like.

"55.0"
"55.0"
"32.0"
"21.0"
"51.0"
"52.0"
"63.0"
"41.0"
"35.0"
"20.0"
"51.0"
"23.0"
"53.0"

Backend SQL query :

```
SELECT `Victim Age` from data;
```

Explanation :

Like Firebase this command will first fetch the ages of the victims and then will print them in the above format on terminal.

PUT command :

The PUT command can be used to both insert new records and update existing records in Firebase. Likewise, this application would both insert and update records (**UPSERT**). Below are the commands for both.

Example :

To **INSERT a new record**, please type :

Command :

```
curl -X PUT 'http://localhost:8000/data/ANI58.json' -d '{"Date Occurred":"2021-12-23","Area Name" :"Kudghat","Victim Age" : 34.09,"Victim Sex" : "F","Victim Descent" : "Bengali","Address" : "386A BP Road","Location": "(9,10)"}'
```

Output : The command would add a new record to the table with key “ANI58”. The result of the command printed on the terminal would be :


```
{
  "Address": "386A BP Road",
  "Area Name": "Kudghat",
  "Date Occurred": "2021-12-23",
  "Location": "(9,10)",
  "Victim Age": 34.09,
  "Victim Descent": "Bengali",
  "Victim Sex": "F"
}
```


Backend SQL query :

```
INSERT INTO data (`Dr Number`,`Date Occurred`,`Area Name`,`Victim Age`,`Victim Sex`,`Victim Descent`,`Address`,`Location`) values ("ANI58","2021-12-23","Kudghat",34.09,"F","Bengali","386A BP Road","(9,10)");
```

Explanation : This command would add a new record with key “ANI58” to the database. The values of specific columns will be as provided in the CURL query.

PUT to insert (another way) :

Command :

curl -X PUT 'http://localhost:8000/data.json' -d '{"ANI58" : {"Date Occurred":"2021-12-23","Area Name" :"Kudghat","Victim Age" : "","Victim Sex" : "F","Victim Descent" : "Bengali","Address" : ""}}'

Output : This command will add a record with key “ANI58” to the data. The difference between the first and this command is here **we don’t specify the key of the record in the path of request**, we rather **pass it as a value in the data** -d part. But both ways, PUT will add a new record to the table if it doesn’t exist.


```
{
  "Address": "",
  "Area Name": "Kudghat",
  "Date Occurred": "2021-12-23",
  "Victim Age": "",
  "Victim Descent": "Bengali",
  "Victim Sex": "F"
}
```


Backend SQL query :

INSERT INTO data (`Dr Number`,`Date Occurred`,`Area Name`,`Victim Age`,`Victim Sex`,`Victim Descent`,`Address`) values ("ANI58","2021-12-23","Kudghat",NULL,"F","Bengali",NULL);

Explanation : This command would add a new record with key “ANI58” to the database. Notice that the **columns not specified with a value in the data -d in CURL query is added with a NULL value.**

PUT to UPDATE :

As said before, Firebase PUT can also be put to **update an existing record.**

Command to type for **PUT UPDATE** :

```
curl -X PUT 'http://localhost:8000/data.json' -d '{"ANI58" : {"Date Occurred":"2021-12-23","Area Name" :"Kudghat","Victim Age" : "","Victim Sex" : "F","Victim Descent" : "Bengali","Address" : "","Location": "(19,10)"}}'
```

Output : This command would update the record having key “ANI58”. The result of the CURL command would look like :


```
{
  "Address": "",
  "Area Name": "Kudghat",
  "Date Occurred": "2021-12-23",
  "Location": "(19,10)",
  "Victim Age": "",
  "Victim Descent": "Bengali",
  "Victim Sex": "F"
}
```


Backend SQL query :

```
UPDATE data set `Date Occurred`="2021-12-23",`Area Name`="Kudghat",`Victim Age`=NULL,`Victim Sex`="F",`Victim Descent`="Bengali",`Address`="",`Location`="(19,10)" where `DR Number` = "ANI58";
```

Explanation :

The command updates the specific column values as mentioned with the data -d of the request path. Note that there can be column names having a blank value as

Victim Age above. Fot these columns, NULL value would be stored. I have printed all the column names and their values(even the blank ones) as per a Piazza post.

Another PUT to update :

Command :

curl -X PUT 'http://localhost:8000/data.json' -d '{"ANI58":{"Address":"fhghg5","Location":"fgf45g","Date Occurred":""}}'

Output :

This command would update the values of columns Address, Location and Date Occurred with the specific values as provided with the data of the CURL request. Rest of the values would be the same. The output will look as follows :


```
{
  "Address": "fhghg5",
  "Date Occurred": "",
  "Location": "fgf45g"
}
```


Backend SQL query :

UPDATE data set `Address`="fhghg5",`Location`="fgf45g",`Date Occurred`=NULL where `DR Number` = "ANI58";

Explanation :

This command would update the values of columns Address, Location and Date Occurred with the specific values as provided with the data of the CURL request. Rest of the values would be the same in the database table. Note that the value of column Data Occurred has an empty string so for this, a NULL value will be updated to the database.

Note : For **updating a record**, please type commands only in the **following format** :

curl -X PUT 'http://localhost:8000/data.json' -d '{"ANI58" : {"Date Occurred":"2021-12-23","Area Name" :"Kudghat","Victim Age" : "","Victim Sex" : "F","Victim Descent" : "Bengali","Address" : "","Location": "(19,10)"}}'

OR

curl -X PUT 'http://localhost:8000/data.json' -d '{"ANI58":{"Address":"fhghg5","Location":"fgf45g","Date Occurred":""}}'

The first command updates values of columns Date Occurred, Area Name, Victim Age, Victim Sex, Victim Descent, Address and Location with the respective values.

The second command updates values of columns Date Occurred, Address and Location with the respective values.

The value **can be a blank string for any column** but the format above i.e. **specifying the key with data and not with the path of request** has to be followed to be able to see changes on the database.

Please follow the specified format for updating records with PUT command.

POST command :

The POST command can be used to add records to Firebase. For POST, Firebase automatically generates a key for storing the record as specified with the request path. The key generated by Firebase gets displayed on the terminal.

Examples :

Command :

curl -X POST 'http://localhost:8000/data.json' -d
'{"Address":"Scarff","Location":"riffith"}'

Output :

The POST command would insert a new record to the database with a key that it generates. This key would be returned to the terminal.


```
{
  "name": "ANI48"
}
```


Backend SQL query :

INSERT INTO data (`Dr Number`,`Address`,`Location`) values
("ANI48","Scarff","riffith");

Explanation :

The POST command **inserts a new record to the database with respect to key "ANI48".** This key is generated automatically and isn't provided by the data in the request path as shown above. The new key is printed on the terminal.

PATCH command :

The PATCH command is used to **update an existing record** in Firebase.

Example :

Command :

curl -X PATCH 'http://localhost:8000/data/ANI48.json' -d '{"Address":"ghgh"}'

Output :

The PATCH command updates the value of the record stored with respect to key “ANI48”. The record with key “ANI48” already exists in the database because it was created with POST method(above). The specific column values updated gets displayed on the terminal.


```
{
  "Address": "ghgh"
}
```


Backend SQL query :

UPDATE data set `Address`="ghgh" where `DR Number` = "ANI48";

Explanation :

The PATCH command **updates the value of the record** stored with respect to key “ANI48”. The record with key “ANI48” already exists in the database because it was created with POST method(above). The specific column values updated gets

displayed on the terminal. Here as per the data specified, the **value of column 'Address' gets updated to 'ghgh'.**

Another PATCH command :

Command :

```
curl -X PATCH 'http://localhost:8000/data/ANI48.json' -d
'{"Address":"ghgh","Victim%20Age":34}'
```

Output : The CURL PATCH command **updates the value** stored with respect to key "ANI48". The results printed on terminal would look like :


```
{
  "Address": "ghgh",
  "Victim%20Age": 34
}
```


Backend SQL query :

```
UPDATE data set `Address`="ghgh",`Victim Age`=34 where `DR Number` =
"ANI48";
```

Explanation :

The PATCH command **updates the value of the record** stored with respect to key "ANI48". The record with key "ANI48" already exists in the database because it was created with POST method(above). The specific column values updated gets displayed on the terminal. Here as per the data specified, the **value of column 'Address' gets updated to 'ghgh' and the value of column 'Victim Age' gets updated to 34.**

Note :

For seeing changes with PATCH command, please **ensure the record already exists** like here the key specified with request path curl -X PATCH 'http://localhost:8000/data/ANI48.json' i.e. ANI48 already exists in the database. If not, **you won't be able to see the changes** made to the database because of the PATCH command.

DELETE command :

The DELETE command is used to delete a record from the Firebase. If the record **exists** in Firebase database, the message with **status 200** is printed. If not, the message is printed with **status 'Error'**.

Example :

Command :

curl -X DELETE 'http://localhost:8000/data/ANI48.json'

Output : The DELETE command will delete the record stored with respect to key 'ANI48'. If the deletion is successful, the following message is printed on terminal.

```
{
  "Status": "200"
}
```

Backend SQL query :

DELETE from data where `DR Number`="ANI48"

Explanation : The CURL DELETE command deletes the record stored with key “ANI48” in the database. If the record exists, the deletion is successful so status : 200 will be printed.

Another example of **DELETE** : (**when the record doesn’t exist**)

Command : (same as above, but now the record doesn’t exist)

curl -X DELETE 'http://localhost:8000/data/ANI48.json'

Output :


```
{
  "Status": "Error"
}
```


Explanation : The record with key “ANI48” doesn’t exist in the database now so the DELETE would return a status error.

Invalid Command :

In Firebase, if there’s an **error with the JSON structure** of the data, it would show us an error message on terminal like ‘Invalid parse with JSON’. Similarly, here if there’s a problem with the JSON format, we would see an error message printed on terminal.

Example :

Command :

curl -X PATCH 'http://localhost:8000/data/ANI48.json' -d '{"Address":"ghgh",'Victim%20Age':""}'

Output :

{
"error": "Invalid data; couldn't parse JSON object. Are you sending a JSON object with valid key names?"
}

Explanation : Here there's a problem with key 'Victim Age' that has been **specified within single quotes**. This isn't a valid JSON format to represent data in Firebase. Hence it prints the above error.

Example :

Command :

curl -X GET 'http://localhost:8000/data.json?orderBy="AreaName"&startAt="Southeast"&endAt="Southwest"'

Output :

Null

Explanation :

There's no such column as AreaName in database. The actual column name is **"Area Name"** so it returns a "null".

My key learnings and possible extensions :

1. This project helped me **understand the mapping** between Firebase CURL commands and equivalent SQL queries.
2. After running the server for all CURL commands, I realized that **MySQL beats Firebase for multiple-rows transaction** (less latency) e.g. for queries of bulk-update/bulk-insert.
3. The structure of the queries is **easier in Firebase** than in MySQL.
4. The error messages displayed on Firebase are much **user-friendly and easy** to follow unlike MySQL which has a generic error or stack trace.
5. Yet again, one of the biggest constraints of Firebase, is that it **works with only JSON data**. In MySQL, we can use data from any file easily like for this I used to dump data from a CSV file.

Possible extensions :

1. Using **LONGBLOB in MySQL** to store variable amount of data.
2. **Split counter over rows** for volatile data.
3. Use **ProxySQL/Vitess** for MySQL Sharding/scalability.

Overall, working on the project was a source of great learning for me. This project would also be of utmost importance to people new to the concept of Firebase or Big Data system.