Big Data Homework-3

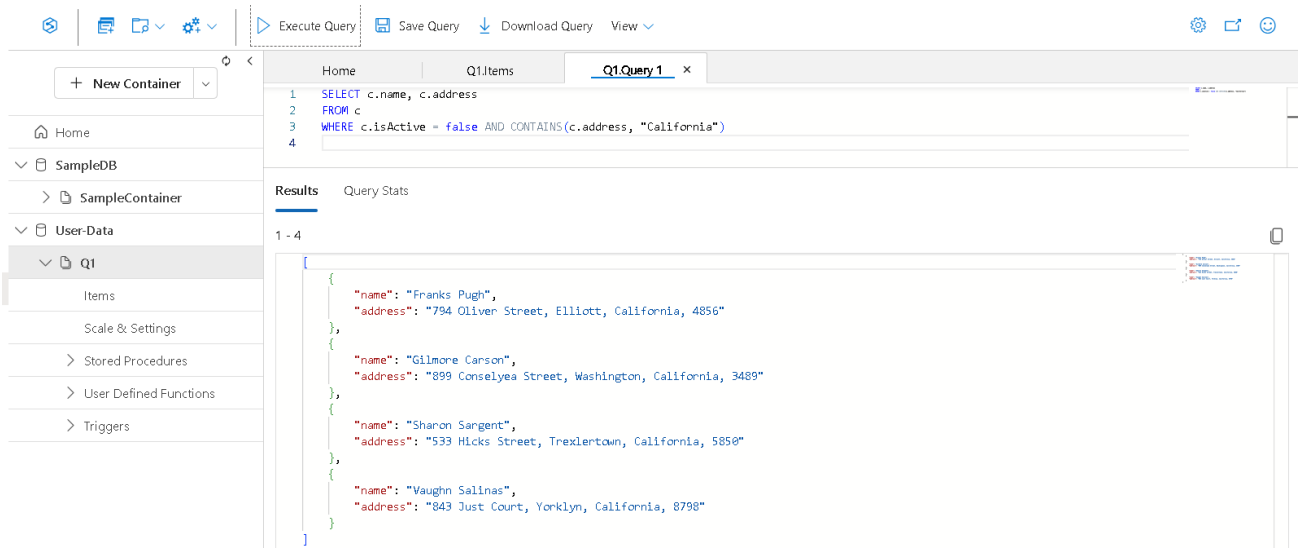
Name: Akshat Mishra | net_id: am15111 | Spring 2025

Step 1: Password for Dataset 1:
Concatenate the names of two Azure Cosmos DB functions: one that retrieves the current date and time, and another that calculates the square root of a given number. (In CAPS without brackets):
In **Azure Cosmos DB**, the two functions you need are:
GETCURRENTDATETIME() – Retrieves the current date and time.
SQRT() – Calculates the square root of a given number.

So, the password is: GETCURRENTDATETIMESQRT

Question1a: User Information Data Analysis Retrieve the name and address of all inactive users who live in the state of California

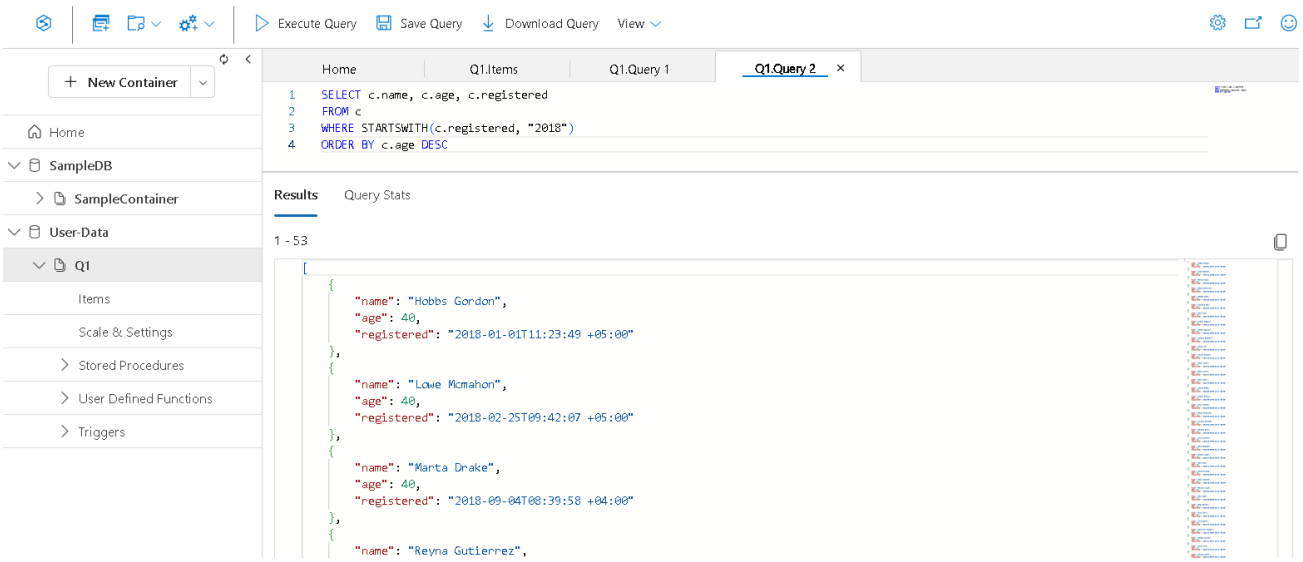
```
SELECT c.name, a.address
FROM c
WHERE c.isActive = false AND CONTAINS(c.address, “California”)
```



1b. Query to retrieve the list of users who registered in 2018, sorted by their age in descending order. (Hint: A UDF can be used to solve this problem.)

Query:

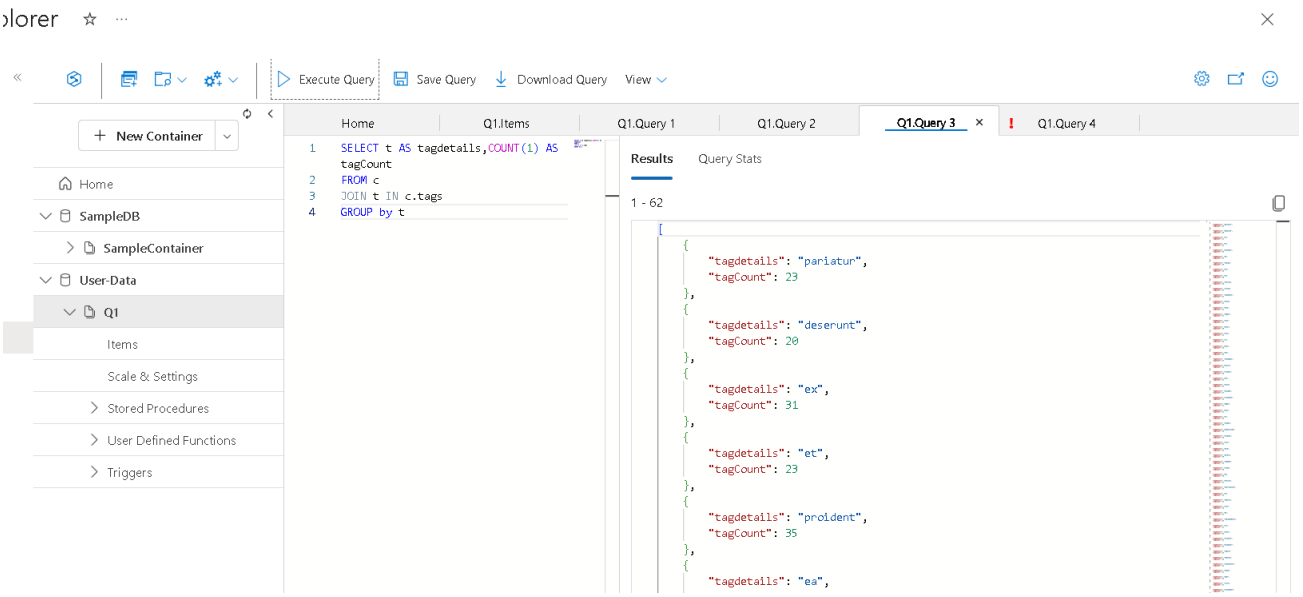
```
SELECT c.name, c.age, c.registered
FROM c
WHERE STARTSWITH(c.registred, “2018”)
ORDER BY c.age DESC
```



1c. Each user has a list of tags in their tags array. Write a query to find all unique tags in the collection and count how many times each tag appears across all users.

Query:

```
SELECT t AS tagdetails, COUNT(1) AS tagcount
FROM c
JOIN t IN c.tags
GROUP by t
```

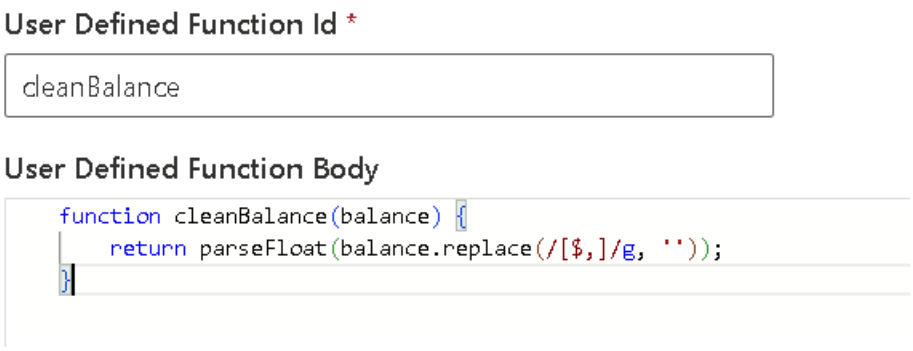


1d. Query to calculate the total sum of balance for all active users. (Hint :A UDF can be used to solve this problem.)

User Defined Function:

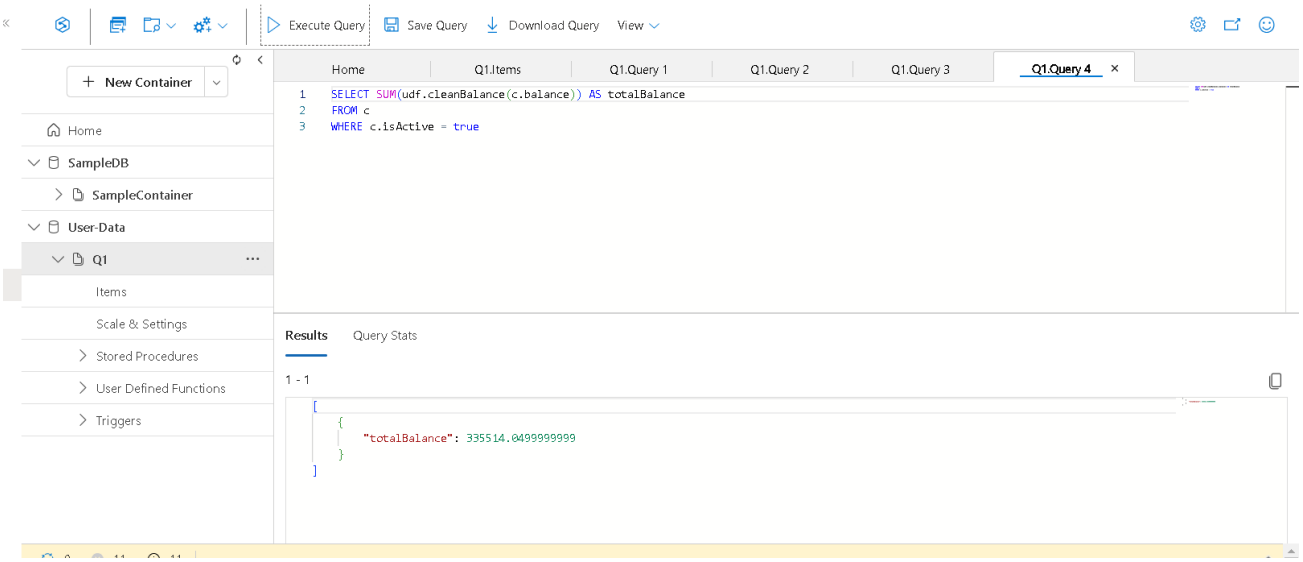
```
function getBalance(balanceStr) {
return parseFloat(balanceStr.replace(/[$,]/g, ''));
}
```

UDF Screenshot:



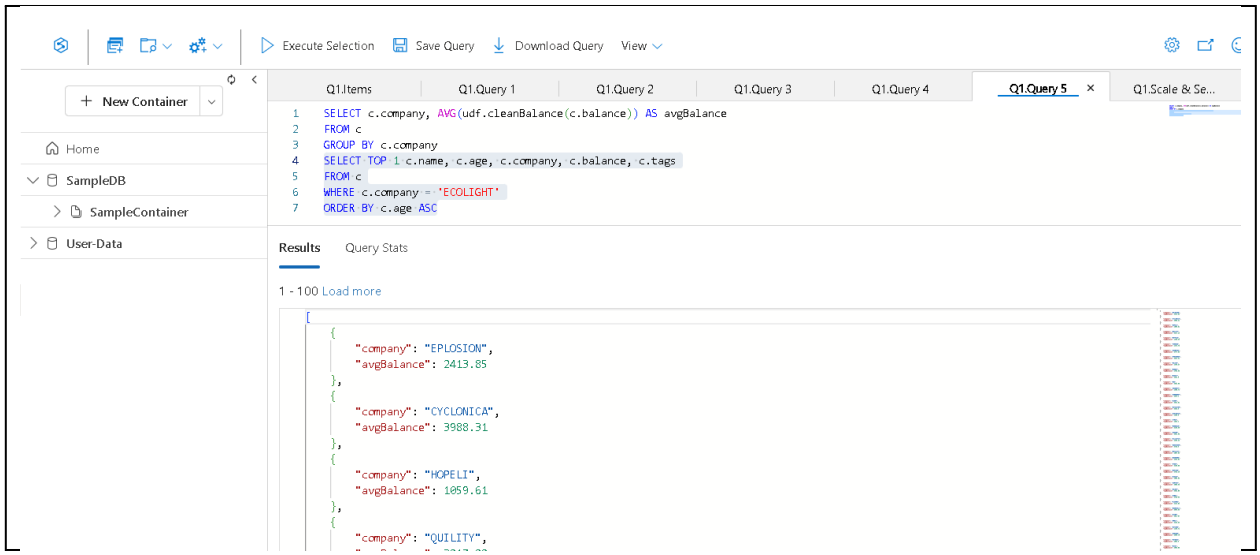
Query Execution:

```
SELECT SUM(udf.cleanBalance(c.balance)) AS totalBalance
FROM c
WHERE c.isActive = true
```



1e. Query to find the name,age,company,balance and tags of the youngest user from the company with the highest avg balance (You may use two queries to derive this result):

```
SELECT c.company, AVG(udf.cleanBalance(c.balance)) AS avgBalance
FROM c
GROUP BY c.company
```

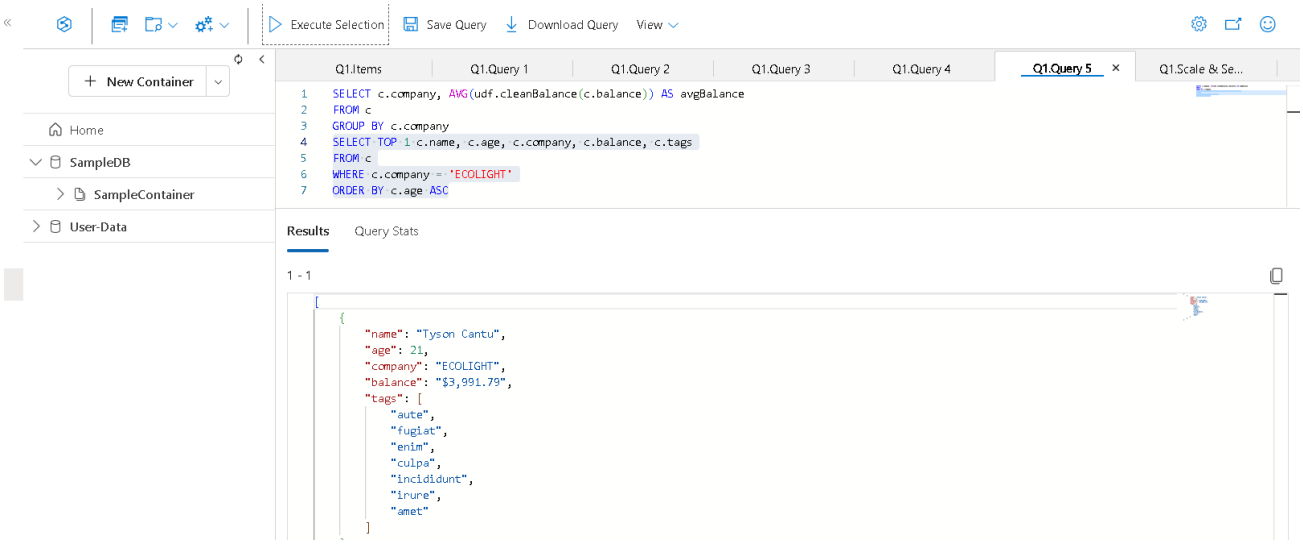


Company with Highest Average Balance: “ECOLIGHT”

Searching for youngest user for ‘ECOLIGHT’

Query2:

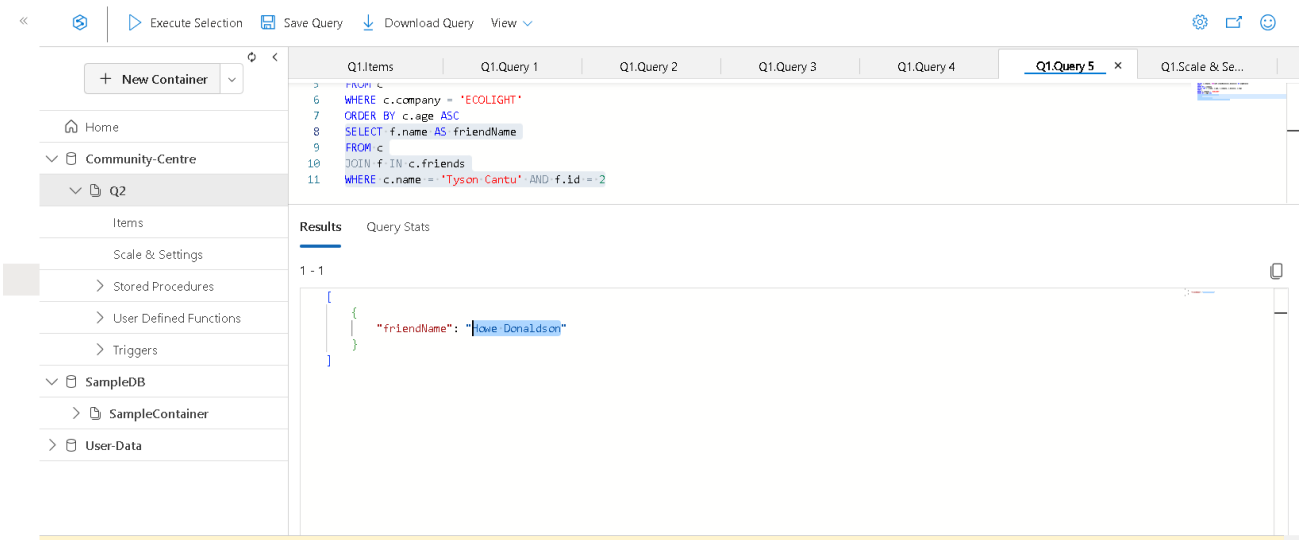
```
SELECT TOP 1 c.name, c.age, c.company, c.balance, c.tags
FROM c
WHERE c.company = 'ECOLIGHT'
ORDER BY c.age ASC
```



Query output: Tyson Cantu

Now, To unlock the data file for Part(2) - We need to find the full name of the friend with id=2 of Tyson

```
SELECT f.name AS friendName
FROM c
JOIN f IN c.friends
WHERE c.name = ‘Tyson Cantu’ AND f.id = 2
```



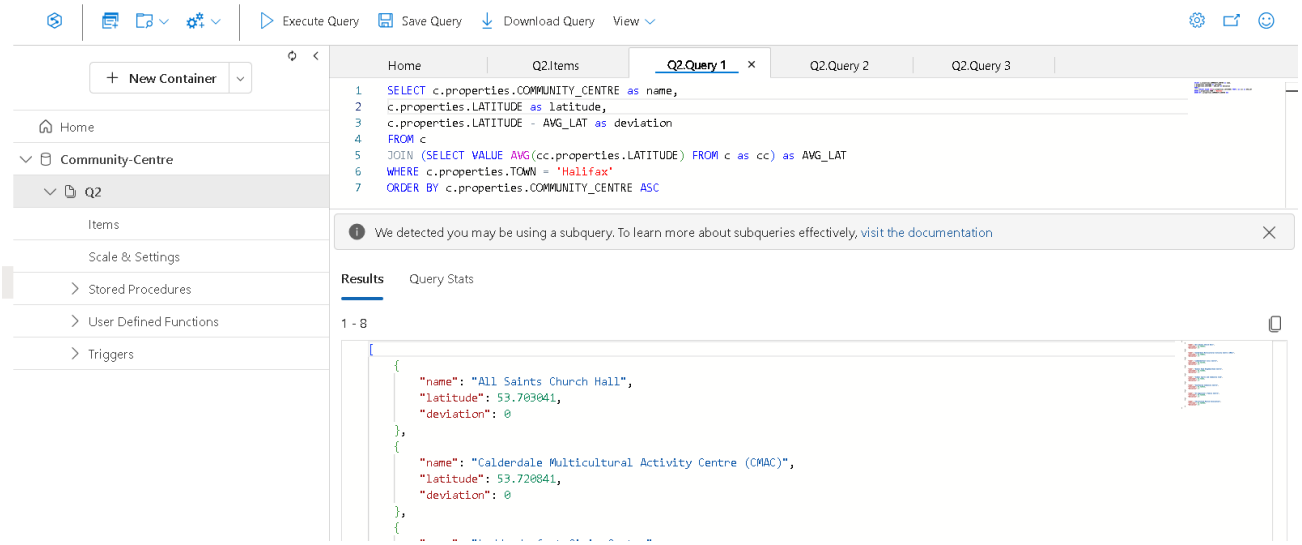
The name with (id = 2) is HOWE DONALDSON
SO PASSWORD is “HOWEDONALDSON”

Q2) Question 2:

For each community centre in Halifax,query to compute the deviation of its latitude from the average latitude of all the Community centres in our json file and display the results sorted in ascending order by community center name. Attempt to achieve this in a single query in Cosmos DB and share the output. Do you think the results are accurate? If not, explain why the one-query approach may not work as expected. Include screenshots of the output obtained.

Query)

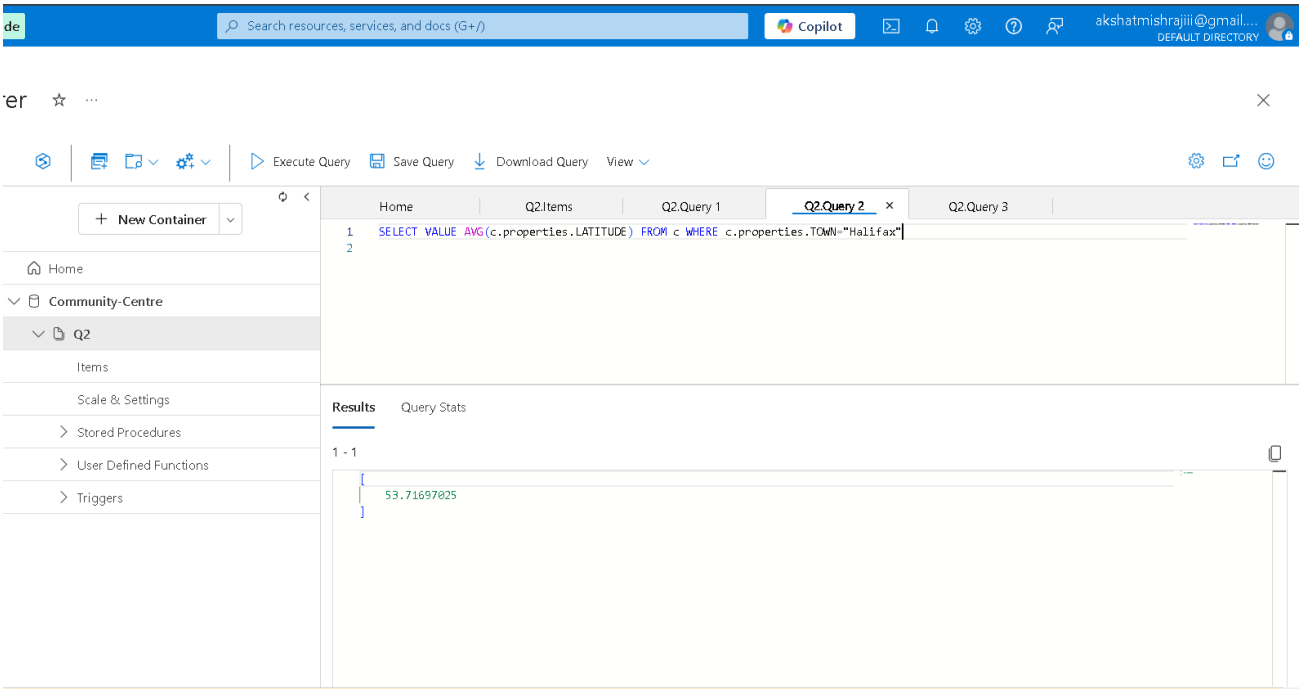
```
SELECT c.properties.COMMUNITY_CENTRE as name, c.properties.LATITUDE as latitude,
c.properties.LATITUDE - AVG_LAT as deviation
FROM c
JOIN (SELECT VALUE AVG(cc.properties.LATITUDE) FROM c as cc) as AVG_LAT WHERE
c.properties.TOWN = 'Halifax'
ORDER BY c.properties.COMMUNITY_CENTRE ASC
```



Using a single-query approach with a JOIN subquery to compute the average latitude causes all deviations to appear as 0. This happens because Cosmos DB's query engine does not properly apply the calculated average to each document. To resolve this issue, the solution involves executing two separate queries instead.

Now we have to use 2 queries to solve 2.a. Keep the name of the community centre with the least absolute deviation.

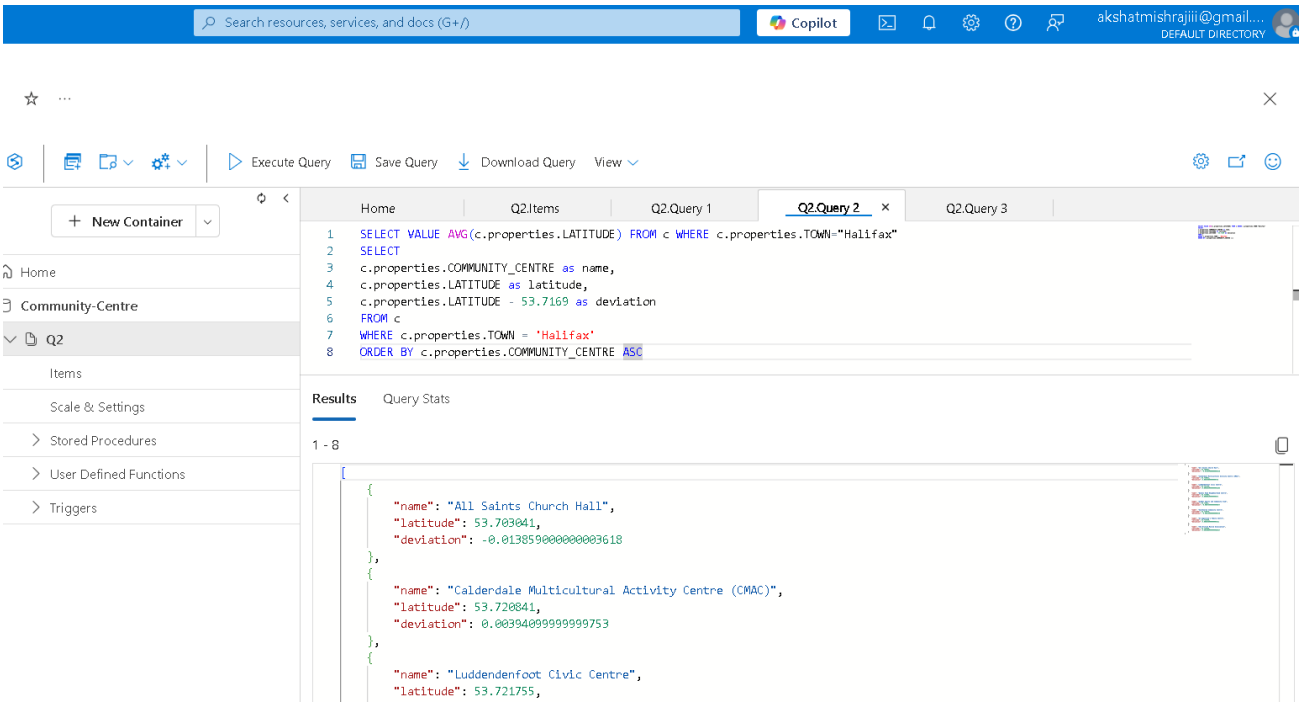
```
SELECT AVG(c.properties.LATITUDE) as avg_latitude
FROM c
WHERE c.properties.TOWN = “Halifax”
```



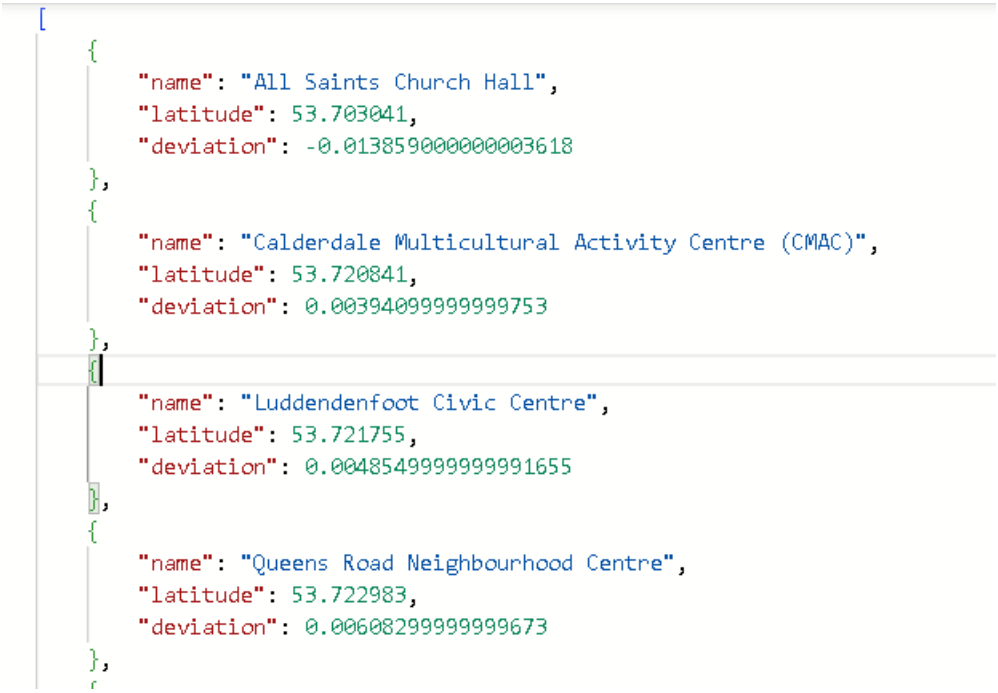
After getting the average value, use that exact number in next query

```
SELECT
properties.COMMUNITY_CENTRE as name, c.properties.LATITUDE as latitude,
c.properties.LATITUDE - 53.7169 as deviation
FROM c
WHERE c.properties.TOWN = 'Halifax'
ORDER BY c.properties.COMMUNITY_CENTRE ASC
```

In the two-query approach, we first compute the average latitude (53.7169) and then use this calculated value directly in the second query.



In the two-query approach, the average latitude (53.7169) is first calculated separately and then manually inserted into the second query. As a result, the deviations are accurately computed, yielding precise values. This highlights why a single-query approach fails in Cosmos DB as its query engine struggles with cross-document aggregations when the computed result is used in further calculations within the same query. Attached below are the execution result screenshots:



```
[
    {
        "name": "All Saints Church Hall",
        "latitude": 53.703041,
        "deviation": -0.013859000000003618
    },
    {
        "name": "Calderdale Multicultural Activity Centre (CMAC)",
        "latitude": 53.720841,
        "deviation": 0.00394099999999753
    },
    {
        "name": "Luddendenfoot Civic Centre",
        "latitude": 53.721755,
        "deviation": 0.0048549999999991655
    },
    {
        "name": "Queens Road Neighbourhood Centre",
        "latitude": 53.722983,
        "deviation": 0.00608299999999673
    },
    {
        "name": "Siddal Sports and Community Club",
        "latitude": 53.71014,
        "deviation": -0.006759999999999877
    },
    {
        "name": "Southowram Community Centre",
        "latitude": 53.710734,
        "deviation": -0.006166000000000338
    },
    {
        "name": "St Augustine's Family Centre",
        "latitude": 53.725559,
        "deviation": 0.008658999999994421
    },
    {
        "name": "The British Muslim Association",
        "latitude": 53.720709,
        "deviation": 0.0038089999999968427
    }
]
```


The community center with the smallest absolute deviation is "The British Muslim Association" [Absolute deviation = 0.00380]

2c. Now, using the same dataset on which you wrote the previous two queries you need to make a modification. Your task is to write a Python script that restructures this dataset into GeoJSON format for geospatial operations in databases like Cosmos DB. Each city's coordinates (latitude and longitude) should be stored as a "Point" type to enable efficient geospatial querying. Ensure the dataset is ready for ingestion into Cosmos DB, retaining all the other information about the community centre and the transformed geospatial data. Refer to the Cosmos DB documentation for guidance on its geospatial data format.

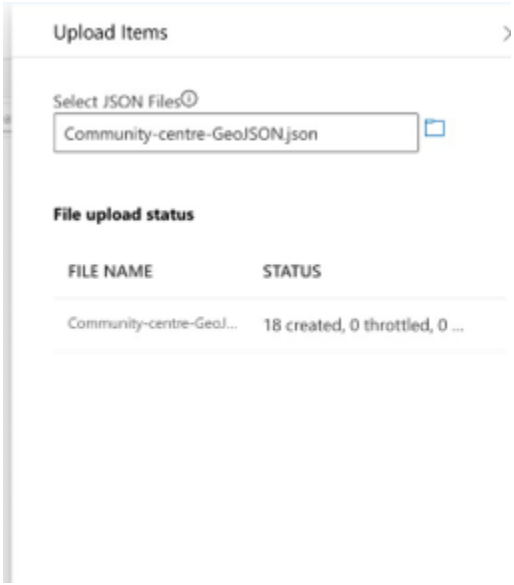
```python
import json

# Read the input file
with open('Community-centre.json', 'r') as file:
    data = json.load(file)

# Check if input is a FeatureCollection and extract features
if isinstance(data, dict) and 'features' in data:
    features = data['features']
else:
    features = data  # Assuming input is already a list of features

geojson_data = []

for feature in features:
    try:
        properties = feature.get('properties', {})
        geojson_feature = {
            "id": feature.get("id", ""),
            "type": "Feature",
            "properties": {
                "COMMUNITY_CENTRE": properties.get("COMMUNITY_CENTRE", ""),
                "ADDRESS1": properties.get("ADDRESS1", ""),
                "ADDRESS2": properties.get("ADDRESS2", ""),
                "TOWN": properties.get("TOWN", ""),
                "POSTCODE": properties.get("POSTCODE", ""),
                "PHONE": properties.get("PHONE", ""),
                "UPRN": properties.get("UPRN", ""),
                "OTHER_INFORMATION": properties.get("OTHER_INFORMATION", "")
            },
            "geometry": {
                "type": "Point",
                "coordinates": [
                    float(properties.get("LONGITUDE", 0)),
                    float(properties.get("LATITUDE", 0))
                ]
            }
        }
        geojson_data.append(geojson_feature)
    except Exception as e:
        print(f"Error processing feature {feature.get('id', 'unknown')}: {e}")

# Write the output file
with open('community-centre-GeoJSON.json', 'w') as outfile:
    json.dump(geojson_data, outfile, indent=2)

print(f"Changed {len(geojson_data)} community centers to GeoJSON format")
```

```
import json

# Read the input file
with open('Community-centre.json', 'r') as file:
    data = json.load(file)

# Check if input is a FeatureCollection and extract features
if isinstance(data, dict) and 'features' in data:
    features = data['features']
else:
    features = data  # Assuming input is already a list of features

geojson_data = []

for feature in features:
    try:
        properties = feature.get('properties', {})
        geojson_feature = {
            "id": feature.get("id", ""),
            "type": "Feature",
            "properties": {
                "COMMUNITY_CENTRE": properties.get("COMMUNITY_CENTRE", ""),
                "ADDRESS1": properties.get("ADDRESS1", ""),
                "ADDRESS2": properties.get("ADDRESS2", ""),
                "TOWN": properties.get("TOWN", ""),
                "POSTCODE": properties.get("POSTCODE", ""),
                "PHONE": properties.get("PHONE", ""),
                "UPRN": properties.get("UPRN", ""),
                "OTHER_INFORMATION": properties.get("OTHER_INFORMATION", "")
            },
            "geometry": {
                "type": "Point",
                "coordinates": [
                    float(properties.get("LONGITUDE", 0)),
                    float(properties.get("LATITUDE", 0))
                ]
            }
        }
        geojson_data.append(geojson_feature)
    except Exception as e:
        print(f"Error processing feature {feature.get('id', 'unknown')}: {e}")

# Write the output file
with open('community-centre-GeoJSON.json', 'w') as outfile:
    json.dump(geojson_data, outfile, indent=2)

print(f"Changed {len(geojson_data)} community centers to GeoJSON format")
```

Output file: **Community-centre-GeoJSON.json**

2d. Load the data from 2.c to a new container for part 2.d and 2.e. Find the community centre with the highest latitude where the longitude is less than or equal to the longitude for the community center found in 2.b . Display the community name, latitude, and longitude, and sort the result by latitude in descending order.

First, We upload this output file to a new container. Execution screenshot is shown below:



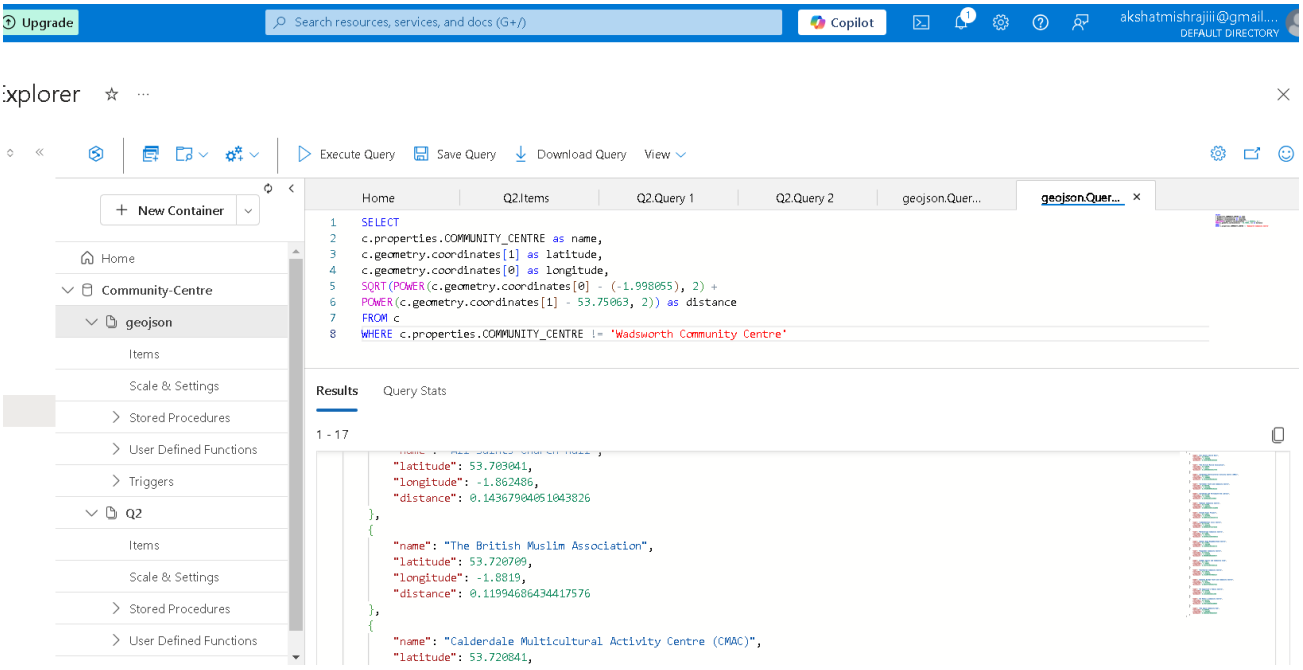

```
SELECT TOP 1
c.properties.COMMUNITY_CENTRE as name, c.geometry.coordinates[1] as latitude,
c.geometry.coordinates[0] as longitude
FROM c
WHERE c.geometry.coordinates[0] <= -1.8819
ORDER BY c.geometry.coordinates[1] DESC
```

The community centre with the highest latitude where the longitude is less than or equal to the longitude of Southowram Community Centre is

[
{
"name": "Wadsworth Community Centre",
"latitude": 53.75063,
"longitude": -1.998055
}
]

2e. Query to find the closest community centre (Euclidean) to the community centre found in 2d. This city will be the password for your final file. (You may use two queries) To find the closest center to this reference center, we need to calculate the Euclidean distance between all other community centers and this one.

```
SELECT
c.properties.COMMUNITY_CENTRE as name, c.geometry.coordinates[1] as latitude,
c.geometry.coordinates[0] as longitude, SQRT(POWER(c.geometry.coordinates[0] - (-1.998055), 2) +
POWER(c.geometry.coordinates[1] - 53.75063, 2)) as distance
FROM c
WHERE c.properties.COMMUNITY_CENTRE != 'Wadsworth Community Centre'
```



```
"name": "Dodnaze Community Centre",
"latitude": 53.742468,
"longitude": -2.003923,
"distance": 0.010052445871526896
```


{
"name": "Dodnaze Community Centre", "latitude": 53.742468,
"longitude": -2.003923,
"distance": 0.010052445871526896
}

The Dodnaze Community Centre community centre is closest to the reference point. Password for next file : Final.txt
Final Password: DODNAZECOMMUNITYCENTRE

We have now unlocked the Final.txt using this password