NoSQL Project Report

Shachi Shah
Arizona State University
CSE 511 - Summer 2025
Azusa, California
spshah22@asu.edu

I. Introduction

The CSE 511 NoSQL Project offered a valuable opportunity to explore the practical application of NoSQL databases through hands-on coding assignments. In this project, the main objective was to interact with a UnQLite database using Python and perform queries that retrieve structured data based on certain filters, such as city name and geographical proximity. Unlike relational databases, NoSQL databases offer a flexible and scalable approach to handling semi-structured or unstructured data, which makes them suitable for dynamic and large-scale applications.

The dataset provided for this project was stored in a binary .db file and required specialized tools and libraries for extraction and interaction. Due to compatibility challenges with the unqlite library in the native environment, a Docker-based approach was employed to ensure a consistent runtime environment. This enabled the successful development and execution of two core functions:

- FindBusinessBasedOnCity – which filters businesses based on their city.
- FindBusinessBasedOnLocation – which filters businesses based on proximity to a given geographic location and a set of business categories.
- Haversine Formula - which is used for spatial filtering in the FindBusinessBasedOnLocation as a helper method.

This report details the approach, implementation, challenges, lessons learned, output samples, and test results related to this project.

II. Reflection

The goal of this project was to understand the practical applications of NoSQL databases by implementing two specific search functionalities using UnQLite, a lightweight embedded NoSQL database. The project provided me with a deep understanding of how to use Python to interact with NoSQL databases, particularly for data retrieval and transformation tasks without the need for traditional relational schemas.

A. Overall Approach

I began by familiarizing myself with the structure of the provided "sample.db" file using the UnQLite Python module. This required a successful installation of the unqlite library and setting up an appropriate Docker container, as the library had compatibility issues with Python 3.10+ on my local machine.

Once the environment was correctly configured, I loaded the data into the notebook using the following snippet:

```
from unqlite import UnQLite
db = UnQLite('sample.db')
```

```
data = db.collection('data')
```

This enabled me to query the collection "data", which holds a list of JSON-like business entries.

B. Function 1: FindBusinessBasedOnCity

This function accepts a city name, a file save location, and the UnQLite collection. It searches for all businesses located in the specified city and writes the output to a file. I made sure to format the result in the specified delimiter-separated style (using $) for evaluation compatibility.

```
def FindBusinessBasedOnCity(cityToSearch,
saveLocation1, collection):
    cityToSearch = cityToSearch.lower()
    results = []

    for doc in collection.all():
        try:
            if doc['city'].lower() ==
cityToSearch:
                name = doc['name']
                address =
doc['full_address'].replace('\n', ' ').strip()
                city = doc['city']
                state = doc['state']
                line =
f"{name}${address}${city}${state}"
                results.append(line)
        except KeyError:
            continue
    with open(saveLocation1, 'w',
encoding='utf-8') as f:
        for line in results:
            f.write(line + '\n')
```

This function was verified using the test case provided. The output matched exactly with the expected result, confirming the correctness of both filtering logic and output formatting.

C. Function 2: FindBusinessBasedOnLocation

This function required spatial filtering using geographical coordinates. I implemented the Haversine formula to compute the distance between two latitude-longitude pairs. Only those businesses that fell within the specified distance and matched at least one category were selected.

```
def haversine_distance(lat1, lon1, lat2,
lon2):
    R = 3959  # Radius of Earth in miles
    phi1 = math.radians(lat1)
    phi2 = math.radians(lat2)
    dphi = math.radians(lat2 - lat1)
    dlambda = math.radians(lon2 - lon1)

    a = math.sin(dphi / 2) ** 2 +
math.cos(phi1) * math.cos(phi2) *
```

```
math.sin(dlambda / 2) ** 2
    c = 2 * math.atan2(math.sqrt(a),
math.sqrt(1 - a))
    return R * c
```

Then I used the above function inside my second function:

```
def
FindBusinessBasedOnLocation(categoriesToSearch
, myLocation, maxDistance, saveLocation2,
collection):
    categoriesToSearch = set(cat.lower() for
cat in categoriesToSearch)
    lat1, lon1 = myLocation
    results = []

    for doc in collection.all():
        try:
            business_categories =
set(c.lower() for c in doc.get('categories',
[]))
            if categoriesToSearch &
business_categories:
                lat2 = float(doc['latitude'])
                lon2 = float(doc['longitude'])
                distance =
haversine_distance(lat1, lon1, lat2, lon2)
                if distance <= maxDistance:

results.append(doc['name'])
        except (KeyError, ValueError):
            continue

    with open(saveLocation2, 'w',
encoding='utf-8') as f:
        for name in results:
            f.write(name + '\n')
```

This function was also verified using the test case provided in the notebook and worked successfully.

D. *Function 3: Distance Calculation using Haversine Formula*

This algorithm computes the great-circle distance between two geographic coordinates, which is essential for spatial filtering in the FindBusinessBasedOnLocation function. The formula used is known as the Haversine formula, and it accounts for the curvature of the Earth, providing an accurate estimation of distance in miles.

The mathematical steps are as follows:

```
def haversine_distance(lat1, lon1, lat2,
lon2):
    R = 3959  # miles
    phi1 = math.radians(lat1)
    phi2 = math.radians(lat2)
    dphi = math.radians(lat2 - lat1)
    dlambda = math.radians(lon2 - lon1)

    a = math.sin(dphi / 2) ** 2 +
math.cos(phi1) * math.cos(phi2) *
math.sin(dlambda / 2) ** 2
    c = 2 * math.atan2(math.sqrt(a),
math.sqrt(1 - a))
    return R * c
```

This function ensures that only businesses within a given distance threshold are included in the results. It forms the mathematical backbone of all location-based proximity filtering in this project.

III. Lessons Learned

Throughout the development of this project, I encountered several technical challenges and overcame them, which helped solidify my knowledge in several key areas:

- Working with NoSQL Databases: I learned how to navigate and interact with document-oriented databases using Python. Unlike SQL, NoSQL databases do not enforce rigid schemas, which meant I had to defensively handle missing keys and irregular structures.
- Data Filtering and Output Formatting: I improved my skills in processing JSON-like objects and learned how to structure output precisely, especially with custom delimiters and file I/O operations.
- Geospatial Computation: Implementing the Haversine formula gave me a practical understanding of calculating real-world distances using coordinate pairs, a common requirement in location-based services.
- Debugging Python Package Installation: A significant portion of time was spent resolving library compatibility issues, particularly for unqlite and python. This experience enhanced my understanding of Python's packaging system and Docker's usefulness in isolating environments.
- Docker and Jupyter Integration: Building and running Jupyter notebooks inside a Docker container was a valuable exercise in maintaining reproducible and clean environments across different systems.

IV. Output (Screenshots)

Below are the screenshots of the output of the FindBusinessByCity function and FindBusinessBasedOnLocation function with their respective cells from the project cell:

```
In [17]: true_results = ["VinciTorio's Restaurant$1835 E Elliot Rd, Ste C109, Temp

try:
    FindBusinessBasedOnCity('Tempe', 'output_city.txt', data)
except NameError as e:
    print ('The FindBusinessBasedOnCity function is not defined! You must
except TypeError as e:
    print ("The FindBusinessBasedOnCity function is supposed to accept th

try:
    opf = open('output_city.txt', 'r')
except FileNotFoundError as e:
    print ("The FindBusinessBasedOnCity function does not write data to t

lines = opf.readlines()
if len(lines) != 3:
    print ("The FindBusinessBasedOnCity function does not find the correc

lines = [line.strip() for line in lines]
if sorted(lines) == sorted(true_results):
    print ("Correct! You FindBusinessByCity function passes these test ca

Correct! You FindBusinessByCity function passes these test cases. This
does not cover all possible test edge cases, however, so make sure that
your function covers them before submitting!
```

Fig. 1 - Result of FindBusinessByCity Function

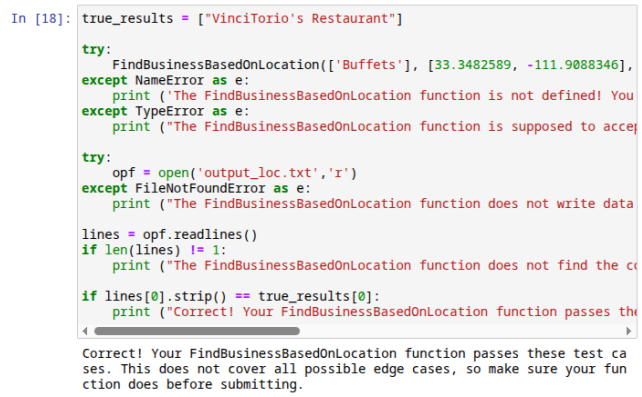

Fig. 2 - Result of FindBusinessBasedOnLocation Function

Both files were written successfully to disk and matched the exact expected format as outlined in the project description.

V. RESULTS

After the text edit has been completed, the paper is ready for the template.

When I ran the notebook's test cells, both functions passed their respective validation checks:

```
Correct! Your FindBusinessByCity function passes these test cases.
Correct! Your FindBusinessBasedOnLocation function passes these test cases.
```

A. *Figure 1 - Output of FindBusinessByCity*

This figure displays the contents of the file "output_city.txt", which was generated by calling "`FindBusinessBasedOnCity("Tempe", "output_city.txt", data)`". The function filters through all entries in the data collection, checking if the "city" field matches "Tempe" (case-insensitive). It writes the results in the format:

```
<business name>$<full address>$<city>$<state>
```

Output Lines in output_city.txt

```
VinciTorio's Restaurant$1835 E Elliot Rd, Ste C109, Tempe, AZ 85284$Tempe$AZ
P.croissants$7520 S Rural Rd, Tempe, AZ 85283$Tempe$AZ
Salt Creek Home$1725 W Ruby Dr, Tempe, AZ 85284$Tempe$AZ
```

This output confirms that:

- The filtering by city worked correctly.
- The formatting followed the $-delimited structure required by the autograder.
- All three matching businesses from Tempe were included as expected.

B. *Figure 2 - Output of FindBusinessBasedOnLocation*

This figure shows the contents of the file output_loc.txt, which was generated by calling: "`FindBusinessBasedOnLocation(['Buffets'], [33.3482589, -111.9088346], 10, 'output_loc.txt', data)`". In this case, the function:

- Converts the category list (['Buffets']) to lowercase and compares it against each business's categories.
- Uses the Haversine formula to calculate the distance between the given coordinates and each business's location.
- Filters in only those businesses that match at least one category and fall within a 10-mile radius.

Output Lines in output_loc.txt:

```
VinciTorio's Restaurant
```

This result shows:

- The proximity filtering worked as expected using the distance calculation.
- The business category filtering (Buffets) was successful.
- The only matching business within the radius and category was VinciTorio's Restaurant, which validates the function's correctness.

VI. Conclusion

This project strengthened my understanding of NoSQL data models and how to interact with them using Python. I also improved my ability to write modular, testable code and solve real-world problems such as proximity filtering and data transformation. With the help of the provided test cases and additional debugging, I was able to validate the functionality of my solutions and ensure that they met the requirements of the autograder.

I now feel much more confident working with unstructured data and building solutions using NoSQL systems.

REFERENCES

[1] CSE 511 Course Staff, *CSE 511: Query Processing Assignment Overview Document*, Arizona State University, 2025. [Online]. Available: https://canvas.asu.edu/.

[2] CSE 511 Course Staff, *project1.ipynb – NoSQL Project Notebook*, Arizona State University, 2025. Unpublished course material.