

Name: Rutu Devang Kansara
USC ID: 8368526607

Project Final Report: Air Quality Monitoring System

Introduction

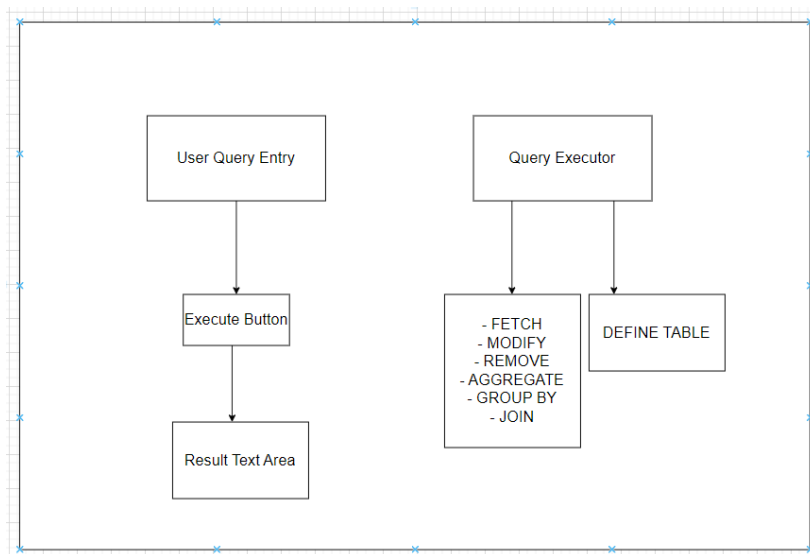
The Air Quality Monitoring System is designed to analyze and provide insights into the Sofia air quality dataset. This report outlines the project's design and implementation, emphasizing key components and functionalities.

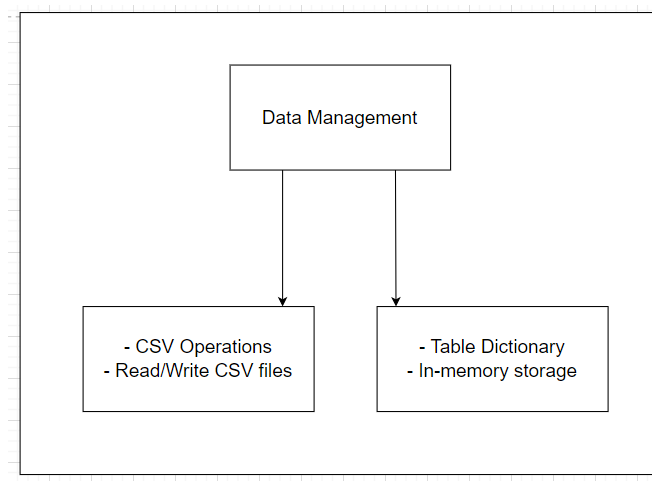
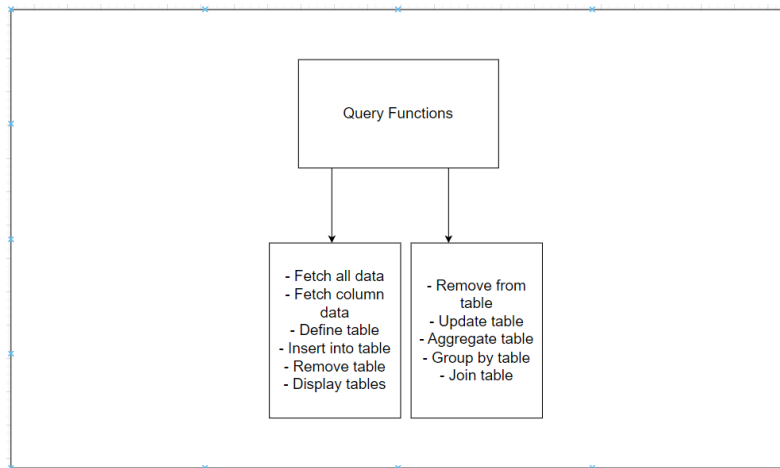
Planned Implementation (From Project Proposal)

The initial plan focused on leveraging the Sofia air quality dataset from Kaggle. The project aimed to implement a relational data model for efficient data storage and advanced querying.

Choosing a relational data model was a strategic decision. This approach fulfilled immediate data needs and laid the groundwork for future expansion. The following sections will delve into the journey from this initial plan to developing the Air Quality Monitoring System.

Architecture Design





Description

The system follows a modular data import, cleaning, exploration, and database design architecture. It includes tables for sensors, measurements, label counts, datetime counts, label ranges, and geographical ranges.

The data flow starts with dataset import, cleaning, exploration, and database design. The architecture ensures flexibility in querying for information.

Implementation

1. Data Cleaning and Exploration

- Data Import:

- I imported the Sofia air quality dataset from 'air_quality_data.csv.'

- Column Management:

- Removed unnecessary columns.

- Date and Time Parsing:

- Parsed 'timestamp' into 'date' and 'time' columns.

- Redundant Column Removal:

- Eliminated redundant columns.

- Missing Data Check:

- Checked for missing data.

- Data Summary:

- Computed summary statistics.

```
data1.describe()
```

	sensor_id	location	lat	lon	temperature	humidity
count	701548.000000	701548.000000	701548.000000	701548.000000	701548.000000	701548.000000
mean	2835.419842	1426.366892	42.679722	23.333739	24.754990	48.348949
std	798.018890	403.446639	0.025568	0.039365	14.013001	20.907247
min	1764.000000	879.000000	42.622000	23.240000	-145.120000	0.000000
25%	2224.000000	1118.000000	42.665000	23.310000	20.630000	34.140000
50%	2323.000000	1172.000000	42.685000	23.332000	24.780000	48.350000
75%	3474.000000	1751.000000	42.694000	23.360000	29.870000	62.940000
max	4661.000000	2343.000000	42.738000	23.419000	61.170000	100.000000

2. Database Design and Query Language Development

- Table Definition:

- Defined tables for sensors, measurements, label counts, datetime counts, label ranges, and geographical ranges.

- Query Language Implementation:

- Developed a simple query language for data retrieval and modification.

3. Functionalities

Data Retrieval

- Fetch All Data:

- Implemented functionality to retrieve all data from specified tables.

- Fetch Specific Columns:

- Retrieved specific columns with and without conditions.

```
15 # function for fetching tables
16 def fetch_all_data(table):
17     return table
18
19 # function for fetching specific columns
20 def fetch_column_data(table, column_names, query_parts, chunk_size=10000):
21     conditions_index = query_parts.index('FOR') if 'FOR' in query_parts else None
22
23     if conditions_index is not None:
24         if len(query_parts) >= conditions_index + 4:
25             column_name = query_parts[conditions_index + 1]
26             condition_sign = query_parts[conditions_index + 2]
27             condition_value = query_parts[conditions_index + 3]
28
29             valid_conditions = ['<', '>', '=']
30             if condition_sign not in valid_conditions:
31                 return "Error: Invalid filter condition. Only <, >, and = are supported."
32
33             try:
34                 condition_value = float(condition_value)
35             except ValueError:
36                 return "Error: Condition value must be a numeric value."
37
```

Database Modification

- Insertion of Values:

- Successfully inserted values into tables.

- Row Removal:

- Removed rows based on specified conditions.

- Value Updates:

- Updated table values based on conditions.

```
84 # function for ADD INTO command
85 def insert_into_table(table_name, values):
86     if table_name not in tables:
87         return f"Error: Table '{table_name}' not found."
88
89     table_df = tables[table_name]
90     # Check if the number of values matches the number of columns
91     if len(values) != len(table_df.columns):
92         return "Error: Number of values doesn't match the number of columns."
93
94     # Create a dictionary with column names as keys and corresponding values
95     row_data = dict(zip(table_df.columns, values))
96
97     # Convert the row data to a DataFrame and append it to the existing table
98     new_row = pd.DataFrame([row_data], columns=table_df.columns)
99
100    # Append the new row to the existing table in chunks
101    table_df = pd.concat([table_df, new_row], ignore_index=True)
102
103    # Update the table in the tables dictionary
104    tables[table_name] = table_df
105
106    # Update the CSV file in chunks
107
108    # Update the CSV file in chunks
109    chunk_size = 10000
110    num_chunks = len(table_df) // chunk_size + 1
111    for i in range(num_chunks):
112        chunk = table_df.iloc[i * chunk_size:(i + 1) * chunk_size]
113        chunk.to_csv(f'../data/{table_name}_chunk_{i}.csv', index=False)
114
115    return f"Values inserted into table '{table_name}'."
116
117 # function for removing specific data from table
118 def remove_from_table(table_name, condition, air_quality_data):
119     if table_name not in tables:
120         return f"Error: Table '{table_name}' not found."
121
122     table_df = tables[table_name]
123
124     try:
125         # Split the condition into column, operator, and value
126         column, operator, value = condition.split()
127
128         # Convert the value to the appropriate type
129         try:
130             chunk = table_df.iloc[i * 10000:(i + 1) * 10000]
131             condition_mask = (chunk[column] == value) if operator == '=' else (
132                 chunk[column] < value if operator == '<' else chunk[column] > value
133             )
134             chunk = chunk[~condition_mask]
135             updated_chunks.append(chunk)
136
137             # Concatenate updated chunks
138             table_df = pd.concat(updated_chunks, ignore_index=True)
139
140             # Update the table in the tables dictionary
141             tables[table_name] = table_df
142
143             # Update the CSV file in chunks
144             chunk_size = 10000
145             num_chunks = len(table_df) // chunk_size + 1
146             for i in range(num_chunks):
147                 chunk = table_df.iloc[i * chunk_size:(i + 1) * chunk_size]
148                 chunk.to_csv(f'../data/{table_name}_chunk_{i}.csv', index=False)
149
150             return f"Rows removed from table '{table_name}' based on condition: '{condition}'."
151         except Exception as e:
152             return f"Error: Invalid condition '{condition}'. {str(e)}"
153
154     except Exception as e:
155         return f"Error: Invalid condition '{condition}'. {str(e)}"
156
157 # function for modification of specific data
158 def update_table(table_name, set_value, where_column, where_operator, where_value):
159     if table_name not in tables:
160         return f"Error: Table '{table_name}' not found."
161
162     table_df = tables[table_name]
163
164     try:
165         # Convert the set value to the appropriate type
166         try:
167             set_value = float(set_value)
168         except ValueError:
169             return f"Error: SET value must be a numeric value."
170
171         # Initialize an empty list to store chunks after update
172         updated_chunks = []
173
174         # Iterate over chunks and apply the update
175         for i in range(len(table_df) // 10000 + 1):
176             chunk = table_df.iloc[i * 10000:(i + 1) * 10000]
177             condition_mask = (chunk[where_column] == float(where_value) if where_operator == '=' else (
178                 chunk[where_column] < float(where_value) if where_operator == '<' else chunk[where_column] > float(where_value) if where_operator == '>'
179             ))
180             chunk = chunk[~condition_mask]
181             updated_chunks.append(chunk)
182
183             # Concatenate updated chunks
184             table_df = pd.concat(updated_chunks, ignore_index=True)
185
186             # Update the table in the tables dictionary
187             tables[table_name] = table_df
188
189             # Update the CSV file in chunks
190             chunk_size = 10000
191             num_chunks = len(table_df) // chunk_size + 1
192             for i in range(num_chunks):
193                 chunk = table_df.iloc[i * chunk_size:(i + 1) * chunk_size]
194                 chunk.to_csv(f'../data/{table_name}_chunk_{i}.csv', index=False)
195
196             return f"Rows updated in table '{table_name}' based on condition: '{where_column} {where_operator} {where_value}'."
197         except Exception as e:
198             return f"Error: Invalid condition '{where_column} {where_operator} {where_value}'. {str(e)}"
199
200     except Exception as e:
201         return f"Error: Invalid condition '{where_column} {where_operator} {where_value}'. {str(e)}"
```

Aggregation and Grouping

- Aggregation Functions:

- Performed basic aggregation functions.

- Grouping Data:

- Grouped data by specified columns.

```
200 def aggregate_table(table_name, aggregation_function, column):
201     if table_name not in tables:
202         return f"Error: Table '{table_name}' not found."
203
204     table_data = tables[table_name]
205
206     try:
207         column_data = table_data[column]
208
209         if aggregation_function == 'COUNT':
210             result = len(column_data)
211         elif aggregation_function == 'SUM':
212             result = sum(column_data)
213         elif aggregation_function == 'MIN':
214             result = min(column_data)
215         elif aggregation_function == 'MAX':
216             result = max(column_data)
217         elif aggregation_function == 'AVG':
218             result = sum(column_data) / len(column_data)
219         else:
220             return f"Error: Invalid aggregation function '{aggregation_function}'. Supported functions are COUNT, SUM, MIN, MAX, AVG."
221
222     except Exception as e:
223         return f"Error: Invalid aggregation function '{aggregation_function}'. Supported functions are COUNT, SUM, MIN, MAX, AVG. {str(e)}"
224
225     return aggregation_function, result
226
227 # function for grouping by along with aggregation of the tables
228 def group_by_table(table_name, aggregation_function, column):
229     if table_name not in tables:
230         return f"Error: Table '{table_name}' not found."
231
232     table_data = tables[table_name]
233
234     try:
235         unique_values = table_data[column].unique()
236
237         grouped_data = pd.DataFrame(columns=[column, aggregation_function])
238
239         for value in unique_values:
240             if aggregation_function == 'COUNT':
241                 result = len(table_data[table_data[column] == value])
242             elif aggregation_function == 'SUM':
243                 result = sum(table_data[table_data[column] == value][column])
244             else:
245                 return f"Error: Invalid aggregation function '{aggregation_function}'. Supported functions are COUNT, SUM, MIN, MAX, AVG."
246
247             grouped_data = grouped_data.append([value, aggregation_function, result], ignore_index=True)
248
249     except Exception as e:
250         return f"Error: Invalid aggregation function '{aggregation_function}'. Supported functions are COUNT, SUM, MIN, MAX, AVG. {str(e)}"
251
252     return grouped_data
```

JOIN Operations

- Table Joins:

- Executed JOIN operations on specified tables.

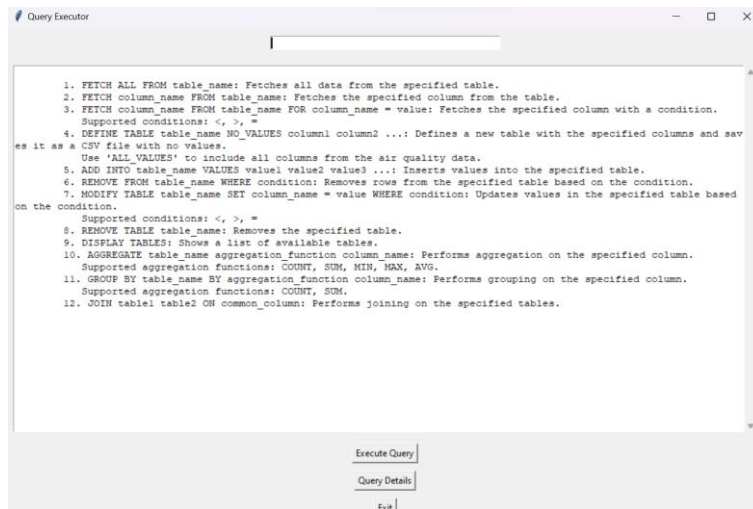
```
251 # function to join tables and chunk them (chunk size can be increased or decreased according to need)
252 def join_tables(table1_name, table2_name, join_column, chunk_size=10000):
253     if table1_name not in tables or table2_name not in tables:
254         return f"Error: One or both tables not found."
255
256     table1 = tables[table1_name]
257     table2 = tables[table2_name]
258
259     try:
260         if join_column not in table1.columns or join_column not in table2.columns:
261             return f"Error: Join column '{join_column}' not found in one or both tables."
262
263         # Perform the JOIN operation
264         result_table = pd.merge(table1, table2, on=join_column)
265
266         # Generate a unique name for the result table
267         result_table_name = f"{table1_name}_{table2_name}_JOIN"
268
269         # Save the result table to the tables dictionary
270         tables[result_table_name] = result_table
271
272         # Save the result table to a CSV file in chunks
273         num_rows = len(result_table)
274
275         result_table_name = f"{table1_name}_{table2_name}_JOIN"
276
277         # Save the result table to the tables dictionary
278         tables[result_table_name] = result_table
279
280         # Save the result table to a CSV file in chunks
281         num_rows = len(result_table)
282         num_chunks = (num_rows // chunk_size) + (1 if num_rows % chunk_size != 0 else 0)
283
284         # Split the result table into chunks and save each chunk
285         for i in range(num_chunks):
286             start_idx = i * chunk_size
287             end_idx = min((i + 1) * chunk_size, num_rows)
288             chunk = result_table.iloc[start_idx:end_idx]
289
290             # Save the chunk to a CSV file
291             chunk.to_csv(f'../data/{result_table_name}_chunk_{i}.csv', index=False)
292
293         return f"Tables '{table1_name}' and '{table2_name}' joined on column '{join_column}'. Result table: '{res"
294
295     except Exception as e:
296         return f"Error: Unable to perform JOIN operation. {str(e)}"
```

Tech Stack

- Python
- Pandas
- Tkinter (for GUI)
- Memory Profiler
- Tabulate (for table display)

Implementation Screenshots

Simple interface for user-friendliness:



DISPLAY TABLES:

Query Executor

DISPLAY TABLES

	Table Name	Columns
0	air_quality_data	sensor_id, location, lat, lon, temperature, humidity, date, time
1	all	sensor_id, location, lat, lon, temperature, humidity, date, time
2	b	sensor_id, lat, lon
3	d	sensor_id, temperature
4	demographics	sensor_id, location, lat, lon
5	demographical	sensor_id, temperature, humidity
6	e	lat
7	reading	sensor_id, temperature, humidity, date, time
8	reading1	sensor_id, lat, lon

FETCH ALL FROM demographics1

Query Executor

FETCH ALL FROM demographics

sensor_id	temperature	humidity
2266	23.46	62.48
2292	23.06	59.46
3096	26.53	44.38
3428	20.34	38.28
3472	26.31	46.37
1952	22.66	56.55
1846	23.87	50.76
3512	24.92	55.53
2228	26.29	45.7
3438	24.62	57.97
1954	26.26	44.46
3620	25.27	58.19
3436	26.57	45.01
3092	24.31	0
2036	24.01	51.41
1962	25.95	46.26
3474	26.7	45.18
2232	25.92	50.87
2607	24.23	56.28
2224	24.99	49.53
3738	27.8	42.03
3102	25.99	43.24
2040	25.64	46.25
2216	26.79	42.78
3432	24.79	52.6
2294	24.12	45.73
2230	27.68	45.6

Execute Query
Query Details
Exit

FETCH sensor_id FROM demographics1

Query Executor

FETCH sensor_id FROM demographics1

sensor_id
2266
2292
3096
3428
3472
1952
1846
3512
2228
3438
1954
3620
3436
3092
2036
1962
3474
2232
2607
2224
3738
3102
2040
2216
3432
2294
2230

Execute Query

Query Details

Exit

Fetch sensor_id, humidity from demographics1

Query Executor

FETCH sensor_id, humidity FROM demographics1

sensor_id	humidity
2266	62.48
2292	59.46
3096	44.35
3428	38.20
3472	46.37
1952	56.55
1846	50.76
3512	55.53
2228	45.7
3438	57.97
1954	44.46
3620	50.19
3436	45.01
3092	0
2036	51.41
1962	46.26
3474	49.15
2232	50.07
2607	56.28
2224	49.53
3738	42.03
3102	43.24
2040	46.25
2216	42.75
3432	52.6
2294	45.73
2230	45.6

Execute Query

Query Details

Exit

FETCH sensor_id, humidity FROM demographics1 FOR sensor_id = 2266

Query Executor

CH sensor_id, humidity FROM demographics1 FOR sensor_id = 2266

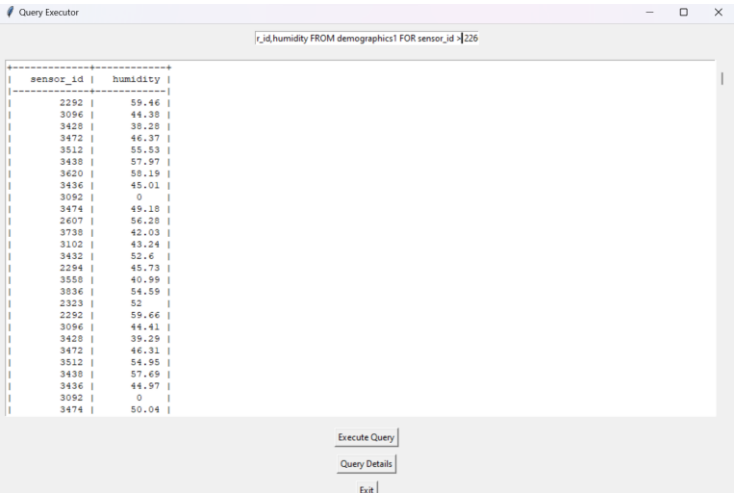
sensor_id	humidity
2266	62.48
2266	63.4
2266	63.22
2266	63.07
2266	63.37
2266	63.57
2266	62.36
2266	62.78
2266	63.79
2266	62.17
2266	61.95
2266	61.1
2266	62.33
2266	63.81
2266	64.41
2266	64.7
2266	64.74
2266	69.15
2266	67.28
2266	65.82
2266	65.48
2266	66.63
2266	67.56
2266	68.5
2266	67.31
2266	68.57
2266	68.72

Execute Query

Query Details

Exit

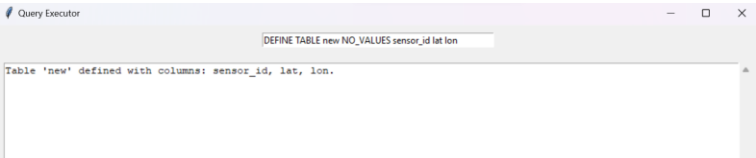
FETCH sensor_id,humidity FROM demographics1 FOR sensor_id > 2266



Query Executor window showing the result of the query: `sensor_id,humidity FROM demographics1 FOR sensor_id > 2266`. The results are displayed in a table with two columns: sensor_id and humidity.

sensor_id	humidity
2262	59.46
3096	44.39
3420	39.29
3472	46.37
3512	55.53
3438	57.97
3420	59.19
3436	45.01
3092	0
3474	49.18
2607	56.29
3739	42.03
3102	43.24
3432	52.6
2294	45.73
3558	40.99
3836	54.59
2323	52
2292	59.46
3096	44.41
3428	39.29
3472	46.31
3512	54.95
3438	57.69
3436	44.97
3092	0
3474	50.04

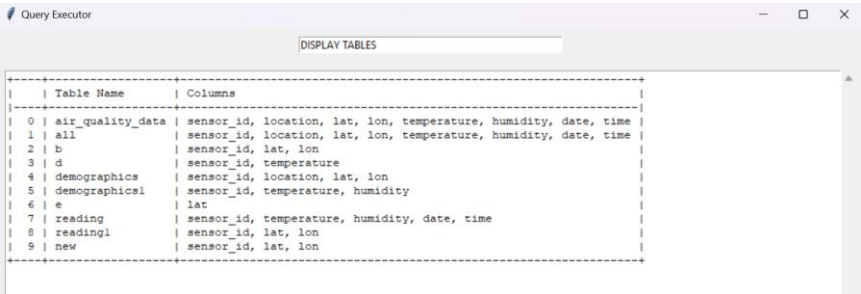
DEFINE TABLE new NO_VALUES sensor_id lat lon



Query Executor window showing the result of the query: `DEFINE TABLE new NO_VALUES sensor_id lat lon`. The result is a message indicating that the table 'new' has been defined with columns: sensor_id, lat, lon.

Table Name	Columns
new	sensor_id, lat, lon

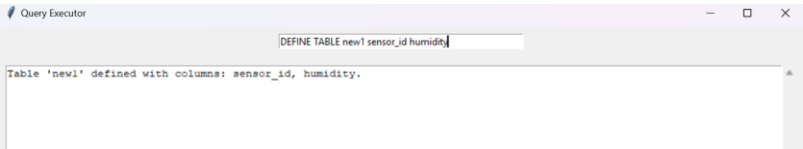
New table created



Query Executor window showing the result of the query: `DISPLAY TABLES`. The result is a table listing the tables in the database.

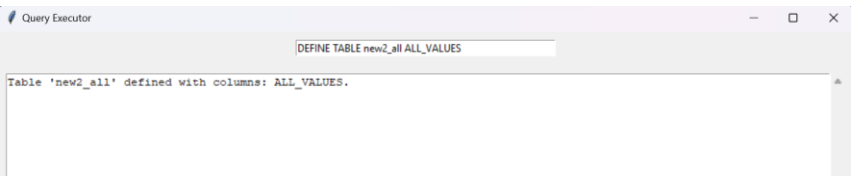
Table Name	Columns
air_quality_data	sensor_id, location, lat, lon, temperature, humidity, date, time
all	sensor_id, location, lat, lon, temperature, humidity, date, time
b	sensor_id, lat, lon
d	sensor_id, temperature
demographics	sensor_id, location, lat, lon
demographics1	sensor_id, temperature, humidity
e	lat
reading	sensor_id, temperature, humidity, date, time
reading1	sensor_id, lat, lon
new	sensor_id, lat, lon

DEFINE TABLE new1 sensor_id humidity



Query Executor window showing the result of the query: `DEFINE TABLE new1 sensor_id humidity`. The result is a message indicating that the table 'new1' has been defined with columns: sensor_id, humidity.

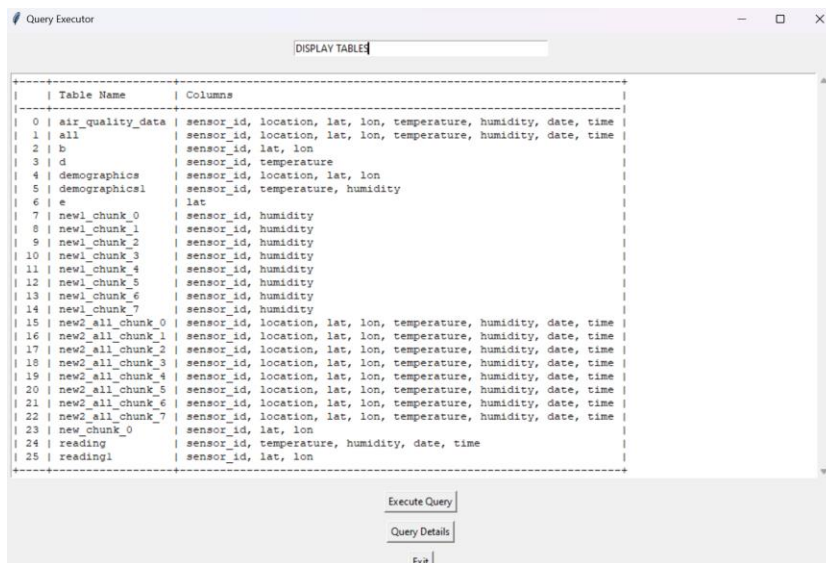
Table Name	Columns
new1	sensor_id, humidity



Query Executor window showing the result of the query: `DEFINE TABLE new2_all ALL_VALUES`. The result is a message indicating that the table 'new2_all' has been defined with columns: ALL_VALUES.

Table Name	Columns
new2_all	ALL_VALUES

Chunking



Query Executor

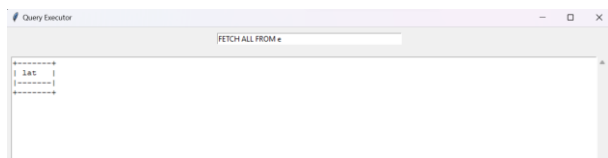
DISPLAY TABLES

	Table Name	Columns
0	air_quality_data	sensor_id, location, lat, lon, temperature, humidity, date, time
1	all	sensor_id, location, lat, lon, temperature, humidity, date, time
2	b	sensor_id, lat, lon
3	d	sensor_id, temperature
4	demographics	sensor_id, location, lat, lon
5	demographics1	sensor_id, temperature, humidity
6	e	lat
7	new1_chunk_0	sensor_id, humidity
8	new1_chunk_1	sensor_id, humidity
9	new1_chunk_2	sensor_id, humidity
10	new1_chunk_3	sensor_id, humidity
11	new1_chunk_4	sensor_id, humidity
12	new1_chunk_5	sensor_id, humidity
13	new1_chunk_6	sensor_id, humidity
14	new1_chunk_7	sensor_id, humidity
15	new2_all_chunk_0	sensor_id, location, lat, lon, temperature, humidity, date, time
16	new2_all_chunk_1	sensor_id, location, lat, lon, temperature, humidity, date, time
17	new2_all_chunk_2	sensor_id, location, lat, lon, temperature, humidity, date, time
18	new2_all_chunk_3	sensor_id, location, lat, lon, temperature, humidity, date, time
19	new2_all_chunk_4	sensor_id, location, lat, lon, temperature, humidity, date, time
20	new2_all_chunk_5	sensor_id, location, lat, lon, temperature, humidity, date, time
21	new2_all_chunk_6	sensor_id, location, lat, lon, temperature, humidity, date, time
22	new2_all_chunk_7	sensor_id, location, lat, lon, temperature, humidity, date, time
23	new_chunk_0	sensor_id, lat, lon
24	reading	sensor_id, temperature, humidity, date, time
25	reading1	sensor_id, lat, lon

Execute Query
Query Details
Exit

ADD INTO:

Empty table e:



Query Executor

FETCH ALL FROM e

lat

ADD INTO e VALUES 0.1

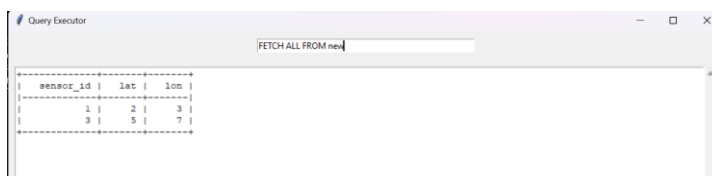


Query Executor

ADD INTO e VALUES 0.1

Values inserted into table 'e'.

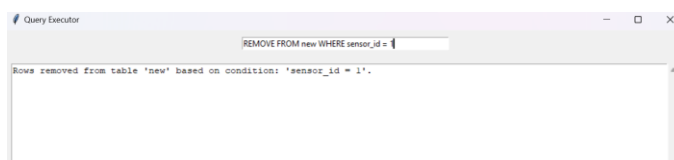
REMOVE FROM



Query Executor

FETCH ALL FROM new

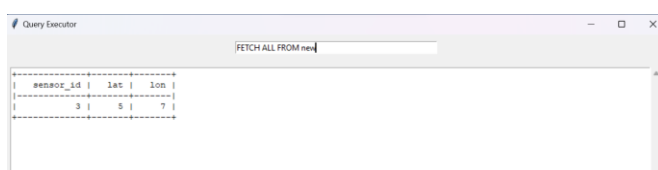
sensor_id	lat	lon
1	2	3
3	5	7



Query Executor

REMOVE FROM new WHERE sensor_id = 1

Rows removed from table 'new' based on condition: 'sensor_id = 1'.



Query Executor

FETCH ALL FROM new

sensor_id	lat	lon
3	5	7

REMOVE TABLE

Query Executor

DISPLAY TABLES

	Table Name	Columns
0	air_quality_data	sensor_id, location, lat, lon, temperature, humidity, date, time
1	all	sensor_id, location, lat, lon, temperature, humidity, date, time
2	b	sensor_id, lat, lon
3	d	sensor_id, temperature
4	demographics	sensor_id, location, lat, lon
5	demographics1	sensor_id, temperature, humidity
6	e	lat
7	e_chunk_0	lat
8	new	sensor_id, lat, lon
9	new1_chunk_0	sensor_id, humidity
10	new1_chunk_1	sensor_id, humidity
11	new1_chunk_2	sensor_id, humidity
12	new1_chunk_3	sensor_id, humidity
13	new1_chunk_4	sensor_id, humidity
14	new1_chunk_5	sensor_id, humidity
15	new1_chunk_6	sensor_id, humidity
16	new1_chunk_7	sensor_id, humidity
17	new2_all_chunk_0	sensor_id, location, lat, lon, temperature, humidity, date, time
18	new2_all_chunk_1	sensor_id, location, lat, lon, temperature, humidity, date, time
19	new2_all_chunk_2	sensor_id, location, lat, lon, temperature, humidity, date, time
20	new2_all_chunk_3	sensor_id, location, lat, lon, temperature, humidity, date, time
21	new2_all_chunk_4	sensor_id, location, lat, lon, temperature, humidity, date, time
22	new2_all_chunk_5	sensor_id, location, lat, lon, temperature, humidity, date, time
23	new2_all_chunk_6	sensor_id, location, lat, lon, temperature, humidity, date, time
24	new2_all_chunk_7	sensor_id, location, lat, lon, temperature, humidity, date, time
25	reading	sensor_id, temperature, humidity, date, time
26	reading1	sensor_id, lat, lon

Execute Query

Query Details

Exit

Query Executor

REMOVE TABLE b

Table 'b' removed.

Query Executor

DISPLAY TABLE

	Table Name	Columns
0	air_quality_data	sensor_id, location, lat, lon, temperature, humidity, date, time
1	all	sensor_id, location, lat, lon, temperature, humidity, date, time
2	d	sensor_id, temperature
3	demographics	sensor_id, location, lat, lon
4	demographics1	sensor_id, temperature, humidity
5	e	lat
6	e_chunk_0	lat
7	new	sensor_id, lat, lon
8	new1_chunk_0	sensor_id, humidity
9	new1_chunk_1	sensor_id, humidity
10	new1_chunk_2	sensor_id, humidity
11	new1_chunk_3	sensor_id, humidity
12	new1_chunk_4	sensor_id, humidity
13	new1_chunk_5	sensor_id, humidity
14	new1_chunk_6	sensor_id, humidity
15	new1_chunk_7	sensor_id, humidity
16	new2_all_chunk_0	sensor_id, location, lat, lon, temperature, humidity, date, time
17	new2_all_chunk_1	sensor_id, location, lat, lon, temperature, humidity, date, time
18	new2_all_chunk_2	sensor_id, location, lat, lon, temperature, humidity, date, time
19	new2_all_chunk_3	sensor_id, location, lat, lon, temperature, humidity, date, time
20	new2_all_chunk_4	sensor_id, location, lat, lon, temperature, humidity, date, time
21	new2_all_chunk_5	sensor_id, location, lat, lon, temperature, humidity, date, time
22	new2_all_chunk_6	sensor_id, location, lat, lon, temperature, humidity, date, time
23	new2_all_chunk_7	sensor_id, location, lat, lon, temperature, humidity, date, time
24	reading	sensor_id, temperature, humidity, date, time
25	reading1	sensor_id, lat, lon

Execute Query

Query Details

Exit

MODIFY TABLE

Query Executor

FETCH ALL FROM new

sensor_id	lat	lon
3	5	7

Query Executor

MODIFY TABLE new SET sensor_id = 1 WHERE sensor_id = 3

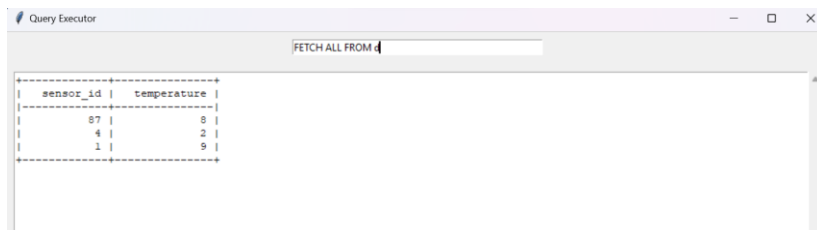
Table 'new' updated.

Query Executor

FETCH ALL FROM new

sensor_id	lat	lon
1	5	7

AGGREGATE

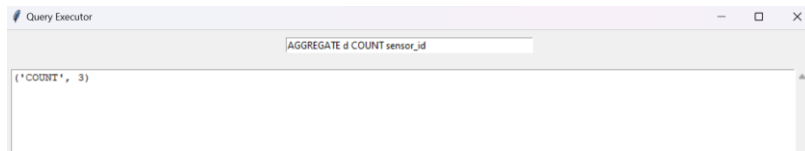


Query Executor

FETCH ALL FROM d

sensor_id	temperature
87	8
4	2
1	9

COUNT

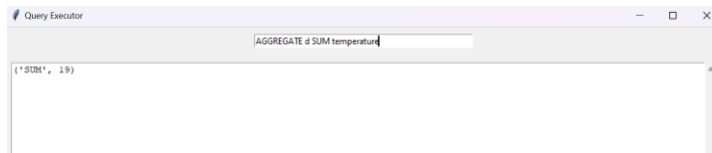


Query Executor

AGGREGATE d COUNT sensor_id

(*COUNT*, 3)

SUM

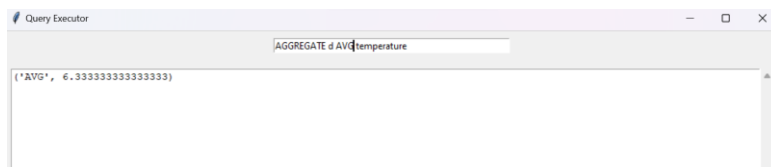


Query Executor

AGGREGATE d SUM temperature

(*SUM*, 19)

AVG

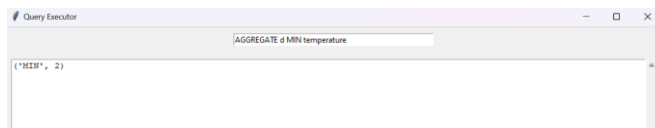


Query Executor

AGGREGATE d AVG temperature

(*AVG*, 6.333333333333333)

MIN

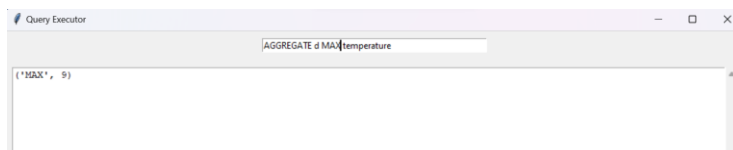


Query Executor

AGGREGATE d MIN temperature

(*MIN*, 2)

MAX

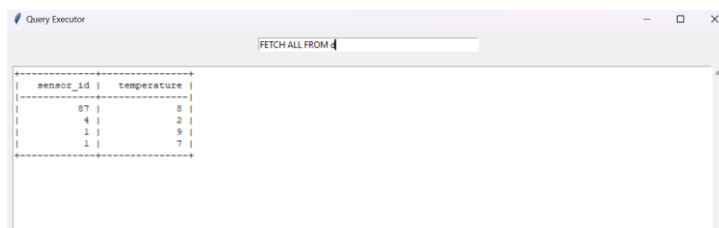


Query Executor

AGGREGATE d MAX temperature

(*MAX*, 9)

GROUP BY



Query Executor

FETCH ALL FROM d

sensor_id	temperature
87	8
4	2
1	9
1	7

Query Executor

GROUP BY d BY COUNT sensor_id

sensor_id	COUNT
87	1
4	1
1	2

JOIN

Query Executor

FETCH ALL FROM demographics

sensor_id	temperature	humidity
2266	23.46	62.48
2292	23.06	59.46
3096	26.53	44.38
3428	28.34	38.28
3472	26.31	46.37
1952	22.66	56.55
1846	23.87	50.76
3512	24.92	55.53
2228	26.29	45.7
3438	24.62	57.97
1954	26.26	44.46
3620	25.27	58.19
3436	26.57	45.01
3092	24.31	0
2036	24.01	51.41
1962	25.95	46.26
3474	26.7	49.18
2232	25.92	50.87
2607	24.23	56.28
2224	24.99	49.53
3738	27.8	42.03
3102	25.99	43.24
2040	25.64	46.25
2216	26.79	42.78
3432	24.79	52.6
2294	24.12	45.73
2230	27.68	45.6

Execute Query

Query Details

Exit

Query Executor

FETCH ALL FROM reading1

sensor_id	lat	lon
2266	1	1
2292	2	3
3096	4	9
3428	5	8
3472	6	3
1952	7	0
1846	8	4
3512	24.92	55.53
2228	26.29	45.7
3438	24.62	57.97
1954	26.26	44.46
3620	25.27	58.19
3436	26.57	45.01
3092	24.31	0
2036	24.01	51.41
1962	25.95	46.26
3474	26.7	49.18
2232	25.92	50.87
2607	24.23	56.28
2224	24.99	49.53
3738	27.8	42.03
3102	25.99	43.24
2040	25.64	46.25
2216	26.79	42.78
3432	24.79	52.6
2294	24.12	45.73
2230	27.68	45.6

Execute Query

Query Details

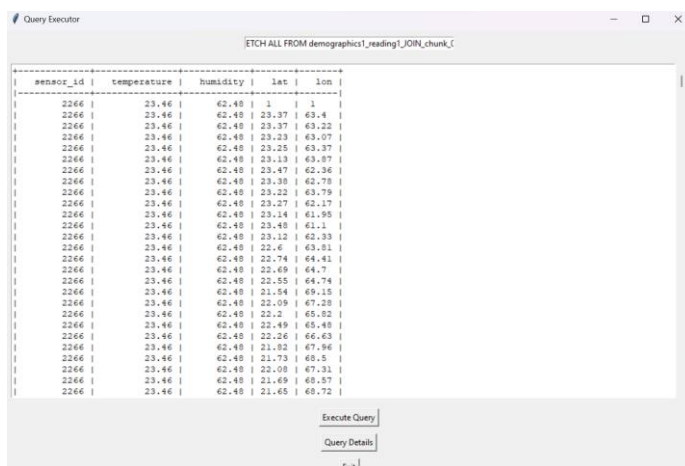
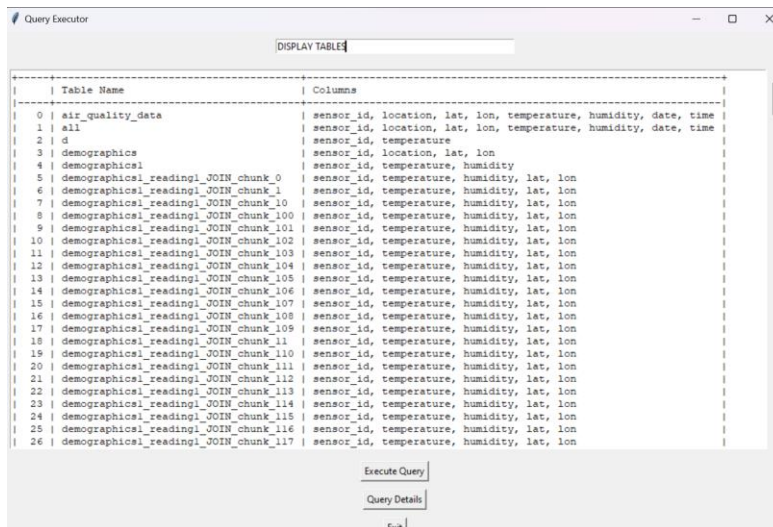
Exit

Query Executor

JOIN demographics1 reading1 ON sensor_id

Tables 'demographics1' and 'reading1' joined on column 'sensor_id'. Result table: 'demographics1_reading1_JOIN'.

Chunks created after JOIN operation



Learning Outcomes

1. Technical Skills

Python Proficiency: Significantly enhanced skills in Python programming, utilizing it for diverse tasks within the project. This involved not only scripting logic but also effective use of Pandas for data manipulation and analysis.

GUI Development: Acquired hands-on experience in creating Graphical User Interfaces (GUIs) using Tkinter. This practical knowledge contributes to the ability to develop interactive and user-friendly applications.

2. Database Design

Relational Database Concepts: Developed a solid understanding of relational database design principles. This encompasses structuring data in a way that ensures efficiency, consistency, and meaningful relationships.

Query Languages: Gained proficiency in query languages for databases, enabling effective communication with the underlying data structures. This skill set is fundamental in extracting, updating, and managing data within the project's database.

Challenges Faced

- Interface Design:

- Not having used tkinter before, crafting a user-friendly Tkinter interface posed challenges, but I navigated them by prioritizing simplicity. The goal was to create an interface anyone could understand and interact with.

- Data Handling:

- Initially ran into some errors while managing complexities in parsing data and time for accurate analysis. Dealing with date and time intricacies in the data required meticulous handling. I tackled this challenge to ensure precise analysis.

Conclusion

To wrap up, the Air Quality Monitoring System project has achieved its goals, providing a solid foundation for further exploration of Sofia's air quality data. This marks a significant step in understanding and enhancing air quality monitoring capabilities.

Future Scope

1. Visualization

- Enhancing the visual representation of air quality data remains a key focus. By implementing more engaging graphical elements, the system can provide users with a clearer understanding of air quality trends. Additionally, exploring real-time data streaming capabilities will offer up-to-the-minute insights.

2. User Interface Enhancement

- Refine and enhance the website/interface design.
- Striving for an even more intuitive design will provide a seamless user experience.

3. Advanced Analytics

- The system's capabilities can be extended by delving into advanced analytics. Exploring machine learning models for predictive analysis opens the door to anticipating air quality changes. This proactive approach can be invaluable in decision-making and environmental planning.

Google Drive Submission Link

https://drive.google.com/drive/folders/1yTM69XO8SIthFuYAtX4zEFRk4igIHYS5?usp=drive_link

I appreciate your help and guidance throughout this course and project.

Thank you!