Practical Project Part 4 Report

Name: David Shaju Menachery  
Course: CST8002 – Programming Language Research Project  
Professor: Tyler DeLay  
Assignment: Practical Project Part 4 – Multi-Column Filter  
Date: August 3, 2025

# Evidence of Learning - Multi-Column Search Feature

The following code demonstrates the implementation of a multi-column search/filtering feature as the novel addition to this project.

Sample method in RecordManager (business/manager.py):

# File: coumarin\_data\_manager.py

2 def search\_records(self, conditions):

3 def record\_matches(record):

4 for cond in conditions:

5 field = cond['field']

6 operator = cond['operator']

7 value = cond['value']

8 try:

9 record\_value = getattr(record, field)

10 record\_value = float(record\_value)

11 value = float(value)

12 except:

13 pass

14 if operator == "==" and not record\_value == value:

15 return False

16 elif operator == "!=" and not record\_value != value:

17 return False

18 elif operator == "<" and not record\_value < value:

19 return False

20 elif operator == ">" and not record\_value > value:

21 return False

22 elif operator == "<=" and not record\_value <= value:

23 return False

24 elif operator == ">=" and not record\_value >= value:

25 return False

26 return True

27 return list(filter(record\_matches, self.records))

* **Line 2–3:** Defines a nested function to evaluate if each record matches all conditions (functional programming).
* **Line 9:** Uses getattr() to dynamically access fields — demonstrating dynamic attribute handling.
* **Line 10–11:** Applies float() casting for numeric comparisons — shows exception-safe type conversion.
* **Lines 14–25:** Implements all major comparison operators (==, !=, <, >, <=, >=) — demonstrates conditional logic.
* **Line 27:** Uses filter() with a custom predicate — example of functional filtering in Python.

# Program Changes

The main changes occurred in the following files:

1. business/manager.py – added ‘search\_records()’ method  
   2. presentation/main.py – added menu option '8' for searching records by multiple fields  
   3. Updated ‘display\_menu()’ function to include the new option

# Program Demonstration via Screenshots

Below is a simulated example. Replace with your own screenshots:

A screen shot of a computer

AI-generated content may be incorrect.

[**Screenshot1** : User selecting Option 8]

A screenshot of a computer program

AI-generated content may be incorrect.

[**Screenshot2**: Entering field names and values]

A screenshot of a computer

AI-generated content may be incorrect.

[**Screenshot3**: Filtered output displayed]

# Source Code Commenting Example

Excerpt from manager.py demonstrating comments:

### business/manager.py

class RecordManager:

def \_\_init\_\_(self, records=[]):

"""Initialize with an optional list of records."""

self.records = records

def add(self, record):

"""Add a new record to the list."""

self.records.append(record)

def delete(self, index):

"""Delete the record at the given index, if valid."""

if 0 <= index < len(self.records):

del self.records[index]

def update(self, index, record):

"""Update the record at the given index with a new one."""

if 0 <= index < len(self.records):

self.records[index] = record

def get\_all(self):

"""Return the list of all records."""

return self.records

def get(self, index):

"""Get a single record by index, or None if out of range."""

return self.records[index] if 0 <= index < len(self.records) else None

def reload(self, path, reader):

"""Reload records from a file using a reader function."""

self.records = reader(path)

def search\_records(self, conditions):

"""

Filter records based on multiple user-defined conditions.

Each condition is a dictionary with:

- 'field': attribute of the Record object (e.g., 'origin')

- 'operator': comparison operator as a string (==, !=, <, >, <=, >=)

- 'value': the value to compare against

"""

def record\_matches(record):

for cond in conditions:

field = cond['field']

operator = cond['operator']

value = cond['value']

try:

record\_value = getattr(record, field)

except AttributeError:

print(f"Invalid field: {field}")

return False

# Try numeric comparison

try:

record\_value = float(record\_value)

value = float(value)

except ValueError:

pass # Fall back to string comparison

if operator == "==" and not record\_value == value:

return False

elif operator == "!=" and not record\_value != value:

return False

elif operator == "<" and not record\_value < value:

return False

elif operator == ">" and not record\_value > value:

return False

elif operator == "<=" and not record\_value <= value:

return False

elif operator == ">=" and not record\_value >= value:

return False

return True

return list(filter(record\_matches, self.records))

# References

[1] Python Software Foundation, “Built-in Functions – filter().” [Online]. Available: <https://docs.python.org/3/library/functions.html>. [Accessed: Jul. 19, 2025].

[2] GeeksForGeeks, “Python | filter() function,” [Online]. Available: <https://www.geeksforgeeks.org/filter-in-python/>. [Accessed: Jul. 18, 2025].

[3] Stack Overflow, “How to filter a list of dictionaries based on multiple keys?” [Online]. Available: <https://stackoverflow.com/questions/8653516/filtering-a-list-of-dictionaries>. [Accessed: Aug. 3, 2025].