Name: Randive Laxmi Rajaram

Program for Rule Engine with AST

!pip install ast

import ast

import operator

import sqlite3

import json

from datetime import datetime

# Node class to represent the AST structure

class Node:

def \_\_init\_\_(self, node\_type, value=None, left=None, right=None):

self.type = node\_type # "operator" or "operand"

self.value = value # For operand nodes, the condition (e.g., "age > 30")

self.left = left # Left child node

self.right = right # Right child node

def \_\_repr\_\_(self):

if self.type == 'operand':

return f"Operand({self.value})"

elif self.type == 'operator':

return f"Operator({self.value}, {self.left}, {self.right})"

def create\_rule(rule\_string):

"""Parses a rule string and converts it into an AST."""

try:

tree = ast.parse(rule\_string, mode='eval') # Parse rule string to AST

return tree

except Exception as e:

raise ValueError(f"Invalid rule string: {e}")

def combine\_rules(rules):

"""Combine multiple rules into a single AST using OR."""

combined\_tree = None

for rule in rules:

rule\_tree = create\_rule(rule)

if combined\_tree is None:

combined\_tree = rule\_tree

else:

combined\_tree = ast.BoolOp(

op=ast.Or(),

values=[combined\_tree.body, rule\_tree.body]

)

return combined\_tree

def evaluate\_node(node, data):

"""Recursively evaluates the AST node against the data."""

ops = {

ast.Gt: operator.gt,

ast.Lt: operator.lt,

ast.Eq: operator.eq,

ast.NotEq: operator.ne,

ast.GtE: operator.ge,

ast.LtE: operator.le,

}

if isinstance(node, ast.BoolOp):

if isinstance(node.op, ast.And):

return all(evaluate\_node(value, data) for value in node.values)

elif isinstance(node.op, ast.Or):

return any(evaluate\_node(value, data) for value in node.values)

elif isinstance(node, ast.Compare):

left = evaluate\_node(node.left, data)

right = evaluate\_node(node.comparators[0], data)

op = ops[type(node.ops[0])]

return op(left, right)

elif isinstance(node, ast.Name):

return data.get(node.id)

elif isinstance(node, ast.Constant):

return node.value

def evaluate\_rule(rule\_ast, data):

"""Evaluates the rule against the provided data."""

try:

return evaluate\_node(rule\_ast.body, data)

except Exception as e:

raise ValueError(f"Error evaluating rule: {e}")

# Set up a SQLite database for rules and metadata

def init\_db():

conn = sqlite3.connect('rule\_engine.db')

cursor = conn.cursor()

# Create tables for rules and metadata

cursor.execute('''

CREATE TABLE IF NOT EXISTS Rules (

id INTEGER PRIMARY KEY AUTOINCREMENT,

rule\_string TEXT NOT NULL,

description TEXT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

)

''')

conn.commit()

conn.close()

# Initialize the database

init\_db()

# Function to save a rule to the database

def save\_rule\_to\_db(rule\_string, description):

conn = sqlite3.connect('rule\_engine.db')

cursor = conn.cursor()

cursor.execute('''

INSERT INTO Rules (rule\_string, description)

VALUES (?, ?)

''', (rule\_string, description))

conn.commit()

conn.close()

# Function to fetch all rules from the database

def fetch\_rules\_from\_db():

conn = sqlite3.connect('rule\_engine.db')

cursor = conn.cursor()

cursor.execute('SELECT \* FROM Rules')

rules = cursor.fetchall()

conn.close()

return rules

# Function to delete a rule by its ID

def delete\_rule\_by\_id(rule\_id):

conn = sqlite3.connect('rule\_engine.db')

cursor = conn.cursor()

cursor.execute('DELETE FROM Rules WHERE id = ?', (rule\_id,))

conn.commit()

conn.close()

# Function to clear all rules from the database and reset the auto-increment ID

def clear\_all\_rules():

conn = sqlite3.connect('rule\_engine.db')

cursor = conn.cursor()

# Delete all records from the table

cursor.execute('DELETE FROM Rules')

conn.commit()

# Reset the auto-increment value back to 1 for the Rules table

cursor.execute('DELETE FROM sqlite\_sequence WHERE name = "Rules"')

conn.commit()

conn.close()

print("All rules cleared and rule IDs reset to start from 1.")

def input\_rule():

"""Takes user input to create a rule."""

rule\_string = input("Enter the rule (e.g., age > 30 and department == 'Sales'): ")

description = input("Enter a description for the rule: ")

save\_rule\_to\_db(rule\_string, description)

print("Rule saved successfully!")

def input\_data():

"""Takes user input to provide data for rule evaluation."""

age = int(input("Enter age: "))

department = input("Enter department: ")

salary = int(input("Enter salary: "))

experience = int(input("Enter years of experience: "))

return {"age": age, "department": department, "salary": salary, "experience": experience}

def run\_rule\_engine():

"""Main function to run the rule engine."""

while True:

choice = input("\nChoose an option: \n1. Create a rule \n2. Evaluate a rule \n3. Delete a rule \n4.

Clear all rules \n5. Exit\n")

if choice == '1':

# Create a new rule

input\_rule()

elif choice == '2':

# Evaluate a rule

rules = fetch\_rules\_from\_db()

if not rules:

print("No rules found. Please create a rule first.")

continue

# Show available rules

print("Available rules:")

for rule in rules:

print(f"ID: {rule[0]}, Rule: {rule[1]}, Description: {rule[2]}")

rule\_id = int(input("Enter the ID of the rule you want to evaluate: "))

selected\_rule = None

for rule in rules:

if rule[0] == rule\_id:

selected\_rule = rule[1]

break

if not selected\_rule:

print("Invalid rule ID.")

continue

# Input data for evaluation

data = input\_data()

try:

rule\_ast = create\_rule(selected\_rule)

result = evaluate\_rule(rule\_ast, data)

print(f"Evaluation result: {result}")

except ValueError as e:

print(f"Error: {e}")

elif choice == '3':

# Delete a specific rule

rules = fetch\_rules\_from\_db()

if not rules:

print("No rules found.")

continue

print("Available rules:")

for rule in rules:

print(f"ID: {rule[0]}, Rule: {rule[1]}, Description: {rule[2]}")

rule\_id = int(input("Enter the ID of the rule you want to delete: "))

delete\_rule\_by\_id(rule\_id)

print(f"Rule with ID {rule\_id} deleted successfully.")

elif choice == '4':

# Clear all rules and reset IDs

clear\_all\_rules()

elif choice == '5':

# Exit the program

print("Exiting the rule engine.")

break

else:

print("Invalid choice. Please choose again.")

# Run the rule engine

run\_rule\_engine()