Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

Дисциплина «Операционные системы»

**Лабораторная работа 3**

**Базы знаний и онтологии**

Выполнил

Студент группы ИВТАСбд-42

Сулейманов М.З.

Проверил:

преподаватель кафедры «ВТ»

Хайрулин И.Д.

Ульяновск

2024

# Постановка задачи

Необходимо разработать программу на языке python, которая реализует предложенное вариантом задание. Разработать систему, которая контролирует условия хранения на складе: Система должна поддерживать заданные температуру и влажность для хранения продукции.

**Цель работы**

Целью данной лабораторной работы является разработка и реализация системы кормления животных на ферме, которая принимает решения на основе времени, веса животных и температуры. Для хранения и обработки данных используется база данных **Neo4j**, позволяющая строить взаимосвязи между состояниями системы и действиями.

# Реализация

# Для начала нужно создать Graph DBMS в Neo4j, после его создания нужно его активировать и зайти в него.

# 

# Создание базы данных Neo4j

* Удаление всех существующих данных

|  |
| --- |
| MATCH (n) DETACH DELETE n; |

Эта команда удаляет все данные из базы. MATCH (n) находит все узлы, а DETACH DELETE n удаляет их и все их связи. Это полезно для очистки базы данных перед загрузкой новых данных.

* Создание узлов для состояний температуры

|  |
| --- |
| CREATE (temp1:Condition {value: 'Cold', type: 'Temperature'}),  (temp2:Condition {value: 'Optimal', type: 'Temperature'}),  (temp3:Condition {value: 'Hot', type: 'Temperature'}) |

Здесь создаются узлы для состояния температурного диапозона: низкое, оптимальное, высокая. Каждый узел имеет тип Condition с двумя свойствами: value (значение состояния) и type (тип состояния).

* Создание узлов для состояния влажности

|  |
| --- |
| CREATE (hum1:Condition {value: 'Dry', type: 'Humidity'}),  (hum2:Condition {value: 'Optimal', type: 'Humidity'}),  (hum3:Condition {value: 'Humid', type: 'Humidity'}) |

Создаются узлы, представляющие состояния веса животных: легкий, оптимальный и тяжелый. Аналогично, каждый узел имеет тип Condition с соответствующими свойствами.

* Связывание состояний с действиями

|  |
| --- |
| WITH temp1, temp2, temp3, hum1, hum2, hum3  MERGE (temp1)-[:ACTION {name: 'IncreaseTemperature'}]->(temp2)  MERGE (temp3)-[:ACTION {name: 'DecreaseTemperature'}]->(temp2)  MERGE (temp2)-[:ACTION {name: 'MaintainTemperature'}]->(temp2)  MERGE (hum1)-[:ACTION {name: 'IncreaseHumidity'}]->(hum2)  MERGE (hum3)-[:ACTION {name: 'DecreaseHumidity'}]->(hum2)  MERGE (hum2)-[:ACTION {name: 'MaintainHumidity'}]->(hum2) |

Здесь происходит связывание узлов состояния времени с узлами состояния температуры и влажности через связи типа ACTION.

Используется команда MERGE, которая создаёт связь, если она не существует, или игнорирует, если она уже есть, избегая дублирования.

# Логика управления

# Для определения текущих условий температуры, веса, времени реализованы методы фаззификации, которые переводят численные значения температуры веса и времени в лингвистические термины:

# Температура:

# Cold: T < 15 C

# Optimal: 15 C <= T <= 25 C

# Hot: T > 25 C

# влажность:

# Dry H < 30

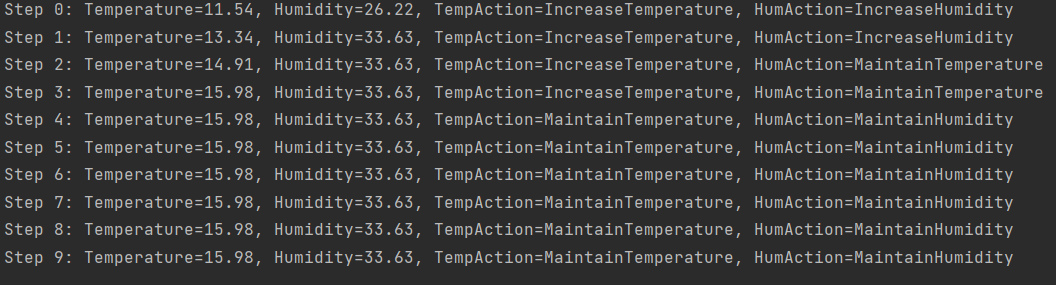
# Optimal: 30 <= H <= 60

# Humid: H > 40

# На основе фаззифицированных условий система формирует запрос к базе Neo4j для определения подходящего действия:

# 

# Тестирование



# Вывод

1. **Эффективность Neo4j:** Использование графовой базы данных Neo4j позволяет наглядно моделировать и обрабатывать связи между состояниями и действиями.
2. **Модульность кода:** Логика системы четко разделена на компоненты (фаззификация, запросы, симуляция), что упрощает поддержку и дальнейшее расширение функционала.
3. **Гибкость:** Система легко адаптируется к изменению логики или добавлению новых состояний, действий и условий.

Работа показала, что использование графовых баз данных эффективно для задач, связанных с управлением состояниями, действиями и их взаимосвязями.

# Приложение. Код main.py

|  |
| --- |
| from neo4j import GraphDatabase  import numpy as np  class StorageControlSystem:  def \_\_init\_\_(self, uri, user, password): # Инициализация драйвера для подключения к Neo4j  self.driver = GraphDatabase.driver(uri, auth=(user, password))  def close(self):  self.driver.close()  # Метод для получения действия на основе текущих условий  def get\_action(self, temperature, humidity):  # Фаззификация температуры и влажности  temp\_condition = self.fuzzify\_temperature(temperature)  hum\_condition = self.fuzzify\_humidity(humidity)  # Запрос к Neo4j  query = (  "MATCH (temp:Condition {value: $temp\_condition})-[temp\_action:ACTION]->(temp\_next:Condition), "  "(hum:Condition {value: $hum\_condition})-[hum\_action:ACTION]->(hum\_next:Condition) "  "RETURN temp\_action.name AS temp\_action, hum\_action.name AS hum\_action "  "LIMIT 1"  )  with self.driver.session() as session:  result = session.run(query, temp\_condition=temp\_condition, hum\_condition=hum\_condition)  actions = result.single()  if actions:  return actions["temp\_action"], actions["hum\_action"]  return None, None  # Фаззификация температуры  def fuzzify\_temperature(self, temperature):  if temperature < 15:  return "Cold"  elif 15 <= temperature <= 25:  return "Optimal"  else:  return "Hot"  # Фаззификация влажности  def fuzzify\_humidity(self, humidity):  if humidity < 30:  return "Dry"  elif 30 <= humidity <= 60:  return "Optimal"  else:  return "Humid"  # Метод симуляции  def simulate(self, initial\_temperature, initial\_humidity, steps=10):  temperature = initial\_temperature  humidity = initial\_humidity  for step in range(steps):  # Получаем действия из Neo4j  temp\_action, hum\_action = self.get\_action(temperature, humidity)  # Адаптация температуры  if temp\_action == "IncreaseTemperature":  temperature += np.random.uniform(1, 3)  elif temp\_action == "DecreaseTemperature":  temperature -= np.random.uniform(1, 3)  # Адаптация влажности  if hum\_action == "IncreaseHumidity":  humidity += np.random.uniform(5, 10)  elif hum\_action == "DecreaseHumidity":  humidity -= np.random.uniform(5, 10)  # Поддержание значения в пределах допустимого диапазона  temperature = max(0, round(temperature, 2))  humidity = max(0, round(humidity, 2))  # Вывод текущего состояния  print(f"Step {step}: Temperature={temperature}, Humidity={humidity}, TempAction={temp\_action}, HumAction={hum\_action}")  # Запуск симуляции  if \_\_name\_\_ == "\_\_main\_\_":  uri = "neo4j+s://aaf5a1c2.databases.neo4j.io"  user = "neo4j"  password = "XR7syNnjzjKhXVccWJyko\_CzFVlYZDsD\_KmTswf3bkk"  system = StorageControlSystem(uri, user, password)  try:  system.simulate(initial\_temperature=10, initial\_humidity=20, steps=10)  finally:  system.close() |

**Код Cypher**

|  |
| --- |
| // Удаляем все существующие данные  MATCH (n) DETACH DELETE n;  // Создаем узлы для состояний температуры  CREATE (temp1:Condition {value: 'Cold', type: 'Temperature'}),  (temp2:Condition {value: 'Optimal', type: 'Temperature'}),  (temp3:Condition {value: 'Hot', type: 'Temperature'})  // Создаем узлы для состояний влажности  CREATE (hum1:Condition {value: 'Dry', type: 'Humidity'}),  (hum2:Condition {value: 'Optimal', type: 'Humidity'}),  (hum3:Condition {value: 'Humid', type: 'Humidity'})  // Создаем действия для управления температурой и влажностью  WITH temp1, temp2, temp3, hum1, hum2, hum3  MERGE (temp1)-[:ACTION {name: 'IncreaseTemperature'}]->(temp2)  MERGE (temp3)-[:ACTION {name: 'DecreaseTemperature'}]->(temp2)  MERGE (temp2)-[:ACTION {name: 'MaintainTemperature'}]->(temp2)  MERGE (hum1)-[:ACTION {name: 'IncreaseHumidity'}]->(hum2)  MERGE (hum3)-[:ACTION {name: 'DecreaseHumidity'}]->(hum2)  MERGE (hum2)-[:ACTION {name: 'MaintainHumidity'}]->(hum2)  // Дополнительные связи для сложных сценариев  MERGE (temp1)-[:ACTION {name: 'IncreaseTemperature'}]->(temp3)  MERGE (hum1)-[:ACTION {name: 'IncreaseHumidity'}]->(hum3) |