

| МИНОБРНАУКИ РОССИИ |
| --- |
| Федеральное государственное бюджетное образовательное учреждениевысшего образования"МИРЭА - Российский технологический университет"РТУ МИРЭА |

**Институт** Информационных Технологий

**Кафедра** Вычислительной Техники

**ПРАКТИЧЕСКАЯ РАБОТА**

**по дисциплине**

**«Системный анализ данных СППР»**

Студент группы:ИКБО-42-23 \_\_\_Голев С.С.\_\_\_ *(Ф. И.О.студента)*

Преподаватель \_\_Железняк Л.М.\_\_ *(Ф.И.О. преподавателя)*

|  |  |  |
| --- | --- | --- |

# СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_c8yb8aayhuci)

[1 ОНТОЛОГИЯ 4](#_dja0po3asjlb)

[1.1 Цель и задачи практической работы 4](#_r4w1mdcptpe)

[1.2 Постановка задачи 5](#)

[1.3 Проектирование базы знаний 6](#)

[1.4 Разработка базы знаний в инструменте Protege 7](#)

[1.5 Разработка базы знаний в кодовом виде 12](#_407w44x5s44e)

[1.6 Результат работы 13](#)

[ЗАКЛЮЧЕНИЕ 14](#_58lr19plam6o)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 15](#_lf774iyqz02r)

[ПРИЛОЖЕНИЯ 16](#_sllqjcbbegb)

# 

# ВВЕДЕНИЕ

Онтология представляет собой формализованное описание предметной области, включающее основные понятия и связи между ними. Она служит основой для представления знаний в интеллектуальных системах и обеспечивает их семантическую совместимость. Основной принцип построения онтологии заключается в четком определении терминов и их иерархических отношений. Онтологический подход позволяет структурировать информацию и повысить эффективность обработки данных. При разработке онтологии важно соблюдать принципы целостности, непротиворечивости и расширяемости. Таким образом, онтология становится важным инструментом для организации знаний и поддержки интеллектуального анализа данных.

# 

# 1 ОНТОЛОГИЯ

# 1.1 Цель и задачи практической работы

Цель работы заключается в формировании умений по применению методов системного анализа данных при проектировании базы знаний для заданной предметной области.

Задачи работы включают:

1. Определить предметную область, для которой будет разрабатываться база знаний;
2. Определение состава объектов, их свойств и взаимосвязей;
3. Построение концептуальной модели базы знаний;
4. Формирование связей различных видов между объектами;
5. Перенос базы знаний в инструмент Protege;
6. Перенос базы знаний в кодовый формат.

# 

# 1.2 Постановка задачи

В рамках практической работы необходимо выбрать предметную область и выполнить её формализацию в виде базы знаний. Для этого требуется определить множество объектов и их атрибутов, выделить связи между объектами. На основе полученных данных следует построить концептуальную модель базы знаний и спроектировать её структуру, обеспечивающую возможность дальнейшего применения для решения задач системного анализа.

Построить данную модель в инструменте Protege и программном виде на языке Python.

# 

# 1.3 Проектирование базы знаний

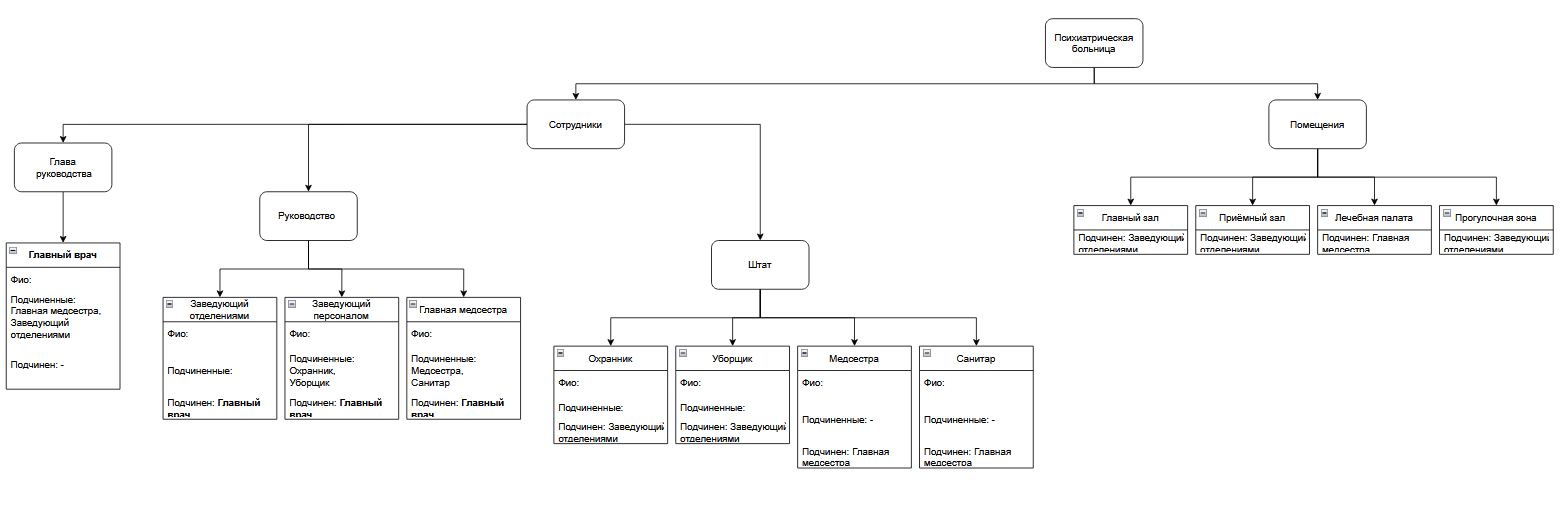
В рамках практической части работы была выбрана предметная область — психиатрическая больница, которая является главным, абстрактным классом. Для данной области выполнено построение базы знаний, отражающей структуру управления, штат сотрудников и используемые помещения.

Основными классами базы знаний являются:

* Руководство;
* Сотрудники;
* Помещения.

Каждый объект обладает набором характеристик, включая ФИО, подчинённых и подчинение, что позволяет формализовать как вертикальные связи управления, так и горизонтальные взаимосвязи между элементами системы.

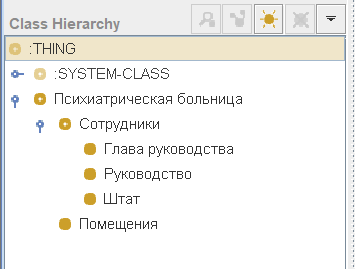
Построенная иерархическая схема отражает организационную структуру психиатрической больницы: от руководства к сотрудникам и помещениям (Рис.1).



**Рисунок 1 – Схема базы знаний**

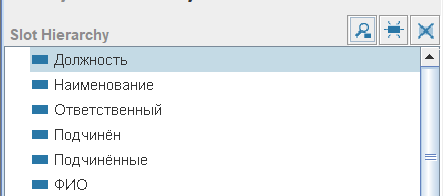
# 1.4 Разработка базы знаний в инструменте Protege

Перенесем построенную базу знаний в инструмент Protege. Для начала построим иерархию классов, отображающую общую структуру.



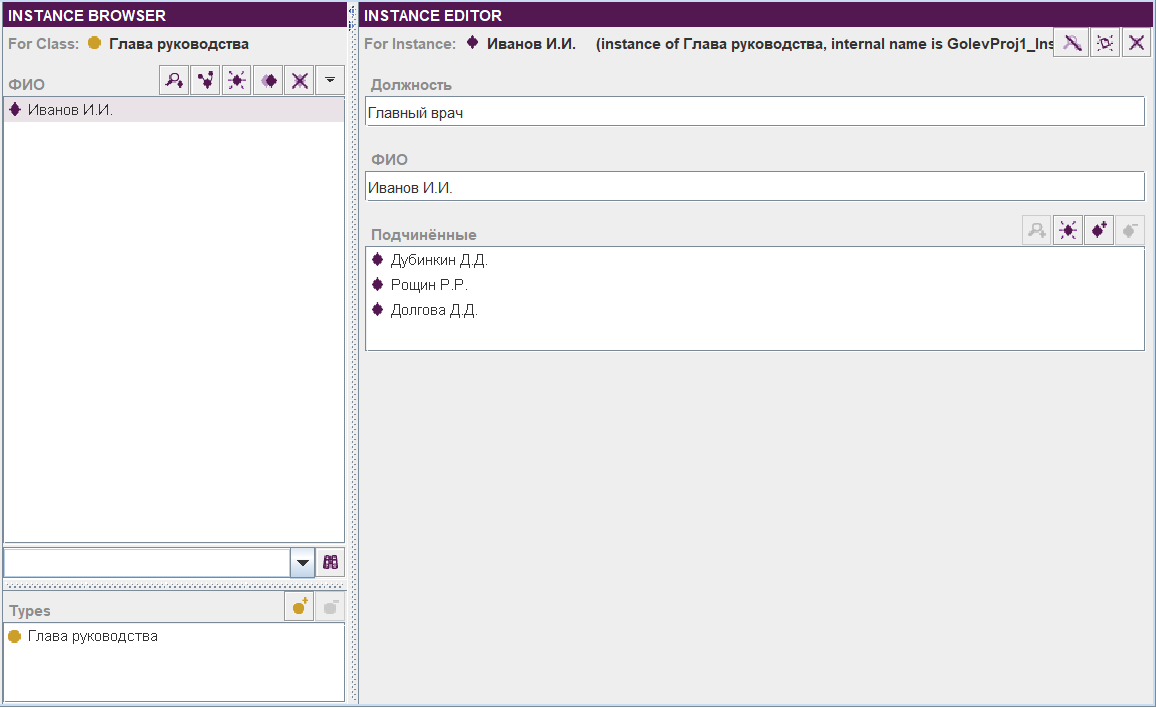
**Рисунок 2 – Иерархия базы знаний**

Также напишем поля, которые будут присвоены к определенным класса, дабы потом реализовать объекты этих классов.

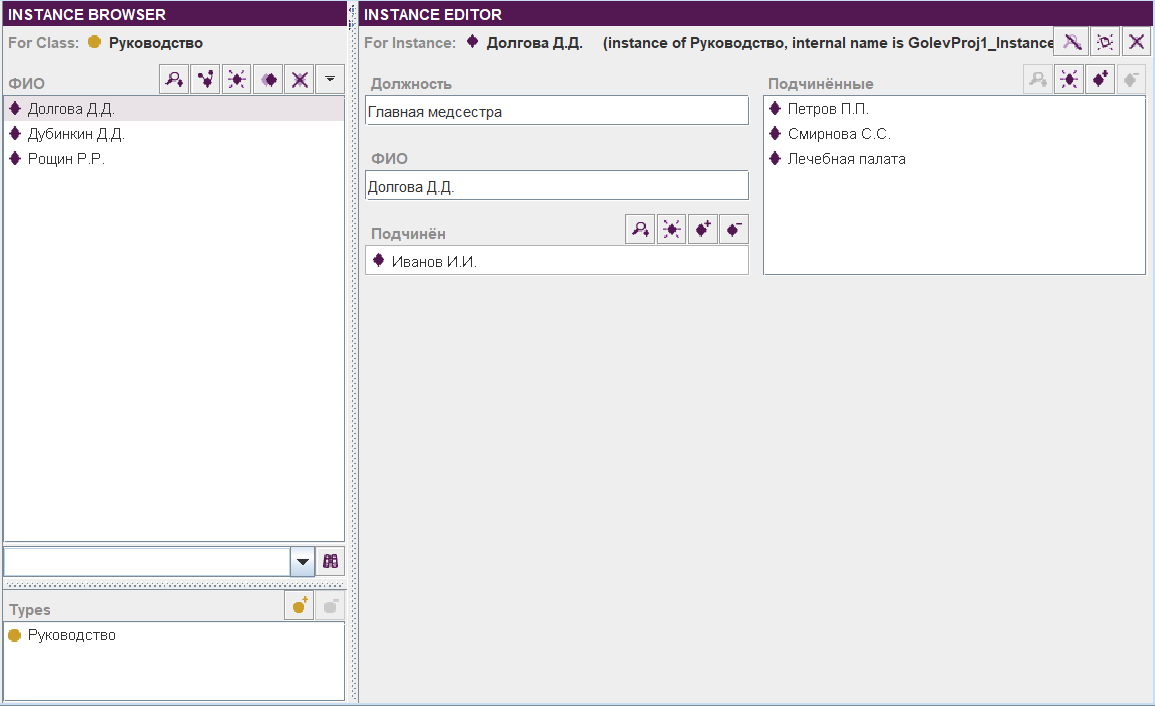


**Рисунок 3 – Поля используемые в базе знаний**

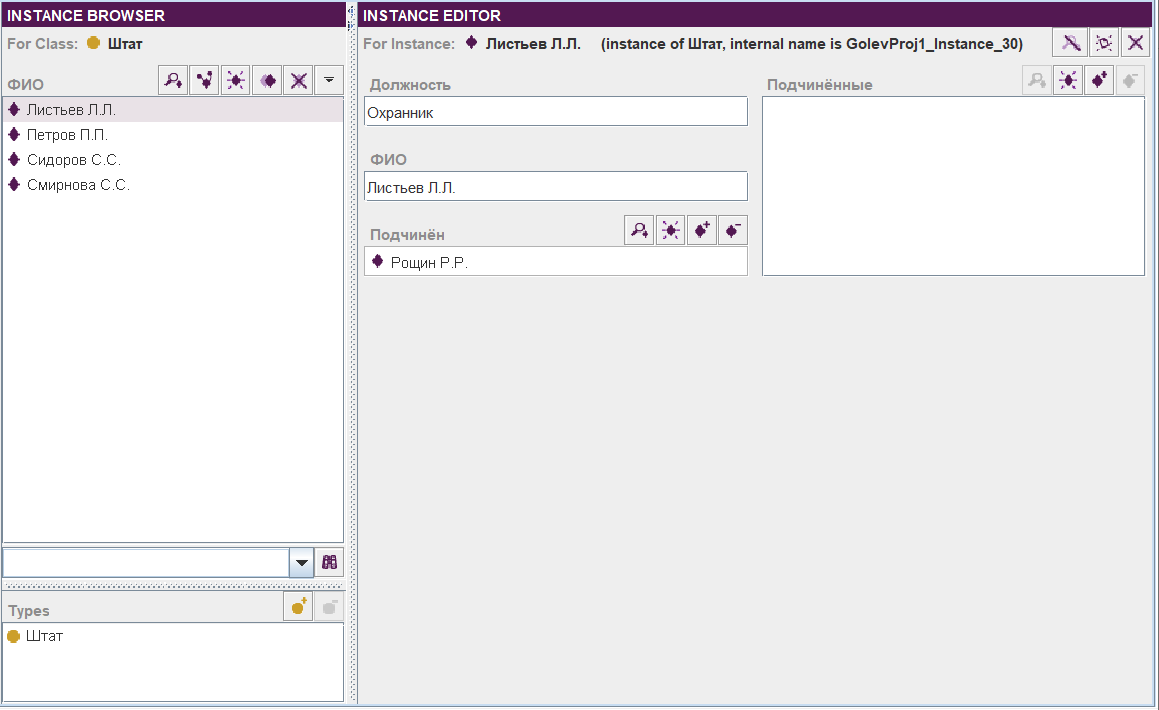
Создадим объекты всех классов данной базы знаний и заполним поля, дабы каждый объект был персонализирован.



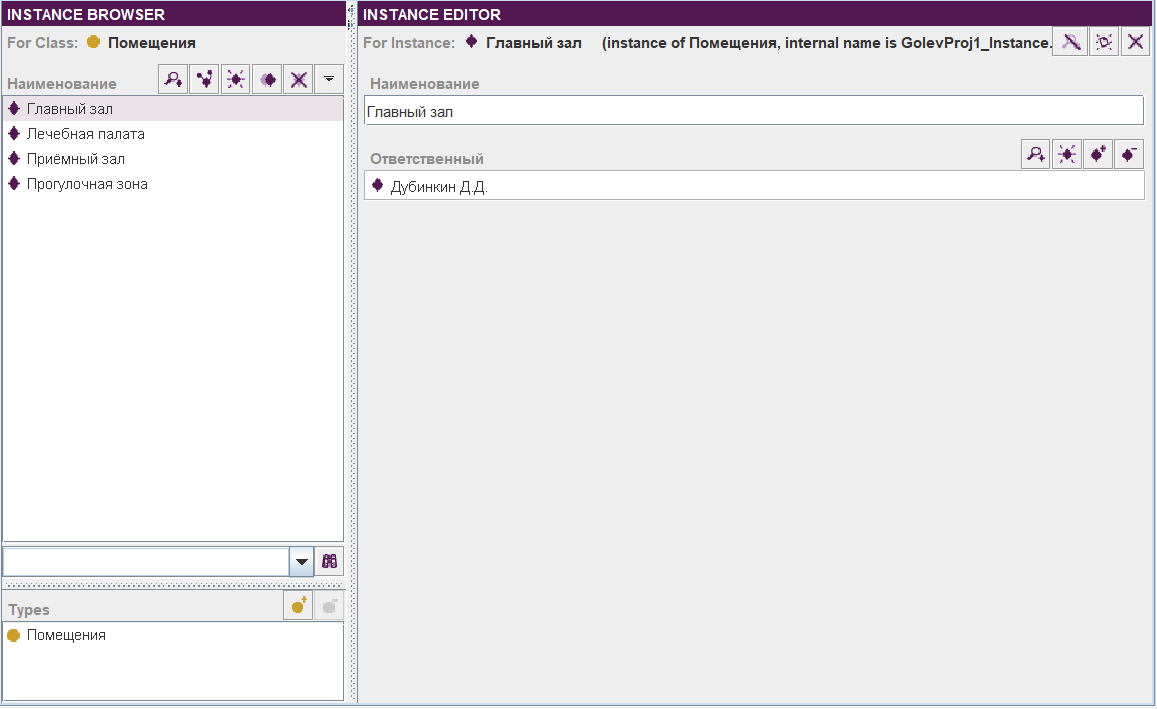
**Рисунок 4 – Объект класса “Глава руководства”**



**Рисунок 5 – Объекты класса “Руководство”**



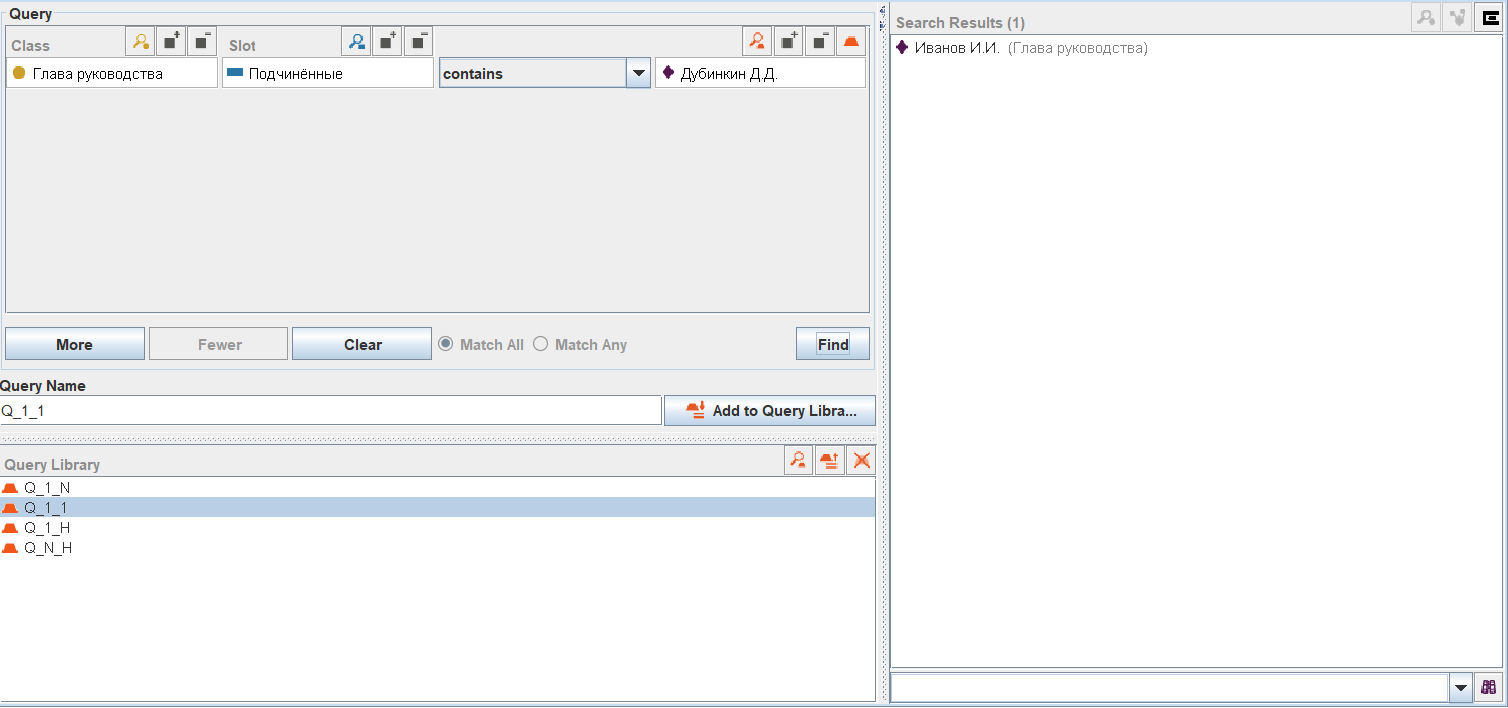
**Рисунок 6 – Объекты класса “Штат”**



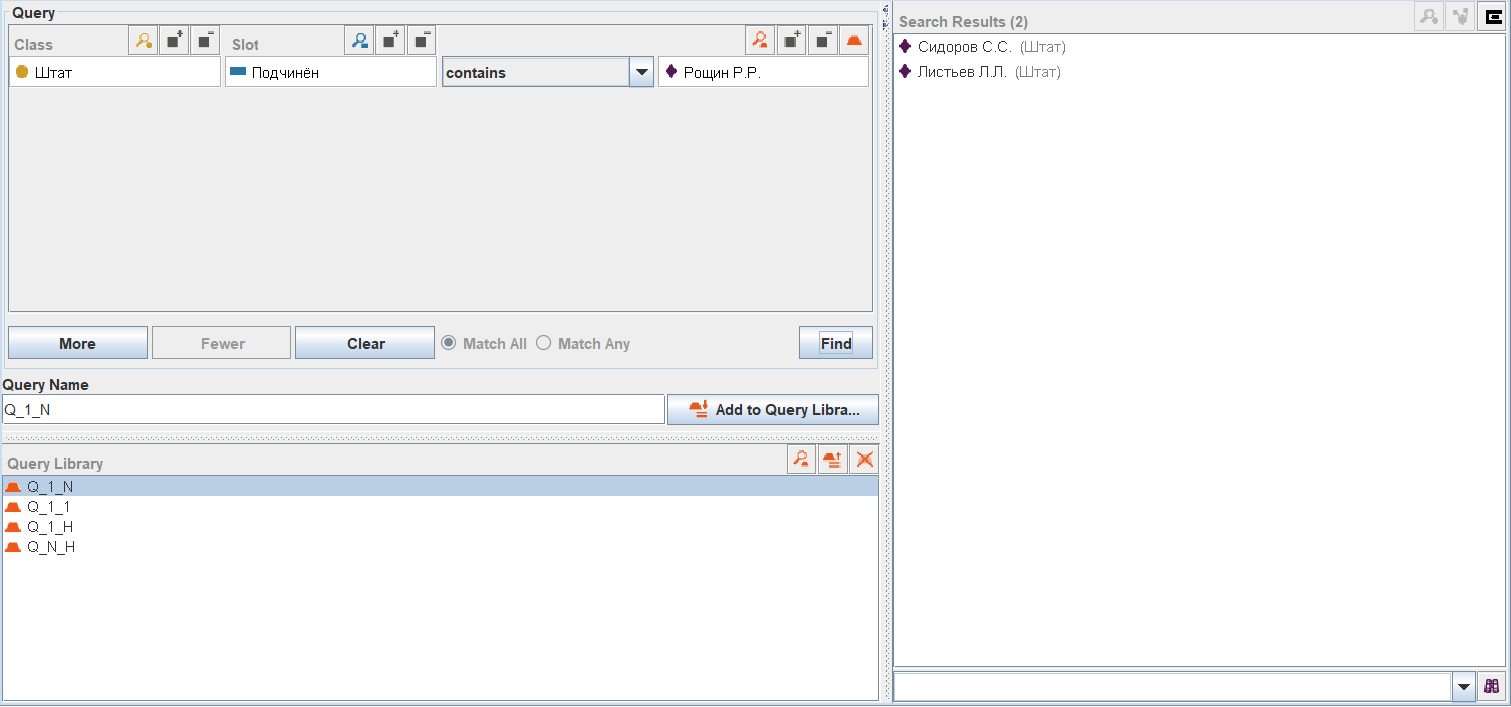
**Рисунок 7 – Объекты класса “Помещения”**

Напишем запросы, с помощью которых можно будет посмотреть связи между объектами различных классов, реализуем связи:

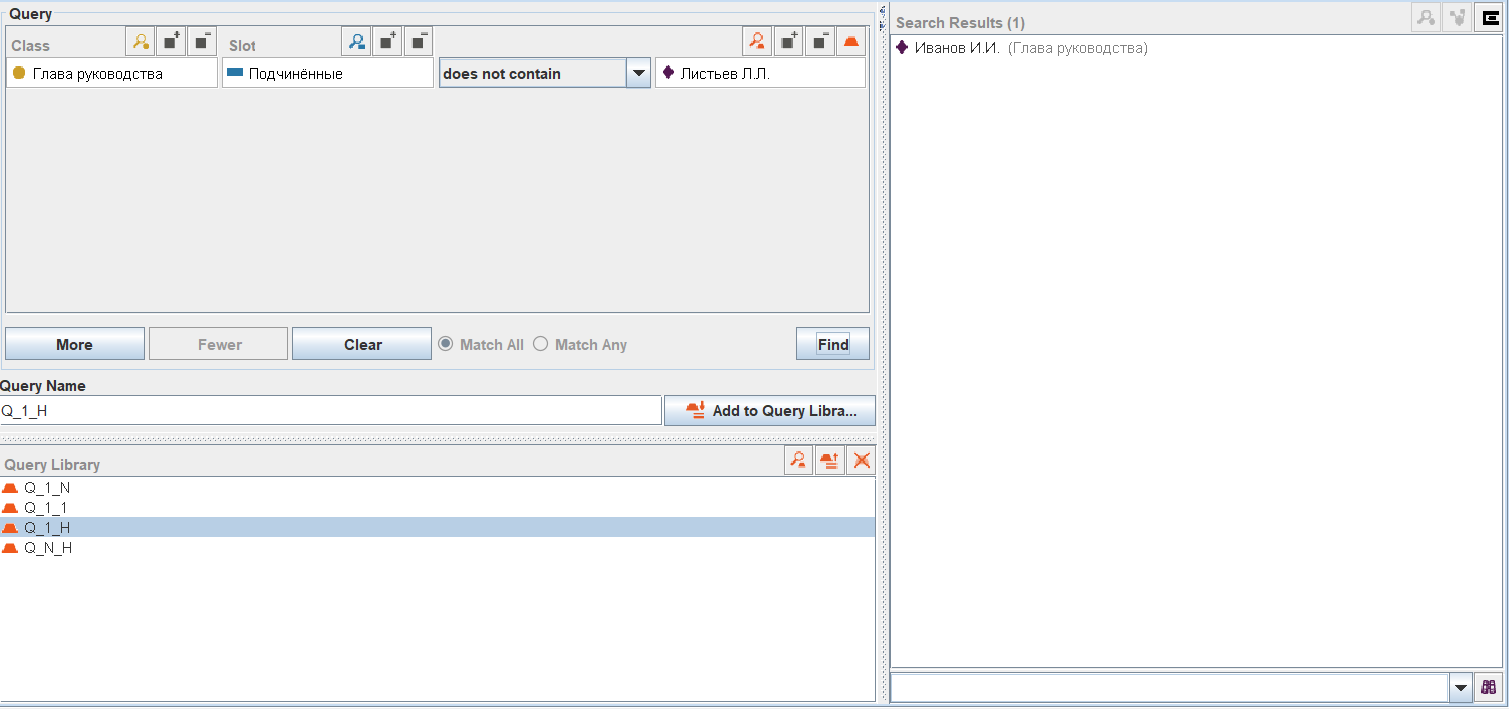
1. Один к одному;
2. Один ко многим;
3. Один к одному через рукопожатие;
4. Один ко многим через рукопожатие.



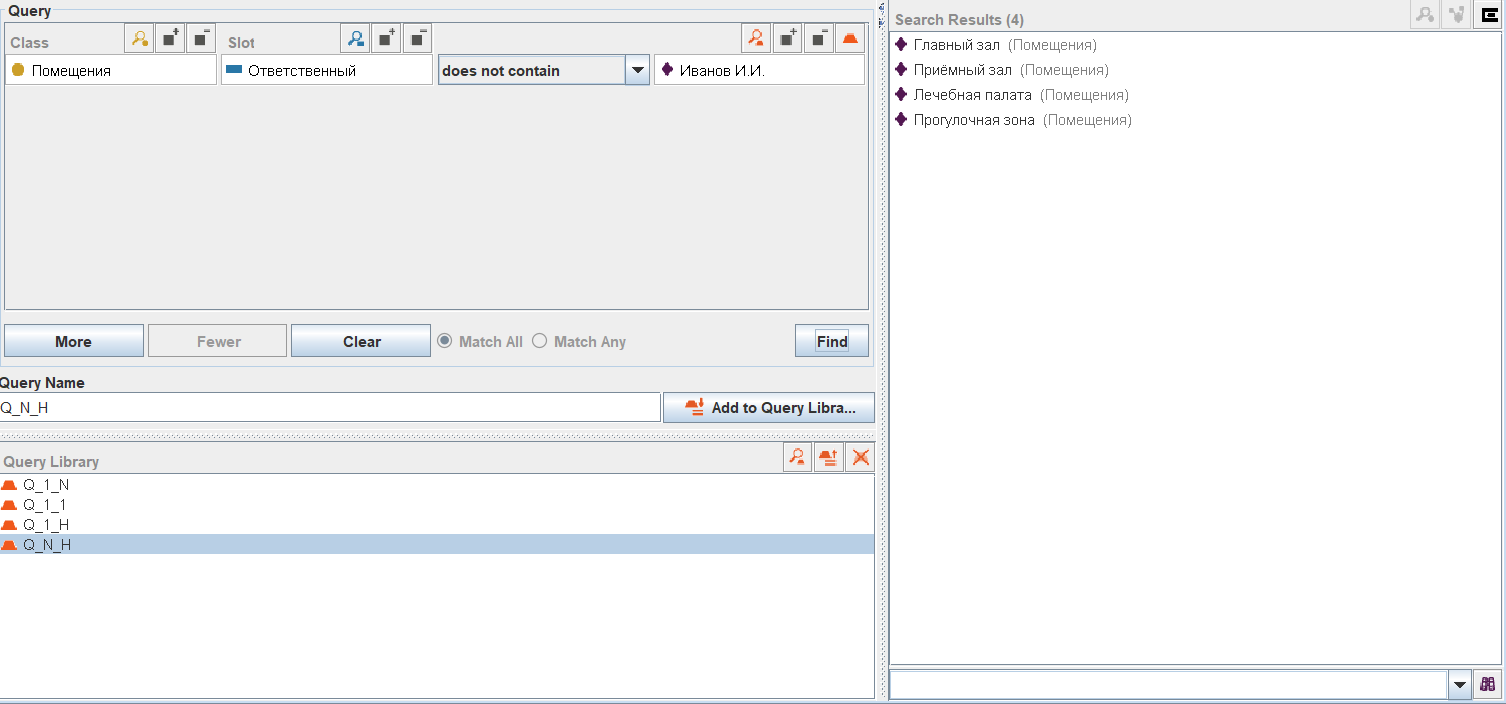
**Рисунок 8 – Запрос один к одному**



**Рисунок 9 – Запрос один ко многим**



**Рисунок 10 – Запрос один к одному через рукопожатие**



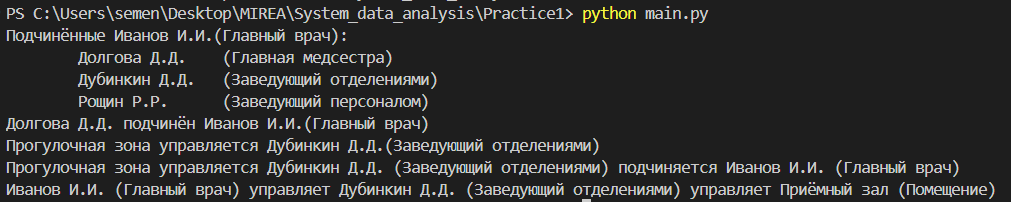
**Рисунок 11 – Запрос один ко многим через рукопожатие**

# 

# 1.5 Разработка базы знаний в кодовом виде

Перенесем базу знаний в программный вид, выполним реализацию на языке Python. Реализация представлена в приложении А.

Проверим написанную базу знаний через запросы.



**Рисунок 12 – Пример выполнения запросов объектов программного кода**

# 

# 1.6 Результат работы

В ходе выполнения практической работы была реализована база знаний по предметной области *«*Психиатрическая больница*»*.

1. Проектирование модели. На основе системного анализа построена иерархическая схема, включающая руководство, сотрудников и помещения учреждения, с отображением связей подчиненности;
2. Перенос в среду Protégé. База знаний была формализована с использованием онтологического редактора Protégé. В онтологии определены классы, экземпляры и отношения, что обеспечило структурированное представление информации о предметной области;
3. Реализация на языке Python. Построенная база знаний была дополнительно реализована кодово. В программе созданы структуры данных для хранения объектов и связей между ними, а также реализованы запросы, позволяющие получать сведения о подчинённых, руководителях и закрепленных помещениях;
4. Проверка работоспособности. Тестирование программы подтвердило корректность работы базы знаний: система успешно возвращала информацию об объектах и их отношениях, что соответствует построенной модели.

Таким образом, результатом работы стала разработанная и реализованная база знаний, представленная как в визуальной форме (схема), так и в цифровой (онтология в Protégé и программная реализация на Python).

# 

# ЗАКЛЮЧЕНИЕ

Создание онтологии позволяет систематизировать знания и обеспечить их единое понимание среди различных пользователей и приложений. Она способствует повышению качества информационных систем и облегчает обмен данными между ними. В процессе разработки особое внимание уделяется логической согласованности и точности определений. Применение онтологических моделей расширяет возможности автоматизированного вывода и интеллектуального поиска. Основные принципы построения онтологий делают их универсальным средством представления знаний. В целом, онтология играет ключевую роль в развитии современных технологий обработки информации и искусственного интеллекта.

# 

# СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Python Software Foundation. Python Documentation — [Электронный ресурс]. URL: https://docs.python.org/3/ (дата обращения: 15.09.2025).
2. Лутц М. Изучаем Python. 5-е изд. / пер. с англ. — Санкт-Петербург: Символ-Плюс, 2019. — 1648 с.
3. Баляев С. А. Объектно-ориентированное программирование. Учебное пособие. — Москва : ФОРУМ, ИНФРА-М, 2020. — 256 с.
4. Гринберг Д. Программирование на Python 3. Подробное руководство. — Москва : Вильямс, 2014. — 832 с.

# 

# ПРИЛОЖЕНИЯ

Приложение А – Код программы Онтологии

**Приложение А**

Код программы Онтологии

*Листинг А.1 — Основной алгоритм программы*

| *f*rom abc import ABC, abstractmethod  class PsyHospital:  pass  class Staff:  pass  class HeadHosp:  pass  class AdmHosp:  pass  class WorkerHosp:  pass  class RoomHosp:  pass  class PsyHospital(ABC):  pass  class Staff(PsyHospital):  pass  class HeadHosp(Staff):  def \_\_init\_\_(self, name: str, post: str):  self.name = name  self.post = post  self.subs = []  def print\_subs(self):  print(f"Подчинённые {self.name}({self.post}):")  for obj in self.subs:  print(f"\t{obj.name}\t({obj.post})")  def find\_boss(self, boss\_name):  if (self.boss.name == boss\_name):  return f"{self.boss.name} ({self.boss.post})"  else:  return None    def find\_staff (self, staff\_name):  for obj in self.subs:  if (obj.name == staff\_name):  return f"{self.name} ({self.post}) управляет {obj.name} ({obj.post})"  else:  find\_staff = obj.find\_staff(staff\_name)  if (find\_staff != None):  return f"{self.name} ({self.post}) управляет " |
| --- |

*Листинг А.2 — Продолжение листинга А.1*

| + find\_staff  return None  class AdmHosp(Staff):  def \_\_init\_\_(self, name: str, post: str, boss: HeadHosp):  self.name = name  self.post = post  self.boss = boss  self.subs = []  boss.subs.append(self)  def print\_subs(self):  print(f"Подчинённые {self.name}({self.post}):")  for obj in self.subs:  print(f"\t{obj.name}\t({obj.post})")  def print\_boss(self):  print(f"{self.name} подчинён {self.boss.name}({self.boss.post})")  def find\_boss(self, boss\_name):  if (self.boss.name == boss\_name):  return f"{self.name} ({self.post}) подчиняется {self.boss.name} ({self.boss.post})"  else:  boss\_of\_boss = self.boss.find\_boss(boss\_name)  if (boss\_of\_boss != None):  return f"{self.name} ({self.post}) подчиняется " + boss\_of\_boss  else:  return None    def find\_staff (self, staff\_name):  for obj in self.subs:  if (obj.name == staff\_name):  return f"{self.name} ({self.post}) управляет {obj.name} ({obj.post})"  else:  find\_staff = obj.find\_staff(staff\_name)  if (find\_staff != None):  return f"{self.name} ({self.post}) управляет " + find\_staff  return None  class WorkerHosp(Staff):  def \_\_init\_\_(self, name: str, post: str, boss: AdmHosp):  self.name = name  self.post = post  self.boss = boss  boss.subs.append(self)  def print\_boss(self):  print(f"{self.name} подчинён |
| --- |

*Листинг А.3 — Продолжение листинга А.2*

| {self.boss.name}({self.boss.post})")  def find\_boss(self, boss\_name):  if (self.boss.name == boss\_name):  return f"{self.name} ({self.post}) подчиняется {self.boss.name} ({self.boss.post})"  else:  boss\_of\_boss = self.boss.find\_boss(boss\_name)  if (boss\_of\_boss != None):  return f"{self.name} ({self.post}) подчиняется " + boss\_of\_boss  else:  return None    def find\_staff (self, staff\_name):  return None  class RoomHosp(PsyHospital):  def \_\_init\_\_(self, name: str, boss: AdmHosp):  self.name = name  self.boss = boss  self.post = "Помещение"  boss.subs.append(self)  def print\_boss(self):  print(f"{self.name} управляется {self.boss.name}({self.boss.post})")  def find\_boss(self, boss\_name):  if (self.boss.name == boss\_name):  return f"{self.name} управляется {self.boss.name} ({self.boss.post})"  else:  boss\_of\_boss = self.boss.find\_boss(boss\_name)  if (boss\_of\_boss != None):  return f"{self.name} управляется " + boss\_of\_boss  else:  return None    def find\_staff (self, staff\_name):  return None  if \_\_name\_\_ == '\_\_main\_\_':  # Глава руководства  Head = HeadHosp("Иванов И.И.", "Главный врач")  # Руководство  MainNurse = AdmHosp("Долгова Д.Д.", "Главная медсестра", Head)  HeadOfRooms = AdmHosp("Дубинкин Д.Д.", "Заведующий отделениями", Head)  HeadOfStaff = AdmHosp("Рощин Р.Р.", "Заведующий персоналом", Head) |
| --- |

*Листинг А.4 — Продолжение листинга А.3*

| # Штат  Secur = WorkerHosp("Листьев Л.Л.", "Охранник", HeadOfStaff)  MedBrat = WorkerHosp("Петров П.П.", "Санитар", MainNurse)  Cleaner = WorkerHosp("Сидоров С.С.", "Уборщик", HeadOfStaff)  Nurse = WorkerHosp("Смирнова С.С.", "Медсестра", MainNurse)  # Помещения  MainHall = RoomHosp("Главный зал", HeadOfRooms)  MedRoom = RoomHosp("Лечебная палата", MainNurse)  Reception = RoomHosp("Приёмный зал", HeadOfRooms)  WalkingArea = RoomHosp("Прогулочная зона", HeadOfRooms)  Head.print\_subs()  MainNurse.print\_boss()  WalkingArea.print\_boss()  print(WalkingArea.find\_boss("Иванов И.И."))  print(Head.find\_staff("Приёмный зал")) |
| --- |