Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

«Реалізація бази знань за допомогою системи PROTEGE»

Текстова частина до курсової роботи
за спеціальністю «Комп'ютерні науки» 122

Керівник курсової роботи

доц. Жежерун О.П.

_____

(підпис)

"____" _____ 2023 р.

Виконав студент БП КН-3

Вавдійчик В.О.

"____" _____ 2023 р.

Київ 2023

Тема: **Реалізація бази знань за допомогою системи PROTEGE**

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	15.04.2023	
2.	Огляд літератури за темою роботи.	17.04.2023	
3.	Проведення досліджень	20.04.2023	
4.	Опис результатів дослідження	25.04.2023	
5.	Аналіз отриманих результатів з керівником	03.05.2023	
6.	Корегування роботи	10.05.2023	
7.	Створення презентації та написання доповіді.	10.05.2023	
8.	Остаточне оформлення пояснювальної роботи та слайдів.	14.05.2023	
9.	Здача курсової роботи на перевірку	15.05.2023	

Студентка Гальченко М.О.

Керівник Жежерун О.П.

“_____” _____

Зміст

Анотація

У своїй роботі на основі існуючих підходів та систем створення онтологій представлено технологію побудови онтології засобами Protége. Розроблена онтологія орієнтована на пошук та аналіз інформації про продукти в мазагині. Також описується процес проектування та розробки бази знань, використання SPARQL запитів до побудованої онтології та створення застосунку на мові Python з її використанням.

Вступ

В сучасному світі збільшується кількість даних, які потрібно обробляти та аналізувати. У зв'язку з цим, виникає необхідність в розробці та застосуванні баз знань. База знань є інструментом, який дозволяє організувати інформацію в структурованій формі, що спрощує пошук та аналіз цієї інформації.

Онтологія — це система, що складається з набору понять і набору тверджень про ці поняття. На них будуються класи, об’єкти, зв’язки, функції та теорії. Для доступу до даних онтології краще використовувати мову запитів SPARQL. Актуальність цієї роботи полягає в новому підході до аналізу продуктів та покупки в магазині та візуалізації результатів за допомогою SPARQL запитів.

В даній курсовій роботі я буду розглядати реалізацію бази знань за допомогою системи PROTEGE. PROTEGE - це вільна та відкрита програма для розробки онтологій та баз знань. У роботі буде досліджено онтологічне моделювання, формальний опис та особливості реалізації онтологічної моделі.

У першому розділі будуть описані задачі онтологічного моделювання, формальний опис онтологічної моделі та особливості її реалізації. Другий розділ присвячений реалізації онтології за допомогою редактора Protégé. Буде надано загальний опис онтології "Grocery store" та реалізація цієї онтології в редакторі Protégé. Буде розглянуто отримання нових знань за допомогою reasoner “Pellet”. Також, у другому розділі буде розглянуто приклади запитів SPARQL та отриманню інформації з онтології "Grocery store". У третьому роділлі буде розглянуто реалізація демонстраційної програми на мові Python.

Отже, ця курсова робота дозволить вам ознайомитись з базами знань, онтологічним моделюванням та розробкою онтологій за допомогою системи PROTEGE. Вона допоможе вам отримати практичні навички роботи з системою PROTEGE та розуміння принципів розробки баз знань.

Розділ 1. Онтологічне моделювання

1.1 Задачі онтологічного моделювання

На сьогоднішній день, більшість систем бізнес-аналізу, насправді, побудовані на основі реляційних баз даних.. Проте зростає потреба в системах, які здійснюють інтерпретацію даних, а не тільки їх зберігання та обробку. Тут в гру вступають семантичні технології, які пропонують новий підхід до моделювання даних, який основується на онтологічних моделях.

Головна задача для онтологічного моделювання є забезпечення однозначної інтерпретації та розуміння даних. В цьому полягає їх відмінність від реляційних баз даних, що семантичні моделі дозволяють включати у себе не тільки дані, але й їхню семантику та взаємозв'язки між ними. Це дає змогу зберігати та обробляти не просто дані, але інформацію, з якою можна взаємодіяти, шукати залежності та отримувати нові знання.

Іншим важливим задаванням онтологічного моделювання є забезпечення інтеграції даних з різних джерел та їх стандартизація. Семантичні технології дозволяють включати у себе дані з різних джерел, а також забезпечують стандартизацію для їх представлення та інтерпретації. Це спрощує процес обробки даних та знижує ризик помилок.

Ще одною задачею онтологічного моделювання є забезпечення можливості автоматичної обробки великих об’ємів складно структурованих даних та їхньої інтерпретації. Семантичні технології можуть створювати програмні платформи, які здійснюють автоматичне зчитування та обробку даних, зокрема використовуючи машинне навчання та штучний інтелект. Це дозволяє значно зменшити навантаження на людські ресурси та збільшити швидкість процесу обробки даних.

Крім того, онтологічне моделювання допомагає вирішити проблему неоднозначності та нечіткості даних. Семантичні технології дозволяють описувати дані за допомогою онтологічних термінів та надають можливість автоматичної інтерпретації даних відповідно до заданих правил та контексту. Це дозволяє уникнути неоднозначності та забезпечити більш точну та однозначну інтерпретацію даних.

Розглянемо, також, конкретну проблему, яка виникає при вирішенні задач оптимізації. Для вирішення оптимізаційних завдань часто використовують кількісні методи. Такий підхід полягає у створенні "цільової функції", що описує бажаний стан системи у вигляді числа, наприклад, "забезпеченість населення деякими послугами". Далі встановлюються обмеження та параметри, які можуть змінюватись, і на основі обчислень отримується набір "оптимальних" рішень. Однак, практичне застосування таких методів може привести до непередбачуваних результатів. Наприклад, забезпеченість послугами зросте до потрібного рівня, але деякі групи населення їх не отримають, або якість послуг знизиться до такої міри, що вони стануть непотрібними для споживачів. Це пов'язано з технічними обмеженнями обробки інформації та складністю алгоритмів, що потрібні для розрахунку. Це може призводити до використання недоцільних або неправильних методів, що зумовлює недоліки моделювання.

Зазначені методичні проблеми безпосередньо пов'язані зі способами автоматизованої обробки інформації – точніше, з обмеженістю тієї їхньої частини, яку переважно використовує бізнес . Адже чим складніший та достовірний алгоритм розрахунку, тим більше ресурсів піде на його створення. Отже, за думкою замовника, його реалізація стає недоцільною. Це призводить до застосування невірного, грубого, проте технологічно зрозумілого методу розрахунку.

Для вирішення проблеми, яка зазначена вище, потрібно навчити нашу систему вирішувати задачі способом подібним до того як мислить людина. А

конкретно, мова іде про логічне мислення. Для цього нам потрібно створити в цифровому вигляді інформаційні структури і процеси подібні до людських. І після отримання деякого результату, ми зможемо обробити його і покращити.

Для цього нам і потрібні семантичні технології, які забезпечують роботу з сенсом інформації. Вони забезпечують роботу саме зі знаннями, бо вони вже вираженні за допомогою тих понять, якими користується людина. Окрім того, з такими онтологіями (знаннями) можуть виконуватися, як це вже було зазначено, і повністю автоматичні операції. Результатом цього стане отримання нових знань.

Отже, семантичні технології та онтологічне моделювання відкривають нові можливості для бізнесу та дозволяють зберігати, обробляти та інтерпретувати дані з більшою точністю, порівняно з реляційними базами даних.

1.2 Формальний опис онтологічної моделі

Для початку розглянемо поняття «модель», та її загальні характеристики. Це допоможе краще орієнтуватися при реалізації онтологічної моделі.

Модель – це інформаційне представлення будь-якої сукупності об'єктів та явищ реального світу, що характеризується:

- Спрощенням – у моделі представлена інформація не про всі характеристики розглянутого фрагмента дійсності, а лише про ті, які вважливі з прагматичної точки зору конкретного суб'єкта.
- Концептуалізованістю – кожен об'єкт, явище, властивість
- виражені за допомогою понять абстракції різного рівня.
- Взаємопов'язаністю – всі елементи моделі пов'язані між собою.
- Наявністю логічних правил взаємодії елементів.
- Прогностичним потенціалом – ми можемо виконати уявний експеримент, додавши до моделі ті чи інші параметри чи події, і за допомогою зазначених вище логічних правил зробити висновок про те, що відбудеться в тому чи іншому випадку.

Далі розглянемо процес створення формальної онтологічної моделі, який базується на тому як люди сприймають інформацію. Цей процес іде за певною послідовністю:

1) Декомпозиція – виділення сутностей, які приймають участь в моделі.

2) Ідентифікація – створення індивідуальних сутностей.

3) Класифікація – створення класів, які відповідають групам сутностей. Тобто, додання сутностей в певний клас.

4) Опис властивостей – визначення способів задання інформації про характеристики і відносини сутностей.

5) Значення, зв’язки – присвоєння значень властивостям, створення зав’язків.

Розглянемо детальніше кожний пункт та розберемося як саме він працює для онтологічних моделей.

Найперше, що нам потрібно зробити при побудові інформаційної моделі, як вже було зазначено вище, це зробити **декомпозицію**. Цей метод полягає у поділі модельованого фрагмента реальності на окремі елементи, які стануть базовими одиницями інформаційної моделі. У комп'ютерній моделі ми відображатимемо саме деякі образи. Якщо на картинці зображено дві людини та м’яч, то першими образами будуть: «людина», «м’яч», «людина». Звичайно декомпозицію можна продовжувати виділяючи все нові та нові об’єкти. Наприклад, якщо при першому розгляді нашого фрагменту реальності, було виявлено людину, то далі ми можемо виділяти інші об’єкти такі як елементи гардеробу або аксесуари.

Глибина декомпозиції, визначення меж об'єктів, а також рішення про те, чи створювати (чи видаляти) об'єкти в моделі при істотній зміні їх стану, залежать лише від прагматики – практичного призначення моделі. Якщо ми розглядаємо деяку картинку реальності, де нам не важливі одяг або предмети, які має при собі людина, то ми і не будемо включати це в нашу модель. Наприклад, якщо ми розглядаємо модель професійної гри в футбол, де у всіх гравців однаково

хороший одяг та взуття, то не має сенсу включати це до нашої моделі. На противагу, якщо ми розглядаємо ситуацію де предмети впливають на подальший розвиток подій, то нам потрібно додавати їх до нашої моделі. Наприклад, ситуація затримання злочинця, де, в залежності від того чи є пістолет у злочинця, буде залежати порядок дій працівника поліції. Також зрозуміло, що якщо ми будемо детально моделювати всі наявні у підозрюваного предмети, то складання та інтерпретація моделі зажадають надто багато часу та ресурсів, що недоцільно з економічної точки зору. Завдання правильного обілення полягає у пошуку практично обґрунтованого та оптимального співвідношення між рівнем деталізації моделі (і, як наслідок, достовірності результатів моделювання) та необхідних для цього ресурсів.

По-друге, ідентифікація. Тобто присвоєння унікального номера кожному об’єкту. Тут важливо зазначити, що потрібно дотримуватися принципу нейтральності ідентифікаторів. Ідентифікатор сам по собі не повинен нести жодного сенсу, не має права ґрунтуватися на жодних властивостях об'єкта. Це потрібно для того, щоб не було потреби змінювати ідентифікатор при зміні властивостей самого об’єкта.

По-третє, класифікація. Для того, щоб почати робити класифікацію, нам потрібно визначити набір доступних нам класів. Саме поняття «клас» в онтологічних моделях відрізаєтеся від класу в ООП. Тут в це поняття не входить ні признаки, по яким відносять об’єкти до цього класу, ні загальні характеристики, які притаманні всім об’єктам даного класу. Вся ця інформація буде вказана в нашій моделі. Отже, відношення об’єкту до класу буде повністю залежати від практичного застосування, тобто як саме буде використовуватися наша модель.

Класифікація в інформаційних моделях потрібна для того, щоб отримати можливість формулювати знання та будувати логічні висновки. Для цього потрібно включити до моделі твердження про класи, які відповідають арсеналу

логіки першого порядку. Одним із видів таких тверджень є вирази формату «Всі члени класу X є членами класу Y» (наприклад, «усі собаки – тварини»).

Також потрібно зазначити принципи побудови ієрархії класів. Перш за все, потрібно зрозуміти, що кожен об’єкт може відноситися зразу до декількох класів. Такий принцип класифікації називається фасетним. Зауважимо, що основи класифікації самі по собі можуть утворювати досить розгалужені ієрархії. На верхньому рівні їх можна розділити на ті, що описують внутрішні властивості об'єкта (матеріал, форма), і ті, що характеризують види відносин, у яких об'єкт може вступати з іншими об'єктами (призначення, роль). Бажано використовувати наступні основи для класифікації:

- область застосування товару;

- спосіб використання;

- функціональне призначення.

Це потрібно для того, щоб запобігти деяким труднощам. Наприклад, орієнтування в інтернет магазині. Де може бути ситуація, коли кухонний рушник знаходиться в розділі «Товари для дому – Для кухні», проте користувач може шукати їх в розділі «Текстиль». Підкреслюю, що показані способи класифікації не є «правильними» самі по собі – вони лише можуть бути більш раціональними у світлі якихось конкретних завдань. Якщо зміняться цілі, іншою має стати і класифікація.

По-четверте, опис властивостей. Властивості об'єкта дозволяють описати його характеристики та атрибути, які допомагають розрізнити один об'єкт від іншого та надають більш детальну інформацію про нього.

У формальній онтологічній моделі кожна властивість та відношення між об'єктами має бути формально визначене і описане, щоб забезпечити їх коректну обробку та використання в системі.

У моделі формальної онтології властивості об’єктів поділяються на дві категорії:

Обов’язкові та необов’язкові. Обов'язкові властивості відображають основні атрибути об'єкта, необхідні для його ідентифікації. Наприклад, при моделюванні автомобіля серед необхідних властивостей можуть бути марка, модель, тип двигуна, ємність бака тощо. Необов'язкові властивості відображають другорядні характеристики об'єкта, які корисні для певного типу аналізу даних, але не потрібні для його ідентифікації. Для автомобіля, наприклад, це може бути колір кузова, тип салону, максимальна швидкість тощо. Властивості об’єктів також можуть мати значення, які можуть бути представлені за допомогою різних типів даних, таких як числа, рядки та логічні значення. Наприклад, якщо ви моделюєте автомобіль, значення властивості «паливний об’єм» можна виразити в літрах, а значення властивості «максимальна швидкість» можна виразити в кілометрах на годину.

Однією з важливих властивостей об’єктів у моделі онтології є властивість «клас». Кожен об’єкт має належати до певного класу, який описує його властивості та поведінку в системі. Наприклад, об’єкт «Автомобіль» належить до класу «Транспортний засіб», а об’єкт «Морква» — до класу «Овоч».

По-п’яте, значення і зв’язки. Після визначення властивостей певного об’єкта, потрібно задати значення цим властивостям. Тут варто зазначити, що основні характеристики властивостей це:

- назва

- обмеження на тип та діапазон значень

- набір об'єктів, які можуть та/або повинні бути носієм цього властивості

Отже, потрібно подбати про те, щоб значення властивостей не перевищували обмеження, а також щоб чітко було задано набір об’єктів яким ця властивість підходить. Крім того, об'єкти можуть мати відношення між собою, що описують залежності та зв'язки між ними. Наприклад, об'єкт "книга" може мати

відношення "має автора", "належить до жанру", "видана в певному році" та інші.

Підсумовуючи все вище сказане, описуючи формальну онтологічну модель, я зобразив також принципи її побудови для кращого розуміння самої сутності моделі.

1.3 Особливості реалізації онтологічної моделі

Семантичні технології є наступним кроком у розвитку способів подання інформації. Вони дають можливість реалізувати в електронному вигляді концептуальні моделі, побудовані за принципами, описаним у попередньому розділі, дозволяють передавати що міститься в цих моделях інформацію та автоматично обробляти її, у тому числі – отримувати логічні висновки виходячи з правил.

Перед тим як перейти до реалізації деяких функції, необхідно сказати про основний спосіб вираження інформації, представленій в онтологічних моделях, його називають триплетом. Триплет – синтаксична структура, що складається з трьох елементів:

суб'єкт – предикат -- об'єкт

Суб'єкт – це завжди будь-яка сутність моделі, про яку повідомляється інформація.

Предикат – властивість, значення якого хочемо задати цього об'єкта.

Об’єкт – може бути або літералом (числовим, рядковим значенням) або іншою сутністю.

Розробка онтологічної моделі може бути проведена за допомогою різних інструментів. У цьому розділі я розгляну особливості реалізації онтологічної

моделі за допомогою RDF, OWL, редактора онтологій Protege а також, буде розглянуто мову запитів SPARQL.

RDF (Resource Description Framework) – це формат для представлення даних у вигляді триплетів (суб'єкт-предикат-об'єкт), де суб'єкт і об'єкт – це ресурси, а предикат – це відношення між ними. Реалізація онтологічної моделі за допомогою RDF полягає у створенні файлу з триплетами, які описують концепти, властивості та відносини між об'єктами в галузі. Оскільки RDF використовується для опису семантичних мереж, модель може бути використана для розробки систем, що працюють з великими масивами даних.

Protege – це інструмент для розробки онтологій, який використовує формат OWL (Web Ontology Language) для представлення знань. OWL – це формальна мова для опису онтологій, яка має більш розширені можливості порівняно з RDF. За допомогою Protege можна створювати класи, підкласи, індивідуальні екземпляри та відносини між ними. Крім того, Protege надає можливість виконувати перевірку правильності моделі та автоматично генерувати код для реалізації систем, що працюють з онтологією.

SPARQL (Protocol and RDF Query Language) - це мова запитів для RDF-даних. SPARQL дозволяє виконувати запити до онтологічних моделей, що забезпечує можливість отримувати знання з різних джерел та забезпечує узгодженість знань зі світовими стандартами.

Інтеграція Protege та SPARQL дозволяє ефективно виконувати запити до онтологічних моделей за допомогою мови SPARQL та робити вибірку необхідних даних, що значно спрощує аналіз інформації та забезпечує точність результатів. Також SPARQL дозволяє виконувати запити до декількох онтологій одночасно, що забезпечує можливість інтеграції даних з різних джерел та забезпечує зручний доступ до знань.

Створення класу

Почнемо розгляд реалізації онтології зі створення класу в редакторі Protégé.

Я вже пояснював як представляються сутності в онтологічних моделях (триплетом). Отже, факт існування класу буде представлений наступним чином:

Суб'єкт	Предикат	Об'єкт
Унікальний ідентифікатор класу	Має тип	Клас

Словами "Має тип" я позначив предикат у форматі RDF, який використовується для визначення типу сутностей онтологічних моделей. URI для цієї властивості виглядає так: http://www.w3.org/1999/02/22-rdf-syntax-ns#type, чи скорочено – rdf:type.

В RDF файлі це буде виглядати наступним чином:

<owl:Class rdf:about="http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#Product"/>

Для того щоб покращити читабельність використовують скорочення використовуючи префікси, які заміняють спільну для всіх частину URI. Ось як у мене це представлено на початку RDF файлу:

<rdf:RDF xmlns="http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3/"

xml:base="http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3/"

xmlns:owl="http://www.w3.org/2002/07/owl#"

xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

xmlns:xml="http://www.w3.org/XML/1998/namespace"

xmlns:xsd="http://www.w3.org/2001/XMLSchema#"

xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

Щоб створити цей клас у файлі RDF я використав Protégé. Перейшовши у вкладку Classes, я натиснув «Add subclass», надав йому назву «Product».

Виконавши наступний SPARQL-запит : `SELECT * WHERE { ?a ?b ?c }`

Я побачив наступний триплет:

Суб'єкт	Предикат	Об'єкт
<http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#Product>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2002/07/owl#Class>

Запити SPARQL я задаю в Apache Jena Fuseki. Для запуску локального сервера необхідний Java компілятор.

Також до класу є можливість додати властивість label, щоб покращити його читабельність. Для цього потрібно перейти в "Associations" та вибрати "label".

В RDF файлі це виглядає наступним чином:

<owl:Class rdf:about="http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#Product">

<*rdfs:label*>*Продукт*</*rdfs:label*>

</owl:Class>

Тепер триплет виглядає ось так :

Суб’єкт	Предикат	Об’єкт
<http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#Product>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	Продукт

Створення індивідуального об’єкта

У Protégé сутності всіх типів створюються, використовуючи різні закладки редактора:

- Classes – класи,

- Individuals – індивідуальні об'єкти,

- Data properties – властивості-літерали,

- Object properties – властивості-покажчики на об'єкти.

Закладка Entities дозволяє побачити всі ці елементи одночасно.

У RDF/XML отримані сутності відрізнятимуться типом, відповідно:

- http://www.w3.org/2002/07/owl#Class

- http://www.w3.org/2002/07/owl#NamedIndividual

- http://www.w3.org/2002/07/owl#DatatypeProperty

- http://www.w3.org/2002/07/owl#ObjectProperty

Щоб створити сутність в редакторі Protégé, треба перейти до вкладки Individuals. Щоб додати приналежність об’єкта до якогось класу треба натиснути «+» біля Types.

Я створив сутність Apple та зазначив, що вона відноситься до класу Product. Ось як це виглядає в RDF файлі:

<owl:NamedIndividual rdf:about="http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#Apple">

<rdf:type rdf:resource="http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#Product"/>

<rdfs:label>Яблуко</rdfs:label>

</owl:NamedIndividual>

Також я додав сутності Apple label «Яблуко».

Триплет даної сутності буде виглядати наступним чином:

<http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#Apple>	<http://www.w3.org/2000/01/rdf-schema#label>	Яблуко

Створення властивості

Щоб створити властивість, значення якої це літерал, потрібно перейти в «Data properties» та натиснути «Add sub property». В правій частині вікна буде видно кнопки Domain і Range.

Domain – це клас, до екземплярів якого можна застосувати ця властивість. Range – це тип значень, які властивість може набувати. Відповідно, значення для Domain вибирається зі списку класів, існуючої в нашій моделі, а Range – із певних стандартом типів даних (використовуються типи даних XSD)

Я створив властивість productName, яка застосовується до екземплярів класу “Product” і має тип значення string.

Ось так це представлено в RDF файлі:

<owl:DatatypeProperty rdf:about="http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#productName">

<rdfs:domain rdf:resource="http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#Product"/>

<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>

</owl:DatatypeProperty>

Далі я створив властивість, яка відображає відносини між об’єктами в моїй онтології (Object properties). Для цього я перейшов у вкладку «Object properties» та натиснув кнопку «Add sub property». Я створив властивість “isLocatedIn”. В Domains я вказав клас Product, а в Ranges вказав Store. Тобто виходить така властивість: Product isLocatedIn Store.

Ось так це виглядає в файлі RDF:

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#isLocatedIn">

<rdfs:domain rdf:resource="http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#Product"/>

<rdfs:range rdf:resource="http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#Store"/>

</owl:ObjectProperty>

Тепер ми можемо присвоїти значення властивості “productName” конкретному продукту “Apple”. Для цього необхідно перейти до вкладки Individuals, натиснути “+” біля Data property assertion та присвоїти “productName” конкретне значення, в моєму випадку це «Симиренко» (сорт яблука).

В RDF це виглядає наступним чином:

```
<owl:NamedIndividual
rdf:about="http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#Apple">
    <rdf:type
rdf:resource="http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#Product"/>
    <untitled-ontology-3:productName>Симиренко</untitled-ontology-3:productName>
    <rdfs:label>Яблуко</rdfs:label>
</owl:NamedIndividual>
```

Отже, в цьому розділі я розглянув особливості створення онтологічної моделі використовуючи редактор онтології Protégé. Також, було розглянуто як саме записуються в rdf файлах класи, об’єкти та властивості.

Розділ 2. Реалізація онтології за допомогою редактора Protégé.

2.1 Загальний опис онтології «Grocery store»

Данна онтологія призначена для пошуку товарів в магазині, їхніх характеристик, пошуку товарів за їхніми характеристиками або за їх категорією, а також для пошуку куплених товарів деяким покупцем.

Онтологія «Grocery store» описує невеличкий продуктовий магазин, в якому є різні товари та покупці, які їх купляють. Кожен продукт має мати категорію до якої він належить («Для дому», «Хлібобулочні вироби» і так далі). Також, для кожного продукту визначається його виробник, ціна та назва. При купівлі товару (або декількох товарі) формується чек, в який пов'язує покупця та товари які він купив. Для кожного покупця визначається його ім'я та номер телефону. Також, у цій онтології є знання про те, які товари додавав менеджер.

2.2 Реалізація онтології «Grocery store» в редакторі Protégé.

Про редактор онтологій Protégé було написано в попередньому розділі. Також, там зазначалося як створюються класи, об'єкти та властивості.

Для реалізації онтології «Grocery store» спочатку була реалізована ієрархія класів, яка зображена на Рис. 2.1.

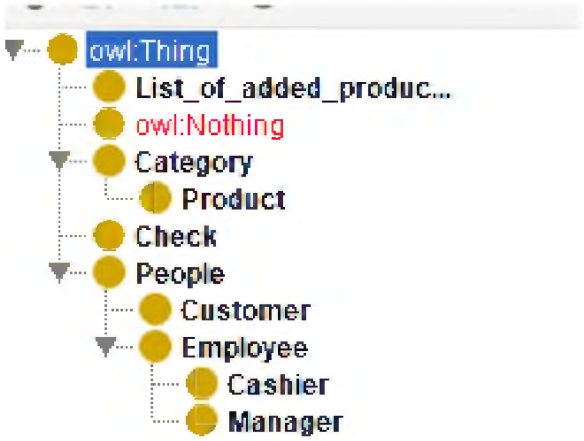


Рис. 2.1

Наддам пояснення для чого було створено кожен з представлених класів:

- Об’єкт класу Category позначає деяку категорію продуктів.
- Клас Product позначає множину продуктів певної категорії, так як він є підкласом класу Category.
- Об’єкт класу Check позначає чек, який отримує конкретний покупець при купівлі товарів в магазині. Об’єкт цього класу створюють зв'язок між об’єктом класу Customer та об’єктами класу Product.
- Клас People позначає множину всіх людей в магазині, а саме працівників магазину (Employee) та покупців (Customer).
- Клас Customer позначає множину всіх покупців магазину.
- Клас Employee позначає множину всіх працівників магазину.
- Клас Cashier позначає множину всіх касирів магазину.
- Клас Manager позначає множину всіх менеджерів магазину.
- Клас List_of_added_products позначає множину списків доданих до магазину нових товарів.
- owl:Thing – це початковий клас від якого унаслідуються всі інші класи.

Також, зобразити дану ієрархію класів можна у вигляді графа, як це показано на

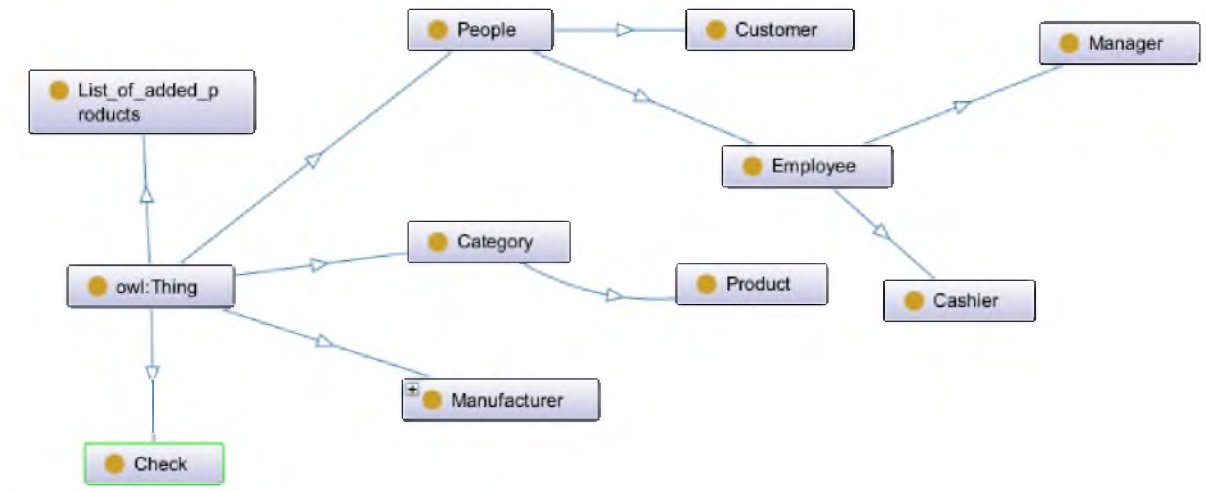


Рис. 2.2

На Рис. 2.2 синя стрілка означає «має підклас» (has subclass).

Список всіх object properties надано на Рис. 2.3.

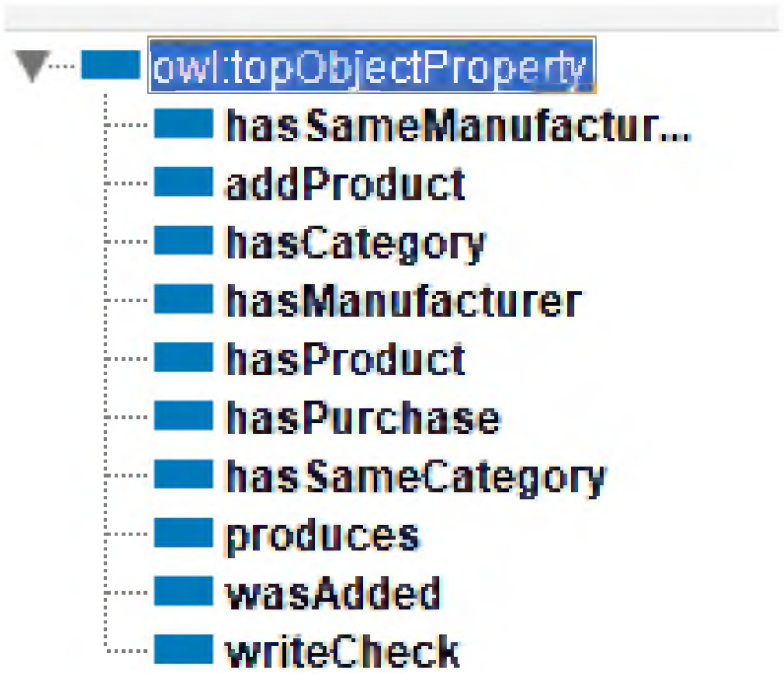


Рис. 2.3

Об’єктні властивості можуть мати наступні характеристики:

1) Функціональність (Functional) Якщо властивість є функціональною, то для одного екземпляра може існувати не більше одного екземпляра, який має відношення до першого через цю властивість.
2) Обернена-функціональність (Inverse functional). Якщо властивість є обернено-функціональною, то дана властивість є оберненою до функціональної властивості.
3) Транзитивність (Transitive). Якщо властивість є транзитивною, то виконується умова транзитивності: якщо екземпляр a зв’язаний з b, a b зв’язаний з c, то можемо зробити висновок, що a зв’язаний з c через транзитивну властивість.
4) Симетричність (Symmetric). Якщо властивість x є симетричною, і екземпляр a зв’язаний з b через таку властивість, то можемо зробити висновок, що b також зв’язаний з a через властивість x.

5) Асиметричність (Asymmetric). Якщо властивість x є асиметричною, і екземпляр a зв'язаний з b через таку властивість, то b не може бути зв'язаний з a через властивість x.
6) Рефлексивність (Reflexive). Властивість x є рефлексивною, якщо екземпляр a зв'язаний сам із собою.
7) Ірефлексивність (Irreflexive). Якщо властивість x є ірефлексивною, то вона зв'язує екземпляр a і b через таку властивість, проте екземпляри a та b обов'язково мають бути різними.

Отже, описуючи object properties, які були представлені на Рис. 2.3, можна зазначити наступне:

- wasAdded позначає властивість, яка зв'язує клас List_of_added_products та клас Product. Тобто, доменом цієї властивості є List_of_added_products, а діапазоном значень є Product.
- hasSameCategory позначає властивість, яка зв'язує продукти одної категорії. Тобто, доменом цієї властивості є Product і діапазоном значення є, також, Product. Ця властивість симетрична (якщо продукт А має таку ж категорію, що і продукт Б, то очевидно, що продукт Б буде мати таку ж категорію, що і продукт А), транзитивна (A hasSameCategory Б, Б hasSameCategory B => A hasSameCategory B).
- hasManufacturer позначає властивість, яка зв'язує продукт та виробника цього продукту. Тобто, доменом цієї властивості є клас Product, а діапазоном значень є Manufacturer. Ця властивість також є функціональною, адже у конкретного продукту буде лише один виробник.
- produces позначає властивість, яка зв'язує виробника та продукти, які він виготовив. Ця властивість є оберненою до властивості hasManufacturer. Позначається це через "InverseOf", як показана на Рис. 2.4

Рис. 2.4

- addProduct позначає властивість, яка зв'язує клас Manager та клас List_of_added_products. Тобто, доменом цієї властивості є Manager, а діапазоном значень є List_of_added_products.
- hasCategory позначає властивість, яка зв'язує клас Product та клас Category. Тобто, доменом цієї властивості є Product, а діапазоном значень є Category.
- hasProduct позначає властивість, яка зв'язує клас Check та клас Product. Тобто, доменом цієї властивості є Check, а діапазоном значень є Product.
- hasPurchase позначає властивість, яка зв'язує клас Customer та клас Check. Тобто, доменом цієї властивості є Customer, а діапазоном значень є Check.
- writeCheck позначає властивість, яка зв'язує клас Cashier та клас Check. Тобто, доменом цієї властивості є Cashier, а діапазоном значень є Check.

Властивості даних (Data property), які були створені, представлені на Рис 2.5.

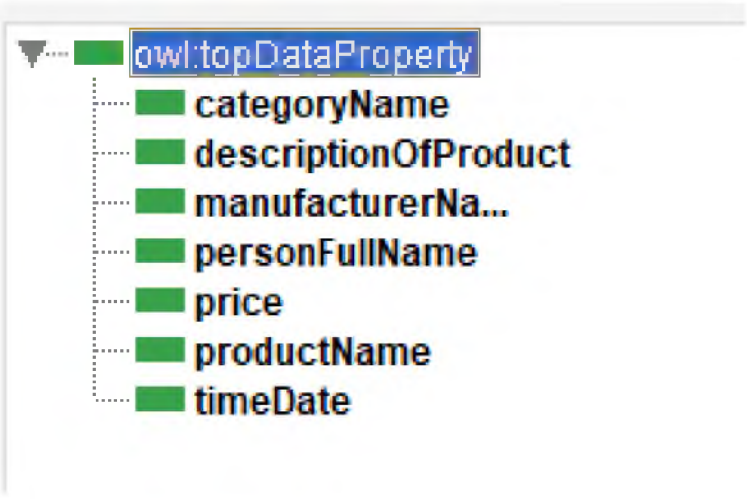


Рис 2.5

Опис властивостей, представлених на Рис. 2.4, знаходиться в Табл. 2.1

Назва	Domains	Тип даних	Призначення
timeDate	Check, List_of_added_products	dateTime	Для запису часу створення чеку або листа доданих нових продуктів
categoryName	Category	string	Для назви категорії
descriptionOfProduct	Product	string	Для опису продукту. (Маса нетто та інші характеристики товару)
productName	Product	string	Назва продукту (Молоко, Рогалики і т. д. і т. п.)
price	Product	decimal	Ціна продукту
personalFullName	Person	string	ПІБ людини
manufacturerName	Manufacturer	string	Назва виробника

Табл. 2.1

Підсумовуючи, під час створення онтології було додано дев'ять класів, шість властивостей об'єкту (object properties), сім властивостей даних (data properties) та біля сотні індивідуальних об'єктів всіх класів.

2.3 Робота Reasoner та деякі SWRL правила

Reasoner (машина логічного висновка) – це клас програмного забезпечення, яке призначене для роботи з онтологічними моделями. Він дозволяє вичислити значення логічних виразів, перевірити правильність онтології та автоматично додавати до неї нову інформацію у відповідності з правилами. Машині логічного висновку дозволено оперувати іменами класів, властивостей і сутностей, і «задавати моделі питань», абстрагуючи користувача від деталей внутрішньої будови моделі.

Для роботи був використаний reasoner «Pellet», який, на мою думку, є найбільш оптимальним рішенням в рамках даної онтології.

Після визначення object properties та їхніх характеристик, можна поглянути на роботу Reasoner. Спочатку його необхідно запустити, натиснувши «Start reasoner» у меню Reasoner. Після його запуску запропоновані висновки будуть підсвічуватися жовтим кольором.

- Задавши характеристики властивості hasSameCategory та вказавши значення, які представлені в Табл. 2.2:

Об'єкт	Предикат	Суб'єкт
Bread01	hasSameCategory	Bread02
Bread03	hasSameCategory	Bread01
Bread04	hasSameCategory	Bread01

Табл. 2.2

Було отримано результат, який представлений на Рис. 2.5.

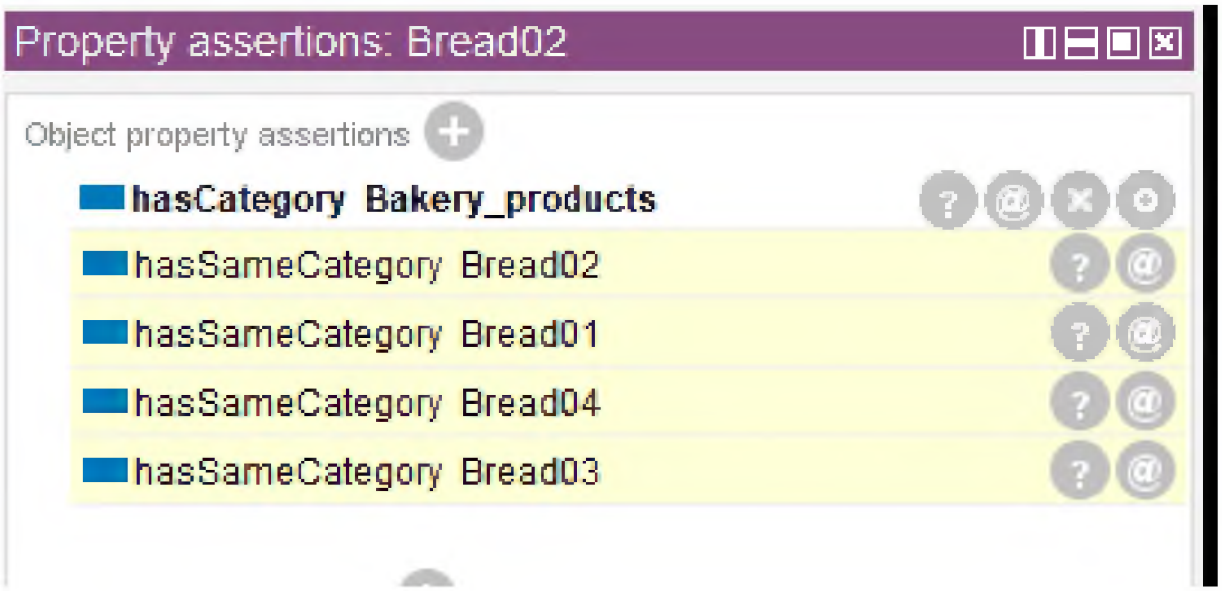


Рис. 2.6.

На Рис. 2.6 представлено висновки для об'єкта Bread02. Висновки для об'єктів Bread01, Bread03, Bread04 будуть схожі. Нагадую, що властивість hasSameCategory є симетричною та транзитивною. На Рис. 2.5 можна наглядно бачити як працюють ці властивості.

- Властивість є hasManufacturer є функціональною, це означає те, що у одного товару може бути лише один виробник. Отже, якщо задати певному продукту два виробника, то reasoner має видати помилку. Хоча, якщо не задано, що всі об'єкти класу Виробник є різними об'єктами, то reasoner припустить, що KyivHlib та MBM_My_home це один і той самий об'єкт. На Рис. 2.7 представлено, як буде виглядати помилка.

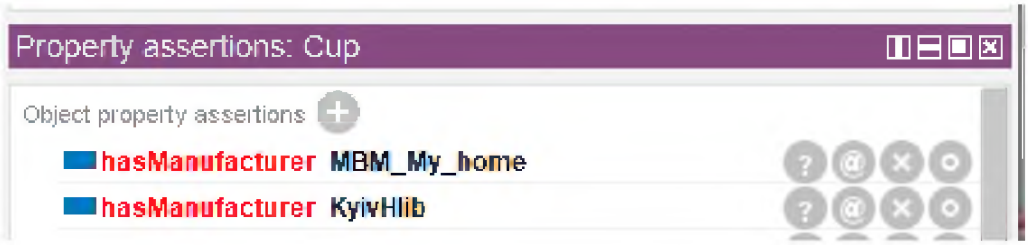


Рис. 2.7.

- Властивість produces (виробляє) є оберненою до властивості hasManufacturer. Отже, після запуску reasoner буде видавати висновки представлені на Рис. 2.8.

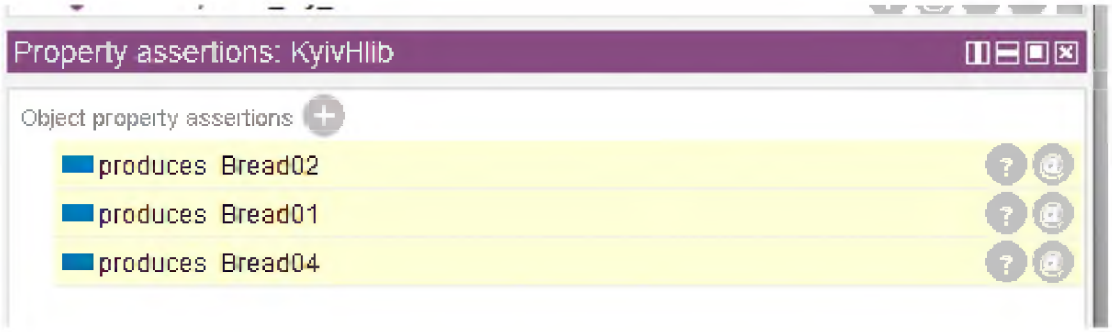


Рис. 2.8.

SWRL правила

SWRL — це мова правил семантичної павутини , яка може використовуватися для вираження правил, а також логіки, поєднуючи OWL DL або OWL Lite з підмножиною мови розмітки правил.

Для даної онтологічної моделі було задано два SWRL правила:

1) hasCategory(?x, ?category), hasCategory(?y, ?category) -> hasSameCategory(?x, ?y)

   Перше правило означає, що для всіх ?x та ?y, на яких виражено властивість hasCategory до одної категорії (?category), тобто, це означає, що вони мають однакову категорію. Тоді для всіх x та y буде виражено властивість hasSameCategory (мають однакову категорію).
   Для прикладу було створено об’єкт класу Product, який має ідентифікатор Bread101, та додано йому властивість: Bread101 -- hasCategory – Bakery_products. Після запуску reasoner було отримано висновки, які продемонстровано на Рис. 2.9.

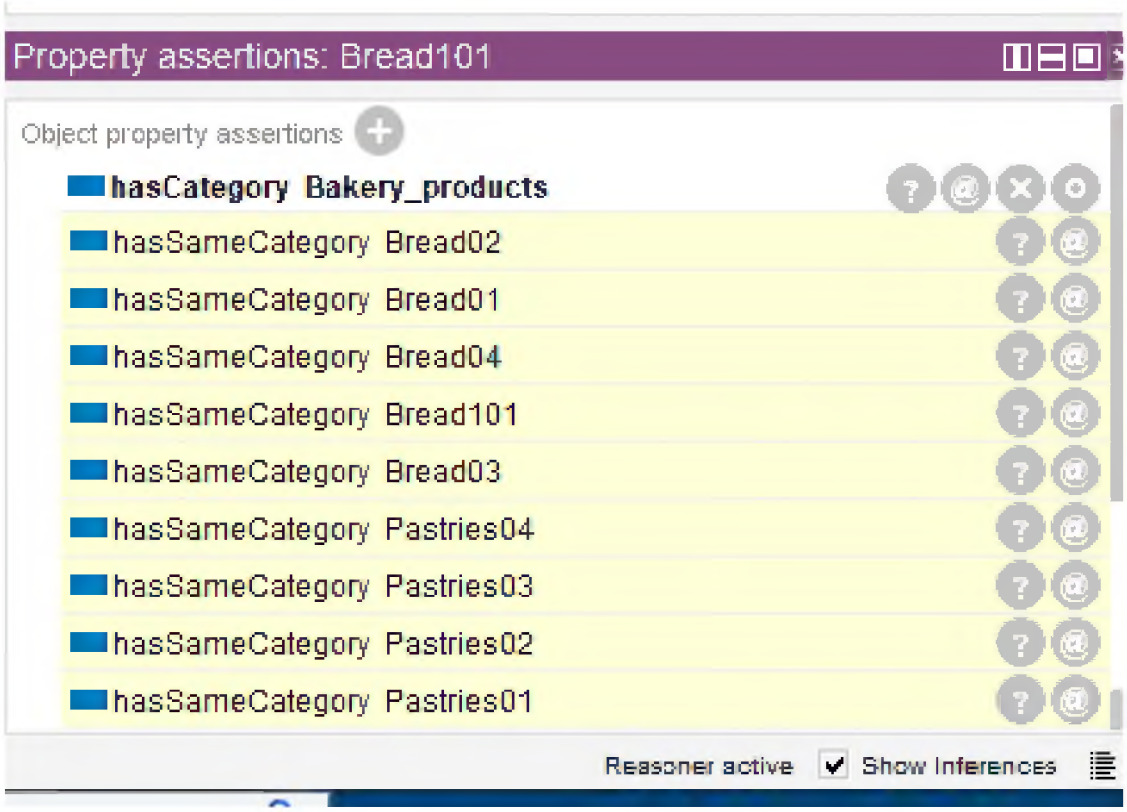


Рис. 2.9.

2) hasSameCategory(?x, ?y), hasCategory(?y, ?category) -> hasCategory(?x, ?category)

Друге правило означає, що для всіх x та y, на яких виражено властивість hasSameCategory та для y виражено властивість hasCategory до певної категорії (?category)), то з цього слідує, що для x теж має бути виражена властивість hasCategory до ?category.

Для прикладу було створено об’єкт класу Product, який має ідентифікатор Bread102, та додано йому властивість: Bread101 -- hasSameCategory – Bread01. Після запуску reasoner було отримано висновки, які продемонстровано на Рис. 2.10.

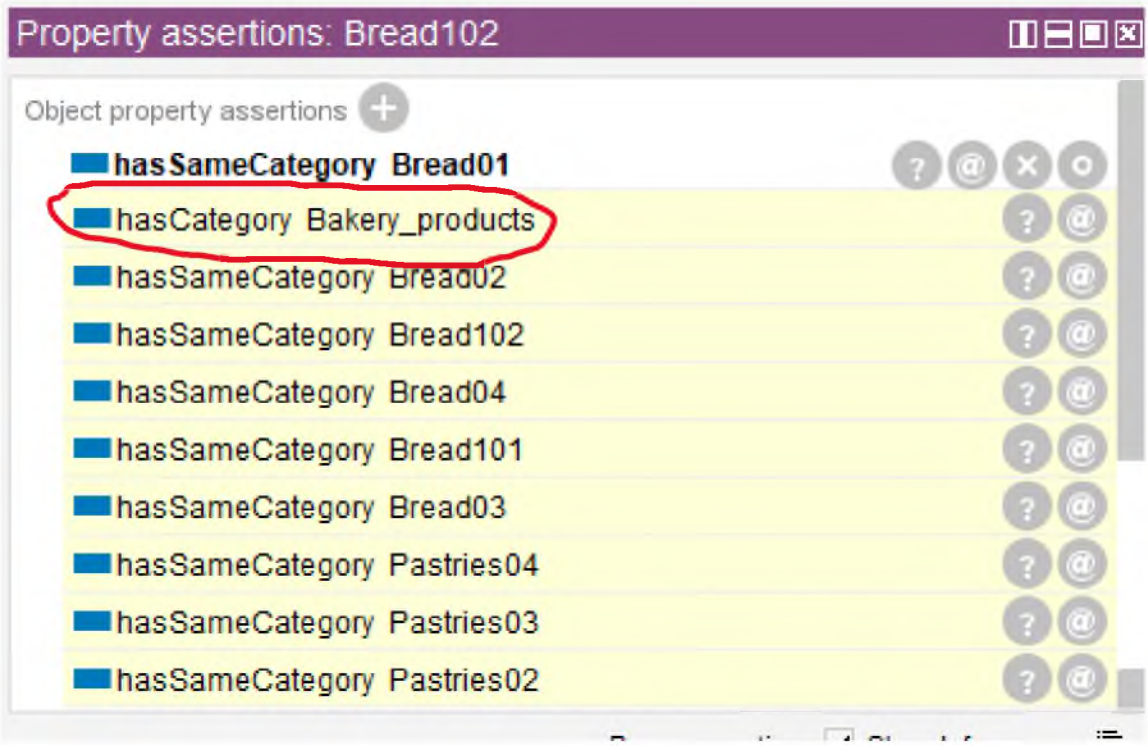


Рис. 2.10

2.4 Запити SPARQL та отримання інформації з онтології «Grocery store»

Мова запитів SPARQL – це мова запитів до даних, представлених по моделі RDF, а також протокол для передачі цих запитів і відповідей на них. SPARQL є рекомендацією консорціуму W3C і одною з технологій семантичної павутини. Представлення SPARQL-точок доступу (SPARQL endpoint) є рекомендованою практикою при публікації даних у всесвітній павутині.

Для отримання результатів запитів SPARQL до онтології «Grocery store» було використано Apache Jena Fuseki версії 4.8.0.

Apache Jena — це фреймворк семантичної мережі з відкритим кодом на Java. Він надає API для вилучення даних із графіф RDF і запису в них. Графи представлені у вигляді абстрактної «моделі». Модель може бути джерелом даних з файлів, баз даних, URL-адрес або їх комбінації.

В свою чергу, Apache Jena Fuseki — це сервер SPARQL. Він може працювати як служба операційної системи, як веб-додаток Java (файл WAR) і як окремий сервер.

Очевидно, що для його використання необхідно мати встановлену Java.

Сервер запускається за допомогою bat файлу, який знаходиться в папці програми.

На Рис 2.11 показано, як виглядає веб інтерфейс Apache Jena Fuseki.

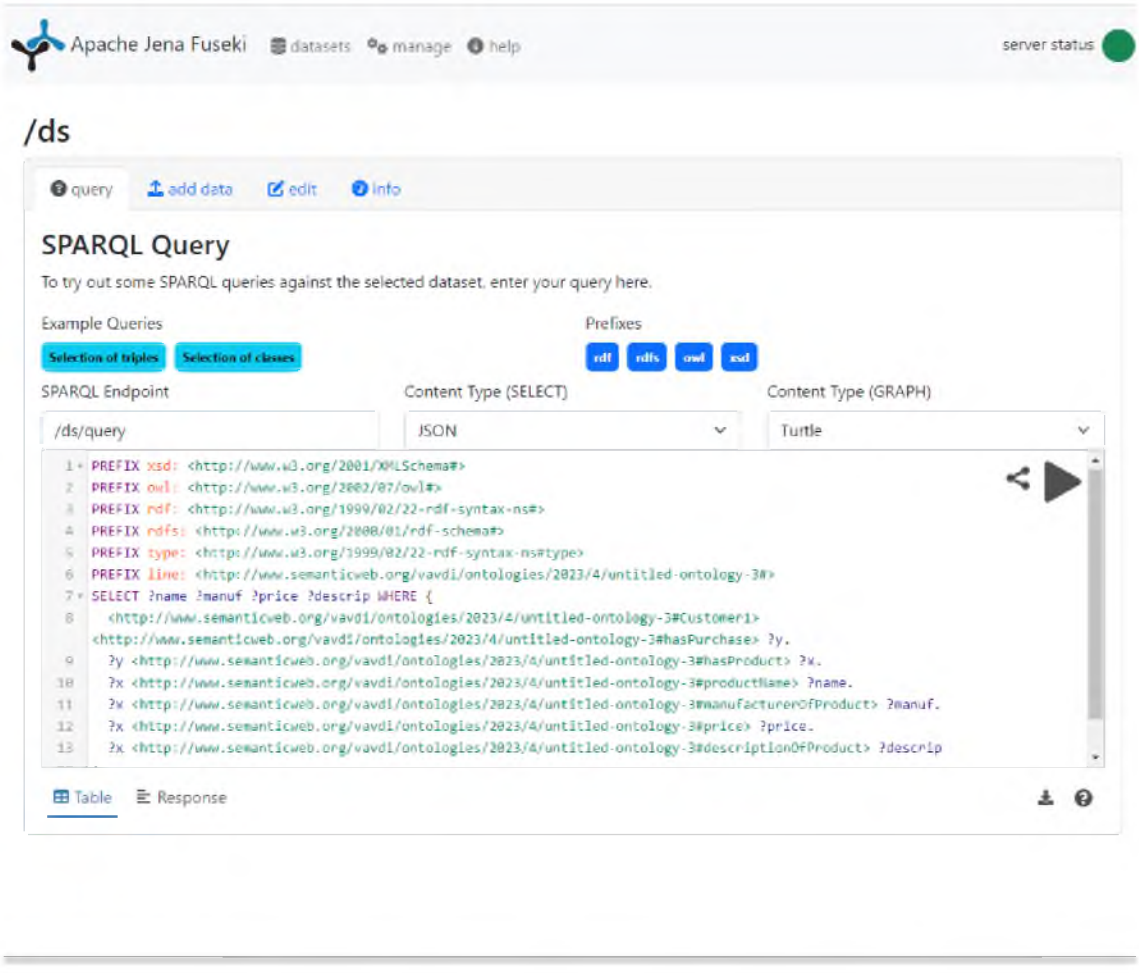


Рис 2.11

SPARQL запити

Для кращої читабельності SPARQL запитів використовують префікси (PREFIX). Вони потрібні аби зайвий раз не прописувати повний URI. В усіх запитах будуть використані наступні префікси:

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

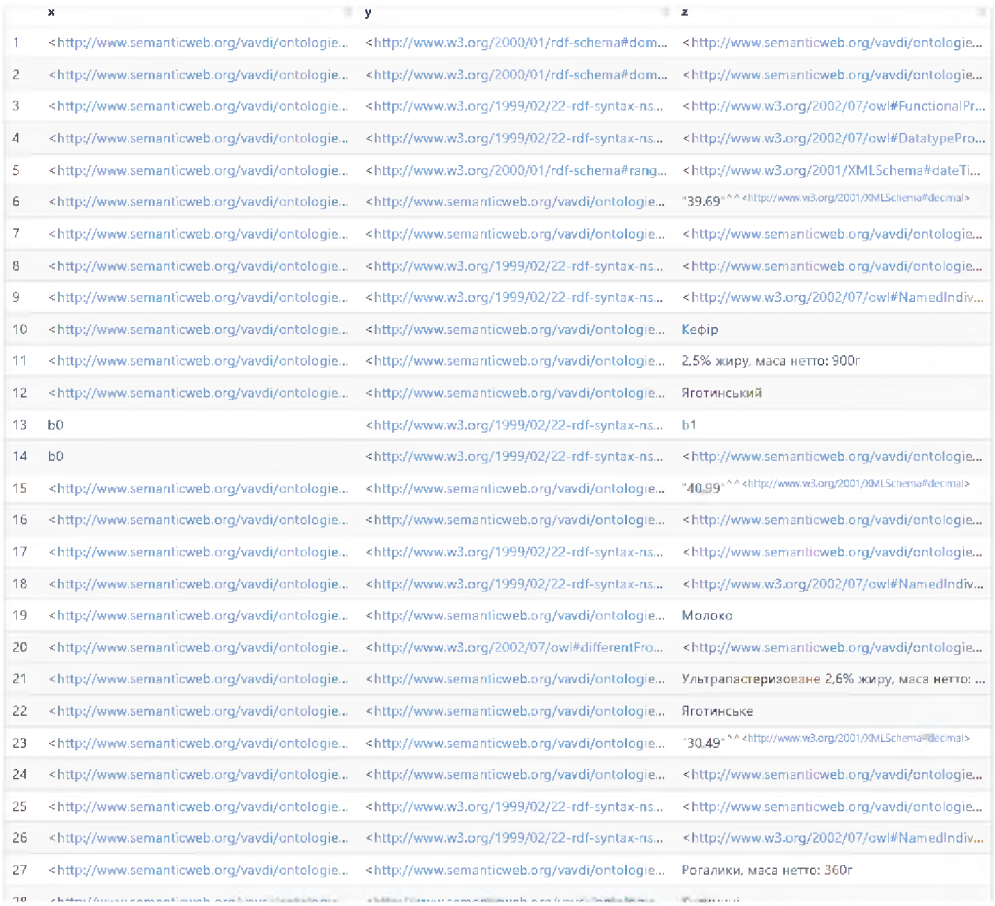
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX type: http://www.w3.org/1999/02/22-rdf-syntax-ns#type

1) SELECT ?x ?y ?z WHERE {
?x ?y ?z
}
LIMIT 100

Результатом виконання першого запиту на онтології «Grocery store» стане виведення всіх триплетів даної онтології, як це показано на Рис. 2.12.



	x	y	z
1	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.w3.org/2000/01/rdf-schema#dom...	<http://www.semanticweb.org/vavdi/ontologie...
2	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.w3.org/2000/01/rdf-schema#dom...	<http://www.semanticweb.org/vavdi/ontologie...
3	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.w3.org/1999/02/22-rdf-syntax-ns...	<http://www.w3.org/2002/07/owl#FunctionalPr...
4	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.w3.org/1999/02/22-rdf-syntax-ns...	<http://www.w3.org/2002/07/owl#DatatypePro...
5	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.w3.org/2000/01/rdf-schema#rang...	<http://www.w3.org/2001/XMLSchema#dateTi...
6	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.semanticweb.org/vavdi/ontologie...	"39.69"^^<http://www.w3.org/2001/XMLSchema#decimal>
7	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.semanticweb.org/vavdi/ontologie...
8	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.w3.org/1999/02/22-rdf-syntax-ns...	<http://www.semanticweb.org/vavdi/ontologie...
9	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.w3.org/1999/02/22-rdf-syntax-ns...	<http://www.w3.org/2002/07/owl#NamedIndiv...
10	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.semanticweb.org/vavdi/ontologie...	Кефір
11	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.semanticweb.org/vavdi/ontologie...	2,5% жиру, маса нетто: 900г
12	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.semanticweb.org/vavdi/ontologie...	Яготинський
13	b0	<http://www.w3.org/1999/02/22-rdf-syntax-ns...	b1
14	b0	<http://www.w3.org/1999/02/22-rdf-syntax-ns...	<http://www.semanticweb.org/vavdi/ontologie...
15	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.semanticweb.org/vavdi/ontologie...	"40.99"^^<http://www.w3.org/2001/XMLSchema#decimal>
16	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.semanticweb.org/vavdi/ontologie...
17	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.w3.org/1999/02/22-rdf-syntax-ns...	<http://www.semanticweb.org/vavdi/ontologie...
18	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.w3.org/1999/02/22-rdf-syntax-ns...	<http://www.w3.org/2002/07/owl#NamedIndiv...
19	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.semanticweb.org/vavdi/ontologie...	Молоко
20	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.w3.org/2002/07/owl#differentFro...	<http://www.semanticweb.org/vavdi/ontologie...
21	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.semanticweb.org/vavdi/ontologie...	Ультрапастеризоване 2,6% жиру, маса нетто: ...
22	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.semanticweb.org/vavdi/ontologie...	Яготинське
23	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.semanticweb.org/vavdi/ontologie...	"30.49"^^<http://www.w3.org/2001/XMLSchema#decimal>
24	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.semanticweb.org/vavdi/ontologie...
25	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.w3.org/1999/02/22-rdf-syntax-ns...	<http://www.semanticweb.org/vavdi/ontologie...
26	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.w3.org/1999/02/22-rdf-syntax-ns...	<http://www.w3.org/2002/07/owl#NamedIndiv...
27	<http://www.semanticweb.org/vavdi/ontologie...	<http://www.semanticweb.org/vavdi/ontologie...	Рогалики, маса нетто: 360г

Рис. 2.12

2) SELECT DISTINCT ?name ?manuf ?price ?descrip WHERE {

?x type: <http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#Product>.

?x <http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#hasCategory> <http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#Dairy_products>.

?x <http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#productName> ?name.

OPTIONAL {

?x <http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#manufacturerOfProduct> ?manuf.

```
    ?x <http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#price> ?price.
    ?x <http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#descriptionOfProduct> ?descrip
  }
}
LIMIT 100
```

Результатом виконання другого запиту стане виведення всіх назв товарів категорії Dairy_products (молочні продукти). Також, якщо у об'єкта класу продукт, який належить до згаданої вище категорії, вказана ціна, виробник або опис, це теж буде виведено. Саме для ситуацій, коли результатом запиту може бути порожня множина, використовують OPTIONAL. Результат виконання другого запиту частково наданий в Табл. 2.2.

Кефір	Яготинський	39.69	2,5% жиру, маса нетто: 900г
Молоко	Яготинське	40.99	Ультрапастеризоване 2,6% жиру, маса нетто: 900г
Вершки	Яготинські	23.22	10% жиру, маса нетто: 200г
Кефір	Ферма	40.39	2,5% жиру, маса нетто: 840г
Масло	Галичина	76.59	Жирність 82,5%, маса нетто 180г
Вершки	Простоквашино	30.29	15% жиру, маса нетто: 200г
Сир	Ферма	49.59	Сметанковий, маса нетто 150г
Йогурт	Фанні	23.22	Десерт сирковий "Вершковий", маса

			нетто: 150г
Йогурт	Галичина	23.99	Карпатський, вишня-черешня, маса нетто: 260г
Масло	Ферма	92.99	Жирність 82,5%, маса нетто 180г
Молоко	Селянське	33.59	Ультрапастеризоване 3,2% жиру, маса нетто 900г
Сметана	President	41.59	15% жиру, маса нетто: 325г
Сир	Комо	48.40	Старий Голагдець, маса нетто 100г

Табл. 2.2

3) SELECT ?name ?manuf ?price ?descrip WHERE {
<http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#Check000000>
<http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#hasProduct> ?x.
?x <http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#productName> ?name.
?x <http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#manufacturerOfProduct> ?manuf.
?x <http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#price> ?price.
?x <http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#descriptionOfProduct> ?descrip
}
LIMIT 100

Результатом виконання третього запиту стане знаходження всіх продуктів за унікальним ідентифікатором чеку, в даному прикладі за Check000000. Так само, як і в попередньому прикладі запиту тут будуть виводитися назва продукту, його опис, ціна та бренд. Результат виконання третього запиту наданий в Табл. 2.3.

Назва	Бренд	Ціна	Опис
Молоко	Яготинське	40.99	Ультрапастеризоване 2,6% жиру, маса нетто: 900г
Хліб	Київхліб	21.49	Звичайний нарізний, маса нетто 500г
Чашка	МВМ My home	195	Червона з малюнком, 300 мл
Йогурт	Галичина	23.99	Карпатський, вишня-черешня, маса нетто: 260г

Табл. 2.3

4) SELECT ?name ?manuf ?price ?descrip WHERE {
<http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#Customer1>
<http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#hasPurchase> ?y.
?y <http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#hasProduct> ?x.
?x <http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#productName> ?name.
?x <http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#manufacturerOfProduct> ?manuf.
?x <http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#price> ?price.
?x <http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#descriptionOfProduct> ?descrip
}
LIMIT 100

Результатом виконання четвертого запиту стане знаходження всіх продуктів які купляв певний покупець, в даному прикладі Customer1. Так само, як і в попередніх прикладах запитів, тут будуть виводитися назва продукту, його опис, ціна та бренд. Результат виконання третього запиту наданий в Табл. 2.4.

Назва	Бренд	Ціна	Опис
Молоко	Яготинське	40.99	Ультрапастеризоване 2,6% жиру, маса нетто: 900г
Хліб	Київхліб	21.49	Звичайний нарізний, маса нетто 500г
Чашка	MBM My home	195	Червона з малюнком, 300 мл
Йогурт	Галичина	23.99	Карпатський, вишня-черешня, маса нетто: 260г
Булочки	Київхліб	14.19	"Малятко", маса нетто: 250г
Масло	Яготинське	75.25	Жирність 82,5%, маса нетто 180г

Табл. 2.4

Розділ 3. Реалізація демонстраційної програми на Python

3.1 Отримання інформації з RDF файлу

Для отримання даних з онтології було використано бібліотеку rdflib, яка надає можливості для задання SPARQL запитів до онтології.

RDFLib — це бібліотека Python для роботи з RDF. Ця бібліотека містить аналізатори/серіалізатори для майже всіх відомих серіалізацій RDF, таких як RDF/XML, Turtle, N-Triples і JSON-LD, багато з яких тепер підтримуються в оновленій формі (наприклад, Turtle 1.1). Бібліотека також містить вбудовані в пам'ять і постійні серверні модулі Graph для зберігання інформації RDF і численні зручні функції для оголошення просторів імен графів, розміщення запитів SPARQL і так далі.

На Рис. 3.1 зображено підключення бібліотеки rdflib та прасинг RDF файлу.

```
from rdflib import Graph


g = Graph()
g.parse("D:\course_work\ontology_model.rdf")
```

Рис. 3.1

3.2 SPARQL запити в програмі

Весь SPARQL запит задається у зміну. Для зручності всі PREFIXES було винесено в окрему зміну, яка виглядає наступним чином:

```
  prefixes = """
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX type: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
"""
```

Як виглядає запит в коді програми можна побачити на Рис. 3.2.

```
all_products_query = prefixes + """
SELECT ?name ?manufName ?price ?descrip WHERE {
  ?x type: <http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#Product>.
  ?x <http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#productName> ?name.
  ?x <http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#hasManufacturer> ?manuf.
  ?manuf <http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#manufacturerName> ?manufN
  ?x <http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#price> ?price.
  ?x <http://www.semanticweb.org/vavdi/ontologies/2023/4/untitled-ontology-3#descriptionOfProduct> ?descri
}
LIMIT 100
"""
```

Рис. 3.2

Задання запиту до онтології відбувається використовуючи метод g.query(), якому передається запит у вигляді рядка. Це виглядає в коді наступним чином:

```
g.query(all_products())
```

Для прикладу було взято запит, який виводить інформацію про всі продукти в магазині.

3.3 Демонстрація виводу результатів SPARQL запитів

Для виведення інформації в табличному вигляді було використано бібліотеку tkinter. Також, за її допомогою було зроблено весь інший графічний інтерфейс програми.

Tkinter — багатоплатформна графічна бібліотека інтерфейсів на основі засобів Tk (широко розповсюджена у світі GNU/Linux та інших UNIX подібних систем, портована в тому числі і на Microsoft Windows, Apple Mac OS), поширювана з відкритими вихідними текстами.

Результат запиту, який показує інформацію про всі продукти в магазині, наведено на Рис. 3.3

Назва	Виробник	Опис	Ціна
Хліб	Київхліб	Звичайний нарізний, маса нетто 5(	21.49
Хліб	Київхліб	"Супертост", маса нетто 500г	26.59
Хліб	Кулиниці	Звичайний нарізний, маса нетто 2!	14.99
Хліб	Київхліб	"Оксамитовий", маса нетто: 350г	26.99
Масло	Яготинське	Жирність 82,5%, маса нетто 180г	75.25
Масло	Ферма	Жирність 82,5%, маса нетто 180г	92.99
Масло	Галичина	Жирність 82,5%, маса нетто 180г	76.59
Сир	Ферма	Сметанковий, маса нетто 150г	49.59
Сир	Комо	Старий Голагдець, маса нетто 100г	48.40
Сир	Комо	Традиційний, маса нетто 150г	57.99
Сир	Комо	Парменталь, маса нетто: 160г	59.99
Вершки	Яготинське	10% жиру, маса нетто: 200г	23.22
Чашка	MBM_My_home	Червона з малюнком, 300 мл	195
Кефір	Яготинське	2,5% жиру, маса нетто: 900г	39.69
Кефір	Ферма	2,5% жиру, маса нетто: 840г	40.39
Молоко	Яготинське	Ультрапастеризоване 2,6% жиру, м	40.99
Молоко	Селянське	Ультрапастеризоване 3,2% жиру, м	33.59
Молоко	Селянське	Пастиризоване 2,6% жиру, маса не	61.99
Булочки	Київхліб	"Малятко", маса нетто: 250г	14.19
Рогалики	Кулиниці	Маса нетто: 360г	30.49
Кекс	Кулиниці	"Рум'янець" вишневий, маса нетто	15.99
Пиріг	Кулиниці	"Рум'янець" з маком, маса нетто: 3	59.99
Сметана	President	15% жиру, маса нетто: 325г	41.59

Рис. 3.3

Результат запиту, який показує інформацію про всі продукти певної категорії , наведено на Рис. 3.4. Для прикладу було взято категорію молочних подуктів.

Назва	Виробник	Опис	Ціна
Масло	Яготинське	Жирність 82,5%, маса нетто 180г	75.25
Масло	Ферма	Жирність 82,5%, маса нетто 180г	92.99
Масло	Галичина	Жирність 82,5%, маса нетто 180г	76.59
Сир	Ферма	Сметанковий, маса нетто 150г	49.59
Сир	Комо	Старий Голагдець, маса нетто 100г	48.40
Сир	Комо	Традиційний, маса нетто 150г	57.99
Сир	Комо	Парменталь, маса нетто: 160г	59.99
Вершки	Яготинське	10% жиру, маса нетто: 200г	23.22
Кефір	Яготинське	2,5% жиру, маса нетто: 900г	39.69
Кефір	Ферма	2,5% жиру, маса нетто: 840г	40.39
Молоко	Яготинське	Ультрапастеризоване 2,6% жиру, м	40.99
Молоко	Селянське	Ультрапастеризоване 3,2% жиру, м	33.59
Молоко	Селянське	Пастиризоване 2,6% жиру, маса не	61.99
Сметана	President	15% жиру, маса нетто: 325г	41.59
Сметана	Яготинське	20% жиру, маса нетто 400г	42.63
Сметана	Молокія	20% жиру, маса нетто: 300г	35.99
Йогурт	Галичина	Карпатський, вишня-черешня, ма	23.99
Йогурт	Фанні	Десерт сирковий "Вершковий", ма	23.22
Вершки	Простоквашено	15% жиру, маса нетто: 200г	30.29

Рис. 3.4

Результат запиту, який показує інформацію про всі продукти, які були в деякому чеку, наведено на Рис. 3.5. Для прикладу було взято чек з id: Check000000.

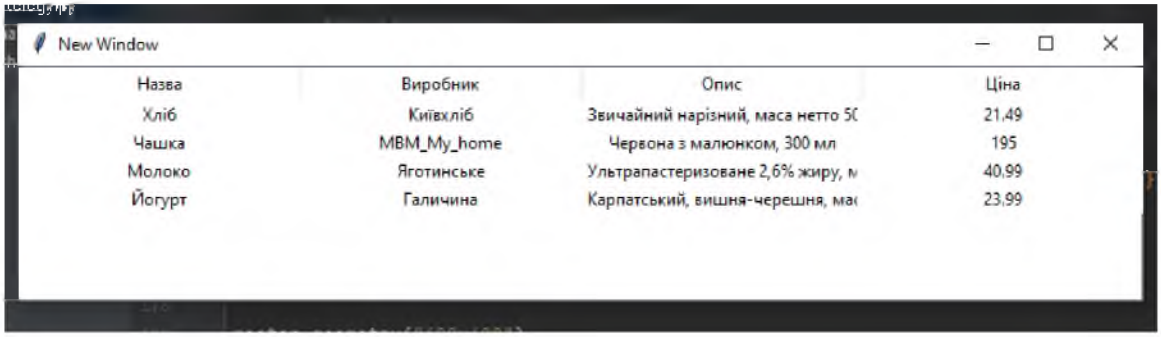


Рис. 3.5

Результат запиту, який показує інформацію про всі продукти, які були куплені деяким покупцем, наведено на Рис. 3.6. Для прикладу було взято покупця з id:Customer1.

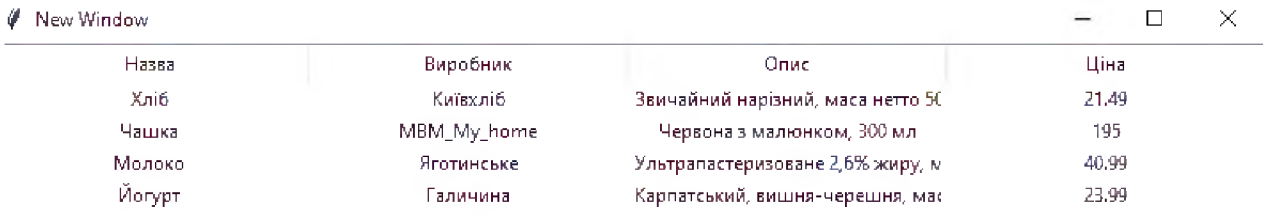


Рис. 3.6.

Оскільки у Customer1 тільки один чек (Check000000), тому продукти співпадають з Рис. 3.5.

Висновки

Отже, заплановані цілі були успішно виконані. У ході роботи було проаналізовано існуючі підходи та методи розробки онтологій. Також було проведено аналіз існуючих технологій, середовищ розробки, фреймворків та плагінів. Програма Protégé, на моє переконання, найкраще підходить для створення онтологій.

У результаті роботи було успішно створено онтологію "Grocery store" та реалізовано її в системі Protégé. Було протестовано створену онтологію за допомогою reasoner, а також за його допомогою було отримано нові знання. Було розглянуто мову запитів SPARQL та виконано декілька запитів до побудованої онтології. Також, було розглянуто реалізацію демонстраційної програми на мові Python. Досягнення поставлених цілей підтверджується результатами моєї роботи, а саме: успішна реалізація бази знань та можливість ефективного пошуку та аналізу інформації про продукти в магазині.

Список використаної літератури

1. Gruber T.R. The role of common ontology in achieving sharable, reusable knowledge bases // Principles of Knowledge Representation and Reasoning.

Proceedings of the Second International Conference. J.A. Allen, R. Fikes, E. Sandewell – eds. Morgan Kaufmann, 1991, 601-602.

2. Gruber T.R. A translation approach to portable ontologies. Knowledge Acquisition, 5(2): 199-220, 1993.

3. The KAON. – Режим доступу: http://kaon.semanticweb.org/

4. The OilEd. – Режим доступу: http://oiled.man/

5. The OntoEdit. – Режим доступу: http://www.ontoprise.de/com/ontoedit.htm

6. The Ontosaurus. – Режим доступу: http://www.isi.edu/isd/ontosaurus.html

7. The OpenCyc. – Режим доступу: http://www.opencyc.org/

8. The Protege Project. – Режим доступу: http://protege.stanford.edu.

9. Briukhov D.O., Shumilov S.S. Ontology Specification and Integration Facilities in a Semantic Interoperation Framework. Proc. of the Second Intern. Workshop ADBIS'95. Moscow, 1995. - P. 195-200.

10. Natalya F. Noy and Deborah L. McGuinness. «Ontology Development 101: A

Guide to Creating Your First Ontology». Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, March 2001.