Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Кафедра мережних технологій факультету інформатики

Розподіленні бази знань

Текстова частина до курсової роботи

за спеціальністю «Інженерія програмного забезпечення»- 121

Керівник курсової роботи
к.ф.-м.н., доцент
Жежерун О.П.
_____
(підпис)
“____” _____ 2023 р.
Виконав слухач 3-го року навчання:
Мисько Ю.М.
“____” _____ 2023 р

Київ-2023

Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Кафедра мережних технологій факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри мультимедійних систем,

к.ф.-м.н., доц.

_____ Жежерун О.П.

(підпис)

„____”_____2023 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КУРСОВУ РОБОТУ

Слухачеві 3 р.н. Миську Юрію Мирославовичу факультету інформатики

ТЕМА Розподіленні бази знань

Вихідні дані:

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

РОЗДІЛ 1: Теоретична частина

РОЗДІЛ 2: Опис алгоритму, який пропонується в роботі

РОЗДІЛ 3: Опис реалізації програми

Висновки

Список використаної літератури

Додатки

Дата видачі „___” _____ 2023 р. Керівник _____

(підпис)

Завдання отримав _____

(підпис)

Тема: Розподілені бази знань

Календарний план виконання роботи:

№	Назва етапу	Термін виконання	Примітка
1	Отримання теми курсової	03.02.2023	
2	Ознайомлення з науковою літературою	18.02.2023	
3	Розбір предметної області	12.03.2023	
4	Перші спроби реалізації програми	18.03.2023	
5	Написання першого розділу курсової роботи	30.03.2023	
6	Завершення роботи над програмою	25.04.2023	
7	Написання другого і третього розділу курсової	07.05.2023	
8	Перегляд змісту роботи з керівником	10.05.2023	
9	Внесення змін до роботи	12.05.2023	
10	Завантаження курсової роботи	17.05.2023	

Студент Мисько Ю.М..

Керівник Жежерун О. П.

“_____” _____

Зміст

Анотація

У даній курсовій роботі описано бази знань, онтології та проведено паралель баз знань з базами даних. Надано детальний опис використаних технологій, їх переваг та причини застосування.

Крім того, в даній курсовій роботі досліджено використання розподілених баз знань. І досліджено онтологічний підхід отримання даних розташованих на різних вузлах.

Також у роботі описується створення демонстраційного десктоп застосунку, основна функція якого - це доступ до даних під керівництвом онтологій.

В роботі використовується такі технології: Java 19, Java Swing, OWLAPI, ProtegeApi, apache-jena, MySQL, MongoDB, Neo4j, Protege.

Ключові слова: база знань, онтологія, база даних, розподілені бази знань, триплет

Вступ

З урахуванням швидкого розвитку інформаційних технологій, створення ефективних інструментів для зберігання, обробки та поширення знань є важливим завданням. Одним з таких інструментів є бази знань.

Метою даної курсової роботи є дослідження можливостей зберігання та обробки знань у базах знань онтологічного типу та їх порівняння з базами даних та дослідження можливості їх поділу. Дослідження також охопить питання доступу до знань під керівництвом онтологій.

У практичній частині буде створено десктоп застосунок для демонстрації можливостей отримання доступу до розподілених на різних вузлах даних за допомогою онтологій та використання онтологічних знань для отримання цих даних.

Результатом виконання даної роботи буде отримання цінного досвіду у роботі з базами знань та поглиблення розуміння їхньої ролі в інформаційних технологіях.

1. Теоретична частина

1.1. Онтології та бази знань

Слово "онтологія" було введене вперше в роботі Томаса Грубера. За його трактуваннями, онтологія - це формальна специфікація концептуальних знань, яка допомагає створювати явний та стандартизований опис області знань[1] . Томас Грубер виділяє такі основні властивості онтологій:

- онтології створюються за допомогою спеціальної формальної мови, в якій фіксуються узгоджені домовленості між групою експертів про те, як називати речі та які властивості вони мають у певній галузі знань;
- кожна онтологія має свою власну логічну структуру, яка складається зі сигнатури та аксіом (логічних тверджень), а іноді - фіксованої моделі (набору об'єктів та зв'язків між ними);
- онтології зазвичай створюються за принципом модульності, тобто при створенні нової онтології можна використовувати вже існуючі.

Онтології використовують для опису знань та розуміння зв'язків між поняттями(об'єктами) предметної області. У галузі інформаційних технологій, онтології знайшли застосування в таких сферах як:

- семантичний веб: використовуються для створення зв'язків між веб-ресурсами, що дозволяє більш ефективно шукати інформацію в Інтернеті;
- розробка програмного забезпечення: використання онтологій допомагає розробникам програмного забезпечення більш ефективно структурувати знання та забезпечити їх більш точну інтерпретацію;
- медична діагностика: використовуються для створення систем медичної діагностики, що дозволяє швидше та точніше визначати діагнози та плани лікування;

- електронна комерція: допомагають структурувати та класифікувати продукти та послуги, що покращує пошук та порівняння продуктів, а також використовуються для підвищення ефективності рекомендацій.

Також онтології допомагають покращити збір і структурування знань, та сприяють автоматичній обробці та аналізу цих знань. Прикладами найвідоміших онтологій є:

- OntoBio - це онтологія, яка описує біологічні об'єкти та їх взаємозв'язки. Вона використовується для інтеграції даних про гени, білки, хімічні сполуки та біологічні процеси з різних джерел [2].
- SUMO (Suggested Upper Merged Ontology) - це онтологія, яка містить загальні поняття та відносини між ними для різних областей знань. Вона використовується як основа для розробки інших онтологій та для підтримки автоматичної обробки текстів [3].
- Cyc - це онтологія, яка описує знання про світ навколо нас. Вона містить поняття про об'єкти, події, відносини та дії. Вона використовується для розробки систем штучного інтелекту та для підтримки прийняття рішень в різних областях[4].

База знань. Існує безліч різних тлумачень, що таке база знань і чим вона відрізняється від онтології. Одне з визначень: “Онтологія описує домен, тоді як база знань (на основі онтології) описує конкретний стан справ”[5]. Іншими словами база знань - це онтологія, яка містить і екземпляри об'єктів.

У більш загальному розумінні, база знань може бути розглянута, як сукупність знань про деяку предметну область, яка зберігаються та використовуються для підтримки прийняття рішень. Бази знань можуть також включати інтерфейси користувача, що дозволяють взаємодіяти з даними та отримувати різноманітну інформацію на основі запитів.

На відмінну від онтологій, які більш орієнтовані на структуру та семантику понять та відносин між ними, бази знань зосереджені на збереженні та

використанні знань у практичних цілях, таких як автоматизація процесів, підтримка прийняття рішень та навчання систем.

1.2. Семантична павутина. Мови опису онтологій

W3C визначає поняття семантичної павутини, як мережу пов’язаних даних. Основна мета цієї мережі - це створення технологій, які забезпечать комп'ютерам здатність виконувати більш корисну роботу та створити системи, що можуть забезпечувати надійну взаємодію в мережі. Ці технології дозволяють зберігати дані у мережі, створювати словники та встановлювати правила обробки даних[6].

У семантичному вебі, онтології грають важливу роль, вони забезпечують формалізований спосіб опису знань, що дозволяє комп'ютерам розуміти значення та зв'язки між різними об'єктами в Інтернеті. Наприклад, в сфері електронної комерції за допомогою онтологій можна автоматично класифікувати товари, зрозуміти їх характеристики та властивості, що полегшує процес пошуку та порівняння товарів. Також онтології можуть покращити результати пошукових видач: пошукові системи можуть шукати ключові слова не тільки у контенті веб ресурсів, а і у метаданих, описаних онтологіями.

Саме ідея семантичної павутина стала поштовхом для створення таких мов опису онтологій, як OWL, RDF і мови запитів SPARQL.

RDF - це стандарт для обміну даними в Інтернеті, який допомагає збирати та об'єднувати дані, що мають різні формати. За допомогою унікальних ідентифікаторів (URI), RDF визначає зв'язки між різними об'єктами. Це дозволяє змішувати структуровані та напівструктуровані дані, відображати та спільно використовувати різні дані, утворюючи зв'язну структуру для обміну даними. Основним елементом мови RDF є трійка або триплет, що складається з трьох сутностей: суб'єкта, об'єкта та предиката [6].

Мова веб-онтології W3C (OWL) призначена для представлення складних знань про речі, їх групи та зв'язки між ними в семантичному вебі. OWL - це мова,

яка базується на обчислювальній логіці, що дозволяє комп'ютерним програмам використовувати знання, виражені в OWL, для перевірки їх узгодженості та виявлення неявних знань. OWL-документи, які називають онтологіями, можуть бути опубліковані в Інтернеті та містити посилання на інші онтології OWL або на них посилатися [6].

SPARQL (SPARQL Protocol and RDF Query Language) - це мова запитів та протокол для роботи з даними в форматі RDF, розроблений як стандарт W3C для роботи з даними семантичного вебу [6]. SPARQL є потужним інструментом для роботи з онтологіями, який дозволяє виконувати різноманітні запити, включаючи пошук даних, фільтрування та сортування результатів, визначення зв'язків між об'єктами. Крім того, SPARQL підтримує як прямі, так і зворотні посилання, що дозволяє знаходити взаємозв'язки між різними об'єктами.

Саме ці три описані вище інструменти стали ключовими у розробці практичної частини роботи.

1.3. Розподіленні бази знань

Оскільки поняття розподілених баз знань тісно пов’язане з розподіленими базами даних, потрібно спочатку розглянути, що ж таке бази даних та що спільного і відмінного вони мають з базами знань.

База даних (БД) - це колекція даних з певною структурою, яка використовуються для зберігання та організації даних.

Розподілена база даних - це сукупність БД розподілених на різних вузлах, які співпрацюють через мережу. Така БД має певні переваги перед не розподіленою:

- можливість маштабування обсягу даних завдяки збільшенню кількості вузлів;

- можливість резервного копіювання та реплікації даних на різних вузлах мережі, що забезпечує збереження даних у разі відмови системи;
- можливість оптимізації виконання запитів до бази даних

Існує багато різних способів поділу баз даних, найпоширенішими є:

- розшарування - розділення даних на декілька фізичних рівнів. Кожен рівень має свою власну базу даних і свій власний набір даних. Цей вид поділу може бути корисним, коли даних дуже багато і їх неможливо зберегти на одному фізичному пристрої;
- реплікація - створення копій баз даних на різних серверах. Цей вид поділу може бути корисним, коли потрібно забезпечити доступ до даних в різних частинах світу або коли потрібно забезпечити високу доступність даних[7].

Обидва ці способи поділу передбачають можливість використання різних типів баз даних і користуватись їхніми перевагами.

Бази знань на основі онтологій є розподіленні за своєю суттю, оскільки зберігають знання про певну предметну область і однією з основних властивостей онтолій є їх модульність, тобто при створенні нової онтології можна використовувати вже існуючі для цього часто потрібно лиш додати новий URI.

Навідмінну від бази даних база знань виконує функції не тільки зберігання даних, а також містить в собі певну логіку, яку наприклад може використовувати для автоматичної генерації нових знань. З іншої сторони зберігати великі об'єми даних у БЗ часто є не дуже практично і з цією задачею краще справляється саме БД.

Наступні розділи описуватимуть практичну частину роботи, де досліджено саме можливість комбінування різного типу розподілених баз даних з базами знань

2. Опис алгоритму, який пропонується в роботі

2.1. Загальний опис програми

Основним завданнями моєї курсової роботи було дослідити розподіленні бази знань. В ході написання теоретичної частини, я дійшов висновку, що часто зберігати великі обсяги даних у базі знань недоцільно, адже з цією метою набагато краще справляється звичайна база даних. Тому було прийнято рішення дослідити саме можливість скомбінувати всі переваги бази знань і бази даних, і реалізувати демонстраційний застосунок.

Отже, основна задача застосунку - це, під керівництвом онтологій, отримати дані різного типу БД.

Було реалізовано графічний інтерфейс, через який користувач може під'єднати реляційну, документну та графову бази даних. А також завантажити онтологію з знаннями про дані цих БД.

Графічний інтерфейс також дозволяє виконувати запитит до всіх баз даних за допомогою мови запитів SPARQL. Це суттєво полегшує роботу з базами даних різного типу, адже не потрібно писати запити на трьох різних мовах, це також вирішує питання об'єднання результатів запитів.

Також програма дозволяє відфільтровувати таблиці, переглядати їх поля та назви, лінкувати поля з різними назвами, та багато іншого.

Для демонстрації роботи програми було створено три тестові бази даних, які містять колекції з інформацією про IT-спеціалістів. Також було створено тестову онтологію з певними знаннями цієї ж предметної області.

База даних традиційного типу була створена за допомогою MySQL. MySQL є системою керування реляційною базою даних з відкритим вихідним кодом, яка використовує мову структурованих запитів (SQL) для управління та обробки даних [9]. MySQL є дуже популярною системою з широкою спільнотою розробників та користувачів , її застосовують світові лідери індустрії, такі як Facebook, Twitter та YouTube, для зберігання та обробки великих обсягів даних.

За допомогою MySQL була створена таблиця та заповнена тестовими даними про IT спеціалістів. Схема, яка описує таблицю з IT спеціалістами зображена на малюнку 1.

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
name	varchar(255)	NO		NULL	
job_title	varchar(255)	NO		NULL	
programming_languages	varchar(255)	NO		NULL	
certifications	varchar(255)	YES		NULL	
years_of_experience	int	YES		NULL	
email	varchar(255)	NO		NULL	
phone_number	varchar(20)	YES		NULL	

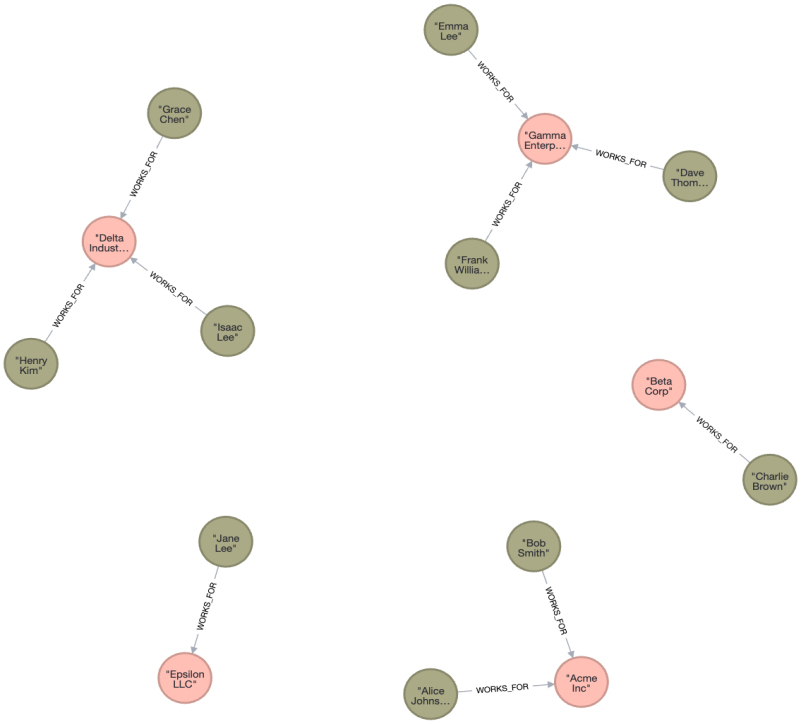
Малюнок 1. Схема MySQL таблиці

Документоорієнтовану базу даних я обрав MongoDB, через її простоту та гнучкість. MongoDB - це документоорієнтована база даних, що зберігає дані у вигляді документів у форматі BSON (Binary JSON) [10]. За допомогою MongoDB була також створена колекція з IT спеціалістами та заповнена тестовими даними. На малюнку 2 наведено схематичне зображення колекції з IT спеціалістами у MongoDB.

it_specialists		
_id	objectId	NN
name	string	NN
email	string	NN
position	string	NN
salary	double	NN
department	string	NN
location	string	NN
hireDate	string	NN
skills[ ]	string	NN
certifications[ ]	string	NN
projects[ ]	string	NN
seniority	string	NN

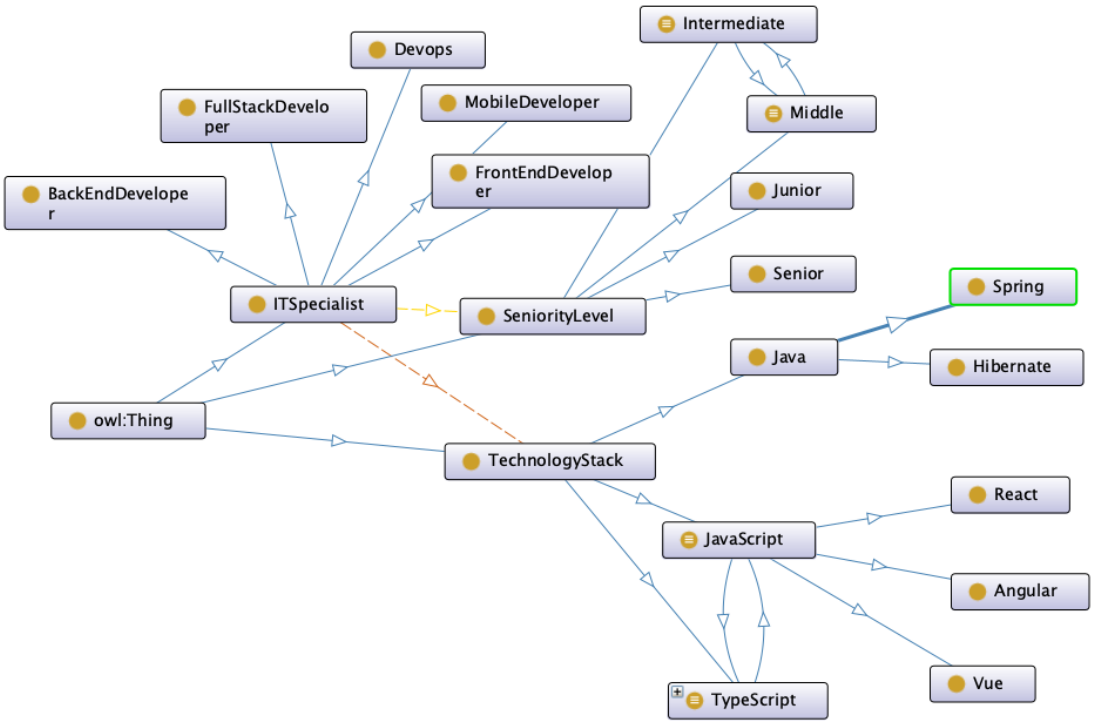
Малюнок 2. Схема MongoDB колекції

В якості бази даних графового типу була використана Neo4j. Звісно Neo4j, як і будь яка інша графова база даних, найоптимальніше підходить для зберігання та обробки даних зі складними зв'язками між ними, але для демонстрації роботи програми був створений досить простий граф: IT спеціалістів та компаній в яких вони працюють. Схематичне зображення графів зображене на малюнку 3.



Малюнок 3. Схема Neo4j графів

Оскільки в базах даних зберігаються тільки дані, нам потрібно додати ще і певні знання, для цього в редакторі онтологій Protege було створено онтологію з знаннями про про IT спеціалістів. Protege - це безкоштовний редактор онтології з відкритим вихідним кодом і структура для створення інтелектуальних систем[8]. Графічне зображення онтології зображене на малюнку 4.



Малюнок 4. Онтологія IT спеціалістів

2.2. Загальний опис алгоритму та огляд існуючих аналогів

Для реалізації даної програми, спочатку потрібно було вирішити питання, об'єднання даних розподілених на різного типу базах даних з онтологією. Щоб досягнути такого результату, потрібно спершу привести всі частини системи до спільного типу(формату). Тож я вирішив перетворювати дані з різних БД у RDF формат, а потім об'єднати їх з вхідною онтологією.

Зі схожими задачами добре справляється плагін Ontop та бібліотека OWL API, які є частиною системи Protege. Ontop використовує відомий алгоритм Relational Databases to RDF Mapping[11]. Основна ідея цього алгоритму полягає в тому, що кожен елемент реляційної бази даних (таблиця, колонка, рядок) може бути представлений як об'єкт RDF. Атрибути (стовпці) реляційної бази даних можуть бути представлені, як властивості цього об'єкта. Кожен рядок в таблиці може бути представлений як окремий елемент RDF, який має властивості, що відповідають значенням атрибутів для цього рядка.

Однак Ontop розрахований для роботи тільки з реляційними базами даних. Тому було прийняте рішення на основі алгоритму RDF Mapping, розробити власні, які для конвертації документо орієнтованої та графової баз даних. А після конвертації, обєднати їх в одну модель і використовуючи їні унікальні ідентифікатори, надати користувачеві можливість робити SPARQL запити та отримувати інформацію з всіх підєднаних джерел даних одночасно. Вся ця логіка виконується в фоновому режимі, і надає користувачеві зручний інтерфейс для роботи з програмою.

Звичайно нам також потрібно було подумати про консистентність даних. Адже дані в різних джерелах інформації дані можуть мати різні назви чи рівні абстрактності. І як вирішення цієї проблеми, програма пропонує користувачеві додати додаткові знання (онтологію). За допомогою цієї онтології програма отримує інформацію про певні зв'язки між даними. Наприклад, з онтології можна дізнатись, що якісь поняття є ідентичними, чи що одне поняття наслідує інше.

3. Опис реалізації програми

3.1. Технічні засоби

Основною мовою програмування використаною у застосунку є Java 19. Мій вибір впав саме на Java, через її широку підтримку різних білбліотек для роботи з антологіями. Такими як: OWLAPI, ProtegeApi, apache-jena та інші. Також Java є досить простою мовою програмування з чіткою синтаксичною структурою та багатьма вбудованими функціями, що дозволяє швидко та зручно створювати кроссплатформені додатки.

Як середовище розробки, я вибрав IntelliJ IDEA, тому що IntelliJ IDEA може легко інтегруватися з іншими інструментами розробки, такими як Maven, Gradle. А також має дуже зручний графічний інтерфейс з різними вбудованими функкціями.

Основною бібліотекою для роботи з онтологіями, я обрав apache-jena. Основні можливості Apache Jena включають:

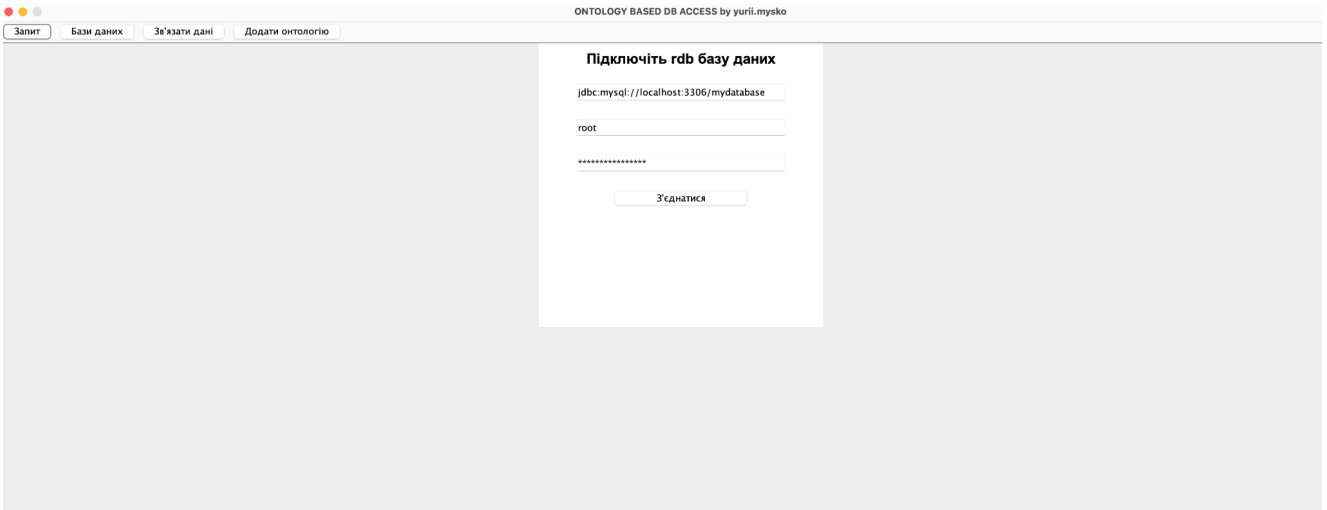
- робота зі схемами знань RDF (OWL, RDFS);
- збереження та завантаження даних у форматі RDF;
- виконання запитів SPARQL;
- створення та редагування триплетів RDF.

Графічний інтерфейс був реалізований за допомогою Java Swing. Java Swing - це бібліотека з набором інструментів для розробки графічного інтерфейсу користувача у мові програмування Java. Також ця бібліотека має великий набір готових компонентів, таких як кнопки, поля введення, таблиці та інші.

Також для роботи з базами даних були використанні драйвери такі як: mongodb, neo4j.driver, java.sql. За допомогою цих драйверів створюється з'єднання з базами даних та виконуються запити.

3.2. Реалізація алгоритму та інтерфейсу програми

Перед початком роботи з програмою користувачеві необхідно під'єднати свої бази даних. Для цього було створено три різні вікна з можливістю ввести дані для підключення. Вигляд цих вікон зображеней на малюнку 5.



Малюнок 5. Вікно підключення бази даних

Після того, як користувач з’єднається з базою даних, створюється один з трьох класів: MySQL, MongoDB, Neo4j, в залежності від типу бази даних. Всі ці класи мають один інтерфейс DB але різну імплементацію. Його сигнатура зображена на малюнку 6.

```
public interface DB {
    void connect(String DB_URL, String DB_USER, String DB_PASS);
    void close();
    boolean isConnected();
    String[] getCollections();
    String[] getCollectionProps(String collectionName);
    String getCatalog();
    String getUserName();
    Model convertToRDF(String data);
}
```

Малюнок 6. Спільний інтерфейс баз даних

За перетворення даних з колекцій MySQL, MongoDB, Neo4j - відповідають класи RDBToRdfConverter, ODBToRdfConverter, GDBToRdfConverter. Вони мають різну реалізацію, але загальнинй алгоритм схожий. На малюнку 7 зображений метод перетворення даних з колекціїї MongoDB у RDF. RDF триплет створюється наступним чином: з ідентифікатора об'єкта створюється суб'єкт, в ході ітерації по властивостях створюються предикати, та значення цих властивостей перетворюється в об'єкт.

```
public Model convert(FindIterable<Document> documents, String databaseName) {
    RDFModel rdf = new RDFModel(tableType, databaseName);
    Model model = rdf.getModel();
    for (Document document : documents) {
        Set<Map.Entry<String, Object>> entries = document.entrySet();
        for (Map.Entry<String, Object> entry : entries) {
            String key = entry.getKey();
            Object value = entry.getValue();

            Resource subject = model.createResource(document.get(COLLECTION_ID).toString());
            Property predicate = model.createProperty(rdf.getNS(), key);
            RDFNode object = model.createLiteral(value.toString());
            model.add(subject, predicate, object);
        }
    }
    return model;
}
```

Малюнок 7. Метод перетворення MongoDB колекції у RDF

Щоб додати онтологію до програми користувач може скористатись відповідною кнопкою верхньої панелі. Після цього створюється клас OWLModel, його інтерфейс зображений на малюнку 8. Цей клас відповідальний за зчитування файла з онтологією та створення онтологічної моделлі. Він має методи, які використовують jena-reasoner для того, щоб отримати додаткову інформацію про якийсь об'єкт онтології. Наприклад ідентичні значення, суб класи та об'єкти класу та інші.

```
public interface OWLModelInterface {
    OntModel createOntModel(String path);
    OntModel getModel();
    String getOntologyFileName();
    String getURI();
    List<Resource> getSameAsNodes(String nodeID);
    List<Resource> getIndividualsAndSubClasses(String nodeID);
    List<String> getAllPossibleSameValues(String nodeId);
}
```

Малюнок 8. Інтерфейс класу OWLModel

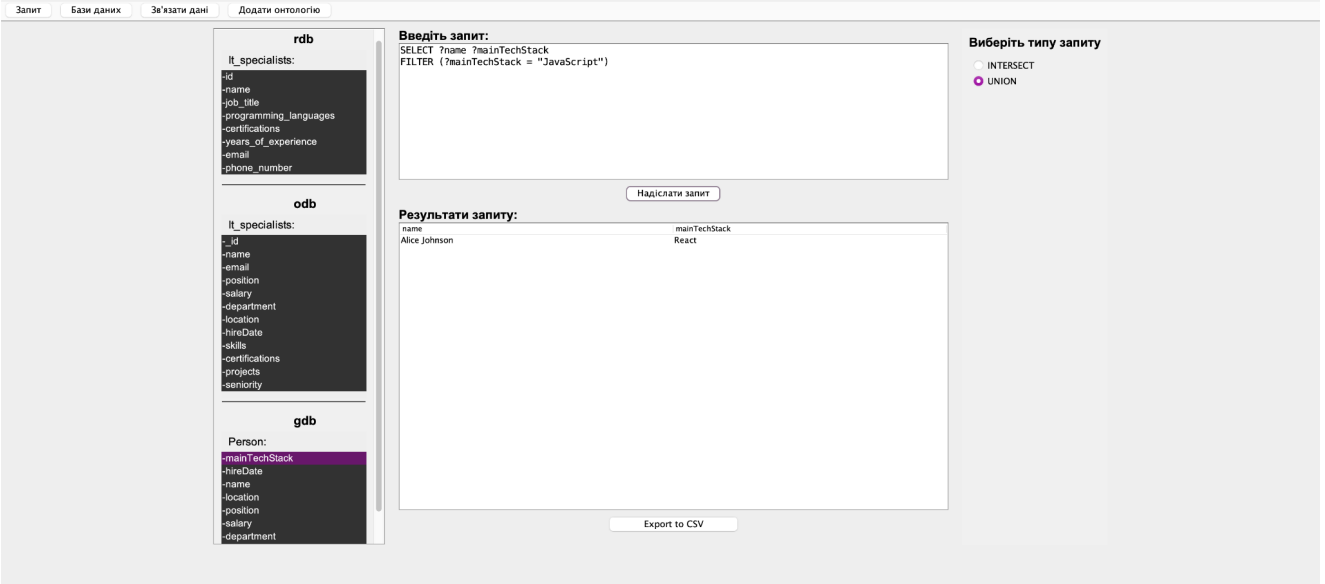
Для того, щоб отримати потрібну інформацію користувач має скористатись вікноп “Запит”. Це вікно зображена на малюнку 9. Зліва зображені таблиці з якими користувач хоче працювати і їх атрибути. По центру є поле для вводу SPARQL запиту. Користувачеві не потрібно вводити частину запиту “Where…”, це програма робить замість нього. Також користувач може додати до запиту певну фільтрацію, яка використовуватиме знання з онтології.

Програма також пропонує користувачеві вибрати режим запиту “Union” або “Intersection”. І автоматично трансформує запит під потрібний режим.

Операція UNION узгоджує результати двох або більше запитів в один набір результатів. Якщо окремі запити повертають значення змінних, які перетинаються, то в об'єднаному наборі результатів будуть міститися всі різні значення цих змінних.

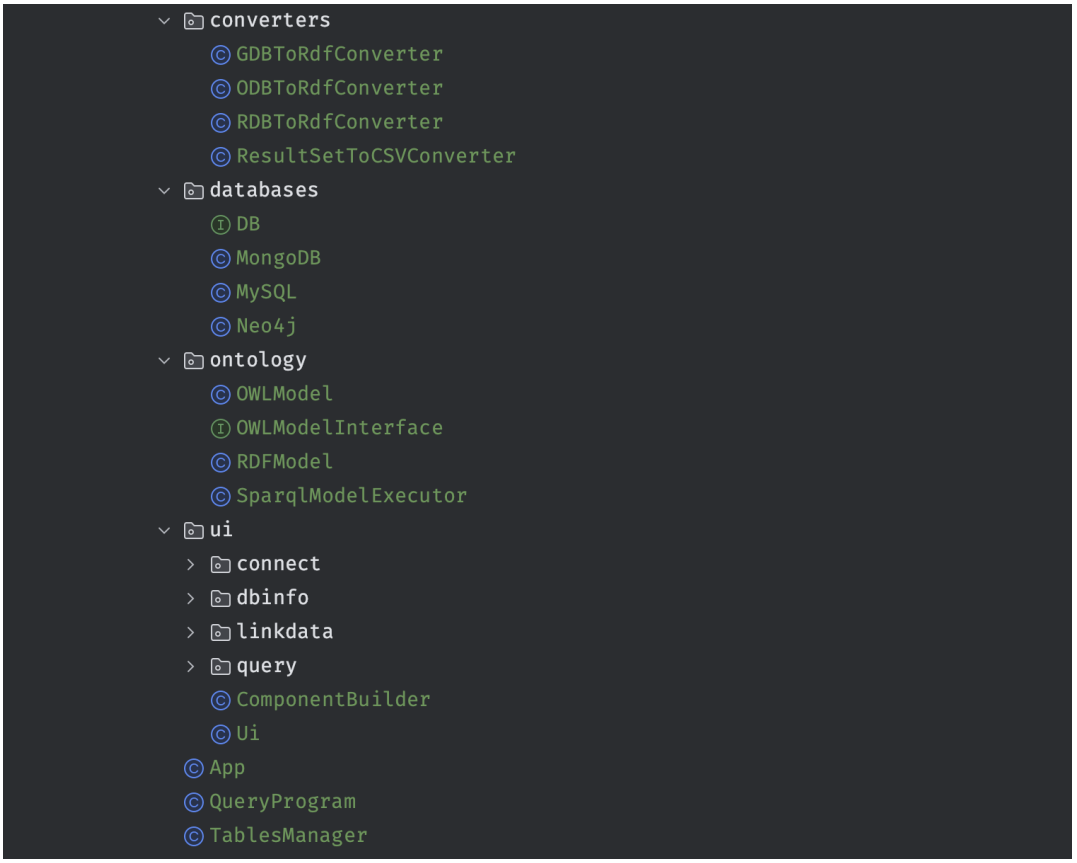
Тоді як, INTERSECTION об'єднує результати двох або більше запитів, шляхом пошуку спільних елементів. Це означає, що кінцевий набір результатів буде містити тільки ті результати, які зустрічаються в усіх запитах одночасно.

Після того, як користувач відправить запит, знизу буде доступна таблиця з результатами цього запиту. Цю таблицю користувач може експортувати в “CSV” форматі.



Малюнок 9. Головне вікно програми

За виконання запитів, генерацію правильного тексту запиту та використання знань онтології відповідає клас QueryProgram. А загальна структура класів зображена на малюнку 10.



Малюнок 10. Загальна структура класів

Висновки

Отже, в цій роботі я ознайомився з поняттями онтологій та баз знань онтологічного типу та провів порівняння з базами даних. Також дослідив можливість поділу баз знань.

В ході виконання роботи я зрозумів, що бази знань онтологічного типу є розподіленими за своєю суттю, адже однією з основних властивостей онтолій є їх модульність, тобто при створенні нової онтології можна використовувати вже існуючі для цього потрібно лиш додати новий URI.

Також я дійшов висновку, що часто зберігати великі обсяги даних у базі знань недоцільно, оскільки з цією метою набагато краще справляються звичайні бази даних. Тому було прийнято рішення дослідити саме можливість скомбінувати всі переваги бази знань і бази даних, і дослідити можливість доступу до розподілених баз даних під керівництвом онтологій. При ознайомленні з можливостями зберігання знань у базах даних та їх поділу, стало зрозуміло, що для даних різного типу підходять різного типу бази даних. Але отримувати дані з БД різного типу - є не простою задачею, адже для цього потрібно використовувати різні мови запитів, вирішити питання консистентності даних і їх злиття. Для вирішення цих проблем я спробував скористатись онтологічним підходом.

Тож, у практичній частині я створив десктоп застосунок, в якому постарався продемонструвати, як за допомогою онтологій можна отримати доступ до розподілених на різних вузлах даних. А також використати онтологічні знання для отримання цих даних.

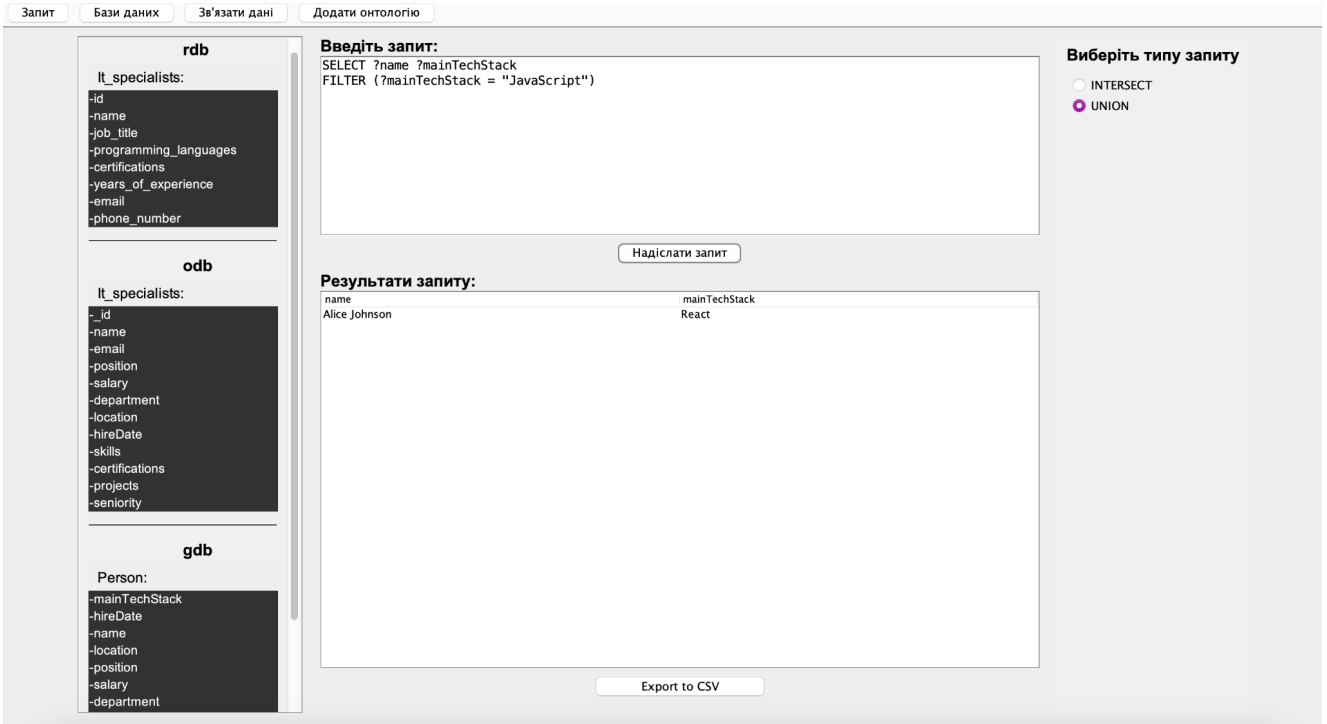
Загалом, дана курсова робота дозволила зрозуміти, що бази знань онтологічного типу є потужним інструментом для зберігання та обробки знань. Окрім того, в роботі було продемонстровано можливості поєднання баз знань та баз даних за допомогою онтологічного підходу. В цілому, виконання даної роботи дало змогу отримати цінний досвід у роботі з базами знань та поглибити розуміння їхньої ролі в інформаційних технологіях.

Список використаної літератури

1. Thomas R. Gruber / The Role of Common Ontology in Achieving Sharable, Reusable Knowledge Bases / A translation approach to portable ontologies
2. [Електронний ресурс] - https://ontobio.readthedocs.io/en/latest/
3. [Електронний ресурс] - https://www.ontologyportal.org/
4. [Електронний ресурс] - https://cyc.com/
5. [Електронний ресурс] - https://www.gabormelli.com/RKB/Ontological_Knowledge_Base#:~:text=Ontology%20describes%20a%20domain%2C%20while,used%20in%20the%20knowledge%20base.
6. [Електронний ресурс] - https://www.w3.org
7. Kleppmann, Martin / Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems
8. [Електронний ресурс] - https://protege.stanford.edu/
9. [Електронний ресурс] - https://www.mysql.com/products/enterprise/
10. [Електронний ресурс] - https://www.mongodb.com/
11. [Електронний ресурс] - https://www.w3.org/TR/rdb-direct-mapping/

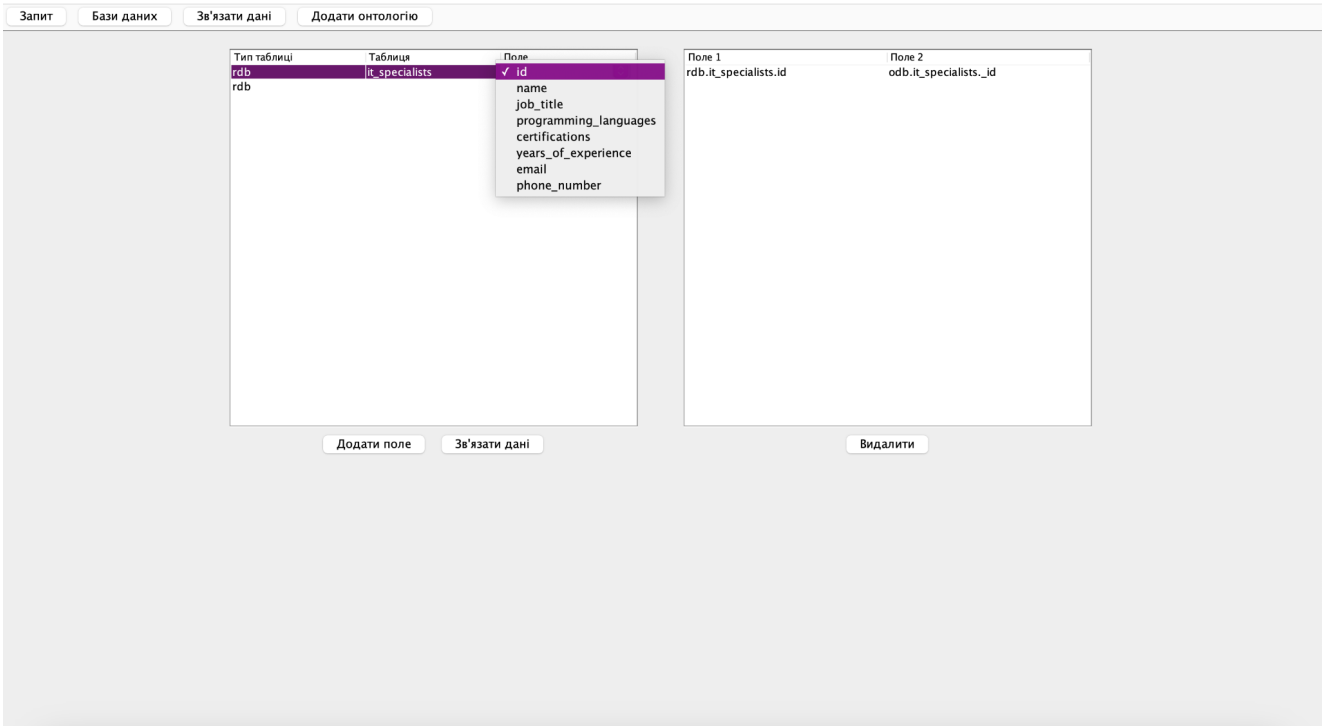
Додаток А

Результат виконання запиту з фільтрацію ?mainTechStack = "JavaScript". Повернуте значення “React” оскільки онтологія містить знання, що “React” є субкласом "JavaScript".



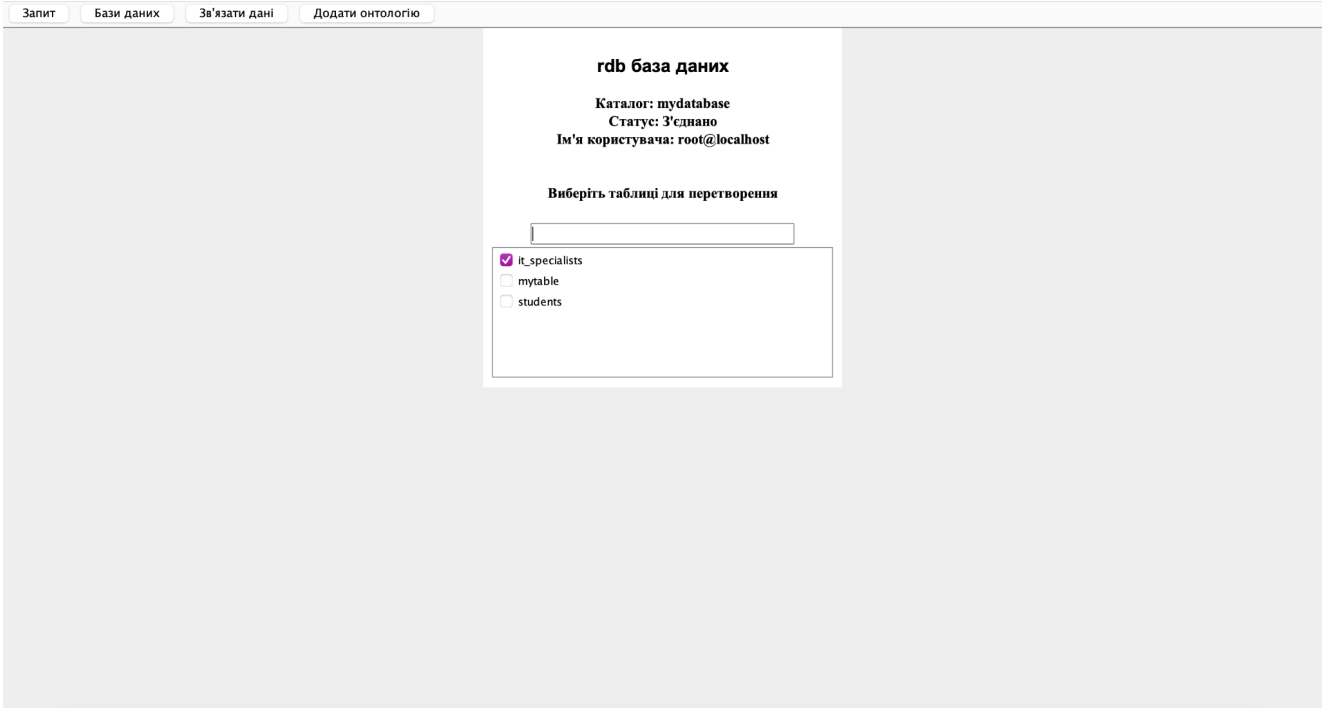
Додаток Б

Графічний інтерфейс для введення додаткової інформації про дані колекцій, таких як ідентичні дані з різними назвами, у програмний спосіб



Додаток В

Вікно з інформацією про під'єднану базу даних, з можливістю філтрації, пошуку та вибору потрібних колекцій



Додаток Г

Результат виконання синтаксично не правильного запиту