НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КУРСОВИЙ ПРОЄКТ

з дисципліни «Організація сховищ даних»

на тему:

БАЗА ЗНАНЬ ЯК СХОВИЩЕ ДАНИХ ІНТЕЛЕКТУАЛЬНОГО АГЕНТА

Виконав: _____/ Харченко Ю. Б. /
підпис

Керівник курсового проєкту: _____/_____/
підпис

Робота захищена:

« ___ » _____ 2022 р.

Оцінка за національною шкалою: _____

Кількість балів: _____

Члени комісії: _____ _____
(підпис) (прізвище та ініціали)

_____ _____
(підпис) (прізвище та ініціали)

КИЇВ – 2022

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

DDW - багатовимірне сховище даних (dimensional data warehouse)

DW - сховище даних (data warehouse)

ELT - витяг, завантаження, перетворення (extract, load, transform)

ETL - витяг, перетворення, завантаження (extract, transform, load)

IA - інтелектуальний агент (intelligent agent)

KB - база знань (knowledge base)

KR - представлення знань (knowledge representation)

OLAP - аналітична обробка у реальному часі (on-line analytical processing)

SN - семантична мережа (semantic network)

SW - семантичний веб (semantic web)

ВСТУП

Історично в підходах до проєктування та побудови DW простежуються два головних напрямки започатковані основоположниками галузі Ральфом Кімболом (Ralph Kimball) [1] та Біллом Інмоном (Bill Inmon) [2].

Підхід Кімбола слідує висхідному підходу до проектування архітектури DW, у якому спочатку формуються вітрини даних. Потім використовується інструмент ETL, щоб отримати дані з кількох джерел і завантажити їх у проміжну область реляційної бази даних. Наступний етап включає завантаження даних у DDW, денормалізоване за своєю природою. Процес розбиває дані на таблицю фактів, які є числовими транзакційними даними, та таблицю вимірів, які є довідковою інформацією, що визначає факти. Щоб інтегрувати дані, цей підхід передбачає узгодження вимірів даних. Це гарантує, що окремий елемент даних визначається однаково для всіх фактів [3].

Інмон пропонує концепцію розробки DW, яке визначення предметної області та об'єктів, з якими працює підприємство, наприклад клієнтів, продуктів, постачальників тощо. Він визначає DW як «предметно-орієнтований, інтегрований, незмінний набір даних, що підтримує хронологію та організований для цілей підтримки управління». Створюється логічна модель для кожної первинної сутності з усіма атрибутами, пов'язаними з цією сутністю. Ця логічна модель може включати всі деталі, аспекти, відносини, залежності та зв'язки. Перевага цього низхідного підходу полягає в тому, що DW діє як єдине джерело істинності для всієї предметної області, всі дані інтегровані [3].

Білл Інмон та його компанія розробили технологію, відому як «текстове усунення неоднозначності». Ця технологія застосовує контекст до необробленого тексту та переформатує його у стандартний формат бази даних. Текстове усунення неоднозначності здійснюється за допомогою виконання спеціального текстового ETL/ELT. Воно може бути застосовано скрізь, де є необроблений текст, наприклад, у документах, Hadoop, електронній пошті тощо. Вводиться поняття семантичного шару, представлення корпоративних даних через загальні терміни. Семантичний шар відображає складні дані у визначені терміни та робить уніфіковане консолідоване уявлення про дані всієї організації [4] [5].

Методологію розробки DW, яка інтегрує дані з різних джерел шляхом поєднання технологій DW та онтології розглядають в [6], де кожне джерело може мати свою локальну онтологію, яка посилається на глобальну, з можливістю її спеціалізації чи розширення.

Поєднання технологій DW, OLAP та SW розглядається в [7]. Технології SW застосовуються для моделювання та представлення даних, включаючи семантичну анотацію і процеси ETL з урахуванням семантики.

В [8] пропонується оригінальна методологія побудови сховища текстових даних і виконання операцій OLAP над текстовими даними після категоризації текстових документів в ієрархії понять шляхом фіксації контекстуальної подібності між текстовими документами.

Проєкт КВ для ІА слідує шляхом запропонованим Біллом Інмоном, що дозволяє уникнути надмірності даних завдяки нормалізованій формі структури сутностей та спростити процедури оновлення і формування запитів використовуючи єдину логічну модель.

Метою проєкту є дослідження підходів та методів створення КВ ІА на основі SM, яка відображається в реляційній моделі даних.

Об’єктом дослідження є ІА заснований на знаннях в сенсі визначення даного в [9].

Предметом дослідження є KR в ІА в формі SM, як математичної моделі концептуальної структури, що складається з набору понять і когнітивних зв’язків між ними. SM представляється узагальненим графом, де поняття та сутності відповідають вузлам графа, а зв’язки між поняттями – дугам [10].

Досягнення мети проєкту складається з виконання таких завдань:

- формулювання математичної моделі структури SM;
- створення на основі математичної моделі реляційної моделі даних;
- імплементація реляційної моделі для конкретної бази даних;
- розробка системи запитів до побудованої SM засобами логічного програмування;
- виконання тестового прикладу на побудованій системі.

1 МАТЕМАТИЧНА МОДЕЛЬ

KB = <V, C, O, CO, R, RC, RCO, A, AC, AR, ARC, ACO, ARCO>,

де KB - база знань,

V - множина зареєстрованих значень (Values),

C - множина категорій (Categories),

O - множина об’єктів (Objects),

CO - множина пар <c ∈ C, o ∈ O>,

R - множина відношень (Relations),

RC - множина трійок <r ∈ R, c_{from} ∈ C, c_{to} ∈ C>,

RCO - множина трійок <rc ∈ RC, o_{from} ∈ O, o_{to} ∈ O>,

A - множина атрибутів (Attributes),

AC - множина трійок <c ∈ C, a ∈ A, v ∈ V>,

AR - множина трійок <r ∈ R, a ∈ A, v ∈ V>,

ARC - множина трійок <rc ∈ RC, a ∈ A, v ∈ V>,

ACO - множина трійок <co ∈ CO, a ∈ A, v ∈ V>,

ARCO - множина трійок <rco ∈ RCO, a ∈ A, v ∈ V>.

2 РЕЛЯЦІЙНА МОДЕЛЬ

Кожна множина математичної моделі представлена таблицею в реляційній моделі даних (рис. 1). Кожна таблиця має сурогатний автоматично генерований первинний ключ - id.

Опис інших полів наведено в таблиці 1.

Таблиця 1

Поле	Тип	Зовнішній ключ	Унікальний індекс	Призначення
V - таблиця значень				
value	TEXT	-	value	Унікальний екземпляр значення
count	INTEGER	-	-	Лічильник посилань на значення
C - таблиця категорій				
v	INTEGER	V(id)	v	Посилання на значення назви
O - таблиця об’єктів				
flag	INTEGER	-	-	Ознака для інтерпретації значення за посиланням v
v	INTEGER	V(id)	v	Посилання на значення зовнішньої ідентифікації (назви, джерела, посилання)
CO - таблиця відношень many-many між об’єктами та категоріями				
c	INTEGER	C(id)	c, o	Посилання на категорію
o	INTEGER	O(id)	c, o	Посилання на об’єкт
R - таблиця відношень				
v	INTEGER	V(id)	v	Посилання на значення назви
RC - таблиця відношень many-many між парою категорій та відношеннями				
r	INTEGER	R(id)	r, cf, ct	Посилання на відношення
cf	INTEGER	C(id)	r, cf, ct	Посилання на категорію (“звідки”)
ct	INTEGER	C(id)	r, cf, ct	Посилання на категорію (“куди”)
RCO - таблиця відношень many-many між парою об’єктів та таблицею RC				
rc	INTEGER	RC(id)	rc, of, ot	Посилання на RC
of	INTEGER	O(id)	rc, of, ot	Посилання на об’єкт (“звідки”)
ot	INTEGER	O(id)	rc, of, ot	Посилання на об’єкт (“куди”)
A - таблиця атрибутів				
v	INTEGER	V(id)	v	Посилання на значення назви

AC - таблиця відношень many-many між категоріями та атрибутами				
c	INTEGER	C(id)	c, a	Посилання на категорію
a	INTEGER	A(id)	c, a	Посилання на атрибут
v	INTEGER	V(id)	-	Посилання на значення атрибута
AR - таблиця відношень many-many між відношеннями та атрибутами				
r	INTEGER	R(id)	r, a	Посилання на відношення
a	INTEGER	A(id)	r, a	Посилання на атрибут
v	INTEGER	V(id)	-	Посилання на значення атрибута
ARC - таблиця відношень many-many між таблицею RC та атрибутами				
rc	INTEGER	RC(id)	rc, a	Посилання на RC
a	INTEGER	A(id)	rc, a	Посилання на атрибут
v	INTEGER	V(id)	-	Посилання на значення атрибута
ACO - таблиця відношень many-many між таблицею CO та атрибутами				
co	INTEGER	CO(id)	co, a	Посилання на CO
a	INTEGER	A(id)	co, a	Посилання на атрибут
v	INTEGER	V(id)		Посилання на значення атрибута
ARCO - таблиця відношень many-many між таблицею RCO та атрибутами				
rco	INTEGER	RCO(id)	rco, a	Посилання на RCO
a	INTEGER	A(id)	rco, a	Посилання на атрибут
v	INTEGER	V(id)	-	Посилання на значення атрибута

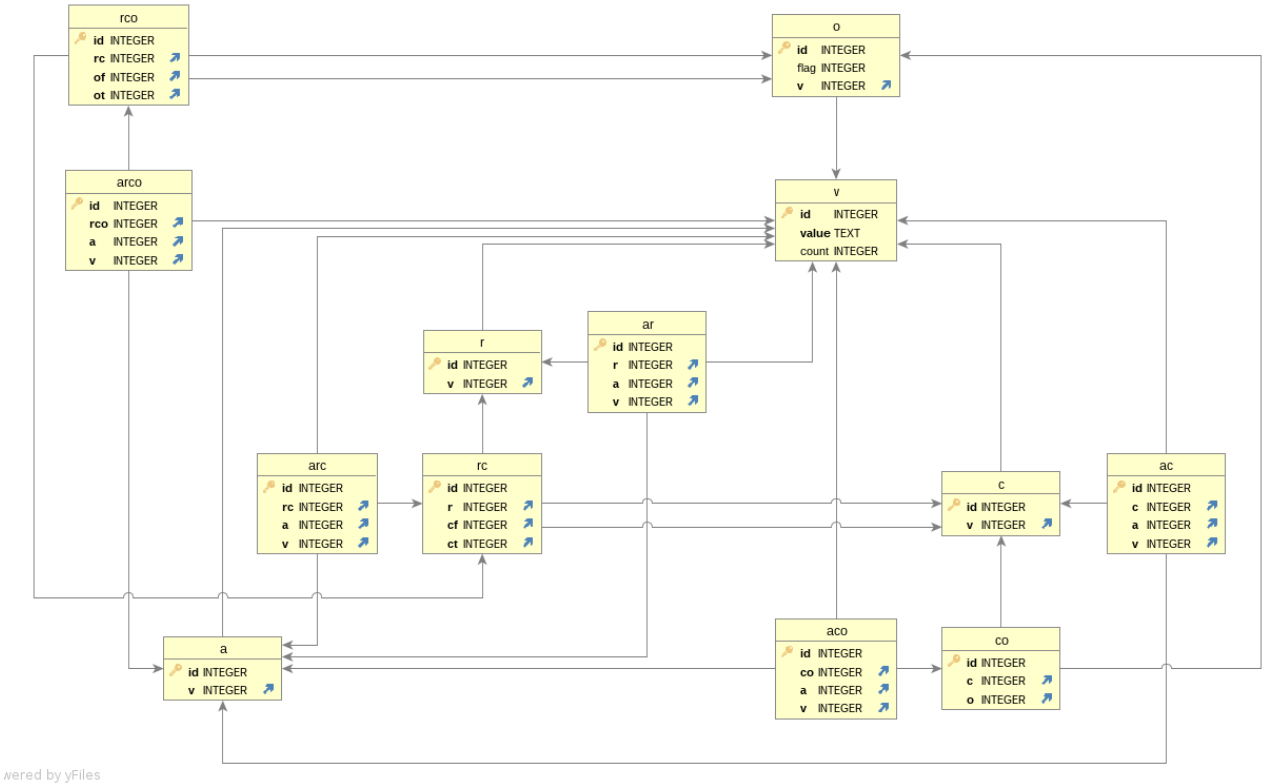

Рисунок 1. Реляційна модель даних SM

3 СЕМАНТИКА МОДЕЛІ

Відображення предметної області на пропоновану в проєкті модель DW продемонструємо на прикладі ІА, який може виконувати функції асистента при роботі з текстовими матеріалами в e-learn. Сценарії предметної області окреслені на рис. 2.

КВ має бути зконфігурована для відображення сутностей предметної області шляхом задання метаданих моделі.

З точки зору взаємовідносин між категоріями частина метаданих може мати такий склад. Категорії (С): “Автор”, “Документ”, “Речення”, “Слово”. Відношення (R): “є автором”, “складається з”. Відношення між категоріями (RC): “Автор”-“є автором”-“Документ”, “Документ”-“складається з”-“Речення”, “Речення”-“складається з”-“Слово”.

Наприклад, відношення між категоріями “Автор”-“є автором”-“Документ” (rc) говорить про те, що від об’єкта (o_{from}), який приєднано до категорії “Автор” (co) може бути встановлено відношення (rco) “є автором” (r) до об’єкта (o_{to}), який приєднано до категорії “Документ” (co).

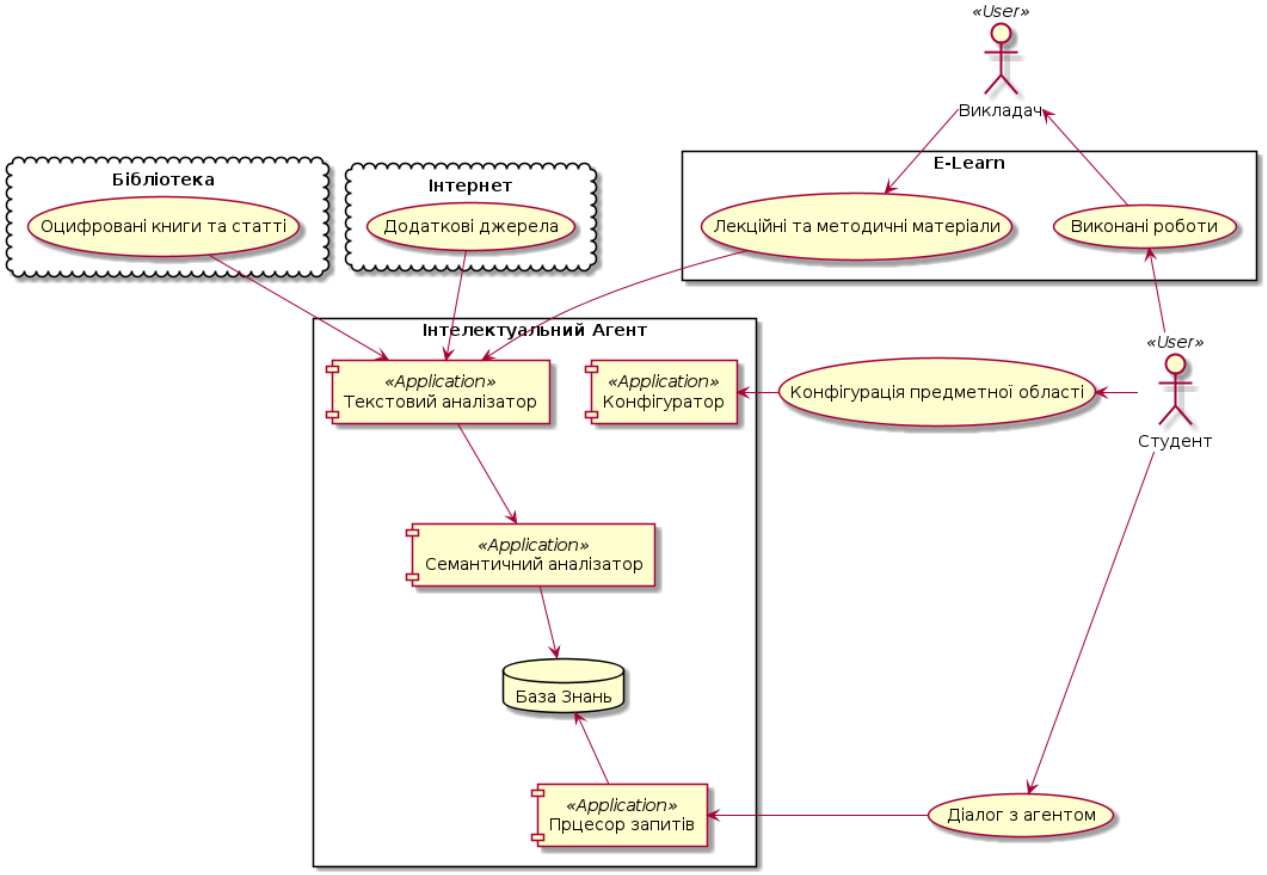

Рисунок 2. Сценарії предметної області

Метадані типу атрибут дозволять приєднувати значення атрибутів до всіх сутностей моделі.

Атрибути (A): “Ім’я”, “Назва”, “Номер”. Атрибути категорій (AC): “Назва”-“Документ”, “Ім’я”-“Автор”. Атрибути відношень (AR): “Номер”-“складається з”.

Наприклад, атрибут категорії “Ім’я”-“Автор” (ac) говорить про те, що об’єкт (o), який приєднано до категорії “Автор” (co) може мати атрибут (a), атрибут відношення “Номер”-“складається з” (ar) говорить про те, що відношення між категоріями (rc) та між об’єктами (rco), можуть мати атрибут (a).

4 ІМПЛЕМЕНТАЦІЯ МОДЕЛІ

Реляційна модель даних може бути реалізована на будь якій системі реляційних баз даних, які підтримують автоматичну генерацію числових первинних ключів. В додатку А наведені твердження запитів SQL для побудови об’єктів бази даних.

5 СИСТЕМА ЗАПИТІВ ЗАСОБАМИ ЛОГІЧНОГО ПРОГРАМУВАННЯ

Для ефективного використання побудованої КВ потрібна система запитів до неї як до SM. В проєкті використані засоби пакету логічного програмування Julog [11] мови програмування Julia [12]. Обґрунтування використання в проєкті мови Julia наведено в додатку Б.

Нижче наведено текст фрагменту програми на Julia, який формує твердження для системи логічного програмування Julog. Ця система подібна до системи Prolog, але реалізована як пакет Julia і дозволяє використовувати всі переваги цієї мови.

```
rules = @julog [
Attr(ID, Name) <<= A(ID, V) & V(V, Name, _),

Cat(ID, Name) <<= C(ID, V) & V(V, Name, _),

Rel(ID, Name) <<= R(ID, V) & V(V, Name, _),

Obj(ID, Name) <<= O(ID, _, V) & V(V, Name, _),

CatObj(ID, CatID, ObjID, CatName, ObjName) <<= CO(ID, CatID, ObjID,) &
Cat(CatID, CatName) & Obj(ObjID, ObjName),

RelCat(ID, RelID, CatFromID, CatToID, RelName, CatFromName, CatToName) <<=
RC(ID, RelID, CatFromID, CatToID) & Rel(RelID, RelName) & Cat(CatFromID, CatFromName)
& Cat(CatToID, CatToName),

RelCatObj(ID, RelCatID, CatFromID, ObjFromID, CatToID, ObjToID, RelName,
CatFromName, ObjFromName, CatToName, ObjToName) <<= RCO(ID, RelCatID, ObjFromID,
ObjToID) & RelCat(RelCatID, RelID, CatFromID, CatToID, RelName, CatFromName,
CatToName) & Obj(ObjFromID, ObjFromName) & Obj(ObjToID, ObjToName),

AttrCat(ID, CatID, AttrID, ValueID, CatName, AttrName, Value) <<= AC(ID, CatID,
AttrID, ValueID) & Attr(AttrID, AttrName) & Cat(CatID, CatName) & V(ValueID, Value, _),

AttrCatObj(ID, CatObjID, CatID, ObjID, AttrID, ValueID, CatName, ObjName,
AttrName, Value) <<= ACO(ID, CatObjID, AttrID, ValueID) & CatObj(CatObjID, CatID, ObjID,
CatName, ObjName) & Attr(AttrID, AttrName) & Cat(CatID, CatName) & V(ValueID, Value,
_),

AttrRel(ID, RelID, AttrID, ValueID, RelName, AttrName, Value) <<= AR(ID, RelID,
AttrID, ValueID) & Attr(AttrID, AttrName) & Rel(RelID, CatName) & V(ValueID, Value, _),
```

AttrRelCat(ID, RelCatID, RelID, CatFromID, CatToID, AttrID, ValueID, RelName, CatFromName, CatToName, AttrName, Value) <<= ARC(ID, RelCatID, AttrID, ValueID) & RelCat(RelCatID, RelID, CatFromID, CatToID, RelName, CatFromName, CatToName) & Attr(AttrID, AttrName) & V(ValueID, Value, _),

AttrRelCatObj(ID, RelCatObjID, RelID, CatFromID, ObjFromID, CatToID, ObjToID, AttrID, ValueID, RelName, CatFromName, ObjFromName, CatToName, ObjToName, AttrName, Value) <<= ARCO(ID, RelCatObjID, AttrID, ValueID) & RelCatObj(RelCatObjID, RelCatID, CatFromID, ObjFromID, CatToID, ObjToID, RelName, CatFromName, ObjFromName, CatToName, ObjToName) & Attr(AttrID, AttrName) & V(ValueID, Value, _)
]

Ці твердження дозволяють робити запити на вибірку з SM в стилі Prolog.

6 ТЕСТОВИЙ ПРИКЛАД

Протокол виконання тестового прикладу наведено в додатку В.

ВИСНОВКИ

В результаті роботи над проєктом були досягнуті такі результати:

- досліджено джерела, які розкривають різні аспекти теорії та практики DW;
- визначено методи та засоби, які можуть бути застосовані при проєктуванні та побудові KB IA як текстового DW;
- створено математичну та реляційну моделі SM;
- розроблено систему логічних правил для запитів до KB;
- виконано тестовий приклад по завантаженню документа лекції в SM.

Процес розробки та тестування можна відслідкувати за посиланням: https://youryharchenko.github.io/m-work/

Вихідні коди скриптів розміщено в репозиторії на Github за посиланням: https://github.com/youryharchenko/m-work/tree/main/labs/DW

Отримані результати апробації запропонованої моделі SM дозволять використати її в подальшому для дослідження, проєктування та реалізації програмних інтелектуальних агентів.

ДОДАТОК А

```
CREATE TABLE IF NOT EXISTS V
(
id INTEGER PRIMARY KEY,
value TEXT NOT NULL UNIQUE,
count INTEGER DEFAULT(0)
)

CREATE TABLE IF NOT EXISTS C
(
id INTEGER PRIMARY KEY,
v INTEGER NOT NULL UNIQUE,
FOREIGN KEY(v) REFERENCES V(id)
)

CREATE TABLE IF NOT EXISTS R
(
id INTEGER PRIMARY KEY,
v INTEGER NOT NULL UNIQUE,
FOREIGN KEY(v) REFERENCES V(id)
)

CREATE TABLE IF NOT EXISTS RC
(
id INTEGER PRIMARY KEY,
r INTEGER NOT NULL,
cf INTEGER NOT NULL,
ct INTEGER NOT NULL,
FOREIGN KEY(r) REFERENCES R(id),
FOREIGN KEY(cf) REFERENCES C(id),
FOREIGN KEY(ct) REFERENCES C(id),
UNIQUE(r, cf, ct)
)

CREATE TABLE IF NOT EXISTS O
(
id INTEGER PRIMARY KEY,
flag INTEGER DEFAULT(0),
v INTEGER NOT NULL UNIQUE,
FOREIGN KEY(v) REFERENCES V(id)
)
```

```
CREATE TABLE IF NOT EXISTS CO
(
id INTEGER PRIMARY KEY,
c INTEGER NOT NULL,
o INTEGER NOT NULL,
FOREIGN KEY(c) REFERENCES C(id),
FOREIGN KEY(o) REFERENCES O(id),
UNIQUE(c, o)
)

CREATE TABLE IF NOT EXISTS RCO
(
id INTEGER PRIMARY KEY,
rc INTEGER NOT NULL,
of INTEGER NOT NULL,
ot INTEGER NOT NULL,
FOREIGN KEY(rc) REFERENCES RC(id),
FOREIGN KEY(of) REFERENCES O(id),
FOREIGN KEY(ot) REFERENCES O(id),
UNIQUE(rc, of, ot)
)


CREATE TABLE IF NOT EXISTS A
(
id INTEGER PRIMARY KEY,
v INTEGER NOT NULL UNIQUE,
FOREIGN KEY(v) REFERENCES V(id)
)

CREATE TABLE IF NOT EXISTS AC
(
id INTEGER PRIMARY KEY,
c INTEGER NOT NULL,
a INTEGER NOT NULL,
v INTEGER NOT NULL,
FOREIGN KEY(c) REFERENCES C(id),
FOREIGN KEY(a) REFERENCES A(id),
FOREIGN KEY(v) REFERENCES V(id),
UNIQUE(c, a)
)

CREATE TABLE IF NOT EXISTS AR
(
id INTEGER PRIMARY KEY,
r INTEGER NOT NULL,
a INTEGER NOT NULL,
v INTEGER NOT NULL,
FOREIGN KEY(r) REFERENCES R(id),
```

```
FOREIGN KEY(a) REFERENCES A(id),
FOREIGN KEY(v) REFERENCES V(id),
UNIQUE(r, a)
)

CREATE TABLE IF NOT EXISTS ARC
(
id INTEGER PRIMARY KEY,
rc INTEGER NOT NULL,
a INTEGER NOT NULL,
v INTEGER NOT NULL,
FOREIGN KEY(rc) REFERENCES RC(id),
FOREIGN KEY(a) REFERENCES A(id),
FOREIGN KEY(v) REFERENCES V(id),
UNIQUE(rc, a)
)


CREATE TABLE IF NOT EXISTS ACO
(
id INTEGER PRIMARY KEY,
co INTEGER NOT NULL,
a INTEGER NOT NULL,
v INTEGER NOT NULL,
FOREIGN KEY(co) REFERENCES CO(id),
FOREIGN KEY(a) REFERENCES A(id),
FOREIGN KEY(v) REFERENCES V(id),
UNIQUE(co, a)
)

CREATE TABLE IF NOT EXISTS ARCO
(
id INTEGER PRIMARY KEY,
rco INTEGER NOT NULL,
a INTEGER NOT NULL,
v INTEGER NOT NULL,
FOREIGN KEY(rco) REFERENCES RCO(id),
FOREIGN KEY(a) REFERENCES A(id),
FOREIGN KEY(v) REFERENCES V(id),
UNIQUE(rco, a)
)
```

ДОДАТОК Б

Чому Julia [12] ?

Сучасний мовний дизайн і методи компіляції дозволяють усунути компроміс продуктивності та забезпечити єдине середовище, достатньо продуктивне для створення прототипів і достатньо ефективне для розгортання високопродуктивних програм.

Мова програмування Julia виконує цю роль: це гнучка динамічна мова, яка підходить для наукових і чисельних обчислень, з продуктивністю, порівнянною з традиційними мовами зі статичними типами.

Julia має опціональну типізацію, множинну диспетчеризацію та продуктивність, досягнуту за допомогою виведення типу та JIT-компіляції, реалізованої за допомогою LLVM.

Julia мультипарадигмальна, поєднує в собі риси імперативного, функціонального та об'єктно-орієнтованого програмування. Julia забезпечує легкість і виразність для чисельних обчислень високого рівня, як і такі мови, як R, MATLAB і Python, але також підтримує загальне програмування. Щоб досягти цього, Julia спирається на родовід мов математичного програмування, але також запозичує багато з популярних динамічних мов, зокрема Lisp, Perl, Python, Lua та Ruby.

Що отримуємо.

- Безкоштовний і відкритий код (ліцензія MIT)
- Визначені користувачем типи такі ж швидкі та компактні, як і вбудовані
- Немає необхідності векторизувати код для продуктивності; девекторизований код швидкий
- Призначений для паралелізму та розподілених обчислень
- Полегшене «зелене» потокування (coroutines)
- Ненав'язлива, але потужна система типів
- Елегантні та розширювані перетворення та просування для числових та інших типів
- Ефективна підтримка Unicode, включаючи, але не обмежуючись, UTF-8
- Безпосередній виклик функцій C (не потрібні обгортки чи спеціальні API)

- Потужні можливості, подібні до оболонки, для керування іншими процесами
- Lisp-подібні макроси та інші засоби метапрограмування

Pluto notebook.

Pluto — це не просто написання остаточного документа, він дає змогу експериментувати та досліджувати:

- реактивний - при зміні функції або змінної Pluto автоматично оновлює всі залежні комірки;
- легкий - написаний чистою Julia і простий в установці (тільки Julia і browser);
- простий - немає прихованого стану робочої області, дружній інтерфейс користувача.

ДОДАТОК В

Заповнення сховища даних семантичної мережі

Створюємо базу даних

```
tables = [
(drop = true, create = true, file = "sql/create_V.sql", name = "V"),
(drop = true, create = true, file = "sql/create_C.sql", name = "C"),
(drop = true, create = true, file = "sql/create_R.sql", name = "R"),
(drop = true, create = true, file = "sql/create_A.sql", name = "A"),
(drop = true, create = true, file = "sql/create_RC.sql", name = "RC"),
(drop = true, create = true, file = "sql/create_AC.sql", name = "AC"),
(drop = true, create = true, file = "sql/create_AR.sql", name = "AR"),
(drop = true, create = true, file = "sql/create_ARC.sql", name = "ARC"),
(drop = true, create = true, file = "sql/create_O.sql", name = "O"),
(drop = true, create = true, file = "sql/create_CO.sql", name = "CO"),
(drop = true, create = true, file = "sql/create_ACO.sql", name = "ACO"),
(drop = true, create = true, file = "sql/create_RCO.sql", name = "RCO"),
(drop = true, create = true, file = "sql/create_ARCO.sql", name = "ARCO"),
]

file = "semantic.duckdb"

function create_table(db, t)
        o = ""
        if t.drop
                sql = "DROP TABLE IF EXISTS $(t.name);"
                r = DuckDB.execute(db, sql)
                o = o * "executed:\n$sql\nresult: $r\n"
        end
        if t.create
                sql = read(t.file, String)
                r = DuckDB.execute(db, sql)
                o = o * "executed:\n$sql\nresult: $r\n"
        end
        o == "" ? "table: $t.name - nothing executed\n" : o
end

function drop_kb(db)
        ts = ["ARCO","RCO","ACO","CO","O","ARC","AR","AC","RC","V","A","R","C"]
        for t in ts
                DuckDB.execute(db, "DROP TABLE IF EXISTS $t;")
        end
        for t in ts
                DuckDB.execute(db, "DROP SEQUENCE IF EXISTS SEQ_$t;")
```

```
        end
end

function create_seq(db)
        ts = ["ARCO","RCO","ACO","CO","O","ARC","AR","AC","RC","V","A","R","C"]
        for t in ts
                DuckDB.execute(db, "CREATE SEQUENCE IF NOT EXISTS SEQ_$t;")
        end
end

function create_kb(ts, file)
        db = DuckDB.open(file)
        out = ""
        drop_kb(db)
        create_seq(db)
        for t in ts
                out = out * create_table(db, t)
        end
        (db, Text(out))
end
```

Завантажуємо документ та робимо попередню обробку тексту

```
function proc_document(file)

        l = StringDocument(text(FileDocument(file)))
        TextAnalysis.remove_whitespace!(l)
        sents = TextAnalysis.sentence_tokenize(Languages.Ukrainian(),
        TextAnalysis.text(l))
        crps = Corpus([StringDocument(String(s)) for s in sents])
        languages!(crps, Languages.Ukrainian())
        remove_case!(crps)
        sent_lcase = [TextAnalysis.text(d) for d in crps]
        prepare!(crps, strip_punctuation | strip_numbers)
        #remove_words!(crps, ["на","і","що","в","до","не","для"])
        update_lexicon!(crps)
        #lex = lexicon(crps)
        update_inverse_index!(crps)
        inverse_index(crps)
        (sent_lcase, crps)
end


l1_sent_lcase, l1_crps = proc_document("Лекція 1.txt")
```

Словники запитів до бази даних

```
sql_inserts = Dict(
```

```
                :A => "INSERT INTO A (id, v) VALUES(nextval('SEQ_A'), ?) RETURNING id",
                :C => "INSERT INTO C (id, v) VALUES(nextval('SEQ_C'), ?) RETURNING id",
                :V => "INSERT INTO V (id, value) VALUES(nextval('SEQ_V'), ?) RETURNING id",
                :AC => "INSERT INTO AC (id, c, a, v) VALUES(nextval('SEQ_AC'), ?, ?, ?)
RETURNING id",
                :R => "INSERT INTO R (id, v) VALUES(nextval('SEQ_R'), ?) RETURNING id",
                :AR => "INSERT INTO AR (id, r, a, v) VALUES(nextval('SEQ_AR'), ?, ?, ?)
RETURNING id",
                :RC => "INSERT INTO RC (id, cf, r, ct) VALUES(nextval('SEQ_RC'), ?, ?, ?)
RETURNING id",
                :ARC => "INSERT INTO ARC (id, rc, a, v) VALUES(nextval('SEQ_ARC'), ?, ?, ?)
RETURNING id",
                :O => "INSERT INTO O (id, v) VALUES(nextval('SEQ_O'), ?) RETURNING id",
                :CO => "INSERT INTO CO (id, c, o) VALUES(nextval('SEQ_CO'), ?, ?)
RETURNING id",
                :ACO => "INSERT INTO ACO (id, co, a, v) VALUES(nextval('SEQ_ACO'), ?, ?, ?)
RETURNING id",
                :RCO => "INSERT INTO RCO (id, rc, of, ot) VALUES(nextval('SEQ_RCO'), ?, ?, ?)
RETURNING id",
                :ARCO => "INSERT INTO ARCO (id, rco, a, v)
VALUES(nextval('SEQ_ARCO'), ?, ?, ?) RETURNING id",
)


        sql_selects = Dict(

                :A => "SELECT id, v, (SELECT V.value FROM V WHERE V.id = A.v) as name
FROM A",
                :C => "SELECT id, v, (SELECT V.value FROM V WHERE V.id = C.v) as name
FROM C",
                :V => "SELECT * FROM V",
                :AC => """SELECT AC.id, AC.c, AC.a, AC.v,
(SELECT V.value FROM C, V WHERE C.id = AC.c AND V.id = C.v) as c_name,
(SELECT V.value FROM A, V WHERE A.id = AC.a AND V.id = A.v) as a_name,
(SELECT value FROM V WHERE V.id = AC.v) as value
FROM AC
""",
                :R => "SELECT id, v, (SELECT V.value FROM V WHERE V.id = R.v) as name
FROM R",
                :AR => """SELECT AR.id, AR.r, AR.a, AR.v,
(SELECT V.value FROM R, V WHERE R.id = AR.r AND V.id = R.v) as r_name,
(SELECT V.value FROM A, V WHERE A.id = AR.a AND V.id = A.v) as a_name,
(SELECT value FROM V WHERE V.id = AR.v) as value
FROM AR
""",
                :RC => """SELECT RC.id, RC.r as r, RC.cf as cf, RC.ct as ct,
(SELECT V.value FROM C, V WHERE C.id = RC.cf AND V.id = C.v) as cf_name,
(SELECT V.value FROM R, V WHERE R.id = RC.r AND V.id = R.v) as r_name,
(SELECT V.value FROM C, V WHERE C.id = RC.ct AND V.id = C.v) as ct_name
FROM RC
```

```
"""",
                :ARC => """SELECT ARC.id, ARC.rc as rc, ARC.a as a, ARC.v as v,
(SELECT V.value FROM R, RC, V WHERE RC.id = ARC.rc AND R.id = RC.r AND V.id = R.v) as
r_name,
(SELECT V.value FROM C, RC, V WHERE RC.id = ARC.rc AND C.id = RC.cf AND V.id = C.v) as
cf_name,
(SELECT V.value FROM C, RC, V WHERE RC.id = ARC.rc AND C.id = RC.ct AND V.id = C.v) as
ct_name,
(SELECT V.value FROM A, V WHERE A.id = ARC.a AND V.id = A.v) as a_name,
(SELECT value FROM V WHERE V.id = ARC.v) as value
FROM ARC
""",
                :O => "SELECT id, flag, v, (SELECT V.value FROM V WHERE V.id = O.v) as
name FROM O",
                :CO => """SELECT CO.id, CO.c as c, CO.o as o,
(SELECT V.value FROM C, V WHERE C.id = CO.c AND V.id = C.v) as c_name,
(SELECT V.value FROM O, V WHERE O.id = CO.o AND V.id = O.v) as o_name
FROM CO
""",
                :ACO => """SELECT ACO.id, ACO.co, ACO.a, ACO.v,
(SELECT V.value FROM C, CO, V WHERE C.id = CO.c AND CO.id = ACO.co AND V.id = C.v)
as c_name,
(SELECT V.value FROM O, CO, V WHERE O.id = CO.o AND CO.id = ACO.co AND V.id = O.v)
as o_name,
(SELECT V.value FROM A, V WHERE A.id = ACO.a AND V.id = A.v) as a_name,
(SELECT value FROM V WHERE V.id = ACO.v) as value
FROM ACO
""",
                :RCO => """SELECT RCO.id, RCO.rc as rc, RCO.of as of, RCO.ot as ot,
(SELECT V.value FROM R, V WHERE R.id = RC.r AND V.id = R.v) as r_name,
(SELECT V.value FROM O, V WHERE O.id = RCO.of AND V.id = O.v) as of_name,
(SELECT V.value FROM O, V WHERE O.id = RCO.ot AND V.id = O.v) as ot_name
FROM RCO, RC
WHERE RC.id = RCO.rc
""",
                :ARCO => """SELECT ARCO.id, ARCO.rco, ARCO.a, ARCO.v,
(SELECT V.value FROM R, V WHERE R.id = RC.r AND V.id = R.v) as r_name,
(SELECT V.value FROM O, V WHERE O.id = RCO.of AND V.id = O.v) as of_name,
(SELECT V.value FROM O, V WHERE O.id = RCO.ot AND V.id = O.v) as ot_name,
(SELECT V.value FROM A, V WHERE A.id = ARCO.a AND V.id = A.v) as a_name,
(SELECT value FROM V WHERE V.id = ARCO.v) as value
FROM ARCO, RCO, RC
WHERE RC.id = RCO.rc AND RCO.id = ARCO.rco
""",
)

        sql_select_ids = Dict(

                :A => "SELECT id FROM A WHERE v = ?",
```

```
        :C => "SELECT id FROM C WHERE v = ?",
        :V => "SELECT id FROM V WHERE value = ?",
        :AC => "SELECT id FROM AC WHERE c = ? AND a = ?",
        :R => "SELECT id FROM R WHERE v = ?",
        :AR => "SELECT id FROM AR WHERE r = ? AND a = ?",
        :RC => "SELECT id FROM RC WHERE cf = ? AND r = ? AND ct = ?",
        :ARC => "SELECT id FROM ARC WHERE rc = ? AND a = ?",
        :O => "SELECT id FROM O WHERE v = ?",
        :CO => "SELECT id FROM CO WHERE c = ? AND o = ?",
        :ACO => "SELECT id FROM ACO WHERE co = ? AND a = ?",
        :RCO => "SELECT id FROM RCO WHERE of = ? AND rc = ? AND ot = ?",
        :ARCO => "SELECT id FROM ARCO WHERE rco = ? AND a = ?",
)
```

Допоміжні функції для роботи з базою даних

```
function select_all(db, table)

    #db = DuckDB.DB(file)
    ret = DataFrame(DuckDB.execute(db, sql_selects[table]))
    #DBInterface.close!(db)
    ret
end

function id(kb, table, params)

    sql_id = sql_select_ids[table]
    df_id = DataFrame(DuckDB.execute(kb, sql_id,
    table in [:AC, :AR, :ARC, :ACO, :ARCO] ? params[1:end-1] : params))
    ret = nrow(df_id) == 0 ? -1 : df_id[1, :id]
end


function value(kb, id)

    sql = "SELECT value FROM V WHERE id = ?"
    df = DataFrame(DuckDB.execute(kb, sql, [id]))
    v = nrow(df) == 0 ? "not found" : df[1, :value]
end


function value_count!(db, param)

    sql_upd = "UPDATE V SET count = count + 1 WHERE id = ?"
    df = DataFrame(DuckDB.execute(db, sql_upd, [param]))
    param
end

function id(df, s)
    df[only(findall(==(s), df.name)), :][:id]
end
```

```
function insert(db, table, params)
        sql_id = sql_select_ids[table]
        ret = id(db, table, params)
        if ret == -1
                sql_ins = sql_inserts[table]
                df = DataFrame(DuckDB.execute(db, sql_ins, params))
                if table in [:A, :C, :R, :O, :AC, :AR, :ARC, :ACO, :ARCO]
                        value_count!(db, params[end])
                end
                ret = nrow(df) == 0 ? -1 : df[1, :id]
        end
        ret
end
```

Завантаження даних в семантичну мережу із встановленням атрибутів та їх значень

```
kb = let

        kb, out = create_kb(tables, file)
        v_nothing = insert(kb, :V, ["nothing"])
        v_author1 = insert(kb, :V, ["Голуб Б.Л."])
        v_doc1 = insert(kb, :V, ["Лекція 1"])

        v_c_doc = insert(kb, :V, ["Документ"])
        v_c_author = insert(kb, :V, ["Автор"])
        v_c_sentence = insert(kb, :V, ["Речення"])
        v_c_word = insert(kb, :V, ["Слово"])

        v_a_title = insert(kb, :V, ["Назва"])
        v_a_name = insert(kb, :V, ["Ім'я"])
        v_a_number = insert(kb, :V, ["Номер"])

        v_r_is_author = insert(kb, :V, ["є автором"])
        v_r_has_parts = insert(kb, :V, ["складається з"])

        # select_all(kb, :V)
        insert(kb, :C, [v_c_doc])
        insert(kb, :C, [v_c_author])
        insert(kb, :C, [v_c_sentence])
        insert(kb, :C, [v_c_word])
        # select_all(kb, :C)

        insert(kb, :A, [v_a_title])
        insert(kb, :A, [v_a_name])
        insert(kb, :A, [v_a_number])
        # select_all(kb, :A)
```

```
		insert(kb, :AC, [id(kb, :C, [v_c_author]), id(kb, :A, [v_a_name]), v_nothing])
		insert(kb, :AC, [id(kb, :C, [v_c_doc]), id(kb, :A, [v_a_title]), v_nothing])

		# select_all(kb, :AC)

		insert(kb, :R, [v_r_is_author])
		insert(kb, :R, [v_r_has_parts])
		# select_all(kb, :R)

		insert(kb, :AR, [id(kb, :R, [v_r_has_parts]), id(kb, :A, [v_a_number]),
v_nothing])
		# select_all(kb, :AR)

		insert(kb, :RC, [id(kb, :C, [v_c_author]), id(kb, :R, [v_r_is_author]), id(kb, :C,
[v_c_doc])])
		rc1 = insert(kb, :RC, [id(kb, :C, [v_c_doc]), id(kb, :R, [v_r_has_parts]),
id(kb, :C, [v_c_sentence])])
		rc2 = insert(kb, :RC, [id(kb, :C, [v_c_sentence]), id(kb, :R, [v_r_has_parts]),
id(kb, :C, [v_c_word])])

		# select_all(kb, :RC)

		insert(kb, :ARC, [rc1, id(kb, :A, [v_a_number]), v_nothing])
		insert(kb, :ARC, [rc2, id(kb, :A, [v_a_number]), v_nothing])
		# select_all(kb, :ARC)

		insert(kb, :O, [v_author1])
		insert(kb, :O, [v_doc1])
		co1 = insert(kb, :CO, [id(kb, :C, [v_c_author]), id(kb, :O, [v_author1])])
		co2 = insert(kb, :CO, [id(kb, :C, [v_c_doc]), id(kb, :O, [v_doc1])])
		# filter(r -> r.c in [id(kb, :C, ["Автор"]), id(kb, :C, ["Документ"])],
		select_all(kb, :CO))

		insert(kb, :ACO, [co1, id(kb, :A, [v_a_name]), v_author1])
		insert(kb, :ACO, [co2, id(kb, :A, [v_a_title]), v_doc1])

		c = id(kb, :C, [v_c_sentence])
		for s in l1_sent_lcase
			o = insert(kb, :O, [insert(kb, :V, [s])])
			if o > 0
				insert(kb, :CO, [c, o])
			end
		end
		# filter(r -> r.c == c, select_all(kb, :CO))

		c = id(kb, :C, [v_c_word])
		for s in keys(lexicon(l1_crps))
			o = insert(kb, :O, [insert(kb, :V, [s])])
```

```
            if o > 0
                insert(kb, :CO, [c, o])
            end
        end
        # filter(r -> r.c == c, select_all(kb, :CO))

        r = id(kb, :R, [v_r_has_parts])

        cf = id(kb, :C, [v_c_doc])
        ct = id(kb, :C, [v_c_sentence])

        rc = id(kb, :RC, [cf, r, ct])
        of = id(kb, :O, [v_doc1])

        a = id(kb, :A, [v_a_number])
        for i in eachindex(l1_sent_lcase)
            ot = id(kb, :O, [insert(kb, :V, [l1_sent_lcase[i]])])
            rco = insert(kb, :RCO, [rc, of, ot])
            insert(kb, :ARCO, [rco, a, insert(kb, :V, ["$i"])])
        end
        # filter(r -> r.rc == rc, select_all(kb, :RCO))
        # select_all(kb, :ARCO)
        # DuckDB.execute(kb, "EXPORT DATABASE 'semantic.duckdb'")
        kb
end
```

Запити до бази знань логічного програмування

```
function select(db, table)

    DataFrame(DuckDB.execute(db, "SELECT * FROM $table"))
end


function table2clauses(db, t)

    df = select(db, t)
    clauses = Vector{Julog.Clause}(undef, nrow(df))
    i = 0
    for r in eachrow(df)
        i+=1
        clauses[i] = Clause(
        Compound(t, [Const(x) for x in r]), [])
    end
    clauses
end
```

Перетворюємо базу даних в базу знань логічного програмування

```
clauses = let

        cl = Clause[]
        for t in tables
                cl = vcat(cl, table2clauses(db, t))
        end
        cl
end
```

Об'єднуємо факти і правила (розділ 5) в єдину базу знань

```
kb = vcat(clauses, rules)
```

Виконуємо тестові запити до бази знань, використовуючи створені правила

```
resolve(@julog([Attr(ID, Name)]), kb)

resolve(@julog([Cat(ID, Name)]), kb)

resolve(@julog([Rel(ID, Name)]), kb)

resolve(@julog([Obj(ID, Name)]), kb)

resolve(@julog([CatObj(ID, CatID, ObjID, CatName, ObjName)]), kb)

resolve(@julog([RelCat(ID, RelID, CatFromID, CatToID, RelName, CatFromName,
CatToName)]), kb)

resolve(@julog([RelCatObj(ID, RelCatID, CatFromID, ObjFromID, CatToID, ObjToID,
RelName, CatFromName, ObjFromName, CatToName, ObjToName)]), kb)

resolve(@julog([AttrCat(ID, CatID, AttrID, ValueID, CatName, AttrName, Value)]), kb)

resolve(@julog([AttrCatObj(ID, CatObjID, CatID, ObjID, AttrID, ValueID, CatName,
ObjName, AttrName, Value)]), kb)

resolve(@julog([AttrRel(ID, RelID, AttrID, ValueID, RelName, AttrName, Value)]), kb)

resolve(@julog([AttrRelCat(ID, RelCatID, RelID, CatFromID, CatToID, AttrID, ValueID,
RelName, CatFromName, CatToName, AttrName, Value)]), kb)

resolve(@julog([AttrRelCatObj(ID, RelCatObjID, RelID, CatFromID, ObjFromID,
CatToID, ObjToID, AttrID, ValueID, RelName, CatFromName, ObjFromName, CatToName,
ObjToName, AttrName, Value)]), kb)
```

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kimball R. The Data Warehouse Toolkit : the complete guide to dimensional modeling (2nd ed) : Wiley Computer Publishing. 2002. 449 c.
2. Inmon W. H. Building the Data Warehouse (4th Edition) : Wiley Publishing. 2005. 576 c.
3. Naeem T. Data Warehouse Concepts: Kimball vs. Inmon Approach. URL: https://www.astera.com/type/blog/data-warehouse-concepts/. 2020.
4. Inmon B., Mariani D., Rapien D., Srivastava R. Text Analytics Simplified : Published by self. 180 c.
5. Inmon B. Textual ETL – Opening Up New Worlds of Opportunity : MIT Information Quality Industry Symposium, July 15-17, 2009
6. Khouri S., Boukhari I., Bellatreche L., Sardet E., Jean S., Baron M. Ontology-based structured web data warehouses for sustainable interoperability: requirement modeling, design methodology and tool : Computers in Industry. Volume 63, Issue 8, October 2012, Pages 799-812. DOI: https://doi.org/10.1016/j.compind.2012.08.001
7. Using Semantic Web Technologies for Exploratory OLAP: A Survey / Abelló A. and co : IEEE Transactions on Knowledge and Data Engineering (Volume: 27, Issue: 2, 01 February 2015). DOI: https://doi.org/10.1109/TKDE.2014.2330822
8. Roy S., Cortesi A., Sen S., Context-aware OLAP for textual data warehouses : International Journal of Information Management Data Insights. Volume 2, Issue 2, November 2022, 100129. DOI: https://doi.org/10.1016/j.jjimei.2022.100129
9. Russell S., Norvig P. Artificial Intelligence: A Modern Approach (3d ed) : Pearson Education. 2010. 1151 c.
10. Helbig H. Knowledge Representation and the Semantics of Natural Language : Springer. 2006. 666 c.
11. A Julia package for Prolog-style logic programming. URL: https://docs.juliahub.com/Julog
12. The Julia Programming Language. URL: https://julialang.org/