НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ННК "Інститут прикладного системного аналізу"
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»

Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)

"___"_____2016 р.

Дипломна робота

першого (бакалаврського) **рівня вищої освіти**
(першого (бакалаврського), другого (магістерського))

зі спеціальності7.05010102, 8.05010102 Інформаційні технології проектування
7.05010103, 8.05010103 Системне проектування
(код та назва спеціальності)

на тему: Архітектури та застосування сучасних графових баз даних ___
_____

Виконав (-ла): студент (-ка) __4__ курсу, групи ___ДА-22___
(шифр групи)

Кошкін Єгор Геннадійович _____ _____
(прізвище, ім'я, по батькові) (підпис)

Керівник асистент Свірін П.В _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант економічний професор, д.е.н. Семенченко Н.В. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент асистент Кухарєв С.О. _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль ст. викладач Бритов О.А. _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент _____
(підпис)

КИЇВ 2016

Національний технічний університет України
«Київський політехнічний інститут»

Факультет (інститут) ННК "Інститут прикладного системного аналізу"
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти Перший(Бакалаврський)
(перший (бакалаврський), другий (магістерський) або спеціаліста)

зі спеціальності7.05010102, 8.05010102 Інформаційні технології проектування
7.05010103, 8.05010103 Системне проектування
(код та назва спеціальності)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)

«___»_____2016 р.

ЗАВДАННЯ
на дипломний проект (роботу) студенту
Кошкіну Єгору Геннадійовичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи)) Архітектури та застосування сучасних графових баз даних

керівник проекту (роботи)) Свірін Павло Володимирович, асистент ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від 12 травня 2016 р. № 50-ст

2. Строк подання студентом проекту (роботи) 08.06.2016

3. Вихідні дані до проекту (роботи)
Розгорнути графову базу даних Neo4j.
Розгорнути реляційну базу даних MySQL.
Привести 2 тестових приклади для отримання часу обробки запитів. Проаналізувати результати, та вказати на сильні та слабкі сторони Neo4j.

4. Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити)

1. Розглянути задачу оперування графовими базами даних.

2. Дослідити та провести аналіз предметної області теорії графів.
3. Проаналізувати існуючі графові бази даних.
4. Розглянути архітектуру графових баз даних.
5. Провести аналіз переваг на розробленому прикладі застосування графої бази даних.
6. Привести приклади сучасного застосування графових баз даних.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів тощо)

1. Блок-схеми архітектури графової БД – плакат.
2. Порівняльна характеристика – плакат.
3. Соціальні графи – плакат.

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічно-організаційна частина праці	професор, д.е.н. Семенченко Н.В.		

7. Дата видачі завдання 01.02.2016

Календарний план

№ з/п	Назва етапів виконання дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Отримання завдання	01.02.2016	
2	Збір інформації	15.02.2016	
3	Дослідження предметної області - теорії графів	28.02.2016	
4	Дослідження існуючих графових баз даних	10.03.2016	
5	Дослідження Neo4j як графову БД	15.03.2016	
6	Розробка прикладу застосування графової БД	25.03.2016	
7	Тестування прикладу та аналіз переваг порівняно з іншими підходами	25.04.2016	
8	Аналіз сучасного застосування графових баз даних	30.04.2016	
9	Оформлення дипломної роботи	31.05.2016	
10	Отримання допуску до захисту та подача роботи в ДЕК	08.06.2016	

Студент _____ Є.Г.Кошкін
(підпис) (ініціали, прізвище)

Керівник проекту (роботи) _____ П.В.Свірін
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

бакалаврської дипломної роботи Кошкіна Єгора Геннадійовича

на тему «Архітектури та сучасне використання графових баз даних»

Дана дипломна робота присвячена аналізу архітектури графових баз даних та приведення прикладної реалізації їх використання у світі сучасних технологій.

В роботі розглянуто загальну архітектуру графових баз даних на конкретному прикладі, були показані основні алгоритми для обробки даних представлені у вигляді графу. Були наведені переваги та недоліки використання графових баз даних в порівнянні з іншими класичними підходами. Також були проілюстровані приклади сучасного застосування конкретної графої бази даних Neo4j в різних наукових сферах.

Загальний обсяг роботи: 92 сторінок, 40 рисунків, 13 таблиць, 13 посилань.

Ключові слова: граф, теорія графів, обробка графів, графова база даних, Neo4j, порівняльна характеристика.

АННОТАЦИЯ

бакалаврской дипломной работы Кошкина Егора Геннадиевича

на тему «Архитектуры и современное использование графовых баз данных»

Данная дипломная работа посвящена анализу архитектуры графовых баз данных и приведения примеров их использования в мире современных технологий.

В работе рассмотрена общая архитектура графовых баз данных на конкретном примере, были показаны основные алгоритмы для обработки данных представленных в виде графа. Были показаны преимущества и недостатки использования графовых баз данных по сравнению з другими классическими подходами. Также были проиллюстрированы примеры современного использования конкретной графой базы данных Neo4j в разных научных сферах.

Общий объем работы: 92 страниц, 40 рисунков, 13 таблиц, 13 ссылок.

Ключевые слова: граф, теория графов, обработка графов, графовая база данных, Neo4j, сравнительная характеристика.

ANNOTATION

a bachelor’s degree work of Koshkin Yehor

entitled "Architecture and application of the modern graph database"

This project is devoted to the analysis of the architecture graph databases and shown examples of it`s usage in the modern technologies.

The project considers to overall architecture of the database graph in the certain example, the basic algorithms have been shown to process the data presented in a graph. The advantages and disadvantages of graph usage were compared with the other classic approaches. Also, the examples of the modern usage of the specific graphs Neo4j database in different scientific fields were illustrated.

Total volume of work: 92 pages, 40 figures, 13 tables, 13 references.

Keywords: graph, graph theory, processing of graphs, graph database, Neo4j, comparative characteristics.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ І ПОЗНАЧЕНЬ

БД – база даних

СУБД – система управління базами даних

ОС – операційна система

ПП – програмний продукт

ФВА – функціонально-вартісний аналіз

ВСТУП

Зародившись при розв'язуванні головоломок і цікавих задач, теорія графів нині стала потужним засобом розв'язування задач широкого спектру проблем. Слово «граф» з'явилося в математичній літературі порівняно недавно. Але поняття графа використовується дуже часто не тільки в математиці, але і повсякденному житті під різними назвами: схема, діаграма, карта, лабіринт тощо. Теорія графів на даний час бурхливо розвивається, її результати застосовують, проектуючи різноманітні електронні пристрої, вивчаючи автомати, у програмуванні, фізиці, хімії, біології, економіці, статистиці та багатьох інших галузях діяльності людини. Також поняття граф проникло навіть в спосіб збереження інформаційних ресурсів

Ще з 80 років XX сторіччя традиційною моделлю збереження даних є реляційна - така модель, яка сприймається користувачем як набір нормалізованих відношень різного ступеню. Інформація у цих базах структуровано зберігається у таблицях з наперед визначеними колонками конкретних типів.

Це змушує розробників створювати програми, строго структуруючи дані, які використовуються у їхньому додатку. Швидкі темпи збільшення обсягу інформації вимагають нових підходів до вирішення проблеми її збереження. Часом вдається модернізувати вже наявні рішення, але інколи потрібні нові засоби, які принципово відмінні від попередників. Більшою мірою мають попит продукти, які використовують реляційну базу даних (РБД), але як альтернатива баз даних SQL десь з початку 2000-х розвивається напрямок NoSQL.

Останні розділяють на декілька типів, залежно від їх масштабування, систем збереження даних, моделей даних та запитів. До єдиної класифікації не дійшли, тому виокремимо основні: ключ/значення, документоорієнтовані, стовпчиково-орієнтовані та графові. База даних типу ключ/значення зручна для організації даних. Вона дає змогу зберігати за деяким ключем будь-які дані. У

документоорієнтованих базах кожен запис зберігається як окремий документ, що має власний набір полів. Стовпчиково-орієнтовані бази зберігають дані не у вигляді кортежів, а стовпчиками. Одна з інших гілок таких баз даних – графові бази даних [1].

Зрештою, для представлення даних у графовій базі даних використовують вершини та ребра, які їх з'єднують [2]. А зважаючи на те, що останнім часом класичні моделі збереження інколи не задовольняють потреб розробників, графові бази даних проявлять величезну перспективу більш зручного та простішого шляху обробки даних.

Таким чином при розробці структури бази даних необхідно допускати різні варіанти проектування, можливо саме ця база даних підтримує використання графових схем і це може істотно скоротити доступ до даних, а на великих обсягах значно зменшити час обробки інформації. Особливо великий інтерес до використання графових баз даних виникає при розробці соціальних мереж. Надалі буде проведено розгляд загальної архітектури графових баз даних на конкретному прикладі, будуть показані основні алгоритми для обробки даних представлені у вигляді графу. Також будуть наведені переваги та недоліки використання графових баз даних в порівнянні з іншими класичними підходами.

1 ДОСЛІДЖЕННЯ ЗАДАЧІ

1.1 Задачі дипломної роботи

Задачами дипломної роботи є:

1. Розглянути задачу оперування графовими базами даних.
2. Дослідити та провести аналіз предметної області теорії графів.
3. Проаналізувати існуючі графові бази даних.
4. Розглянути архітектуру графових баз даних.
5. Розробити приклад використання графої бази та привести переваги
6. Привести приклади сучасного використання графових баз даних
7. Зробити висновки щодо отриманих результатів.

1.2 Ціль дипломної роботи

Основною ціллю дипломної роботи є аналіз архітектури та використанням графових баз даних для оцінки всіх функціональних переваг.

База даних (англ. database) – це сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами; ця сукупність підтримує щонайменше одну з областей застосування. В загальному випадку база даних містить схеми, таблиці, подання, збережені процедури та інші об'єкти. Дані у базі організовують відповідно до моделі організації даних. Таким чином, сучасна база даних, крім саме даних, містить їх опис та може містити засоби для їх обробки.

У сучасних інформаційних системах для забезпечення роботи з базами даних використовують **системи керування базами даних (СКБД).** Система керування базами даних — це система, основана на програмних та технічних засобах, яка забезпечує визначення, створення, маніпулювання, контроль,

керування та використання баз даних. Застосунки для роботи з базою даних можуть бути частиною СКБД або автономними.

Графова база даних — різновид баз даних з реалізацією мережевої мопеді у вигляді графу та його узагальнень.

1.3 Актуальність задачі

Нереляційні бази даних використовуються ще з кінця 90-х років, а їх широке використання почалося з 2009 р. [2]. Останні розділяють на декілька типів, залежно від їх масштабування, систем збереження даних, моделей даних та запитів. До єдиної класифікації не дійшли, тому виокремимо основні: ключ/значення, документоорієнтовані, стовпчиково-орієнтовані та графові. База даних типу ключ/значення зручна для організації даних. Вона дає змогу зберігати за деяким ключем будь-які дані. У документоорієнтованих базах кожен запис зберігається як окремий документ, що має власний набір полів. Стовпчиково-орієнтовані бази зберігають дані не у вигляді кортежів, а стовпчиками. Зрештою, для представлення даних у графовій базі даних використовують вершини та ребра, які їх з'єднують [2]. А зважаючи на те, що останнім часом класичні моделі збереження інколи не задовольняють потреб розробників, графові бази даних проявлять величезну перспективу більш зручного та простішого шляху обробки даних.

1.4 Особливості і проблематика

В реляційних базах даних зв'язок одного запису на інший запис таблиці ідентифікується за допомогою посилання на первинний ключ першого запису через зовнішній ключ другого. Для того, щоб вибрати певні дані спільно з двох таблиць, потрібно застосувати механізм JOIN. JOIN виконуються під час вибірки пошуком співпадінь між первинним і зовнішнім ключами багатьох

записів таблиць, які ми хочемо об’єднати. Ця операція дуже затратна і в обчисленнях, і в пам’яті і її складність зростає експотенціально.

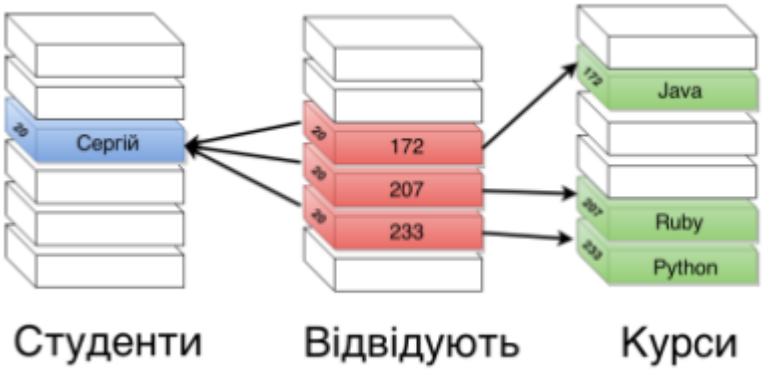

Рисунок 1.1 – Зв’язок “багато-до-багатьох” у реляційній моделі [3]

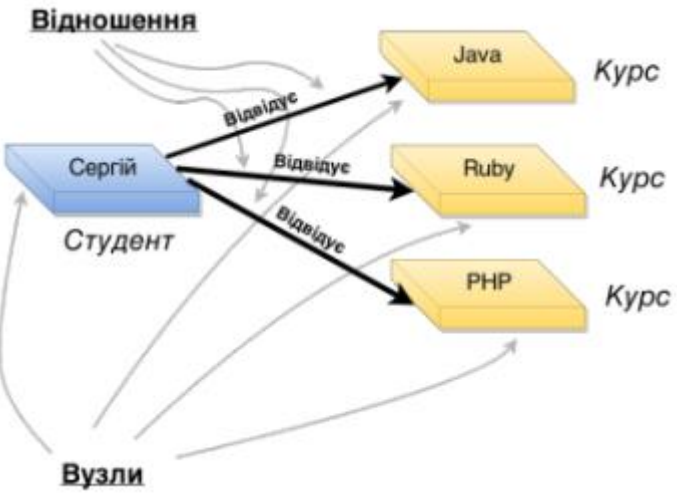

Рисунок 1.2 – Зв’язок “багато-до-багатьох” у графовій моделі [3]

Якщо у базі є відношення багато до багатьох, тоді для ідентифікації зв’язку між двома таблицями створюється спеціальна JOIN-таблиця, що містить у собі зовнішні ключі таблиць, які потрібно об’єднати. Це ускладнює виконання нашого запиту і відповідно збільшує час очікування відповіді.

Через складніть операції JOIN розробники часто змушені денормалізовувати таблиці з метою зменшення кількості об’єднань, що руйнує чітку і логічну структуру сутностей. Для побудови систем з великою кількістю

зв’язків, запити до яких вимагатимуть обєднання більше ніж двох таблиць, використовувати реляційну базу даних нераціонально. В минулому відсутність життєздатних альтернатив і величезне ком’юніті підтримки цього виду баз майже унеможливило широке росповсюдження інших реалізацій БД.

2 ТЕОРІЯ ГРАФІВ

2.1 Введення

У теоретико-графових термінах формулюється значна кількість задач, пов'язаних з дискретними об'єктами [4]. В деякій мірі через теорію графів відбувається проникнення математичних методів в науку і техніку. Слово «граф» з'явилося в математичній літературі порівняно недавно. Тим часом поняття графа використовується дуже часто не тільки в математиці, але і повсякденному житті під різними назвами: схема, діаграма, карта, лабіринт тощо. Теорія графів на даний час бурхливо розвивається, її результати застосовують, проектуючи різноманітні електронні пристрої, вивчаючи автомати, у програмуванні, фізиці, хімії, біології, економіці, статистиці та багатьох інших галузях діяльності людини.

2.2 Історія виникнення теорії

Початок теорії графів як математичної дисципліни прийнято повязувати з Ейлером, який знайшов умову існування циклу у зв’язному графі в його знаменитому міркуванні про Кенігсбергські мости. Проте стаття Ейлера 1736 року була єдиною протягом майже ста років. Інтерес до проблем теорії графів відродився в середині 19-го сторіччя і був зосереджений головним чином в Англії. Було багато причин для такого пожвавлення вивчення графів: природничі науки зробили свій вплив завдяки дослідженням електричних ланцюгів, моделей кристалів і структур молекул. Розвиток формальної логіки привів до вивчення бінарних відношень у формі графів. У 1847 році інженер-електрик Г. Кіргоф розробив теорію дерев для дослідження електричних ланцюгів, а математик А. Келі у 1857 році описав будову вуглеводів за допомогою трьох типів дерев. Г. Кірхгоф при складанні повної системи рівнянь

для струмів і напруги в електричній схемі запропонував по суті представляти таку схему графом і знаходити в цьому графові кістякові дерева, за допомогою яких виділяються лінійно незалежні системи контурів. Велике число популярних головоломок подавалося формулюванням безпосередньо в термінах графів, і це приводило до розуміння, що багато задач такого роду містять деяке математичне ядро, важливість якого виходить за рамки конкретного питання. 20-те століття було свідком неухильного розвитку теорії графів, яка вступила в новий період інтенсивних розробок. У цьому процесі явно помітний вплив запитів нових областей: теорії ігор і програмування, теорії передачі повідомлень, електричних мереж і контактних ланцюгів, а також проблем психології і біології.

Починаючи з 30-х років XIX століття, популярність графів і кількість праць з чистої теорії графів та її застосувань неухильно зростає. За допомогою графа моделюються будь-які схеми, в яких виділяються більш прості частини (вершини) і зв'язки між ними (ребра). Не дивно, що графи зустрічаються у дослідженнях з соціології, психології, економіки, теорії ігор, логіки, програмування, теорії ймовірностей (ланцюги Маркова), квантової механіки (діаграми Фейнмана), хімії (структура молекул), статистичної механіки, кристалофізики, медицини (нервові, судинні та інші мережі), електро- і радіотехніки, лінгвістики, теорії розкладів, з транспортних мереж і течій, вентиляційних мереж та ін.

У 20 ст. задачі, пов'язані з графами, почали виникати не лише у фізиці, хімії, електротехніці, біології, економіці, соціології тощо, але й усередині математики, у таких розділах, як топологія, алгебра, теорія ймовірності, теорія чисел. У 20-30-х роках 20 ст. з'явилися перші результати, що відносяться до вивчення властивостей зв'язності, планарності, симетрії графів, які привели до формування ряду нових напрямів в теорії графів.

Термін «граф» уперше зявився в книзі угорського математика Д. Кеніга «Theorie der endlichen und unendlichen Graphen», що вийшла у 1936 році.

Значно розширилися дослідження з теорії графів у кінці 40-х - початку 50-х років, перш за все через розвиток кібернетики і обчислювальної техніки. Завдяки розвитку обчислювальної техніки, вивченню складних кібернетичних систем, зріс інтерес до теорії графів, а проблематика теорії графів істотним чином збагатилася. Крім того, використання ЕОМ дозволило вирішувати конкретні завдання, що виникають на практиці, пов'язані з великим об'ємом обчислень. Для ряду екстремальних задач теорії графів були розроблені методи їх розв’язання, наприклад, один з таких методів дозволяє розв’язувати задачі про побудову максимального потоку через мережу

2.3 Задача про кенісбергські мости

Задача виникла у пруському містечку Кенігсберг (зараз Калінінград, Російська Федерація) на річці Прегал [5]. Мешканці Кенігсбергу полюбляли прогулюватися по стежці, яка включала сім мостів через річку Прегал. Людям було цікаво, чи можуть вони, почавши шлях з однієї ділянки суші, обійти всі мости, побувавши в кожному лиш один раз, і повернутися в точку початку шляху (перепливати річку також заборонялось). Сім мостів з’єднували два береги річки та два островки так, як показано на рисунку 2.1.

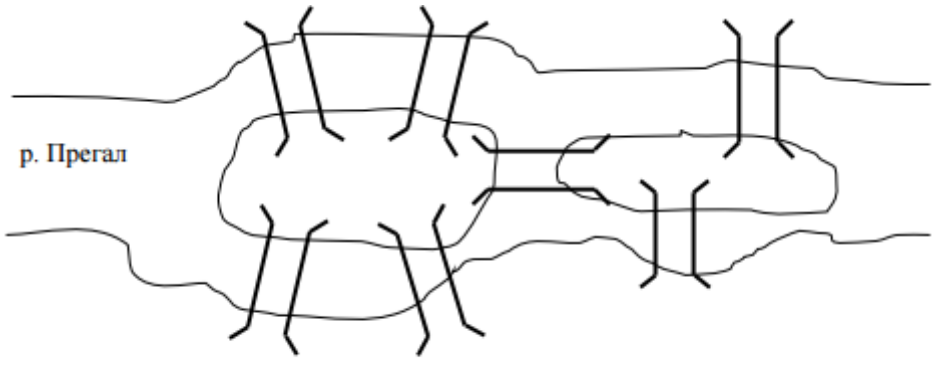

Рисунок 2.1 – Схематичне відображення мостів [5]

Ейлер побудував мультиграф, зображений на рисунку 2.2.

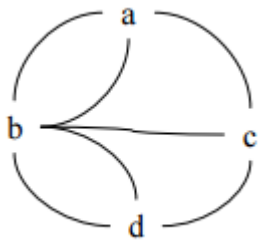

Рисунок 2.2 – Граф

В цьому мультиграфі ділянки суші зображені як вершини, а стежки через мости – як ребра. В такому випадку задача здобуває наступне формулювання: починаючи з довільної вершини, проходячи по кожному ребру тільки один раз, повернутися у вихідну вершину. Виявилося, ця задача не має розв’язків. Виявилося, ця задача не має розв’язків. Ейлер показав, що можливість пройти через граф, пройшовши кожне ребро рівно один раз, залежить від степенів вершин. Степінь вершини це кількість ребер, що торкаються її. Аргументи Ейлера показали, що необхідною умовою прогулянки бажаного виду через граф є зв'язність графа і відсутність або наявність рівно двох вершин непарного степеня. Ця умова виявилась і достатньою, що стверджував Ейлер і пізніше довів Карл Гьєхолзер. Такий шлях називається ейлерів шлях. Далі, якщо присутні дві вершини непарного степеня, тоді Ейлерів шлях почнеться з однієї з них і закінчиться в іншій. Через наявність чотирьох вершин непарного степеня, історична задача не має розв'язку. Іншим формулюванням задачі є запит на шлях, який проходить усіма ребрами і початкова та кінцева точки якого збігаються. Такий шлях називаєть ейлерів цикл. Такий шлях існує тоді і тільки тоді, коли граф зв'язний і не містить вершин непарного степеня. Всі ейлерові цикли є ейлеровими шляхами, але не навпаки.

2.4 Основи побудови графів

Граф — це множина X і множина відношень R, заданих на X. Графічно множина X зображається точками, що називаються вершинами графа, а множина R — лініями, так званими ребрами (дугами) графа. На рис.8 (а) точки 1, 2, 3,... ,7— вершини графа, а лінії 1—2, 2—3, 2—4, 4—5, 2-6 та ін. — його ребра[6].

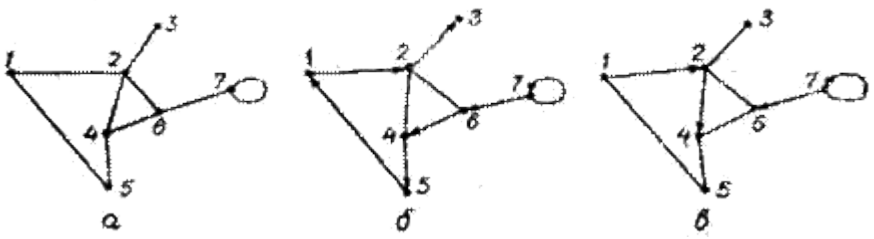

Рисунок 2.3 – Граф неорієнтований (а), орієнтований (б), змішаний (в) [6]

Ребро, яке з'єднує дві вершини графа, є інцидентним цим вершинам, а самі вершини — інцидентні ребру. Дві вершини графа іменуються суміжними; якщо вони визначають ребро графа. Два ребра іменуються суміжними, якщо вони мають спільну вершину. Вершина, інцидентна тільки одному ребру, називається висячою, а не інцидентна жодному ребру — ізольованою. Граф, утворений тільки ізольованими вершинами — це нуль-граф.

Якщо кінці ребра інцидентні одній і тій же вершині, то таке ребро називається петлею. На рисунку 2.3 зображене ребро-петля, яке виходить з вершини 7 і входить у неї.

Інколи пари вершин з'єднуються не одним, а декількома ребрами. Такі графи називаються мультиграфами. Ними моделюються пари населених пунктів, що з'єднані декількома шляхами одного виду транспорту чи лініями декількох видів транспорту.

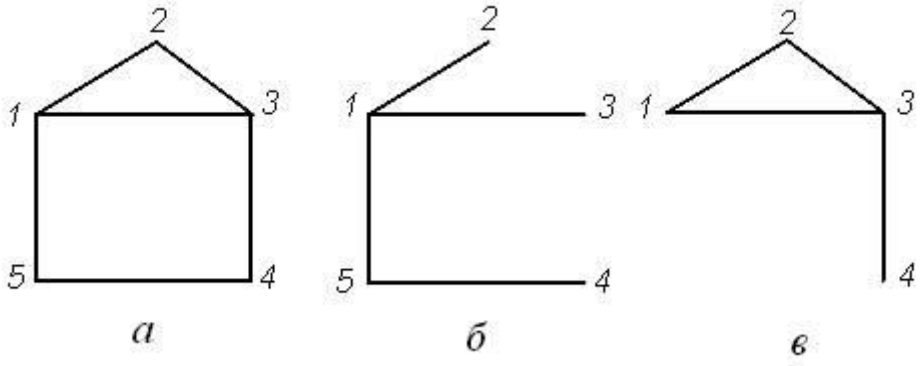

Рисунок 2.4 – Граф (а), його частковий граф (б), підграф (в) [6]

Якщо порядок кінців ребер графа не береться до уваги, то такий граф називається неорієнтованим (рис. 2.3, а). У протилежному випадку, тобто коли пари вершин упорядковані, граф називається орієнтованим (орограф). Орієнтовані ребра — це дуги графа (рис. 2.3, б). Змішаним називається такий граф, в якому частина ребер орієнтована, а частина неорієнтована (рис. 2.3, б). Графи бувають скінченими, якщо множина їхніх вершин $X = \{x1,x2 , ...,xn\}$ є скінченою, і нескінченними, якщо ця множина змінюється від одиниці до нескінченності. В будівництві використовують положення теорії скінчених графів. Якщо граф має п вершин $(n > 1)$ і кожна пара вершин з'єднана ребром, він називається повним. У протилежному випадку маємо неповний граф. Існують поняття доповнення до графа, часткового графа і підграфа. Доповнення до графа — це такий граф, ребра якого разом з цим графом утворюють повний граф (позначається через G). Частковим називається граф, ребра якого є підмножиною цього графа і множина вершин якого збігається з множиною вершин цього графа (рис. 2.4, б). Підграфом називається граф, вершинами якого є підмножина вершин певного графа, а ребрами — усі ребра, обидва кінці яких інцидентні цим вершинам (рис. 2.4, в). Так, граф на рис. 2.4 (б) є підграфом графа на рис. 2.4 (а), а цей останній є його надграфом. Коли

задається або відшукується певна послідовність ребер (дуг), тоді йдеться про маршрути — ланцюги і шляхи відповідно на неорієнтованому і орієнтованому графах, а також про цикли і контури теж відповідно на цих графах. Ланцюг — така послідовність ребер неорієнтованого графа, в якій кінець одного ребра є початком іншого, а початкова і кінцева вершини не збігаються. Аналогічно визначається шлях на орографі. Перша у ланцюгу чи шляху вершина називається початковою, остання — кінцевою, всі інші — проміжними. Ланцюги чи шляхи можуть бути простими і складними, елементарними і неелементарними. Прості вони в тому випадку, коли усі ребра (дуги) різні. В протилежному випадку — це складні ланцюги чи шляхи. Якщо у ланцюгу чи шляху жодна вершина не повторюється двічі, то він називається елементарним, і навпаки, ланцюг, який два або більше разів проходить через певну вершину, називається неелементарним. Цикл — це ланцюг, який починається і закінчується однією і тією ж вершиною неорієнтованого графа. Аналогічно визначається контур орографа. Цикли і контури також бувають прості і складні, елементарні і неелементарні. Це залежить від виду ланцюгів, які їх утворюють.

На рис. 2.3, (а) маршрут 1-2, 2-6, 6-4, 4-5 є ланцюгом, а маршрут 1-2, 2-6, 6-4, 4-5, 5-1 — циклом. На рис. 2.3, (б) маршрут 1-2, 2-6, 6-4, 4-5 є шляхом. Цей же шлях плюс дуга 5-1 утворюють контур.

Виділяються також незалежні і залежні цикли чи контури. На рис. 2.3, а цикли 1-2, 2-4, 4-5, 5-1 та 2-6, 6-4, 4-2 незалежні, а цикл 1-2, 2-6, 6-4, 4-5, 5-1 залежний [6].

Ланцюг (шлях), цикл (контур) мають довжину, яка вимірюється кількістю ребер між початковою і кінцевою вершинами. Довжина ланцюга, що проходить через вершини 1-2-6-4-5 (рис. 2.3, а), дорівнює чотирьом, а довжина циклу 1-2-6-4-5-1 - п'ятьом.

Довжина найкоротшого ланцюга між певними двома вершинами графа називається відстанню між ними. Якщо для кожної вершини підрахувати

відстань до найбільш віддаленої від неї вершини, то найменше з цих чисел називатиметься радіусом, а найбільше — діаметром графа.

Поняття ланцюга чи шляху вихідне для розуміння фундаментального поняття зв'язаності графа.

Дві вершини графа називаються зв'язаними, якщо існує ланцюг чи шлях з кінцями в цих вершинах. Тоді неорієнтований граф вважається зв'язаним, якщо будь-яка пара його вершин зв'язана.

Для орієнтованих графів існує поняття дуже і малозв'язаних графів. Дуже зв'язаний — це орограф, в якому для будь-якої пари вершин існує шлях, що з'єднує їх. У протилежному випадку орограф малозв'язаний.

Графом-деревом називається довільний зв'язаний граф, який не має циклів (див. рис. 2.4, б). Ребра такого графа іменуються вішками, а ребра графа — доповнення до цього графа-дерева — хордами.

2.5 Висновки

Теорія графів — це розділ дискретної математики, особливістю якого є геометричний підхід до вивчення об'єктів. Основне поняття теорії — граф. Поняття графа опирається на основні поняття теорії множин, тому що граф можна розглядати як об'єкт. При цьому зовсім несуттєво, чи з'єднані вершини графа відрізками прямих ліній або криволінійних дуг, яка довжина ліній, як розташовані вершини графа на площини й інші геометричні характеристики графа.

3 РОЗГЛЯД АРХІТЕКТУРИ ГРАФОВИХ БАЗИ ДАНИХ

3.1 Огляд існуючих графових баз даних

Перш за все проведемо короткий огляд трьох графових БД с метою дослідження їх основних можливостей та протестуємо їх для вибору оптимальної графової БД. Дані, які будуть використані для експерименту, мають вигляд білінгової інформації [7]. Вона буде представлена у вигляді графа, де вершина – абонент, а ребро – дзвінок між абонентами. Перша задача буде у виді тестування часу імпорту графу з файлу деякої структури в БД. Дані зберігаються в текстових фалах, де кожний рядок описує ребро графу та має наступний вигляд: *{link_id; source_node; destination_node}.*

Де *source_node* – рядкове значення, яке описує вершину-джерело; *destination_node* - рядкове значення, яке описує значення вершини-досягнення; *link_id* - цілочисельне 64-розрядне число, додатковий атрибут на ребрі.

Можна сказати, що такої моделі достатньо для опису будь-якого графу, так як атрибут може бути посиланням на рядок в таблиці реляційної БД, який має в собі всі параметри ребра.

Друга задача – це пошук сусідніх вершин заданої вершини на певній відстані. Для цієї задачі була реалізована функція, яка передає будь-яке *n*,

а вертає всі сусідні вершини, до яких мінімальна відстань *n*.

Третя задача - це пошук перетину сусідніх вершин. Для цього була реалізована функція, яка шукала сусідні вершини від заданої вершини на певній відстані, а потім для всіх пар вершин проводився перетин множин. Також для інтересу будуть представлені результати тестування реляційної бази даних Microsoft SQL Server 2012 для цих задач.

3.1.1 Sones GraphDB

Ця графова БД була розроблена компанією Sones у 2009 році та підтримувалася до кінця 2011року, після чого компанія збанкрутіла. За час підтримки була випущена версія 2.1, доступна у двох варіантів: версія Community постачається по ліцензії AGPL v3, для комерційних проектів потрібно придбати версію Enterprise. Крім цього, важлива різниця між платною версією та безкоштовною складається в можливості стабільного збереження бази на жорсткому диску. Безкоштовна версія дозволяє зберігати дані тільки в оперативній пам’яті.

Оскільки компанія-розробник припинила своє існування, єдиною версією БД, яку можна протестувати – безкоштовна версія. Тестування буде обмежено невеликим об’ємом даних, які можуть помістится в оперативній памяті.

Sones не має вбудованої підтримки алгоритмів обходу графа, тому для тестування задачі був реалізований пошук в ширину. Крім цього, для тестування пошуку множин сусідніх вершин була реалізована робота з множинами знайдених вершин.

3.1.2 Neo4j GraphDB

Ця графова БД розроблена компанією Neo Technology в 2009 році. Вона доступна у трьох варіантів: Community, Advanced та Enterpise. Версія Community постачається за ліцензією AGPL v3, для комерційних проектів потрібно придбати версію Advanced, яка включає в собі додаткові можливості моніторингу стану бази. Версія Enterprise, крім цього, підтримує резервне копіювання та масштабованість. На сьогоднішній день Neo4j є найбільш популярною графовою БД, тому що безкоштовна версія має в собі всі інструменти для нормальної роботи.

Neo4j, на відміну від Sones, має можливість стабільно зберігати дані на жорсткому дискі. Тому об’єм інформації обмежений тільки в рамках пам’яті жорсткого диску. Для досягнення максимальної продуктивності в Neo4j існує

два види кешування: файловий кеш (file buffer cache) та об’єктний кеш (object cache). Перший кешує дані з жорсткого диску, ціллю цієї дії є збільшення швидкості читання/запису на жорсткий диск даних. Другий кеш зберігає в собі різні об’єкти графу: вершини, ребра та параметри в спеціальному оптимізованому форматі для покращення продуктивності обходу графів.

Neo4j володіє режимом імпорту великого об’єму даних BatchInsert. В цьому режимі проходить відключення транзакцій в БД, тому різко підвищується швидкість імпорту. Крім цього, в Neo4j підтримується алгоритми обходу графів, в тому числі можливий обхід в ширину до заданого рівня, що як раз потрібно для вирішення задачі пошуку сусідніх вершин. Для перетину двох множин, як і в випадку з Sones, була реалізована робота з множинами.

3.1.3 DEX GraphDB

Ця графова БД була розроблена компанією Spersity Technologies в 2008 році. Існує три види ліцензій, за якими надається право використання DEXGraphDB: Community, Commercial, Education. При використанні ліцензії Community у базі сумарно може зберігатися не більше одного міліонну вершин та дозволений доступ до читання тільки одному потоку. За ліцензією Commercial немає ніяких обмежень. Ліцензія Education надається для безкоштовного використання у цілях вивчення та використання у некомерційних проектах.

DEX GraphDB, як і Neo4j, має повноцінну підтримку стабільного зберігання даних. Але на відміну від Neo4j ядро DEX написано на C++. Графова БД DEX має тільки один об’єктний кеш, який зберігає в оперативній пам’яті всі часто використані об’єкти сховища.

БД DEX має широкі можливості по обходу графів. Вона надає можливості вбудованих алгоритмів обходу графів. Також є вбудована підтримка роботи с множинами.

3.1.4 Тестування

Під час тестування операції були реалізовані на мовах програмування C# (Sones, DEX) та Java (Neo4j) за допомогою API, наданому кожною з БД. Для конфігурації надавались стандартні настройки, а також деякі рекомендації з технічної документації: використання BatchInserter при імпорті даних в Neo4j та цілочисельних ідентифікаторів в DEX. В Microsoft SQL Server 2012 створюються дві таблиці (для списку вершин та для списку ребер), а обхід графа реалізований за допомогою збереженої процедури.

Тестові задачі виконувались в порядку, як вони представлені в наступних таблицях: імпорт, пошук сусідів, пошук перетину множин сусідів. У випадку якщо виконання не закінчувалася на протязі дванадцяти годин, операція буде перервана згідно не актуальності роботи з таким об’ємом даних.

Тестування проходило на наступному обладнанні:

Intel Core i5 3.1 GHz, 8 Gb DDR3, 1 Tb 7200 rpm hard drive, Windows Server 2008 x64. В БД DEX та Neo4J значення розміра кешу налаштоване як 4 Gb. При роботі з Neo4J Java Virtual Machine запускалася з наступними параметрами: -d64 -server -Xmx4096m - XX:+UseConcMarkSweepGC.

Для тестування було надано чотири набори даних, які містили в собі різну кількість вершин та ребер.

Талиця 3.1 - 10 000 000 вершин та 90 000 000 ребер

Тестова БД	Час, с			Розмір БД, МБ
	імпорт даних	пошук сусідів до максимального рівні	пошук перетину множин сусідів для двох вершин	
Sones	9	5	10	22 (RAM)
Neo4j	6	3	9	21
DEX	11	3	8	30
SQL Server	9	4	8	45

Талиця 3.2 - 10 000 000 вершин та 90 000 000 ребер

Тестова БД	Час, с			Розмір БД, МБ
	імпорт даних	пошук сусідів до максимального рівні	пошук перетину множин сусідів для двох вершин	
Sones	78	23	150	1510 (RAM)
Neo4j	13	10	31	21
DEX	20	9	32	30
SQL Server	19	16	42	45

Талиця 3.3 - 10 000 000 вершин та 90 000 000 ребер

Тестова БД	Час, с			Розмір БД, МБ
	імпорт даних	пошук сусідів до максимального рівні	пошук перетину множин сусідів для двох вершин	
Sones	4110	301	622	5792 (RAM)
Neo4j	899	24	73	5797
DEX	770	23	72	7238
SQL Server	2115	215	506	10586

Талиця 3.4 - 10 000 000 вершин та 90 000 000 ребер

Тестова БД	Час, с			Розмір БД, МБ
	імпорт даних	пошук сусідів до максимального рівні	пошук перетину множин сусідів для двох вершин	
Sones	-	-	-	-
Neo4j	2589	78	156	13215
DEX	2045	73	143	18103
SQL Server	7392	612	1398	26209

Не дивлячись на те, що дані в Sones повністю зберігаються в оперативній пам’яті та операції читання/запису з жорсткого диску не відбуваються, імпорт зайняв більше всього часу. При операціях імпорту на невеликих об’ємах даних (тест 1 и 2) найкращу продуктивність показав Neo4j. На великих об’ємах даних

(тест 3 и 4) Neo4j та DEX показали майже рівні результати, а в БД Sones взагалі не вдалося завантажити граф через відсутність великого об’єму оперативної пам’яті.

3.1.5 Висновки

Проаналізувавши результати можна зробити висновок, що всі БД життєздатні та показують непогані результати, особливо на маленьких об’ємах даних. Але для Sones буде складно знайти застосування в реальному світі, потрібно дуже багато оперативної пам’яті для роботи з нею. DEX показує трохи кращий результат, майже не відчутний, при порівнянні з Neo4j. Але Neo4j є найбільш популярною графовою БД на даний момент, вона надає більш великий набір різноманітних API для різних мов програмування та різних додаткових функцій, що робить її більш простою в роботі. Тому надалі буде розглянуто більш детально саме ця графову БД.

3.2 Neo4j як графова база даних

Розглянемо реалізацію графових баз даних на основі Neo4j, показуючи, як вони відрізняються від інших засобів зберігання і виконання запитів в напівстрктурованих щільно з'єднаних даних. Хоча не існує ідеальної архітектури графової баз даних, все ж таки розглянемо найбільш використовувані зараз у світі варіанти. Neo4j є графовою базою даних з власними можливостями обробки. Також вона постачається з відкритим вихідним кодом, що робить більш прозорим момент дослідження.

3.2.1 Нативна обробка графів

Хоча сама модель повністю узгоджується з реалізацією графової бази даних, існує безліч способів для кодування і представлення графу в рушії базі даних. З безлічі різних архітектур рушіїв, графова база даних має власні можливості обробки, одна з таких можливостей це використання індексу

вільної суміжності. Коли рушій використовує цей індекс, кожен вузол має посилання на прилеглі до нього вузлів: кожен вузол має мікроіндекс, який вказує на сусідні вузли, тому ця технологія значно «дешевша», ніж використання глобальних індексів [8]. Це означає, що час обробки запитів не залежить від загального розміру графа. Нерідний рушій графової бази даних, на відміну від цього, використовує глобальні індекси, щоб зв'язати вузли разом, як показано на малюнку 3.1. Ці індекси додають додаткові умови пошуку до кожного обходу, тим самим несучи велику обчислювальні витрати. Тому розробники стверджують, що індекс вільної суміжності має вирішальне значення для швидкого та ефективного обходу графа.

Щоб зрозуміти, чому рідні методи обробки графа є більш ефективним, ніж графи засновані на важкій індексації, розглянемо наступне. Залежно від реалізації, пошук за індексом може мати складність *O(log N)* в порівнянні з

O (1) для пошуку безпосередніх відносин. Для того, щоб пройти через граф за m кроків, складність індексованої підходу буде *O (m log n),* в порівнянні з складністю *O (ь)* для реалізації, яка використовує індекс вільних суміжностей.

На рисунку 3.1 показаний нерідний підхід обробки графу. Щоб знайти друзів Аліси, ми повинні спочатку виконати пошук за індексом з скдадністю

O (log N). Це нормально для дрібних пошуків, але це швидко стає «дорогим», коли змінюється напрямок обходу. Якщо, замість того щоб знайти друзів Аліси, потрібно з'ясувати хто дружить з Алісою, доведеться виконати кілька операцій пошуку за індексом, по одному для кожного вузла, який може бути потенційним другом Аліси. Беручи до уваги, що складність є *O(log n)*, щоб з'ясувати хто є друзями Аліси, а складність – *O(m log n)*, щоб з'ясувати хто з людей дружать з Алісою.

Пошук за індексом прекрасно підходять для невеликих мереж, таких як на рисунку 3.1, але занадто незручний за часом виконання складних запитів на великих графів. Замість того щоб використовувати пошук за індексом, Neo4j з вбудованими можливостями обробки графа використовує індекс вільних

суміжності для забезпечення обходів з високою продуктивністю. Рисунок 3.2 показує, як використання відносин усуває необхідність в індексі пошуку.

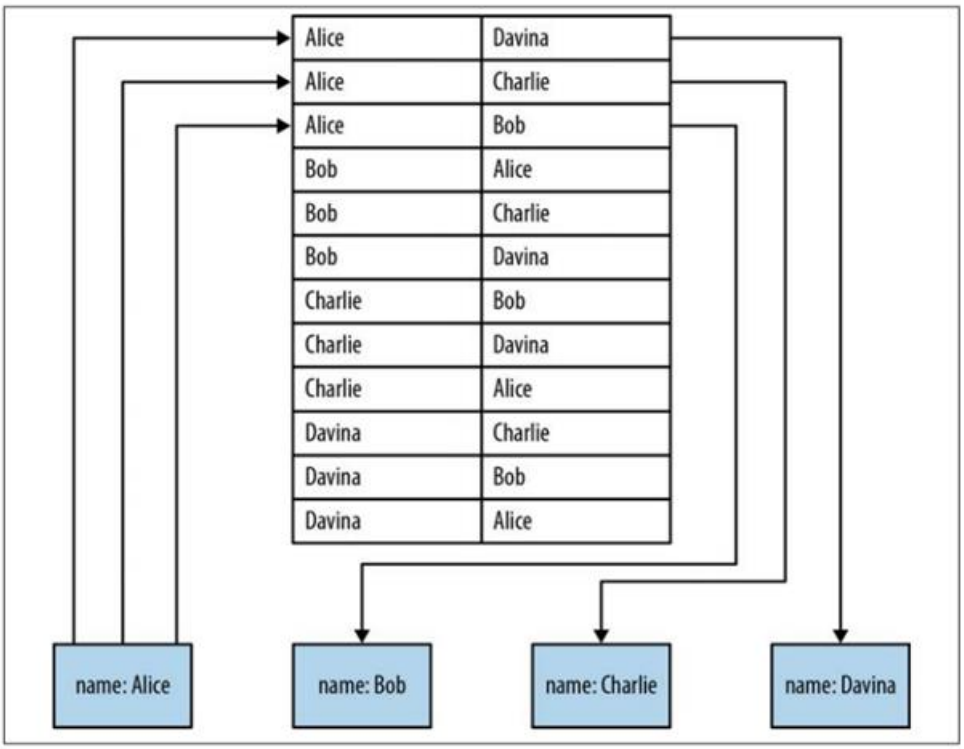

Рисунок 3.1 – Використання глобальних індексів [8]

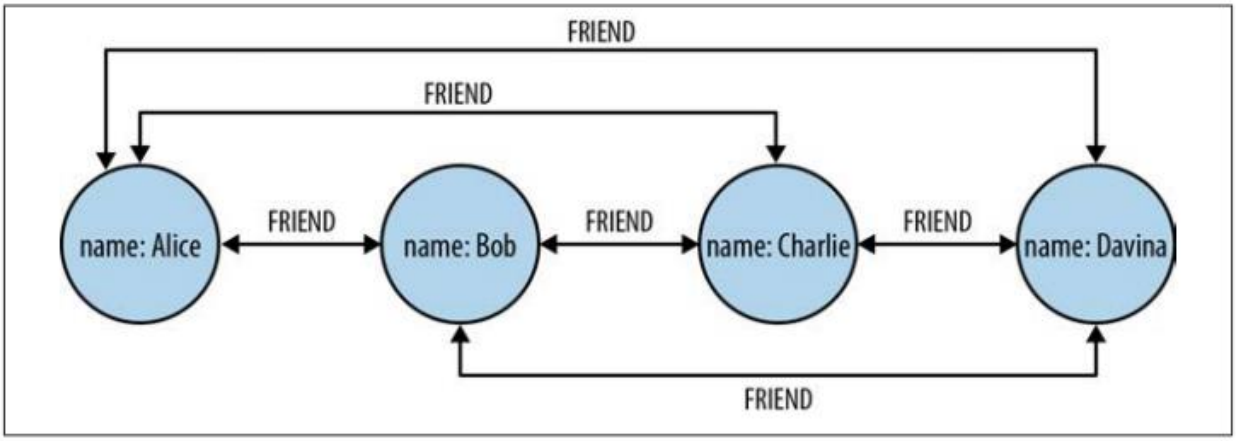

Рисунок 3.2 Використання без глобальних індексів [8]

Нагадаємо, що в універсальних відносинах графової бази даних заданий шлях може бути пройдений в будь-якому напрямку (від хвосту до голови або від голови до хвоста) з мінімальним використанням ресурсів. Як бачимо на рисунку 3.2, щоб знайти друзів Аліси з допомогою графа, ми просто проходимо по вихідним відносинам з типом ДРУГ зі складністю *O(1)*. Щоб дізнатися, хто дружить з Алісою, ми просто стежимо за всіма вхідними відносинами ДРУГ до Аліси, знову складність *O (1)*. З огляду на це, стає ясно, що обхід графу може бути дуже ефективним. Але такі обходи стають високопродуктивними тільки тоді, коли вони підтримуються архітектурою.

3.2.2 Нативне зберігання графів

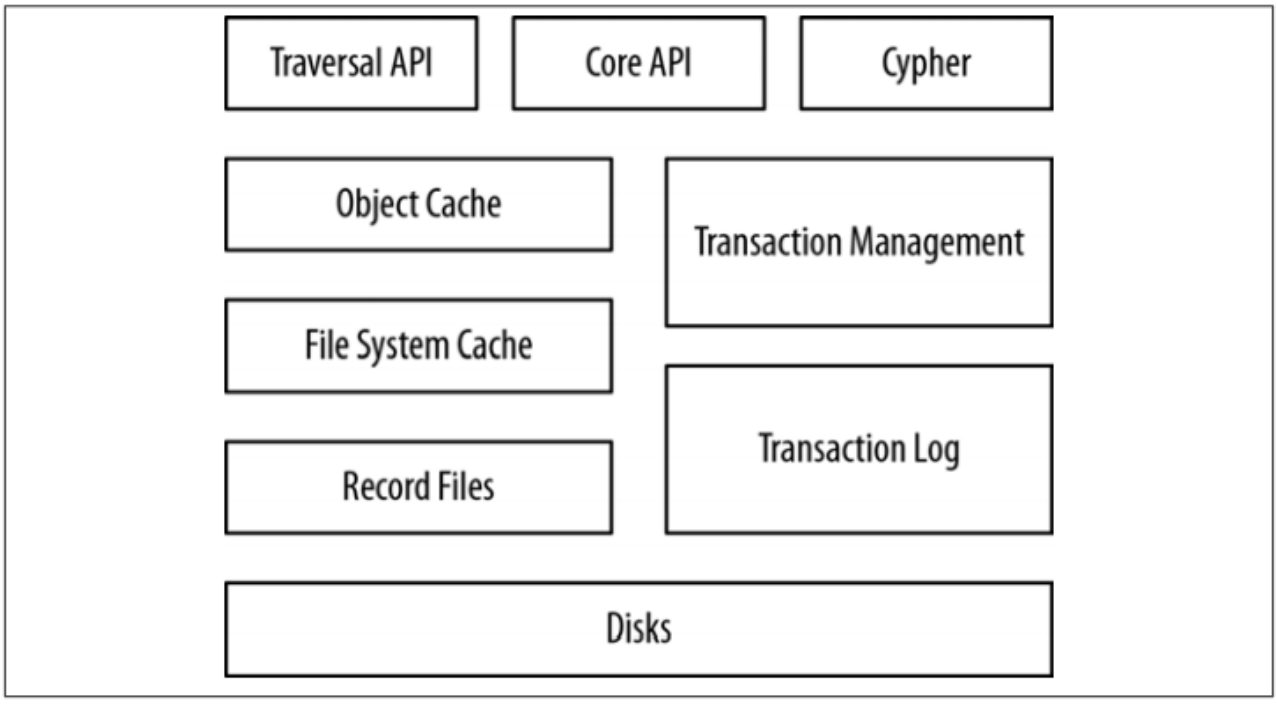

Рисунок 3.3 – Архітектура Neo4j [8]

Якщо індекс вільної суміжності є ключем до високопродуктивних обходів, запитів та запису, то один з ключових аспектів проектування графової бази даних є поняття, в якому варіанті зберігаються графи. Ефективний, нативний формат зберігання графа підтримує надзвичайно швидкі обходи з

використанням будь-яких графових Для наочності, розглянемо базу даних Neo4j як приклад архітектури графової бази даних. Архітектура Neo4j високого рівня представлена на рисунку 3.3. Напрям розгляду буде знизу вгору, з файлів на диску, через програмні інтерфейси і до мови запитів Cypher. Під час огляду розглянемо експлуатаційні характеристики і надійність Neo4j, а також проектні рішення, які роблять Neo4j як одну з продуктивних, надійних графових баз даних. Neo4j зберігає дані графа в різних файлах сховищах.

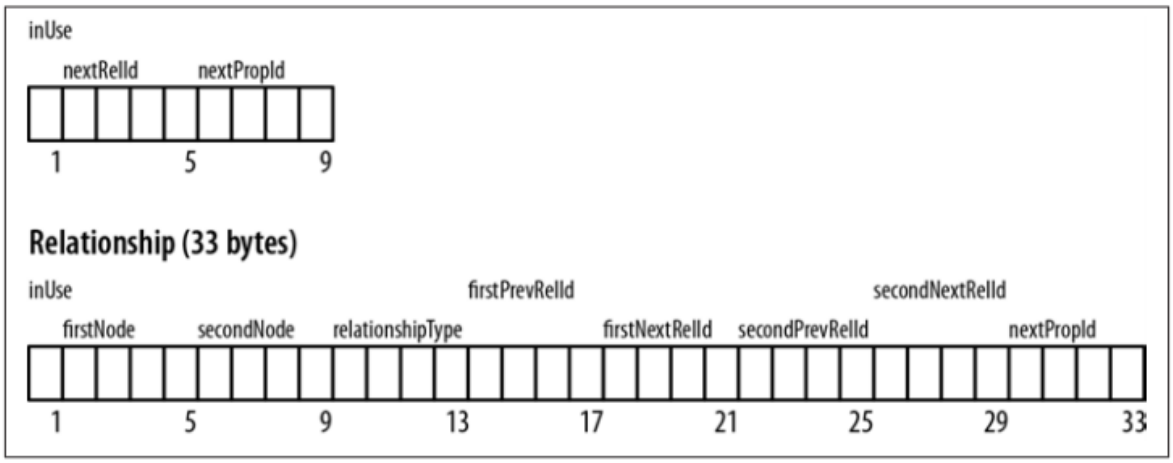

Рисунок 3.4 – Структура файлів сховища вузла [8]

Кожен файл сховища містить дані певної частини графу (наприклад, вузли, відносини, властивості). Розподіл обов'язків, особливо для зберігання поділу структури графа від властивостей даних, полегшує продуктивний обхід.

Почнемо наше дослідження з фізичної пам'яті, дивлячись на структуру вузлів і відносин на диску, як показано на рисунку 3.4. Файл сховища вузлів зберігає записи даних

Кожен вузол створюється в графі на рівні користувача та займає своє місце у фізичному файл (neostore.nodestore.db). Як і більшість файлів сховища Neo4j, файл вузлів являє собою фіксований розмір записів, який займає дев'ять байт в довжину. Записи фіксованого розміру дозволяють проводити швидкий пошук: якщо у нас є вузол з ідентифікатором 100, то ми знаємо, що його запис

починається з 900 байту у файлі. На основі цього формату, база даних може безпосередньо обчислити місцеположення записів за складністю O (1), а не виконувати пошук, який буде коштувати O (log N).

Перший байт запису вузла є прапором. Це означає умову чи використовується запис в даний час зберігання вузла, або чи може він бути утилізований від імені нового вузла (Neo4j .id файлів дозволяє відстежувати невикористовувані записи). Наступні чотири байти представляють ідентифікатор перших відносин, підключених до вузла, а останні чотири байти представляють ідентифікатор першої властивості для вузла. Запис вузла досить легкий: це дійсно тільки покажчики на списки відносин і характеристик. Відносини зберігаються в файлі сховища відносини, neostore.relationshipstore.db.

Як і в сховищі вузлів, сховище відносини складається з фіксованого розміру записів, кожен запис має довжину в 33 байта. Кожен запис відносини містить ідентифікатори вузлів на початку і в кінці відносин, покажчик на тип відносин (який зберігається в файлах сховища типів), а також покажчики на наступні і попередні відносини записів для кожного з початкових і кінцевих вузлів. Ці останні покажчики є частиною того, що часто називають відносинами ланцюга. На рисунку 3.5 видно, як по різному зберігаються файли на диску та їх взаємодія. Кожен з двох записів вузлів містить покажчик на першу властивість цього вузла і на перші відносини в ланцюжку відносин. Для читання властивостей вузла починаємо з покажчика на першу властивість. Для того, щоб знайти зв'язок до вузла, слідуємо за покажчиком відносини цього вузла до його перших відносин (як відносини ПОДОБАЄТЬСЯ в даному прикладі). Потім слідуємо за списком подвійних посилань відносин для даного конкретного вузла, поки не знайдемо конкретні відносини. Виявивши запис для відносин, можемо прочитати їх властивості (якщо такі є), або можемо досліджувати зв’язаний запис вузла, використовуючи ідентифікатори

початкового і кінцевого вузлів. Ці ідентифікатори, помножені на розмір запису вузла, дають негайний перехід на зв’язаний вузол у сховища.

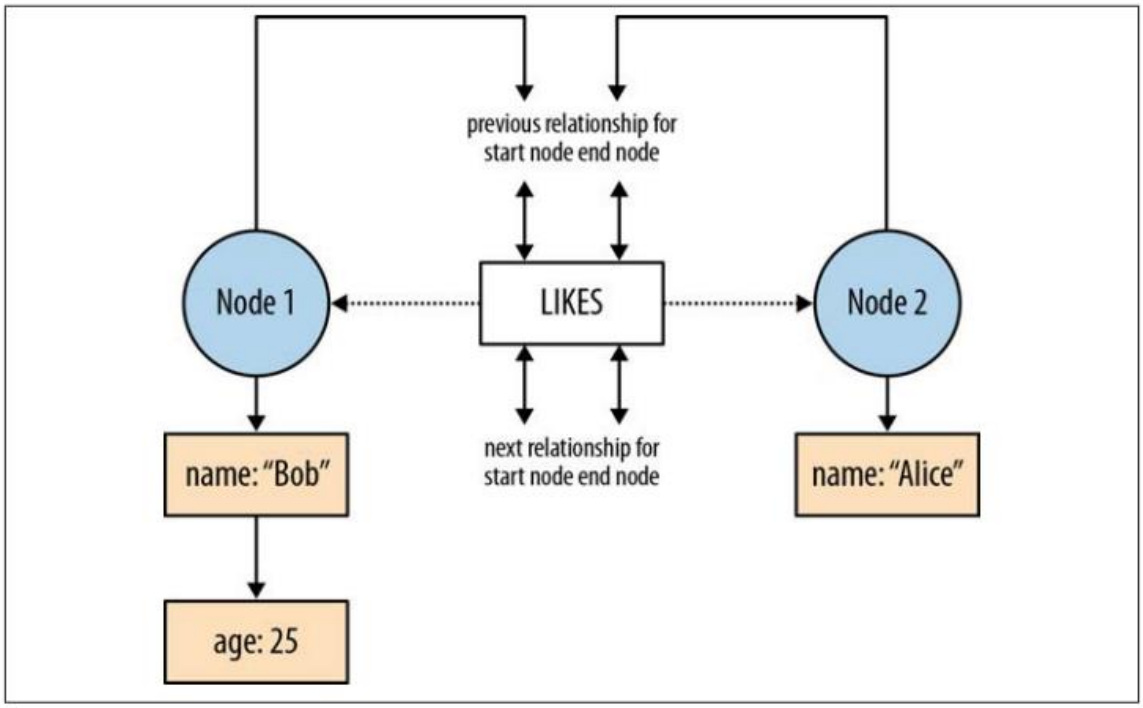

Рисунок 3.5 – Фізична структура графа [8]

За допомогою фіксованого розміру записів, обходи структури даних можуть бути виконані в дуже високих швидкостях. Щоб пройти за шляхом відносин від одного вузла до іншого, база даних виконує кілька нескладних обчислень з ідентифікаторами:

1. З цього запису вузла, знайдіть перший запис в ланцюжку відносин шляхом обчислення його зміщення в сховищі відносин, тобто шляхом множення його ідентифікатору на фіксовану довжину (це 33 байт на момент написання). Це повертає нас безпосередньо до правильного запису в сховищі відносин.
2. Із запису відносин, дізнаємося в полі другого вузла його ідентифікатор. Множимо цей ідентифікатор на розмір запису вузла (дев'ять байт на момент написання), щоб знайти правильний запис вузла в сховище вузлів.

Також крім сховищ вузлів і відносини, є підтримання сховищ властивостей. Ці сховища зберігають пари ключ - тип значення. Нагадаю, що Neo4 дозволяє додавати властивість (пара ключ – тип значення), яка повинна бути приєднана до обох вузлів і відносин між ними. Записи в сховищі властивостей фізично зберігаються в neostore.propertystore.db файлі. Як з вузлами і відносинами, записи властивостей мають фіксований розмір. Для значення кожної властивості запис містить покажчик запису в динамічному сховищі або вбудоване значення. Динамічне сховище дозволяють зберігати великі типи значень. Є два динамічних сховища: динамічне сховище рядків (Neostore.propertystore.db.strings) і динамічне сховище масивів (neostore.prop ertystore.db.arrays). Динамічні записи містять пов'язані списки фіксованих розмірів записів.

Наявність ефективної компоновки зберігання тільки половина картини. Незважаючи на те, що зберігання файлів оптимізоване для швидких проходжень, апаратні міркування все ще можуть зробити істотний вплив на продуктивність.

Обсяг пам'яті значно зросла в останні роки; тим не менш, дуже великі графи буде як і раніше перевищують здатність утримувати їх повністю в основній пам'яті. Обертові диски мають можливість надавати мілісекундний час пошуку в порядку однозначних цифр, які, хоча за людськими мірками легкі, важкуваті при обчисленні строк. Тому час пошуку також залежить від характеристик машин, на яких він проводиться. Для покращення цієї ситуації Neo4j використовує дворівневу архітектуру кешування. Найнижчий рівень в стеці кешування Neo4j є кеш файлової системи. Використовується кешування сторінок, тобто кеш ділить кожне сховище на окремі сегменти, а потім має фіксовану кількість сегментів в файлі сховища. Фактичний обсяг пам'яті, який буде використовуватися для кешування сторінок кожного сховища може бути доопрацьований, хоча при відсутності вхідного сигналу від користувача, Neo4j буде використовувати розумні значення за замовчуванням, грунтуючись на

якості базового обладнання. Сторінки видаляються з кешу за ознакою, що найменш використовувані. Реалізація Neo4j за замовчуванням залишає право роботи базової операційної системи над пам'ятю файлової системи, щоб вирішити, які сторінки видаляти; тобто, операційна система сама визначає, які сегменти віртуальної пам'яті залишити в реальній пам'яті і які для прибирати на диск. Кеш файлової системи особливо корисний, коли відповідні елементи графа змінюються одночасно таким чином, що вони займають ту ж сторінку.

У кеші на високому рівні зберігаються об'єкти уявлення вузлів, відносин і властивостей для швидкого обходу графа. В кеші об'єктів містять властивості вузлів і посилання на їх відносини. Об'єкти відносини містять тільки їх властивості. Відносини вузла будуть згруповані по типу і напрямку відносини; ці угруповання забезпечують швидкий пошук конкретних відносин, заснованих на цих характеристиках, як ми бачимо на рисунку 3.6.

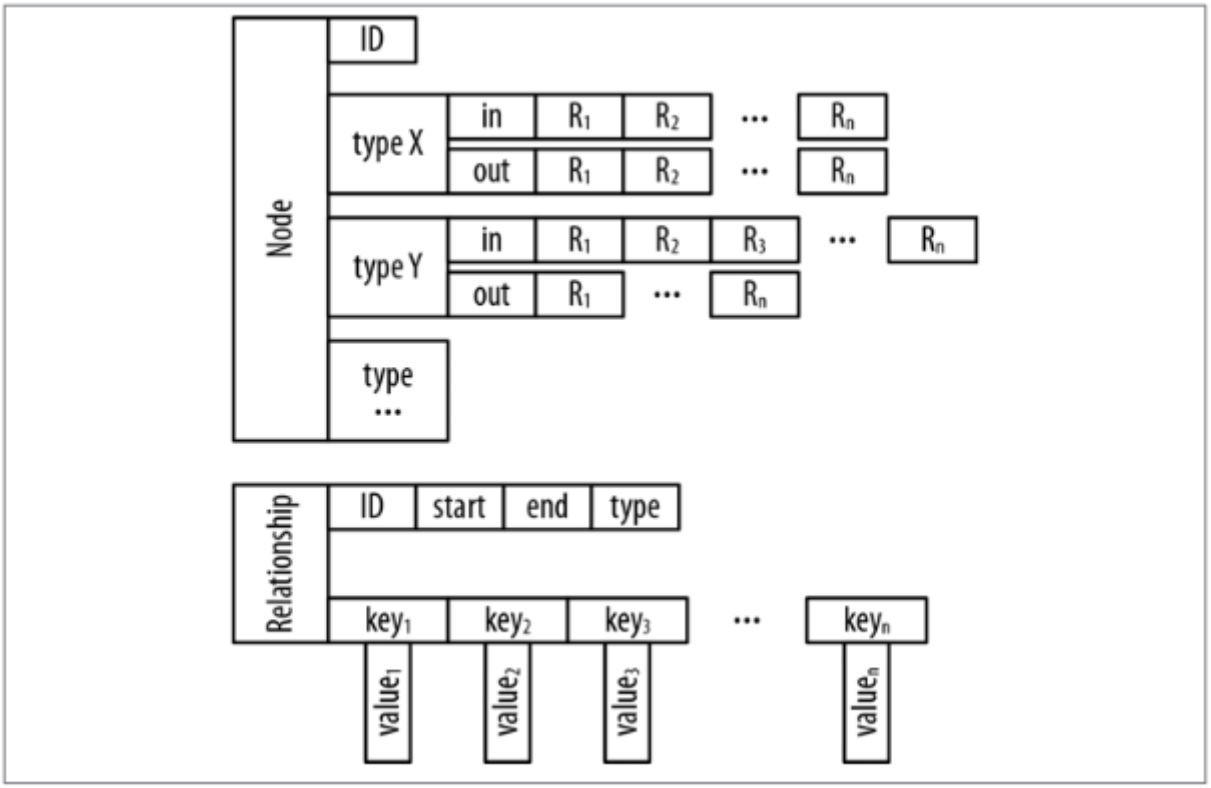

Рисунок 3.6 Схема кешування [8]

3.2.3 API Neo4j

Хоча файлова система і кешування інфраструктури захоплюючі самі по собі, розробники рідко взаємодіють з ними безпосередньо. Замість цього, вони хочуть маніпулювати базою даних графа за допомогою мови запитів. Neo4j використовує мову запитів Cypher, декларативну мову запитів до рідної Neo4j. Існують і інші інтерфейси API. Важливо розуміти вибір інтерфейсів і їх можливостей при старту нового проекту. Ці API-інтерфейси можна розглядати як стек, як показано на рисунку 3.7. Знову ж таки, ці інтерфейси надані ілюстративно. Кожен API має свої переваги і недоліки, зважаючи на це, потрібно робити вибір з них в залежності від проекту.

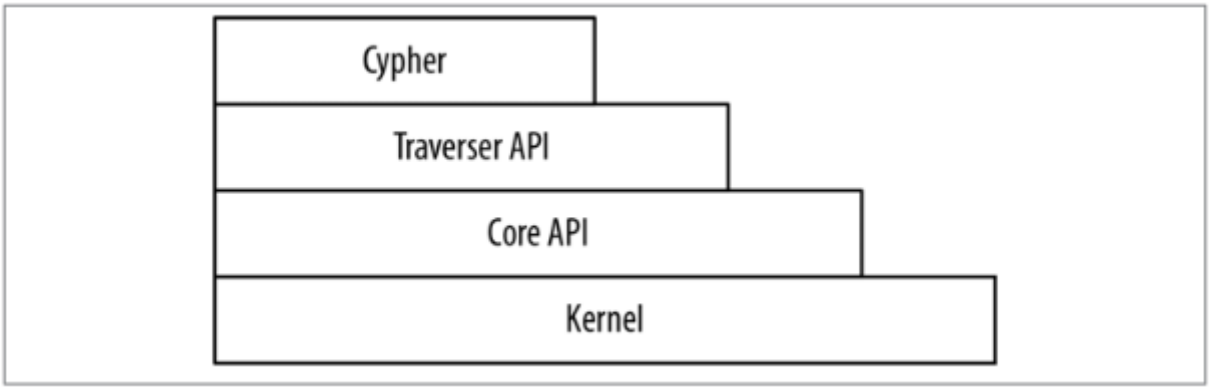

Рисунок 3.7 API [8]

3.2.4 Транзакції

В Neo4j проходить дотримання технології ACID, підтримання розбиття на кластери та поновлення після сбою в системі. ACID (Atomicity, Consistency, Isolation, Durability) — набір характеристик, які гарантують надійну роботу транзакцій. Транзакції були основою надійних обчислень систем на протязі десятиліть. Хоча сховища NOSQL не транзакционной, операції в Neo4j семантично ідентичні традиційному поняттю транзакції бази даних. Запис відбуваються в межах контексту транзакції з блокуванням доступу для інших дій для забезпечення узгодженості на будь-яких вузлах і відносин, що беруть участь в транзакції. У разі, якщо транзакція зазнала невдачу з якоїсь

причини, запис прибирається і права на доступ поновлюється, тим самим зберігаючи граф в колишньому узгодженому стані. Якщо дві або більше транзакції намагаються змінити той же елемент одночасно, Neo4j виявить потенційну тупикову ситуацію і серіалізує транзакції. Запис в межах одного транзакційного контексту не буде видно для інших транзакцій, тим самим зберігаючи ізоляцію.

3.2.5 Висновки

У цьому розділі ми показали, що графи є відмінним вибором для прагматичного моделювання даних. Була показана архітектура графової бази даних, з особливим акцентом на архітектурі Neo4j, і також показані різні характеристики реалізацій графової бази даних.

3.3 Аналіз переваг застосування графових баз даних

3.3.1 Підготовка до порівняння

Для проведення порівняння потрібно обрати предметну область, розробити ER-модель, конвертувати ER-модель у реляційну, підняти відповідні бази даних, створити потрібні таблиці в MySQL, заповнити Neo4j та MySQL однаковими даними, отримати статистичні дані. Предметною областю для дослідження обрано соціальну мережу, яка містить користувачів, їхні повідомлення, групи, аудіо та фото. Розроблену ER-модель зображено на рисунку:

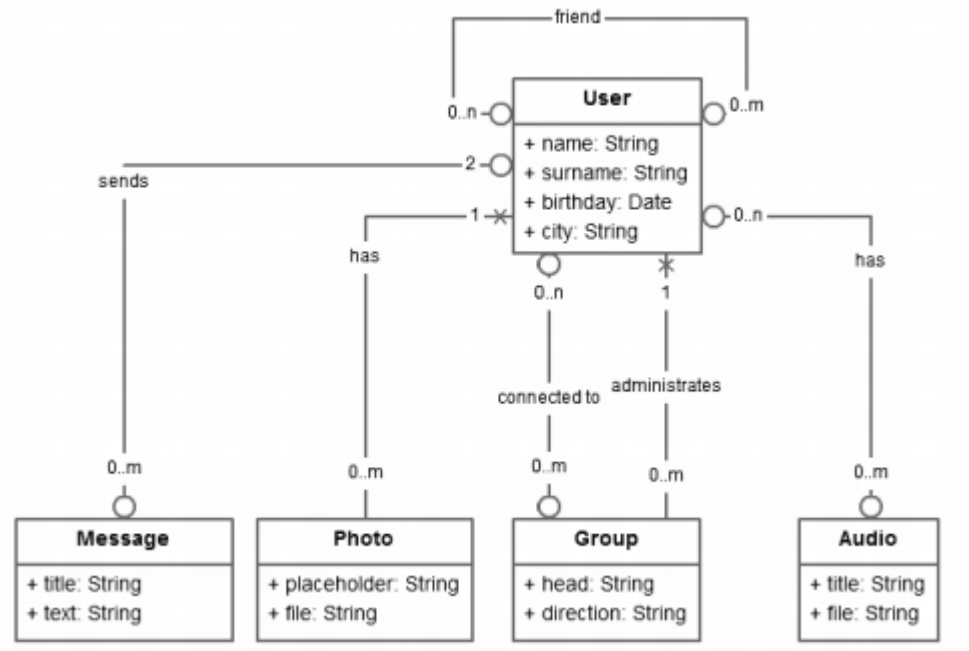

Рисунок 3.8 – ER-модель соціальної графової бази даних

При перетворенні цієї ER діаграми в реляційну додалися нові таблиці: user_audio, user_group, user_photo та friendship. Для того щоб користуватися Neo4j та MySQL, потрібно:

- встановити MySQL Installer для доступу до програмних продуктів, які використовуються для роботи з MySQL;
- завантажити та встановити сервер для роботи.

За замовчуванням MySQL сервер піднімається автоматично. Для редагування та моніторингу стану цієї бази даних було використано програмний продукт MySQL Workbranch. Для роботи з реляційною базою даних ми створили такі сутності: Audio, Friendship, Group, Message, Photo, User, UserAudio, UserGroup, UserPhoto. Заповнення баз даних екземплярами цих сутностей відбувалося у два етапи: генерування даних та вставка в MySQL, копіювання даних з MySQL у Neo4j. Для генерації тестових даних було створено два класи: Creator (відповідає за створення нових сутностей) та NameGenerator (відповідає за створення стрічок, схожих на ім’я). Ми створили

два DAO: MySQLDAO (для роботи з реляційною базою даних) та Neo4j (для роботи з графовою базою даних). Встановлення з'єднання з MySQL проводилося з використанням JDBC driver (Java DataBase Connectivity) – технології, яка дозволяє взаємодію java коду з базою даних, використовуючи URL до бази даних, логін та пароль користувача. Для встановлення зв'язку з Neo4j достатньо мати необхідні бібліотеки та створену базу даних. Набір цих бібліотек можна знайти в папці з інстальованим Neo4j сервером.

Реалізація методів у DAO принципово відмінна. Для роботи з реляційною базою даних використовуються PreparedStatement, створені на основі SQL запитів. Роботу з Neo4j забезпечували бібліотечні методи [9]. Для об'єктивності аналізу потрібні великі обсяги даних. На невеликій вибірці різниця може бути не суттєва. Тому необхідно згенерувати дані для MySQL не менше ніж на 300 мегабайт. Для цього було створено клас MySQLDatabaseInfilling. Дані з MySQL копіювалися за допомогою розробленого класу Neo4jDatabaseInfilling. Враховуючи, що кожна таблиця в MySQL мала властивість auto increment, ми можемо отримувати значення з бази даних за ідентифікатором. Після заповнення обох баз даних було виявлено, що місце, потрібне для того, щоб зберігати одні й ті самі дані, суттєво відрізняється: для MySQL необхідно 351,5 МБ, а для Neo4j – 3,45 ГБ. Після заповнення баз даних потрібно вибрати критерії для порівняння. Ми зробили наголос на ефективності пошуку в колекції та складності написання запитів.

3.3.2 Аналіз порівнянь

Аналіз був проведений на основі трьох дослідів. Дослід 1. Вимірювання часу пошуку користувача за його ідентифікатором. Щоб зібрати статистичні дані, потрібен метод, який вимірюватиме затрачений час на пошук у реляційній і нереляційній базі даних.

Дані цього експерименту подано в таблиці 3.5. Діаграму, побудовану на основі результатів проведеного досліду, зображено на рисунку 3.9:

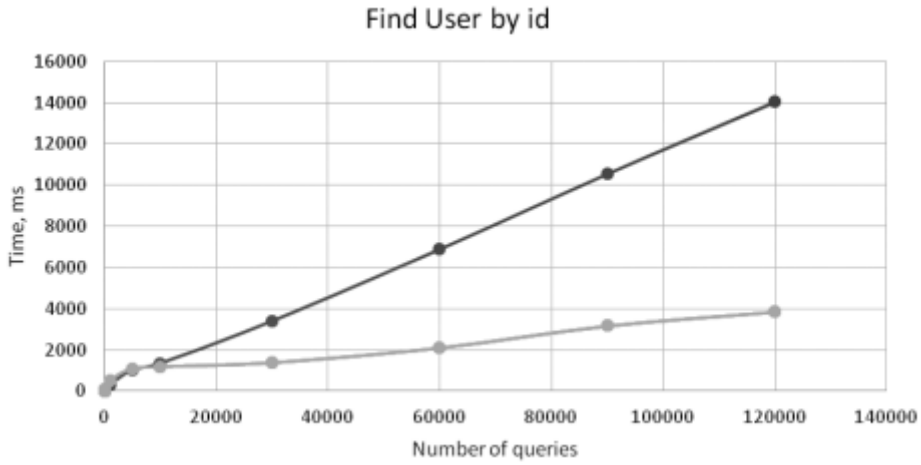

Рисунок 3.9 – Порівняльний графік

Таблиця 3.5 - Залежність часу від кількості запитів

Кількість користувачів = 1000000		
Кількість запитів	Час MySQL, мс	Час Neo4j, мс
10	4	9
100	34	76
1000	286	510
5000	1034	1103
10000	1340	1187
30000	3390	1384
60000	6876	2102
90000	10537	3175
120000	14033	3858

Дослідивши попередню діаграму, можна зробити висновок, що пошук за індексованим полем на великій кількості даних у Neo4j набагато швидший, ніж у MySQL (більше ніж у 3 рази). Дослід 2. Вимірювання часу знаходження друзів користувача зі зміною величини інтервалу входження ідентифікаторів для пошуку. Для отримання статистичних даних було використано аналогічний

метод, як і в досліді 1. Таблиця 3.6 демонструє результати пошуку друзів користувача за ідентифікатором з проміжку від 0 до 5000, на рисунку 3.10 представлено ці результати у вигляді діаграми:

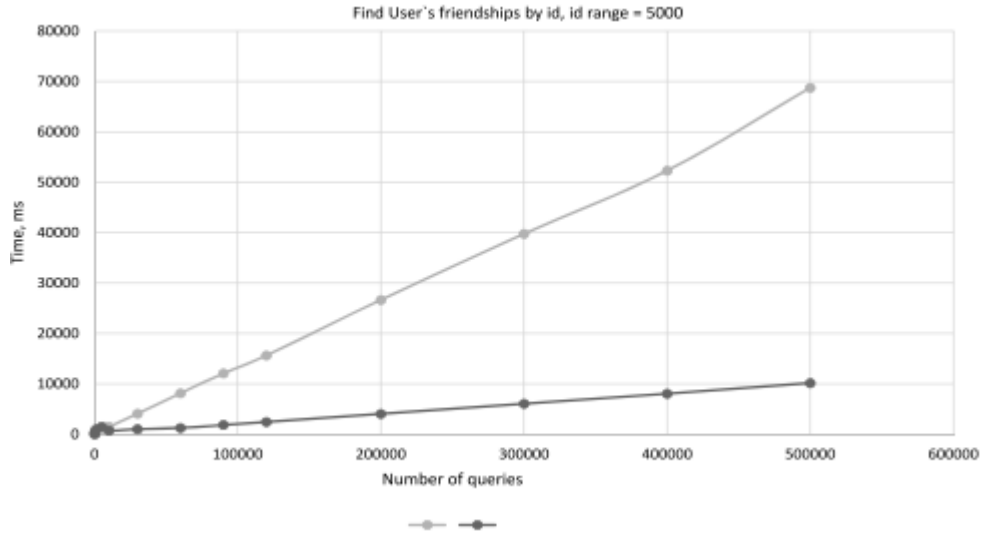

Рисунок 3.10 Порівняльний графік

Таблиця 3.6 - Залежність часу від кількості запитів на вибірці до 5000

Кількість користувачів = 1000000, вибірка до 5000		
Кількість запитів	Час MySQL, мс	Час Neo4j, мс
10	4	68
100	37	367
1000	249	975
5000	1025	1521
10000	1478	723
30000	4082	1008
60000	8132	1248
90000	12065	1848
12000	15622	2433

Таблиця 3.6 - Залежність часу від кількості запитів на вибірці до 5000 (закінчення)

Кількість користувачів = 1000000, вибірка до 5000		
Кількість запитів	Час MySQL, мс	Час Neo4j, мс
200000	26639	4043
300000	39758	6049
400000	52290	8037
500000	68743	10148

Аналогічним чином було отримано дані для проміжків від 0 до 250000 та від 0 до 500000. На основі них було побудовано діаграму, яка зображена на рисунку 3.11.

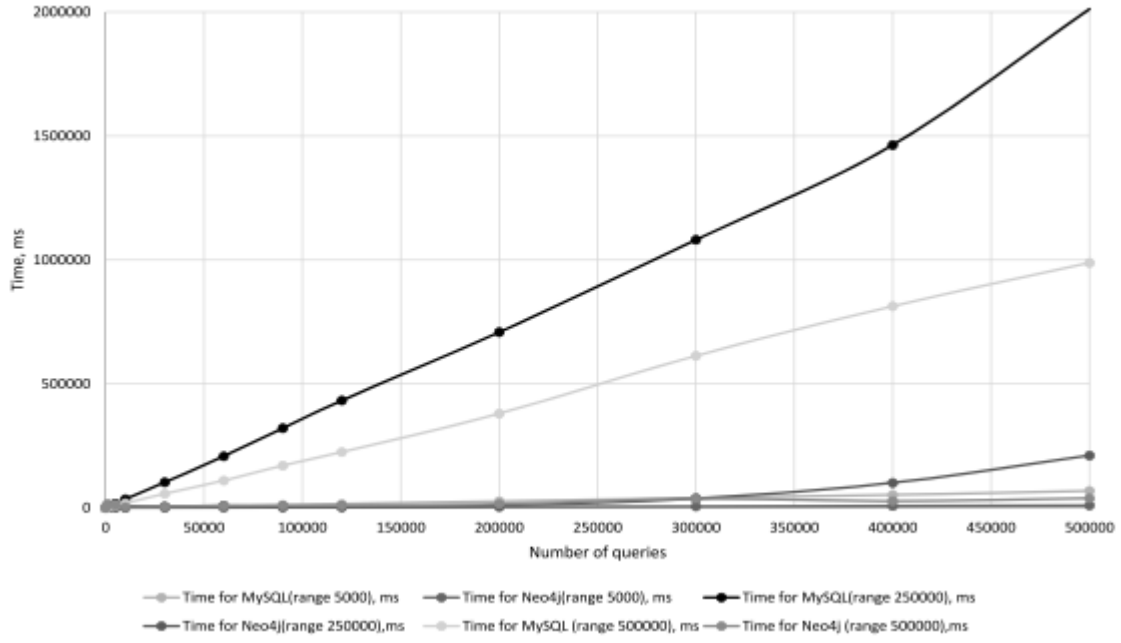

Рисунок 3.11 Порівняльний графік

Проаналізувавши діаграму, зображену вище, бачимо, що час пошуку в Neo4j на великій кількості даних кращий, ніж у MySQL. Також на одних і тих самих даних пошук швидший, якщо робити вибірку з невеликого діапазону. Це

можна пояснити роботою оптимізаторів. Збільшення діапазону не завжди спричиняє збільшення часу пошуку.

3.4 Недоліки графових баз даних

Графові бази даних будуть дуже корисними, як вказувалося раніше, коли потрібно переходити та досліджувати відносини між елементами. Графи дуже гнучкими, коли розмова йде про зміні час від часу взаємозв’язків між даними, або коли потрібно змінити характеристики певних елементів. Але графи дуже сильно залежать від предметної області, на якій проектуються.

Тому вони не можуть бути найкращим вибором в управлінні величезними списками речей. Наприклад, якщо потрібно спроектувати базу даних, яка допоможе в аналітиці транзакцій клієнтів (де є необхідність в створенні бази з 1 мільйону клієнтів, 50 мільйонів транзакцій за один день), то вибір графів не є найкращим. Це відбувається саме через дуже довгий час обробки запитів в моделі даних зі схожою структурою. Також використання графої БД не дуже підходить для роботи з базою даних без зв’язків (без первинних ключів.)

Недоліки саме Neo4j як графової бази даних:

- як було вказано раніше робота з великим обсягом записуючих операцій – не найкраща сторона Neo4j;
- невисокі показники в роботі з повностекстовим пошуком;
- вона не призначена для роботи з мережевою файловою системою;
- не найкраще рішення для роботи з графічними матеріалами та відео.

3.5 Висновки

В цьому розділі були розглянуті деякі з існуючих графових баз даних. Була розкрита архітектура графової БД на прикладі сучасної та популярної

Neo4j. Був спроектований власний приклад використання та проведе тестування для оцінки переваг порівняно з іншими класичними методами.

4 СУЧАСНЕ ЗАСТОСУВАННЯ ГРАФОВИХ БАЗ ДАНИХ

4.1 ARIADNE: Система слідкування відносин в LHCb

LHCb (від англ. Large Hadron Collider beauty) - один із семи експериментів (ALICE, ATLAS, CMS, TOTEM, LHCb, LHCf і MoEDAL) з фізики частинок на Великому Адронному Колайдері (LHC) – прискорювачі частинок у Європейській організації ядерних досліджень CERN, Швейцарія. LHCb є експериментом, спеціалізованим на b-фізиці, призначений для вимірювання параметрів порушення CP-симетрії у взаємодіях b-адронів (адрони, що містять b-кварк). Такі дослідження можуть допомогти пояснити асиметрію речовини-антиречовини у Всесвіті. Детектор здатний проводити вимірювання перерізів утворення за рахунок електрослабкої фізики для вперед направлених процесів.

Обробка даних в LHCb передбачає собою формування гетерогенних метаданих та відносини між ними. Технологічний процес обробки даних повністю залежить від того, чи є дійсним поєднання наборів програмних забезпечень для конкретного кроку робочого процесу [10].

Справа в тому, що відносини між програмними забезпеченнями, які визначають комбінацію, не відслідковуються систематично, а управляються в основному в ручному та в дуже обмеженому режимі. Це може привести до різноманітних проблем в ході роботи LHCb.

При обробці даних в LHCb, функціональність додатків є основою програмного забезпечення, вони виконують специфічні дії на спеціалізованих етапи обробки. Розглянемо деякий функціонал додатків:

- Moore - використовується для високого рівня спрацьовування;
- Brunel - використовується для реконструкції подій;
- DaVinci - використовується для відбору подій і аналізу даних;
- Gauss - використовується для генерації подій і моделювання детектора;
- Boole - використовується для моделювання відгуку детектора і зчитування електроніки.

Ще один важливий аспект призначення додатків виходить з того, що модель обробки даних LHCb вимагає відтворюваності аналізів, яке заключає в собі підтримання версії даних. Також існує база даних станів об’єкту.

Фізичні дані LHCb, реальні дані, зібрані за допомогою детектора, або даних методом Монте-Карло, отримані при моделюванні детектора, проходять через робочий процес, який складається з декількох етапів обробки певним функціоналом, зазначених в пункті. Конфігурація кроків обробки може відрізнятися і ідентифікується типами обробки.

Головним критерієм є сумістність наборів функціональних додатків, що обробляють метадані. Проблема, сформульована в попередньому пункті, вимагає від системи відслідковувати чи є сумісними пара додатків для обробки даних, крім цього потрібно відслудковувати версії цих додатків.

Обробка всіх видів додатків і їх зв'язків в загальному вигляді вимагає модель даних, яка є досить абстрактною. Всі метадані, які містять інформацію про додатки та їх властивостіпоказані, можливо показати у вигляді графу. Елементи графа можна визначити наступним чином:

- Вузол як додаток метаданих
- Ребро як відносини між об'єктами метаданих

В якості прикладу, всі типи сумісності і додатки, можуть бути синтезовані в вигляді графа, показаного на рисунку 4.1. Легко бачити, що граф відображає будь-які види додатків і зв'язків, а також будь-який рівень.

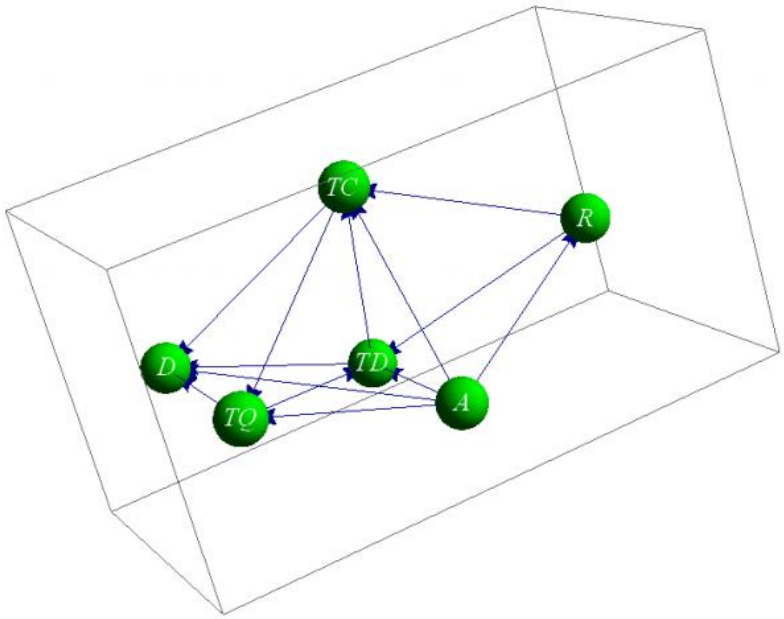

Рисунок 4.1. Граф додатків [10]

A-вузол являє собою додаток, T (D, C, Q) -вузли - три стани, в той час як D- і R-вузли - моделі детектору і обробки даних типів. Повний список атрибутів вузлів цього графу наведено в таблиці 4.1. Атрибути вузла графа показано на рисунку 4.1. Також є додаткові атрибути, які відіграють допоміжну роль, дозволяючи деталізувати дозволи додатків на обробку даних.

Таблиця 4.1

Тип вузла графу	**Основні атрибути**	**Додаткові атрибути**
Application	name, version	release date
CondDB substate	tag name, partition	release date, payload hush sum
Detector type	type name	-
Data processing type	type name	-

Таким чином, повертаючись до випадку проблема сумісності, якщо значення всіх атрибутів графа (рисунок 4.1) дозволяються системою, задовольняють набору обмежень, то граф стає конкретним рішенням сумісності. Тому всі метадані можуть бути представлені у вигляді великого графа знань, в якому вони з'єднані один з одним відповідно до предметної області. Ariadne була розроблена з використанням графої моделі даних для вирішення зазначених вище проблем. Ariadne здатна накопичувати гетерогенні примітиви метаданих в загальному графі знань, а також надає засоби для аналізу графа знань. Архітектура розробленої системи показано на рисунку 4.2. Блок-схема даних системи показує, що система очікує отримати на вхід примітиви метаданих, які можуть бути:

- новими додатками метаданих (або редагування існуючих);
- новими відносини в метаданих (або редагування існуючих).

Вихідний сигнал являє собою рішення, отримане згідно обмежень, описаних в запиті. Рішення зазвичай являє собою граф – який показує яким набором додатків система може обробити дані.

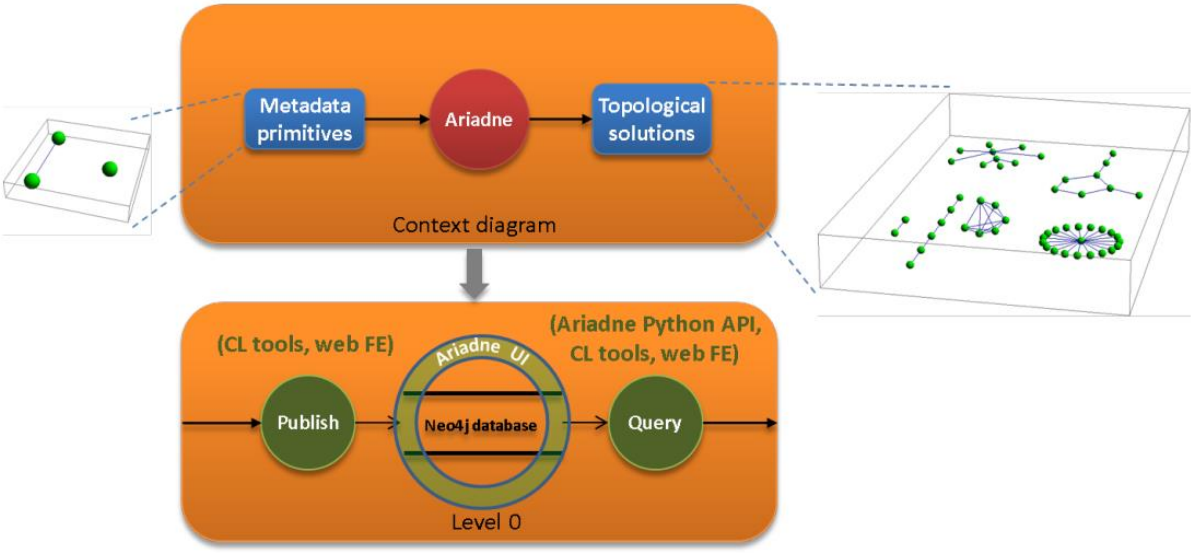

Рисунок 4.2 - Діаграма потоків даних Ariadne [10]

Підпростір графу метаданих LHCb, які зберігаються в базі даних, візуалізується на рисунку 4.3. У даний час весь граф знань містить близько 600 вузлів, з'єднаних навколо 50000 відносин.

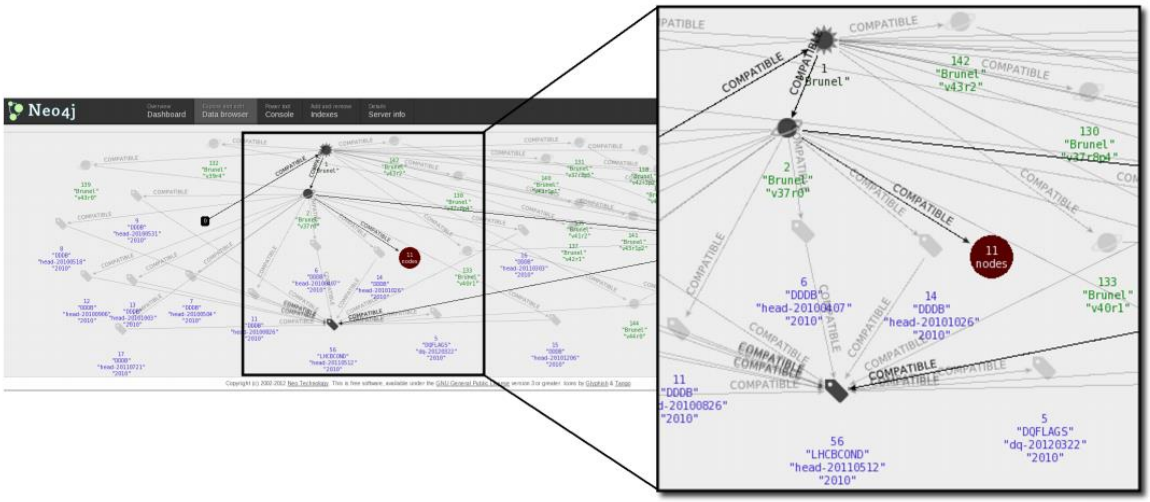

Рисунок 4.3. Інтерфейс веб-адміністрування, що входить в комплект поставки Neo4j, забезпечує інтерактивну середу для перегляду, додавання або змінення вузлів, відносин і їх атрибутів [10]

4.2 Аналіз соціальних відносин

Останнім часом соціальні мережі займають стале місце в нашому житті. Це проекція нашого приватного життя у світі інтернету. Наші вподобання, інтереси та відношення до деяких речей – все це просто дослідити, потрібно зайти тільки на вашу особисту сторінку. Це може бути корисне сфері торгівлі або послуг, навіть для проектування та дослідження соціальної поведінки. Зберігати такі дані найкраще у вигляді графу та з звичайно з використання графової бази даних. Розглянемо який аналіз можливо провести на даних із серії книг «Пісня льоду й полум'я».

4.2.1 Дані

Почнемо, звичайно ж, з головного - даних. У нас є граф соціальної активності, вказаний на рисунку 4.4.

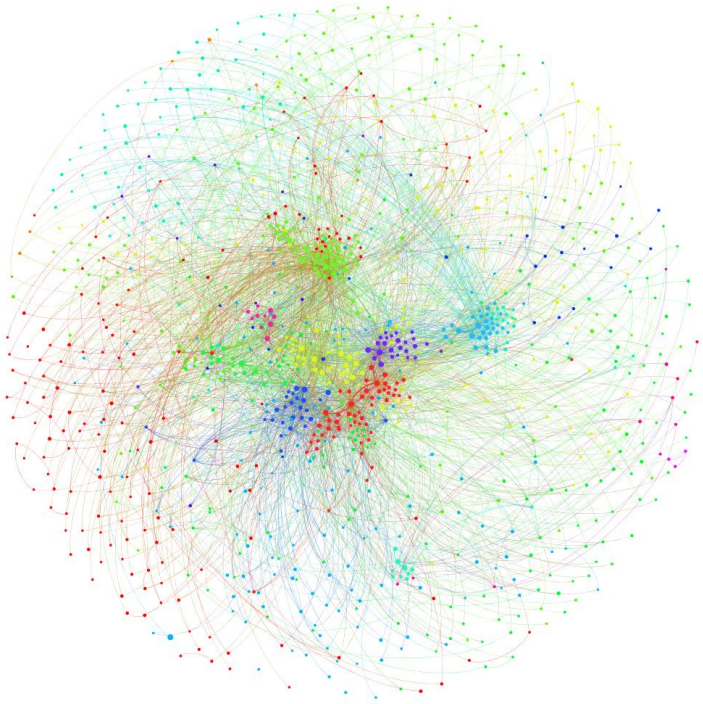

Рисунок 4.4 – Соціальний граф

На ньому зображені персонажі зі світу престолів в вершинах графа, де розмір вершини залежить від сказаних ним слів, і діалогів персонажів на ребрахграфа, де товщина ребра залежить від кількості проведених розмов.

Загальні відомості про граф:

- Граф неорієнтовний;
- Вершин (персонажів): 1105;
- Ребер (діалогів): 3505.

Складання списку всіх розмов, визначення їх залежностей (в 5-ти книгах майже 2 мільйони слів), було проведено за допомогою методу умовних випадкових полів [11]. Так, для збору даних було залучено машинне навчання і, як заявляє «тренер» моделі, точність визначення приналежності розмови становить близько 75%. Отже маємо дані, які далі імпортуємо в графову базу даних Neo4j. Алгоритми будуть реалізовані на мові C ++.

4.2.2 Степінь вершини

Для початку, спробуємо реалізувати щось цікаве та просте , що буде цікаво читачеві. Знайдемо ступінь кожної вершини. Ступінь вершини - це кількість ребер, інцидентних (примикають до) вершин. Цей параметр в контексті графа, який ми маємо, покаже нам скільки ж «друзів» у персонажу. Це можна зробити за один прохід і складність цього алгоритму O (V + E) завдяки засобам Neo4j. Топ 10 багатозв'язкових персонажів:

Тіріон Ланністер: 168

Джон Сноу: 128

Ар'я Старк: 104

Джеймі Ланністер: 102

Серсея Ланністер: 86

Кейтілін Старк: 85

Теона Грейджой: 76

Дайнеріс Таргаріен: 73

Брієнна: 71

Санса Старк: 69

Цікаво те, що в топі не виявилося того, хто знайомий з багатьма і по займаній посаді зобов'язаний підтримувати контакт з людьми, Варіс (він на двадцять п'ятому місці). А ось, здавалося б, другорядний персонаж Брієнна, від імені якої глави ще немає, на дев'ятому місці.

4.2.3 Теорія рукостискань

Наступне що ми знайдемо за допомогою алгоритму пошуку вже в ширину - максимальне число рукостискань в графі, іншими словами діаметр графа, тобто найдовші з найкоротших шляхів між усіма вершинами графа. Як виявилося, максимальна кількість рукостискань - це 8. Непоганий результат для твору про «середньовічних століттях», якщо враховувати теорію 6-ти рукостискань. Для прикладу, одна з таких ланцюжків зв'язку:

Матінка Кротиха (Mother Mole) - Репейниця (Thistle) - Варамір (Varamyr) - Джон Сноу (Jon Snow) - Еддард Старк (Ned Stark) - Баррістан Селмі (Barristan Selmy) - Квентін Мартелл (Quentyn Martell) - Принц-Оборванець (The Tattered Prince) - Льюїс Ланстер (Lewis Lanster)

Такий алгоритм працює за $O(V * V + V * E)$. Так як потрібно запустити алгоритм BFS з кожної вершини графа. А раз знайшли найдовші шляхи для всіх вершин графа, то можна обчислити і середнє значення, а також скласти розподіл максимальних шляхів. Середнє значення для довжини максимальних шляхів виявилося 6,16 (в середньому цілком вписується в теорію про 6ти рукостискання), а загальну середню відстань 3,6, для порівняння у Фейсбук цей параметр 4.57. Розподіл довжин максимальних шляхів вказаний на рисунку 4.5.

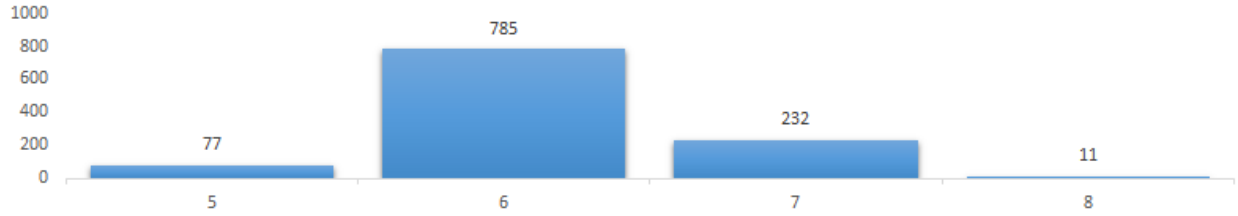

Рисунок 4.5 – Розподіл довжин максимальних шляхів

Якщо поглянути на розподіл, то бачимо, що 77 персонажів знаходяться «в центрі» графа, іншими словами вони можуть зв'язатися з будь-яким іншим персонажем не більше ніж через 4х інших. Серед них всі основні персонажі історії, крім Дайнеріс Таргаріен і Санса Старк, а серед тих, кому в книгах присвячено менше п'яти глав потрапили в список Баррістан Селмі, Джон Коннінгтон і Мелісандра.

4.2.4 Кліки в графі

Алгоритм Брона-Кербоша дозволяє знайти максимальні кліки в графі, іншими словами підмножини вершин, будь-які дві з яких пов'язані ребром. У контексті розглянутого графа це дозволить знайти сильно пов'язані компанії, які спілкувалися між собою в історії. Складність алгоритму лінійна щодо кількості клік в графі, але в гіршому випадку вона експоненційна *O (3V / 3),* алгоритм вирішує NP повну задачу все таки. Сам по собі алгоритм являє рекурсивную функцію, яка для кожної вершини знаходить максимальну кліку, тобто таку, в яку не можна додати жодної іншої вершини. Розподіл обсягів клік показаний на рисунку 4.6.

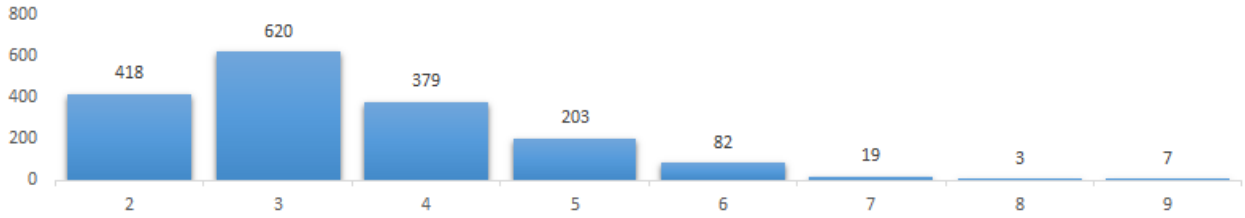

Рисунок 4.6 – Розподіл обсягів клік

Як видно, максимальна кліка розміром в 9 персонажів. Наприклад одна з таких - компанія Ланністерів: Тіріон, Джеймі, Серсея, Варіс, Тайвін, Кеван, Піцель, Петіро і Мейс Тіреллі. Що цікаво, всі кліки розміру 8 і 9 сформовані в Королівській Гавані або поблизу неї. А максимальна кліка з Дайнеріс розміру 5.

4.2.5 Кістякове дерево

А тепер трохи спростимо наш граф зв'язків, залишивши в ньому тільки «кістяк», тобто найбільш важливі ребра зв'язку і при цьому перетворивши граф в дерево з усіма вихідними вершинами. Допоможе нам у цьому алгоритм Крускала. Алгоритм жадібний, для його здійснення потрібно всього лише впорядкувати всі ребра по їх вазі і по черзі додавати кожне ребро, якщо воно пов'язує дві компоненти. Якщо використовувати правильну структуру даних (зазвичай використовують систему непересічних множин), то складність алгоритму вийде *O (E* (E-1)*. Нижче наведено результат графа, який піддався цим маніпуляціям на рисунку 4.7. Для наглядності видалені ребра з вагою 1.

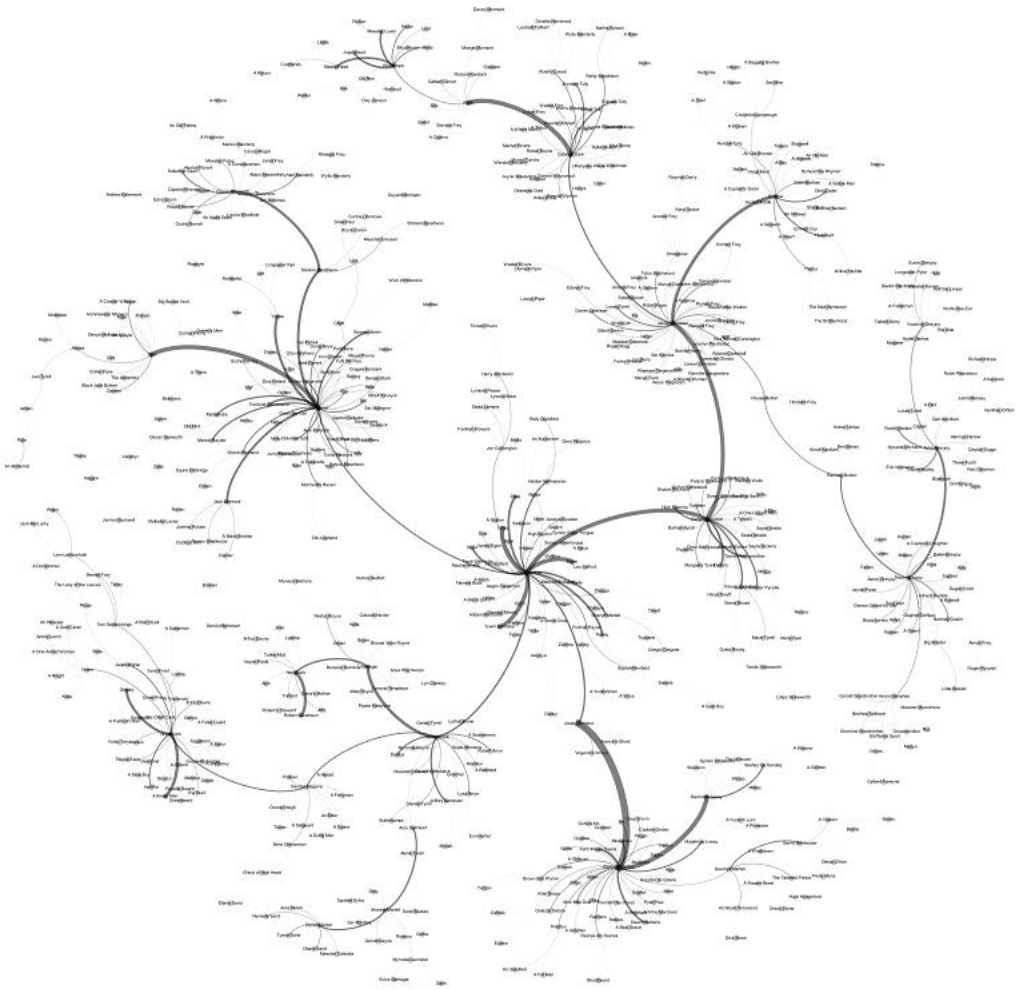

Рисунок 4.7 – Граф після перетворень

Отже, з цього графа можна побачити головних героїв і їх зв'язок між собою. Тіріон Ланністер знаходиться в «центрі» всіх зв'язків, від нього виходять 4 великі гілки: Серсея Ланністер, Джон Сноу, Санса Старк і Джорах Мормонт (гілка Дайнеріс). Останні в свою чергу герої наступного рівня зі зв'язків.

4.2.6 Висновки

Як бачимо використання теорії графів в аналізі соціальних даних дуже корисне, в результаті якого можна отримати будь-які потрібні характеристики.

Дані можна були представити у різних структурах, але вибір саме графової бази даних Neo4j набагато спростив складність алгоритмів саме із-за своєї архітектури (більш детально розділ 3.2).

4.3 Проектування асфальтних доріжок

Проблема в проектуванні асфальтних доріжок на прибудинковій території завжди актуальна. Планувальники цих доріг часто ігнорують раціональне розміщення, в наслідок чого люди витоптують свої шляхи.
Для початку розглянемо, як взагалі місцеві органи влади, що відповідають за благоустрій, борються з проблемою ходіння по газонах.

Варіант перший: ставити паркани.
У більшості випадків неефективний спосіб. Люди будуть знаходити інший шлях, або просто зруйнують паркан.

Варіант другий: визнати помилку і виправити.
Прокласти асфальт або інший матеріал на вже створені пішоходами доріжки.

Цей варіант залежить тільки від місцевої влади.

Ну і варіант третій: передбачити і виправити проблеми ще на стадії проектування. На ньому зупинимося, для огляду представлено опис алгоритму і пара прикладів його роботи для реальних прибудинкових територій.

4.3.1 Постановка задачі

Є карта місцевості, на якій присутні:
Ділянки з різним ступенем прохідності і привабливістю для пішоходів (доріжки, газони) , перешкоди (будинки, паркани) , місця між якими пішоходи пересуваються - під'їзди, магазини, тощо. Їх будемо називати генераторами пішоходів. Необхідно передбачити, де саме пішоходи будуть ходити по газонах, щоб на основі цього можна було прийняти рішення про створення на місці стежок нормальних доріг.

Відразу відзначимо два випадки, які зазвичай першими приходять в голову в якості рішення. Перший - це повний граф, тобто ми просто візьмемо і з'єднаємо кожен генератор з кожним, проклавши дороги по прямій. На словах просто, пішоходам буде, зрозуміло, зручно, проте в підсумку вся територія виявиться в асфальті, що дорого і неестетично. Другий випадок - це мінімальний кістяк графа, вершинами в якому будуть генератори. На жаль, люди не ходять по мінімальним кістяковим деревам, що було показано дослідниками роботі «Modelling the Evolution of Human Trail Systems» (D. Helbing, J. Keltsch, P. Molnar.) Ілюстрацією до цього твердження може послужити наступний рисунок 4.8.

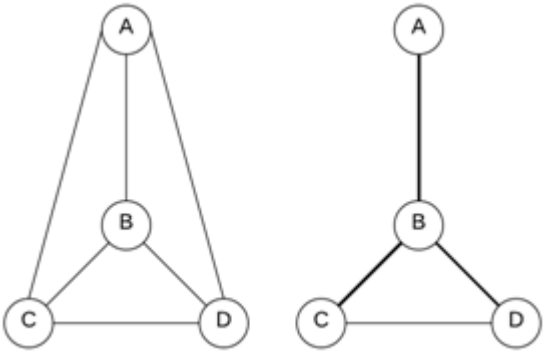

Рисунок 4.8 – Схема шляхів у виді графа

Зліва показана система повних шляхів, а праворуч - отримана в результаті експериментів зі згаданої вище роботи система стежок між зазначеними точками. Видно, що вона є ні повною, ні кістяковим деревом. В результаті деяких роздумів і вивчення статей на тему симуляції руху пішоходів був придуманий і реалізований наступний алгоритм.

4.3.2 Алгоритм

Карта розглянутої ділянки території представляється в форматі GeoJSON. На ній вручну розмічаються генератори пішоходів і прохідність ділянок ландшафту.

На основі карти будується навігаційний граф *G (V, E)*, який для буде потім буде завантажено у графову базу даних Neo4j. Вершини графа *V* - це безліч точок місцевості. У даній роботі вибрано найбільш простий метод побудови цієї безлічі: на карту накладається прямокутна сітка, її вузли стають вершинами графа. Якщо між двома сусідніми вузлами сітки може пройти людина, такі вузли з'єднуються ребрами, складовими безліч E. Вихідний вага кожного ребра E представлений у вигляді різниці двох компонент: фіксованої Wconst (E), яка визначається типом місцевості, і змінною *Wvar (E)*, яку в надалі будемо називати витоптаністю [12].

Спочатку вона дорівнює нулю. У деяких типів ландшафту (доріжок з твердим покриттям) змінною компоненти може не бути.
Витоптаність обмежена знизу нулем (недоторканий газон), а зверху числом *Wmax*. Крім цього, для кожного пішохода p вводиться коефіцієнт порядності k(p). Цей коефіцієнт використовується для симуляції різних категорій пішоходів - як «порядних», що вважають за краще завжди ходити тільки по доріжках, навіть якщо відстань буде значно більше, так і любителів ходити завжди навпростець. Формула для обчислення ваги *W (e, p)* ребра *e* для пішохода *p* має вигляд:

$$W(e,p) = W_{\text{const}}(e) - k(p) * \min\left(W_{\max}, W_{\text{var}}(e)\right)$$

У такому випадку при значенні *k (p)> 1* короткі випрямлені стежки для даного пішохода матимуть більшу привабливість, так як роль компоненти *Wvar* у формулі буде вище. При *k <1* виходить «порядний» пішохід, який буде намагатися частіше ходити по доріжках.

P пішоходів рівномірно розподіляються по генераторам. Цілі для них вибираються випадковим чином зі списку інших генераторів. Коефіцієнт порядності *k (p)* вибирається в такий спосіб:

$$k(p) = \begin{cases} 0.5 \text{ с вероятностью } K_{bad}; \\ N(1, 0.7) \text{ с вероятностью } (1 - K_{bad}). \end{cases}$$

Тут *Kbad* - обов'язкова частка непорядних пішоходів (для прискорення збіжності алгоритму припустимо, що в суспільстві завжди є певна частка любителів пересуватися максимально прямими і короткими шляхами), *N* - нормальний розподіл. Значення коефіцієнтів підібрані емпіричним шляхом, для *Kbad* вибрано значення 0.1.

На кожному кроці симуляції виконуються наступні дії:

1) пішоходи проходять певну відстань, залежне від заданої швидкості руху;

2) витоптаність пройдених пішоходами ребер графа збільшується на фіксовану величину *ΔWped* за кожного пройденого по ним пішохода;

3) дійшли до своєї мети пішоходи замінюються на нові;

4) витоптаність всіх ребер графа зменшується на фіксовану величину *ΔWtime* (заростання згодом).

Таким чином, загальна зміна витоптаністі для ребра *E* після кроку симуляції і виглядає як:

Тут *Pcount (e, i)* - число пішоходів, які пройшли на і кроці по ребру *e*.

$$\Delta W(\mathrm{e}) = \Delta W_{ped} * P_{count}(e, i) - \Delta W_{time}$$

Пішоходи прокладають маршрут за допомогою алгоритму A* с евристикою для випрямлення шляхів (спочатку використано алгоритм Дейкстри, але на прямокутних сітках він любить генерувати досить неприродні шляхи).

Симуляція триває або до збіжності, поки карта стежок не перестане змінюватися, або задане число кроків. Після завершення симуляції карта з розподілом витоптаності демонструє ділянки, на яких пішоходи найчастіше сходять з доріжок на землю, і які варто покрити твердим покриттям.

4.3.3 Приклад роботи

Як приклад роботи приведено обробка ділянки території біля житлового будинку. На рисунку 4.9 перебуває розмічена карта місцевості із зазначенням контурів будинків і перешкод.

Рисунок 4.9 Схема будинку [12]

В результаті симуляції після декількох сотень ітерацій вийшов результат показаний на рисунку 4.10. Чорним відзначені передбачені програмою місця витоптування газонів.

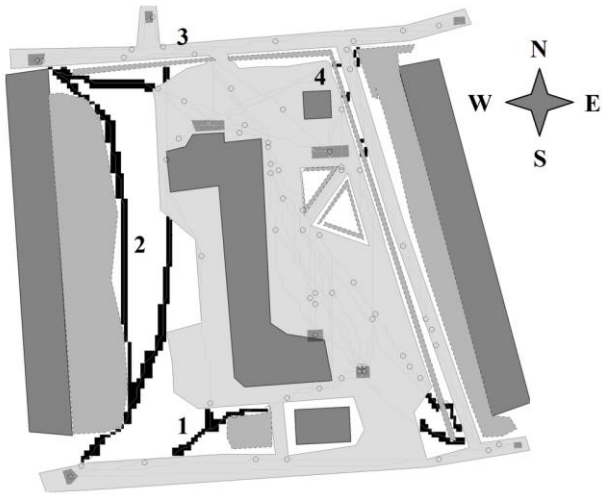

Рисунок 4.10 Результат програми [12]

Для порівняння, фото з реальності, зображені на рисунках 4.11, 4.12, 4.12 відповідно:

Рисунок 4.11 - Стежка по діагоналі до рогу будинку [12]

Рисунок 4.12 - Стежка вздовж усього будинку [12]

Рисунок 4.12 - Стежка від під'їзду [12]

В цілому можна бачити, що передбачення програми досить точні. Іншим прикладом було розглянуто план парку. Результати програми показаний на рисунку 4.13.

Рисунок 4.13 Результат програми [12]

Для порівняння фото з супутника, витоптані стежки дуже добре видно на рисунку 4.14.

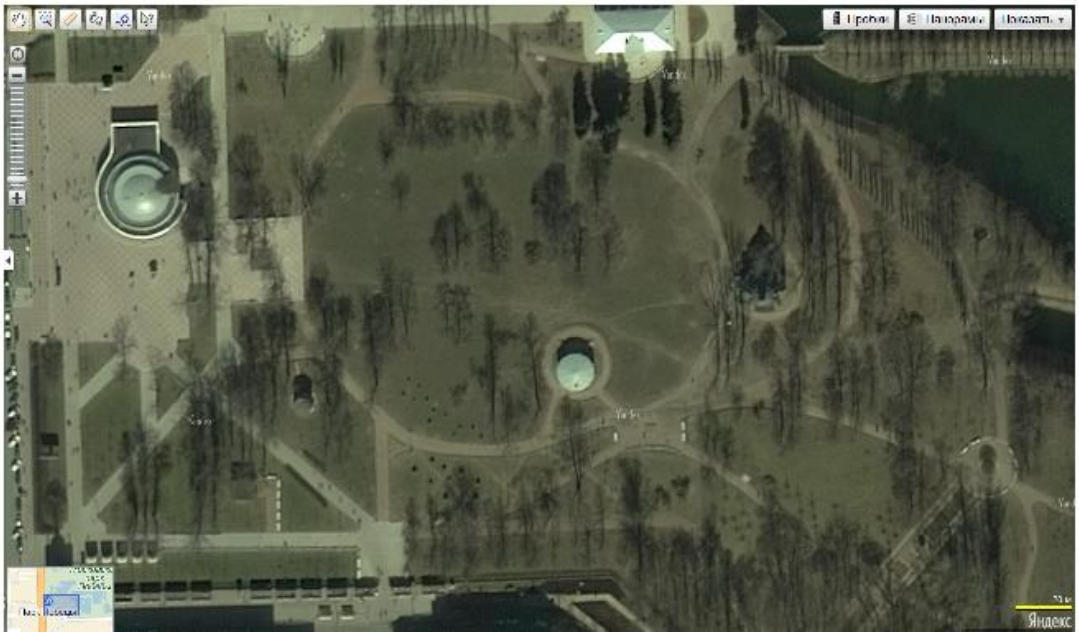

Рисунок 4.14 Результат програми [12]

Звичайно на цьому знімку не видно більш дрібніх стежок, але вони там є (насправді їх більше ніж передбачила програма, але в цілому конфігурація дуже схожа).

4.3.4 Висновки

Якщо повністю реалізувати програму з усіма деталями з використання графової бази даних Neo4j, та потім впровадити її на стадію проектування доріг, можна буде уникнути витоптування газонів та спотворення зеленої території біля будинків.

4.4 Використання графових БД в обробці панамських документів

4.4.1 Введення

Панамські документи (англ. Panama Papers) — це витік конфіденційної інформації юридичної фірми Mossack Fonseca, яка спеціалізується на роботі з офшорами. Цей витік вперше розкрив інформацію про активи та власність політиків та інших публічних діячів.

Панамські документи складаються з 11,5 мільйонів внутрішніх документів фірми Mossack Fonseca. Джерелом витоку є анонімна особа, яка передала документи німецькій газеті «Зюддойче цайтунг» більш ніж за рік до публічного повідомлення про витік. Після того газета поділилася інформацією з Міжнародним консорціумом журналістів-розслідувачів (International Consortium of Investigative Journalists — ICIJ). Документи опрацьовували журналісти 107 ЗМІ з 78 країн. Перші результати розслідувань були опубліковані 3 квітня 2016 року.

International Consortium of Investigative Journalists (ICIJ), «Міжнародний консорціум журналістських розслідувань» — один з проектів Center for Public

Integrity (CPI). Створений в 1997 році. В склад ICIJ входять 160 відомих журналістів з усього світу.

ICIJ викрило багато відомих людей, таких як мільярдери, зірки спорту (як Ліонель Мессі) і політики. Було показано як люди використовують недоліки податкової системи, щоб приховати свої гроші. Розслідування було проведено серед 140 політиків в більш ніж 50 країнах за допомогою спільних зусиль. Серед цих 140 політиків налічувалося 12 діючих або колишніх глав держав. Також ICIJ знайшли усередині 11,5 мільйона файлів ряд кримінальних і корупційних зв'язків [13].

За обсягом пам’яті панамські документи налічували майже 2,7 ТБ. В порівнянні зі світовими обсягами інформації це ніщо.

4.4.2 Рішення

Отже була створена своя соціальна мережа між журналістами, де останні належали до відомої організації для ICIJ або ж знали один одного.

Всередині 2.6 терабайт даних, більшість даних була у вигляді листів. ICIJ також мали багато PDF-файлі і зображень, які повинні були обробити, при класичних підходах займало би багато часу. Статистика даних наведена на рисунку 4.15.

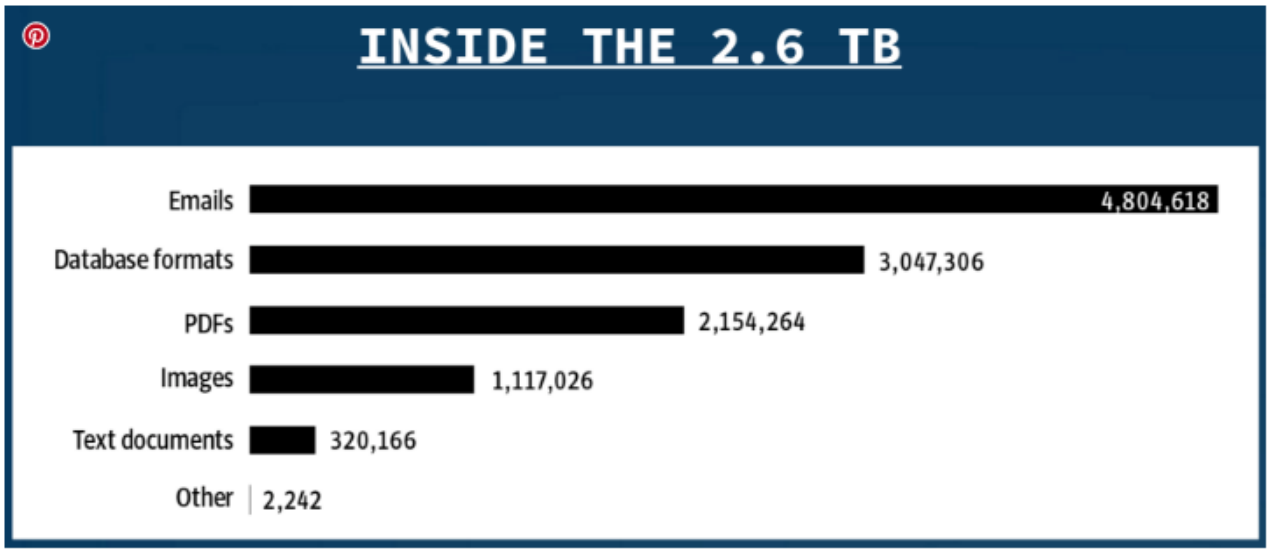

Рисунок 4.15 Статистика даних [13]

Було використано 30-40 серверів, які проводили одночасну паралельну обробку всіх цих документів на базі оптичного розпізнавання символів (OCR). Далі була розроблена програма, яка розпізнавала вміст документів.

Для перегляду документів та спілкування між журналістами потрібно було розробити платформу, надати доступу журналістами до 11,5 мільйона файлів, а також візуально організувати більш 210000 компаній в 21 юрисдикціях і всіх людей, що стоять за ними: акціонери, посередники, адреси і т.д. Можна було виявити, що багато документів, були в форматі бази даних, яка прийшла з внутрішньої бази даних Mossack Фонсекі. ICIJ повинні були реконструювати цю внутрішню базу даних.

Так було випущена в червні 2013 року платформа з простим вікном пошуку. Ви шукайте за ім'ям та отримуєте певну інформацію. Інтерфейс можна побачити на рисунку 4.16.

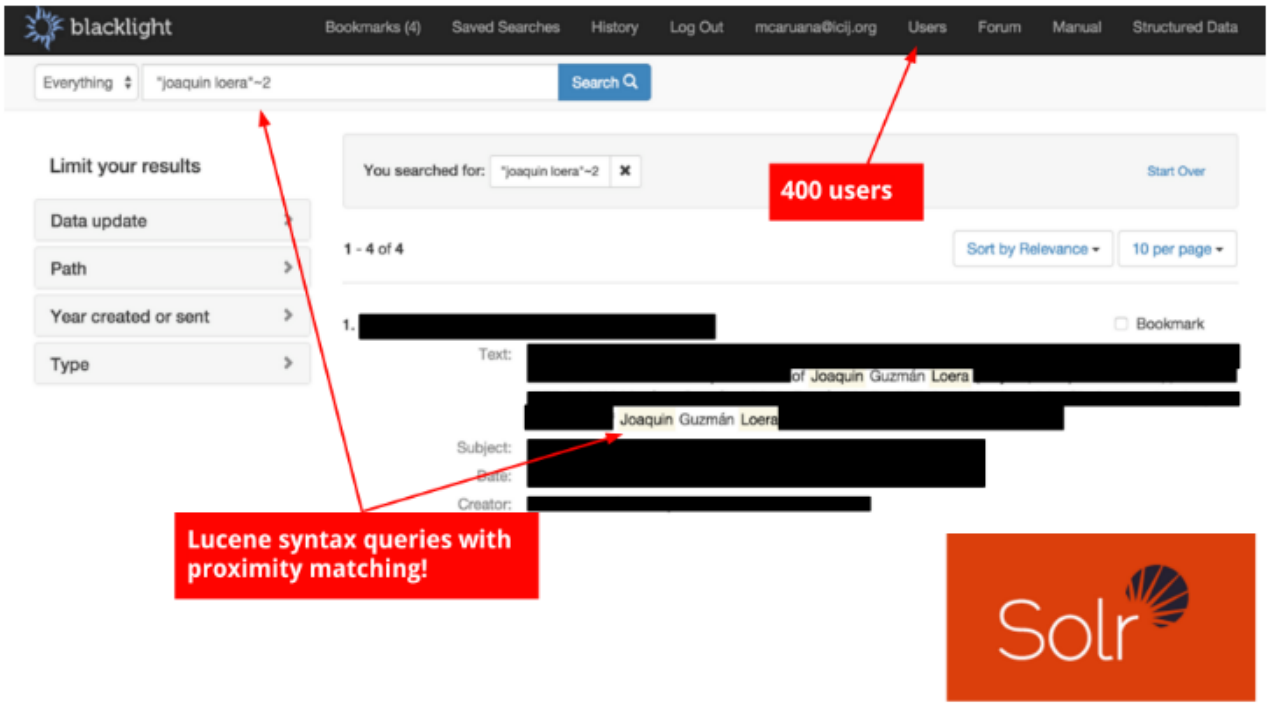

Рисунок 4.16 Первинна версія сайту [13]

Тоді це був просто сайт з базою даних MySQL і з використанням JavaScript бібліотеки, Sigma.js, яка візуалізувала граф, який містив всю інформацію.

ICIJ помітили, що ця візуалізація графа є дуже корисною та зручною. Так було вирішено розвивати напрям використання графів.

Далі було знайдене рішення в використанні Linkurious (програмне забезпечення, яке дозволяє візуалізувати графи). Також було вирішено використовувати графову базу даних Neo4j.

Як вже відомо, бази даних Панамських документів були в SQL, але завдяки інструментам з відкритим кодом - в цьому випадку Talend - було легко трансформувати базу даних в Neo4j. Нижче наведено інтерфейс з використання Linkurious на рисунку 4.17.

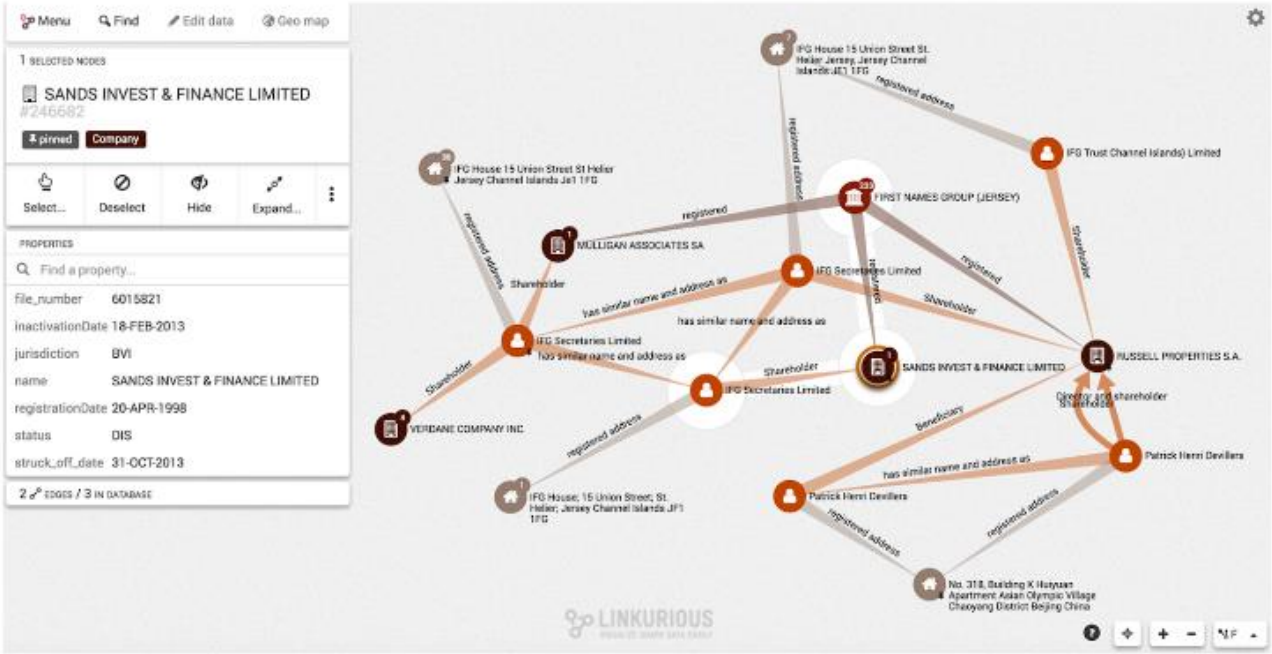

Рисунок 4.17 Нова версія сайту [13]

У лівому верхньому кутку, надана можливість пошуку. Результати пошуку відображаються у вигляді точок, а потім журналісти можуть просто розширити точки. А саме головне, що багато хто з журналістів не надто знаються на технологіях, для них це було дуже зручно.

База даних в кінцевому підсумку містила близько 950000 вузлів і 1,2 мільйона ребер, а розмір її був близько чотирьох гігабайт.

Деякі функції Neo4j були особливо корисними, а саме, наприклад, швидкий пошук найкоротшого шляху. ICIJ також сподобався той факт, що за допомогою Neo4j і Linkurious можна було візуалізувати дані в граф, не показуючи при цьому приватну інформацію, який потім можна було опублікувати на просторах інтернету.

4.4.3 Висновки

На прикладі роботи організації ICIJ було показна корисність використання графової БД Neo4j. Це свідчить про те, що використання графових баз даних полегшують роботу з соціальною інформацією різноманітних напрямків, навіть з приватними даними панамських документів.

4.2 Висновки

Як бачимо використання теорії графів в аналізі соціальних даних дуже корисне, в результаті якого можна отримати будь-які потрібні характеристики.

Дані можна були представити у різних структурах, але вибір саме графової бази даних Neo4j набагато спростив складність алгоритмів саме із-за своєї архітектури (більш детально розділ 3.2).

Якщо повністю реалізувати наведено вище програму з усіма деталями з використання графової бази даних Neo4j, та потім впровадити її на стадію проектування доріг, можна буде уникнути витоптування газонів та спотворення зеленої території біля будинків.

А на прикладі роботи організації ICIJ було показна корисність використання графової БД Neo4j. Це свідчить про те, що використання графових баз даних полегшують роботу з соціальною інформацією різноманітних напрямків, навіть з приватними даними панамських документів.

5 ЕКОНОМІЧНО-ОРГАНІЗАЦІЙНА ЧАСТИНА

В даному розділі проводиться аналіз варіантів реалізації модулю з метою вибору оптимальної, з економічної точки зору. А саме проводиться функціонально-вартісний аналіз (ФВА).

Функціонально-вартісний аналіз — це метод комплексного техніко-економічного дослідження об’єкта з метою розвитку його корисних функцій при оптимальному співвідношенні між їхньою значимістю для споживача і витратами на їхнє здійснення. Є одним з основних методів оцінки вартості науково-дослідної роботи, оскільки ФВА враховує як технічну оцінку продукту, що розробляється, так і економічну частину розробки. Крім того, даний метод дозволяє вибрати оптимальний варіант розв’язання задачі, як з погляду розробника, так і з точки зору покупця. Також він дозволяє оптимізувати витрати й час виконання робіт.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом:

– визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.

– для кожної функції визначаються повні річні витрати й кількість робочих часів.

– для кожної функції на основі оцінок попереднього пункту визначається кількісна характеристика джерел витрат.

– після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

У даній роботі проводиться оцінка основних характеристик програмного продукту призначеного для розпізнавання звукових образів та голосу людини .

5.1 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічний аналізу розробки.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

– програмний продукт повинен функціонувати на персональних комп’ютерах із стандартним набором компонент;

– забезпечувати високу швидкість обробки великих об’ємів даних у реальному часі;

– забезпечувати зручність і простоту взаємодії з користувачем або з розробником програмного забезпечення у випадку використовування його як модуля;

– передбачати мінімальні витрати на впровадження програмного продукту.

5.2 Обґрунтування функцій програмного продукту

Головна функція F_0– розробка програмного продукту, який аналізує процес за вхідними даними та будує його модель для подальшого прогнозування. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

F_1 – вибір мови програмування;

F_2 – вибір оптимальної СКБД;

F_3 – інтерфейс користувача.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

а) мова програмування C#, .NET 4.5.

б) мова програмування Python;

в) мова програмування Java.

Функція F_2:

а) Noe4j;

б) OrientDB;

в) DEX.

Функція F_3:

а) інтерфейс користувача, створений за технологією WPF;

б) інтерфейс користувача, створений за технологією PyQt;

в) інтерфейс користувача, створений за технологією JavaFX.

5.3 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 5.1). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 5.1).

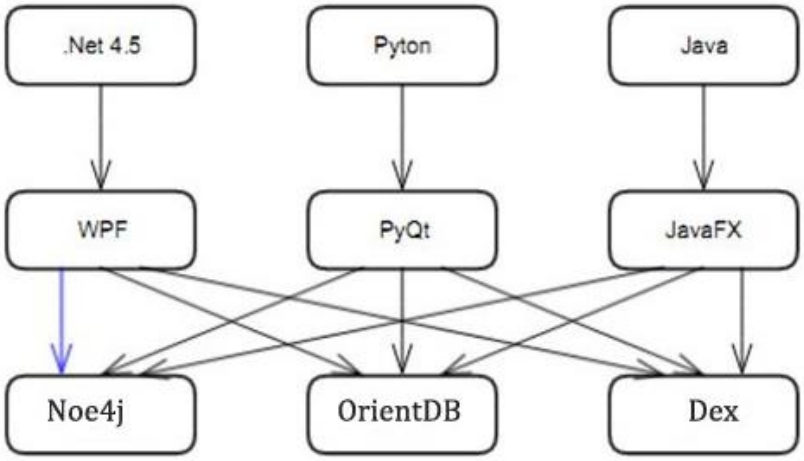

Рис. 5.1 – Морфологічна карта

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

Таблиця 5.1 – Позитивно-негативна матриця

Основні функції	**Варіанти реалізації**	**Переваги**	**Недоліки**
F1	*А*	Займає менше часу при написанні коду	Не кросплатформений
	Б	Безкоштовний	Погана обробка великих об’ємів даних
	В	Кросплатформений	Низька швидкодія
F2	А	Більш дешева вартість корпоративної ліцензіі	Необхідність додаткової інсталяції, низький рівень користувацької підтримки
	Б	Подовжений термін користувацької підтримки	Більш висока вартість корпоративної ліцензії. Необхідність додаткової інсталяції
	В	Простота, багатий функціонал, високий рівень безпеки	Обмежений фунціонал
F3	*А*	Гнучкість интерфейсу, відокремлення бізнес-логіки від користувацького інтерфейсу	Відсутність кросплатформеності
	Б	Кросплатформеність	Не включається із встановленням Python
	В	Простота створення	Відсутність кросплатформеності

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція *F1:*

Оскільки задача вимагає написання та підтримання великої кількості коду необхідно вибрати максимально зручну IDE, для мінімізації часу розробки ПО. Також важливим фактором є швидкість виконання програми та зручний користувацький інтерфейс. Враховуючи ці умови найбільш прийнятним варіантом для побудови інтерфейсної частини є Java FX. А для написання алгоритмів обробки використовуватимемо мову програмування Java. Така комбінація дозволить створити добре розширюваний проект з незалежною серверною частиною та користувацьким інтерфейсом.

Функція *F2:*

Вибір СКБД не відіграє великої ролі для даного програмного продукту, тому вважаємо варіанти а), б) та в) рівноможливими.

Функція *F3:*

Оскільки, програмний продукт реалізується з використанням JAVA використовуємо варіант В (переваги цього варіанту описанні в описі F1).

Таким чином, будемо розглядати такі варіанти реалізації ПП:

1. F1в – F2а – F3в
2. F1в – F2б – F3в
3. F1в – F2в – F3в

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

5.4Обґрунтування системи параметрів ПП

5.4.1 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

– *X1* – швидкодія мови програмування;

– *X2* – об’єм пам’яті для збереження даних;

– *X3* – час обробки даних;

– *X4* – потенційний об’єм програмного коду.

X1: Відображає швидкодію операцій залежно від обраної мови програмування.

X2: Відображає об’єм пам’яті в оперативній пам’яті персонального комп’ютера, необхідний для збереження та обробки даних під час виконання програми.

X3: Відображає час, який витрачається на дії.

X4: Показує розмір програмного коду який необхідно створити безпосередньо розробнику.

5.4.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у табл. 2.1.

Таблиця 5.2 – Основні параметри ПП

Назва Параметра	**Умовні позначення**	**Одиниці виміру**	**Значення параметра**		
			гірші	**середні**	**кращі**
Швидкодія мови програмування	X1	Оп/мс	18000	10000	3000
Об’єм пам’яті для збереження даних	X2	Мб	8128	4096	2048
Час обробки запитів користувача	X3	с	60	20	5
Потенційний об’єм програмного коду	X4	кількість строк коду	15000	10000	7500

За даними таблиці 5.2 будуються графічні характеристики параметрів – рис. 5.2 – рис. 5.5

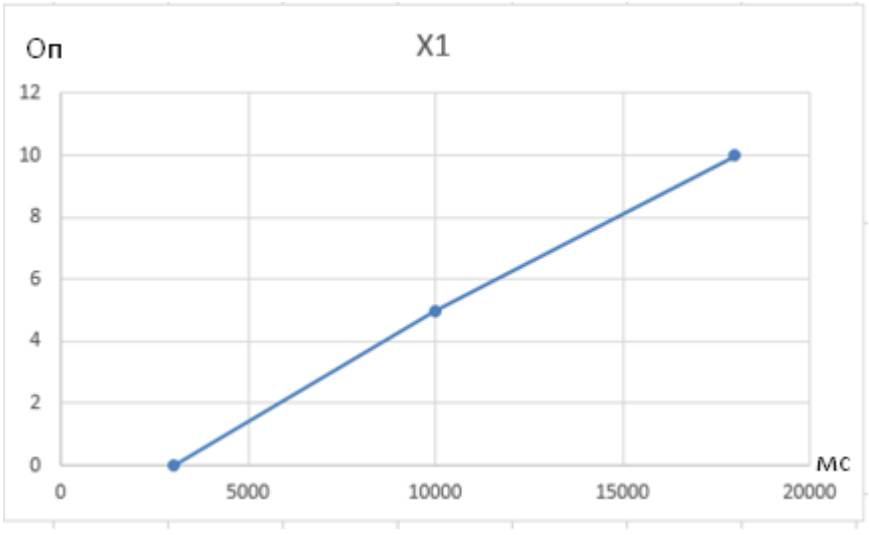

Рисунок 5.2 – X1, швидкодія мови програмування

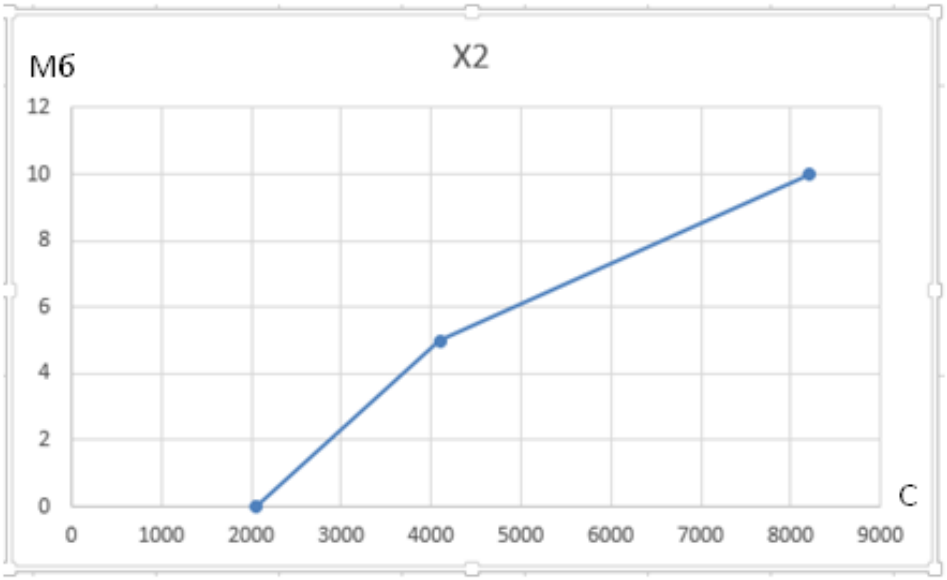

Рисунок 5.3 – X2, об’єм пам’яті для збереження даних

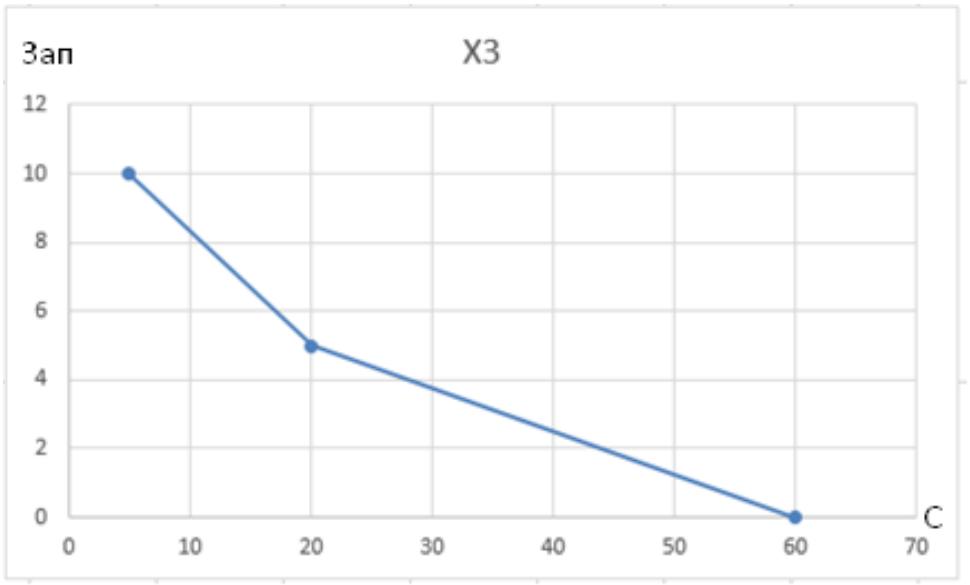

Рисунок 5.4 – X3, час виконання запитів користувача

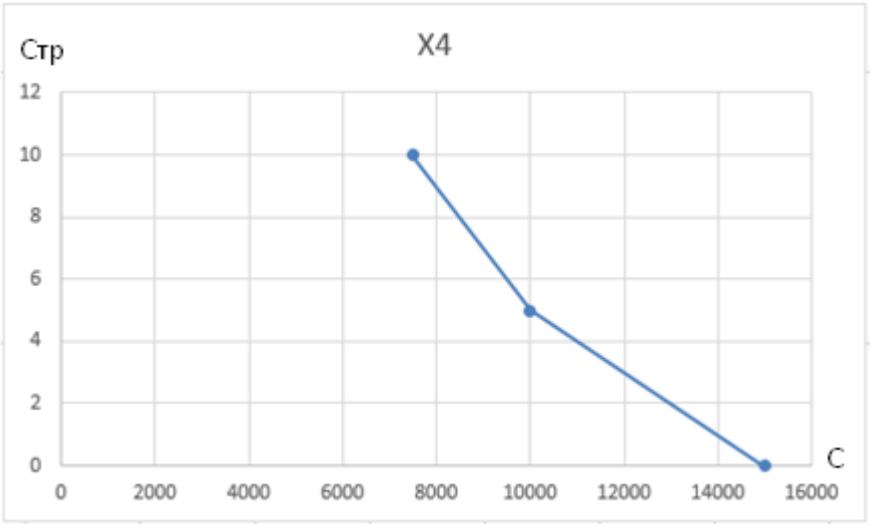

Рисунок 5.5 – X4, потенційний об’єм програмного коду

5.4.3 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 5 людей. Визначення коефіцієнтів значимості передбачає:

– визначення рівня значимості параметра шляхом присвоєння різних рангів;

– перевірку придатності експертних оцінок для подальшого використання;

– визначення оцінки попарного пріоритету параметрів;

– обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 5.3.

Таблиця 5.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта					Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5			
X1	Швидкодія мови програмування	Оп/мс	4	3	4	4	4	19	1,50	2,25
X2	Об’єм пам’яті для збереження даних	Мб	4	4	4	3	4	19	1,50	2,25
X3	Час обробки запитів користувача	Мс	2	2	1	2	1	8	9,50	90,25
X4	Потенційний об’єм програмного коду	кількість строк коду	4	5	5	5	5	24	6,50	42,25
	Разом		14	14	14	14	14	70	0	420,75

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70,$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17,5.$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 137.$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 137}{5^2(4^3 - 4)} = 1{,}096 > W_k = 0{,}67$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 5.4

Таблиця 5.4 – Попарне порівняння параметрів

Параметри	Експерти					Кінцева оцінка	Числове значення
	1	2	3	4	5		
X1 і X2	=	>	=	<	=	<	0,5
X1 і X3	<	<	<	<	<	<	0,5
X1 і X4	>	>	>	>	>	>	1,5
X2 і X3	<	<	<	<	<	<	0,5
X2 і X4	>	>	>	>	>	>	1,5
X3 і X4	>	>	>	>	>	>	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1{,}5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю $A = \| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \text{де } b_i = \sum_{i=1}^{N} a_{ij}.$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}$,де $b_i' = \sum_{i=1}^{N} a_{ij} b_j$.

Як видно з таблиці 5.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 5.5 – Розрахунок вагомості параметрів

Параметриx_i	Параметриx_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1,0	0,5	0,5	1,5	3,5	0,219	22,25	0,216	100	0,215
X2	1,5	1,0	0,5	1,5	4,5	0,281	27,25	0,282	124,25	0,283
X3	1,5	1,5	1,0	1,5	5,5	0,344	34,25	0,347	156	0,348
X4	0,5	0,5	0,5	1,0	2,5	0,156	14,25	0,155	64,75	0,154
Всього:					16	1	98	1	445	1

5.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2*(об’єм пам’яті для збереження даних) та *X1* (швидкодія мови програмування) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X3 (*час обробки даних) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 60 с або варіанту в) 40 с.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 5.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j} ,$$

де n – кількість параметрів;$K_{вi}$– коефіцієнт вагомості i–го параметра;B_i – оцінка i–го параметра в балах.

Таблиця 5.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1(X1)	А	10000	3,6	0,215	0,774
F2(X2)	А	4096	3,4	0,283	0,962
F3(X3,X4)	А	60	2,4	0,348	0,835
	Б	40	1	0,154	0,154

За даними з таблиці 5.6 за формулою

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

K_{K1} = 0,779 + 0,962 + 0,835 = 2,576

K_{K2} = 0,779 + 0,962 + 0,154 = 1,895

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

5.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється як

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М},$$

де T_P – трудомісткість розробки ПП;$K_П$ – поправочний коефіцієнт; $K_{СК}$ – коефіцієнт на складність вхідної інформації; K_M – коефіцієнт рівня мови програмування; $K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм; $K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P =95 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ}$ = 0.8. Тоді, за формулою 3.1, загальна трудомісткість програмування першого завдання дорівнює:

T_1 = 95*1.7*0.8 = 129.2 людино-днів.

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто T_P =30 людино-днів, $K_П$ =0.9,$K_{СК}$ = 1,$K_{СТ}$ =0.8:

T_2 = 30 * 0.9 *0.8 = 21,6 людино-днів.

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

T_I = (129.2 + 21.6 + 4.8 + 21.6) · 8 = 1417.6 людино-годин;

T_{II} = (129.2 + 21.6 + 6.91 + 21.6) · 8 = 1434.48 людино-годин;

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 18500 грн. Визначимо зарплату за годину за формулою:

$$С_ч = \frac{М}{T_m \cdot t} \text{ грн.},$$

де М – місячний оклад працівників;T_m – кількість робочих днів тиждень;t – кількість робочих годин в день.

$$С_ч = \frac{18500 + 18500}{3 \cdot 23 \cdot 8} = 67{,}03 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою

$$С_{ЗП} = С_ч \cdot T_i \cdot К_Д,$$

де $С_ч$– величина погодинної оплати праці програміста;T_i – трудомісткість відповідного завдання; $К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП}$ = 67,03 * 1417,6 * 1,2 = 114026,07 грн.

II. $С_{ЗП}$ = 67,03 * 1434,48 * 1,2 =115383,83 грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0{,}22$ = 114026,07 *0,22 = 25085,74 грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0{,}22$ = 115383,83 *0,22 = 25384,44 грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Оскільки одна ЕОМ обслуговує одного програміста з окладом 18500 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 18500 \cdot 0{,}2 = 44400$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 44400 \cdot (1 + 0,2) = 53280$ грн.

Відрахування на єдиний соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0,2 = 53280 * 0,22 = 11721,6$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 30000 грн.

$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 30000 = 8625$ грн.,

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_А$– річна норма амортизації; $Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 30000 \cdot 0.04 = 1380$ грн.,

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 8 – 12) \cdot 8 \cdot 0.9 = 1735.2$ годин,

де $Д_К$ – календарна кількість днів у році; $Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів; $Д_Р$ – кількість днів планових ремонтів устаткування; t –кількість робочих годин в день; $К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1735,2 \cdot 0,162 \cdot 1,385 \cdot 2,0218 = 787,14$ грн.,

де $N_С$ – середньо-споживча потужність приладу; $К_З$– коефіцієнтом зайнятості приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$С_Н = Ц_{ПР} \cdot 0.67 = 30000 \cdot 0,67 = 20100$ грн.

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н$$

$С_{ЕКС} = 53280 + 11721,6 + 8625 + 1380 + 787,14 + 20100 = 95893,74$ грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М-Г} = С_{ЕКС} / Т_{ЕФ} = 95893,74 / 1735,2 = 55,26$грн/год.

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot Т$$

I. $С_М = 55,26 * 1417,6 = 78336,58$ грн.;

II. $С_М = 55,26 * 1434,48 = 79269,36$грн.;

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67$$

I. $С_Н = 114026,07 * 0,67 = 76397,42$ грн.;

II. $С_Н = 115383,83 * 0,67 = 77306,61$ грн.;

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н$$

I. $С_{ПП} = 114026,07 + 25085,74 + 78336,58 + 76397,42 = 293845,81$грн.;

II. $С_{ПП} = 115383,83 + 25384,44 + 79269,36 + 77306,61 = 297344,24$грн.;

5.7 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕРj} = К_{Кj} / С_{Фj},$$

$К_{ТЕР1} = 2,576 / 293845,81 = 0,877 * 10^{-5}$;

$К_{ТЕР2} = 1,895 / 297344,24 = 0,637 * 10^{-5}$;

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 0,877 \cdot 10^{-5}$.

5.8 Висновки

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП. Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 0{,}982 \cdot 10^{-5}$. Цей варіант реалізації програмного продукту має такі параметри:

– мова програмування – Java;

– СКБД Noe4j;

– інтерфейс користувача, створений за технологією Java FX.

Даний варіант виконання програмного комплексу дає зручний користувацький інтерфейс, найкращі показники надійності програмного продукту, функціонал, що задовольняє заданим вимогам та непогану швидкодію.

ВИСНОВКИ

Зародившись при розв'язуванні головоломок і цікавих задач, теорія графів нині стала потужним засобом розв'язування задач широкого спектру проблем. В деякій мірі через теорію графів відбувається проникнення математичних методів в науку і техніку. Тим часом поняття графа використовується дуже часто не тільки в математиці, але і повсякденному житті під різними назвами: схема, діаграма, карта, лабіринт тощо. Теорія графів на даний час бурхливо розвивається, її результати застосовують, проектуючи різноманітні електронні пристрої, вивчаючи автомати, у програмуванні, фізиці, хімії, біології, економіці, статистиці та багатьох інших галузях діяльності людини.

Зважаючи на це, в даній роботі проведений короткий огляд трьох графових БД с метою дослідження їх основних можливостей та тестуванню їх для вибору оптимальної графової БД.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що варіант реалізації програмного продукту з поєднанням Java та Neo4j дає найкращі показники надійності програмного продукту, функціонал, що задовольняє заданим вимогам та непогану швидкодію.

Також було показано, як графи є відмінним вибором для прагматичного моделювання даних. Була показана архітектура графової бази даних, з особливим акцентом на архітектурі Neo4j, і також показані різні характеристики реалізацій графової бази даних. Було виявлено, після аналізу графової БД порівняно з іншими класичними підходами, що час пошуку в Neo4j на великій кількості даних кращий, ніж у MySQL.

Після досліджень виявлено, графові бази даних будуть дуже корисними, як вказувалося раніше, коли потрібно переходити та досліджувати відносини між елементами. Графи дуже гнучкими, коли час від часу проходить зміна

взаємозв’язків між даними, або коли потрібно змінити характеристики певних елементів. Але графи дуже сильно залежать від предметної області, на якій проектуються. Використання теорії графів в аналізі соціальних даних дуже корисне, в результаті якого можна отримати будь-які потрібні характеристики. Дані можна були представити у різних структурах, але вибір саме графової бази даних Neo4j набагато спрощує складність алгоритмів саме із-за своєї архітектури.

Також виявлено, що після проектувань картографічних систем ГІС на основі Neo4j та впровадження такого продукту на стадію планування доріг прибудинкових територій, можна буде уникнути витоптування газонів та спотворення зеленої території біля будинків.

А на прикладі роботи організації ICIJ було показна корисність використання графової БД Neo4j. Це свідчить про те, що використання графових баз даних полегшують роботу з соціальною інформацією різноманітних напрямків, навіть з приватними даними панамських документів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Глибовець М. М. Інтелектуальні мережі : навч. посіб. / М. М. Глибовець, А. М. Глибовець, М. В. Поляков. – Дніпропетровськ : Нова ідеологія, 2014. – 464 с.
2. Причины и предусловия применения нереляционных баз данных – Режим доступу: http://www.rusnauka.com/4_SND_2012/Informatica/4_99281.doc.htm. - Дата доступу: 10.05.2016
3. Круш І.В. Використання графових баз даних для оптимізація швидкості обробки складних запитів // Круш І.В.// Системний аналіз та інформаційні технології: матеріали 17-ї Міжнародної науково-технічної конференції SAIT 2015, Київ, 22-25 червня 2015 р. / ННК “ІПСА” НТУУ “КПІ”. – К.: ННК “ІПСА” НТУУ “КПІ”, 2015. – 304 с.
4. Історія виникнення і розвиток теорії графів – Режим доступу: http://www.studall.org/all3-46561.html - Дата доступу: 15.05.2016
5. Основні поняття теорії графів – Режим доступу: http://www.oim.asu.kpi.ua/files/DM/24_Graphs.pdf - Дата доступу: 16.05.2016
6. Основи теорії графів - Основні поняття теорії графів – Режим доступу: http://www.elib.lutsk-ntu.com.ua/book/fbd/mbg/2012/12-33/page9.html - Дата доступу: 20.05.2016
7. Бартенев М.В., Вишняков И.Э. Использование графовых баз данных в целях оптимизации анализа биллинговой информации. Инженерный журнал: наука и инновации,2013, вып.11. – Режим доступу http://www.engjournal.ru/catalog/it/hidden/1058.html - Дата доступу: 21.05.2016
8. Robinson, I. and Webber, J. and Eifrem, E.Graph Databases. – Режим доступу http://www.info.neotechnology.com/rs/neotechnology/images/GraphDatabases.pdf - Дата доступу: 22.05.2016

9. Порівняння Neo4j і реляційної бази даних MySQL. – Режим доступу http://www.ekmair.ukma.edu.ua/bitstream/handle/123456789/8938/Hlybovets_Porivniannia_Neo4.pdf - Дата доступу: 25.05.2016
10. ARIADNE: a Tracking System for Relationships in LHCb Metadata – Режим доступу http://iopscience.iop.org/article/10.1088/1742-6596/513/4/042039/pdf - Дата доступу: 27.05.2016
11. Соціальний граф – Режим доступу http://atseajournal.com/asoiaf/resources/asoiaf_exchanges.gephi - Дата доступу: 30.05.2016
12. Алгоритм Ant Road Planner – Режим доступу https://antroadplanner.wordpress.com/ - Дата доступу: 30.05.2016
13. How the ICIJ Used Neo4j to Unravel the Panama Papers – Режим доступу http://www.neo4j.com/blog/icij-neo4j-unravel-panama-papers/ - Дата доступу: 10.05.2016