МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 36.00.00.000 ПЗ

Група ІПм-23-1

Галярник Ростислав

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Галярник Ростислав Романович

(прізвище, ім'я, по батькові)

УДК 004.942

(індекс)

МАГІСТЕРСЬКА РОБОТА

Моделі та методи підвищення ефективності взаємодії клієнтів з базами даних

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Галярник Р.Р.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Мельник Віталій Дмирович, к.т.н., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль

доц. Вовк Р.Б.

(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітній рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ 04 ” вересня 2024 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Галярнику Ростиславу Романовичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “ Моделі та методи підвищення ефективності взаємодії клієнтів з базами даних”

керівник проекту (роботи) Мельник Віталій Дмитрович, к.т.н., доцент

затверджені наказом закладу вищої освіти від “ 22 ” листопада 2024 р. № 781/7

2. Строк подання студентом проекту (роботи) 15 грудня 2024 р.

3. Вихідні дані до проекту (роботи) Теоретичні концепції та формальні моделі побудови та функціонування інформаційних та програмних технологій роботи з базами даних

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

1. Особливості процесів візуалізації при роботі з графовими базами даних
2. Моделі та алгоритми побудови візуалізацій для роботи з базами даних
3. Методологія візуалізації вкладеності SQL запитів
4. Імплементація моделей та методів для підвищення ефективності взаємодії клієнтів з БД

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Ріст даних в світі (2010 - 2025) (рис. 1.1)
2. Різниця між структурою графових і реляційних БД (рис. 1.2)
3. Приклад графа властивостей бази даних (рис. 1.3)
4. Часткове представлення зв’язку вузла бази даних графа (рис. 1.4)
5. Схематичне представлення структури NoSQL баз даних (рис. 1.5)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Перевірка на плагіат	доц., к.т.н. Вовк Р.Б.	

7. Дата видачі завдання 04 вересня 2024 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури по темі магістерської роботи	15.09.2024	виконано
2	Аналіз концепцій та алгоритмів предметної області	29.09.2024	виконано
3	Особливості процесів візуалізації при роботі з графовими базами даних	15.10.2024	виконано
4	Моделі та алгоритми побудови візуалізацій для роботи з базами даних	08.11.2024	виконано
5	Методологія візуалізації вкладеності SQL запитів	20.11.2024	виконано
6	Імплементація моделей та методів для підвищення ефективності взаємодії клієнтів з БД	01.12.2024	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 76 с., 33 рис., 51 джерело.

Тема: Моделі та методи підвищення ефективності взаємодії клієнтів з базами даних

Об'єкт дослідження: процеси візуалізації та взаємодії користувачів із базами даних.

Мета роботи: розробка та імплементація моделей і методів для підвищення ефективності візуалізації та взаємодії користувачів з базами даних.

Предмет дослідження: методи та алгоритми візуалізації даних і інтерфейсні рішення для взаємодії користувачів з базами даних.

Результати дослідження

В роботі запропоновано методологію візуалізації вкладених SQL-запитів та інтерактивний інтерфейс, орієнтований на потреби користувача, що значно покращує доступність і швидкість роботи з базами даними для різних категорій користувачів.

Висновок

Запропоновані рішення можуть застосовуватися для навчальних, аналітичних та бізнес-систем, де важливий простий і доступний інтерфейс взаємодії з базами даних.

ГРАФОВІ БАЗИ ДАНИХ, ВІЗУАЛІЗАЦІЯ ДАНИХ, МОДЕЛІ ДАНИХ, АЛГОРИТМИ ВІЗУАЛІЗАЦІЇ, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС, SQL-ЗАПИТИ, ІНТЕРАКТИВНА ВІЗУАЛІЗАЦІЯ, ЕФЕКТИВНІСТЬ ВЗАЄМОДІЇ.

ABSTRACT

Master Thesis: 76 pp., 33 fig., 51 sources.

Thesis Subject: Models and methods of improving the efficiency of customer interaction with databases

Research object: visualization processes and user interaction with databases.

The purpose of the work: development and implementation of models and methods to improve the efficiency of visualization and user interaction with databases.

Research subject: data visualization methods and algorithms and interface solutions for user interaction with databases.

Research results

The work offers a methodology for visualizing nested SQL queries and an interactive interface focused on user needs, which significantly increases the accessibility and speed of working with databases for different categories of users.

Conclusion

The proposed solutions can be rejected for educational, analytical and business systems, where a simple and accessible interface for interacting with databases is important.

GRAPH DATABASES, DATA VISUALIZATION, DATA MODELS, VISUALIZATION ALGORITHMS, USER INTERFACE, SQL QUERIES, INTERACTIVE VISUALIZATION, INTERACTION EFFECTIVENESS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 9
ВСТУП 10

РОЗДІЛ 1. ОСОБЛИВОСТІ ПРОЦЕСІВ ВІЗУАЛІЗАЦІЇ ПРИ РОБОТІ З ГРАФОВИМИ БАЗАМИ ДАНИХ 13
1.1. Сучасні дані та бази даних як засіб їх обробки 13
1.2. Графи та графічна інтерпретація даних 16
1.3. Історія створення та особливості графових баз даних 18
1.3.1. Об'єктно-орієнтовані бази даних і XML 19
1.3.2. NoSQL і GraphDB 20
1.3.3. RDF і SPARQL 21
1.3.4. Мови графових запитів 22
Висновки до розділу 23

РОЗДІЛ 2. МОДЕЛІ ТА АЛГОРИТМИ ПОБУДОВИ ВІЗУАЛІЗАЦІЙ ДЛЯ РОБОТИ З БАЗАМИ ДАНИХ 24
2.1. Проблеми та особливості візуалізації графових даних 24
2.1.1. Дизайн візуалізації 24
2.1.2. Проблема масштабування 25
2.1.3. Методи представлення даних 26
2.2. Шаблони побудови візуальних систем запитів 29
2.3. Методи та інструменти візуалізації запитів до бази даних 32
2.3.1. Візуальні мови запитів 33
2.3.2. Підсвічування синтаксису 36
2.3.3. Переклади запитів природною мовою 37
2.4. Методологія візуалізації вкладеності SQL запитів 39
2.4.1 Дизайн та ефективність візуалізації 42
2.4.2. Мінімальність візуалізації 46

2.4.3. Перетворення SQL в діаграми 47
2.4.4. Властивості діаграми 49
Висновки до розділу 51

РОЗДІЛ 3. ІМПЛЕМЕНТАЦІЯ МОДЕЛЕЙ ТА МЕТОДІВ ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ВЗАЄМОДІЇ КЛІЄНТІВ З БАЗАМИ ДАНИХ 52
3.1. Представлення моделі даних 52
3.2. Представлення основних вимог до системи взаємодії з базами даних . 56
3.2.1. Основні вимоги 58
3.2.2. Опис випадку використання 59
3.3. Дизайн, орієнтований на користувача 60
3.4. Розробка прототипу інтерфейсу системи ефективної взаємодії користувачів з базами даних 62
Висновки до розділу 69

ВИСНОВКИ 71
ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 73

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

CDB – Cloud Database

VQS - Visual Query Systems

VGQS - Visual Graph Query Systems

ETL – Extract, Transform, Load

OLTP – Online Transaction Processing

EER – Enhanced Entity-Relationship

DDL – Data Definition Language

DML – Data Manipulation Language

DBMS – Database Management System

B-Tree – Balanced Tree (data structure used in indexing)

FSM – Finite State Machine

VRT – Virtual Record Table

CFI – Control Flow Integrity

ERP – Enterprise Resource Planning

SLA – Service Level Agreement

RAID – Redundant Array of Independent Disks

TPC – Transaction Processing Performance Council

COTS – Commercial Off-The-Shelf

ВСТУП

Актуальність теми.

В умовах швидкого зростання обсягів даних у сучасному інформаційному суспільстві графові бази даних набувають все більшого значення. Вони забезпечують ефективну роботу з взаємопов'язаними даними, які зустрічаються в соціальних мережах, рекомендаційних системах, обробці природної мови, біоінформатиці, інтернеті речей (IoT) та багатьох інших сферах. Основною перевагою графових баз даних є їх здатність зберігати і обробляти складні зв’язки між даними, що значно перевищує можливості реляційних баз. Це зумовлює потребу у вдосконалених методах візуалізації, які дозволяють користувачам ефективно аналізувати та інтерпретувати ці зв’язки.

Зростання обсягів графових даних ставить нові виклики для розробників та користувачів графових баз, серед яких виділяються проблеми масштабованості, зручності доступу та візуалізації великих і складних графових структур. Більшість існуючих інструментів візуалізації недостатньо оптимізовані для роботи з великими графами та складними запитами, що знижує ефективність аналізу та обмежує можливості користувачів у дослідженні великих обсягів даних.

Також, важливим аспектом є зручність інтерфейсу взаємодії, оскільки користувачі різного рівня підготовки мають потребу в інтуїтивно зрозумілих і простих засобах доступу до даних. Відсутність належної підтримки з боку інтерфейсу може створювати перешкоди для прийняття рішень, що особливо критично в умовах роботи з бізнес-аналітикою, науковими дослідженнями, а також в інших галузях, де швидкість і точність аналізу даних є пріоритетом.

З огляду на це, актуальним є дослідження моделей, алгоритмів і методів візуалізації графових даних, а також розробка інтуїтивних користувацьких інтерфейсів, що підвищують зручність і швидкість взаємодії з базами даних. Запропоновані в роботі рішення спрямовані на розширення

можливостей користувачів у сфері графових баз даних, що може суттєво підвищити якість аналізу, інтерпретації та управління великими масивами взаємопов'язаних даних.

Мета дослідження - розробка та імплементація моделей і методів для підвищення ефективності візуалізації та взаємодії користувачів з базами даних.

Об'єкт дослідження - процеси візуалізації та взаємодії користувачів із базами даних.

Предмет дослідження - методи та алгоритми візуалізації даних і інтерфейсні рішення для взаємодії користувачів з базами даних.

Задачі дослідження

- Проаналізувати особливості сучасних баз даних, зокрема графових, їх структуру та механізми обробки даних.

- Визначити проблеми та вимоги до ефективної візуалізації графових даних.

- Розробити моделі та алгоритми для візуалізації запитів у базах даних.

- Сформулювати вимоги до користувацького інтерфейсу, що підвищують ефективність взаємодії з графовими базами даних.

- Розробити прототип інтерфейсу для візуалізації графових даних і перевірити його ефективність.

Методи дослідження

Аналіз і систематизація літературних джерел, моделювання структур даних, розробка алгоритмів візуалізації, методи юзабіліті-тестування прототипів інтерфейсів, емпіричний аналіз та експериментальна перевірка ефективності запропонованих моделей.

Наукова новизна отриманих результатів

Наукова новизна полягає в розробці та впровадженні нових моделей і алгоритмів для ефективної візуалізації запитів до баз даних. Запропоновано методологію візуалізації вкладених SQL-запитів та інтерактивний інтерфейс,

орієнтований на потреби користувача, що значно покращує доступність і швидкість роботи з графовими даними для різних категорій користувачів.

Практичне значення результатів

Розроблені моделі та прототип інтерфейсу можуть бути використані для створення інформаційних систем, що потребують високої швидкості доступу до бази даних і простоти візуалізації. Запропоновані рішення можуть застосовуватися для навчальних, аналітичних та бізнес-систем, де важливий простий і доступний інтерфейс взаємодії з базами даних.

Структура магістерської роботи. Робота складається зі вступу, трьох розділів та висновків. Загальний обсяг роботи становить 76 сторінок, і містить 33 рисунки, список використаних джерел із 51 найменування.

РОЗДІЛ 1. ОСОБЛИВОСТІ ПРОЦЕСІВ ВІЗУАЛІЗАЦІЇ ПРИ РОБОТІ З ГРАФОВИМИ БАЗАМИ ДАНИХ

1.1. Сучасні дані та бази даних як засіб їх обробки

У той час як охоплення Інтернету швидко розширюється, кількість даних, які створюються та зберігаються в ньому, зростає експоненціально. За даними IDC, щорічний обсяг цифрових даних зросте до 175ZettaByte у 2025 році. Ці дані генеруються всіма членами суспільства за допомогою різноманітних пристроїв і програм, включаючи розумні термостати та автомобілі. Програмне забезпечення, яке керує цими елементами, генерує дані. Аналіз, який можна виконати на основі цих даних, надає виробникам продуктів багато інформації. У більш особистому масштабі люди також створюють дані за допомогою додатків для відстеження споживання їжі та калорій, руху та місцезнаходження, а також відстеження того, наскільки добре вони сплять. Багато людей шукають способи правильно зрозуміти та інтерпретувати свої дані.

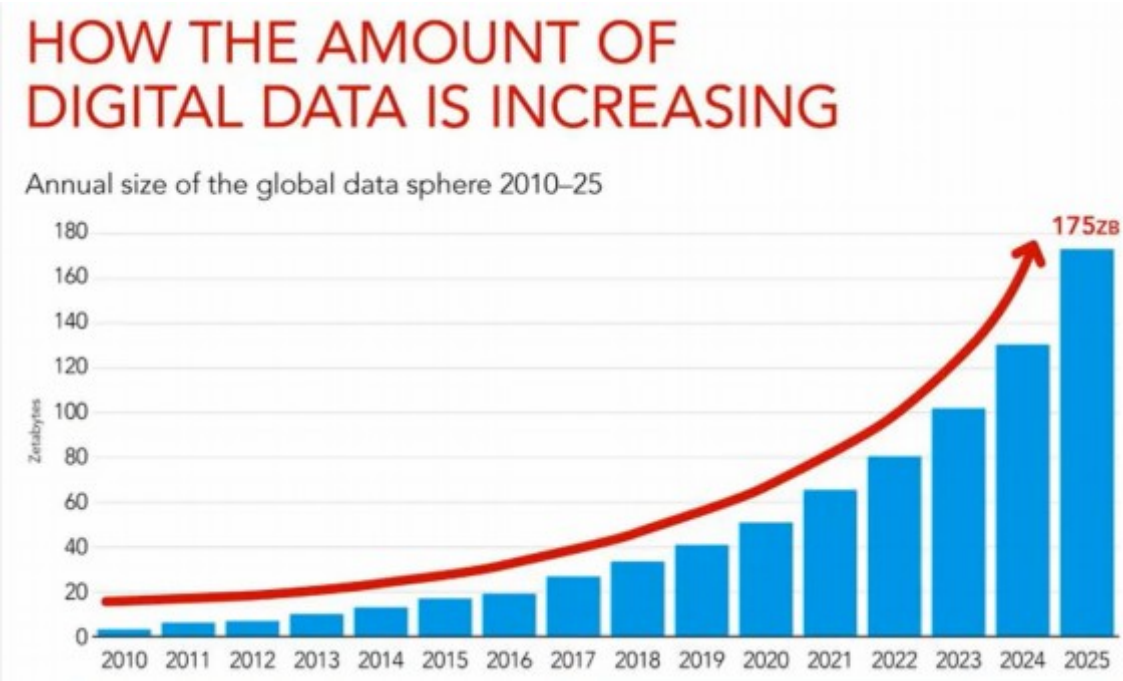

Рис. 1.1. Ріст даних в світі (2010 - 2025)

Великі набори даних найефективніше зберігаються в базах даних. Вони дозволяють розширену взаємодію з даними, наприклад: доступ до даних для постановки запитань (також відомих як запити), додавання нових даних, зміна структури даних, умовне застосування змін до даних тощо. Бази даних зазвичай використовують певний тип мови програмування, який називається мовою запитів, для виконання цих взаємодій. На жаль, це означає, що для користування базою даних користувач повинен добре володіти пов’язаною з нею мовою. Це просто неможливо для користувачів, які шукають простий спосіб взаємодії зі своїми даними.

Крім значного обсягу навчання, необхідного для ефективного використання баз даних, запити також вимагають значних когнітивних зусиль з боку користувача. В [2] зазнача.nm, що таке високе когнітивне навантаження може виникати як через тягар запам'ятовування схеми бази даних, так і через те, що система керування базою даних перехоплює лише певні типи помилок. Це, серед інших аспектів, сприяє неоптимальному використанню баз даних і перешкоджає ненавченим користувачам (початківцям) аналізувати їхні дані. Ми прагнемо полегшити ці проблеми зручності використання.

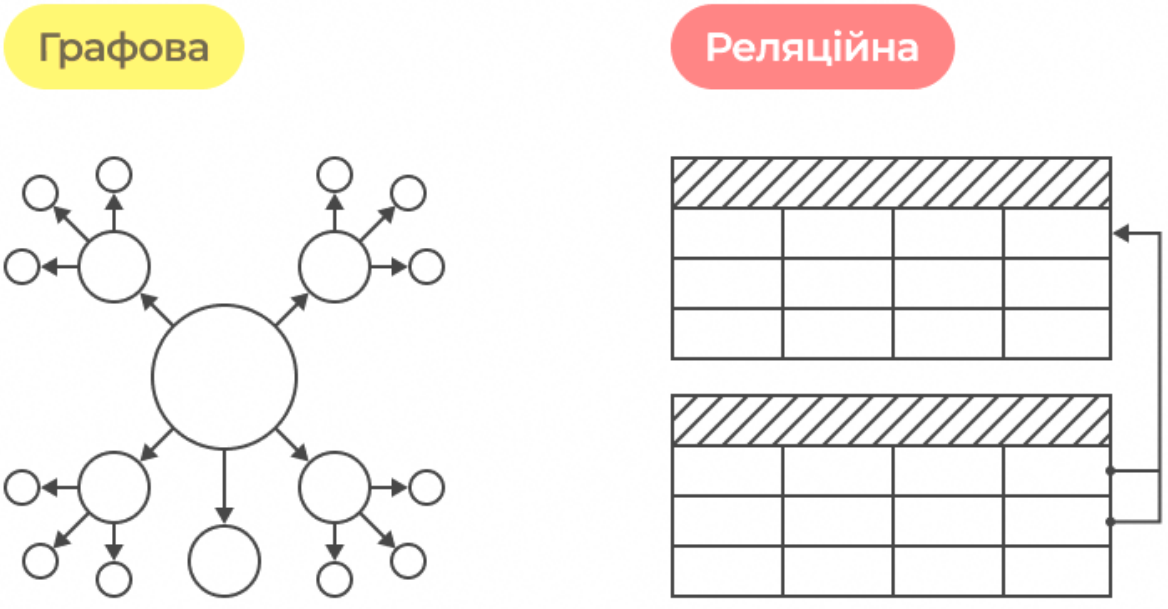

Рис. 1.2. Різниця між структурою графових і реляційних БД

Останні розробки в спільноті дослідників баз даних зосереджені на графових базах даних. Замість того, щоб мати на увазі зв'язки між записами даних, ці бази даних графів явно зберігають усі існуючі зв'язки в даних. Це дозволяє візуалізувати дані графіка більш прямолінійним способом, оскільки ці дані зі зв'язками можна намалювати як графік. Більш чіткий характер даних також полегшує розуміння, оскільки концептуально пов'язані дані також пов'язані фізично. Мови запитів графів для цих даних також більш візуальні за своєю природою, що забезпечує більш інтуїтивно зрозумілу взаємодію.

З точки зору зручності використання бази даних також бракує досліджень, які б зосереджувалися на цій темі. Ми знайшли лише кількох дослідників, які працювали над аналізом зручності використання баз даних. Як правило, системи баз даних розробляються інженерами баз даних без урахування досліджень взаємодії людини та комп'ютера. Дослідники в цих галузях працюють незалежно, не використовуючи принципи один одного для вдосконалення своєї роботи. Це означає, що є системи, які адаптуються до навичок користувача, але не мають продуктивності, а також високопродуктивні системи, яким бракує дизайну взаємодії з користувачем. Будь-які доповнення до досліджень зручності використання бази даних і, отже, співпраці між двома галузями призведуть до покращення взаємодії, оскільки жодна галузь не залишиться позаду.

Відсутність співпраці також проявляється в кількості опублікованих досліджень на тему взаємодії баз даних. Деякі роботи існують, але вони далеко не нещодавні. Це означає, що вони беруть до уваги лише мови запитів того періоду. За роки, що минули після публікації цих статей, було розроблено багато нових мов запитів. Нове дослідження на цю тему може показати, чи працюють старі принципи для нових мов, і може детально розкрити взаємодію з новими типами функцій, такими як бази даних графів.
У цій дипломній роботі ми зробимо внесок у дослідження взаємодії людини з базою даних. Ми переглядаємо літературу за останні десятиліття, проводимо

експерименти, щоб порівняти теорію з практикою, і розробляємо інструмент бази даних через дизайн, орієнтований на користувача, який має полегшити взаємодію з користувачами-початківцями.

1.2. Графи та графічна інтерпретація даних

Багато наукових дисциплін мають власне визначення графа. Найчастіше, коли говорять про граф, на думку спадає двовимірна фігура, що представляє точки даних на площині. Такі графіки також називаються лінійними діаграмами і можуть бути створені як для наборів точок, так і для неперервних функцій.

Зовсім іншим є визначення графа з дискретної математики, де граф зображує набір об'єктів, які можуть бути пов'язані між собою. Як об'єкти, так і зв'язки можуть бути відображені абстрактним чином, де об'єкти малюються як кола, які називаються вузлами або вершинами, а зв'язки зображуються як лінії між ними, які також називаються ребрами.

Крім того, існує також визначення графа з області інформатики. У цьому випадку граф - це абстрактний тип даних, який використовується для зберігання набору об'єктів з математичного графа.

Дані графа названі на честь дискретно-математичної області теорії графів і тому здебільшого відносяться до середнього визначення. Термін "дані графа" конкретно стосується самих даних, наборів об'єктів та зв'язків, які разом утворюють базу даних. У даних графа немає нічого візуального, хоча візуалізація природно випливає з абстракції об'єктів.

Щоб уточнити різницю між реляційними базами даних та графовими базами даних, на рисунку 1.3 показано приклад бази даних графа властивостей. Типова база даних графа не обов'язково містить таблиці властивостей вузлів та ребер, а містить лише самі зв'язки. Ці дані графа властивостей також можна легко перетворити на візуальну інтерпретацію,

яка називається "вузол-зв'язок". Часткове представлення "вузол-зв'язок" бази даних графа з рисунку 1.3 показано на рисунку 1.4.

node properties

NID	property	value
1	name	Amsterdam
1	population	853312
2	name	Utrecht
3	name	den Bosch
4	label	city
4	name	Leiden
5	name	den Haag
6	name	Breda
7	name	Eindhoven
7	population	227100
8	name	Beijing
9	name	Aalst

edge properties

EID	property	value
9	name	A2
10	name	A4
11	name	A2
12	name	A2
12	type	highway
13	name	A12
13	type	highway
14	name	A27
15	name	A2
16	name	N69
16	type	nat. road
17	name	A4
18	name	A4
19	name	A12
19	type	highway
20	name	A27
21	since	1994

connects

SNID	EID	TNID
1	9	2
1	10	4
1	11	7
2	12	7
2	13	5
2	14	6
3	15	2
7	16	9
4	17	5
5	18	4
5	19	2
6	20	2

sisterCityOf

SNID	EID	TNID
8	21	1

Рис. 1.3. Приклад графа властивостей бази даних

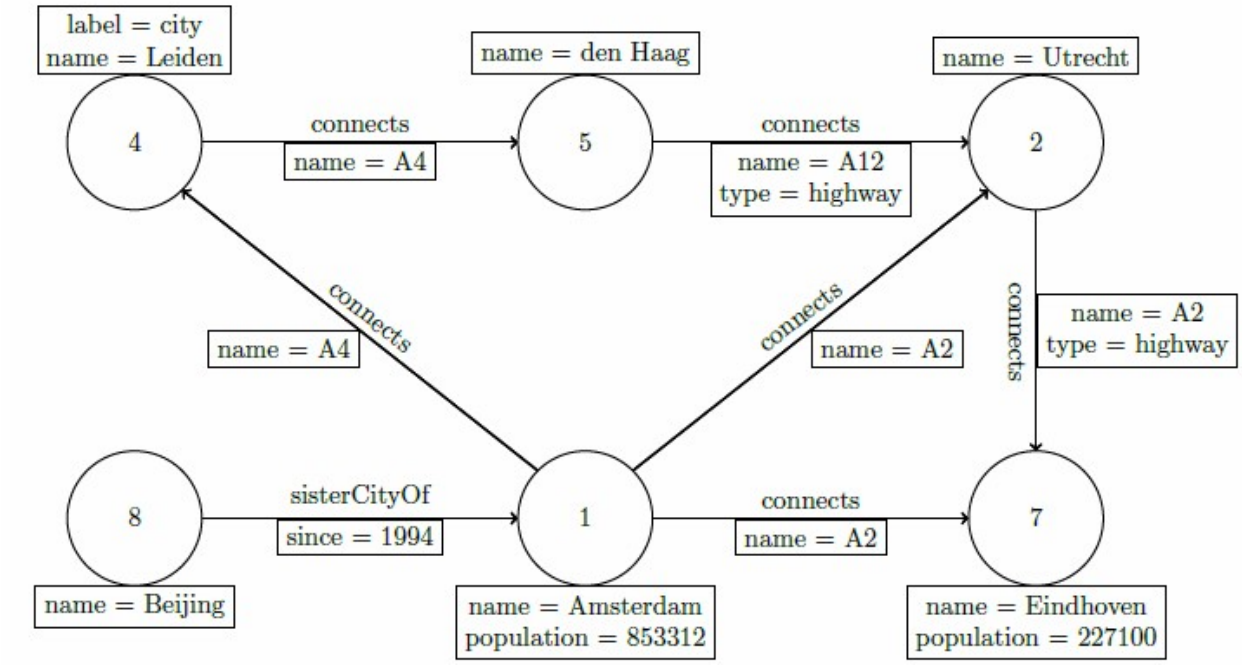

Рис. 1.4. Часткове представлення зв’язку вузла бази даних графа з рисунка 1.3

Частина властивостей міститься в прямокутниках з властивостями. Один елемент, відсутній у представленні "вузол-зв'язок", який присутній у таблиці, - це ідентифікатори ребер. Вони є неявними і недоступні для користувачів або в інтерфейсах. Ідентифікатори ребер - це внутрішня механіка системи баз даних на випадок, якщо два вузли з'єднані двома різними ребрами. Наприклад, якби ми додали маршрути польотів до бази даних, могло б бути додаткове ребро з Пекіна до Амстердама. Ці ребра потім розрізняються за допомогою своїх ідентифікаторів.

1.3. Історія створення та особливості графових баз даних

Щоб надати деякий контекст для того, як ми прийшли до поточного стану техніки в базах даних, у цьому розділі ми опишемо історію графових баз даних. Ми починаємо з основи, старих технік і впровадження реляційної моделі. Через об’єктно-орієнтовані бази даних, рух NoSQL і RDF ми дійдемо до сучасних розробок графових баз даних і мов графових запитів.

Концепція систем баз даних виникла у відповідь на перші методи комп’ютеризованого керування комерційними даними. Старі методи керування даними часто зберігали дані у файлових структурах операційної системи. Ці системи використовували спеціальні прикладні програми для маніпулювання файлами різними способами, наприклад для додавання запису. Крім того, що це складне завдання, було багато проблем із зберіганням таких даних. Проблеми включали надмірність даних, цілісність, одночасний доступ і безпеку даних.

Основна мета системи баз даних — надати користувачам абстрактне уявлення про дані. Це означає, що система приховує певні деталі того, як дані зберігаються та обслуговуються. Програмісти та адміністратори баз даних мають доступ до логічного рівня абстракції, користувач бази даних бачить рівень перегляду. Першим кроком в абстракції систем баз даних була реляційна модель, визначена Коддом в 1970 році. Він припустив, що

управління даними має бути незалежним від змін у типах даних і представленні даних. Хороша система баз даних використовує мову високого рівня, яка повинна забезпечувати максимальну незалежність між програмами та їх машинним представленням.

Хоча абстракція полегшила маніпуляції даними, реляційні бази даних не досягли продуктивності мережевих та ієрархічних баз даних, які раніше використовувалися. Ситуація змінилася з появою IBM System R. До початку 1980-х продуктивність реляційних баз даних стала конкурентоспроможною мережевим та ієрархічним системам баз даних.

1.3.1. Об'єктно-орієнтовані бази даних і XML

У 1980-х роках було багато досліджень паралельних і розподілених баз даних, а також початкова робота над об’єктно-орієнтованими базами даних [4]. Ця розробка була потрібна через проблеми з реляційною моделлю. Однією з перших проблем, з якою зіткнулися реляційні бази даних, було зростання кількості нетекстових даних, таких як зображення, звукові файли та відео. Цю проблему було вирішено шляхом введення таких характеристик, як складні об’єкти та розширюваність даних, що відповідає поняттю об’єктів в об’єктно-орієнтованих мовах програмування. Об'єктно-орієнтовані структури, такі як ієрархії, агрегації та покажчики, вводяться в моделі баз даних під назвою об'єктно-орієнтованих баз даних. Ці характеристики дозволяють об’єктно-орієнтованим базам даних бути першою моделлю бази даних із вбудованою підтримкою даних графів.

Зрештою, об’єктно-орієнтовані бази даних не набули особливого поширення, за винятком ринкових ніш, таких як географічний та інженерний сектори, де існувала потреба в графічних даних для зберігання.

У першій половині 2000-х років XML з'явився як нова технологія баз даних. XML можна розглядати як тип об'єктно-орієнтованої бази даних. Завдяки характеру XML, який дозволяє створювати та використовувати будь-які теги, він може підтримувати складні об’єкти. Тому бази даних XML

можуть представляти будь-яку довільну структуру даних. Крім того, XML має деревоподібну структуру, що дозволяє відносно легко представляти дані графіка. Однак, незважаючи на широкий спектр можливих застосувань XML, реляційні бази даних все ще залишалися основою більшості програм баз даних протягом досить тривалого часу.

1.3.2. NoSQL і GraphDB

Іншою подією, що відійшла від реляційних баз даних, був рух NoSQL. NoSQL намагається позначити появу все більшої кількості нереляційних розподілених сховищ даних. Проблема реляційних баз даних полягала в проектуванні символьних даних і моделюванні за допомогою стовпців і рядків у таблиці. Це погано масштабується, що є проблемою в епоху великих даних. NoSQL часто може запропонувати швидше рішення для меншої групи додатків замість підходу реляційної моделі, який базується на одній базі даних.

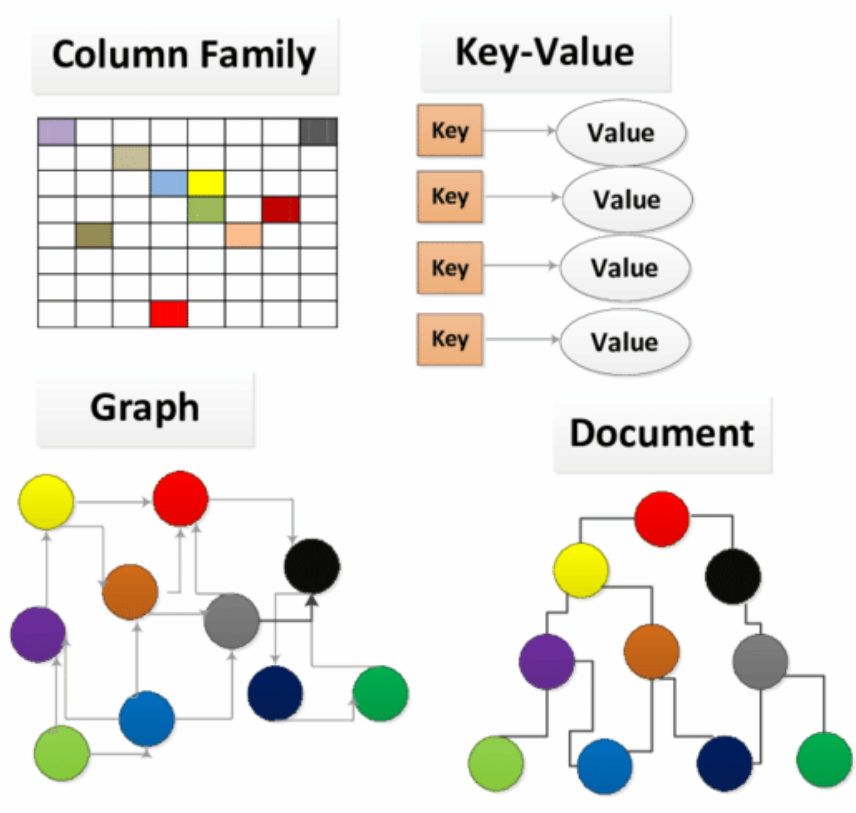

Рис. 1.5. Схематичне представлення структури NoSQL баз даних

Інтерес до NoSQL зріс з двох основних причин. Перш за все, традиційні мови запитів не працювали достатньо швидко для деяких нових технологій. По-друге, дані більше не тільки зберігаються, аналіз і пошук даних також є важливими. Для цього потрібна модель даних, яка могла б ефективно виконувати всі ці операції.

Існують різні типи баз даних NoSQL. Підкатегорія, яка найбільше стосується цього огляду, — це категорія Graph Data Models. Це був єдиний тип бази даних NoSQL, яка все ще застосовувала відносини. Він також зосереджений на візуальних представленнях, щоб базами даних було легше користуватися.

Рух NoSQL і його цілі стали поштовхом до розробки перших виділених графових баз даних. Графові бази даних дозволяють зберігати дані за допомогою прямого зв'язку, зберігаючи взаємозв'язок даних. У багатьох випадках структури також можна отримати за допомогою однієї операції. Однак основний механізм зберігання змінюється. Деякі графові бази даних залежать від реляційного механізму, тоді як інші використовують сховище ключ-значення або орієнтовану на документ базу даних.

1.3.3. RDF і SPARQL

За всі роки, що минули відтоді, як дані графіків були вперше збережені в базах даних, не було зроблено жодних зусиль щодо стандартизації. Коли W3C прийняв RDF як рекомендацію в 1999 році, його простота та здатність моделювати абстрактні поняття призвели до збільшення використання в управлінні даними. Модель даних триплетів RDF була створена як модель метаданих, але вона також представляє позначений різноспрямований граф. Кожен триплет може представляти ребро в поміченому графі. Тому вона добре підходить як стандартна база даних графів. Тому її можна назвати першою стандартизованою базою даних графів.

З тих пір для RDF було створено багато мов запитів. Деякі були засновані на SQL, інші на логіці та мовах правил. В [6] перевірили кілька з

цих мов на виразність графових запитів і виявили, що більшість з них не працюють добре. Лише мови G+ і GraphLog достатньо добре підтримують графові запити, щоб працювати як мови запитів на графіки.

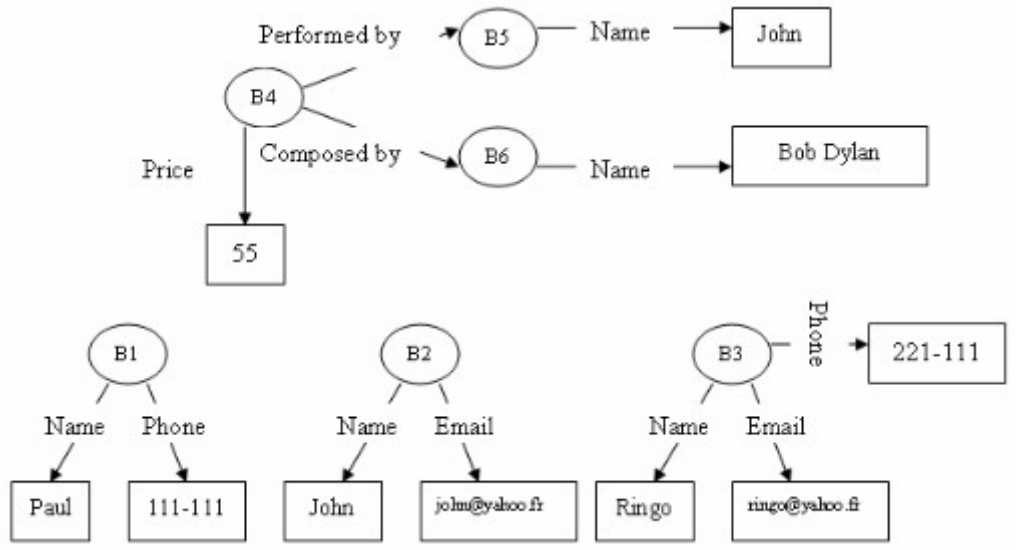

Рис. 1.6. Приклад графу RDF для запиту SPARQL

Під час експерименту [6] мова графових запитів SPARQL була лише чернеткою. Після появи він швидко став найвідомішою та найуживанішою мовою запитів для RDF. SPARQL дозволяє запитувати дані, які відповідають специфікації RDF W3C. У січні 2008 року SPARQL 1.0 став офіційною рекомендацією W3C. SPARQL 1.1 вийшов у березні 2013 року.

Модель даних RDF і мова запитів SPARQL є останніми структурами даних, що підтримують графи, перед розробкою та впровадженням спеціальних мов графових запитів.

1.3.4. Мови графових запитів

Першою мовою графових запитів була G. G не було запропоновано як альтернативу реляційним мовам запитів, а радше як додаткову мову для спрощення формулювання рекурсивних запитів. Розробка G започаткувала рух теоретичних пропозицій щодо інших мов графових запитів. Деякі ранні

мови запитів до графів — це GRAM і GraphDB, які використовують регулярні вирази для визначення шаблону графа. Інші мови, такі як QBD, мали візуальний діаграмний інтерфейс. Протягом 90-х років розвиток GQL був затьмарений появою XML, який розглядався як основна альтернатива реляційним базам даних для напівструктурованих даних. Однак поява додатків із циклічними зв’язками дала новий імпульс розвитку GQL.

Останні розробки GQL включають розвиток мов різних виробників, таких як Cypher і SPARQL.

Висновки до розділу

У першому розділі було проведено огляд особливостей процесів візуалізації при роботі з графовими базами даних. На основі аналізу сучасних даних та баз даних, зроблено висновок про важливість графових структур для представлення та обробки взаємопов'язаних даних.

Зокрема, досліджено графи та їх графічну інтерпретацію як ефективний спосіб відображення зв’язків між об’єктами, що значно спрощує аналіз складних структур даних. Історія розвитку графових баз даних продемонструвала поступову еволюцію підходів до зберігання і обробки даних, включаючи перехід від об'єктно-орієнтованих систем і XML до NoSQL, GraphDB, RDF та SPARQL.

Крім того, розглянуто різні мови графових запитів, що забезпечують можливість гнучкого та ефективного доступу до даних у графових базах. Це підвищує їх функціональність і робить графові бази даних популярними у сферах, де необхідно працювати з високозв’язними даними, такими як соціальні мережі, рекомендаційні системи та інші аналітичні додатки.

Отже, графові бази даних, з їхніми унікальними можливостями для візуалізації та аналізу складних зв'язків, мають значний потенціал для використання в сучасних інформаційних системах.

РОЗДІЛ 2. МОДЕЛІ ТА АЛГОРИТМИ ПОБУДОВИ ВІЗУАЛІЗАЦІЙ ДЛЯ РОБОТИ З БАЗАМИ ДАНИХ

2.1. Проблеми та особливості візуалізації графових даних

У цьому розділі ми опишемо існуючі дослідження щодо візуалізації графових даних. Ця інформація допомагає структурувати оцінку систем візуальних запитів. Кінцева мета візуалізації — надати користувачам розуміння. Дослідження візуалізації доклали багато зусиль для розробки нових методів, які допоможуть користувачам отримати розуміння своїх даних.

2.1.1. Дизайн візуалізації

Першим кроком у процесі створення візуалізації є дизайн. Щоб створити найбільш оптимальну візуалізацію заданих даних, необхідно прийняти дизайнерські рішення. Існує чотири рівні прийняття рішень:

1. Охарактеризуйте завдання та дані у словнику проблемної області. Результатом має бути детальний набір запитів або дій, які виконуються цільовими користувачами для збору даних. Для графічних даних це, як правило, стислий набір запитань, на які користувачі хочуть отримати відповідь, як-от можливість реалізації базових шаблонів графіків.

2. Анотація до операцій і типів даних. Результатом має бути опис операцій і типів даних. Це вхідні дані, необхідні для прийняття рішень щодо візуального кодування на наступному рівні. Іншим аспектом цього етапу є перетворення необроблених даних у типи даних, до яких можуть звертатися методи візуалізації. Для даних графіка існує лише один тип даних, що робить це невеликим кроком у процесі.

3. Дизайн візуального кодування та техніки взаємодії. Вибір техніки взаємодії залежить від мети системи.

4. Створіть алгоритми для ефективного виконання цих методів. Обговорення різних алгоритмів виходить за рамки цього опитування.

Якщо ці кроки використовуються під час процесу візуалізації, дизайнери можуть забезпечити надійний кінцевий результат. Крім того, цю модель також можна використовувати для оцінювання та перевірки візуалізацій. Усі ці кроки можуть виявити загрози дійсності. Помилка на першому рівні може означати, що ваша візуалізація дозволяє дії, які користувачі не зробили б, помилка на четвертому рівні може зробити процес візуалізації надто повільним.

Ще один тип оцінки – це перевірити, чи можна покращити візуалізацію. Є три види оцінювання:

1. Формуюче: «Чи можу я зробити це краще?»
2. Підсумкове: «Це правильно?»
3. Дослідницький: «Чи можу я зрозуміти більше?»

Ці заходи частково пов’язані з когнітивними процесами та людським розумом. Додаткову інформацію про те, як візуалізацію можна покращити за допомогою дизайну, орієнтованого на користувача, див Розділ 3.2.4 .

Однак, навіть дотримуючись наведених вище вказівок, все одно важко створити ідеальний дизайн. Наприклад, можуть бути компроміси у співпраці експертів із візуалізації та експертів із домену. Крім того, самі дані можуть мати властивості, які ускладнюють візуалізацію, наприклад їх масштаб і розміри.

2.1.2. Проблема масштабування

Візуалізація даних графіка швидко стає складною, коли розмір графіка збільшується. Кількість вузлів і ребер на графіку може швидко перевищити кількість пікселів на екрані. Згодом пошук хорошої відправної точки для дослідження даних стає важким завданням.

В [12] автор припускає, що спрямування уваги може допомогти користувачам знайти хороші відправні точки для аналізу. Оскільки аномалії в

даних часто представляють нові знання, спрямування уваги на основі виявлення аномалій може безпосередньо призвести до розуміння.

Інше рішення було запропоновано в [14] де описується, як візуалізацію можна покращити за допомогою спрощення графів. У процесі використовується виділення або спрощення мотивів, що означає, що загальні шаблони вузлів і посилань замінюються значущими гліфами, як показано на рисунку 2.1.

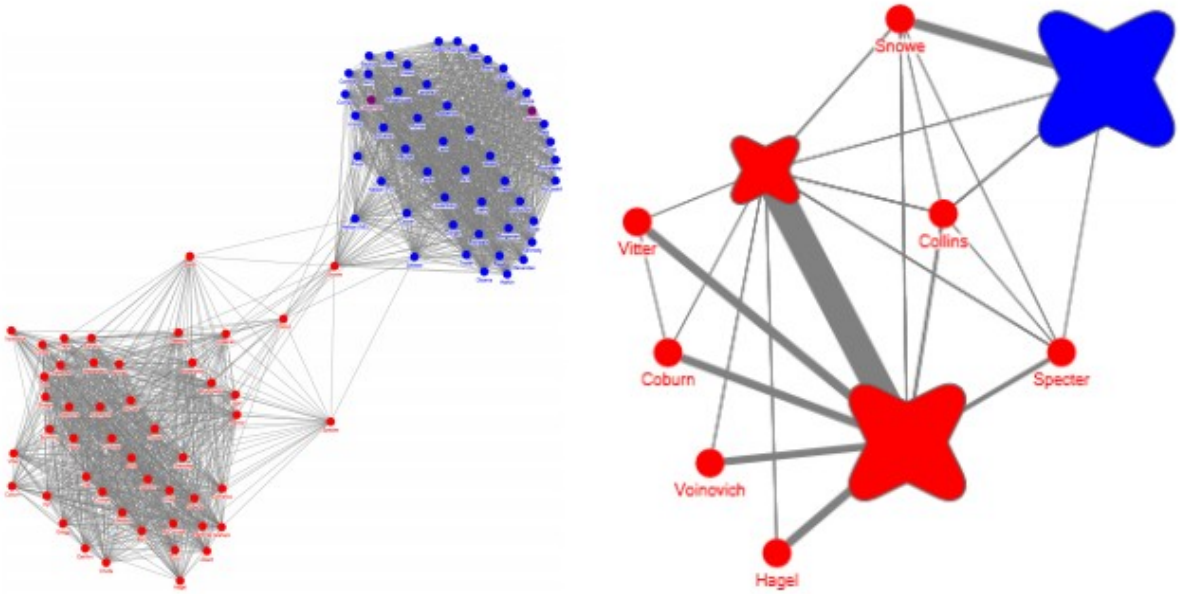

Рис. 2.1. Спрощення графа за допомогою вилучення мотиву. Ключові особливості залишилися видимими

Після застосування цього процесу ключову структуру графіка все ще видно. Однак зображення набагато менше захаращено, а тому його легше зрозуміти. Графік також вимагає менше місця на екрані, але зберігає багато основної інформації.

2.1.3. Методи представлення даних

Інша проблема, яку не розглядають рекомендації щодо проектування, полягає в тому, що існує багато різних представлень даних. Нижче ми

обговоримо всі представлення та макети, які регулярно використовуються для візуалізації даних графіків.

Представлення, яке найчастіше використовується для даних графа, — це діаграма вузол-зв’язок, яку також називають явним представленням. На діаграмі вузол-зв’язок вузли представлені двовимірними геометричними фігурами, такими як кола чи квадрати, або текстовими мітками. Краї найчастіше позначаються відрізками або стрілками.

Часто пропонують такі основні принципи для гарного макета вузлів:

• Зведені до мінімуму перетини країв;

• Мінімізована відстань сусідніх вузлів;

• Мінімізована область графу;

• Рівномірна довжина краю;

• Зведені до мінімуму вигини країв;

• Максимальна кутова відстань між різними краями;

• Співвідношення висота:ширина приблизно 1 (не надто довго і не надто широко);

• Симетрія: схожі структури графів повинні виглядати подібно.

Однак ці вказівки можуть суперечити, як-от мінімізація перетину країв і однакова довжина країв. Цей приклад наведено на рисунку 2.2.

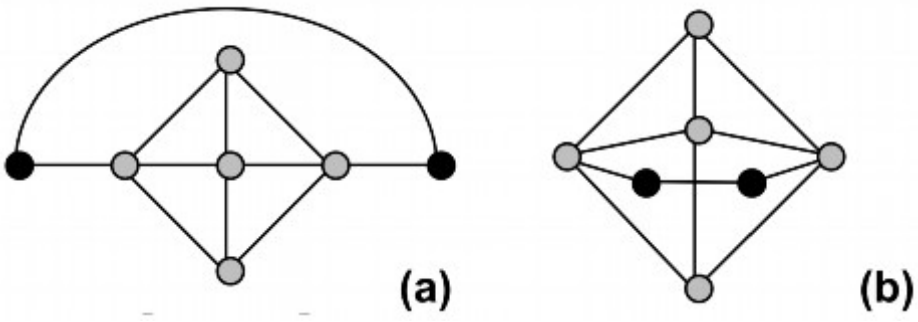

Рис. 2.2. Конфлікт між правилами гарного розташування "вузол-зв'язок".

У цьому випадку (рис. 2.2) мінімізація перетину ребер конфліктує з однаковою довжиною ребер.

Для представлень зв’язків вузлів існує кілька різних макетів графів. По-перше, ієрархічний макет графа малює всі вершини графа горизонтальними шарами. Це найчастіше використовується для орієнтованих графів, де ребра спрямовані вниз. Силово-спрямований макет зосереджується на естетиці, так що всі краї мають більш-менш однакову довжину, а також мінімізує перетинання країв. Макет на основі обмежень фіксує обмеження, які додаються до позиціонування вузлів і країв. Нарешті, макет , керований атрибутами, використовує додаткові атрибути з даних, щоб додати розміри до позиціонування даних. Крім того, можливі гібридні планування, які поєднують попередні підходи.

На відміну від явних представлень існують неявні представлення , де ребра не представлені як явні об’єкти. Як і у випадку з явним представленням, вузли представлені як графічні примітиви. Ребра можуть бути представлені за допомогою включення, перекриття або суміжності вузлів. Прикладами є карти дерев і сонячні промені матриць можна досягти більш компактного та менш складного представлення . Будь-який граф також може бути представлений матрицею суміжності. Фокус матриці суміжності знаходиться на наборі ребер, а не на вузлах. Це означає, що він може добре показувати кліки та кластери. Однак матричні представлення не підходять для розріджених графів, оскільки вони мають квадратичну вимогу до простору на екрані.

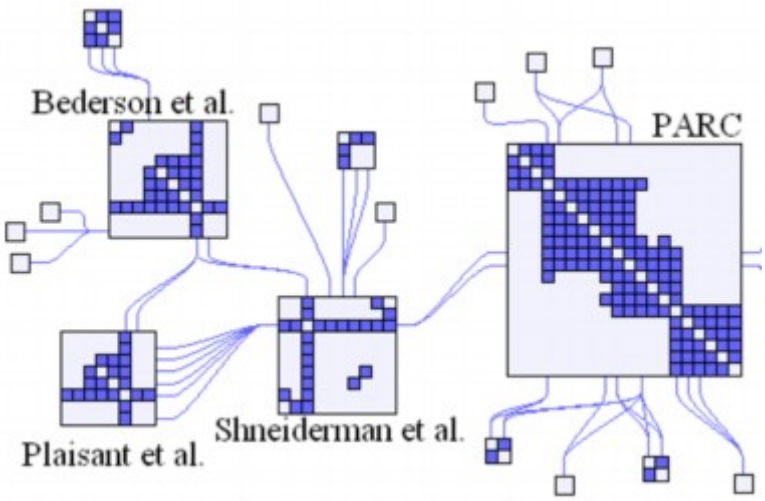

Рис. 2.3. NodeTrix, гібридне представлення графа

Крім того, існують гібридні представлення, такі як менші матриці для клік, з’єднаних посиланнями. Одним із прикладів такого інструменту є NodeTrix. На рисунку 2.3 показано гібридне представлення.

2.2. Шаблони побудови візуальних систем запитів

Різні дослідники писали про тему систем візуальних запитів під різними назвами. У деяких документах використовується термін Visual Query Language (VQL), інші використовують слово інструмент або інтерфейс, щоб описати свою реалізацію. Щоб узагальнити цю область, ми будемо використовувати назву Visual Query Systems (VQS) або Visual Graph Query Systems (VGQS). Системи візуальних запитів — це системи запитів, які використовують візуальні представлення для позначення предметної області та для вираження запитів. Вони дозволяють робити запити нетехнічним користувачам і запроваджують механізм для зручної навігації в будь-яких ситуаціях.

Зазначається, що, на відміну від представлень запиту, візуальне представлення може відрізнятися від представлення бази даних. В попередньому розділі ми показали, що дані графа можуть бути представлені явно, неявно, за допомогою матриць і в гібридній формі. Візуальні представлення запитів часто мають одну або декілька з трьох форм: представлення на основі форм, діаграми та іконки. Нижче ми перевіримо, чи ці представлення також працюватимуть для мов запитів візуальних графів.

Представлення на основі форм є найпоширенішими для запитів, це простий спосіб дозволити користувачам-початківцям формулювати запити. Це пояснюється тим, що користувачам не потрібне уявлення про структуру даних, але вони можуть використовувати пошук за ключовими словами, щоб знайти те, що вони шукають. Основною характеристикою представлення на основі форм є те, що це структуроване представлення, яке відповідає абстракції звичайних паперових форм. Це може включати елементи форми,

такі як перемикачі, прапорці та спадні меню. Підхід, заснований на формі, є привабливим, оскільки не вимагає особливої уваги до деталей з точки зору користувача. Він може надати користувачеві високий рівень виразної сили, а також бути простим у використанні.

Адекватна візуалізація результатів запиту також важлива для зручності використання системи. На додаток до класифікації візуального представлення запиту, також представляють візуальні представлення результатів запиту. Вони класифікують їх на основі форм, на основі діаграм і знову на основі піктограм, але значення дещо відрізняються від значень для представлень запитів. Тому ми розглянемо їх нижче.

Представлення результатів у формі означає, що результати запиту представлені в таблицях або списках, які можна прокручувати. Це дуже компактна форма, особливо у разі видалення дублікатів.

Представлення на основі діаграми відноситься до будь-якої графіки, яка використовує положення та величину для кодування інформації. Для графічних даних деякі VQS можуть відображати результат запиту в діаграмі того самого типу, що й вихідний запит.

Представлення на основі піктограм використовує піктограми для візуалізації результату запиту. Піктограма може показувати тип даних із текстовими мітками, що ідентифікують кожне входження. Таким чином можна легко побачити, який тип даних було отримано, що полегшує аналіз.

Щоб створити найбільш ефективну систему, було б корисно візуалізувати результати так само, як був представлений запит. Особливо це стосується зображень на основі значків і схем. Якщо інтерфейс добре розроблений, запити можуть стати ітеративною та, можливо, дослідницькою дією, де кожен результат пропонує користувачу нові запитання або уточнення запиту.

Кожне із зазначених вище представлень добре працює в певних ситуаціях і має як плюси, так і мінуси. Припускають, що ефективна система повинна адаптувати представлення даних до поставленого завдання, щоб

підтримувати розуміння даних. Наприклад, включення опції природної мови та візуальної опції дозволяє ефективно використовувати систему як початківцям, так і досвідченим користувачам.

Далі ми переходимо до візуалізації, щоб побачити, що являє собою запит на графік. Двома основними концептуальними особливостями сучасних графових мов запитів є шаблони графів і вирази шляху. Більшість мов графових запитів зосереджено на зіставленні цих типів запитів із набором даних.

Шаблон графа має таку саму структуру, як і тип бази даних графів, якій він має відповідати. Однак замість констант шаблон містить змінні як вузли або мітки країв. Збіг для цього шаблону – це відображення змінних у шаблоні на константи в базі даних, таким чином шаблон міститься в оригінальній базі даних. Приклад наведено на рисунку 2.4.

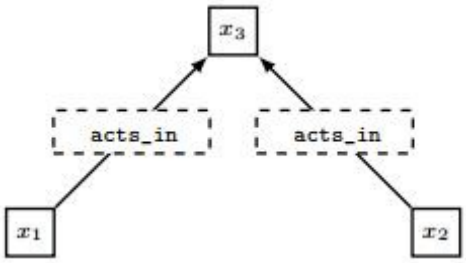

Рис. 2.4. Базовий шаблон графа для графа з мітками ребер

Ці основні моделі графів уже можуть охоплювати реляційні операції природного об’єднання та відбору на основі рівності. Більш складні шаблони потрібні для виконання інших реляційних операцій, таких як проекція, об’єднання, різниця та фільтр.

Вирази шляху або навігаційні запити дозволяють здійснювати навігацію топологією даних, що означає, що значення заповнюються для кожного вузла. Найпростішим типом навігаційного запиту є запит шляху, щоб дізнатися, чи існує шлях між двома названими вузлами. Це можна зробити з урахуванням міток країв або без них. Один із прикладів запиту шляху щодо міток країв, щоб дізнатися, чи є дві названі особи друзями.

Запити шляхів також можна використовувати для розширення шаблонів графів, що створює шаблони навігаційних графів, як показано на рисунку 2.5. Вони подібні до базових шаблонів графів, але мітки ребер можуть бути константами, запитами шляхів або для позначення довільних шляхів.

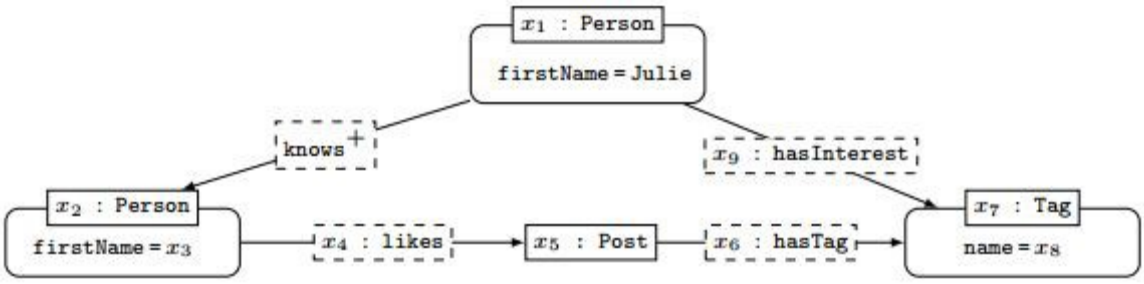

Рис. 2.5. Шаблон навігаційного графа для графа з мітками ребер

Існує ряд функціональних можливостей, які має виконувати мова графових запитів, щоб отримати повну виразність. Два з них були згадані в попередніх розділах. Зіставлення підграфів — це шаблон графа, а пошук вузлів, з'єднаних шляхами, — це вираз шляху або навігаційний запит. Інші функціональні можливості:

• Агрегація, наприклад підтримка MIN і MAX,

• Створення вузла,

• Ранжування, наприклад складання списку вузлів із найбільшою кількістю вихідних посилань,

• Наближена відповідність для великих графіків.

2.3. Методи та інструменти візуалізації запитів до бази даних

Протягом десятиліть SQL був основним стандартом для визначення запитів до реляційних баз даних, і навряд чи це зміниться найближчим часом. Таким чином, ми не пропонуємо нових способів написання запитів для

користувачів, а натомість досліджуємо, як допомогти їм зрозуміти існуючі запити SQL.

2.3.1. Візуальні мови запитів

Візуальні методи визначення запитів широко вивчалися, і багато комерційних продуктів баз даних пропонують певний візуальний інтерфейс для написання SQL користувачами. Ми зосереджуємося на проблемі опису та інтерпретації вже написаного запиту, що дуже відрізняється від проблеми допомоги користувачеві у складанні запиту. Основна відмінність полягає в тому, що розуміння запиту вимагає зосередження на високорівневій структурі, абстрагуючись від низькорівневих деталей та тонкощів. Навпаки, для точного визначення запиту необхідно представити можливі варіанти та конкретні деталі, що впливають на семантику запиту. У мовах програмування цей розрізнення чітко проводиться між візуальним програмуванням для розробки програми та візуалізацією програми для аналізу існуючої програми.

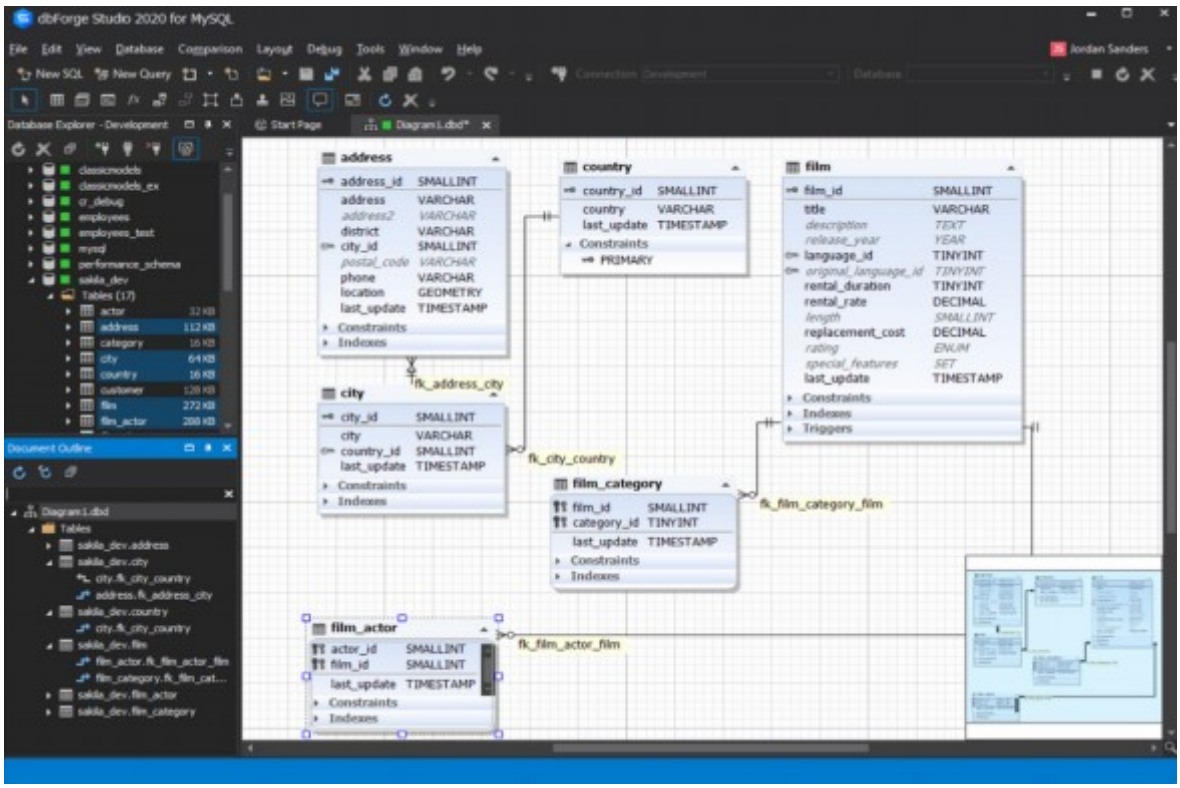

Рис. 2.6. Інструмент dbForge Studio for MySQL

Інтерактивні конструктори запитів використовують візуальні діаграми, якими користувачі можуть маніпулювати (найчастіше для вибору таблиць та атрибутів), використовуючи при цьому окремий конфігуратор запитів (подібний до умовних полів QBE) для визначення предикатів вибору, атрибутів та іноді вкладеності між запитами.

dbForge – найдосконаліший та комерційно підтримуваний інструмент, який ми знайшли для інтерактивного створення запитів. Проте він не показує жодних візуальних ознак для нееквівалентних з'єднань між таблицями, а фактичні значення фільтрації та агрегатні функції можна додати лише в окремому конфігураторі запитів. Більше того, він має обмежену підтримку вкладених запитів: внутрішні та зовнішні запити створюються окремо, а діаграма для внутрішнього запиту представлена окремо та не пов'язана з діаграмою для зовнішнього запиту. Таким чином, візуальне зображення корельованих підзапитів неможливе.

Інші графічні редактори SQL, такі як SQL Server Management Studio (SSMS), Active Query Builder, QueryScope від SQLdep, MS Access та pgAdmin3 PostgreSQL, мають ще більше недоліків у візуальному представленні запитів: більшість не дозволяє вкладені запити, жоден не має єдиного візуального елемента для логічних кванторів NOT EXISTS або FOR ALL, і всі вимагають вказівки деталей запиту в SQL або в кількох вкладках, окремо від візуальної діаграми.

DataPlay дозволяє користувачеві вказати свій запит, інтерактивно змінюючи дерево запитів з кванторами та спостерігаючи за змінами у відповідних/невідповідних даних. Заявлена мета - подолати відсутність синтаксичної локальності в SQL (тобто інакше подібні запити SQL з різною квантифікацією можуть мати дуже різну структуру). QueryVis розроблений з урахуванням тієї ж проблеми, але також використовує знайомі візуальні метафори для кон'юнктивних запитів. Коротше кажучи, сучасні графічні редактори SQL не забезпечують єдиної всеохоплюючої візуалізації запиту. Таким чином, вони не могли б (навіть теоретично) перетворити складний

запит SQL (такий як той, що наведено в розділі 1.1) в єдине візуальне представлення, що є основним напрямком нашої роботи.

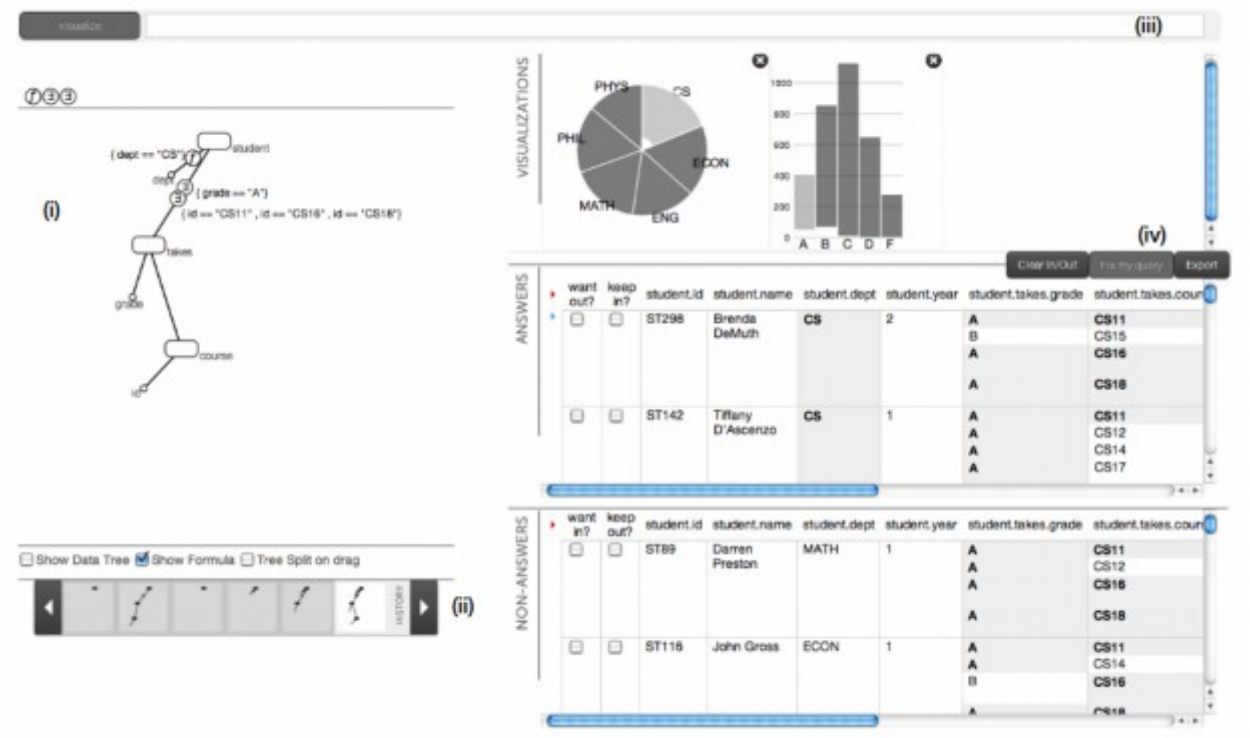

Рис. 2.7. Інтерфейс системи візуалізації запитів DataPlay: (i) дерево запитів (ii) інтерактивний графічний переглядач історії (iii) панель команд (iv) панель даних і візуалізації

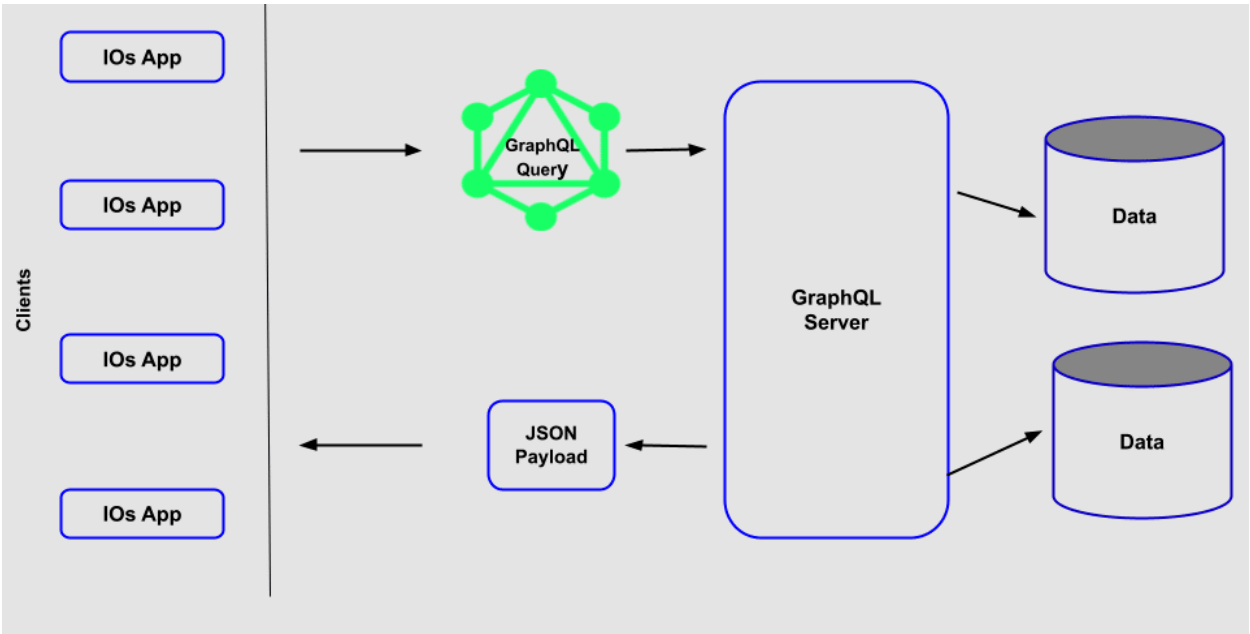

Рис. 2.8. Архітектура GraphSQL

Візуалізації запитів намагаються створити візуальне представлення існуючих запитів. Ця явна зворотна функціональність для SQL не привернула стільки уваги, скільки візуальні конструктори запитів. Два проекти, які найбільше відповідають цьому духу, - це GraphSQL та Visual SQL. Обидва - це візуальні мови запитів, які також підтримують візуалізацію запитів. GraphSQL використовує візуальні метафори, що відрізняються від типових позначень та візуалізацій реляційних схем, навіть прості кон'юнктивні запити можуть виглядати незнайомими. Visual SQL ближчий за дизайном. Зосереджуючись на специфікації запитів, він зберігає однозначну відповідність SQL, а синтаксичні варіанти одного й того ж запиту призводять до різних представлень. Модель графа запитів (QGM) допомагає користувачам зрозуміти плани запитів, а не намір запиту.

StreamTrace фокусується на візуалізації часових запитів над потоками за допомогою діаграм робочих процесів та часової шкали. Ми зосереджуємося на відображенні логіки, що лежить в основі загальних запитів SQL, незалежно від даних.

Інформаційна візуалізація з її метою допомогти користувачам зрозуміти та проаналізувати дан нещодавно привернула велику увагу спільноти баз даних. Подібно до нашої мети, дослідники візуалізації допомагають користувачам зрозуміти складні зв'язки, але в даних, а не в логіці запитів.

2.3.2. Підсвічування синтаксису

Редактори запитів та клієнти для основних СУБД вже давно використовують підсвічування синтаксису та вирівнювання блоків та речень запитів. Це корисно, але недостатньо, щоб допомогти користувачам зрозуміти намір запиту. У наших експериментах запити SQL автоматично мають відступи, а ключові слова пишуться з великої літери та виділяються кольором (див. рис. 2.9). Тим не менш, користувачі загалом краще реагували на наш підхід, ніж на візуально покращений текст SQL.

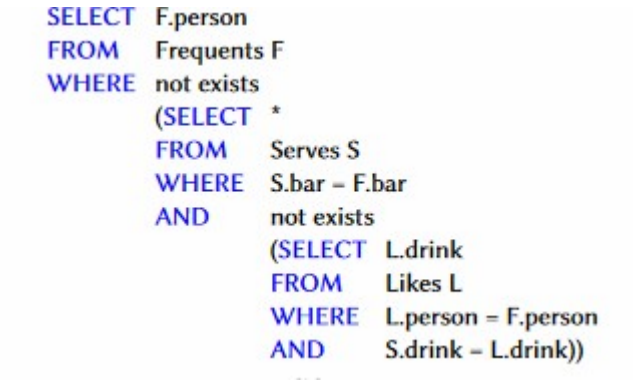

Рис. 2.9. Приклад запиту із підсвічуванням синтаксису

2.3.3. Переклади запитів природною мовою

Переклад між SQL та природною мовою - цікава та ретельно досліджена тема, і пропонуються різні ідеї для пояснення запитів природною мовою. Робота в цій галузі переконливо доводить, що автоматичне створення ефективного вільного тексту із запитів є складним, і що загальне завдання досить відрізняється від попередньої роботи над створенням інтерфейсів природною мовою для СУБД. Основним обмеженням сучасного стану справ є те, що він

1) створює довгі речення,

2) наразі обмежений простими запитами SQL,

3) текстові описи нелегко розкривають загальні логічні закономірності, що стоять за запитами.

Зокрема, нам не відомий жоден інструмент SQL для перекладу на природну мову, доступний сьогодні, який міг би перекласти запит в інтуїтивне представлення природною мовою.

У кількох роботах [2, 17] пропонується ілюструвати семантику операторів у програмі потоку даних або семантику запитів шляхом генерації прикладів вхідних та вихідних даних. Результатом є, по суті, список кортежів для кожного реляційного оператора, який, подібно до пояснення природною мовою, нелегко розкриває логічну закономірність, що лежить в основі запиту.

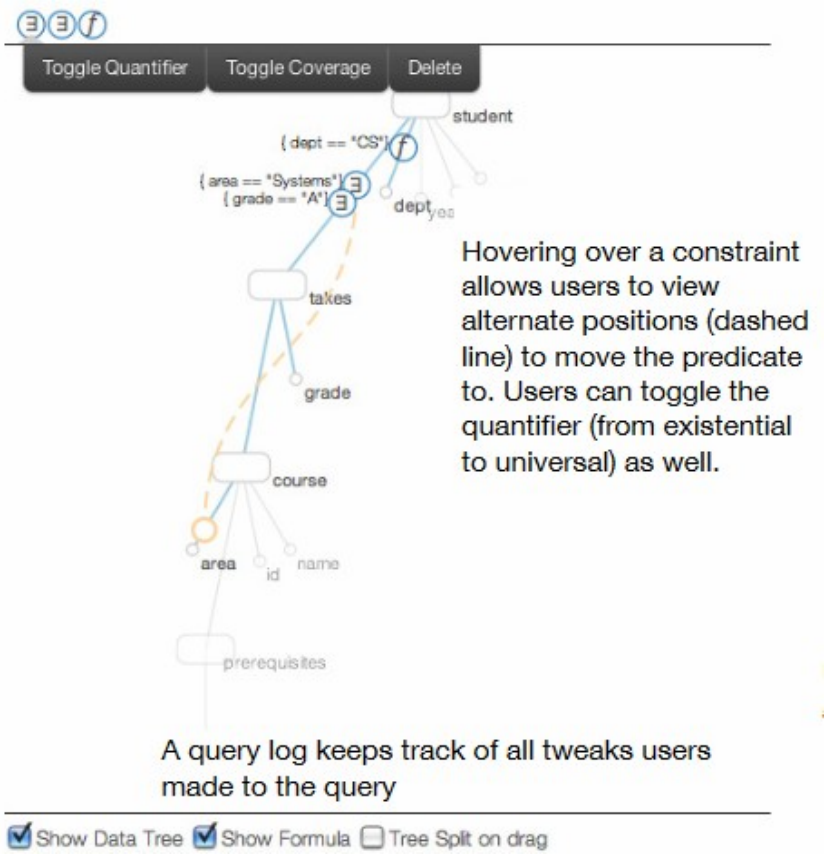

Рис. 2.10. Приклад системи, що виконує ілюстрування семантики операторів в запиті

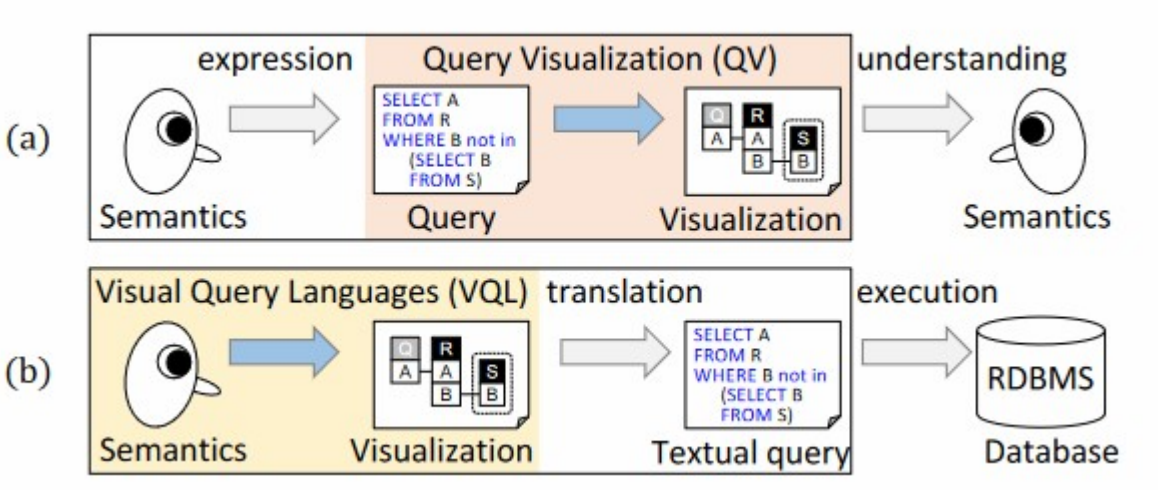

Рис. 2.11. Порівняння візуалізації запитів (зверху (a), помаранчевий колір) з візуальними мовами запитів (VQL) (знизу (b), жовтий колір).

Таким чином, модифікація вже існуючого запиту може бути ефективним способом написання нових запитів. Однак модифікація існуючого запиту вимагає спочатку його розуміння. З цієї причини важливо допомогти користувачам зрозуміти запити, і візуалізація є одним з очевидних шляхів. Хоча візуальні методи вираження запитів широко вивчалися в літературі з баз даних в рамках теми візуальних мов запитів (VQL), проблеми підтримки явної зворотної функціональності створення візуального представлення існуючого запиту ("візуалізація запитів") в цілому відрізняються від проблеми складання нового запиту (рис. 2.11).

2.4. Методологія візуалізації вкладеності SQL запитів

Наша основна мета - додати вкладеність до кон'юнктивних запитів. Ми кажемо, що SQL-запит є вкладеним, якщо він містить принаймні один підзапит. Це важливий крок, оскільки вкладені запити (зокрема, корельовані вкладені запити) може бути дуже важко інтерпретувати, і їх навіть не підтримує більшість візуальних конструкторів запитів. Ми зосереджуємося на найвиразніших підзапитах, тобто тих, що знаходяться всередині речення WHERE та операторів EXISTS, NOT EXISTS, IN, NOT IN, ANY або ALL.

Q::=	SELECT C [, C, ..., C] | *	select clause
|	FROM S [, S, ..., S]	from clause
|	[WHERE P]	where clause
C::=	[T.]A	column or attribute
S::=	T [AS T]	table (table alias)
P::=	P [AND P ... AND P]	conjunction of predicates
|	C O C	join predicate
|	C O V	selection predicate
|	[NOT] EXISTS (Q)	existential subquery
|	C [NOT] IN (Q)	membership subquer
|	C O {ALL | ANY (Q)}	quantified subquery
O::=	< | ≤ | = | <> | ≥ | >	comparison operator
T::=		table identifier
A::=		attribute identifier
V::=		string or number

Рис. 2.12. Граматика підтримуваного фрагмента SQL

На рисунку 2.12 оператори, укладені в [], є необов'язковими; оператори, розділені |, вказують на вибір між альтернативами. Рисунок 2.12 показує наш фрагмент SQL, що підтримується на даний момент. Ці запити мають ту саму виразність, що і реляційне числення та його інтерпретація на основі множин. Однак наразі ми не підтримуємо диз'юнкції і тому називаємо цей фрагмент SQL вкладеними кон'юнктивними запитами з нерівностями. Крім того, SQL-запити повинні відповідати двом незначним обмеженням, і стверджуємо, що вони виконуються будь-яким значущим невиродженим SQL-запитом.

Позначення. Предикат має форму exp1 op exp2, де щонайбільше один з exp є константою (наприклад, 3 або 'Alice'), а інший(і) - імена атрибутів, необов'язково з псевдонімами таблиць (наприклад, table1.attr2). Якщо предикат має константу, це предикат вибору, інакше - предикат з'єднання. Оператор op є елементом {<, ≤, =, <>, ≥, >}. Блок запиту складається з речень SELECT, FROM та WHERE, включаючи визначені в них псевдоніми таблиць та предикати. Ми називаємо блок запиту на глибині вкладеності 0 кореневим блоком запиту. Область дії блоку запиту - це набір блоків запиту, для яких псевдоніми таблиць, визначені в ньому, є дійсними.

Щоб візуалізувати вкладені кон'юнктивні запити з нерівностями, ми повинні розширити візуалізації для кон'юнктивних запитів, щоб додатково включити завдання "Квантори" та "Порядок вкладеності" . В SQL "Квантори" та "Порядок вкладеності" - це важливі концепції, що стосуються логіки та структури запитів, особливо коли мова йде про вкладені запити.

Квантори визначають, як саме підзапит співвідноситься з основним запитом. В SQL існує два основні типи кванторів:

- EXISTS (існує): Перевіряє, чи повертає підзапит хоча б один рядок. Якщо так, умова з квантором EXISTS вважається істинною.

- ALL (всі): Перевіряє, чи задовольняють всі рядки, повернуті підзапитом, певній умові.

З цією метою ми розширюємо візуалізацію кон'юнктивних запитів, щоб дозволити вкладені запити до глибини 3. Ми ілюструємо дизайн для вкладених запитів, використовуючи рис. 2.14 b та пов'язані з ним візуалізації рис. 2.13 b та 2.13 c.

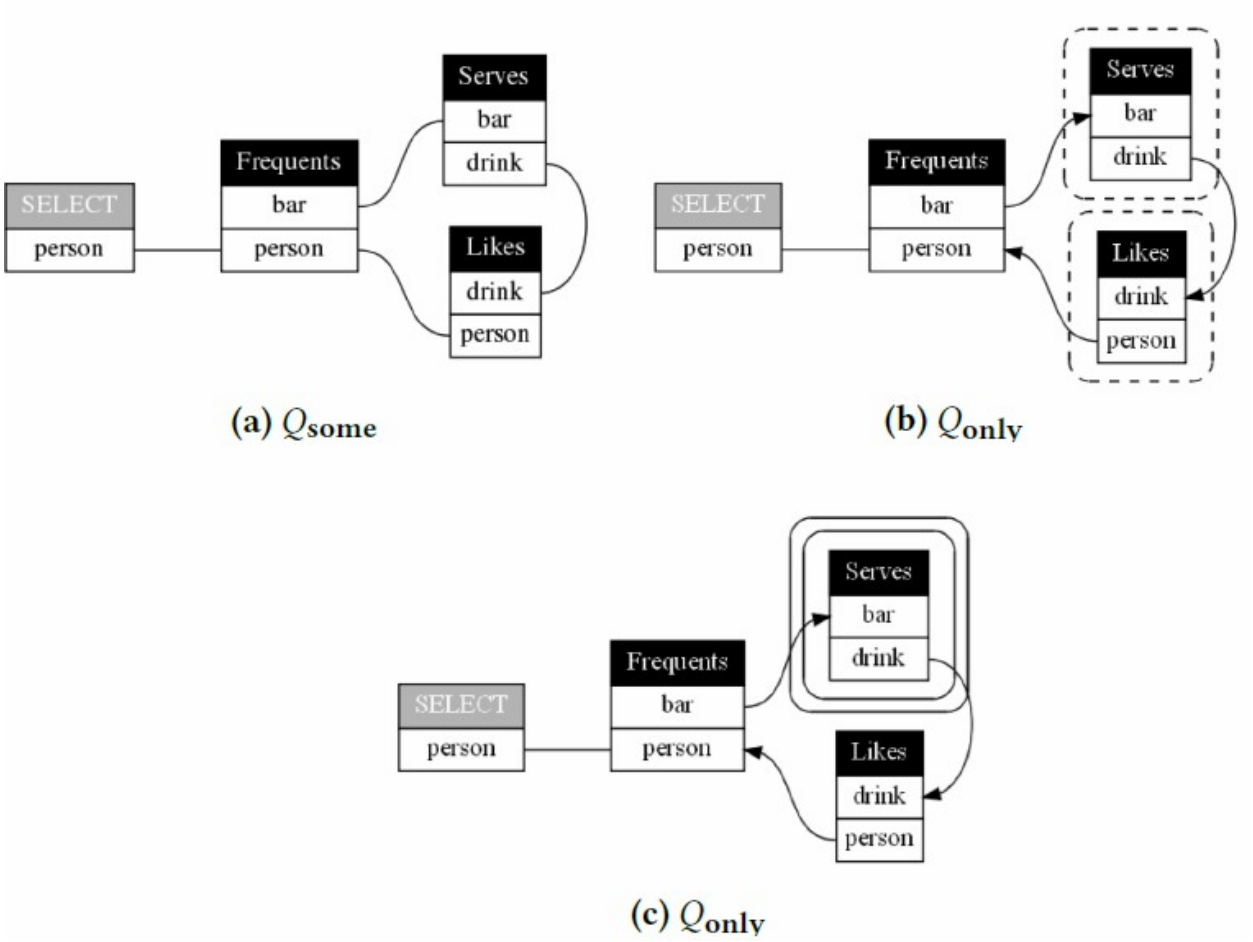

Рисунок 2.13. a) Діаграма кон'юнктивного запиту, показаного на рис. 3a. Зверніть увагу, як діаграма точно відповідає знайомим позначенням реляційної схеми.

b) Діаграма вкладеного запиту, показаного на рис. 3b. Тут додали пунктирну рамку для ∃, а порядок читання можна знайти, слідуючи стрілкам.

c) Рис. 2.13 b можна ще більше спростити за допомогою квантора ∀ (рамка з подвійною лінією), логічного та інтуїтивно зрозумілого оператора, який не існує в SQL.

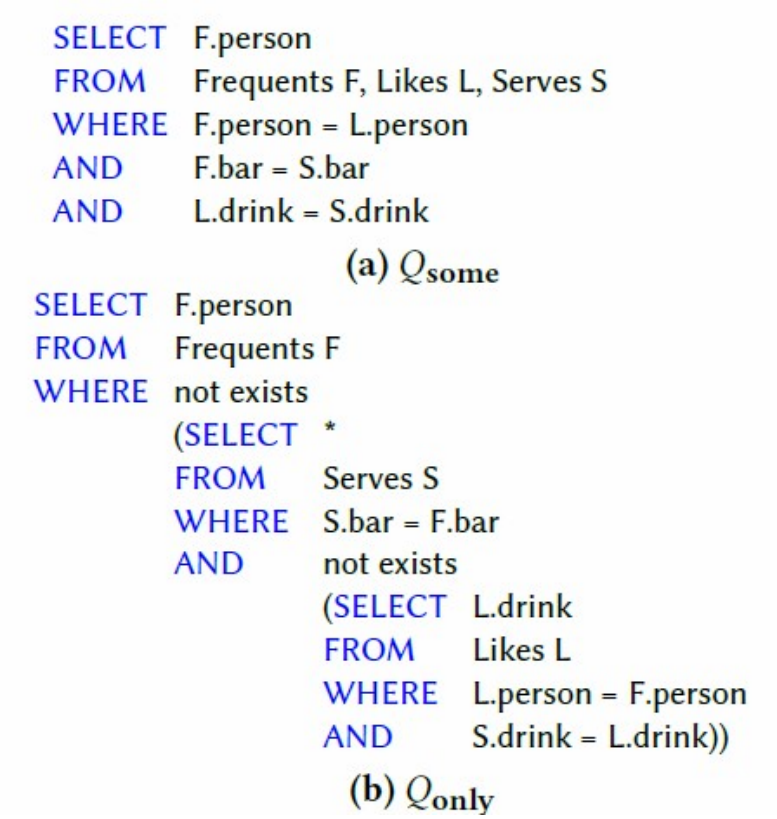

Рис. 2.14. Приклади запитів з використанням кваторів

На рисунку 2.14 (a) Qsome: Знайти осіб, які часто відвідують якийсь бар, де подають ХОЧА Б ОДИН напій, який їм подобається. Відповідно (b) Q only: Знайти осіб, які часто відвідують якийсь бар, де подають ТІЛЬКИ напої, які їм подобаються ≡ ... якийсь бар, де НЕ подають ЖОДНОГО напою, який їм НЕ подобається.

2.4.1 Дизайн та ефективність візуалізації

Ми розширюємо дизайн діаграм, зберігаючи ті самі кодування для завдань, і вибираємо найефективніші додаткові позначки та канали для завдань "Квантори" та "Порядок вкладеності" з абстракцій групування та ієрархії відповідно.

1) Групування. Для кванторів ми повинні діяти на рівні вище завдання "Таблиці та атрибути", щоб згрупувати таблиці на основі квантора. Однак ми можемо використовувати ті самі принципи [7] для розробки позначки

області/обмежувальної рамки, яка охоплює набір таблиць. Щоб відрізнятися від рамок таблиць, ми використовуємо позначку заокругленого прямокутника. Оскільки нам потрібно кодувати лише два квантори {∃, ∀}, ми вирішили використовувати пунктирний та подвійний стиль лінії відповідно, щоб уникнути маркування. ∃ пунктирні лінії показані на рис. 2.13 b, тоді як спрощене представлення з ∀ подвійними лініями показано на рис. 2.13 c.

2) Ієрархія. Ієрархію можна ефективно візуалізувати як кореневе дерево або подібну структуру "вузол-зв'язок". Для завдання "Порядок вкладеності" ми могли б зобразити вкладеність за допомогою логічного дерева, як ми це робимо на рис.2.15.

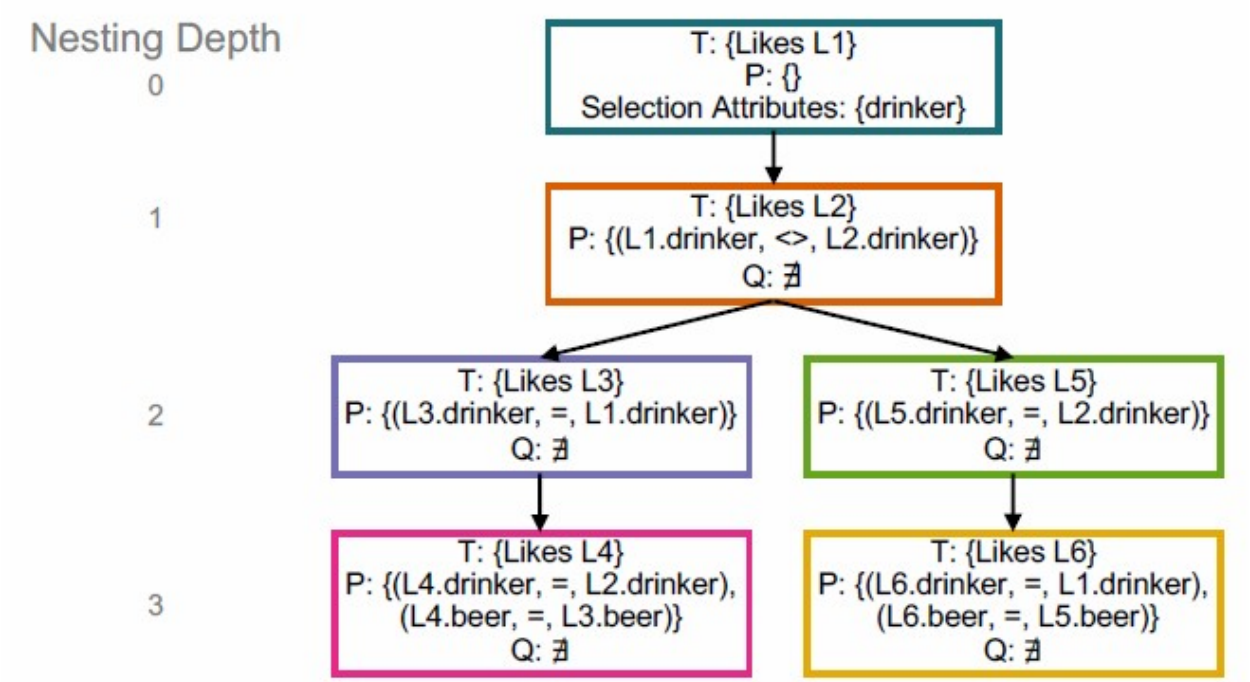

Рис. 2.15. Логічне дерево представлення запиту

Це вимагало б двох типів стрілок: один для представлення порядку вкладеності підзапитів, як у логічних деревах, і інший для представлення з'єднань таблиць, як ми це робимо для кон'юнктивних запитів. Відомо, що одного лише позиційного кодування (також показано на рис. 2.15 для

логічних дерев) недостатньо для зображення складніших вкладених запитів (наприклад, рис. 2.16). Іншим підходом було б візуально вкладати набори таблиць в обмежувальні рамки, але це, ймовірно, призвело б до захаращених візуалізацій.

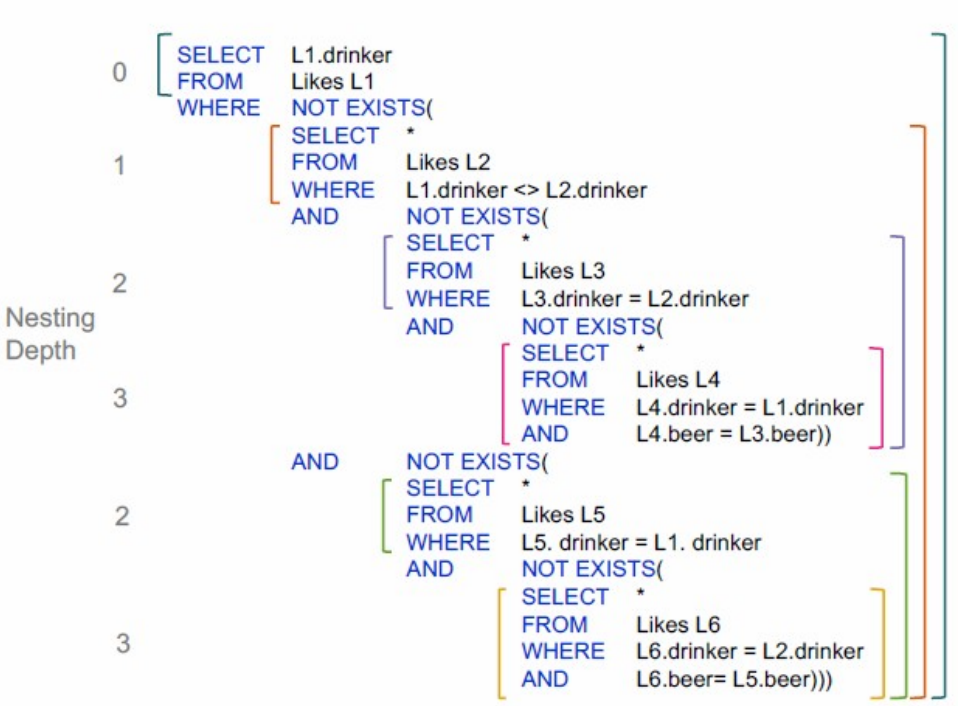

Рис. 2.16. Запит на унікальну множину

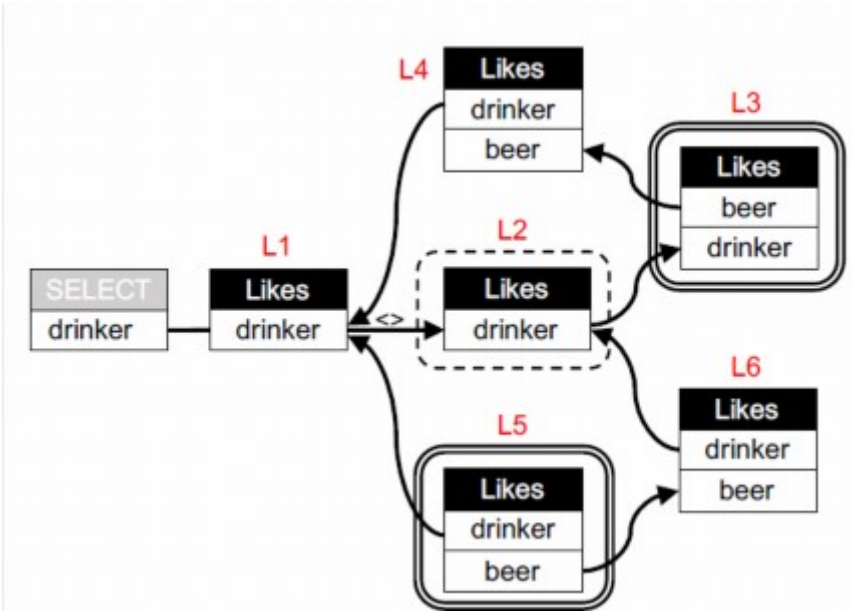

Рис. 2.17. Візуалізація запиту на рисунку 2.16

На рисунку 2.16 показано запиту на унікальну множину за схемою "бар-відвідувач-пиво", метою якого є знаходження відвідувачів, яким подобається унікальний набір сортів пива. Глибина вкладеності кожного підзапиту позначена сірим кольором ліворуч; область дії кожного підзапиту показана дужками праворуч, а їхні відповідні "корені" - дужками ліворуч.

На рисунку 2.17 показана візуальна діаграма для того самого запиту (рис. 2.16). Червоні псевдоніми таблиць поруч із таблицями не є частиною діаграми та розміщені лише для ілюстрації відповідності SQL-запиту. Зверніть увагу, що візуальний шаблон праворуч однаковий для різних SQL-запитів, які відповідають тій самій логічній схемі, наприклад, знайти сорти пива з унікальним набором відвідувачів або знайти фільми з унікальним складом акторів або знайти клієнтів з унікальним набором придбаних товарів. Таким чином, наші діаграми дозволяють користувачам перевіряти та розпізнавати основну логічну схему.

Нижче ми покажемо, що, просто використовуючи правила стрілок, які ми надаємо для порядку читання, ми завжди можемо відновити правильний порядок вкладеності кожної таблиці в SQL-запиті. Отже, додаткові позначки, що кодують вкладеність, були б надлишковими, якщо ми гарантуємо, що стрілки належним чином додані до лінійних позначок для предикатів з'єднання, щоб неявно кодувати порядок вкладеності. Зокрема, ми (a) повинні використовувати спрямовані ребра (лінії зі стрілками) для рівноз'єднань таблиць, які знаходяться на різних глибинах вкладеності, і (b) визначати напрямок стрілки виключно за правилами стрілок, а не за порядком атрибутів навколо оператора.

Як приклад останнього обмеження, припустимо, що ми маємо умову з'єднання A.attr1 > B.attr2, де спрямована лінія, проведена від A до B, позначає порядок операторів. Однак таблиця B є батьківською для таблиці A у вкладеності, і тому правила стрілок стверджують, що спрямоване ребро має бути намальовано B→A. Таким чином, ми повинні переписати з'єднання з еквівалентною умовою B.attr2 < A.attr1.

2.4.2. Мінімальність візуалізації

Так само, як ми це зробили для кон'юнктивних запитів, ми показуємо, що наша візуалізація є мінімальною також для вкладених кон'юнктивних запитів з нерівностями. Видалення будь-якої з її позначок або каналів робить неможливим однозначне виконання всіх завдань користувача. Зокрема, ми обговорюємо дві додаткові та змінені позначки: лінії зі стрілками/мітками та обмежувальні блоки.

1) Обмежувальний блок. Позначка заокругленого прямокутника охоплює блок запиту, тобто всі таблиці, до яких застосовується квантор. Видалення її зробило б завдання "Квантори" нездійсненним. Однією з альтернатив використанню обмежувальної рамки було б приєднання квантора та мітки або кольору блоку до кожної таблиці, але це вимагало б додавання більшої кількості позначок/каналів і не забезпечило б користувачеві миттєвого групування.

2) Стрілки/мітки ліній. Окрім ідентифікації пари атрибутів, що беруть участь у з'єднанні для підтримки завдання "Таблиці та атрибути", лінії та їх стрілки/мітки також необхідні для завдання "Порядок вкладеності". Видалення ліній або їх складових стрілок/міток зробило б ці завдання нездійсненними. Однак правила стрілок можна використовувати для однозначного визначення порядку вкладеності підзапитів у SQL-запиті з цим обмеженим додатковим кодуванням. Альтернативні підходи, такі як позиційне кодування або вкладене обмеження, були б недостатніми в багатьох випадках і, ймовірно, вимагали б набагато більше позначок і, отже, більше "чорнила" для відображення тих самих даних.

Діаграми читаються, починаючи з таблиці SELECT і слідуючи обходу в глибину з перезапусками з невідвіданих вихідних вузлів (тобто тих, що не мають вхідних стрілок). Припустимо, ребро йде від S.attr1 до T.attr2, до T застосовано квантор ∃, а ребро позначено оператором порівняння <. Тоді ми можемо інтерпретувати це так:

Знайти attr1 з S таким, що не існує жодного кортежу в T, де S.attr1 < T.attr2. Згадайте, що непозначені ребра представляють рівноз'єднання. Також зверніть увагу, що якщо дві таблиці з одного блоку запиту, то вони трактуються так, ніби до T застосовано квантор ∃. Після того, як ми закінчимо інтерпретувати ребро, нам потрібно додати AND до нашої інтерпретації, оскільки ми представляємо кон'юнкцію предикатів.

2.4.3. Перетворення SQL в діаграми

Тут ми надаємо огляд процесу створення діаграми. Спочатку ми перетворюємо SQL-запит на кортежне реляційне числення (TRC), яке є добре вивченим перетворенням до FOL. Як наслідок перетворення в FOL, ми розглядаємо множинну семантику, двозначну логіку (без NULL) і без агрегатних функцій. Більше того, ми з'єднуємо кілька предикатів лише кон'юнкціями (тобто диз'юнкції не допускаються). У FOL нам більше не потрібно мати справу з різними синтаксичними варіантами операторів SQL, які не додають виразності. Це означає, що оператори, такі як IN, NOT IN або ALL, будуть перетворені на відповідні квантори FOL ∃, ∄ або ∀.

Замість того, щоб використовувати представлення TRC запиту, стає легше міркувати над еквівалентним представленням, яке робить вкладені області дії кванторів явними у формі дерева, яке ми позначаємо як логічне дерево (LT). Це кореневе дерево, кожен вузол якого представляє блок запиту. Кореневий вузол представляє кореневий блок запиту, а структура дерева кодує ієрархію вкладеності, тобто таблиці та атрибути вузла можна посилати в будь-якому піддереві. Кожен вузол у LT містить таку інформацію:

1) Таблиці (T): Набір таблиць (або псевдонімів таблиць), визначених у його корені області дії;

2) Предикати (P): Набір предикатів, що використовуються в блоці запиту. Кілька предикатів у наборі пов'язані кон'юнкцією (тобто предикат1 ∧ предикат2 тощо);

3) Квантор (Q): Квантор, застосований до предикатів, включаючи його заперечення. Це або ∃, ∄, або ∀.

Крім того, для кореневого вузла логічного дерева (LT) ми також вказуємо атрибути в його списку вибору (див. рис. 2.15).

У SQL запити з універсальними кванторами виражаються через вкладені підзапити NOT EXISTS, що ускладнює їх читання. Ми спрощуємо LT з вкладеними кванторами ∄, застосовуючи стандартне логічне перетворення, щоб перетворити їх на квантори ∀. Зокрема, якщо вузол LT має ∄ як свій квантор і має лише один дочірній вузол, який також має ∄ як свій квантор, тоді ми можемо перетворити так, щоб він мав квантор ∀, а ′-квантор ∃.

Щоб зрозуміти, чому, розглянемо наступне перетворення, де ми застосовуємо закон Де Моргана як виразом пропозиційної логіки та на двох вузлах LT. Тут ми пишемо T і S для набору таблиць у двох блоках запиту:

$$\begin{aligned}
&\neg\exists \mathbf{S}.(p_1 \wedge \cdots \wedge p_k \ \wedge \ \neg\exists \mathbf{T}.(p_{k+1} \wedge \cdots \wedge p_{k+\ell})) \\
&\forall \mathbf{S}.\neg((p_1 \wedge \cdots \wedge p_k) \ \wedge \ \neg\exists \mathbf{T}.(p_{k+1} \wedge \cdots \wedge p_{k+\ell})) \\
&\forall \mathbf{S}.((p_1 \wedge \cdots \wedge p_k) \ \rightarrow \ \exists \mathbf{T}.(p_{k+1} \wedge \cdots \wedge p_{k+\ell}))
\end{aligned}$$

Як приклад, застосування цього перетворення до LT, показаного на рис. 2.15, призвело до нашої діаграми з рис. 2.17.

Потім ми перетворюємо LT на нашу діаграму. Починаючи з кореневого вузла, ми обходимо дерево LT в ширину і обробляємо кожен вузол наступним чином:

1) Створюємо таблицю з її атрибутами для кожної таблиці, визначеної у вузлі.

2) Створюємо відповідну обмежувальну рамку над створеними таблицями на основі квантора, застосованого до вузла.

3) Оновлюємо таблиці, щоб виразити предикати вибору, безпосередньо записуючи їх у новий рядок у таблиці.

4) Створюємо стрілки (спрямовані ребра) між атрибутами таблиць, для яких є предикат з'єднання.

Цікавою частиною є визначення напрямку стрілки на кроці (4): Якщо предикат з'єднання знаходиться між двома таблицями з двох різних вузлів LT ψ1 і ψ 2 відповідно, тоді ми визначаємо напрямок стрілки наступним чином:

1) Якщо вузол 1 є батьківським для вузла 2, то стрілка вказує від атрибута в T1 до атрибута в T2;

2) Інакше стрілка вказує від атрибута в T2 до атрибута в T1.

Нарешті, позначте стрілку на основі оператора з'єднання op, якщо op не дорівнює "=". Як ми показуємо, цей, здавалося б, довільний вибір напрямків стрілок дозволяє нам створювати діаграми, які

1) є однозначними щодо їх логічного значення,

2) є мінімально багатослівними,

3) мають природний порядок читання, що випливає зі стрілок.

2.4.4. Властивості діаграми

Дані діаграми розроблені для покращення розуміння існуючого SQL-запиту, а не для створення запиту з нуля або для його налагодження. Отже, вони надають користувачеві прості візуальні абстракції, які забезпечують такі властивості:

1) Існуючі метафори як відправна точка. Більшість користувачів баз даних раніше бачили діаграми реляційних схем. Простий кон'юнктивний запит не повинен візуалізуватися набагато інакше, ніж представлення схеми бази даних. Ми почали з UML та його знайомих елементів для моделювання даних, а потім додавали візуальні елементи лише за потреби. Кон'юнктивні запити, такі як рис. 2.14 а, мають найнижчу візуальну складність і представлені, як показано на рис. 2.13 а. На відміну від SQL-представлення

запиту, на наших діаграмах не потрібні псевдоніми. Зверніть увагу, як лінії між різними атрибутами візуалізують операцію з'єднання та її умови.

2) Мінімальна візуальна складність. Система використовує мінімальну кількість візуальних елементів (текст, лінія, стрілка, таблиця та обмежувальна рамка), як ми детальніше описуємо в розділі 3. Оскільки наша мета - інтерпретація запиту, а не специфікація запиту, ми можемо абстрагуватися від таких деталей, як обробка значень NULL, які не впливають на намір запиту.

3) Представлення логіки першого порядку. Методика адаптує візуальні метафори з діаграмних міркувань до реляційних схем. Рис. 2.14 b показує складніший вкладений запит; його діаграма показана на рис. 2.13 b. Переклади природною мовою цього запиту та кон'юнктивного запиту на рис. За мають приблизно однакову довжину. Аналогічно, рис. 2.13 b показує лише трохи збільшену візуальну складність (на 13% більше візуальних елементів) порівняно з рис. 2.13 a.

Навпаки, текст SQL набагато складніший (на 167% більше слів), що є відомим недоліком SQL, який був влучно названий поганою синтаксичною локальністю [2] і зазначений раніше [35]. Ми ще більше спростили візуальне представлення, додавши універсальний квантор (∀), конструкцію, яка не існує в SQL. Наприклад, представлення з рис. 2.13 b можна ще більше спростити до представлення на рис. 2.13 c, застосувавши перетворення. Це представлення тепер має лише на 7% більше видимих елементів, ніж кон'юнктивний запит з рис. 2.13 a.

4) Порядок читання. Ще однією концепцією, запозиченою з діаграмних міркувань, є порядок читання за замовчуванням [33]. Зверніть увагу на рис. 2.13 c, як стрілки між відношеннями відповідають перекладу природною мовою. Без стрілок не було б природного порядку для екзистенціальних та універсальних кванторів.

Висновки до розділу

Розділ присвячений дослідженню методів та алгоритмів, які застосовуються для побудови візуалізацій у роботі з базами даних, зокрема для графових даних. Визначено ключові проблеми та особливості, що виникають при розробці візуальних систем запитів і взаємодії з великими обсягами графових даних. Аналіз продемонстрував, що важливими аспектами є дизайн візуалізацій, питання масштабування для великих графових структур та вибір методів представлення даних. Недостатньо продуманий дизайн може значно знизити ефективність візуалізації, а проблема масштабування залишається однією з ключових перешкод у візуалізації великих графових структур, оскільки перевантаження інформацією ускладнює її сприйняття та аналіз.

Обговорено різні шаблони, що використовуються для створення інтуїтивних інтерфейсів користувача. Використання таких шаблонів полегшує навігацію в складних структурах даних та сприяє оптимізації процесу формулювання і виконання запитів до бази даних. Досліджено сучасні інструменти, які включають візуальні мови запитів, підсвічування синтаксису, а також можливості перекладу запитів природною мовою. Візуальні мови запитів спрощують процес формулювання запитів, що важливо для користувачів із недостатнім досвідом у написанні SQL-коду. Синтаксичне підсвічування полегшує сприйняття та редагування запитів, а переклади запитів природною мовою роблять взаємодію з базами даних доступнішою для користувачів, які не володіють спеціалізованою термінологією.

Таким чином, розробка ефективних візуалізацій для роботи з базами даних потребує застосування комплексного підходу, що враховує специфіку графових даних, необхідність масштабованості, а також потреби різних категорій користувачів.

РОЗДІЛ 3. ІМПЛЕМЕНТАЦІЯ МОДЕЛЕЙ ТА МЕТОДІВ ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ВЗАЄМОДІЇ КЛІЄНТІВ З БАЗАМИ ДАНИХ

3.1. Представлення моделі даних

Пропонована модель візуалізації дещо обмежує реляційну модель даних вкладеним універсальним відношенням: хоча це певною мірою обмежує виразну особливість порівняно з SQL, це дозволяє створити набагато ефективніший інтерфейс специфікації для великого класу складних запитів. Ми описуємо модель даних, можливості, які вона надає, та її обмеження.

Система використовує заданий користувачем півот для складання єдиної вкладеної універсальної таблиці з багатьох таблиць бази даних. Ми називаємо схему цієї таблиці деревом даних. Будь-яку реляційну схему можна перетворити на дерево даних: ми відображаємо схему реляційної бази даних у граф, де вузли представляють відношення, а ребра - зв'язки первинний-зовнішній ключ між відношеннями. Ми називаємо цей граф ключовим графом. Дерево даних - це кореневе дерево, що охоплює ключовий граф, з півотом як коренем. Починаючи з півота, ми додаємо його як корінь нашого дерева даних. Ми додаємо всі його атрибути, за винятком будь-яких зовнішніх ключів, як листя. Потім ми обходимо всі ребра, що виходять з півота в ключовому графі, в глибину. Для кожного відношення, яке ми додаємо як вузол до дерева даних, ми додаємо всі його атрибути як листя, за винятком зовнішніх ключів. У дереві даних кожен атрибут ідентифікується за його шляхом від півота. Ми заповнюємо вкладену універсальну таблицю, беручи об'єднання дочірнього відношення з батьківським відношенням у дереві даних.

Вкладена універсальна таблиця - це абстрактний вигляд поверх реляційної бази даних; нам не потрібно фізично реструктуризувати базу

даних або матеріалізувати кілька вкладених універсальних таблиць, по одній для кожного півота.

Хоча вкладена універсальна таблиця не є новою концепцією [13], вона значною мірою забута. Як випливає з назви, вона об'єднує універсальні відношення [14] з вкладеними моделями даних, такими як JSON або XML.

Це дає нам наступне:

- Природна ієрархія групування для великого класу квантифікованих запитів: Для цих запитів користувачі звільняються від двох громіздких кроків SQL: специфікації об'єднання та побудови груп за допомогою підзапитів.

Замкнута система: Запит - це просто розбиття кортежів вкладеної універсальної таблиці на відповіді та невідповіді. Якщо кортежі півота задовольняють всім нашим обмеженням, то вони є відповідями, інакше - невідповідями. Навпаки, невідповіді в традиційній реляційній моделі не є чітко визначеними і їх нелегко обчислити.

Запит у нашій мові називається деревом запитів. Клас запитів, що охоплюються деревами запитів, - це булеві кон'юнктивні квантифіковані запити над вкладеними відношеннями. Кожен квантифікований вираз складається з кон'юнкцій або тверджень над булевими функціями (це обмеження), і дозволені лише кон'юнкції квантифікованих виразів.

Приклади дерев запитів на рисунку 3.1 всебічно описують усі аспекти графічної мови запитів. Дерева запитів - це дерева даних, накладені обмеженнями. Кожен вузол відношення - це дерево запитів, яке відображає його кортежі або на відповіді, або на невідповіді. Порожнє дерево запитів - без обмежень - відображає всі його кортежі на відповіді. Дерева мають кон'юнктивний характер, вони беруть кон'юнкцію всіх обмежень на своїх нащадків - атрибути або кортежі відповідей з вкладеного дерева запитів.

На рисунку 3.1 а порожнє дерево запитів у takes відображає всі його кортежі на відповіді, оскільки воно не має обмежень. Для кожного студента дерево запитів у півоті student визначає, чи існує кортеж у вкладеному наборі

кортежів-відповідей у takes, де оцінка A. Ці студенти є відповідями. Решта студентів - невідповіді.

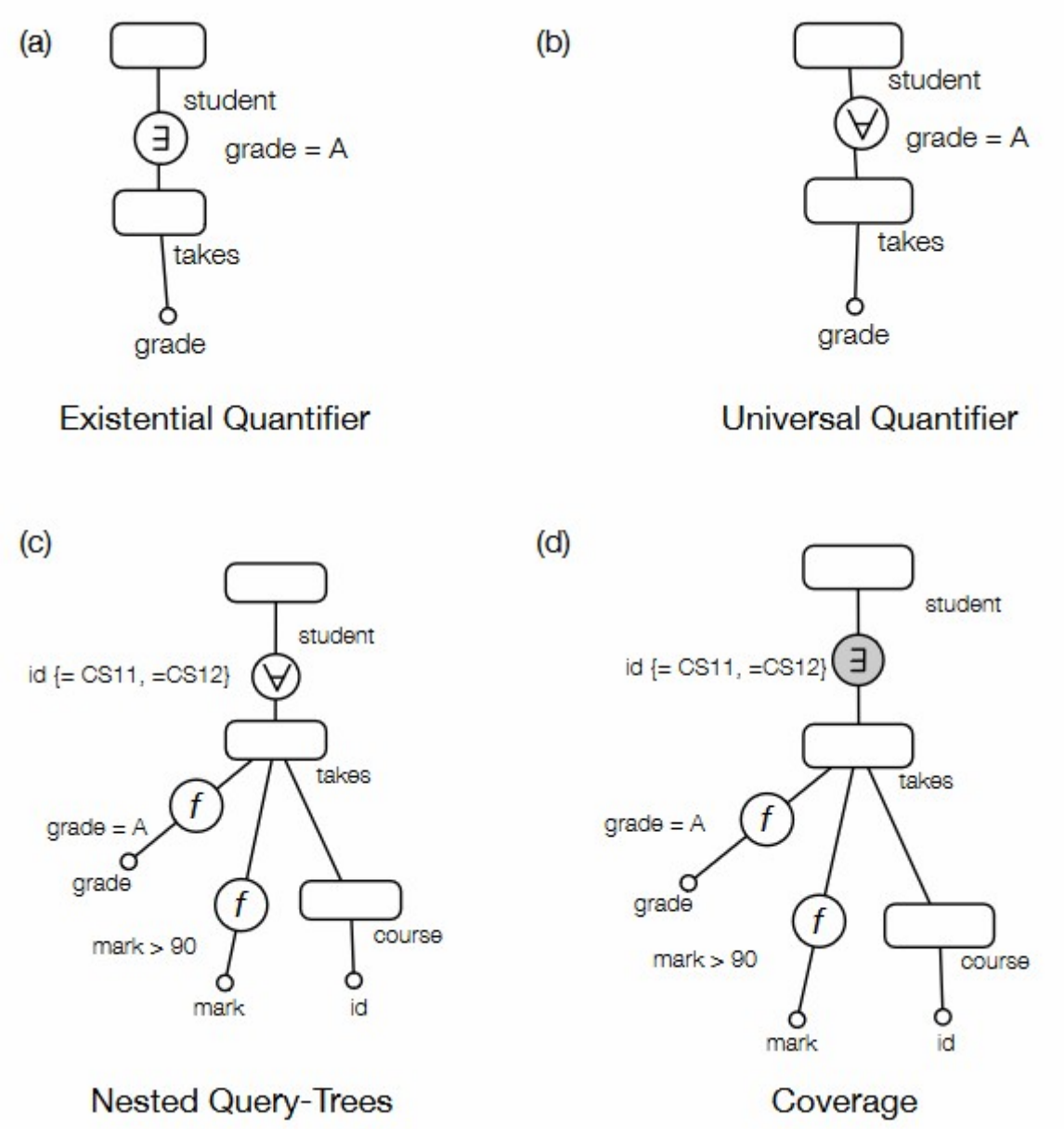

Рис. 3.1. Приклад дерев запитів

На рисунку 3.1 b дерево запитів у півоті student визначає, чи всі оцінки студента у вкладеному наборі кортежів-відповідей у takes є A.

Зверніть увагу, що коли вузол обмеження діє над набором кортежів, він має один із наступних символів квантора: ∀ для універсального квантора або ∃ для екзистенціального квантора. Якщо вузол обмеження діє над окремим

кортежем, а не над набором кортежів, йому не потрібен квантор, а натомість у ньому є символ f.

На рисунку 3.1 c дерево запитів у takes відображає кортежі, де оцінка A, а бал вище 90, на відповіді. Дерево запитів у півоті student тепер перевіряє, чи всі вкладені кортежі відповідей у takes є або CS11, або CS12, і жодних інших курсів. Універсально квантифіковане обмеження курсу тут розглядає лише вкладені кортежі відповідей takes, ігноруючи вкладені кортежі невідповідей.

Дерево запитів на рисунку 3.1 d ілюструє особливу функцію мови: покриття. Покриття дозволяє користувачам гарантувати, що всі пропозиції обмеження (якщо їх більше одного) є істинними для набору кортежів4. Дерево запитів на рисунку 3.1 d гарантує, що студент пройшов і CS11, і CS12 з оцінками A та балами вище 90. Покриття позначається затіненням вузла обмеження. Технічно покриття є надлишковою функцією мови, оскільки можна написати окремі обмеження для кожного курсу, і кон'юнктивний характер системи забезпечить покриття. Однак користувачі можуть концептуалізувати кілька пропозицій як одне самостійне обмеження. Крім того, при очищенні щільних візуалізацій, якщо ми генеруємо одне обмеження для кожної очищеної області або гліфа, а не список пропозицій для одного обмеження, ми можемо перевантажити дерево запитів.

Графічна мова запитів є достатньо виразною, щоб стисло та з синтаксичною локальністю описувати великий клас квантифікованих запитів: порівняйте дерева запитів на рисунку 3.1 a та 3.1 b з їх еквівалентними запитами SQL. Зверніть увагу, що для зміни семантики обмеження з екзистенціальної на універсальну потрібна лише одна зміна символу. Користувачі можуть налаштовувати властивості обмеження незалежно від інших обмежень і без створення нового дерева запитів. Оскільки кожне обмеження підтримує фіксований набір маніпуляцій (перемикання квантора, перемикання покриття або зміна положення), система візуально пропонує ці маніпуляції користувачам. Користувачам

більше не потрібно відчувати себе в глухому куті: вони можуть пробувати різні маніпуляції, доки не досягнуть бажаного запиту.

Більше того, мова гарантує, що для заданої колекції обмежень існує скінченний і доступний для пошуку простір можливих дерев запитів: це дозволяє виконувати автоматичне виправлення. Замість того, щоб безпосередньо маніпулювати деревом запитів для досягнення певного розбиття кортежів на відповіді або невідповіді, користувачі можуть просто вказати це розбиття, позначивши кортежі як відповіді або невідповіді, і система буде шукати в просторі дерев запитів, щоб знайти те, яке досягає необхідного розбиття.

Нарешті, порівняно з текстовим представленням запиту, дерева запитів краще відображають ієрархічну структуру вкладеної універсальної таблиці та краще ілюструють залежності між обмеженнями; вкладеність візуально незручна в лінійному представленні.

3.2. Представлення основних вимог до системи взаємодії з базами даних

Наша головна мета в цій роботі - полегшити взаємодію між користувачами-початківцями та базами даних. Це дозволить їм ефективніше використовувати мови запитів і швидше сприймати концепції.

В минулих розділах ми вже припустили, що додавання візуалізації буде корисним інструментом для новачків, оскільки це може допомогти їм у аналізі результатів і розумінні, чому виникають проблеми.

Існує багато типів додаткових удосконалень, які можна внести, щоб підтримати користувача в процесі надсилання запитів. Наприклад, ми могли б точно визначити місце в запиті, де виникають проблеми. Ми могли б додати взаємодію, щоб користувач міг змінити порядок графіка. Ми могли б дозволити користувачеві порівнювати запити. Ми могли б дозволити

користувачеві комбінувати (під)запити. З технічного боку ми могли б покращити взаємодію, підвищивши продуктивність системи.

У процесі проектування, який ми описуємо в цьому розділі, ми ще не враховуємо ці додаткові зміни. Ці системні умови можуть бути дуже корисними, але щоб виміряти їх вплив, ми повинні застосовувати лише одну значну зміну за раз. Тому ми залишаємо ці доповнення для наступних ітерацій системи .

Є дві сторони системи, яку ми тут розробляємо. По-перше, це загальна система, інтерфейс і його можливості. Потім, окремо, є дизайн графічного представлення запиту.

Ми почнемо з розподілу візуальних завдань інтерфейсу на категорії відповідно до топології завдань:

1. Аналізуйте (споживайте і виробляйте)

a) Відобразіть запит у вигляді графіка, натиснувши кнопку

b) Подайте таблицю результатів, що містить результат запиту

c) Виявлення зв’язків між таблицями в запиті

d) Отримайте відповіді з таблиць результатів

e) Отримувати нові запити з підзапитів

2. Пошук (повертає цілі)

a) Дослідіть модель даних

b) Переглядайте візуальне представлення, перетягуючи

c) Перегляньте вміст таблиць у даних

3. Запит (вміст цілей)

a) Визначте ознаки результату

b) Визначте помилки за допомогою повідомлень про помилки

c) Порівняйте змінні в результаті

d) Порівняйте два або більше представлень запиту

e) Порівняйте , чи поточний запит збігається з попереднім?

Окрім цих візуальних завдань, є деякі технічні аспекти, які ми не змогли виразити в таксономії. До них належать дії написання запиту.

Оскільки метою системи є підтримка новачків, більшість розширених команд SQL їм не потрібні. Таким чином, для цього дизайну ми зосередимося на базовій функціональності SQL, підзапитах і використанні заперечення.

3.2.1. Основні вимоги

Двома основними вимогами до системи є реалізація набору завдань користувача, перерахованих вище, і технічна настройка для виконання запитів SQL до бази даних. Крім того, є деякі неявні вимоги. По-перше, у нашому дослідженні в попередньому розділі ми виявили, що різні учні пишуть допоміжні та дослідницькі запити, щоб допомогти їм у вирішенні проблеми. Якщо ми будемо відображати лише один переклад графіка запиту, користувачеві доведеться запам’ятати свій допоміжний запит. Таким чином, було б корисно, якби користувачі могли «приклеювати» свої представлення графіків і створювати нові. Для цієї взаємодії потрібні додаткові кнопки.

Так само ми виявили, що студенти мають дуже різні способи роботи. Деякі користувачі працювали над проблемами структурним чином, тоді як інші мали більш випадковий підхід, випадково намагаючись виконати будь-який запит. Система повинна мати можливість підтримувати всіх своїх користувачів, незалежно від робочого порядку чи методу написання запитів. У цьому сценарії знову було б корисно зберігати або зберігати старі представлення графіків, щоб користувачі, повертаючись до певного запиту, могли отримати своє (останнє) представлення запиту.

Ми також виявили, що користувачі неодноразово запускали той самий запит. Оскільки результат не повинен змінюватися, коли запит є однаковим, така поведінка є марною тратою часу та ресурсів. Система має знати про цю практику та публікувати відгук, коли користувач використовує цей підхід.

Нарешті, ми виявили, що ці користувачі-початківці роблять багато помилок у процесі написання. Система повинна мати можливість інтуїтивно вказати, що сталася помилка. Бажано також вказати приблизне місце розташування помилки.

Окрім цих функціональних вимог, важливо підтримувати легкість системи. Оскільки ми розробляємо систему для початківців, найкраще було б, щоб вона не потребувала встановлення. Створення системи як веб-сторінки забезпечує високий доступ, оскільки її можна використовувати на більшості звичайних ПК.

3.2.2. Опис випадку використання

У цьому розділі ми обговоримо типовий крок користувача в системі.

1. Система містить курсор у текстовому полі.

2. Користувач вводить (під)запит у текстове поле (дозволяє створювати дані, робити запити та параметри дослідження)

3. Користувач виконує запит, натискаючи кнопку

4. У результаті виводиться таблиця результатів.

5. Користувач запитує візуальне представлення запиту, натиснувши кнопку.

6. Система відображає візуальне представлення.

7. Користувач повертається до кроку 2

Альтернативні курси:

1. Якщо користувач хоче звернутися за допомогою (у будь-який момент під час використання системи)

1.1 Користувач натискає кнопку «Довідка».

1.2 Система відобразить вікно з прикладом запиту та можливістю пошуку за ключовими словами

1.3 Користувач вводить ключове слово, про яке хоче дізнатися більше.

1.4 Система виводить інформацію за ключовим словом.

1.5 Користувач закриває вікно довідки.

1.6 Система повертається до кроку 1.

4. Якщо користувач викликав помилку.

4.1 Система виводить повідомлення про помилку.

4.2 Користувач знаходить помилку.

4.3 Користувач повертається до кроку 2.

7. Якщо користувач хоче написати два підзапити та об’єднати їх.

7.1 Користувач натискає кнопку, щоб закріпити візуальне представлення та відповідний запит.

7.2 Система відобразить порожнє поле для подання нового запиту.

7.3 Користувач вводить другий підзапит.

7.4 Користувач виконує запит, натискаючи кнопку.

7.5 У результаті виводиться таблиця результатів.

7.6 Користувач запитує візуальне представлення запиту, натиснувши кнопку.

7.7 Система відображає візуальне представлення.

7.8 Користувач натискає кнопку, щоб об’єднати два підзапити.

7.9 Система запропонує користувачу необхідний спосіб приєднання.

7.10 Користувач підтверджує спосіб приєднання.

7.11 Система створює нове поле, у якому відображається новий запит і його нове візуальне представлення.

7.12 Користувач може повернутися до кроку 2.

3.3. Дизайн, орієнтований на користувача

Дизайн програмного забезпечення, орієнтований на користувача (User-Centered Design, UCD) - це підхід до розробки, де потреби, бажання та обмеження кінцевих користувачів є головним пріоритетом на кожному етапі процесу. UCD - це не просто набір правил, а філософія розробки, яка ставить користувача в центр уваги. Застосування UCD дозволяє створювати продукти, які дійсно відповідають потребам користувачів та допомагають їм досягати своїх цілей.

Основні принципи UCD:

- Ітеративний дизайн: Процес розробки є ітеративним, тобто продукт поступово вдосконалюється на основі отриманого зворотного зв'язку.

- Фокус на користувачеві: Всі рішення щодо дизайну приймаються з урахуванням потреб, цілей та контексту використання продукту цільовою аудиторією.

- Тестування з користувачами: Прототипи та готові версії продукту тестуються на реальних користувачах, щоб отримати зворотній зв'язок та виявити потенційні проблеми.

- Доступність: Продукт має бути доступним для всіх користувачів, незалежно від їх фізичних можливостей чи рівня досвіду.

Після встановлення технічних характеристик дизайну ми тепер поглянемо на людські аспекти дизайну.

Процес проектування в цьому проекті керується принципами орієнтованої на користувача парадигми проектування. Це означає, що з самого початку процесу створення прототипу зацікавлені сторони беруть участь у проектуванні та керують дизайном через аналіз їхніх вимог. Одним із ключових понять у цьому контексті є ментальні моделі. Функціональні вимоги та технічна реалізація повинні бути пов’язані представленням, яке може зрозуміти будь-який користувач.

Ми застосовуємо цю парадигму, записуючи взаємодію з прототипами, а також різноманітні заходи, такі як анкети, щоб відгуки користувачів могли безпосередньо застосовуватися до прототипів. Це ітеративний процес, коли взаємодії впливають на прототипи, а прототипи впливають на взаємодії. Цей цикл буде застосовано до різних компонентів розробки інструменту.

На практиці це означає, що ми будемо запускати користувальницькі тести для всіх етапів розробки інтерфейсу та взаємодії. У другій половині цього розділу ми описуємо проведені експерименти, їх результати та вплив цих результатів на наступні прототипи.

Ми починаємо цей процес розробки з ключового елемента нашого нового інструменту: візуального представлення запиту. Мета цієї візуалізації полягає в тому, щоб краще зрозуміти текст SQL, щоб користувачі робили менше помилок.

Хоча були проведені деякі дослідження щодо візуалізації даних графіків візуальному представленню запитів не приділено багато уваги. Оскільки наша мета — спростити взаємодію, ми хочемо бачити користувачі інтуїтивно зрозумілим представленням запиту. При такому підході проектування цього елемента можна робити спільно з майбутніми користувачами.

Вимагати від користувачів інтуїтивно візуалізувати всі деталі, доступні нам на мові SQL, було б занадто великим завданням, тому ми вирішили зосередитися на запереченнях і підзапитах. Це два компоненти, з якими ми показали, що студенти стикаються з труднощами.

Окрім відгуків користувачів, ми також хотіли базувати наш дизайн на теорії. Таким чином, у наступному розділі ми спочатку проаналізуємо деякі існуючі дослідження візуалізації заперечень і підзапитів.

3.4. Розробка прототипу інтерфейсу системи ефективної взаємодії користувачів з базами даних

Оскільки реального інтерфейсу Jupyter немає, ми в першу чергу зосереджуємося на функціональності. Дизайн керується золотими правилами де ми зосереджуємося на симетрії, послідовності та чистоті, і цей перший дизайн не містить жодних кольорів.

Перше, що варто розглянути щодо дизайну, — які функціональні можливості включити. Як у випадку з Jupyter: нам потрібні основи, такі як текстова область і область для результатів. Кнопки потрібні для всіх функцій, наприклад для виконання запитів. Ми не хочемо використовувати для нього ярлик, як це робиться в Jupyter.

Щоб спростити процес запиту, ми додаємо три функціональні можливості: детальну функцію довідки, відображення моделі даних і візуалізацію запиту. Крім того, ми додаємо функцію тимчасового зберігання запиту, що може пришвидшити робочий процес. Це робиться клацанням

великого пальця в області візуалізації. На даний момент ми ще не впевнені щодо розташування кнопок. Таким чином, ми беремо до уваги закон Фіттса і розташовуємо кнопки якомога ближче до їх застосовної області. Область історії розміщена у верхній частині екрана, щоб відповідати ментальним моделям на вкладках. Дизайн виконаний максимально симетричним і його представлено на рисунку 3.2.

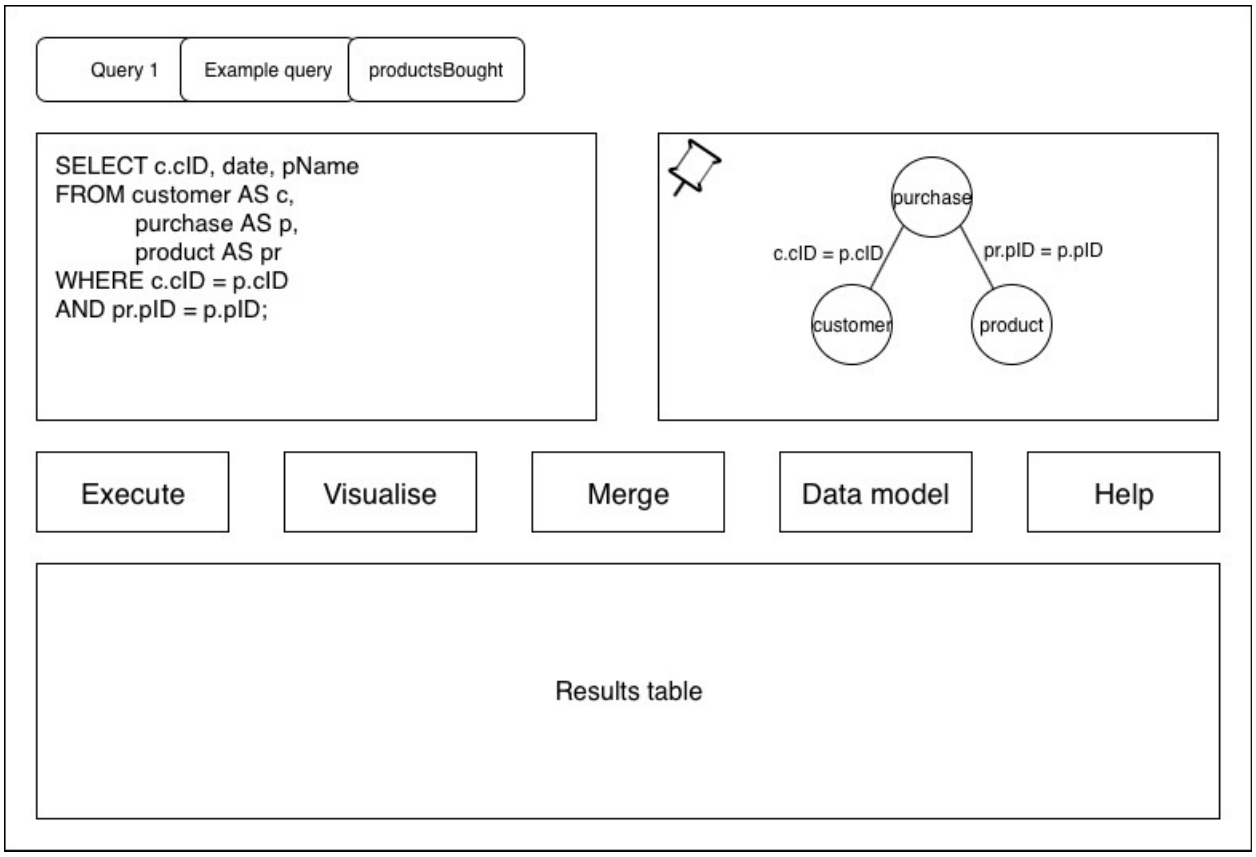

Рис. 3.2. Перший прототип

На основі доопрацювань попереднього розділу ми зробили редизайн інтерфейсу. Другий прототип представлено на рисунку 3.3. Деякі аспекти залишилися незмінними, а інші кардинально змінилися. Елементи, які можна розпізнати з попереднього дизайну, включають текстову область і область візуалізації. Ми обговоримо цей новий дизайн в окремих частинах, починаючи з наших основних особливостей.

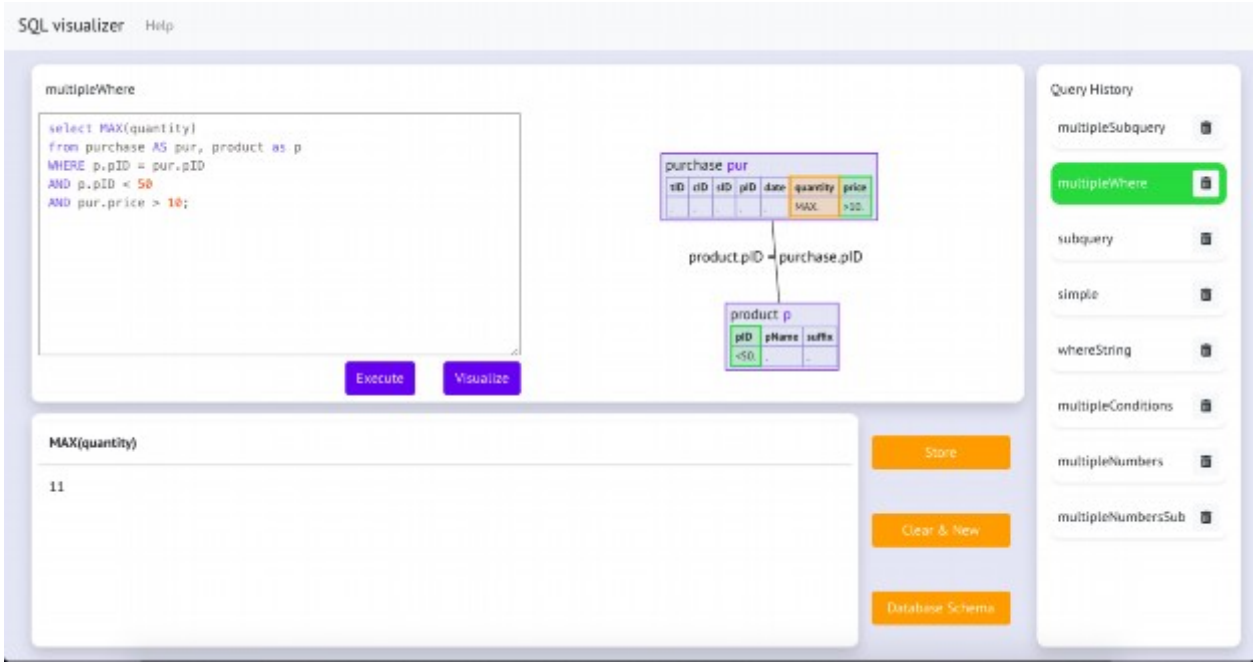

Рис. 3.3. Другий, повністю функціональний прототип

Для зручності доступу ми вирішили реалізувати систему у вигляді веб-додатку. Це означає, що система містить інтерфейс, бек-енд і API для обміну даними між ними. Вибір мови програмування залежав від можливостей візуалізації, які вона надала. Однією з найвідоміших і найпотужніших бібліотек візуалізації є d3.js. JavaScript також пропонує потужну структуру Bootstrap, яка значно спрощує створення інтерфейсу. Тому ми обрали JavaScript як нашу мову програмування.

Однак із міркувань безпеки JavaScript не має доступу до локальної файлової системи. Оскільки для бази даних потрібні дані, які можуть зберігатися локально, а не на сервері, серверній частині потрібна інша мова програмування. Добре відомою мовою для реалізації серверів і API є Python. Кодування RESTful API підтримувалося через Flask , мікровеб-платформу на Python.

Завдяки цій комбінації інструментів ми змогли реалізувати систему відповідно до дизайну, описаного в наступних розділах. Одним із трьох наших основних внесків є візуальне представлення запиту. Як можна побачити на рисунку 3.3, розміщення подібне до попереднього прототипу.

3.5. Представлення елементів візуалізації запитів

Для реалізації візуалізації ми знову зосередилися на підзапитах і запереченні. Поверх структури графіка як представлення ми додали візуальні індикатори для різних інших компонентів запиту.

Це представлення простого запиту. Він включає одну таблицю і, отже, має один вузол і немає посилань. Мітка вузла — це ім'я таблиці бази даних, яку він представляє, у даному випадку таблиця «Клієнт». Якщо запит містить псевдонім для таблиці, він також буде представлений у вузлі дещо іншим кольором тексту.

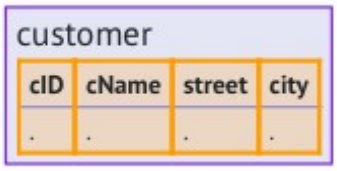

Рис. 3.4. Візуальне представлення таблиці

Кожен вузол можна розгорнути, клацнувши його. Це не тільки розкриє подробиці про схему бази даних, але й візуально представить інші компоненти запиту. У цьому прикладі ми бачимо помаранчеві виділення для стовпців, до яких ви застосували вибір у запиті. У цьому випадку запит містив SELECT *. Виділено лише вибрані елементи верхнього рівня, а ті, що знаходяться в підзапитах, – ні.

Рис. 3.5. Візуальне представлення вузла

Кожен запит може отримати доступ до кількох таблиць. У цьому прикладі показано використання двох таблиць. Умова об'єднання, застосована до таблиць, представлена позначеним посиланням. Вузли, які

мають рамку, є елементами, які використовуються у підзапиті. Ця межа показує, що підзапит позитивного типу, заперечення не використовується.

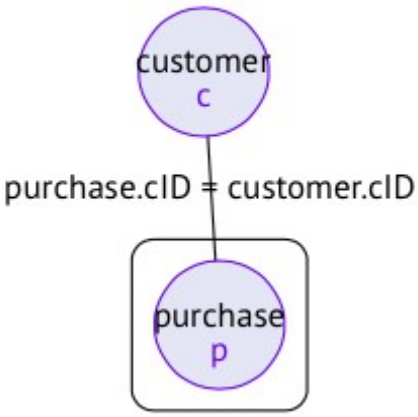

Рис. 3.6. Візуальне представлення об’єднання таблиць

Підзапити можуть бути вкладеними для багатьох рівнів. Цей запит показує два вкладені підзапити. Пунктирна рамка, на відміну від попереднього прикладу, вказує на заперечення підзапиту, який вона містить.

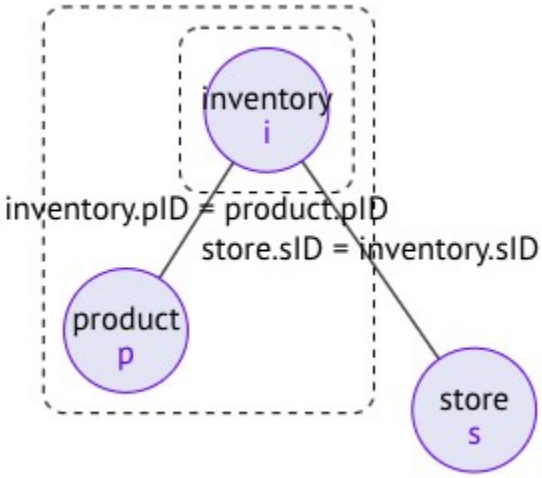

Рис. 3.7. Візуальне представлення під запитів декількох рівнів

Окрім підсвічування вибраних стовпців, система також може відображати всі умови, застосовані в запиті. Це робиться зеленим кольором, як у цьому прикладі (рис. 3.8). Значення умови також записуються в таблицях.

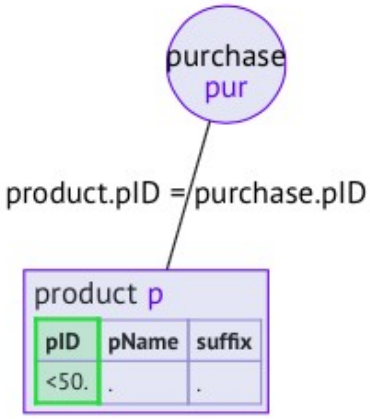

Рис. 3.8. Підсвічування умов вибраних в запиті

Іноді стовпець таблиці може мати як вибір, так і умову. У цьому випадку ми вкладаємо рамки та відображаємо обидва кольори

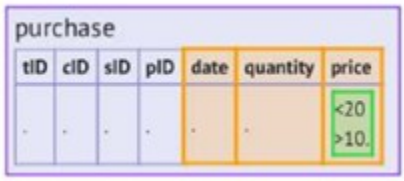

Рис. 3.9. Підсвічування вибору і умови в запиті

Також, запит може містити агрегацію. У такому випадку ми відображаємо метод агрегації в стовпці вибору, до якого він застосовується.

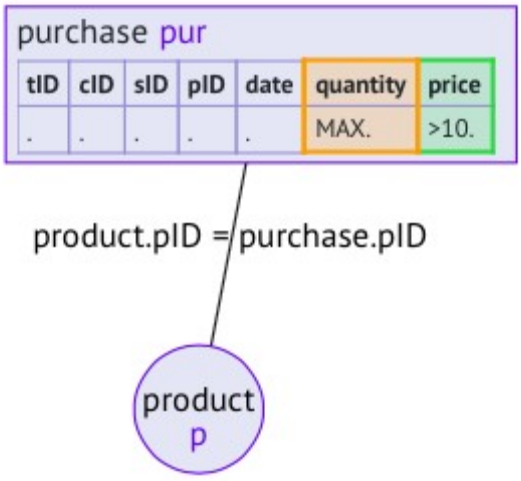

Рис. 3.10. Підсвічування операції агрегації

Візуалізацією також може керувати користувач за допомогою перетягування. Для візуалізації ми повторно використовуємо кольори фону та інтерфейсу.

Як і в першому прототипі, ми реалізували кнопку, яка керує спливаючим вікном, що містить модель даних. Ми вирішили представити схему в її технічному вигляді, який є відкритим текстом з підкресленням для первинних ключів. Кожна назва таблиці написана жирним шрифтом, а всі її стовпці відображаються нижче. Для кожного стовпця ми відображаємо назву та тип у дужках. Уривок показаний на рисунку 3.11.

Рис. 3.11. Фрагмент схеми для таблиці інвентаризації

Іншим варіантом було б відобразити схему більш наочним способом, наприклад, через ER-діаграму. Таке представлення показуватиме посилання для будь-якого первинного та зовнішнього ключів. Однак правильне відображення цих рядків може бути складним технічно та може заплутати користувачів, особливо для великих баз даних із багатьма таблицями.

Щоб допомогти користувачам орієнтуватися в інструменті, було створено базове меню довідки. Він містить опис інструменту, а також велику допомогу щодо написання запитів SQL.

Спливаюче вікно довідки містило чотири частини: базове вступне слово, опис усіх елементів інтерфейсу, підручник із написання запитів SQL та покажчик усіх можливих команд SQL. Уся ця інформація разом має допомогти користувачеві при написанні запитів.

Реалізація відрізнялася від першого прототипу розташуванням кнопок. Перша ідея полягала в тому, щоб запропонувати допомогу з написання

поблизу області запиту, а загальну допомогу — далі. У цьому прототипі ми об’єднали два елементи. Перший аргумент на користь цього рішення полягає в тому, щоб довідку не розбивати, щоб все було в одному місці, і тому його було легко знайти. По-друге, реалізація повідомлень про помилки ще не була достатньо зрозумілою, щоб пов’язувати їх із довідкою щодо написання запитів. У першому прототипі ми вже пропонували зберігання запитів. Там ми розмістили їх у верхній частині сторінки. Відгуки під час першого користувацького тесту свідчать про те, що горизонтальне прокручування може бути не оптимальним у цьому випадку. Тому в цьому новому прототипі сховище запитів було розміщено в правій частині сторінки, дозволяючи вертикальне прокручування. Термін «Історія запитів» був запропонований одним із наших користувачів, і оскільки він може показувати всі запити, які ви запускали раніше, це здавалося підходящою назвою.

Збереження запиту в історію відбувається через кнопку магазину. Усі нові запити автоматично нумеруються, щоб мати унікальні імена. Це ім’я використовується для представлення запиту в списку історії. Користувачі також можуть редагувати цю назву, щоб зробити її більш описовою. Після того, як вони натиснуть кнопку Store, у списку історії з’явиться новий блок із цією назвою запиту. Цей блок також містить маленьку кнопку зі значком кошика. Це дозволяє користувачам видалити запит із сховища. Це має бути інтуїтивно зрозумілим, оскільки ментальна модель користувачів має відповідати сміттєвому баку та функції викидання чогось. Відкриття запиту здійснюється натисканням на блок із назвою відповідного запиту. Це також автоматично виконує запит. Ця дія також має бути інтуїтивно зрозумілою, оскільки вона відповідає подібним системам.

Висновки до розділу

Отже, в цьому розділі представлено процес імплементації моделей та методів для покращення ефективності взаємодії користувачів з базами даних.

Окреслено ключові аспекти, що впливають на продуктивність, зручність і доступність системи, а також розглянуто основні етапи створення інтерфейсу, орієнтованого на користувача.

Обґрунтовано структуру моделі даних, яка забезпечує оптимальне представлення і доступ до даних для досягнення високої швидкості виконання запитів. Правильно розроблена модель сприяє підвищенню ефективності взаємодії системи з базою даних і створює надійну основу для реалізації функціоналу, орієнтованого на користувача. Сформульовано основні вимоги, які враховують як функціональні, так і нефункціональні аспекти. Це включає потребу в простоті інтерфейсу, високій продуктивності, гнучкості для різних типів запитів та масштабованості системи. Також наведено опис випадків використання, що забезпечує краще розуміння сценаріїв роботи користувачів із системою та їхніх очікувань.

Дизайн, орієнтований на користувача: акцент зроблено на створенні інтуїтивного інтерфейсу, який враховує потреби різних категорій користувачів. Основні принципи дизайну включають мінімізацію кроків для виконання запитів, забезпечення доступності необхідних функцій та підтримку зворотного зв'язку, що дозволяє користувачам краще розуміти результати своїх дій.

Було створено прототип інтерфейсу, який забезпечує ефективну та зручну взаємодію з базами даних. Прототип дозволяє оцінити функціональність системи, протестувати зручність її використання та врахувати побажання користувачів ще на етапі розробки, що підвищує ймовірність успішної імплементації.

Загалом, результати цього розділу демонструють важливість продуманого підходу до проектування та імплементації систем взаємодії з базами даних. Завдяки цьому підходу досягається підвищення ефективності роботи користувачів і вдосконалення їхнього досвіду взаємодії із системою.

ВИСНОВКИ

В магістерській роботі розглянуто моделі та методи підвищення ефективності взаємодії клієнтів з базами даних. Пропонована магістерська робота присвячена вивченню процесів візуалізації графових даних та впровадженню моделей і методів для покращення взаємодії користувачів з базами даних. Основні результати та висновки роботи полягають у наступному.

Аналіз особливостей візуалізації графових баз даних: у першому розділі розглянуто сучасні тенденції в базах даних, зокрема графові бази, що мають високий потенціал для обробки складних зв’язків між даними. Досліджено історію та особливості графових баз, таких як NoSQL, GraphDB, RDF, SPARQL, та мови графових запитів, що дозволяють гнучко працювати з взаємопов’язаними даними. Це забезпечує більш природне моделювання складних об'єктів і їхніх зв'язків, а також спрощує управління даними, що мають ієрархічну структуру.

Моделі та алгоритми для візуалізації баз даних: другий розділ детально висвітлює особливості та виклики візуалізації графових даних, включаючи дизайн, проблему масштабування та методи представлення великих масивів інформації. Було розглянуто різні шаблони візуалізації запитів, методи підсвічування синтаксису, використання природної мови для запитів та методологію відображення вкладеності SQL-запитів. Це забезпечує зручний доступ до даних і спрощує їхню інтерпретацію навіть при роботі з великими обсягами інформації, що є критичним для сучасних графових систем.

Імплементація методів підвищення ефективності взаємодії з базами даних: третій розділ присвячений реалізації моделей для підвищення зручності користувацької взаємодії. Сформульовано основні вимоги до систем, орієнтованих на користувача, які передбачають інтуїтивний дизайн інтерфейсу, доступність функцій та простоту роботи з даними. Розроблений прототип інтерфейсу забезпечує ефективну взаємодію, орієнтовану на

потреби користувачів, завдяки продуманій візуалізації, що знижує складність навігації та полегшує роботу з запитами.

Загалом, результати роботи показують, що правильний вибір моделей візуалізації та методів побудови інтерфейсу суттєво впливає на ефективність і зручність роботи з базами даних. Використання графових баз даних з відповідними методами візуалізації сприяє швидкому аналізу великих масивів взаємопов'язаних даних, забезпечуючи простий та інтуїтивний доступ до необхідної інформації.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Elmasri, R., & Navathe, S. B. (2016). Fundamentals of Database Systems. Pearson.
2. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). Database System Concepts. McGraw-Hill Education.
3. Connolly, T., & Begg, C. (2015). Database Systems: A Practical Approach to Design, Implementation, and Management. Pearson.
4. Ramakrishnan, R., & Gehrke, J. (2003). Database Management Systems. McGraw-Hill.
5. Stonebraker, M., & Hellerstein, J. M. (2005). Readings in Database Systems. MIT Press.
6. Garcia-Molina, H., Ullman, J. D., & Widom, J. (2009). Database Systems: The Complete Book. Pearson.
7. Kimball, R., & Ross, M. (2013). The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling. Wiley.
8. Redmond, E., & Wilson, J. R. (2012). Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement. Pragmatic Bookshelf.
9. Date, C. J. (2011). Database Design and Relational Theory: Normal Forms and All That Jazz. O'Reilly Media.
10. Ambler, S. W., & Sadalage, P. J. (2011). Refactoring Databases: Evolutionary Database Design. Pearson.
11. Chen, P. (1976). The Entity-Relationship Model—Toward a Unified View of Data. ACM Transactions on Database Systems.
12. McGovern, J., Tyagi, S., Stevens, M., & Mathew, S. (2003). Java Web Services Architecture. Morgan Kaufmann.
13. Barik, T. (2015, oct). Improving error notication comprehension in IDEs by supporting developer self-explanations. In 2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) (pp. 293{294). IEEE.

14. Bhowmick, S. S. (2014). DB x HCI: towards bridging the chasm between graph data management and HCI. In Database and expert systems applications (pp. 1{11).
15. Berenson, H., Bernstein, P., Gray, J., Melton, J., O'Neil, E., & O'Neil, P. (1995). A Critique of ANSI SQL Isolation Levels. ACM SIGMOD Record.
16. Shasha, D., & Bonnet, P. (2003). Database Tuning: Principles, Experiments, and Troubleshooting Techniques. Morgan Kaufmann.
17. Hoffer, J. A., Ramesh, V., & Topi, H. (2016). Modern Database Management. Pearson.
18. Lahiri, T., & Widom, J. (2002). Disk Striping in Large Database Systems. VLDB Journal.
19. Lee, M. L., & Hsu, W. (2005). SQL Query Optimization and Execution Time Reduction Techniques. IEEE Transactions on Database Systems.
20. Li, X., & Özsu, M. T. (2002). Database Management Principles and Applications. Morgan Kaufmann.
21. Zou, Y., & Godfrey, M. (2007). Analyzing Clustering Techniques for Database Schema Evolution. ACM SIGSOFT.
22. Jain, A., & Ross, A. (2004). Multimodal Biometrics for Database Security and Privacy Protection. IEEE Transactions on Database Systems.
23. Ahmed, F., & Kalashnikov, D. (2008). Improving Database Query Performance using Pre-fetching and Caching Strategies. VLDB Journal.
24. Piatetsky-Shapiro, G., & Frawley, W. (1991). Knowledge Discovery in Databases. MIT Press.
25. Adam, N., & Wortmann, J. (1989). Security-Control Methods for Statistical Databases. ACM Computing Surveys.
26. Tian, X., & Shang, Y. (2006). Adaptive Query Optimization in Distributed Database Systems. IEEE Transactions on Database Systems.
27. Papadimitriou, C. H. (1979). The Complexity of Database Management Systems. ACM SIGMOD Record.

28. Liu, W., & Stonebraker, M. (2003). Distributed Database Systems: Concepts and Design Approaches. Springer.
29. Bright, M., & Hurson, A. (1992). A Taxonomy of Multidatabase Systems. IEEE Transactions on Database Systems.
30. Baker, M., & Mahadev, S. (2015). Efficient Replication Techniques for High-Performance Databases. ACM Transactions on Database Systems.
31. Carey, M. J., & Schneider, D. (1989). Parallelism and Optimization in Database Management Systems. ACM Transactions on Database Systems.
32. Yu, C., & Meng, W. (1998). Principles of Database Query Processing and Optimization. Kluwer Academic Publishers.
33. Chaudhuri, S., & Narasayya, V. (2007). Indexing in Database Management Systems: Theory and Applications. ACM SIGMOD.
34. Franklin, M. J., & Amsaleg, L. (1996). Query Processing in Data Warehousing Environments. VLDB Journal.
35. Gupta, H., & Mumick, I. (1995). Materialized View Selection and Maintenance for Database Performance Optimization. ACM Transactions on Database Systems.
36. Abbott, K., Bogart, C. & Walkingshaw, E. (2015, oct). Programs for people: What we can learn from lab protocols. In 2015 ieee symposium on visual languages and human-centric computing (vl/hcc) (pp. 203{211). IEEE.
37. Abras, C., Maloney-Krichmar, D. & Preece, J. (2004). User-centered design. Bainbridge, W. Encyclopedia of Human-Computer Interaction. Thousand Oaks: Sage Publications,
38. Agosto, D. E. (2002). Bounded rationality and satiscing in young people's web-based decision making. Journal of the American Society for Information Science and Technology
39. Agresta, T. (2015). The hype around graph databases and why it matters. Retrieved from http://www.forbes.com/sites/ciocentral/2015/04/06/the-hype-around-graph-databases-and-why-it-matters/ 12

40. Allen, R. B. (1982, jul). Cognitive factors in human interaction with computers. Behaviour & Information Technology,
41. Angelaccio, M., Catarci, T. & Santucci, G. (1990). QBD*: A graphical query language with recursion. IEEE Transactions on Software Engineering
42. Angles, R., Arenas, M., Barcelo, P., Hogan, A., Reutter, J. L. & Vrgo c, D. (2016, oct). Foundations of Modern Graph Query Languages. ArXiv e-prints.
43. Catarci, T. (2009). Visual Query Language. Springer Encyclopedia of Database Systems, 3399 - 3405.
44. Angles, R. & Gutierrez, C. (2005). Querying RDF data from a graph database perspective. In European semantic web conference (pp. 346{360).
45. Angles, R. & Gutierrez, C. (2008). Survey of graph database models. ACM Computing Surveys (CSUR)
46. Lightstone, S., Teorey, T., & Nadeau, T. (2007). Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More. Morgan Kaufmann.
47. Li, L., & Xie, X. (2008). Improving Database Transaction Performance through Concurrency Control Mechanisms. ACM Transactions on Database Systems.
48. Elmasri, R., & Larson, J. (1985). A Model for Federated Database Systems. ACM Computing Surveys.
49. Atkinson, M. P., Bancilhon, F., DeWitt, D. J., Dittrich, K. R., Maier, D. & Zdonik, S. B. (1989). The Object-Oriented Database System Manifesto. In Deductive and object-oriented databases (p. 223-240).
50. Biswas, P. & Robinson, P. (2009). Modelling perception using image processing algorithms. British Computer Society.
51. Bohner, S. A. & Mohan, S. (2009, oct). Model-Based Engineering of Software: Three Productivity Perspectives. In 2009 33rd Annual IEEE Software Engineering Workshop (pp. 35 - 44). IEEE.