МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра інженерії програмного забезпечення

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

Олексій Горський
"____"_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
МАГІСТРА

Тема: "Інформаційна технологія знаходження актуальних баз даних"

Виконавець: Сніжко Ерік Романович

Керівник: к.т.н., доцент Семко Олексій Вікторович

Нормоконтролер: к.ф.-м.н., ст.викл. Гололобов Дмитро Олександрович

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра Інженерії програмного забезпечення
Освітній ступінь магістр
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Програмне забезпечення систем»

ЗАТВЕРДЖУЮ
Олексій Горський
"____"_____2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи студента
Сніжка Еріка Романовича

1. Тема проекту: Інформаційна технологія знаходження актуальних баз даних.

затверджена наказом ректора від 10.09.2023 р. № 2582/ст

2. Термін виконання проекту: з 10.10.2023 р. до 31.12.2023 р.

3. Вихідні данні до проекту: дослідити технології знаходження актуальних баз даних на прикладі вебу.

4. Зміст пояснювальної записки:

1. Аналіз предметної області.
2. Інформаційна технологія знаходження актуальних баз даних.
3. Пошук релевантних БД у вебі.

5. Перелік обов'язкового графічного матеріалу:

1. Загальна архітектура QA-систем.
2. Архітектура пошукової системи.
3. Архітектура мультибазової БД.
4. Архітектура з посередниками/обгортками.

6. Календарний план-графік

№ пор	Завдання	Термін виконання	Відмітка про виконання
1.	Ознайомлення з постановкою задачі та вивчення літератури Написання 1 розділу, представлення керівнику		
2.	Попередній друк 1 розділу та допоміжних сторінок (черновик) - титульної, завдання, графіка, реферат, список скорочень, зміст, вступ, список джерел. 1-ий нормо-контроль.		
3.	Написання 2 розділу, представлення керівнику		
4.	Написання 3 розділу, представлення керівнику		
5.	Загальне редагування та друк пояснювальної записки, графічного матеріалу		
6.	Проходження нормо-контролю, перепліт пояснювальної записки.		
7.	Розробка тексту доповіді. Оформлення графічного матеріалу для презентації		
8.	Отримання відгуку керівника, рецензії.		
9.	Підготовка матеріалів для передачі секретарю ДЕК (ПЗ, ГМ, CD-R з		

	електронними копіями ПЗ, ГМ, презентації, відгук керівника, рецензія, довідка про успішність, 1 папка, 1 конверт)		

7. Дата видачі завдання 10.09.2023

Керівник: _____ к.т.н., доцент Семко О.В.

Завдання прийняв до виконання: _____ Сніжко Е.Р.

Дата

РЕФЕРАТ

Пояснювальна записка до дипломного проекту “Інформаційна технологія знаходження актуальних баз даних ”: 60 с., 13 рис., 1 табл., 13 інформаційних джерел.

БАЗИ ДАНИХ, ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ, МОДЕЛЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

Об’єкт розробки – засіб знаходження актуальних баз даних.

Мета роботи – розробка інформаційної технології знаходження актуальних баз даних.

ABSTRACT

Explanatory note to the diploma project "Information technology for finding relevant databases": 60 pages, 13 figures, 1 table, 13 information sources.

DATABASES, INFORMATION TECHNOLOGY, SOFTWARE MODELING.

The object of development is a means of finding current databases.

The purpose of the work is to develop information technology for finding up-to-date databases.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ ……

ВСТУП

……

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ……..

1.1. Поняття систем управління мультибазами даних

1.2. Веб як слабо структуровані дані ……..

1.3. Мови веб запитів …….

1.4. Системи «Питання-відповідь» ……..

1.5. Пошук та запити до прихованого вебу …….

1.6. Метапошук ……

1.7. Категоризація баз даних ……..

РОЗДІЛ 2 ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ЗНАХОДЖЕННЯ АКТУАЛЬНИХ БАЗ ДАНИХ ……..

2.1. Архітектура обробки мультибазового запиту …….

2.2. Методи оптимізації та виконання запитів ……

РОЗДІЛ 3 ПОШУК РЕЛЕВАНТНИХ БД У ВЕБІ ……..….

3.1. Управління веб-даними ……..

3.2 Управління веб-графом …….

3.3. Пошук у вебі …….

3.4. Індексування ……...

3.5. Ранжування та аналіз посилань …….

3.6. Пошук за ключовими словами …….

3.7. Запити до вебу
…….
ВИСНОВКИ ……...
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ …….

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

EF – Entity Framework

DB – Database

EDMX – Entity Framework Data Model Extension

ETL – Extract, Transform, Load

MVC – Model – View - Controller

API – Application Programming Interface

REST – Representational State Transfer

LINQ – Language Integrated Query

SVG – Scalable Vector Graphics

GUI – Graphical User Interface

ВСТУП

Користувачі звикли майже миттєво отримувати релевантні результати від пошукових програм. Щоб створювати такі програми, ви повинні опанувати механізми пошуку. Однак для багатьох розробників тема ранжування релевантності залишається досить цікавою для дослідження.

Дана робота присвячена створенню інформаційної технології знаходження актуальних баз даних і показує, що механізм пошуку - це всього лише програмований двигун. Програмувати релевантність, як підключити вторинні джерела даних, класифікатори, організувати аналіз тексту та забезпечити персоналізацію пошуку є досить важливими питаннями. До прийомів налагодження релевантності належать:

– Застосування можливостей механізму пошуку для вирішення нагальних проблем

– Направлення користувачів до потрібних результатів за допомогою інтерфейсу користувача

– Систематичний підхід до релевантності

– Бізнес-культура, спрямована на покращення релевантності пошуку

РОЗДІЛ 1.
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Поняття систем управління мультибазами даних

У системах управління мультибазами даних (СУМБД) або мультибазових системах окремі СУБД повністю автономні і поняття кооперації не існує; вони можуть навіть не знати про існування один одного і не мати жодних засобів взаємодії. Звичайно, нас цікавлять передусім розподілені СУМБД, тобто такі, в яких складові СУБД розміщені в різних вузлах. Багато з розглянутих нами питань є спільними для одновузлових та розподілених СУМБД; у таких випадках ми просто говоритимемо «СУМБД» без вказівки природи. У сучасній літературі найчастіше зустрічається термін інтеграція баз даних. Зауважимо, проте, що терміну «мультибаза» у різних джерелах приписується різний зміст. У цій книзі ми розумітимемо під ним сказане вище, але слід пам'ятати, що таке трактування може не збігатися з літературою, що зустрічається.

Відмінності рівні автономності СУМБД і розподілених СУБД знаходять свій відбиток й у архітектурних моделях. Фундаментальне різницю пов'язані з визначенням глобальної концептуальної схеми. У разі логічно інтегрованих розподілених СУБД глобальна концептуальна схема визначає концептуальне подання всієї бази даних, тоді як у разі СУМБД вона представляє лише набір деяких локальних баз даних, які локальна СУБД готова надати загальне користування. Окремі СУБД можуть надати загальний доступ лише до частини своїх даних. Тому визначення глобальної бази даних у СУМБД та розподілених СУБД різняться. У другий випадок глобальна база даних збігається з об'єднанням локальних, а першому вона є лише підмножиною (може бути, власним) цього об'єднання. У СУМБД глобальна концептуальна схема (іноді звана опосередкованою схемою) визначається шляхом інтеграції локальних концептуальних схем (чи їх частин).

Компонентна архітектурна модель розподіленої СУМБД істотно відрізняється від моделі розподіленої СУБД тим, що кожен повноцінний вузол СУБД керує своєю

базою даних. СУМБД надає шар програмного забезпечення, що працює поверх цих окремих СУБД та пропонує користувачам засоби доступу до різних баз даних (рис. 1.1). Зазначимо, що у розподіленій СУМБД шар СУМБД може працювати у кількох вузлах або перебувати в одному вузлі, який і надає відповідні служби. Також зауважимо, що, з погляду окремих СУБД, шар СУМБД є просто ще одним додатком, який надсилає запити та отримує відповіді.

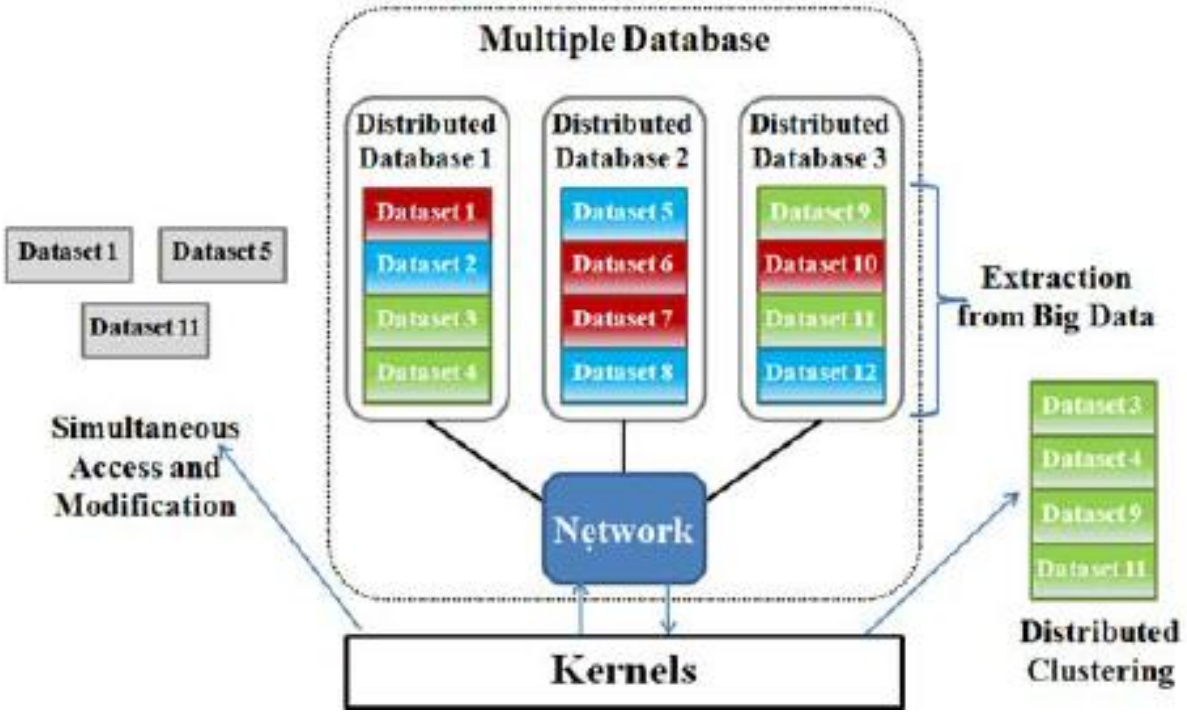

Рис. 1.1 Архітектура мультибазової БД

Популярною реалізацією архітектури СУМБД є підхід на основі посередників та обгорток (рис. 1.2). Посередником називається "програмний модуль, який використовує закодовані знання про деякі множини або підмножини даних для породження інформації, призначеної вищим рівням додатків". Таким чином, кожен посередник виконує певну функцію з чітко визначеним інтерфейсом. За такої архітектури кожен модуль, що належить шару СУМБД на рис. 1.2, реалізований у вигляді посередника. Оскільки посередники може бути надбудовані з інших посередниками, можна організувати багаторівневу реалізацію. Рівень посередників реалізує кортикостероїдів. Саме на цьому рівні обробляються запити користувача до ГКС і надається функціональність СУМБД.

Посередники зазвичай працюють із загальною моделлю даних та мовою визначення інтерфейсу. Щоб упоратися з потенційною гетерогенністю СУБД-джерел, створюються обгортки, завдання яких – надати відображення між СУБД-джерелом та посередником. Наприклад, якщо СУБД джерело реляційне, а посередник об'єктно-орієнтований, необхідні відображення реалізуються обгортками. Точна роль та функція посередників залежить від реалізації. У деяких випадках посередники не роблять нічого, крім трансляції, тоді вони називаються "тонкими". Але іноді посередники беруть він частина функціональності обробки запитів. Набір посередників можна як проміжний рівень, що надає служби, надбудовані над системами-джерелами. ПЗ проміжного рівня – тема, яка активно розроблялася у минулому десятилітті, коли було створено досить розвинені системи, які пропонують просунуті кошти розробки розподілених додатків. Посередники, які описуються, – лише частина функціональності таких систем.

Рис. 1.2. Архітектура з посередниками/обгортками

1.2. Веб як слабо структуровані дані

Один із підходів до опитування веб-даних полягає в тому, щоб розглядати веб як збори слабо структурованих даних. Було розроблено моделі та мови для опитування таких даних. Спочатку вони призначалися не для роботи з веб-даними, а для задоволення вимог до зростаючих наборів даних, які не мають такої суворої схеми, як у реляційному випадку. Але оскільки подібні характеристики притаманні і веб-даним, виявилося, що ці моделі та мови застосовні і в цій галузі. Ми продемонструємо цей підхід на прикладі моделі OEM та мови Lorel; інші системи, наприклад UnQL, схожі.

OEM (Object Exchange Model – модель обміну об'єктами) – це самоописувана слабо структурована модель даних. Під самоописом розуміється, що кожен об'єкт визначає схему, якою він дотримується.

Об'єкт OEM визначається як четвірка 〈label, type, value, oid〉, де label – рядок символів, що описує, що саме представляє об'єкт, type – тип значення об'єкта, value – значення, а oid – ідентифікатор, який відрізняє даний об'єкт від решти. Тип об'єкта може набувати значення atomic – тоді об'єкт називається атомарним, або complex – тоді об'єкт називається складовим. Атомарний об'єкт має значення примітивного типу, наприклад, ціле, речове або рядок, а складовий включає інші об'єкти, які самі можуть бути атомарними або складовими. Значення складового об'єкта є набір ідентифікаторів об'єктів.

Розглянемо бібліографічну базу даних, що складається із набору документів. Витяг з такої бази показано на рис. 1.3. У кожному рядку є один об'єкт OEM, а відступи виявляють структуру об'єкта. Наприклад, у другому рядку <doc, complex, &o3, &o6, &o7, &o20, &o21, &o2> визначено об'єкт з міткою doc, типом complex, ідентифікатором &o2, що складається з об'єктів з ідентифікаторами &o3, &o6, &o7, &o20 та &o2

```
<bib, complex, {&o2, &o22, &034}, &o1>
   <doc, complex, {&o3, &o6, &o7, &o20, &o22}, &o2>
      <authors, complex, {&o4, &o5}, &o3>
         <author, string, "M. Tamer Ozsu", &o4>
         <author, string, "Patrick Valduriez", &o5>
      <title, string, "Principles of Distributed ...", &o6>
      <chapters, complex, {&o8, &o11, &o14, &o9}, &o7>
         <chapter, complex, {&o9, &o10}, &o8>
            <heading, string, "...", &o9>
            <body, string, "...", &o10>
            ...
         <chapter, complex, {&o18, &o19}, &9>
            <heading, string, "...", &o18>
            <body, string, "...", &o19>
      <what, string, "Book", &o20>
      <price, float, 98.50, &o21>
   <doc, complex, {&o23, &o25, &o26, &o27, &o28}, &o22>
      <authors, complex, {&o24, &o4}, &o23>
         <author, string, "Yingying Tao", &o24>
      <title, string, "Mining data streams ...", &o25>
      <venue, string, "CIKM", &o26>
      <year, integer, 2009, &o27>
      <sections, complex, {&o29, &o30, &o31, &o32, &o33}, &28>
         <section, string, "...", &o29>
           ...
         <section, string, "...", &o33>
   <doc, complex, {&o16,&o9,&o7,&o18,&o19,&o20,&o21},&o34>
      <author, string, "Anthony Bonato", &o35>
      <title, string, "A Course on the Web Graph", &o36>
      <what, string, "Book", &o20>
      <ISBN, string, "TK5105.888.B667", &o37>
      <chapters, complex, {&o39, &o42, &o45}, &o38>
         <chapter, complex, {&o40, &o41}, &o39>
            <heading, string, "...", &o40>
            <body, string, "...", &o41>
         <chapter, complex, {&o43, &o44}, &o42>
            <heading, string, "...", &o43>
            <body, string, "...", &o44>
         <chapter, complex, {&o46, &o47}, &45>
            <heading, string, "...", &o46>
            <body, string, "...", &o47>
      <publisher, string, "AMS", &o48>
```

Рис. 1.3. Приклад специфікації в OEM

Ця база даних містить три документи: (&o2, &o22, &o34); перший і третій – книги, другий – стаття. У двох книг (і навіть у книг та статті) є спільні риси, але є й

відмінності. Наприклад, для документа &o2 задана ціна, а для &o34 – ISBN та відомості про видавництво, які для &o2 відсутні.

Як було зазначено, дані в OEM самоописувані, т. е. кожен об'єкт описує себе з допомогою типу і мітки. Легко бачити, що дані можна подати у вигляді графа з поміченими вершинами, в якому вершини відповідають об'єктам OEM, а ребра ведуть від об'єктів до подоб'єктів. Однак у літературі прийнято моделювати дані графом із позначеними ребрами: якщо об'єкт oj є подоб'єктом oi, то мітка oj присвоюється ребру, що з'єднує oi з oj, а мітками вершин служать ідентифікатори об'єктів. На рис. 1.4 зображено подання графа з поміченими вершинами та ребрами для бази даних OEM з прикладу вище. Зазвичай, кожна термінальна вершина (з якої не виходить жодного ребра) містить також значення об'єкта. Але, щоб спростити картинку, ми опустили значення.

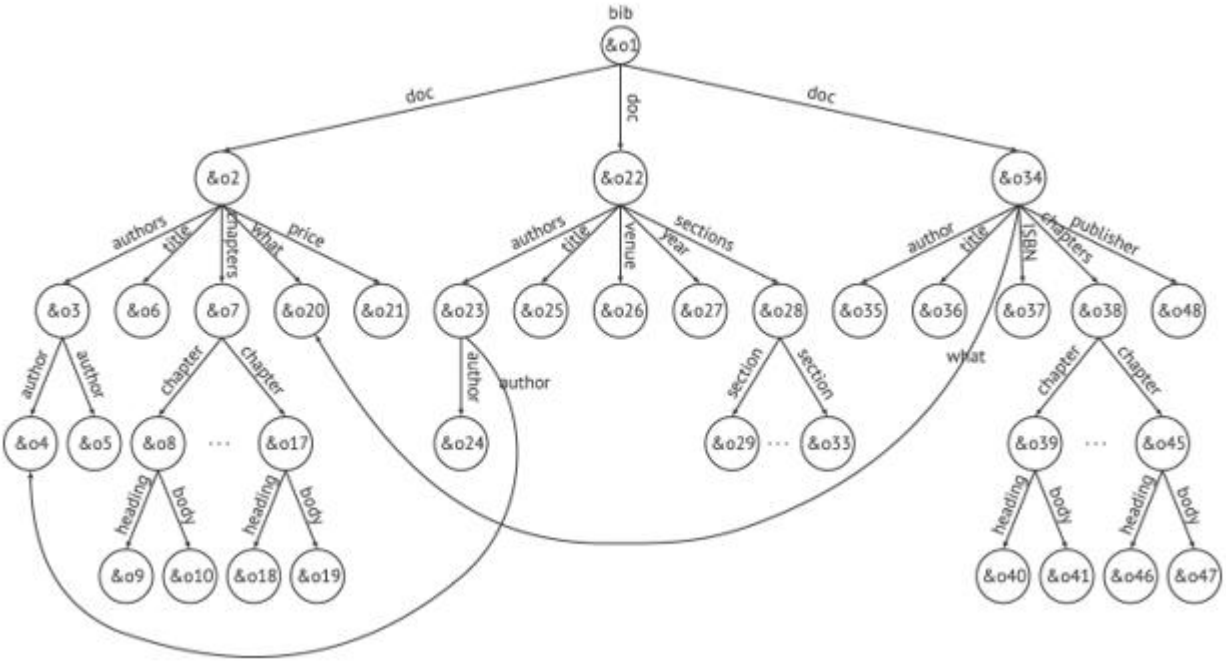

Рис. 1.4. Граф, який відповідає базі даних OEM

Ідея слабкої структурованості добре лягає на моделювання веб-даних, оскільки допускає подання у вигляді графа. До того ж вона застосовна до даних, які мають якусь структуру, але не таку жорстку, повну та регулярну, як у традиційних базах

даних. Користувачам, які запитують дані, не потрібно знати про повну структуру, тому і висловлювати запит не слід вимагати наявності такої інформації. Для запитів слабо структурованих даних розроблено кілька мов. Ми обмежимося розглядом лише одного з них, Lorel, інші схожі на нього.

У Lorel прийнято знайому структуру SELECT-FROM-WHERE, але у фразах SELECT, FROM і WHERE допускаються шляхові вирази. Шляховий вираз взагалі є основною конструкцією у запитах на Lorel. У простій формі це послідовність міток, що починається ім'ям об'єкта або змінною, що позначає об'єкт. Наприклад, bib.doc.title – шляховий вираз, який інтерпретується наступним чином: почати з bib, йти по ребру з позначкою doc, а потім по ребру з позначкою title. На рис. 12.5 є три шляхи, що відповідають цьому виразу: (i) &o1.doc:&o2.title:&o6, (ii) &o1.doc:&o22. title:&o25, (iii) &o1.doc:&o34.title:&o36. Кожен із них називається шляхом до даних. У Lorel шляхові вирази можуть бути більш складними регулярними виразами, за ім'ям об'єкта або змінною може йти не тільки мітка, але і більш загальний вираз, що включає кон'юнкцію, диз'юнкцію (|), ітерацію (? означає 0 або 1 повторення, + – 1 або більше,

∗ – 0 або більше) та метасимволи (#).

Нижче наведено приклади допустимих дорожніх виразів мовою Lorel:

a) bib.doc(.authors)?.author : почати з bib, йти по ребру doc і по ребру author, між якими може бути необов'язкове ребро authors;

b) bib.doc.#.author : почати з bib, йти по ребру doc, потім може зустрітися скільки завгодно багато ребер з будь-якими мітками (описуються метасимволом #), але закінчити слід проходом по ребру author;

c) bib.doc.% price : почати з bib, йти по ребру doc, потім по ребру, мет-

ка якого містить рядок «price», якому можуть передувати якісь символи. □

Нижче наведено запити Lorel, в яких використовуються деякі шляхові вирази з прикладу вище.

a) Знайти назви документів, автором яких є Patrick Valduriez.

SELECT D.title

FROM bib.doc D

WHERE bib.doc(.authors)?.author = "Patrick Valduriez"

У цьому запиті фраза FROM обмежує область перегляду документами (doc), а фраза SELECT задає вузли, досяжні з документів шляхом проходу по ребру з міткою title. Можна було б записати предикат WHERE у вигляді

D(.authors)?.author = "Patrick Valduriez".

b) Знайти авторів усіх книг, ціна яких менша за 100 доларів.

SELECT D(.authors)?.author

FROM bib.doc D

WHERE D.what = "Books" AND D.price < 100 □

Слабо структурований підхід до моделювання та опитування веб-даних простий та гнучкий. Він пропонує природний спосіб опису ієрархічної структури веб-об'єктів, а отже, певною мірою підтримує структуру посилань веб-сторінок. Але в нього є недоліки. Модель даних надто проста – вона не має на увазі якусь внутрішню структуру запису (кожна вершина є простою сутністю) і не підтримує впорядкованість, оскільки на безлічі вершин OEM-графа немає жодного відношення порядку. Крім того, підтримка посилань рудиментарна, тому що ні модель, ні мова не розрізняють посилання різних типів, хоча вони можуть виражати як відношення «є частиною» між об'єктами, так і зв'язки між різними сутностями, що відповідають вершинам. Однак ці відмінності неможливо змоделювати, як неможливо сформулювати запит із зазначенням типу посилання.

Зрештою, структура графа може стати дуже складною, що ускладнює формулювання запитів. Хоча Lorel пропонує низку засобів (зокрема, метасимволи), щоб спростити запити, наведені вище, показують, що користувачу все одно потрібно

знати загальну структуру слабко структурованих даних. OEM-графи великих баз даних складні, і користувачам важко виписувати дорожні вирази. Таким чином, потрібно якось «стиснути» граф, щоб вийшло опис типу схеми розумного розміру, який міг би допомогти при написанні запитів. Для цього запропоновано конструкцію, яка називається DataGuide (гід за даними). DataGuide є графом, у якому кожен шлях, присутній у відповідному OEM-графі, зустрічається тільки один раз. Він динамічний тому, що коли OEM-граф змінюється, відповідний йому Data- Guide теж оновлюється. Таким чином, ми отримуємо лаконічний та точний конспект слабо структурованої бази даних, яку можна використовувати як полегшену схему, корисну для вивчення структури бази даних, формулювання запитів, зберігання статистичної інформації та оптимізації запитів.

1.3. Мови веб-запитів

Мета підходів із цієї категорії – безпосередньо опитувати характеристики веб-даних, приділяючи особливу увагу належній обробці посилань. Відправною точкою є прагнення подолати недоліки пошуку за ключовими словами, запропонувавши абстракції, які правильно вловлюють як структуру документів (як у слабко структурованих підходах), так і зовнішні посилання. Вони поєднують запити до вмісту (наприклад, за ключовими словами) та структурні запити (наприклад, дорожні вирази).

Спеціально для роботи з веб-даними створено кілька мов, які можна віднести до першого та другого поколінь. Мови першого покоління моделюють Інтернет як набір взаємопов'язаних атомарних об'єктів. Отже, на них можна виразити запити, які шукають об'єкти в Інтернеті за структурою посилань та текстовим вмістом, але не запити, в яких враховується внутрішня структура цих об'єктів. Мови другого покоління моделюють веб як набір взаємопов'язаних структурних об'єктів, а отже, дозволяють висловлювати запити до внутрішньої структури документа як у слабко структурованих мовах. До мов першого покоління належать WebSQL, W3QL та

WebLog, а до мов другого покоління – WebOQL та StruQL. Ми продемонструємо спільні ідеї на прикладі мов WebSQL та WebOQL.

WebSQL – одна з ранніх мов, в якій поєднується пошук та навігація. Він безпосередньо запитує дані, що зберігаються у веб-документах (зазвичай у форматі HTML), які містять певний контент і можуть включати посилання на інші сторінки або інші об'єкти (наприклад, PDF-файли або зображення). Посилання розглядаються як повноправні об'єкти, причому виділяється кілька типів посилань, які ми обговоримо нижче. Як і раніше, структуру можна представити графом, але WebSQL запам'ятовує інформацію про веб-об'єкти у двох віртуальних відносинах:

DOCUMENT(URL, TITLE, TEXT, TYPE, LENGTH, MODIF) LINK(BASE, HREF, LABEL)

Щодо DOCUMENT зберігається інформація про кожен веб-документ. Поле URL ідентифікує веб-об'єкт і є первинним ключем відношення, TITLE містить назву веб-сторінки, TEXT – її текстовий вміст, TYPE – тип веб-об'єкта (HTML-документ, зображення тощо), LENGTH – довжину, а MODIF – дату останньої модифікації об'єкта. Усі атрибути, крім URL, можуть набувати значення null. Відносно LINK запам'ятовується інформація про посилання. Поле BASE містить URL того HTML-документа, в якому посилання зустрічається, HREF - URL документа, на який веде посилання, а LABEL - мітку посилання.

Мова запитів WebSQL нагадує SQL, доповнений шляховими виразами, виразнішими, ніж у Lorel; зокрема, вони розрізняють типи посилань:

a) внутрішнє посилання в межах одного документа (#>);

b) локальне посилання між документами на одному сервері (->);

c) глобальне посилання на документ, розташований на іншому сервері (=>);

d) порожній шлях (=).

Ці типи посилань утворюють абетку шляхових виразів. З-поміж них і звичайних конструкторів регулярних виразів можна формувати різні шляхи.

Нижче наведено приклади дорожніх виразів на мові WebSQL:

a) -> | =>: шлях одиничної довжини, локальний чи глобальний;

b) ->*: локальний шлях довільної довжини;

c) =>->*: те саме, що й вище, але на інших серверах;

d) (-> |=>)*: досяжна частина Інтернету.

Крім дорожніх виразів у запитах, WebSQL дозволяє уточнювати область пошуку у фразі FROM:

FROM Relation SUCH THAT domain-condition

де domain-condition може бути дорожнім виразом або визначати пошук за текстом, якщо використовується ключове слово MENTIONS, або включати умову рівності атрибуту (згаданого у фразі SELECT) веб-об'єкту. Зрозуміло, кожному згаданому у запиті відношенню може бути зіставлена змінна, як у стандартному SQL. Нижче наведені приклади запитів демонструють можливості WebSQL.

Нижче наведено приклади застосування WebSQL:

a) у першому прикладі ми просто шукаємо всі документи, що містять слово

"hypertext". При цьому демонструється застосування ключового слова

MENTIONS для уточнення області пошуку:

SELECT D.URL, D.TITLE

FROM DOCUMENT D

SUCH THAT D MENTIONS "hypertext"

WHERE D.TYPE = "text/html"

b) у другому прикладі демонструються два способи завдання області пошуку, а також пошук з урахуванням посилань. Потрібно знайти всі посилання на аплети в документах на тему Java:

SELECT A.LABEL, A.HREF

FROM DOCUMENT D SUCH THAT D MENTIONS "Java"

ANCHOR A SUCH THAT BASE=X

WHERE A.LABEL = "applet"

c) у третьому прикладі демонструється використання різних типів посилань. Потрібно знайти документи, у назві яких зустрічається рядок «database», досяжні з домашньої сторінки цифрової бібліотеки ACM шляхом довжиною не більше 2, що містить тільки локальні посилання:

SELECT D.URL, D.TITLE

FROM DOCUMENT D SUCH THAT "http://www.acm.org/dl"=|->|->-> D

WHERE D.TITLE CONTAINS "database"

d) і в останньому прикладі демонструється одночасне завдання умов пошуку за вмістом та структурою. Потрібно знайти все документи зі згадкою «Computer Science», а також усі документи, пов'язані з ними шляхами довжини не більше 2, що містять лише локальні посилання:

SELECT D1.URL, D1.TITLE, D2.URL, D2.TITLE

FROM DOCUMENT D1 SUCH THAT D1 MENTIONS "Computer Science",

DOCUMENT D2 SUCH THAT D1=|->|->-> D2 □

За допомогою WebSQL можна опитувати веб-дані на основі посилань та текстового вмісту документів, але не можна подавати запити до структури документа. Це наслідок моделі даних, де веб розглядається як набір атомарних об'єктів.

Мови другого покоління, у т. ч. WebOQL, усувають цей недолік, моделюючи веб як граф об'єктів, наділених структурою. У певному сенсі вони поєднують деякі можливості слабко структурованих підходів із засобами мов першого покоління.

Головна структура даних WebOQL називається гіпердеревом, це впорядковане дерево з двома типами помічених ребер: внутрішніми та зовнішніми. Внутрішнє ребро представляє внутрішню структуру веб-документа, а зовнішнє – посилання між об'єктами (т. е. гіперпосилання). Кожне ребро позначене записом, що включає низку атрибутів (полів). Для зовнішнього ребра цей запис повинен містити атрибут URL, крім того, зовнішнє ребро має закінчуватися листовим вузлом гіпердерева (що не має нащадків).

Повернімося до прикладу вище і припустимо, що моделюються не документи в бібліографії, а документи, що стосуються управління даними в Інтернеті. Можливе (часткове) гіпердерево для цього прикладу показано на рис. 1.5. Ми внесли одну зміну, щоб можна було пред'являти деякі з наведених нижче запитів: додали до кожного документа реферат.

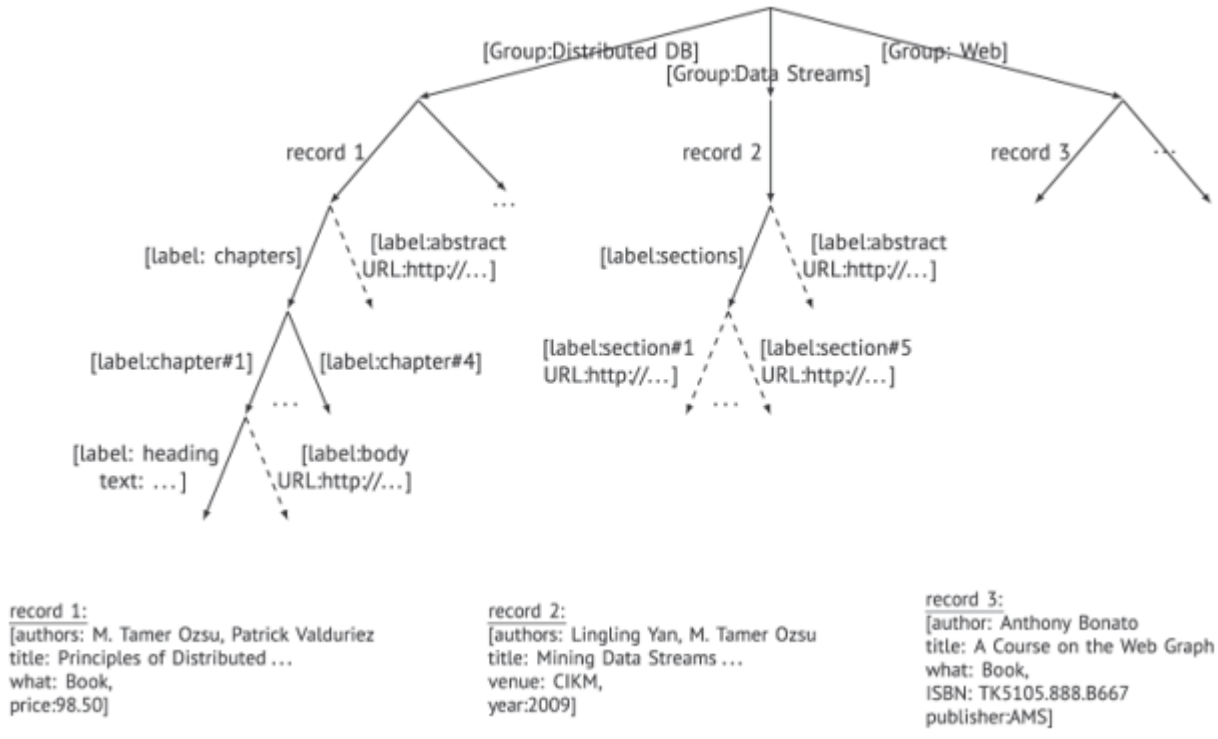

Рис. 1.5. Приклад гіпердерева

На рис. 1.5 документи спочатку групуються за темами, як видно із записів, приєднаних до ребрів, що виходять із кореневої вершини. У цьому поданні внутрішні посилання є суцільними лініями, а зовнішні – штриховими. Нагадаємо, що в OEM (рис. 1.3) ребра репрезентують як атрибути (наприклад, автора), так і структуру документа (наприклад, розділ). У моделі WebOQL атрибути зберігаються у записах, асоційованих з ребрами, а структуру документа представляють внутрішні ребра. □

На основі цієї моделі WebOQL визначає оператори над деревами.

Штрих: повертає перше піддерево свого аргументу (позначається ¢).

Загляд: витягує поле із запису, який позначає перше ребро, що виходить із документа. Наприклад, якщо x вказує на корінь піддерева, досяжного по ребру "Groups = Distributed DB", то x.authors виділяє поле

«M. Tamer Ozsu, Patrick Valduriez».

Конструювання: будує дерево з поміченими ребрами із записом, утвореним аргументами (позначається []).

Припустимо, що дерево на рис. 1.6a є результатом запиту (назвемо Q1). Тоді вираз [label: "Papers by Ozsu" / Q1] будує дерево, показане на рис. 1.6b.

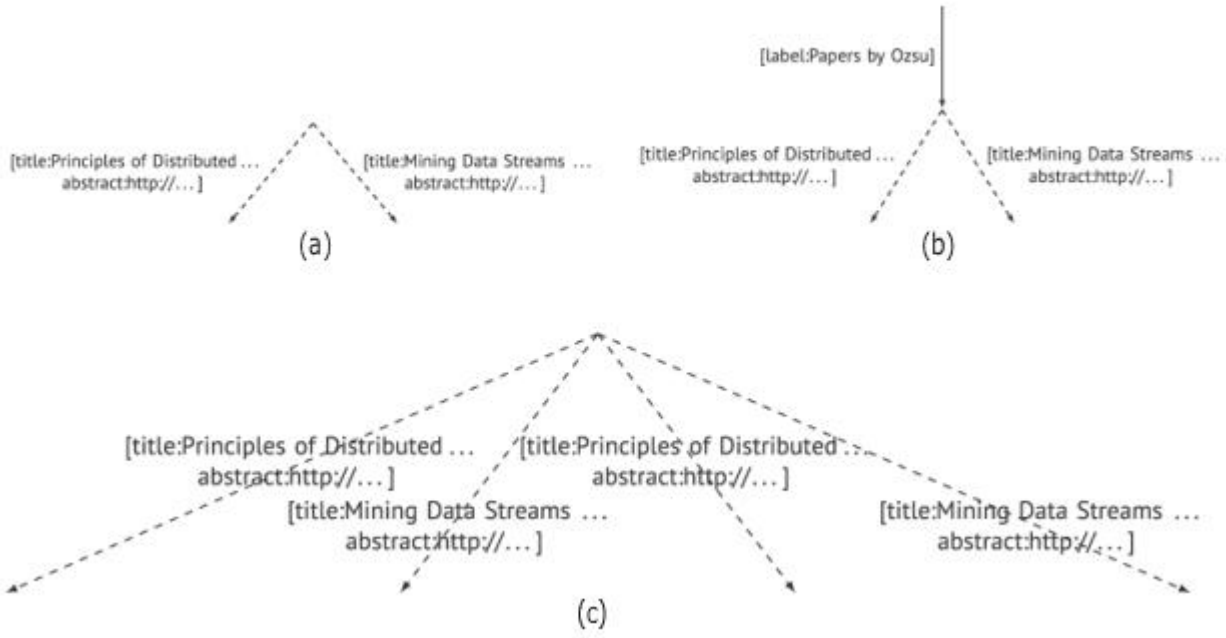

Рис. 1.6 Приклади операторів конструювання та конкатенації

Конкатенація: поєднує два дерева (позначається +).

Знову припустимо, що дерево на рис. 1.6a є результатом запиту Q1, тоді Q1+Q2 породжує дерево на рис. 1.6c.

Голова: повертає перше просте дерево дерева (позначається &). Простим деревом дерева t називається дерево, що складається з одного ребра, за яким слідує (можливо, порожнє) дерево, що росте з кінцевої вершини

цього ребра.

Хвіст: повертає всі дерева дерева, крім першого простого (позначається!).

Додатково у WebOQL є оператор зіставлення зі зразком (позначається ~), лівим операндом якого є рядок, а правим – зразок. Оскільки мова підтримує лише один тип даних – рядок, цей оператор є важливим.

WebOQL – функціональна мова, тому комбінуванням цих операторів можна створювати складні запити. Крім того, всі оператори можна занурювати у звичайні запити на кшталт SQL (або OQL), як показано в наступному прикладі.

Позначимо dbDocuments документи у базі даних, зображеної на рис. 1.5. Тоді наступний запит знаходить назви та реферати всіх документів з автором «Ozsu» та повертає результат, показаний на рис. 1.6a.

SELECT y.title, y¢.URL

FROM x IN dbDocuments, y IN x¢

WHERE y.authors ~ "Ozsu"

Цей запит інтерпретується так. Змінна x пробігає всі прості дерева dbDocuments, і для кожного значення x змінна y перебирає прості дерева єдиного

піддерева x. Запит заглядає в запис ребра, і якщо поле authors зіставляється з рядком «Ozsu» (для порівняння використовується оператор ~), то конструюється дерево, мітка якого містить атрибут title із запису, на який вказує y, та атрибут URL піддерева.

У мовах веб-запитів, що обговорюються в цьому розділі, прийнято більш розвинену модель даних, ніж у слабко структурованих підходах. Ця модель вловлює як структуру документа, так і зв'язки між різними документами. Крім того, мови враховують різну семантику ребер. Крім того, як бачили на прикладах WebOQL, результатом запиту може бути нова структура. Однак для формулювання запитів все одно потрібні знання про структуру графа.

1.4. Системи «Питання-відповідь»

У цьому розділі ми обговоримо цікавий і незвичайний (з точки зору баз даних) підхід до доступу до веб-даних: питання-відповіді (QA) системи. Такі системи приймають питання природною мовою, які потім аналізуються з метою визначити, що мав на увазі користувач. Після цього здійснюється пошук для отримання відповіді.

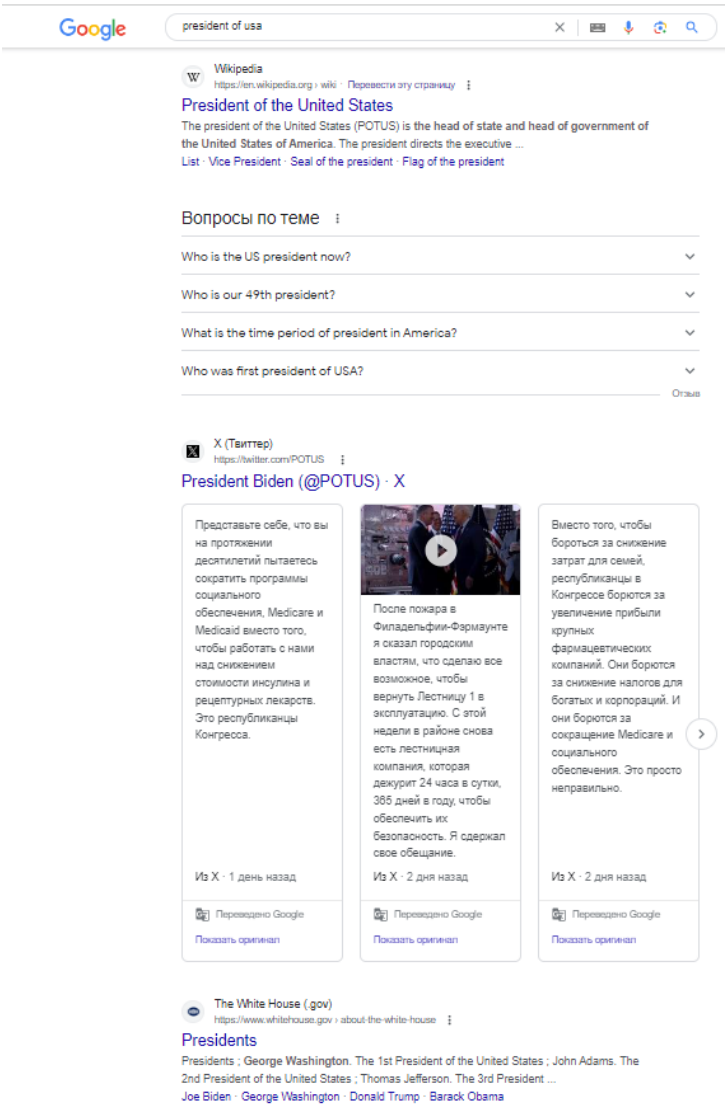

Рис. 1.7. Приклад пошуку за ключовими словами

Питання-відповіді системи розвивалися в контексті систем інформаційного пошуку, мета яких – знайти відповідь на поставлене запитання у чітко визначеному корпусі документів. Їх часто називають системами із замкнутою областю. Вони розширюють можливість пошуку за ключовими словами у двох найважливіших напрямках. По-перше, користувачеві дозволяється ставити складні питання природною мовою, які було б важко сформулювати у вигляді простого запиту за ключовими словами. У контексті веб-запитів це означає, що людина, яка ставить запитання, не повинна знати, як саме організовані дані. Потім застосовуються складні методи обробки природної мови (ОЕЯ, англ. NLP) з метою зрозуміти зміст питання. По-друге, в результаті пошуку в корпусі документів повертається явна відповідь, а не посилання на документи, які відповідають запиту. Це не означає, що

точні відповіді повертаються, як у традиційних СУБД, але може бути повернено ранжований список явних відповідей, а не набір веб-сторінок. Наприклад, у відповідь на запит за ключовими словами «президент США» пошукова система повернула б результат, показаний (частково) на рис. 1.7. Очікується, що користувач знайде відповідь на сторінках, URL-адреси та короткі витримки з яких (цитати) представлені на сторінці результатів. З іншого боку, у відповідь на запитання природною мовою «Хто є президентом США?» система могла б повернути ранжований перелік імен президентів (точний вид відповіді залежить від системи).

Питання-відповіді системи були узагальнені для роботи в Інтернеті. У таких системах веб розглядається як корпус документів (тому вони називаються системами з відкритою областю). Доступ до джерел веб-даних здійснюється за допомогою обгорток, розроблених для отримання відповіді на запити. Створено цілу низку запитально-відповідних систем з різними цілями та функціональністю, наприклад Mulder, WebQA, Start та Tritus. Існують також комерційні системи з різними можливостями (наприклад, Wolfram Alpha http://www.wolframalpha.com/).

Ми опишемо загальну функціональність цих систем, посилаючись на еталонну архітектуру на рис. 1.8. Етап передобробки, що існує не в усіх системах, – це автономний процес виділення та доповнення правил, що використовуються в системі. У багатьох випадках він полягає в аналізі документів, отриманих з Інтернету або повернутих у відповідь на раніше задані питання. Мета такого аналізу – визначити найбільш ефективну структуру запиту, яку слід перетворити питання користувача. Наприклад, у Tritus застосовується підхід на основі машинного навчання з використанням як навчального набору часто задаваних питань та правильних відповідей на них. Система намагається вгадати структуру відповіді, для чого аналізує питання і шукає відповідь на нього у навчальному наборі. Процес складається із трьох етапів. На першому етапі в результаті аналізу виділяється запитальна фраза (наприклад, у питанні «What is a hard disk?» запитальною фразою є «What is a»). Вона служить для класифікації питання. На другому етапі аналізуються пари запитання-відповідь у навчальних даних і генеруються перетворення-кандидати

для кожної запитальної фрази. Наприклад, для запитальної фрази "What is a" (що таке) генеруються обороти "refers to" (ставиться до), "stands for" (означає) і т. д. На третьому етапі кожне перетворення-кандидат застосовується до питань навчального набору і запити, що вийшли, відправляються різним пошуковим системам. Обчислюється схожість повернених результатів з правильними відповідями в навчальному наборі, і на цій основі проводиться ранжування перетворень-кандидатів. Ранжовані правила перетворення зберігаються для подальшого використання під час обробки реальних питань.

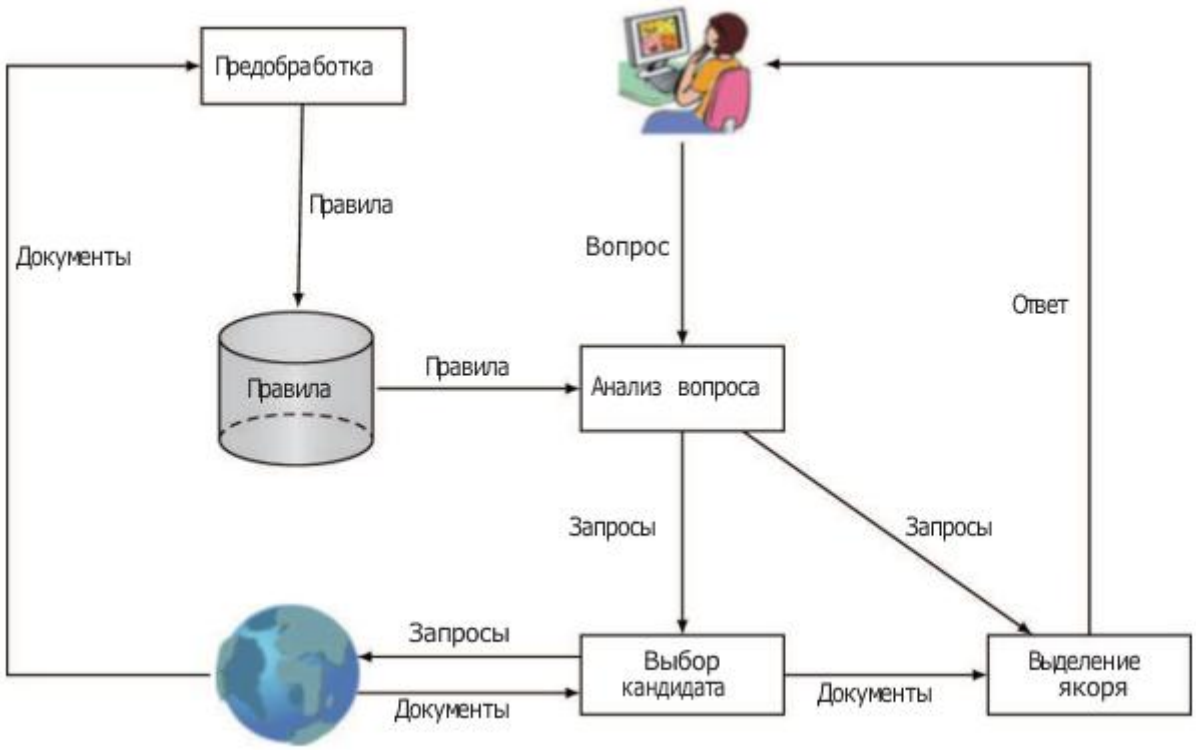

Рис. 1.8. Загальна архітектура QA-систем

Задане користувачем питання природною мовою спочатку проходить процедуру аналізу. Її мета – зрозуміти сенс питання. Більшість систем намагаються вгадати тип питання, щоб класифікувати його, а потім використати результат під час трансляції питання мовою запитів і виділення відповіді. Якщо передобробка проводилася, то як підмога використовуються створені правила перетворення. Хоча загальні цілі завжди однакові, рішення, запропоновані в різних системах, значно розрізняються в залежності від розвиненості застосовуваних методів ОЕЯ (цей етап, як правило, повністю зав'язаний на ОЕЯ). Наприклад, аналіз питання в системі

Mulder складається з трьох стадій: синтаксичний розбір питання, класифікація питання та генерування запиту. На першій стадії генерується дерево аналізу, яке використовується при генеруванні запиту та виділенні відповіді. На стадії класифікації питанню зіставляється один із кількох класів, наприклад: іменний для іменників, числовий – для чисел та тимчасовий – для дат. Така класифікація проводиться у більшості QA-систем, тому що спрощує виділення відповіді. Нарешті, на стадії генерування запиту створене раніше дерево розбору застосовується для конструювання одного або кількох запитів, які можна виконати для отримання відповіді на запитання. У Mulder на цій стадії використовується чотири методи:

 заміна дієслова. Головне та допоміжне дієслова замінюються відмінним дієсловом (наприклад, «When did Nixon visit China?» перетворюється на «Nixon visited China»);

 розширення запиту. Прикметник у запитальній фразі замінюється іменником (наприклад, «How tall is Mt. Everest?» перетворюється на «The height of Everest is»);

 утворення іменної групи. Деякі іменні групи полягають у лапки, щоб на наступній стадії передати їх пошуковій системі разом;

 трансформація. Структура питання перетворюється на структуру чекає-

мого типу відповіді ("Who was the first American in space?" перетворюється на "The first American in space was").

Mulder – приклад системи, у якій використовуються розвинені засоби ОЕЯ для аналізу питання. На іншому кінці спектру знаходиться система WebQA, яка відноситься до аналізу питання не так серйозно.

Після того, як питання проаналізовано та згенеровано один чи декілька запитів, настає черга наступного кроку – генерування потенційних відповідей. Запити, що згенеровані на стадії аналізу питання, використовуються для пошуку релевантних документів за ключовими словами. Багато систем на цьому кроці просто звертаються до універсальних пошукових систем, але є й такі, що вдаються до додаткових джерел інформації, доступних у Інтернеті. Наприклад, система ЦРУ World Factbook

(https://www.cia.gov/library/publications/theworld-factbook/) – дуже популярне джерело надійної фактологічної інформації про країни. Аналогічно надійні відомості про погоду можна отримати з таких джерел метеорологічної інформації, як Network (http://www.theweather-network.com/) або Weather Underground (http://www.wunderground.com/). Ці додаткові джерела іноді можуть давати якісніші відповіді, і різні системи користуються ними по-різному. Оскільки відповіді на різні запити краще шукати в різних джерелах (а іноді навіть у важливий аспект цієї стадії обробки – вибір відповідних систем або джерел даних. Наївне пред'явлення запитів усім пошуковим системам або джерелам даних – не найкраще рішення, тому що в Інтернеті ці операції коштують дуже дорого. Зазвичай для вибору джерел використовується інформація про категорію разом із ранжованими списками пошукових систем та джерел для різних категорій. Для кожної пошукової системи або джерела даних необхідно написати обгортку, яка перетворює запит на формат цієї системи (джерела), а отримані результати – на єдиний формат для подальшого аналізу.

У відповідь на запит пошукові системи повертають посилання на документи разом із короткими витримками, а інші джерела даних можуть повертати результати у різних форматах. Отримані результати перетворюються на «записи» єдиного формату. З цих записів потрібно отримати прямі відповіді – у цьому полягає стадія виділення відповідей. Для порівняння ключових слів із записами (або їх частинами) застосовуються різні методи обробки тексту. Потім результати необхідно ранжувати, застосовуючи різні прийоми, запозичені з інформаційного пошуку (наприклад, частота слів, зворотна документна частота). У цьому процесі використовується інформація про категорію, згенерована на стадії аналізу питання. У різних систем різне уявлення про те, що таке правильна відповідь. Одні повертають ранжований список прямих відповідей (наприклад, на запитання «Хто винайшов телефон» дають відповідь «Олександр Грейам Белл», або

«Грейам Белл», або «Белл», або одразу три у визначеному порядку1), тоді як інші повертають частини записів, що містять слова, що зустрічаються в запиті (резюме релевантної частини документа) у ранжованому порядку.

Питання-відповіді дуже відрізняються від інших підходів до опитування Інтернету, розглянутих у попередніх розділах. Вони гнучкіші, оскільки дозволяють ставити запитання, нічого не знаючи про організацію веб-даних. З іншого боку, вони обмежені особливостями природної мови та труднощами її обробки.

1.5. Пошук та запити до прихованого вебу

В даний час більшість універсальних систем працюють тільки з публічно індексованим вебом (ПІВ), хоча значна частина цінних даних зберігається на прихованих складах – у реляційних базах даних, у вигляді документів та багатьох інших формах. Спостерігається тенденція до пошуку шляхів пошуку також у прихованому інтернеті, і тому є дві причини. Перша – розмір: розмір прихованого вебу (в термінах генерованих HTML-сторінок) набагато перевищує розмір ПІВ, тому можливість знайти відповіді на запити користувачів буде набагато вищою, якщо включити його в простір пошуку. Друга – якість даних: дані, що зберігаються у прихованому вебі, зазвичай набагато вищої якості, ніж ті, що можна знайти на відкритих для загального доступу сторінках, оскільки їх ретельно перевіряють. Якби до них був доступ, то якість відповідей теж покращала б.

Однак пошук у прихованому Інтернеті наштовхується на численні труднощі. Перерахуємо найважливіші з них.

1. Звичайного робота не можна використовувати для пошуку в прихованому вебі, оскільки немає HTML-сторінок, ні посилань, які можна було б відвідати.

2. Як правило, до даних у прихованих базах можна отримати доступ тільки через пошуковий або інший спеціальний інтерфейс, а отже, необхідний доступ до цього інтерфейсу.

3. У більшості (якщо не в усіх) випадках структура бази даних невідома, і постачальники даних неохоче надають інформацію про свої дані, яка могла б

допомогти в процесі пошуку (можливо, через витрати на збір та обслуговування цієї інформації) . Доводиться працювати через інтерфейси, які пропонують джерела даних.

Далі в цьому розділі ми обговоримо ці проблеми та деякі запропоновані рішення.

Обхід прихованого вебу

Один із підходів до пошуку у прихованому вебі – спробувати організувати обхід так само, як для ПІВ. Ми вже говорили, що єдиний спосіб роботи із прихованою базою даних – скористатися пошуковим інтерфейсом. Робот для прихованого вебу повинен вміти вирішувати дві задачі: (a) надсилати запити через пошуковий інтерфейс бази даних та (b) аналізувати повернені сторінки результатів та виділяти з них релевантну інформацію.

Запит через пошуковий інтерфейс

Ми можемо проаналізувати пошуковий інтерфейс та побудувати його внутрішнє уявлення. У цьому поданні мають бути описані поля інтерфейсу, їх типи (наприклад, текстове поле, список, прапорець тощо), їх області значень (наприклад, конкретні значення, як у списку, або довільні текстові рядки, як у текстових полів), а також мітки, асоційовані з полями. Для виділення позначок необхідний повний аналіз HTML-коду сторінки. Потім це уявлення зіставляється з базою даних, створеної спеціально під завдання. Для порівняння застосовуються мітки полів. Якщо мітки збігаються, то базу поміщаються можливі значення поля. Цей процес повторюється всім можливих значень всіх полів у пошуковій формі, форма відправляється серверу з усіма можливими комбінаціями значень, і отримані результати аналізуються.

Інший підхід – користуватися технологією агентів. У цьому випадку розробляються агенти прихованого вебу, які взаємодіють із пошуковими формами та витягують сторінки результатів. Процес складається з трьох кроків: (a) знайти

форми, (b) навчитися заповнювати форми та (c) виявити та отримати сторінки результатів.

Для виконання першого кроку ми починаємо з деякого URL (точки входу), проходимо за посиланнями та застосовуємо евристики для визначення HTML-сторінок, що містять форми, крім тих, що містять поля пароля (вхід до системи, реєстрація, сторінка купівлі). Для заповнення форми потрібно виявити мітки та зв'язати їх з полями форми. Для цього застосовуються евристичні знання про ймовірне розташування мітки щодо поля (ліворуч від нього або над ним). Знаючи мітки, агент визначає предметну область, до якої відноситься форма, і заповнює поля значеннями з цієї предметної області відповідно до міток (значення зберігаються в доступному агенту репозиторії).

Аналіз сторінок результатів

Після того, як форму надіслано, повернену сторінку необхідно проаналізувати та зрозуміти, чи містить вона дані або форму для уточнення пошуку. Для цього можна порівняти значення на цій сторінці зі значеннями з репозиторію агента. Якщо це сторінка даних, обходиться як вона сама, і всі сторінки, куди вона вказує (особливо сторінки з додатковими результатами). Це триває доти, доки не зникнуть усі сторінки, які належать одному домену.

Однак повернені сторінки зазвичай містять багато зайвої інформації, крім власне результатів, тому що в більшості випадків вони створюються за шаблоном, в якому є багато тексту рекламного характеру. Для розпізнавання шаблонів сторінки аналізується текстовий вміст і структура сусідніх тегів, щоб виділити дані, що відносяться до запиту. Веб-сторінка представлена як послідовності текстових сегментів, т. е. ділянок тексту, укладених між двома тегами. Механізм розпізнавання шаблонів влаштований так:

1) текстові сегменти аналізуються на предмет текстового вмісту та сусідніх тегів;

2) початковий шаблон ідентифікується за першими двома вибірковими документами;

3) шаблон генерується, якщо в обох документах знайдено відповідні текстові сегменти та сусідні з ними теги;

4) наступні документи порівнюються зі згенерованим шаблоном. З кожного документа витягуються для подальшої обробки текстові сегменти, відсутні в шаблоні;

5) якщо не знайдено збігів з існуючим шаблоном, вміст документа витягується для генерування майбутніх шаблонів.

1.6. Метапошук

Інший підхід до опитування прихованого Інтернету називається метапошук. Отримавши запит користувача, метапошукач виконує такі дії:

1) вибір бази даних. Вибирається база (або бази) даних, найбільш

релевантна запиту. Для цього необхідно зібрати інформацію про кожну базу даних. Ця інформація називається анотацією вмісту і є статистичну інформацію, яка зазвичай включає документні частоти слів, які у базі даних;

2) трансляція запиту. Запит транслюється у форму, що підходить для кожної бази даних (наприклад, шляхом заповнення деяких полів у пошуковому інтерфейсі бази даних);

3) об'єднання результатів. Дані збираються з різних баз, об'єднуються (скоріше за все також упорядковуються) і повертаються користувачеві.

Нижче ми докладно обговоримо важливі етапи метапошуку.

Виділення резюме вмісту

Перший крок метапошуку – визначити резюме вмісту. У більшості випадків постачальники даних не бажають займатися підготовкою такої інформації. Тому метапошукач виділяє її самостійно.

Один із можливих підходів - зробити вибірку документів з бази даних D і для зустрічається в ній слова w обчислити частоту SampleDF(w). Робиться це так:

1) почати з порожнього резюме вмісту, в якому для кожного слова w частота SampleDF(w) = 0. Крім того, взяти загальний (не спеціалізований для D) словник;

2) вибрати слово та відправити його як запит до бази даних D;

3) із повернутих документів відібрати перші k;

4) якщо кількість повернутих документів перевищує заздалегідь заданий поріг, зупинитися. В іншому випадку повернутись до кроку 2.

Цей алгоритм має дві основні версії, що відрізняються тим, як виконується крок 2. У першому випадку зі словника вибирається випадкове слово. У другому слово для наступного запиту вибирається з тих, що вже було виявлено у процесі вибірки. Перший підхід дозволяє побудувати точніший профіль, але обходиться дорожче.

Альтернатива – скористатися методом сфокусованого апробування, що дозволяє побудувати ієрархічну класифікацію баз даних. Ідея в тому, щоб заздалегідь рознести навчальні документи за кількома категоріями, а потім виділяти з документів терми та використовувати їх як апробуючі запити. Запити з одного слова допомагають визначити фактичні документні частоти цих слів, при цьому для інших слів, що зустрічаються у більш довгих пробах, обчислюються лише вибіркові документні частоти. Вони використовуються як оцінка фактичних документних частот слів.

Ще один підхід полягає в тому, щоб почати з випадкового вибору терма з самого пошукового інтерфейсу у припущенні, що цей терм з великою ймовірністю пов'язаний із вмістом бази даних. Терм вимагають у бази даних, і витягуються перші k документів. Потім з безлічі термів, що зустрілися у вилучених документах, випадково вибирається наступний терм. Цей процес повторюється, доки буде отримано заздалегідь задане число документів, після чого з них обчислюється статистика.

1.7. Категоризація баз даних

Хороший підхід, який допомагає в процесі вибору баз даних, – рознести бази даних за кількома категоріями (як, наприклад, у каталозі Yahoo). Категоризація допомагає знайти базу даних, що відповідає запиту користувача, та зробити більшість повернених результатів релевантними запиту.

Якщо для генерування резюме вмісту застосовується метод сфокусованого апробування, то його можна використовувати, щоб апробувати кожну базу даних запитами з певної категорії та підрахувати кількість відповідностей. Якщо кількість відповідностей перевищує поріг, то вважається, що база даних належить до цієї категорії.

Вибір бази даних

Вибір бази даних – найважливіше завдання процесі метапоиска, оскільки від неї критично залежить ефективність і якість обробки запитів до кількох баз. Алгоритм вибору бази даних намагається знайти най- кращий набір баз, яким варто надіслати запит, на основі інформації про їх вміст. Зазвичай ця інформація включає кількість документів, що містять кожне слово (ця величина називається документною частотою), та іншу просту статистику, наприклад, кількість документів, що зберігаються в базі. Маючи в своєму розпорядженні цю зведену інформацію, алгоритм вибору оцінює релевантність кожної бази даних запиту (наприклад, з точки зору очікуваної кількості повернених базою результатів).

GlOSS – простий алгоритм вибору баз даних, у якому передбачається, що слова, що входять у запит, незалежно розподілені за документами бази, і на основі цієї гіпотези оцінюється кількість документів, що відповідають запиту. GlOSS – представник великої родини алгоритмів вибору баз даних, що спираються на резюме вмісту. Такі алгоритми очікують, що резюме точні та актуальні.

Розглянутий вище алгоритм сфокусованого апробування користується категоризацією баз даних та резюме вмісту для вибору баз. Він складається з двох

кроків: (1) поширити резюме вмісту баз даних на категорії ієрархічної схеми класифікації та (2) використовувати резюме вмісту категорій та баз даних, щоб вибрати бази, сконцентрувавшись на найрелевантніших частинах тематичної ієрархії. Це дозволяє дати відповіді більш релевантні запиту користувача, тому що для їх отримання будуть використовуватися тільки бази, що належать тій же категорії, що і сам запит.

Після того, як вибрані релевантні бази, кожній з них надсилається запит, і повернені результати об'єднуються і пред'являються користувачеві.

.

SECTION 2.
ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ЗНАХОДЖЕННЯ АКТУАЛЬНИХ БАЗ ДАНИХ

2.1. Архітектура обробки мультибазового запиту

Архітектура обробки мультибазового запиту включає в себе значну частину, яка реалізується у рамках архітектури "посередник-обгортка". У цій архітектурі для кожної бази даних існує відповідна обгортка, яка надає інформацію про схему, дані та можливості обробки запитів цієї бази. Посередник централізує інформацію, отриману від обгорток, створює єдине представлення для всіх іменних даних і обробляє запит, використовуючи обгортки для доступу до складу СУБД.

Модель даних посередника може бути реляційною, об'єктно-орієнтованою або навіть слабо структурованою. У даній роботі надається основна увага розподіленій обробці запитів, при цьому використовується реляційна модель, яка є достатньою для пояснення методів обробки мультибазових запитів

У архітектури посередник-обгортка кілька переваг. По-перше, її спеціалізовані компоненти дозволяють по-різному задовольняти потреби різних видів користувачів. По-друге, посередники зазвичай спеціалізуються на наборі взаємопов'язаних складових БД, що містять «схожі» дані, тому експортують схеми та семантику, характерні для конкретної предметної галузі. Спеціалізація компонентів веде до гнучкої та розширюваної розподіленої системи. Зокрема, вона відкриває можливість органічної інтеграції різних даних, які у дуже різних компонентах – від повноцінної реляційної СУБД до простих файлів. Припускаючи використання архітектури посередник-обгортка, можна обговорити різні рівні, що беруть участь у обробці запитів у розподіленій мультибазовій системі. Як і раніше вважається, що на вхід подається запит до глобальних зв’язків, виражений засобами реляційного числення. Оскільки запит адресований глобальним (розподіленим) зв’язкам, розподіленість та гетерогенність даних приховані. У обробці мультибазового запиту беруть

участь три основні рівні. Це розбиття на рівні схоже на те, що є в однорідних розподілених СУБД. Однак, оскільки ніякої фрагментації немає, відпадає необхідність у рівні локалізації даних.

Перші два рівні відображають вхідний запит на оптимізований план його розподіленого виконання (ПВЗ). Вони здійснюють функції переписування запиту, оптимізації запиту та частково виконання запиту. Ці два рівні реалізуються посередником, для чого використовується метаінформація, що зберігається в глобальному каталозі (про глобальну схему, розміщення та можливості). Процедура переписування запиту користується глобальною схемою та перетворює вхідний запит на запит до локальних відносин. Нагадаємо, що існує два основних підходи до інтеграції БД: «глобальна як подання» та «локальна як подання». Таким чином, глобальна схема надає визначення уявлень (тобто відображення між глобальними зв'язками та локальними зв'язками, що зберігаються у складових БД, і запити переписуються з використанням цих уявлень. Переписування можна проводити на рівні реляційного числення або реляційної алгебри. Узагальненою формою реляційного обчислення може бути Datalog, добре пристосовану для такого переписування, таким чином, є додатковий крок трансляції обчислення в алгебру, що нагадує крок декомпозиції однорідних розподілених СУБД.

Другий рівень виконує оптимізацію запиту та (частково) його виконання з урахуванням розміщення локальних відносин та різних можливостей обробки запиту, наявних у складових СУБД та експортованих обгортками. Інформація про розміщення та можливості, що використовується на цьому рівні, може також містити гетерогенні відомості про вартість. Розподілений ПВЗ, породжений цим рівнем, групує всередині підзапитів операції, які можуть бути виконані складовими СУБД та обгортками. Як і в розподілених СУБД, оптимізація запиту може бути статичною чи динамічною. Однак через гетерогенність мультибазових систем (наприклад, деякі складові СУБД можуть повертати відповіді з несподівано великою затримкою) динамічна оптимізація запитів може стати більш критичною. У разі динамічної оптимізації цього рівня

можуть бути додаткові звернення після виконання наступного рівня, як показують стрілки, що позначають результати, що надходять від рівня трансляції та виконання.

Нарешті, цей рівень поєднує результати, що надходять від різних обгорток, щоб дати уніфіковану відповідь на запит користувача. Для цього потрібна можливість виконувати деякі операції над даними, що надходять від обгорток. Оскільки обгортки можуть мати вкрай обмежені можливості виконання, наприклад у випадку дуже простих складових СУБД, посередник повинен самостійно надавати всі можливості, необхідні підтримки оголошеного інтерфейсу.

Третій рівень здійснює трансляцію та виконання запиту, користуючись обгортками. Потім він повертає результати посереднику, який може поєднати результати від різних обгорток. Кожна обгортка підтримує схему обгортки, яка включає локальну схему, що експортується, і інформацію про відображення, необхідну для трансляції вхідного підзапиту (підмножини ПВЗ) із загальної мови на мову складової СУБД. Відтрансльований підзапит виконується складовою СУБД, а локальний результат транслюється назад у загальний формат.

Ця архітектура дозволяє ефективно управляти розподіленими даними, забезпечуючи їхню єдинообразність та доступність для різних користувачів.

2.2. Методи оптимізації та виконання запитів

Якщо взяти до уваги підхід на основі чорної скриньки, то у такому випадку функції вартості виражаються логічно (наприклад, агреговані вартості процесора та введення-виводу, коефіцієнти вибірковості), а не на основі фізичних характеристик (наприклад, потужності відносин, кількості сторінок, кількості унікальних значень у кожному стовпчику). Таким чином, функції вартості для складових СУБД можуть бути визначені наступним чином:

Вартість = вартість ініціалізації + вартість знаходження відповідних кортежів + вартість обробки вибраних кортежів.

Доданки у цій формулі залежать від операторів, проте ці відмінності можна апріорі врахувати. Однак справжній виклик полягає в тому, як визначити коефіцієнти при членах цієї формули для різних складових СУБД. Один із способів вирішення цієї проблеми - побудова штучної бази даних (коли вона називається калібрувальною), виконання в ній ізольованих запитів та визначення коефіцієнтів, вимірявши час.

Невтішно, що результати, отримані на штучній базі даних, можуть бути неінформативними для реальних умов. Альтернативний підхід - виконання пробних запитів у різних СУБД та отримання інформації щодо вартості. Пробні запити можна використовувати для збору відомостей про фактори, що впливають на вартість. Наприклад, вони можуть служити для вилучення інформації зі СУБД, необхідної для побудови та оновлення мультибазового каталогу.

Особливим випадком пробних запитів є вибіркові запити, які класифікуються за різними критеріями, а виконання кожного класу фіксується для отримання часткової інформації про вартість. Класифікація може базуватися на характеристиках запиту, характеристиках відносин-операндів та характеристиках СУБД. Правила класифікації дозволяють ідентифікувати подібні класи запитів і визначати загальні формули вартості. При цьому ефективна класифікація досягається за рахунок комбінування верхнього і нижнього підходів. Глобальна функція вартості, що складається із вартості ініціалізації, вартості вилучення кортежу та вартості обробки кортежу, має параметри, які визначаються за допомогою вибіркових запитів та рівняння регресії.

В кінці кінців, якість моделі вартості контролюється за допомогою статистичних критеріїв, таких як критерій Фішера. Якщо цей критерій не виконується, класифікація запитів уточнюється, і цей процес повторюється, поки якість не визнається задовільною.

У вищезазначених підходах необхідний попередній етап - ініціалізація моделі вартості (через калібрування або вибірку). Проте це може бути неприйнятним, оскільки це може призвести до уповільнення системи при додаванні кожної нової СУБД. Одним з можливих рішень є поступове навчання моделі вартості на запитах. Зазначається, що посередник звертається до складової СУБД у вигляді виклику функції, і вартість виклику складається із часу до отримання першого кортежу, загального часу отримання результату та потужності результату. Це дозволяє оптимізатору запитів мінімізувати час до отримання першого кортежу або час обробки всього запиту, залежно від вимог користувача.

У цьому підході процесор запитів спочатку не має статистики про складові СУБД. Згодом він починає спостерігати за запитами, отримуючи час обробки кожного і-го виклику та запам'ятовуючи його для майбутньої оцінки. Щоб управляти великим обсягом зібраної статистики, менеджер вартості узагальнює її, агрегуючи статистику, що описує подібні запити. Модуль оцінки вартості реалізований декларативною мовою, що дозволяє додавати нові формули вартості для опису поведінки конкретних складових СУБД.

Проте основним недоліком підходу на основі чорної скриньки є те, що модель вартості, хоч і коригована під час калібрування, однакова для всіх СУБД і може не враховувати специфіку кожної з них. Це може призвести до неточної оцінки вартості виконання запиту в кожній окремій СУБД, що може призвести до несподіваної поведінки.

Підхід, заснований на специфічних моделях вартості для кожної СУБД, передбачає, що процесори запитів в різних СУБД дуже відрізняються і тому не можуть бути представлені єдиною моделлю вартості, як у підході на основі чорної скриньки. Також він припускає, що точне оцінювання вартості локальних підзапитів позитивно вплине на глобальну оптимізацію запиту.

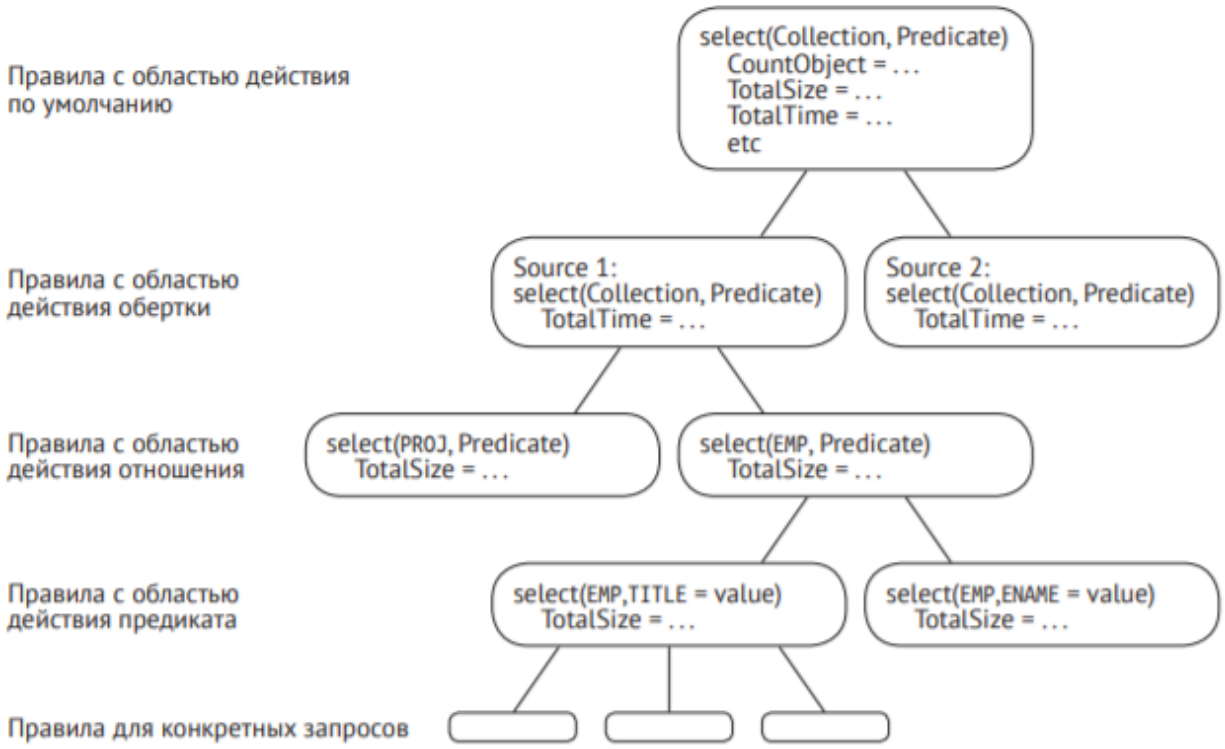

Рис. 2.1. Ієрархічне дерево моделі вартості

Цей підхід надає інфраструктуру для інтеграції моделей вартості складових СУБД в оптимізатори запитів на стороні посередника. Розширюється інтерфейс обгорток, щоб посередник міг отримати від них конкретну інформацію про вартість. Розробник обгортки може надати модель вартості повністю або частково, і проблема полягає в інтеграції цих (можливо, часткових) описів вартості в оптимізатор запитів на стороні посередника.

Існують два рішення. Перше - включити в обгортки логіку обчислення трьох оцінок вартості: час ініціалізації обробки запиту та отримання першого результату (вартість скидання reset_cost), час отримання наступного результату (вартість просування advance_cost) та потужність результату (cardinality).

РОЗДІЛ 3

ПОШУК РЕЛЕВАНТНИХ БД У ВЕБІ

3.1. Управління веб-даними

Всесвітня павутина (або для стислості «веб») стала величезним репозиторієм даних та документів. Хоча результати вимірів відрізняються, немає сумнівів, що Інтернет росте з феноменальною швидкістю. Але важливий не тільки розмір, Інтернет дуже динамічний і швидко змінюється. З практичної точки зору, веб є величезним, динамічним і розподіленим складом даних, і, очевидно, при доступі до веб-даних виникають проблеми управління розподіленими даними.

У своїй нинішній формі Інтернет можна розглядати як дві різні, але взаємопов'язані компоненти. Перша – те, що називається публічно індексованим вебом (ПІВ), – включає всі статичні (і пов'язані перехресними посиланнями) веб-сторінки, розташовані на веб-серверах. Їх можна легко знайти та проіндексувати. Друга називається глибинним (або прихованим) Інтернетом і складається з величезної кількості баз даних, в яких зберігаються дані, приховані від зовнішнього світу. Ці дані зазвичай вилучаються за допомогою інтерфейсів пошуку: користувач вводить запит, він передається серверу бази даних, а результати повертаються користувачеві у вигляді динамічно згенерованої веб-сторінки. Частина глибинного Інтернету, відома під назвою «чорний Інтернет» (або «чорний Інтернет»), складається з зашифрованих даних, для доступу до неї потрібен особливий браузер, наприклад Tor.

Різниця між ПІВ та прихованим вебом переважно полягає в способі пошуку та пред'явлення запитів. Для пошуку в ПІВ необхідно обійти сторінки роботом, який переходить за посиланнями, проіндексувати зібрані сторінки, а потім здійснити пошук у проіндексованих даних. При цьому можна використовувати всім відомий пошук за ключовими словами або питання-відповідну систему (QA-систему). До прихованого вебу такий підхід не застосовується, тому що обійти та проіндексувати ці дані неможливо.

Є два напрями досліджень з управління веб-даними, відповідні спільноти є різними, але перетинаються. Більшість ранніх робіт з пошуку в інтернеті та інформаційного пошуку була присвячена пошуку за ключовими словами і пошуковими системами. Надалі ця спільнота зосередилася на QA-системах. Спільнота баз даних спрямувала свої зусилля на декларативні методи опитування веб-даних. Намітилася тенденція до об'єднання пошуково-переглядового режиму доступу з декларативними запитами, але потенціал тут ще далеко не розкритий. У 2000-х роках на сцену вийшов XML як важливий формат для представлення та інтеграції даних у Інтернеті. Тому управління XML-даними викликало значний інтерес. Хоча XML як і раніше важливий у багатьох прикладних областях, його використання для керування веб-даними зійшло нанівець через надто високу складність. Останнім часом для представлення та інтеграції веб-даних все частіше використовують формат RDF.

Через те, що напрями досліджень розійшлися, не існує уніфікованої архітектури чи каркасу, в рамках якого можна було б обговорювати управління веб-даними, і доводиться розглядати різні напрямки окремо. Крім того, повне освітлення всіх тем, що стосуються Інтернету, зажадало б куди більш глибокого і широкого розгляду. Тому ми зосередимося на питаннях, які безпосередньо пов'язані з управлінням даними.

3.2. Управління веб-графом

Веб складається зі «сторінок», з'єднаних гіперпосиланнями, цю структуру можна змоделювати як орієнтованого графа, який зазвичай називають веб-графом. Його вершинами є статичні HTML-сторінки, а ребрами – посилання між сторінками. Характеристики веб-графа важливі для вивчення проблем керування даними, оскільки ця структура використовується при пошуку в Інтернеті, класифікації та категоризації веб-контенту та вирішенні інших завдань, що стосуються Інтернету. Крім того, RDF-подання формалізує веб-граф із застосуванням точно певної нотації. Перерахуємо важливі характеристики веб-графа:

a) він дуже мінливий. Ми вже обговорювали швидкість, з якою зростає веб-граф. До того ж, значна частина сторінок часто оновлюється;

b) він розріджений. Граф називається розрідженим, якщо середній ступінь його вершин набагато менший від кількості вершин. Це означає, що кожна вершина має мало сусідів, нехай навіть усі вершини якось пов'язані. З розрідженості веб-графа випливає цікава особливість його форми, яку ми обговоримо трохи нижче;

c) він «самоорганізований». Веб містить безліч співтовариств, ко-

Дої з яких складається зі сторінок, присвячених певній темі. Ці спільноти організовуються самі, без будь-якого «централізованого управління», і призводять до появи чітко окреслених підграф у веб-графі;

d) це граф типу «світ тісний». Ця властивість пов'язана з розрідженістю – у кожної вершини графа, можливо, трохи сусідів (тобто її ступінь мала), але багато вершин пов'язані через проміжні. Графи типу «світ тісний» вперше були виявлені в суспільних науках, коли було помічено, що незнайомі між собою люди часто мають ланцюжок спільних знайомих. Це так і для веб-графа;

е) це статечний граф. Розподіл вхідних та вихідних ступенів вершин графа підпорядковується статечному закону. Це означає, що ймовірність того, що вершина має вхідний (вихідний) ступінь і, пропорційна 1/iα для деякого α > 1. Значення α приблизно дорівнює 2.1 для вхідних ступенів і 7.2 для вихідних. Це підводить нас до обговорення будови веб-графа. У ньому мається компонента сильної зв'язності (вузол у центрі), в якій між кожною парою сторінок є шлях. Наведені нижче числа взяті із дослідження 2000 року; вони могли змінитися, але зображена малюнку форма збереглася. Ці числа слід розглядати як відносні, а чи не абсолютні розмірні показники. До компоненту сильної зв'язності (КСС) потрапляє приблизно 28 % веб-сторінок. Ще 21% сторінок потрапляють до компонента. «ВХІД»: для кожної з них є шлях до сторінок, що належать КСС, але ні шляхи з КСС до цієї сторінки. Аналогічно, компоненту «ВИХІД» потрапляють сторінки, куди можна потрапити з КСС, але не можна повернутися; таких теж приблизно 21%. «Вусики» складаються зі сторінок, на які не можна потрапити з КСС та з яких немає шляхів до КСС. Таких приблизно 22%. Це сторінки, які ще не «відкриті» і не мають зв'язків із добре відомими частинами вебу. Нарешті, існують відірвані компоненти – на них не посилається ніхто, крім членів їхньої малої спільноти, і самі вони більше ні на кого не посилаються. Вони становлять близько 8% Інтернету. Ця структура цікава тим, що визначає результати, отримані під час пошуку в Інтернеті. Крім того, вона відрізняється від структури типових графів, що вивчаються, і вимагає розробки спеціальних алгоритмів управління.

3.3. Пошук у вебі

Пошук у вебі (або веб-пошук) має на увазі знаходження «всіх» сторінок, релевантних (тобто містять матеріал, що відповідає темі запиту) ключовим словам, заданим користувачем. Звичайно, знайти всі сторінки неможливо, не можна навіть сказати, чи всі сторінки знайдені, оскільки пошук здійснюється в базі даних сторінок, які були зібрані та індексовані. Зазвичай існує декілька сторінок, релевантних запиту, тому вони пред'являються користувачеві відсортованими за релевантністю, і цей порядок визначається пошуковою системою. Абстрактна архітектура загальної пошукової системи показано на рис. 3.1. Нижче описуються її компоненти.

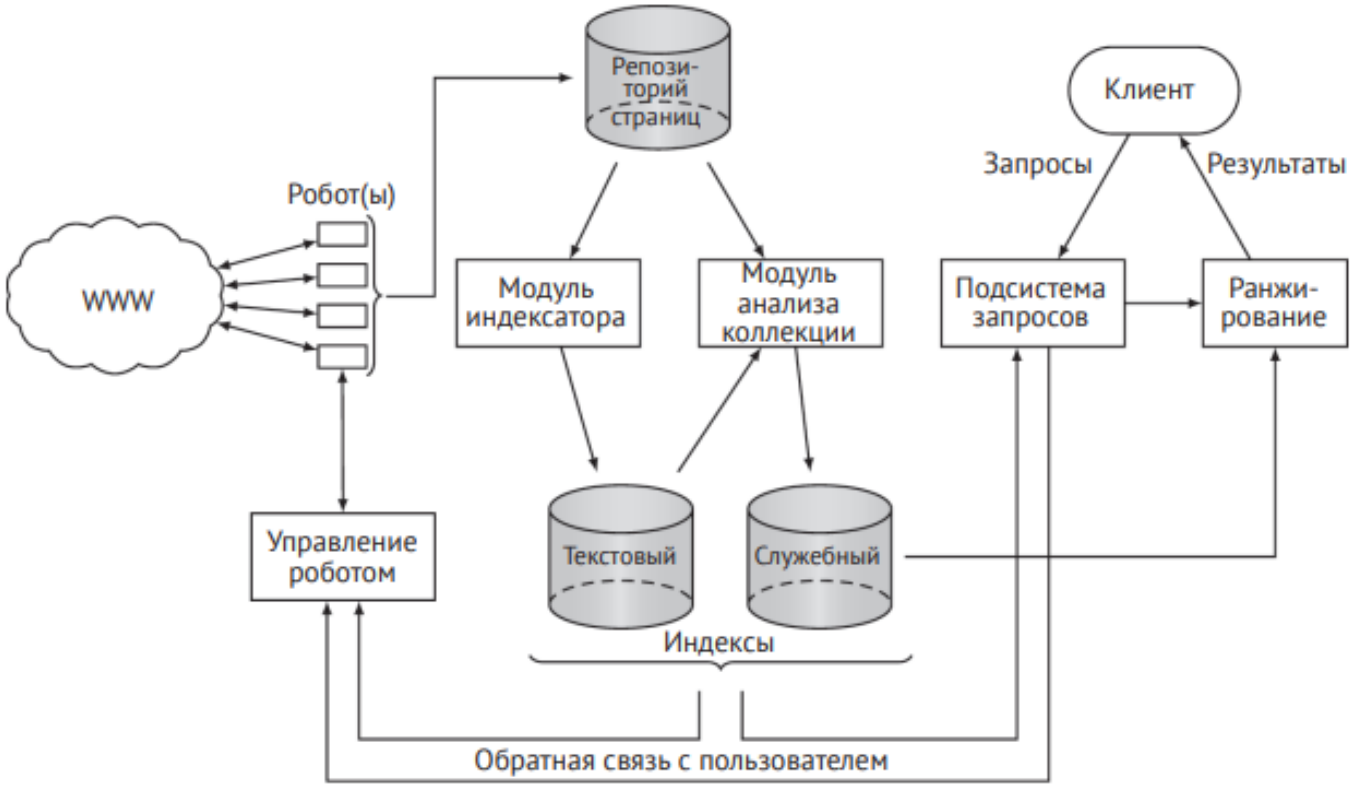

Рис. 3.1. Архітектура пошукової системи

У будь-якій пошуковій системі існує робот (crawler), на який покладається одна з найважливіших ролей. Він переглядає веб від імені пошукової системи та збирає відомості про веб-сторінки. Роботу задається початкова множина сторінок, точніше уніфікованих локаторів ресурсів (URL), що визначають місцезнаходження цих сторінок. Робот зчитує і розбирає сторінки, виділяє URL, що зустрічаються в них, і поміщає їх у чергу. На наступній ітерації циклу робот вибирає URL із черги (у певному порядку) та читає відповідну сторінку. Цей процес повторюється, доки робот не зупиниться. Модуль управління вирішує, який URL відвідати наступним. Прочитані сторінки зберігаються у репозиторії.

Модуль індексатора відповідає за побудову індексів на сторінках, завантажених роботом. Можна побудувати багато різних індексів, але найвідоміші – текстові індекси та індекси посилань. Щоб створити текстовий індекс, індексатор будує величезну довідкову таблицю, з якої можна отримувати URL всіх сторінок, на яких зустрічається дане слово. Індекс посилань описує посилальну структуру веб-сайту та дає інформацію про вхідні та вихідні посилання для кожної сторінки.

Модуль ранжирування відповідає за сортування великої кількості результатів, так що ті, що вважаються найбільш релевантними запиту користувача, опинилися на початку списку. Завдання ранжирування викликало підвищений інтерес у зв'язку з необхідністю вийти за рамки традиційних методів інформаційного пошуку та врахувати особливі характеристики вебу – веб-запити зазвичай малі, але виконуються над великим обсягом даних.

Як було сказано вище, робот переглядає веб від імені пошукової системи, щоб отримати інформацію про відвідувані сторінки. Враховуючи розмір Інтернету, мінливість сторінок та обмежені обчислювальні ресурси та пам'ять роботів, зрозуміло, що обійти весь веб неможливо. Тому робот повинен бути влаштований так, щоб відвідувати «найважливіші» сторінки раніше, ніж інші. Отже, проблема полягає в тому, щоб відвідувати сторінки як важливість.

Під час проектування робота потрібно відповісти на багато запитань. Оскільки основна мета робота – відвідувати важливі сторінки раніше, ніж інші, постає питання, як визначити важливість сторінки. Для цього потрібний показник, що вимірює важливість цієї сторінки. Показник може бути статичним, тобто не залежати від кількості запитів на пошук цієї сторінки, або динамічним, що враховує запити. Приклад статичного показника є кількість посилань, що вказують на сторінку Pi (зворотних посилань). Можна додатково взяти до уваги важливість сторінок, що посилаються, як у популярній метриці PageRank, що використовується Google та іншими системами. Приклад динамічного показника – схожість тексту сторінки Pi із запитом, що обчислюється за допомогою добре відомих метрик інформаційного пошуку.

Ранг сторінки Pi, що позначається PR(Pi), – це просто сума нормованих рангів всіх сторінок Pj, що посилаються на Pi (їх безліч позначається BPi), де нормувальний коефіцієнт для сторінки Pj дорівнює кількості наявних на ній посилань

$$PR(P_i) = \sum_{P_j \in B_{P_i}} \frac{PR(P_j)}{|F_{P_j}|}.$$

Ця формула підраховує ранг сторінки на основі кількості зворотних посилань на неї, але нормує вклад кожної такої сторінки Pj на число посилань з неї. Ідея в тому, що важливіше посилання зі сторінок, які містять не дуже багато добре продуманих посилань, а не вказують на все, що не потрапивши, але тоді внесок сторінки потрібно нормувати на кількість сторінок, на які вона посилається.

Друге питання: як робот вирішує, яку наступну сторінку відвідати після того, як відвідав конкретну сторінку? Ми вже згадували, що робот веде чергу, в якій зберігає URL-адресу виявлених сторінок, і додає до неї елементи в процесі аналізу кожної сторінки. Тому проблема у тому, як упорядкувати цю чергу. Тут можливі різні стратегії. Наприклад, можна відвідувати URL у тому порядку, в якому вони зустрічалися; це називається обхід завширшки. Інший варіант - випадкове впорядкування, коли робот вибирає випадковий URL з черги сторінок, що очікують відвідування. Можна також використовувати метрики, які поєднують упорядкування з описаним вище ранжуванням за важливістю, наприклад, лічильником зворотних посилань або PageRank.

Розглянемо, як можна використовувати PageRank для цього. Нам знадобиться трохи змінити наведену вище формулу. Тепер ми моделюємо випадкового відвідувача: дійшовши до сторінки P, випадковий відвідувач або вибирає один із URL, що зустрічаються на ній, з ймовірністю d, або переходить на якусь випадкову сторінку з ймовірністю 1 – d. Тоді модифікована формула PageRank виглядає так

$$PR(P_i) = (1-d) + d \sum_{P_j \in B_{P_i}} \frac{PR(P_j)}{|F_{P_j}|}.$$

Упорядкування URL за цією формулою дозволяє врахувати важливість сторінки при визначенні порядку відвідування. Іноді перший член нормують на загальну кількість сторінок у Інтернеті.

Оскільки багато веб-сторінок з часом змінюються, обхід Інтернету відбувається безперервно, і сторінки відвідуються знову і знову. Замість кожного разу починати обхід з нуля, краще вибірково відвідувати деякі сторінки і оновлювати зібрану раніше інформацію. Роботи, які так і чинять, називаються інкрементними. Вони намагаються, щоб інформація у їхніх репозиторіях була наскільки можна актуальною. Інкрементний робот вирішує, які сторінки відвідати повторно, виходячи з частоти зміни сторінки, або просто робить вибірку. У підходах на основі частоти зміни для визначення того, як часто потрібно заходити на сторінку, використовується оцінка частоти зміни сторінки. Інтуїтивно здається, що сторінки, що часто змінюються, слід відвідувати частіше, але це не завжди вірно - будь-яка інформація, витягнута з такої сторінки, з великою ймовірністю скоро застаріє, тому іноді краще збільшити інтервал між її відвідуваннями. Можна також розробити адаптивний інкрементний робот такий, що на порядок обходу у черговому циклі впливає інформація, зібрана у попередньому циклі. У підходах з урахуванням вибірки акцент робиться на сайтах, а чи не на окремих сторінках. Щоб оцінити, наскільки змінився сайт, проводиться невелика вибірка з багатьох сторінок, і на основі її аналізу приймається рішення, як часто відвідувати цей сайт.

Деякі пошукові системи спеціалізуються на пошуку сторінок, які стосуються певної теми. Вони використовуються сфокусовані роботи, оптимізовані для цієї теми. Такий робот ранжує сторінки на основі релевантності даної теми та використовує отримані оцінки для вирішення того, яку сторінку відвідати наступній. Для оцінки релевантності застосовуються методи, поширені в інформаційному пошуку, а визначення теми сторінки – методи машинного навчання. Ці методи виходять за рамки книги, але зазначимо, що їх чимало, наприклад, наївний класифікатор байесу, його узагальнення, навчання з підкріпленням та інші.

Для досягнення масштабованості обхід можна розпаралелити, запустивши паралельні роботи. У будь-якому проекті паралельних роботів необхідно передбачити мінімізацію накладних витрат на розпаралелювання. Наприклад, два працюючих паралельно робота можуть завантажити той самий набір сторінок. Очевидно, що такий перетин бажано запобігти, для чого дії роботів потрібно координувати. Центральний координатор динамічно призначає кожному роботу набір сторінок. Інша схема координації передбачає логічне розбиття Інтернету на розділи. Кожен робот знає про свій розділ, тому в центральному координаторі немає потреби. Ця схема називається статичним призначенням.

3.4. Індексування

Для ефективного пошуку сторінок, проаналізованих роботом, будується ряд індексів, показаних на рис. 3.1. Найбільш важливими є два: структурний індекс (або індекс посилань) і текстовий індекс (або індекс за вмістом).

Структурний індекс

Структурний індекс ґрунтується на графовій моделі, в якій граф представляє структуру відвіданої роботом частини Інтернету. Дуже важливо організувати ефективне зберігання та пошук цих сторінок, для чого розроблено два методи, описані в розділі 12.1. Структурний індекс можна використовувати для отримання важливої інформації про посилання між сторінками, наприклад про околиці сторінки та про сторінки, розташовані на одному рівні з нею.

Текстовий індекс

Найважливішим і найчастіше використовується текстовий індекс. Індекси для підтримки текстового пошуку можна реалізувати за допомогою будь-якого з методів доступу, які традиційно застосовуються для пошуку в наборах документів, наприклад: суфіксові масиви, інвертовані файли або індекси та файли сигнатур. Повний розгляд усіх цих індексів виходить за рамки книги, але використання інвертованих індексів ми все ж таки обговоримо, оскільки вони найбільш популярні в цьому контексті.

Інвертований індекс є набором інвертованих списків, кожен список асоційований з одним словом. Загалом елементами інвертованого списку для цього слова є ідентифікатори документів, у яких це слово зустрічається. В списку, що інвертується, можна також зберегти розташування слова на сторінці. Ця інформація потрібна для пошуку з урахуванням близькості та для ранжування результатів запиту. Алгоритми пошуку часто використовують додаткову інформацію про терми, що зустрічаються на сторінці. Наприклад, термам, набраним напівжирним шрифтом (оточеним тегами), що зустрілися в заголовках (всередині HTML-тегів <H1> або <H2>) або в текстах посилань, алгоритми ранжирування можуть призначати більшу вагу.

Крім інвертованого списку, у багатьох текстових індексах зберігається лексикон – список всіх термів, які у індексі. Додатково лексикон може містити статистику, пов'язану з термами, яка використовується алгоритмами ранжирування.

Побудова та супровід індексу стикаються з трьома основними труднощами:

1) в загальному випадку для побудови інвертованого індексу потрібно обробити кожну сторінку - прочитати всі слова, що зустрічаються на ній, і запам'ятати місце розташування кожного слова. Наприкінці цієї процедури інвертовані файли записуються на диск. Для невеликих статичних наборів це завдання тривіальне, але стає дуже складним, якщо потрібно обробляти такий великий і динамічно змінюваний набір, як веб;

2) у зв'язку з швидкою зміною Інтернету виникає питання, як забезпечити актуальність індексу. У попередньому розділі ми говорили, що для цієї мети слід використовувати інкрементні роботи, але все одно індекси необхідно періодично перебудовувати, тому що більшість відомих методів інкрементного оновлення погано працюють, коли обсяг змін великий, як, наприклад, між двома складовими. сідніми обходами Інтернету;

3) формати зберігання інвертованого індексу необхідно ретельно продумувати. Необхідно знайти компроміс між виграшем від використання стисненого індексу, що дозволяє кешувати частини індексу в пам'яті та накладними витратами на розпакування під час виконання запиту. Досягнення прийнятного балансу стає серйозною проблемою під час роботи з наборами документів масштабу Інтернету.

Щоб вирішити ці проблеми і отримати текстовий індекс, що добре масштабується, ми можемо розподілити його, побудувавши або локальний інвертований індекс на кожній машині, де працює пошукова система, або глобальний інвертований індекс, який розділяється всіма машинами.

3.5. Ранжування та аналіз посилань

Типова пошукова система повертає багато сторінок, імовірно релевантних запиту. Однак ці сторінки відрізняються за якістю та ступенем релевантності. Ніхто не хоче, щоб користувач переглядав весь цей набір у пошуках високоякісної сторінки. Потрібен алгоритм ранжирування, який поміщав би більш цікаві користувачеві сторінки на початок списку.

Для ранжування набору сторінок можна скористатися алгоритмами з урахуванням посилань. Повторюючи сказане вище, нагадаємо, що якщо сторінка Pj містить посилання на сторінку Pi, то, мабуть, автори Pj вважали сторінку Pi гідною. Тому сторінка, яку веде багато посилань, мабуть, хорошої якості, отже, кількість вхідних посилань можна як критерій ранжирования. Це інтуїтивне міркування лежить в основі алгоритмів ранжирування, але, звісно, кожен алгоритм формалізує його по-своєму. Ми вже обговорювали алгоритм PageRank, який використовується не тільки для обходу Інтернету, але й для ранжування. А зараз обговоримо ще один алгоритм, HITS, щоб проілюструвати різні підходи до вирішення проблеми.

Алгоритм HITS також ґрунтується на посиланнях. У ньому використовуються поняття

«авторитетних сторінок» та «хаб-сторінок». Добра авторитетна сторінка отримує високий ранг. Між авторитетними сторінками та хабами є взаємно посилюючий зв'язок: гарною авторитетною сторінкою вважається та, на яку посилаються багато хороших хабів, а гарною хаб-сторінкою – та, яка посилається на багато авторитетних сторінок. Тому сторінка, яку вказує багато хабів (хороша авторитетна сторінка), мабуть, має високу якість.

Почнемо з веб-графа G = (V, E), де V – безліч сторінок, а E – безліч зв'язків між ними. Кожній сторінці Pi в V призначається дві невід'ємні ваги (aPi, hPi), які мають її авторитетність і цінність як хаба. Ці значення оновлюються в такий спосіб. Якщо на сторінці Pi вказує багато хороших хаб-сторінок, то aPi збільшується, щоб відобразити всі сторінки Pj, що посилаються на неї (нотація Pj → Pi означає, що на сторінці Pj є посилання на Pi):

$$a_{P_i} = \sum_{\{P_j | P_j \to P_i\}} h_{P_j};$$

$$h_{P_i} = \sum_{\{P_j | P_j \to P_i\}} a_{P_j}.$$

Таким чином, авторитетність (цінність хаба) сторінки Pi дорівнює сумі цінностей хаба (авторитетностей) сторінок, що посилаються на Pi.

3.6. Пошук за ключовими словами

Системи пошуку за ключовими словами – найпопулярніші інструменти пошуку інформації в Інтернеті. Вони прості і дозволяють ставити нечіткі запитання, на які може не бути точної відповіді, але є наближені відповіді, які містять факти, схожі на ключові слова. Проте існують очевидні обмеження щодо можливості простого пошуку за ключовими словами. Насамперед такого пошуку недостатньо, щоб висловити складні запити. Частково цю проблему можна вирішити за допомогою ітеративних запитів, коли відповідь на попередній запит користувача є кон- текстом для наступних запитів від нього. Друге обмеження полягає в тому, що пошук за ключовими словами не використовує глобальне уявлення про інформацію в Інтернеті в тому сенсі, в якому запити до бази даних використовують інформацію, що міститься у схемі. Звичайно, можна заперечити, що для веб-даних саме поняття схеми не має сенсу, проте відсутність глобального уявлення про дані є проблемою. Третя проблема пов'язана з можливими помилками в інтерпретації намірів користувача – неправильний вибір ключових слів може спричинити видачу великої кількості нерелевантних результатів.

Пошук за категорією вирішує одну з проблем пошуку за ключовими словами, а саме відсутність глобального уявлення про Інтернет. Пошук за категорією має багато назв: веб-каталог, жовті сторінки або тематичний довідник. Існує кілька публічних веб-каталогів, наприклад World Wide Web Virtual Library (http://vlib.org)1. Веб-каталог є ієрархічною класифікацією людських знань. Хоча класифікатор представлений у вигляді дерева, це насправді ациклічний орієнтований граф, пов'язаний перехресними посиланнями.

Якщо метою є категорія, то веб-каталог є корисним інструментом. Однак не всі сторінки класифіковані, тому використовувати каталог для пошуку не завжди можливо. Крім того, для класифікації веб-сторінок природна мова не є повністю ефективним засобом. Для оцінки сторінок, що подаються на розгляд, потрібні люди, так що цей підхід не надто рентабельний і погано масштабується. Нарешті, деякі сторінки згодом змінюються, тому підтримання актуальності каталогу пов'язані зі значними витратами.

Були також спроби використовувати кілька пошукових систем для покращення точності та повноти відповіді на запит. Метапошуковик – це веб-служба, яка приймає запит від користувача та надсилає його кільком різним пошуковим системам. Потім метапошуковик збирає відповіді та повертає користувачеві об'єднаний результат. Він уміє сортувати результати за різними атрибутами, наприклад, за власником, ключовим словом, датою чи популярністю. Прикладами можуть бути системи Dogpile (http://www.dogpile.com/), MetaCrawler (http://www.metacrawler.com/) та lxQuick (http://www.ixquick.com/). Різні метапошукачі застосовують різні способи об'єднання результатів та трансляції запитів користувачів мовою конкретної пошукової системи. Користувач може звернутися до метапошуку за допомогою клієнтської програми або на сайті. Кожна пошукова система покриває невелику частину Інтернету. Мета метапошука – охопити більше сторінок, ніж одна пошукова система, об'єднавши результати пошуку.

3.7 Запити до ВЕБу

Декларативні запити та методи їх ефективного виконання завжди перебували у центрі уваги розробників технологій баз даних. Було б добре, якби вдалося застосувати методи баз даних до Інтернету. Тоді доступ до Інтернету можна було б розглядати як віддалений аналог доступу до великої бази даних. У цьому розділі ми обговоримо кілька таких пропозицій. При перенесенні традиційних ідей опитування баз даних на веб-дані ми стикаємося з декількома труднощами. Найважливіша, мабуть, у тому, що опитування бази даних передбачає наявність суворої схеми. Вище ми зазначали, що для веб-даних говорити про таку схему не доводиться1. У кращому разі веб-дані слабо структуровані – у даних може бути якась структура, але, на відміну від баз даних, вона нежорстка, нерегулярна та неповна, тому різні екземпляри даних можуть бути структурно схожі, але не ідентичні (які- то атрибути можуть бути відсутні, інші, навпаки, додаються, структура може змінюватися). Вочевидь, що опитування безсхемних даних деякі проблеми внутрішньо властиві.

Друга проблема полягає в тому, що веб - не просто збори слабо структурованих даних (і документів). Між сутностями, представленими веб-даними (наприклад, сторінками), є посилання, які потрібно враховувати. Як і при пошуку, який ми обговорювали в попередньому розділі, під час виконання запитів до Інтернету слідування за посиланнями може принести відчутну користь. Отже, посилання слід розглядати як повноправні об'єкти.

Третя проблема – відсутність загальноприйнятої мови запиту до веб-даних, аналогічного SQL. Вище ми зазначали, що для пошуку за ключовими словами використовується дуже проста мову, але її недостатньо для більш складних запитів до веб-даних. Поступово виробляється загальна думка щодо базових конструкцій такої мови (наприклад,

дорожні вирази), але ніякого стандарту немає. Проте з'явилися стандартизовані мови таких моделей даних, як XML і RDF (XQuery для XML і SPARQL для RDF).

ВИСНОВКИ

Дана кваліфікаціна робота присвячена розробці інформаційної технологія знаходження актуальних баз даних. В роботі проведено дослідження мульбазових БД, описані їх характеристики та особливості. Також на прикладі вебу аналізується пошук релевантних БД.

REFERENCES

1. Sadalage P., Fowler M. Evolutionary Database Design [Electronic resource] / Martin Fowler, – 2016 – 1 p. – Mode of access: https://martinfowler.com/articles/evodb.html
2. Simmons T.J. Database Migration: What It Is and How to Do It [Electronic resource] / Rollout, – 2018 – 1 p. – Mode of access: https://rollout.io/blog/database-migration/
3. Garrett A. What is Database Migration [Electronic resource] / Alooma, – 2018 – 1 p. – Mode of access: https://www.alooma.com/blog/what-is-database-migration
4. Code-First vs Model-First vs Database-First: Pros and Cons [Electronic resource] / Ryadel, – 2018 – 1 p. – Mode of access: https://www.ryadel.com/en/code-first-model-first-database-first-vs-comparison-orm-asp-net-core-entity-framework-ef-data/
5. A begginer's guide to SQL [Electronic resource] / Learn to code with me, – 2018 – 1 p. – Mode of access: https://learntocodewith.me/posts/sql-guide/
6. Pearlman S. Understanding Data Migration: Strategy and Best Practices [Electronic resource] / Talend, – 2019 – 1 p. – Mode of access: https://www.talend.com/resources/understanding-data-migration-strategies-best-practices/
7. Garrett A. A Modern Approach to Data Migration [Electronic resource] / Database zone, – 2018 – 1 p. – Mode of access: https://dzone.com/articles/a-modern-approach-to-data-migration
8. Arif Ur Rahman, Gabriel D., Cristina R. Transformation Rules for Model Migration in Relational Database Preservation [Electronic resource] / Phaidra, – 2018 – 2 p. – Mode of access: https://services.phaidra.univie.ac.at/api/object/o:294268/diss/Content/get

9. Jeffrey R. Common Language Runtime Via C#, K: Piter, 2016. - 815 c.
10. Margaret Rouse RESTful API [Electronic resource] / Search micro services, – 2018 – 1 p. – Mode of access: https://searchmicroservices.techtarget.com/definition/RESTful-API
11. jQuery 3.4.1: triggering focus events in IE and finding root elements in iOS 10 [Electronic resource] / jQuery Blog Foundation, – 2018 – 1 p. – Mode of access: https://blog.jquery.com/2019/05/01/jquery-3-4-1-triggering-focus-events-in-ie-and-finding-root-elements-in-ios-10/
12. jQuery: The write less, do more, JavaScript library [Electronic resource] / The jQuery Project. – 2018 – 1 p. – Mode of access: https://jquery.com/
13. Data migration diagrams. [Electronic resource] / Togaf-modeling. – 2018 – 1 p. – Mode of access: https://www.togaf-modeling.org/models/data-architecture/data-migration-diagrams.html