Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Прикладних інформаційних технологій та електроінженерії
(повна назва факультету)
Комп'ютерно-інтегрованих технологій
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка системи керування базами даних при виконанні запитів з різнорідних джерел

Виконав(ла): студент(ка) 4 курсу, групи КТс-41
спеціальності 151 – Автоматизація та комп'ютерно-інтегровані технології
(шифр і назва спеціальності)

(підпис) Бойчук Я.А. (прізвище та ініціали)

Керівник (підпис) Курко А.М. (прізвище та ініціали)

Нормоконтроль (підпис) Левицький В.В. (прізвище та ініціали)

Завідувач кафедри (підпис) Микитишин А.Г. (прізвище та ініціали)

Рецензент (підпис) Капаціла Ю.Б. (прізвище та ініціали)

Тернопіль
2022

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет прикладних інформаційних технологій та електроінженерії
(повна назва факультету)

Кафедра комп'ютерно-інтегрованих технологій
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Микитишин А.Г.
(підпис) (прізвище та ініціали)
« » 20___ р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня **бакалавр**
(назва освітнього ступеня)

за спеціальністю 151 Автоматизація та комп'ютерно-інтегровані технології
(шифр і назва спеціальності)

студенту Бойчук Ярослав Андрійович
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка системи керування базами даних при виконанні запитів з різнорідних джерел

Керівник роботи **Курко Андрій Михайлович, к.т.н., доцент**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*28*» *01* 2022 року № *4/7-59*

2. Термін подання студентом завершеної роботи 21.06.2022 р.

3. Вихідні дані до роботи технічна документація

4. Зміст роботи (перелік питань, які потрібно розробити)
1.Аналітична частина 2.Проектна частина 3.Спеціальна частина 4.Безпека життєдіяльності, основи охорони праці

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1.Актуальність роботи. 2.Мета роботи. 3.Цілі і завдання, які необхідно було вирішити. 4.Централізована обробка даних. 5.процес обробки інформації та його складові. 6.Діаграма розгортання ядра виконання запитів. 7.Спосіб виконання операції підключення. 8.Приклад трансляції запиту у форматі JSON. 9.Інтерфейси взаємодії ядра виконання та зовнішніх компонентів системи. 10.Діаграма класів для уніфікованого формату представлення даних. 11.Діаграма класів для уніфікованого представлення плану виконання запиту. 12.Діаграма розгортання прототипу. 13.Процес обробки запитів. 14.Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Гурик О.Я., доцент кафедри МТ		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	**Примітка**
1	Аналітична частина	29.01-26.02	
2	Проектна частина	27.02-28.03	
3	Спеціальна частина	29.03-01.05	
4	Безпека життєдіяльності, основи охорони праці	02.05-01.06	

Студент _____ Бойчук Я.А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Курко А.М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота бакалавра складається з пояснювальної записки та графічної частини (ілюстративний матеріал – слайди).

Об’єм графічної частини роботи становить 15 слайдів.

Об’єм пояснювальної записки складає 58 друкованих сторінок формату А4 (210×297), об’єм додатків – - друкованих сторінок формату А4.

Робота складається з чотирьох розділів, в яких нараховується 22 рисунків та 3 таблиць з даними.

В роботі використано 22 літературних джерел.

Метою цієї роботи є створення системи управління базами даних при виконанні запитів з різнорідних джерел.

У даній роботі розглянуті способи системи керування базами даних при виконанні запитів з різнорідних джерел. У процесі реалізації поставленої задачі було розроблено архітектуру ядра виконання запитів, забезпечена можливість інтеграції з мовними компонентами та оптимізатором, розроблені інтерфейси взаємодії із зовнішніми компонентами системи, розроблено уніфікований тип предстявлення інформації для внутрішнього використання, розроблено формат для представлення плану виконання запиту; забезпечено підтримку конвеєрної обробки даних у рамках розробленої архітектури.

Ключові слова: БАЗИ ДАНИХ, ПРОГРАМНИЙ ІНТЕРФЕЙС, АРХІТКЕТУРА, ПЛАТФОРМА

ЗМІСТ

ВСТУП

В недалекому минулому організація обробки інформації на комп'ютері здійснювалася одним і тим же керованим об'єктом (наприклад, в залежності від його типу регулювання, ціноутворення і т. д.) і ступеня гомеостазу. Приклади двох типів лінійних структур, які використовували підхід до формування масивів на основі інформації з матеріальних ресурсів: умовно - постійні (з використанням інформації, що використовується багато разів з плином часу) і умовно - змінні (з використанням фактичної або поточної інформації).

Утворення та повторне використання масиву з умовно постійною інформацією має перевагу значно спростити ключову документацію, видаливши з її конфігурації багато постійних деталей. Знижує складність завдання на етапі первинної документації, підготовки та заповнення типу фактів або поточної інформації на комп'ютер.

Масиви інформації з лінійною структурою мають недоліки а саме те, що одна і та ж керована інформація поширюється на безліч різних масивів (положення, плани і т. д.). Це неминуче призводить до деяких деталей дублювання і складності. Не допускайте реалізації принципів незалежності від користувацьких додатків, таких як спільна обробка, а головне, користувацьких додатків. Лінійні масиви, утворені звичайними методами, як правило, ефективні в положенні одного застосування.

З розширенням програмного забезпечення та підтримки систем автоматичної обробки інформації, прагнення забезпечити впровадження сучасних методів обробки даних в системі реального часу та реалізувати мультидоступ до прихованих даних показало новий тренд компіляції інформаційної підтримки розподілених баз даних. Такі базові терміни використання створюють складний масив нелінійних структур з усіма даними про цю або її предметну область, або керовані об'єкти, як постійні, так і змінні символи.

Коли утворюється хороший варіант структурованої інформаційної бази даних, яка широко відображає деталі об'єктів управління, розробники стикаються з вимогою організації потенційно конфліктуючих масивів. До них

відносяться:

- увесь масим подачі даних.
- мінімальна структура даних.
- оптимальний час відбірки даних.
- структура масиву не залежить від програмного забезпечення організації.
- гнучкість структури системи даних

Стабільну роботу інформаційної системи об'єкта можна реалізувати тільки при правильній організації інформаційної бази (організованого набору інформації, що використовується в роботі ІС і розділеної зовні) і машинної бази (ГОСТ 34. 003-90).

Система інформаційної бази даних зі зовнішніх носіїв є - збірка повідомлень, різних надходжень і документів, які є складовою інформаційної бази і призначені для прямого сприйняття без використовування комп'ютерних технологій.

Інформаційна база in-machine - частина інформаційної бази, що використовується ІС на носії зберігання. Ця зовнішня машина ІС зазнала безліч змін від презентацій у вигляді паперових повідомлень, запитів на екранах дисплеїв, голосового зв'язку з комп'ютерами і т. д.

Інформаційна база даних пройшла 3-и еволюційні етапи.

Етап перший характеризується відключеними елементами баз даних.

1) Програма для вирішення індивідуальних задач була одна з обробленим масивом.

2) Використання масивів в різних завданнях забезпечувалося індивідуально шляхом адаптації до формату представлення даних, структури елементів масиву і так далі. і т. д. ;

3) Структура відома давно, тому немає необхідності пояснювати дані.

4) Регулювання масиву здійснювалося індивідуальними засобами.

5) Проблема вирішувалася в пакетному режимі, а результати користувач отримував тільки у вигляді типових скриптів і задіяних документів, через систему підготовки і обробки даних.

Об’єми інформації переглядаються на трьох рівнях, і існує прямий зв'язок між логічним рівнем програми, фізичним рівнем утримання і логічним рівнем.
Етап другий - централізована база даних.

1. Інформаціє з даними окремо від процедури обробки і реалізується в бібліотеку публічних масивів. Представлення інформації, формат елементів даних, структура масивів є однорідними і незалежними від конфігурації або конфігурації пам'яті.

2. Характеристика даних відєднується як від програми, так і від самих даних, так що дані і програма для їх обробки практично незалежні. Такий етап спростовує зміну структур і програм даних. Однак, щоб реорганізувати окремі групи бібліотек та їх компоненти, необхідно змінити програму обробки.

Сьогодні кількість інформації, що зберігається на електронних носіях, стрімко зростає (рисунок 1). Дані можуть бути структуровані (база даних), неструктуровані (текстові файли, веб-сайти і т. д.), або погано структуровані (текст і т. д.).

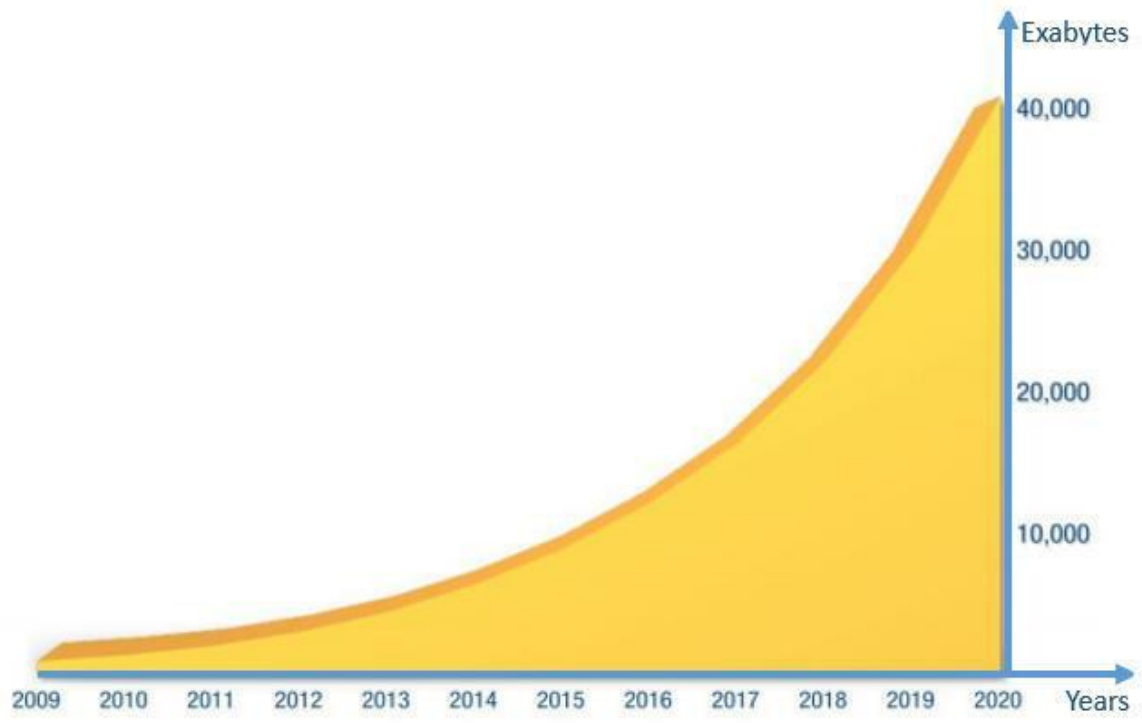

Рис. 1: Графік зростання обсягу даних.

Слід зазначити, що крім всього різноманіття і різноманітності форматів, використовуваних при роботі з інформацією, використовувану базу даних можна розділити на безліч груп з різних причин. Наприклад, внутрішній формат відображення інформації має такий вигляд так:
• Підтримує реляційні моделі (PostgreSQL, Oracle DB, MySQL і т. д.),
• NoSQL інформаційні дані(MongoDB, CouchDB, Redis, AsterixDB і т. д.).
Інформаційнідані можуть бути розподілені відповідно до інтерфейсів (ODBC, JDBC, ADO і т. д.), які надаються для доступу до інформації. Також зверніть увагу, що єдина база даних може забезпечити доступ і управління декількома типами збережених даних.
Різні завдання можуть вимагати централізованого доступу до різних інформаційних ресурсів. За час його роботи багато існуючих баз даних накопичили великі обсяги інформації, що зробило передачу даних в новий централізований репозиторій довгим і трудомістким завданням.

Подібно до того, що використовується в системі звітності Pentaho[18], підхід, заснований на обробці даних з джерел, які надають можливість доступу до інформації з загального інтерфейсу, звужує обсяг піддержуваних використовуваних систем до баз даних, які надають доступ до інтерфейсу.

Щоб реалізувати дані вимоги потрібно створення додаткових компонентів в базі даних, що забезпечують централізований доступ і управління інформацією з існуючих і існуючих джерел. Основні принципи створення такої системи викладені в [10]. Існують також шаблони дизайну для організації доступу до розрізнених інформаційних ресурсів, один з яких показаний в [19].
Інсталяція даного типу інструменту залежна від такими питань, як передача підзапитів джерел, вибір алгоритмів маніпулювання даними, організація обробки даних трубопроводу з різних джерел [2]. Також відзначимо, що в більшості випадків розглянуте завдання зажадає допомоги в наближеному виконанні і обробці нечітких запитів.

Встановлено [7], що етап обробки запитів на неоднакові джерела логічно можна поділити на складові застосування: аналіз запитів, оптимізація для підвищення продуктивності та продуктивності.
Тому систему обробки і управління інформацією з різних джерел можна розділити на ряд наступних функціональних компонентів:
По-перше, є можливість виділити мовний інструмент для написання нечітких запитів. Цей компонент повинен перетворити представлення мови запиту в розгалужену структуру, що має місце використання підпрограм з розширеної реляційної алгебри[10].

Слідуючим компонентом є оптимізатор запитів. Він приймає деревоподібне представлення запиту з мовного компонента і робить оптимізації (без зміни типу запиту) для підвищення продуктивності на основі моделі економічної ефективності кожного виконавця або вихідного адаптера.

Наступним елементом системи має стати ядро виконання запиту, яке одержує удосконалені запити від оптимізатора, збирає необхідні дані з джерела, і виконує операції над ними відповідно до отриманого від оптимізатора плану виконання запиту.

Розроблений на підприємстві повязвного з інформаційним аналізом та систем, є частиною цього дослідження, проект створює систему управління даними з різноманітних інформаційних систем на основі принципів, викладених нижче [10] і включає в себе перераховані вище компоненти.

1. АНАЛІТИЧНА ЧАСТИНА

1.1 Постановка задачі

:

Метою є впровадження виконавців запитів у пристрої обробки даних з неоднакових джерел. Він підтримує здатність виконувати розширені реляційні алгебри[10] операції на інформації, витягнутої з неоднорідних середовищ. У цьому контексті були поставлені такі задачі:

• створення архітектури ядра запиту;

• дати можливість підтримку обробки даних конвеєрним способои в рамках розробленої архітектури;

• дати спосіб реалізації у інтеграцію з мовними елементами та оптимізаторами;

• надає можливість встановлення нових виконавців і вихідних адаптерів;

• перевірити архітектуру, розроблену шляхом взаємодії з раніше реалізованими виконавцям (джерелами) даних на способі AsterixDB[13] та мовними інструментами високого рівня для написання запитів.

1.2 Огляд існуючих аналогів

Огляд підходів до обробки неоднорідної інформації

Програмні інтерфейси для отримання доступу до баз даних

Існують програмні інтерфейси, такі як ODBC, JDBC та ADO для інтеграції доступу до бази даних.

ODBC (“Open Database Connectivity”) — це інтерфейс бази даних з відкритим доступом.

ODBC призначений для забезпечення сумісності між різними базами даних, сумісними з SQL.

Технологія ODBC створює додатковий шар між додатком і використовуваними

СКБД. Архітектура ODBC використовує один менеджер драйверів ODBC і кілька драйверів ODBC, відповідальних за реалізацію функцій доступу для окремих СКБД.

Перевага:

• не складність розробки додатків.

• програмний інтерфес ODBC дозволяє створювати децентралізовані, неоднорідні додатки, незалежно від конкретних СКБД. Додаток не залежить від системи управління базамии даних.

Недоліки:

• Повільний доступ до даних. Виникає потреба у перетворенні запиту.

• Збільшення терміну обробки запитів, пов'язане з введенням додаткових програмних продуктів.

• Кожна робоча станція повинна попередньо встановити і налаштувати драйвер ODBC (вказавши драйвер СКБД, мережевий шлях до сервера, базу даних і т. д.). Параметр для цього параметра статичний. Додаток не може змінити їх самостійно.

• Надає доступ тільки до реляційних SQL-орієнтованих баз даних.

OLE DB

Однак інформація в базі даних може показуватися в будь-якому форматі і також у виді (електронна таблиця, документ формату RTF, поштові дані і т. д.). Вам потрібно створити єдиний формат зберігання даних, який є дорогим і неефективним, або побудувати існуючу технологію з інтерфейсом для доступу до любого типу даних.

Ця вимога реалізується технологією OLE DB.

Тип даних OLE (База даних зв'язування та встроювання об'єктів) - це технологія, яка надає рішення для забезпечення доступу до даних com-додатків, від любого джерела даних.

Технологія OLE DB використовує механізми постачальника, які розуміються як постачальники даних.

Окрім постачальників даних, є також постачальники послуг, які реалізують різні функції обслуговування.

ADO、DAO

Відкритий часто задіяний програмний інтерфейс ODBC і OLEDB вважаються хорошими інтерфейсами передачі даних, але як інтерфейси програмування вони мають багато обмежень, оскільки вони мають низький рівень. Щоб зняття ці обмеження були розроблені технології DAO і ADO. Ці технології є високорівневими об'єктними моделями (функціональними бібліотеками), які забезпечують ще один рівень абстракції між додатком та функціями ODBC та OLEDB.

Інтерфейс ADO представляє ієрархічну об'єктну модель для доступу до різних постачальників даних OLEDB. Об'єктна модель ADO включає об'єкти, які забезпечують підключення до постачальників даних, створення SQL-запитів проти даних тощо.

У загальному технологію ADO можна описати як новітні технології розробки додатків для роботи з розподіленими базами даних з можливістю застосування клієнт-серверних технологій.

BDE

BDE (Borland Data Engine) - це технологія програмного середовища компаныъ Borland.

Розробка створена як динамічно пов'язані бібліотеки і має досить сучасний інтерфейс прикладної програми під назвою IDAPI (Integrated Database Application Program Interface).

Ця система являє собою набір функцій для маніпулювання базою даних.

JDBC

+ JDBC (Java Database Connectivity) - це продвинутий інтерфейс до бази даних на основі Java. Це середовище прикладного програмування для створення SQL-запитів у базі даних з відповідної програми, написаної незалежною від платформи мовою Java, що дає можливість робити як незалежні програми, так і додатки, вбудовані у веб-сторінки.

ODBC і JDBC

Як правило, програмний продукт пишеться на конкретній мові програмування (Java, C#і т. д.), але база даних приймає запити на другій з використання мові бази даних (наприклад, SQL). Отже, коли програмному продукту необхідно одержати доступ до елементів в базі даних, йому потрібен пристрій обробки, який може перетворювати мови одину на іншу (додаток і базу даних). В другому випадку програміст програми зобов’язаний вивчити мову мови бази даних і включити її в програму. ODBC (підключення до відкритої бази даних) і JDBC (Підключення до Java DatabBase) - це два інтерфейси, які дають спроможність реалізувати цю конкретну вимогу. ODBC - це доступний інтерфейс для викорстання, мови програмування та операційної системи, що можна застосовувати для цієї ідеї. Аналогічним чином, JDBC є API даних середовища використання Java. Програмісти Java можуть задіювати вектор від JDBC до ODBC для зв'язку з сумісними базами даних ODBC.

ODBC

ODBC - це середовище для доступу до об’єктів керування базами даних. ODBC була створена SQLAccessGroup в 1992 році, коли не було стандартного середовища між базою даних і додатком. Він не підлягає впливу від любої мови програмування, системи баз даних або операційної середовища застосування. Користувачі програмним продуктом можуть задіювати інтерфейс ODBC для збереження програм, які мають можливість запитувати інформацію з іншої бази даних, вільно від програмного застосунку на виконання або типу використовуваної бази даних.

Драйвери ODBC виступають перекладачами між додатками та базами даних, тому ODBC може бути вільним від мови тпрограмування та платформи. Мається на увазі, що застосунок звільняється від потреби знання мови відповідної бази даних. Замість цього він розпізнає та застосовує лише симантику ODBS, а драйвер переводить запит у базу даних програмними елементами, які він може розпізнати. Отримані дані повертаються у форматі, який може зрозуміти додаток. API програмного забезпечення ODBC може використовуватися як у табличних, так і в неструктурованих системах баз

даних. Ще однією великою перевагою використання ODBC як простого у використанні тимчасового програмного забезпечення між вашою програмою та базою даних є те, що не має необхідності переналагоджувати на нову версію програмне забезпечення щоразу, коли змінюються технічні характеристики вашої бази даних. Досить просто оновити драйвер ODBC.

JDBC

JDBC - це програмний інтерфейс даних, створений для мови програмування Java. Який був запущений з JDK 1. 1 SunMicrosystems (оригінальний власник Java). На даний момент поточна версія JDBC 4. 0 (актуальна ан JAVA SE6). В пакетах Java. sql і Javax. sql розміщені класи JDBC. Такий додаток, допомагає користувачам отримати доступ до системи баз даних, даючи можливість встановлювати спосіб запиту та абгрейду даних у базі. JDBC підходить для об'єктно-орієнтованих баз даних. За допомогою JDBC до ODBC можна отримати доступ до баз даних, сумісних з ODBC.

ODBC - це вільне середовище, який люба програма може використовувати для зв'язку з різними системами баз даних, а JDBC - це середовище, яке програми Java можуть використовувати для зєднання з базою даних. Отож, в порівнянні з JDBC, ODBC є незалежним від якої програмування мови використовується. Однак використання моста Java від JDBC до ODBC також дозволяє програмам Java проводити обмін даними з сумісними базами даних ODBC.

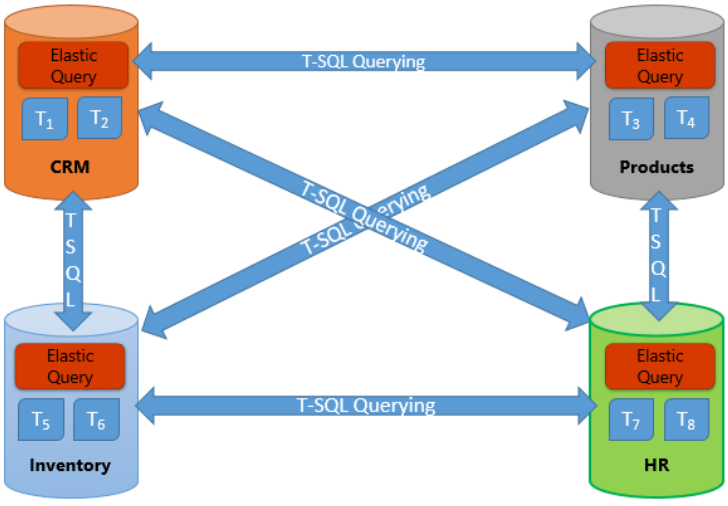

Рис.1.1 Запит до декількох хмарних баз даних з різними схемами

Об’єми даних з вертикальними розділами використовує різні набори таблиць з різних баз даних. Воно означає, що різні бази даних мають різні схеми. Наприклад, усі таблиці, пов'язані з інвентаризацією, зберігаються в одній базі даних, а таблиці, пов'язані з бухгалтерським обліком, зберігаються в іншій базі даних.

Передумовою

•Користувач повинен мати дозвіл змінити будь-який зовнішній ресурс. Цей дозвіл включено до зміни структури таблиці ALTERDATABASE.

•Для доступу до основного джерела даних необхідно мати дозвіл alter any externaldatasource.

Spark SQL

Spark SQL є елементом на SparkCore, який вводить нове пойняття даних під назвою SchemaRDD, яка надає підтримку структурованих і напівструктурованих даних. Ядро ApacheSpark. Spark Core є основним поширеним двигуном виконання платформи Spark, де побудовані всі інші функції. Воно дає можливість проводити розрахунки в пам'яті та посилання на різні дані у зовнішніх елементах зберігання даних [20].

Spark SQL є частиною фреймворку Spark для структурування та обробки розподілених даних, і дає можливість надсилати запити в SQL (мова структурованих запитів). Spark SQL використовує рамки даних для обробки даних. Рамка даних — це структура даних, яка є таблицею, яка містить дані, упорядковані за іменованими стовпцями. Spark SQL підтримує завантаження даних з різних зовнішніх джерел (Hive, JSON, Avro і т. д.) для формування рамки даних.

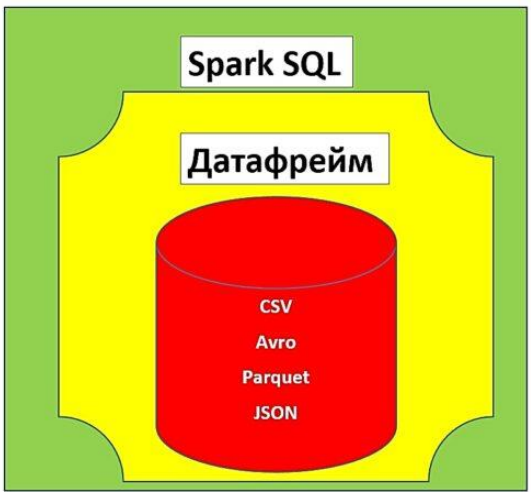

Рис. 1.2 Структура Spark SQL

Spark SQL є елементом фреймворку ApacheSpark для структурованої (відповідно зорієнтованої) обробки даних. Оскільки внутрішні механізми обчислень і перетворення в Spark SQL є незалежними від мови програмування, розробники мають можливість самостійно визначати інтерфейси для мов програмування (Java, Python, Scala, R і т.д.). SparkSQL має такі характеримтики:

•Інтеграція – Spark SQL може отримувати при запиті дані за допомогою SQL-запитів з різних СКБД (MySQL, MS SQL Server, Oracle, Hive і т.д.) у вигляді розподілених збірок даних (Стійкий розподілений набір даних, RDD).

•Універсальний доступ до даних - Доступ до даних надається через механізм SQL-запитів, тому робота над усіма джерелами даних ведеться за однією схемою.

• Джерела даних - Spark SQL підтримує різні джерела файлів (JSON, CSV, Паркет і т.д.). Також має місце змішувати дані з різних джерел.

Архітектура Spark SQL включає в себе наступні компоненти:

1. Program API (Application Program Interface) - Spark SQL може використовувати такі мови програмування, як Java, Python, Scala, R і також надає методи і функції для обробки даних. Також працювати з Hive, підтримуючи SQL-подібні діалекти HiveQL та надаючи доступ до магазину Hive.

2. SchemaRDD - таблиця в не постійному (проміжному) сховищі даних. Вона

використовується для маніпулювання SQL-запитами при розробці додатків на Python, Java, Scala або R. SchemaRDD також дозволяє встановити типи даних, необхідні для створення рамки даних на основі цього. Дані

Так, завдяки можливості маніпулювати та інтеграції модуля Spark SQL фреймворк Apache Spark має досить широкий спектр можливостей для обробки великими даними будь-якої структури. Все це робить ApacheSpark дуже корисним інструментом для обслуговування даних і розробників додатків для великих даних

1.3. Централізована обробка даних у проекті FORWARD

Задіяна розробка Forward, що дає можливість централізованого доступу до різних типів інформації [11]. Проект використовує джерела даних, які підтримують власні мови запитів, такі як Hive, Jaql, Pig, Cassandra, JSONiq, MongoDB, Couchbase, SQL, AsterixDB, BigQuery та UnityJDBC як джерела даних[11]. Отже, відбувається перетворення з мови запиту SQL ++ на мову, що використовується певною базою даних за допомогою адаптерів та спеціального програмного рівня (рисунок 2). Переклади виконуються мовами, які використовують реляційну модель або JSON[11]. Мови запитів, які підтримуються для трансляції в програмі FORWARD, перелічені в таблиці. 1. 1

	Language	Version
A	Hive	0.10.0
B	Jaql	0.5.1
C	Pig	0.11.0
D	CQL (Cassandra)	3.1.5
E	JSONiq	1.0.11
F	MongoDB	2.4.7
G	N1QL (Couchbase)	dev.preview 2
H	SQL	SQL:2011
I	AQL (AsterixDB)	0.8.3 beta
J	BigQuery	Released Mar 25 2014
K	Mongo JDBC	4.2.316

Табл. 1.1 Бази даних, що підтримуються SQL++ [11].

В даній роботі реалізовано підхід, який дає можливість обробляти дані не обовязково зі структурованих надходжених даних, які підтримують власну мову запитів і конкретний вид представлення даних, але і з неструктурованих інформаційних даних (файлів, веб-сайтів і т. д.).

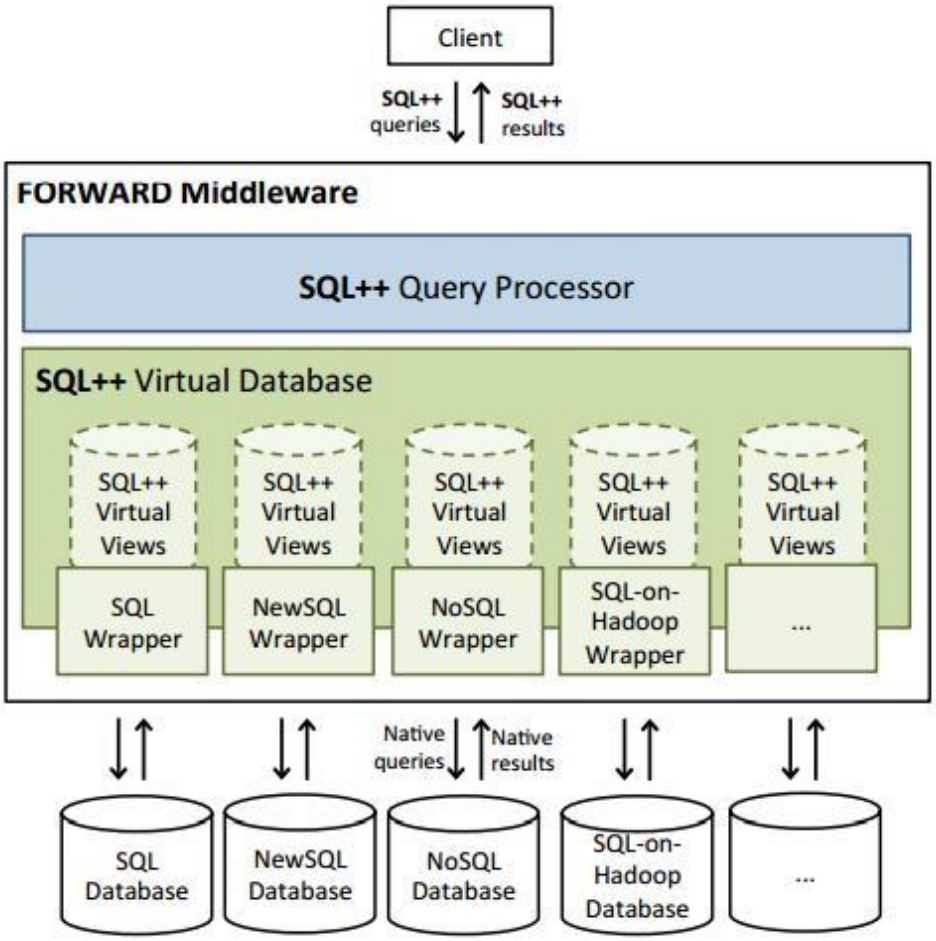

Рис.1.3 Архітектура SQL ++ .

1.4. Шаблон доступу до гетерогенних даних (DAMP)

Ресурс [19] показує підхід вирішення задачі до створення доступу до неоднорідних джерел інформації (CRUD) в запитах дій на (копіювання, отримання, оновлення, видалення). У цій статті ми познайомимо вас з шаблонами дизайну DAMP (Шаблон керування доступом до даних) та його

впровадження в C++. Цей шаблон описує службу для маніпулювання різними інформаційними ресурсами. Його основними компонентами є сесія (сесія), яка надає інтерфейс користувача для маніпулювання сервісом, і медіатор (медіатор), який є обгорткою для виконання певних операцій. Target. У цьому шаблоні користувач використовує клас SessionData для визначення формату запитуваних даних (див. Рис. 1.4).

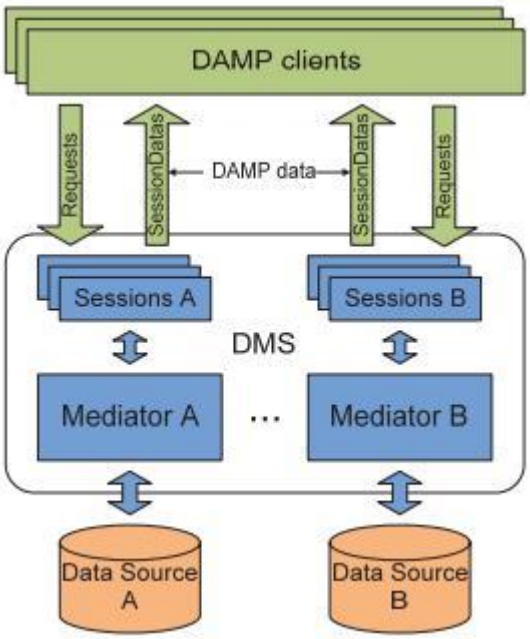

Рис. 1.4 Архітектура системи DAMP.

Хоча в цій роботі використовується концепція використання адаптерів для конкретних джерел і перетворення даних в єдиний формат, описаний вище шаблон зосереджений в першу чергу на виконанні операцій CRUD, і ця робота більшу увагу зосереджує на операцій з використанням реляційної алгебри [10].

1.5. Проект Pentaho

Іншим прикладом системи обробки даних з надходження інформаціїї яка є різнотипною є програмний комплекс для бізнес-аналізу Pentaho. Цей проект розробляє інструменти для звітності при використанні підходу OLAP. Як один з інструментів системного адміністрування, він забезпечує високорівневу мову MDX для написання запитів. Оскільки виконання такого запиту у джерела є дуже тривалим процесом, модулі ETL використовуються для забезпечення можливості швидкого виконання аналітичних операцій. Модуль ETL генерує сховище даних в результаті своєї роботи [1]. Звернемо увагу на те, що середовище Pentaho тримає звязок з базами даних, які підтримують інтерфейс JDBC[16], у роботі задіяно підтримка обробки інформації з різних за типом джерел.

Створення моделі реалізації інформації в неоднорідному середовищі на основі Q-set та збільшеної по можливостях реляційної алгебри

Модель, яка є ядром виконання запитів, розроблена в цій роботі, заснована на основі виконання обробки інформації з різних за типом джерел, як зазначено в статті [10]. Даний підхід фокусується саме на здатності обробляти нечіткі запити. Одним з важливих понять, є показаних є q-set. Це формат запиту в потрійному стилі. Запит, багато об'єктів і функція, яка оцінює релевантність об'єктів. Розширена реляційна алгебра містить в собі такі операції: aggregation, nest, filter, fusion, join, group join.unnest.

1.6. Принципи побудови систем обробки даних

У даному джерелі [10] описується виклад інформації та загальні деталі операції з їх обробки. Реалізація цих концепцій вимагає ідей і принципів для побудови інструментів для обробки різних частин інформації, показаних в [9]. Згідно з цією статтею, інструменти обробки даних можна розкласти на

функціональні компоненти, такі як інструменти написання запитів, інструменти оптимізації запитів і ядра виконання запитів (див. малюнок 1.5).

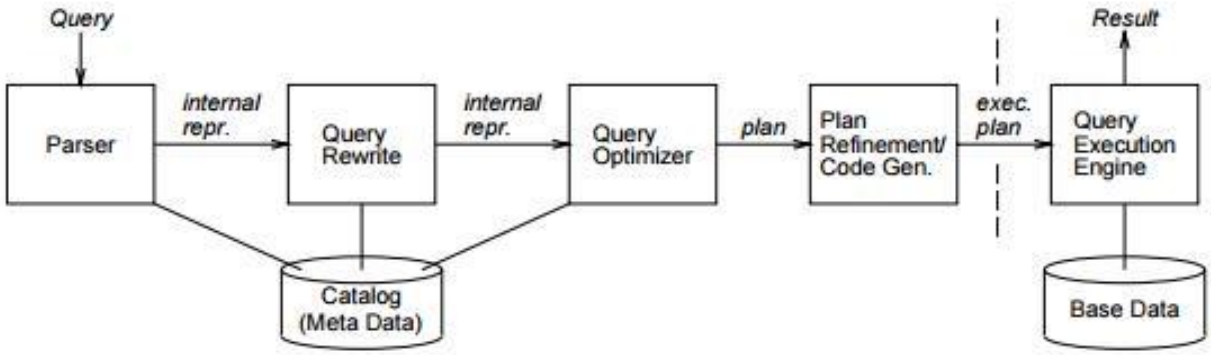

Рис. 1.5 Процес обробки інформації та його складові.

Дане джерело додатково описує проміжний підхід до обробки неоднорідної інформації та перетворення даних у загальний формат (див. малюнок 1.6).

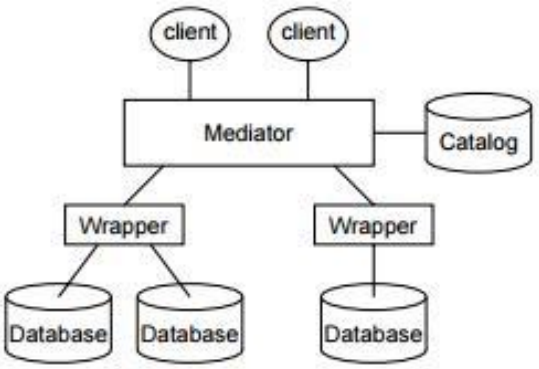

Рис. 1.6 Використання посередника для обробки різнорідної інформації.

Джерело [9] ілюструє важливі аспекти, такі як алгоритми оптимізації планів виконання запитів та деякі алгоритми для виконання конкретних операцій, що використовуються в цій роботі.

2 ПРОЕКТНА ЧАСТИНА

Архітектура виконавця запитів

2.1. Функціональні вимоги до ядра виконання запитів

Потрібні умови до функціональних характеристик при виконанні для запиту.
• програма, що реалізовує запит повиненна підтримувати можливість одержання даних з різних джерел (різнотипних баз даних, матеріалів документообігу, веб-сайтів тощо), а що до вимогів до ядро то воно має бути легко масштабованим для зєднання з різними постачальниками даних.
• програма, що реалізовує запит повиненна підтримувати можливість виконання розширених операцій реляційної алгебри згідно з структурою запуску запиту.
• щоб скоротити час відгуку та забезпечити рівномірне завантаження хостів розподілених систем запитів, потрібна підтримка конвеєрної обробки даних.

2.2. Основні компоненти ядра виконання запитів

Відповідно до наведених вище потрібних вимог, ядро що робить запит дає можливість поділу на три головні функціональні компоненти: На першій позиції компонента, являє собою планувальник, який отримує плани запитів від зовнішніх компонентів системи (або від оптимізатора, або безпосередньо від мовних інструментів). На основі плану визначаються компоненти ядра виконавця, необхідного для виконання цього запиту. Підзапити для цього конкретного компонента, отримані із общого плану запуску запитів, і деяка метаінформація (адреса передачі даних, адреса сповіщення тощо).

Наступний компонент (другий) – є оператор операцій з розширеної реляційної алгебри, представленої у виді розгалуженої структури. Будь-який ктристувач може підтримувати різні операції і алгоритми виконання з використанням існуючиї різноманітних технологій (наприклад, викорирстованих програм з керування баз даних AsterixDB, PostgreSQL, і т. д.). Масштабованість досягається для підтримки нових операцій і різних алгоритмів виконання.

Третій компонент - вихідний адаптер. Він надає інтерфейс для вилучення даних з різних джерел. Таким чином, масштабованість досягається для підтримки нових джерел даних (отже для видалення даних з утвореної бази даних або типу документа достатньо використати цей адаптер).

Якщо зауважити, що деякі джерела (за допомогою баз даних) мають можливість виконувати операції за допомогою власних засобів виконання плану, і дають можливість виконати інтерфейс користувача запиту.

2.3. Основні засади функціонування виконавця запитів

На рис. 2.1 показана схема взаємодії елементів ядра виконання запитів.

Коли планувальник отримує план виконання запиту, тоді запускає запит на виконання.

Планувальник генерує завдання для кожного необхідного джерела та адаптера виконання, включаючи підзапити, і надсилає ці завдання на хост системи.

Потім джерело отримує інформацію відповідно до отриманого завдання, обробляє її та надсилає виконавцю.

Підрядники, як правило, організовані в деревоподібну структуру, де обробка даних трубопроводу організована природним шляхом відповідно до загального плану виконання запитів.

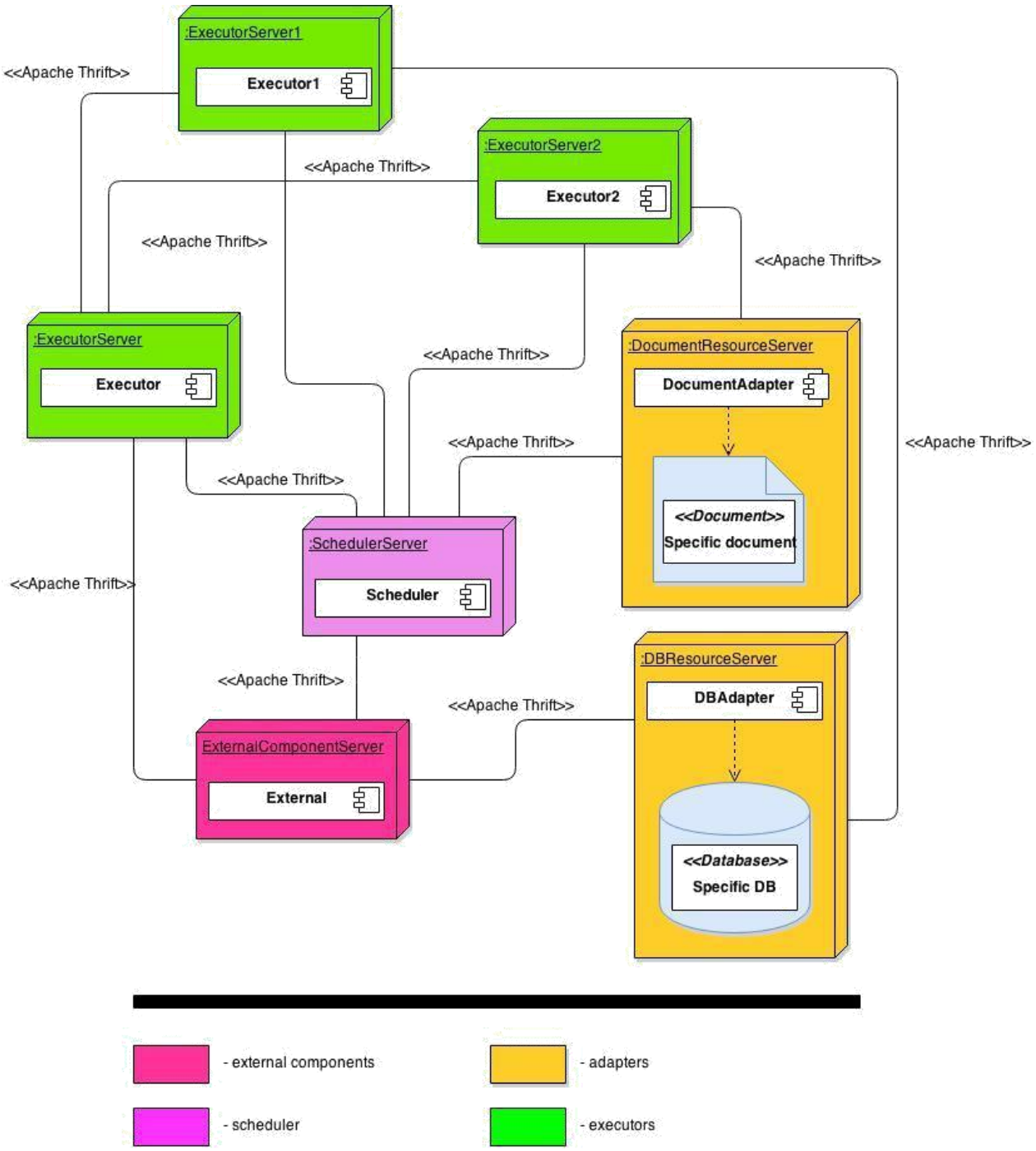

Рис.2.1 Діаграма розгортання ядра виконання запитів.

2.4 Підтримка конвеєрної обробки даних

Паралелізм можна використовувати для скорочення часу відповіді при виконанні запитів у реляційних базах даних. У цьому контексті паралелізм можна розділити на дві лінії [14].

• паралельна обробка незалежних операцій,

• обробка даних конвеєрним способом.

Перша версія паралелізму в межах отриманої архітектури підтримується виконанням незалежних підзапитів різних компонентів розподілених запитів на виконання ядра.

Обробка даних у вигляді конвеєрному виконується під час послідовних операцій як компонент розподіленої архітектури ядра виконання запиту та між окремими виконавцями. На додаток до скорочення часу відгуку, використання конвеєрної обробки даних при виконанні операцій в розширеній реляційній алгебрі [10] може викликати побічні ефекти, такі як збільшення об’єму потрібної пам'яті, необхідної для тимчасового зберігання інформації при запуску певних операцій.

2.5. Організація конвеєрної обробки даних при виконанні операції з'єднання

Звернемо увагу на підтримувані алгоритми виконання конвеєрних операцій з'єднання (join) [8]. Тотого ж можна застережити, що існує кілька типів алгоритмів для виконання даної операції (hash join, sort-merge join, nested-loop join) [6].

Якщо звернути увагу на поступовість дій конвеєрної обробки при реалізації цих алгоритмів єдиною відмінністю є структура, що задіяна у проміжному зберігання записів (впорядковані множини, хеш-таблиці, невпорядковані множини) і спосіб порівняння записів. Далі розглянемо деякі варіанти обробки даних конвеєрного типу при виконанні операцій hash join (для sort-merge join і

nested-loop join описані нижче кроки схожі, аж до наведених вище різностей) [12, 15].

Один з варіантів (перший) послідовності обробки даних конвеєрного типу складається з таких етапів (див. малюнок 7):

• сприйняти всі дані з одного історника та генерації на їх основі хеш-таблиці;

• послідовний пошук і обробка записів з другого джерела, використовуючи хеш-таблицю, створену на першому етапі.

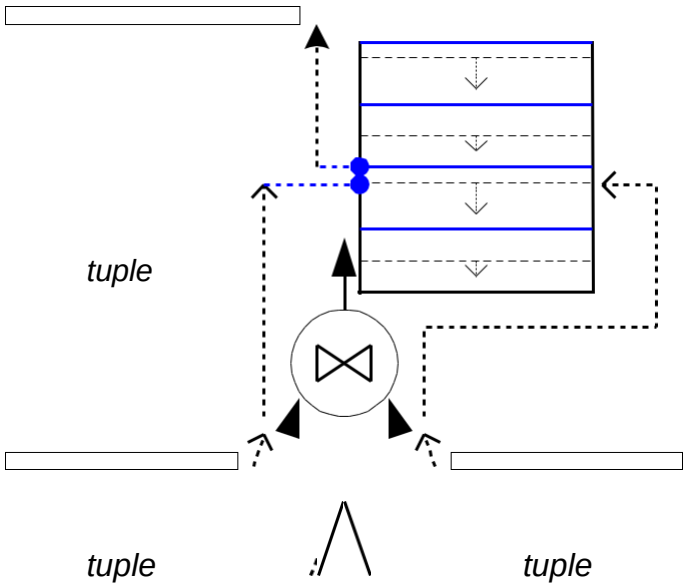

Рис. 2.2 Організація конвеєрного виконання операції hash join #1 .

Наступний спосіб (другий) організації виконання конвеєрного типу для операцій хеш-об'єднання [5] можна охаректирезувати таким чином. Коли запис отримано з джерела, обробка виконується відповідно до даних, доступних у хеш-таблиці, що дорівнює другому джерелу, далі цей запис доплюсовується до хеш-таблиці, яка містить інформацію з першого джерела (див. малюнок.2.3）：

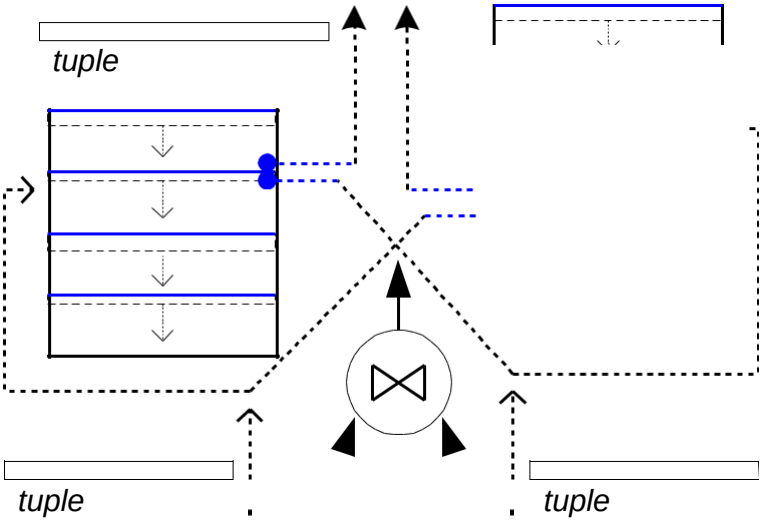

Рис. 2.3 Організація конвеєрного виконання операції hash join #2.

Такий спосіб виконання операції підключення значно скорочує час відгуку і робить виконавця більш рівномірно завантаженим (див. малюнок 2.4). Це збільшує мінімальний обсяг пам'яті, потрібний для виконання цієї операції, оскільки потрібно створити другу хеш-таблицю.

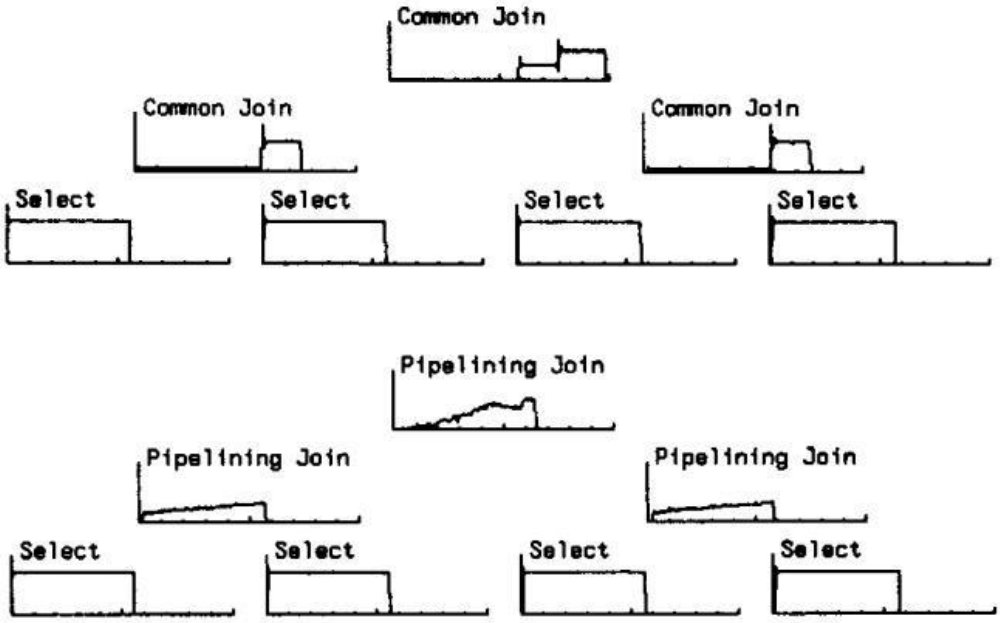

Рис. 2.4 Порівняння розподілу завантаження обчислювача при виконанні операції з'єднання у разі конвеєрної обробки (знизу) та у випадку без конвеєрної обробки (зверху) .

2.6 Організація конвеєрної обробки даних на етапі застосування операції фільтрації

При способі обробки даних конвеєрним типом, аналіз даних при виконанні операцій фільтрації має місце організований (для виконання цієї операції) на послідовне отримання інформації, що значно скорочує час відгуку. При виконанні цієї операції маємо зауважити, що, крім використання конвеєрної обробки, паралельна обробка даних може бути організована через кілька хостів системи для зниження навантаження на деякі використовувані елементи системи та пришвидшення інформаційної обробки.

.

3 СПЕЦІАЛЬНА ЧАСТИНА

3.1 Інтеграція з мовними компонентами та оптимізатором

Для поєднання з словниковими компонентами систем обробки інформації та оптимізаторів потрібно використати подачу запитів у форматі JSON (рисунок 3.1), додатково можна застосовувати передачу запитів та даних за допомогою двійкових транспортних протоколів.

```
select * from Persons where age >= 22 and age <= 23
```

```
"1": {"str": "filterExpression"},
"2": {"i32": 28},
"9": {
    "lst": [
      "rec",
      1,
      {
        "1": {"str": ""},
        "2": {"i32": 15},
        "9": {
          "lst": [
            "rec",
            2,
            {
              "1": {"str": "age"},
              "2": {"i32": 23},
              "5": {"str": "age"}
            },
            {
              "1": {"str": ""},
              "2": {"i32": 22},
              "5": {"str": "30"}
            }
          ]
        }
      }
    ]
  }
```

Рис. 31. Приклад трансляції запиту у форматі JSON.

JSON (“JavaScript Object Notation”) —формат обміну даними у вигляді тексту на основі використання JavaScript. Але в той же час цей формат Є JS-independent і може брати участь у застосунках на різних мовах програмування.

Як значення JSON можна використовувати такі:

• Об'єкти JSON

• Масиви

• Числовий (цілочисельний або дійсний)

• істина (true), хибне (false) і нульове (null)

• рядкова змінна.

Приймають значення:

•Запис — це набір невпорядкованих пар ключ-значення, укладених у фігурні дужки "{}". Ключ записується у вигляді рядків, з символом ":" між ключами та значеннями. Подвійне слово ‘ключ-значення’ , що розділені комами.

• Масив (одновимірний) - це впорядкований набір значень. Масив укладено в квадратні дужки "[]". Результат, що розділені комами. Масив може бути порожнім. Іншими словами, він не повинен бути у відповідних значеннях. Значення в одному масиві є різних типів.

• Числовий (цілочисельний або дійсний).

• Буквальне істинне, хибне і нульове.

•рядок — це впорядкований набір з нуля чи значення збільшення символів Юнікоду, взятих у подвійні лапки. Символи можна вказати за допомогою послідовності виходу, яка починається з зворотної косої риски "" (підтримувані варіанти \ ", \\, \\, \/, \t, \n, \r, \f і \b), або є можливість записати в закодованому шістнадцятковому коді. В Юнікоді як \uFFFF.

Через свою лаконічність у зрівнянні з XML формат JSON може підходити для використання у форматі складних структур. Він використовується у веб-додатках як між переглядачем веб-сторінок і сервером (AJAX), так і при обміні даними між сервером (програмні HTTP застосунки).

Формат JSON є підмножиною синтаксису мови JavaScript, тому можемо застосовувати у форматі за допомогою вбудованої функції JSON. parse().

JSON також має значні позитиви перед другими форматами даних. На відміну від них, вони більш компактні, що дуже важливо при обміні даними через інтернет. Крім того, JSON простіший у використанні і набагато простіше читати і записувати.
У веб-додатку JSON часто використовується як інтерфейс передачі даних з веб-сервера клієнту (веб-браузеру) в запиті AJAX.

Маніпулювання JSON на JavaScript зазвичай здійснюється двома способами.
• перетворення рядків JSON в об'єкти (цей процес також відомий як розбір).
• перетворення об'єкта на рядок JSON (тобто протилежність розбору).

Позитивні сторони формату JSON

Формат що представлений у вигляді даних JSON має такі позитивні сторони:
• Зручні і швидкі методи перетворення (розбору) рядків JSON в об'єкти JavaScript (або навпаки).
• Чітка і зручна у використанні структура даних.
• Дуже малий за розміром у зрівнянні з другими методами зберігання даних (наприклад, XML). За рахунок того, що формат JSON має найменш можливий формат. При його написанні використовується всього кілька спеціальних символів. Це дуже важлива перевага. Дані, що відображаються у форматі JSON, завантажуються швидше, ніж при перегляді в інших форматах.

Завдяки своїм численним перевагам цей формат став використовуватися не обовязково на JavaScript, але в декількох мов програмування, а саме C, Perl, Ruby, Python і PHP.
Порівняння двох форматів JSON та XML
Формат з типом JSON має такі переваги над форматом XML:
1. При передачі інформації, розмір JSON стане меншим, ніж розмір XML.
2. JSON є доступним спосібом для перетворення структур даних JavaScript, в порівнянні з XML.
3. Формат у фигляді JSON простіше згенерувати, ніж XML.

3.2. Інтерфейси взаємодії із зовнішніми компонентами системи

Метод спряження між планувальником і мовним компонентом (або оптимізатором) утворюється через відповідний задіяний інтерфейс (рисунок 3.2).

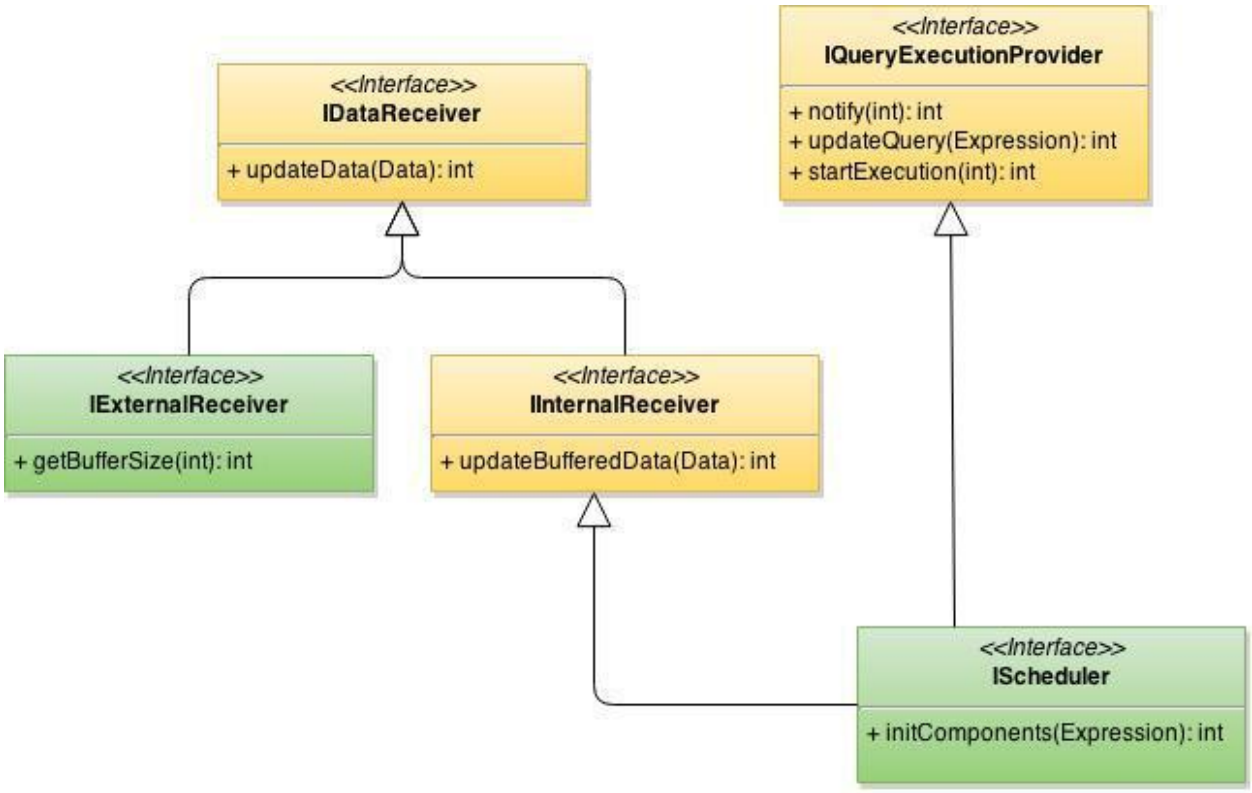

Рис. 3.2 Інтерфейси взаємодії ядра виконання та зовнішніх компонентів системи.

Відповідно, щоб організувати взаємодію між ядром запиту та зовнішніми компонентами, потрібно запустити інтерфейс IScheduler у частині виконавця запиту, який відповідає за одержання та створення запитів від мовного елемента або оптимізатора. Далі потрібно реалізувати інтерфейс IExternalReceiver, щоб деякі зовнішні компоненти системи могли отримувати результати запиту.

Розглянемо докладніше інтерфейси.

- *IDataReceiver*- Компонент інтерфейсу, який може отримувати дані, включаючи наступні методи:
  - *updateData(Data): int*– спосіб отримання даних у єдиному форматі (Data);
- *IExternalReceiver (extends IDataReceiver)*- спосіб обробника даних для зовнішніх, по відношенню до виконавця запитів елементів системи, які визначає метод:
  - *getBufferSize(int): int*– метод, що відтворює розмір буфера передачі даних зовнішньому елементу;
- *IInternalReceiver (extends IDataReceiver)*– інтерфейс обробника даних для внутрішніх елементів виконавця запитів, що виконує метод:
  - *updateBufferedData(int): int*- метод, потрібний для утворення конвеєрної обробки даних між виконавцями;
- *IQueryExecutionProvider*– спільний інтерфейс запитів що обробляються, та містить методи:
  - *notify(int): int*– відправлення оповіщень (потрібно для синхронізації),
  - *updateQuery(Expression): int*– оновлення конфігурації діючого в даний момент запиту на виконання (Expression – послідовність виконання запиту);
  - *startExecution(int): int*- старт обробки діючого запиту, відповідно до потрібної конфігурації;
- *IScheduler (extends IInternalReceiver, IQueryExecutionProvider)*- інтерфейс планувальника, що показує метод:

- *initComponents(Expression): int*- налаштування внутрішніх елементів ядра виконання, відповідно з конфігурацією запиту.

3.3. Уніфікований формат для відображення внутрішніх даних

Представлено форму для узагальненого представлення даних (див. рис.3.3).

Перетворювачі джерел утворюють інформацію до даного виду у наступному виконанні операцій розширеної реляційної алгебри з [10] та віддачі зовнішнім елементам системи обробки інформації.

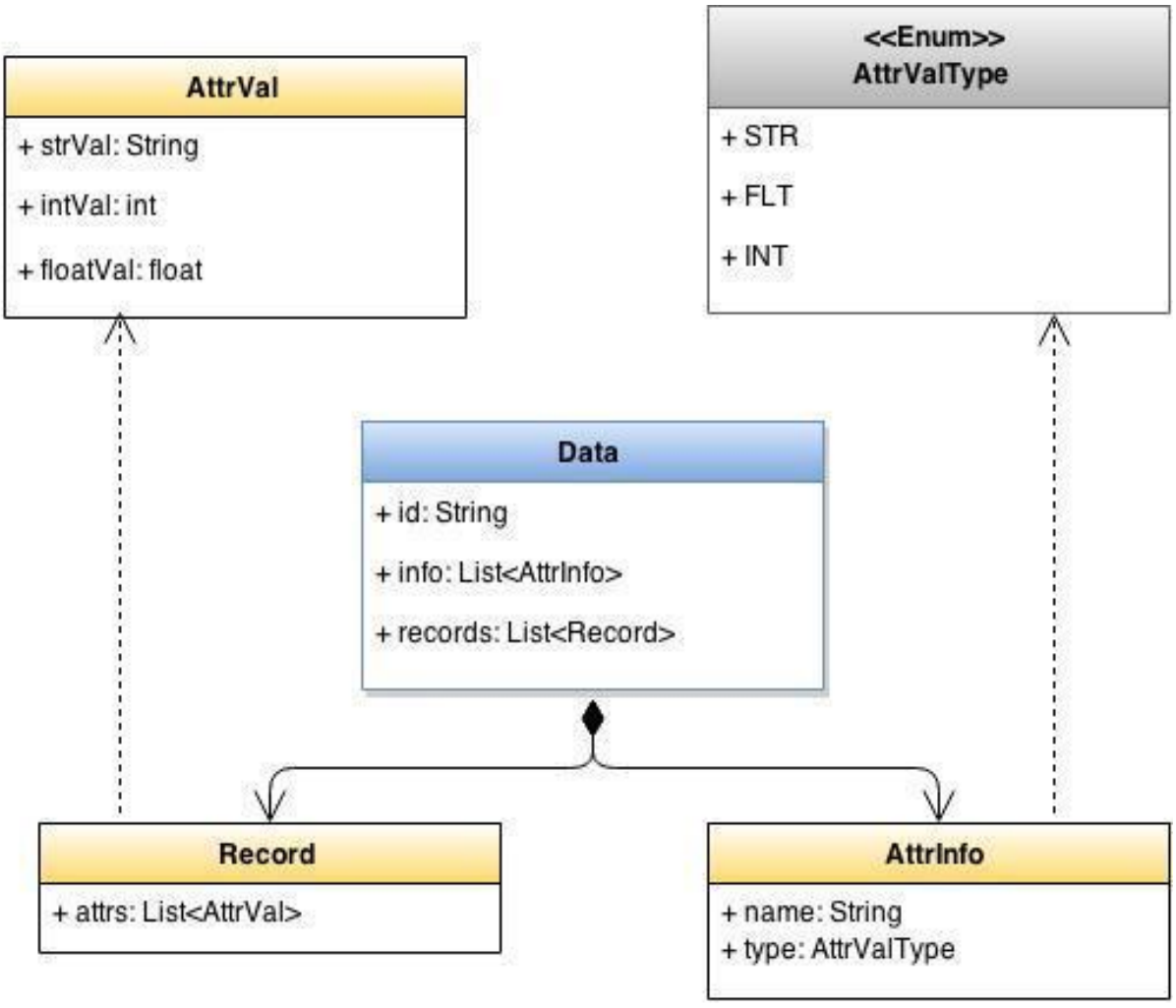

Рис. 3.3 Діаграма класів для уніфікованого формату представлення даних.

Основним класом, що утворює формат даних, є клас Data, який утворює у собі опис мета-дані про структуру кожного запису (клас AttrInfo), а також додатково інформацію, що є у записах (клас Record). У дані ієрархії типів використовується розподіл показу схеми представлення інформації та структури зберігання даних. В такому випадку відпадає необхідність у дублюванні мета-даних про будову частинок переданого відношення, що дає можливість дій до скорочення обсягів пам'яті, потрібних для зберігання інформації та утворення операцій розширеної реляційної алгебри [10], а також дає можливість прискорення передачі даних та зменшення завантаженості комп’ютерної мережі та зниження часу відгуку під час запуску запитів.

3.4 Формат представлення плану дій виконання запиту

Коли відбувається запит від користувача зовнішній елемент системи надсилає планувальнику план послідовності виконання запиту в відповідному форматі (див. малюнок 3.4). Завдання, які планувальник відповідає виконавцю підзапитами, також відображаються в такому вигляді.

Клас Expression — це опис вузлів у розгалуженій структурі дій виконання запиту. Будь-який вузол є описом операції в розширеній реляційній алгебрі [10] і деякою мета-ланими (наприклад, алгоритмом запуску цієї операції). Він також використовує ієрархічну структуру об'єктів цього класу для опису виразів, які задіяні як аргументи для операцій у методі виконання запиту.

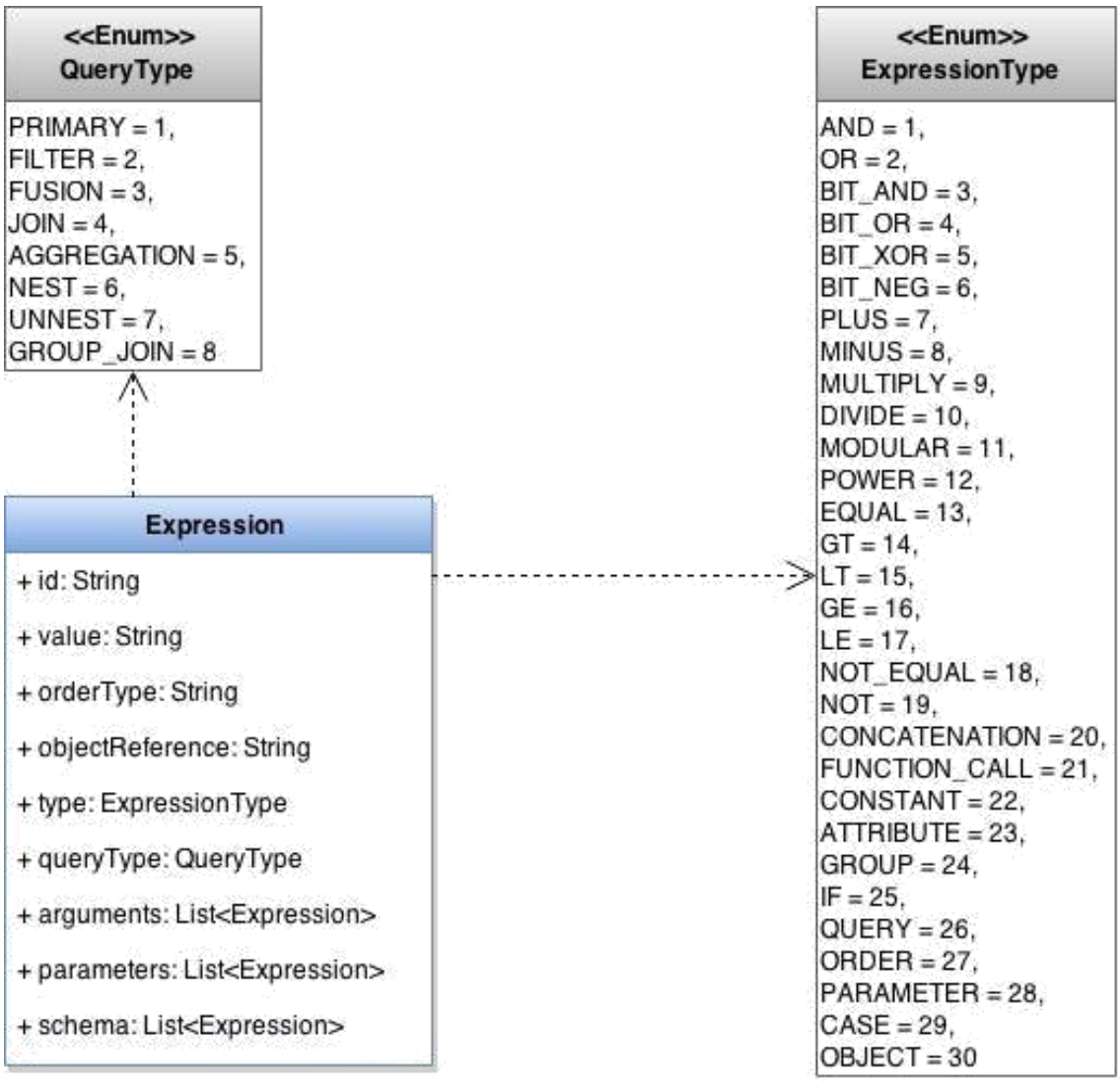

Рис. 3.4 Діаграма класів для уніфікованого представлення плану виконання запиту.

Інтеграція виконавців та адаптерів для джерел

3.5 Інтерфейси виконавців та адаптерів джерел

Щоб реалізувати або інтегрувати наявного виконавця та вихідного адаптера, слід перемістити наявного реалізованого виконавця запиту (наприклад, [11]) до вказаного узагальненого інтерфейсу у виконавиці запиту (рисунок 3.5).

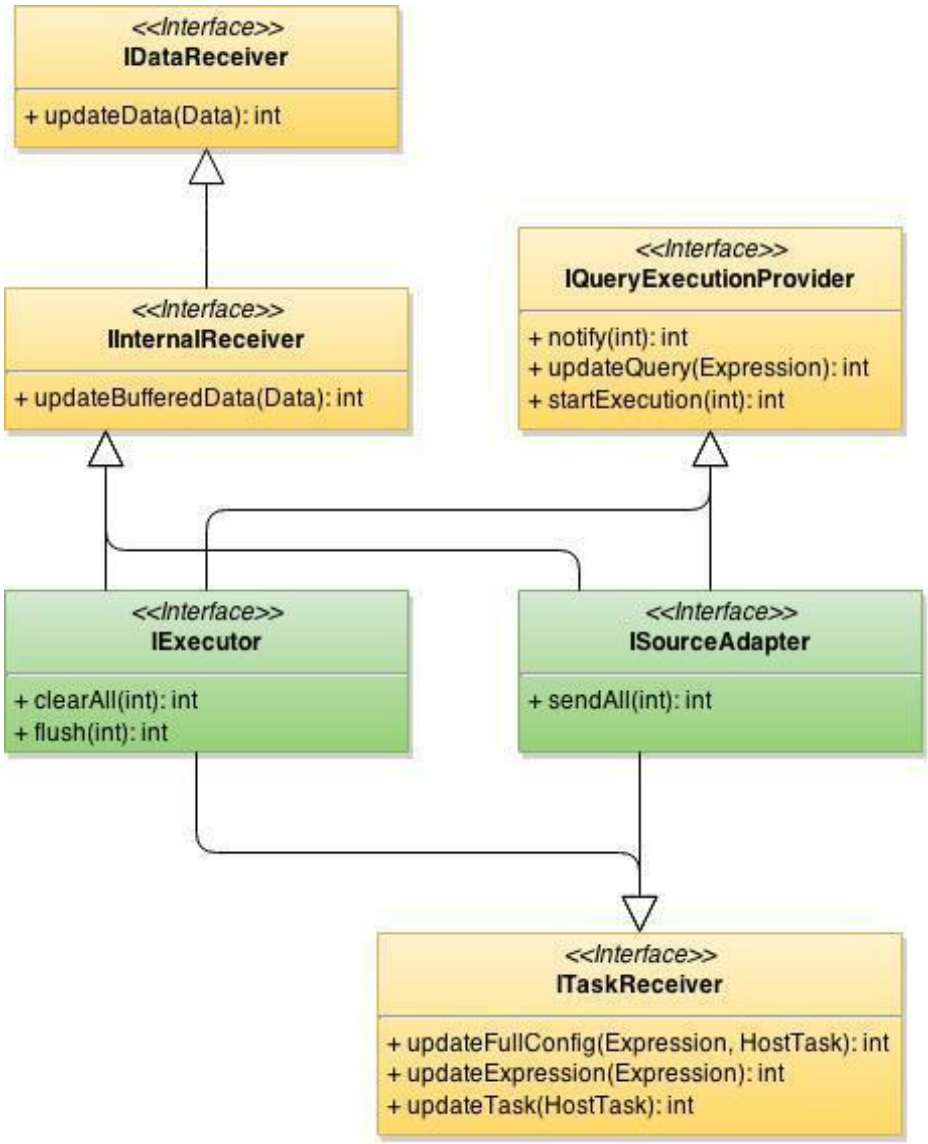

Рис. 3.5 Інтерфейси взаємодії внутрішніх компонентів ядра виконання запитів.

Як показано, щоб під’єднати нового виконавця і вихідного адаптера до ядра запиту необхідно реалізувати інтерфейси IExecutor і ISourceAdapter відповідно. Розглянемо докладніше ієрархію інтерфейсів, показаних на малюнку. :

- *ITaskReceiver*- це інтерфейс, який фіксує можливість передачі цього компонента завдання, такого як план запиту або деяка мета-даних, і містить такі методи:
  - *updateFullConfig(Expression, HostTask): int*– одержання плану запуску запиту та мета-даних (HostTask – див. додаток 3),
  - *updateExpression(Expression): int*– оновлення плану запуску запитів,
  - *updateTask(HostTask): int*- Поновлення мета-даних;
- *ISourceAdapter(extends ITaskReceiver, IQueryExecutionProvider, IInternalReceiver)*- Інтерфейс адаптера джерела, який підтримує метод:
  - *sendAll(int): int*– надісляння всіх наявної інформації відповідно до встановлених мета-даних;
- *IExecutor(extends ITaskReceiver, IQueryExecutionProvider, IInternalReceiver)*- Інтерфейс виконавця запитів, що визначає такі методи:
  - *clearAll(int): int*– видалення інформації зі всіх буферів,
  - *flush (int) :int*- вичислення і переказ всіх накопичених даних, не очікуючи коли буфер заповнеться, з подальшим їх звільненням.

3.6 Подання мета-інформації для конфігурування виконавців

Планувальник налаштовує всі адаптери джерел та виконавець за допомогою мета-інформації, що показана у певному форматі (див. рис. 3.6)

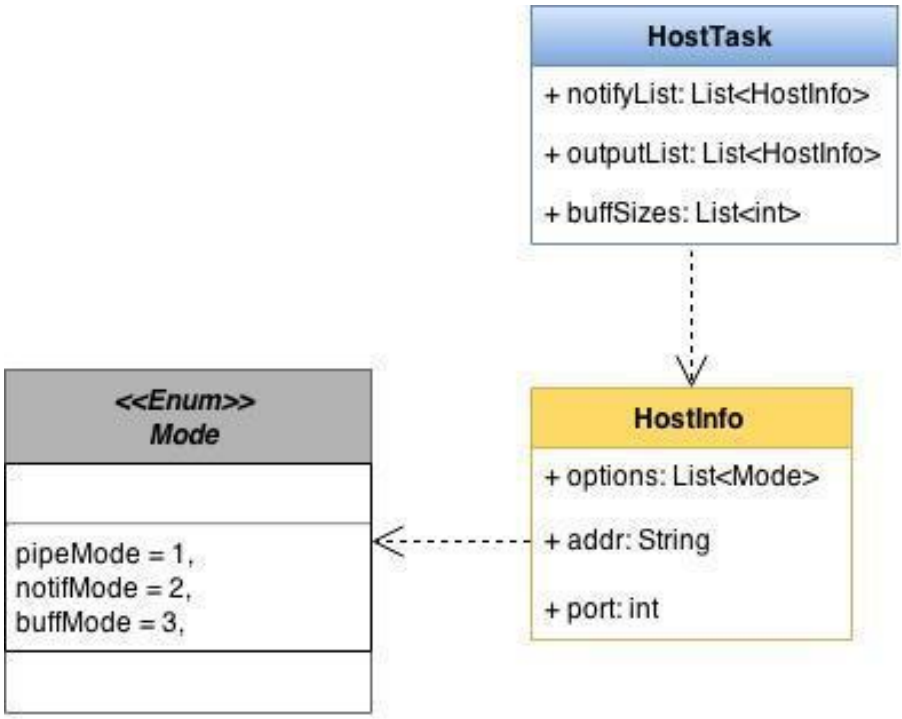

Рис. 3.6 Діаграма класів для уніфікованого представлення мета-інформації внутрішніх компонентів ядра виконання запитів.

Тип HostTask одержує дані про хост, якому передається інформація, хост, який буде сповіщено (для синхронізації при виконанні запиту), і розмір буфера для відправки інформації. Потім дані хоста представлені класом HostInfo. Цей клас містить адреси, номери портів і режими конфігурації для надсилання інформації та оповіщень.

3.7 Практичне застосування

Архітектура прототипу

Отримаємо приклад інтеграції з мовним компонентом і конкретним виконавцем, заснований на демонстраційному прототипі системи (рисунок 3.6).

В якості одного з вихідних і виконавських адаптерів був використаний результат роботи з використання розширених операцій реляційної алгебри на методі AsterixDB [13]. Даний елемент реалізований за допомогою Java і мовних компонентів (вбудований в інтерфейс - ця діаграма (рисунок 3.7). Другі елементи ядра запиту реалізуються в C++. Це ілюструє ще одну особливість реалізації ядра запиту: підтримка впровадження окремих елементів системи з використанням різних технологій, платформ з іншим типом.

Платформи та мови реалізації

Тому розглянутий прототип буде складатися з наступних частин: планувальник (C++), система виконання, яка підтримує операції об'єднання (C++), адаптер для текстових файлів (C++), адаптер для AsterixDB (Java) і зовнішній мовний інструмент Java. Apache Thrift використовується для зв'язку з внутрішніми і зовнішніми компонентами в ядрі. Хости для встановлення прототипів показані наступним чином:

- *JoinExecutorServer*:
  - апаратне забезпечення: Intel Core i7 3630QM, 2.4 GHz, 8GB RAM,
  - програмне забезпечення: x86_64 Ubuntu 14.04 LTS, ядро 3.13;

- *SchedulerServer*:
  - апаратне забезпечення: Intel Core i7 4700MQ, 2.4GHz, 8GB RAM,
  - програмне забезпечення: x86_64 Ubuntu 14.04 LTS, ядро 3.13;

- *TextDataServer*:

- апаратне забезпечення: Intel Core i5 2430M, 2.4 GHz, 4GB RAM,
- програмне забезпечення: x86_64 Ubuntu 14.04 LTS, ядро 3.13;

 AsterixDBServer:

- апаратне забезпечення: Intel Core i5 3470, 3.2 GHz, 6GB RAM,
- програмне забезпечення: x86_64 Ubuntu 14.04 LTS, ядро 3.13;

 FrontEndServer:

- апаратне забезпечення: Intel Core i5 3470, 3.2 GHz, 6GB RAM,
- програмне забезпечення: x86_64 Ubuntu 14.04 LTS, ядро 3.13.

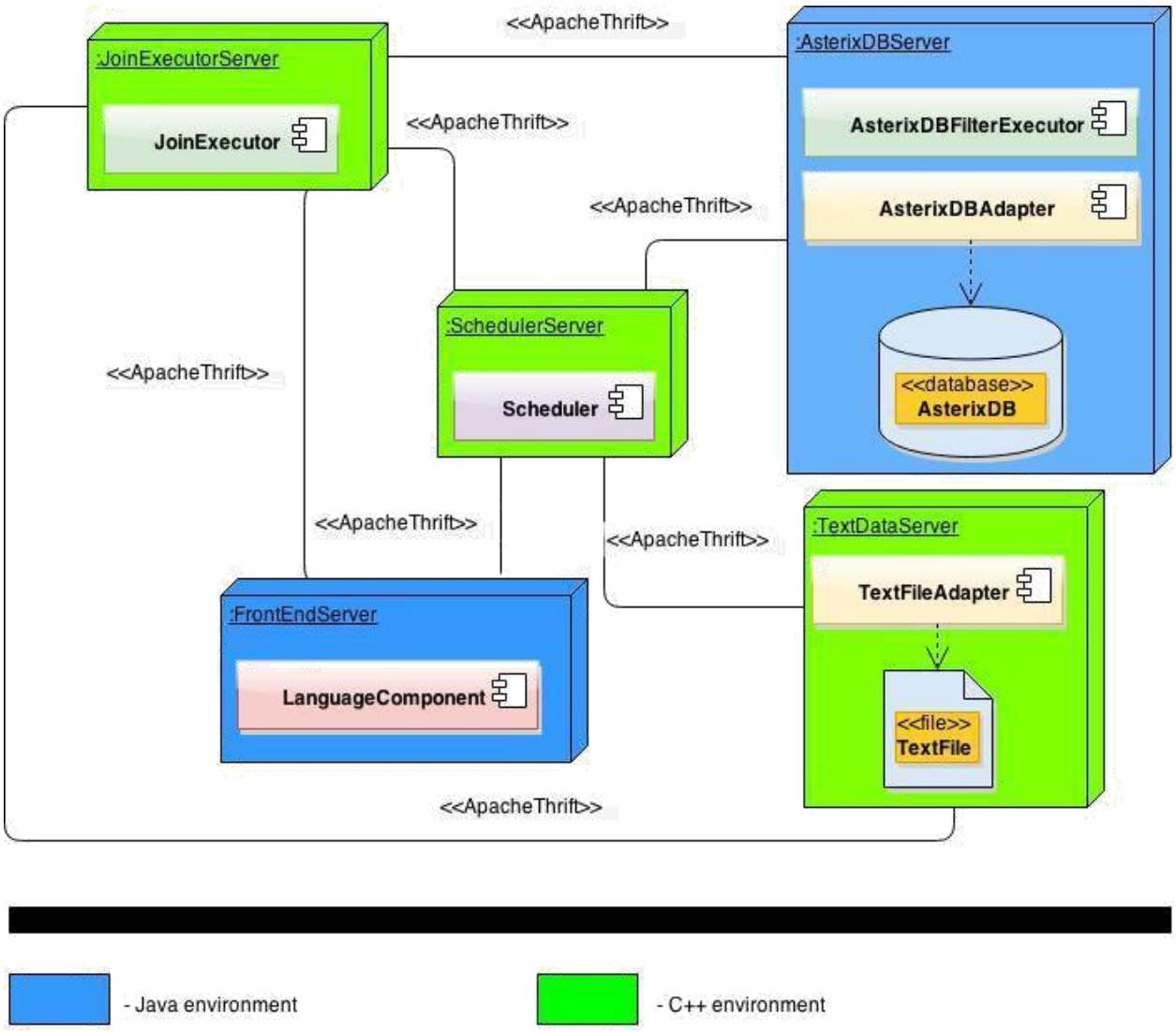

Рис. 3.7 Діаграма розгортання прототипу.

Процес обробки запитів

Подивимось більш докладніше роботу цього прототипу.

Front end генерує запит. На данлму етепі показано його представлення у форматі JSON та надсилання його планувальнику (рис. 3.8).

Планувальник (Joiner, адаптер для Asterix_DB_adapter).

query:>
default
[main] WARN com.viosng.hetqueryfront.CommandLineClient - Use default query: select asterixQuery.name, file.name from file join (select * from P
rsons where age < 30) as asterixQuery on asterixQuery.department_id = file.id
[main] INFO com.viosng.hetqueryfront.CommandLineClient - Start to send to localhost:9060 query:{"1":{"str":""},"2":{"i32":26},"3":{"i32":2},"7"
{"lst":["rec",2,{"1":{"str":"name"},"2":{"i32":23},"5":{"str":"asterixQuery.name"}},{"1":{"str":"name"},"2":{"i32":23},"5":{"str":"file.name"}}
},"9":{"lst":["rec",1,{"1":{"str":"asterixQuery"},"2":{"i32":26},"3":{"i32":4},"8":{"lst":["rec",2,{"1":{"str":"joinType"},"2":{"i32":28},"9":{
lst":["rec",1,{"1":{"str":""},"2":{"i32":22},"5":{"str":"inner"}}]}},{"1":{"str":"onCondition"},"2":{"i32":28},"9":{"lst":["rec",1,{"1":{"str":
"},"2":{"i32":13},"9":{"lst":["rec",2,{"1":{"str":"department_id"},"2":{"i32":23},"5":{"str":"asterixQuery.department_id"}},{"1":{"str":"id"},"
":{"i32":23},"5":{"str":"file.id"}}]}]}]}]},"9":{"lst":["rec",2,{"1":{"str":"file"},"2":{"i32":26},"3":{"i32":1},"8":{"lst":["rec",1,{"1":{"str
:"sourceName"},"2":{"i32":28},"9":{"lst":["rec",1,{"1":{"str":""},"2":{"i32":22},"5":{"str":"file"}}]}]}]}},{"1":{"str":"asterixQuery"},"2":{"i3
":26},"3":{"i32":2},"8":{"lst":["rec",1,{"1":{"str":"filterExpression"},"2":{"i32":28},"9":{"lst":["rec",1,{"1":{"str":""},"2":{"i32":15},"9":{
lst":["rec",2,{"1":{"str":"age"},"2":{"i32":23},"5":{"str":"age"}},{"1":{"str":""},"2":{"i32":22},"5":{"str":"30"}}]}]}]}]},"9":{"lst":["rec",1
{"1":{"str":"Persons"},"2":{"i32":26},"3":{"i32":1},"8":{"lst":["rec",1,{"1":{"str":"sourceName"},"2":{"i32":28},"9":{"lst":["rec",1,{"1":{"str
:""},"2":{"i32":22},"5":{"str":"Persons"}}]}]}]}]}]}]}]}]}]}

Рис. 3.8 Генерація та надсилання запиту на планувальник.

(Scheduler) створює завдання для всіх елементів файлу - Document_adapter, адаптер для Asterix_DB -

Адаптер файлу одержує дані з файлу, надсилає їх на виконавець join (рис. 3.9).

Рис. 3.9 Виконання завдання на адаптері файлу.

Адаптер AsterixDB включає інструменти для виконання запитів, які використовуються як виконавці для фільтрації та отримання відповідних підзапитів разом із завданнями. Потім відфільтровується з використанням AsterixDB і надсилання для дії join.

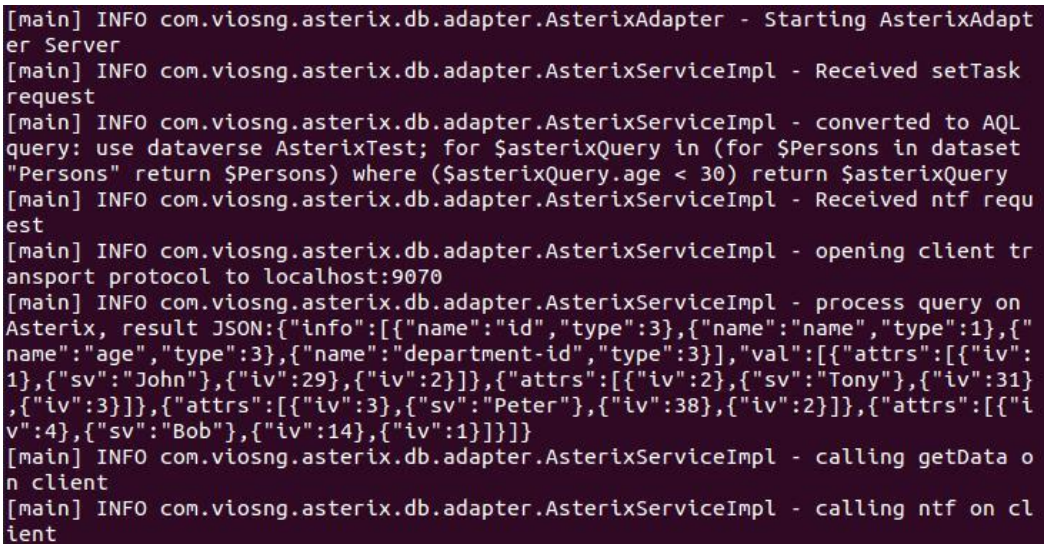

Рис. 3.10 Виконання завдання на адаптері AsterixDB.

На наступному етепі виконується join і результат надсилається на front end (рис. 3.11).

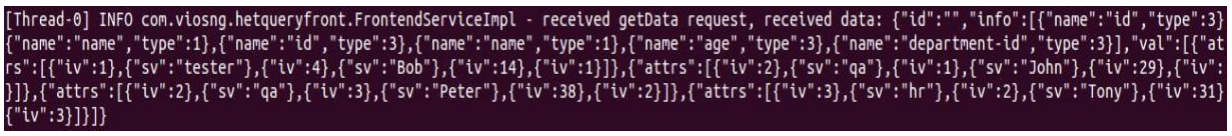
[Thread-0] INFO com.viosng.hetqueryfront.FrontendServiceImpl - received getData request, received data: {"id":"","info":[{"name":"id","type":3}{"name":"name","type":1},{"name":"id","type":3},{"name":"name","type":1},{"name":"age","type":3},{"name":"department-id","type":3}],"val":[[{"attrs":[{"iv":1},{"sv":"tester"},{"iv":4},{"sv":"Bob"},{"iv":14},{"iv":1}]},{"attrs":[{"iv":2},{"sv":"qa"},{"iv":1},{"sv":"John"},{"iv":29},{"iv":}]},{"attrs":[{"iv":2},{"sv":"qa"},{"iv":3},{"sv":"Peter"},{"iv":38},{"iv":2}]},{"attrs":[{"iv":3},{"sv":"hr"},{"iv":2},{"sv":"Tony"},{"iv":31}{"iv":3}]]]}

Рис. 3.11 Отримання результату від виконавця join.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОНОРОНИ ПРАЦІ

4.1 Аналіз шкідливих і небезпечних факторів в обчислювальному центрі

Проведені дослідження виявили зв'язок між роботою на комп'ютері і таких хворобах, як астенопія (послаблення зору), болі у спині та шиї, зап'ясний синдром (хворобливе ураження нерва зап’ястя). Зусилля, що необхідні для того, щоб руки постійно знаходилися паралельно клавіатурі, викликають перевантаження м’язів.

Усі перераховані хвороби прямо або побічно викликані не зовсім зручним сидінням людини за комп'ютером.

Під робочім місцем розуміють положення тіла при роботі, розміщення екрану монітора відносно очей, клавіатури, пози користувача. Форма спинки крісла бажано було б щоб була такою ж самою як форму спини користувача. Крісло треба встановити на такій висоті, щоб не відчувалося тиску на куприк (крісло розміщено трохи занизько) або на стегна (крісло розміщене трохи зависоко). Ще донедавна фахівці з ергономіки вважали, що кут між стегнами і хребтом повинен складати 90 градусів, проте сучасні дослідження показали, що більшість людей віддають перевагу сидіти дещо відкинувшись. Ідеального положення тіла за екраном комп’ютера не існує. Робоче місце повинно бути таким, щоб упродовж робочого дня користувач мав змогу змінювати своє положення. В загальному випадку, робоче місце повинно забезпечувати можливість займати наступні два крайні положення [21].

Перше – з випрямленою спиною, екран монітора повинен знаходитися нижче рівня очей під кутом близько 20° при відстані від екрану до очей 50 см. Друге положення, відкинувшись, при цьому центр екрану знаходиться нижче рівня очей. В цих двох положенням клавіатура та маніпулятор “миша” повинні бути розміщені так, щоб за ними не тягнулися руки при роботі. Ці вимоги

можливо задовільнити використанням крісла з відкидною спинкою із регулюванням висоти.

Пофарбовані у світлі кольори меблі приміщення і великих вікон є додатковими джерелами світла. У занадто світлому помешканні погано видно букви і цифри на екрані монітора. Це викликає головну біль, погіршення зору, зниження концентрації, а також призводить до помилок у роботі через некоректне сприйняття інформації.

Комфортними умовами при роботі на ЕОМ вважаються умови, при яких температура повітря у теплий період року складає 23-25° С, а у холодний 22-24° С, вологість 40-60 %, атмосферний тиск 730-780 мм р. ст.

Існує ряд правил, які необхідно виконувати для забезпечення нормальної працездатності упродовж усього робочого дня.

Найбільш сприятливим є режим роботи, при якому 45 хвилин роботи змінюються 15 хвилинами перерви. При цьому упродовж восьмигодинного робочого дня загальний час роботи за комп'ютером не повинен перевищувати 5-6 годин. Під час перерв користувач може виконувати роботу, що не пов'язана з напругою зору.

Для зменшення зорового навантаження рекомендується примусове часте моргання, відволікання уваги через визначені проміжки часу на інший об'єкт, скорочення тривалості зорової роботи за рахунок виготовлення твердих копій документів та роботи вже з ними.

У залах і кабінетах робочі місця операторів необхідно розташовувати в зоні найкращого бачення інформаційного поля, що повинно забезпечити однозначне сприйняття знакової індикації.

Не менш важливу роль у роботі з програмою є обране сполучення кольорів фону і знаків. За даними багаточисельних дослідів виявлено, що найгіршим сполученням кольорів виявився білий фон і чорні знаки, а найкращим темно-зелений фон і білі знаки.

Жінки з моменту встановлення вагітності до роботи не допускаються.

При роботі на комп'ютері необхідно додержуватися трудової та виробничої дисципліни, виконувати тільки ту роботу, по якій проведено інструктаж, не

допускати присутності на робочому місці сторонніх людей, вміти подати першу допомогу потерпілому на виробництві.

Занальна площа робочої тееторії з комп'ютером повинна містити не менше 6,0 кв. м, кабінет повинен мати систему опалення, кондиціювання повітря.

Комп'ютер повиннен бути розташований не менш як 0,8 м від столу до стіни з вікнами. Відстань між столами, де працюють два працівники повинна бути не менше 1,2 м. Не допускається знаходження іншого робочого місця зі сторони задньої стінки комп'ютера.

Загальна висота поверхні столу, де працює працівник повинна бути у межах 680-800 мм, а клавіатуру потрібно ставити на поверхні стола на відстані 100-300 мм від краю до користувача.

У приміщенні слід здійснювати наскрізну вентиляцію для поліпшення якісного складу повітря.

Основний потік світла повинен бути зліва, не допускається його наявність праворуч, ззаду чи спереду працюючого на комп'ютері. Заборонено застосовувати для вікон чорні штори.

Другим за величиною впливу фактором є випромінювання моніторів комп'ютерів. Види електромагнітних випромінювань електронно-променевих трубок (ЕПТ), що використовуються в моніторах ЕОМ, середні дані замірів, нормовані значення та їх діапазон наведені в таблиці 4.1.

Таблиця 4.1- Види електромагнітних випромінювань ЕПТ

Види випромінювань	Діапазон	Фактичні (середні) дані замірів	Нормовані значення
Рентгенівське	Понад 1.2 КеВ	9-10-12 мкр/г	75.0 мкр/г
Ультрафіолетове випромінювання	220-280 нм 280-320 нм	0 мкр/г 0-0.02 мкр/г	0.01 Вт/м 0.01 Вт/м
Видимий діапазон	320-400 нм 400-700 нм	0.1-2.0 мкр/г 2.5-4.0 мкр/г	10 Вт/м

ІЧ-випромінювання	700 нм-1 мм	0.05-4.0 мкр/г	100 Вт/м
Електричне поле	0 Гц	15 кВ/м	20-60 кВ/м

При роботі з програмою користувач може відчувати навантаження на очі, що пов’язане з особливостями роботи за монітором. Електронно-променева трубка (ЕПТ) - це електронна гармата. Це означає, що ЕПТ заряджена негативно, а отже, поза ЕПТ відбувається накопичення позитивно заряджених часток[22].

Людина почуває себе добре, коли в навколишньому середовищі співвідношення позитивних і негативних іонів майже однаково. Проте перед екраном монітора утворюється надлишок позитивних іонів. В будь-якому помешканні завжди наявні в повітрі мікрочастинки (пилюка, дим, тютюн та інше), які розганяються потоком позитивно заряджених іонів і осідають на самому операторові або його очах.

У результаті такого "бомбардування" в оператора може виникати головна біль, безсоння, подразнення слизової оболонки очей.

Для якісного зображення на екрані необхідно мати у розпорядженні монітор, що підтримує область екрану не менше 800x600 точок, діагональ не менше 14 дюймів, а ще краще – 15, 17. Це забезпечить користувачу бачення досить великої ділянки “поля”, при цьому елементи зображення можуть бути досить великими. Для того, щоб зір користувача при цьому не стомлювався, зображення на екрані не повинно мерехтіти, тому монітор у цьому режимі повинен мати частоту кадрової синхронізації не менше 75 Гц. Відповідно до цих вимог повинен бути підібраний відеоадаптер комп’ютера. Причому від об’єму пам’яті залежить в основному кількість кольорів, що одночасно відображаються на екрані. Тому об’єм відеопам’яті повинен бути не менше 1Мб, для одночасного відображення 65536 кольорів.

Екран монітора – це джерело світла. Тому необхідно організувати освітлення та розташувати його так, щоб у полі зору користувача не було інших

яскравих джерел світла, а також освітленість екрану не збільшувалась за рахунок будь-якого стороннього джерела світла, наприклад лампи на стелі або сонячного світла.

Рентгенівське випромінювання шкідливо впливає на кісткові тканини і кровотворні функції кісткового мозку, інфрачервоне – на зір, втомлюючи очі, при тривалій дії порушує нормальне сприйняття оком кольору.

В таблиці 4.2 наведені скарги користувачів ЕОМ.

Таблиця 4.2 – Характеристика скарг користувачів ЕОМ, що працюють з ЕПТ, порівняно з контрольною групою

Симптоми втоми	Відсоток від кількості досліджуваних			
	Користувачі ЕПТ			Мікромонтаж-ниці
	інженери-програмісти	оператори	Учні	
Загальна втома	29.4	22.2	38	97
Млявість	17.6	18.5	7.7	43
Сонливість	11.8	7.4	11.5	0
Головний біль	17.6	18.5	15.4	54
Важкість у голові	11.8	11.1	3.8	51
Втома м’язів рук	5.8	14.8	15.4	36
Різь в очах	58.8	37	7.7	67
Розпливчатість меж	35.3	55.5	19.2	4
Сумарно-очні симптоми	94.1	92.6	26.9	71

В умовах навчального закладу при відсутності роботи машин і механізмів, що є джерелами шуму, зазвичай рівень шуму і вібрацій є в нормі.

При закритому кожусі системного блоку джерела шуму на робочому місці користувача ЕОМ – це працюючий принтер. Рівень шуму залежить від типу принтера і для матричного складає приблизно 35-40 дБ (в залежності від моделі принтера), що перевищує межу допустимого рівня, значення якого 40 дБ, у лазерних та струйних принтерах шумність значно нижче. Підвищений рівень шуму викликає труднощі у розпізнаванні кольорових сигналів, знижує швидкість сприйняття, гостроту зору, зорову адаптацію, порушує сприйняття візуальної інформації, знижує спроможність швидко і чітко виконувати координовані дії, зменшує на 5-10% продуктивність праці. Тривалий вплив підвищеного рівня шуму з рівнем звукового тиску 90 Дб знижує продуктивність праці на 30-60 %. Медичні обстеження інженерів-системотехніків показали, що крім зниження продуктивності праці високі рівні шуму призводять до стомлення, погіршення слуху. Крім того, при загальній дії підвищений рівень шуму викликає порушення ритму серцевої діяльності, зміну кров'яного тиску, погіршення органів дихання. Втома інженера-системотехніка прямо залежить від ступеня напруженості процесів, що супроводжує зорове сприйняття. Це такі процеси, як адаптація (пристосування ока до зміни освітлення), акомодація (пристосовуваність ока до зміни відстані) і конвергенція (властивість очей зосереджувати зір на предметі).

Усю енергію, що випромінюється джерелами світла, можна поділити на ультрафіолет, видиме світло й інфрачервоне випромінювання. Рентгенівське випромінювання шкідливо впливає на кісткові тканини і кровотворні функції кісткового мозку. Інфрачервоне випромінювання шкідливо діє на зір, втомлюючи очі, при тривалій дії порушує нормальне сприйняття кольору оком.

Частина енергії, що сприймається оком як світло, називається світловим потоком F [люмен]. Сила світла (I) – визначається як відношення світлового потоку до тілесного кута, у котрому рівномірно поширюється цей світловий потік. Вимірюється сила світла в канделах ([Кд]=[лм] /[срад]). Освітленість (E) - поверхнева щільність світлового потоку, люкс [лк]: $E = \frac{F}{S}$,

де S - площа [$м^2$].

Штучне освітлення виробничих помешкань буває:

- загальне;
- комбіноване;
- аварійне.

Крім штучного, в помешканні є також природне освітлення з вікон. Таким чином, таке освітлення можна назвати змішаним.

Система загального освітлення призначена не тільки для освітлення робочих поверхонь, але і для всього помешкання в цілому. Система комбінованого освітлення містить елементи загального та освітлення робочого місця: світильники місцевого освітлення дають можливість добре освітити робочій поверхні, а загального вирівнюють яскравість у полі зору. Використання одного місцевого освітлення не припускається через різкий контраст у яскравостях.

Система аварійного освітлення передбачається у випадках, коли виникають неприпустимі перерви в робочому циклі.

Більш яскраве джерело світла у полі зору користувача буде осліплювати його. Для читання інформації з екрану необхідно більше напружувати зір. Підвищена освітленість екрану розмиває зображення оригиналу на сітківці ока.

Щоб уникнути цього, необхідно:

1) Розмістити монітор екраном перпендикулярно до вікна;
2) Приміщення повинно мати штори, які затримують пряме сонячне світло;
3) При неможливості усунути блікі на екрані монітора його переміщенням в інше місце, необхідно застосувати захисні фільтри, що дозволяють виключити відбиття світла та збільшити контрастність;
4) Рівень освітленості приміщення повинен лежати у діапазоні 210-540 лк, освітленість на робочому місці користувача – 400 лк;
5) Яскравість монітора повинна бути приблизно у 5 разів вище яскравості поверхонь, що оточують користувача;

6) При наявності у помешканні інших робочих місць, що не використовують ЕОМ, необхідно використовувати комбіноване освітлення цього приміщення.

4.2 Забезпечення нормальних санітарних умов праці на робочому місці

При плануванні та організації робочого місця оператора, що обслуговує пульт контролю і управління, згідно інженерної психології та санітарних норм, розробка пульта проводиться у відповідності до правил конструювання.

АРМ розміщується у зручному місці. Місця розміщення обладнання вибирають в залежності від функціональних вимог.

2. Органи управління встановлюються в легкодоступній центральній зоні пульта, щоб кожний з них достатньо відокремлювався, був зручний при використанні, правильно розташований відносно приладу, на показники якого він впливає.

3. Монітори комп’ютерів орієнтують так, щоби кількість бліків була мінімальна, а відстань до очей оператора оптимальна.

4. Колір тих ділянок робочих місць, де зосереджена вся увага оператора, грає велику роль в забезпеченні надійності роботи. Кольори забарвлення приладів, органів управління та фон панелі слід підібрати спокійними, щоб не відбувалося подразнення очей оператора. Для створення світлового комфорту для оператора необхідно використовувати природнє освітлення.

ВИСНОВКИ

В ході розробки даної роботи були отримані такі результати:

- розроблено архітектуру ядра виконання запитів;
- забезпечена можливість інтеграції з мовними компонентами та оптимізатором:
- розроблені інтерфейси взаємодії із зовнішніми компонентами системи,
- розроблено уніфікований формат для внутрішнього застосування даних,
- розроблено формат для представлення плану виконання запиту;
- забезпечено підтримку конвеєрної обробки даних у рамках розробленої архітектури;
- забезпечено можливість інтеграції нових виконавців та адаптерів джерел:
- розроблені інтерфейси для взаємодії з виконавцями та адаптерами для джерел інформації,
- розроблено формат конфігурування компонент системи;
- зроблена апробація (див. розділ апробація) розробленої архітектури при взаємодії з реалізованим раніше виконавцем/джерелом даних на основі AsterixDB [13] та високорівневими мовними засобами для опису запитів. Внаслідок чого було створено прототип системи із тестовим покриттям, що свідчить про виконання функціональних вимог до архітектури ядра виконання запитів (див. розділ архітектура виконавця запитів);

- результати виконання даної дипломної роботи представлені на ISSAT2015.

Слід зазначити, що прототипи ядра виконання запитів, реалізовані на основі даної архітектури, перевищують продуктивність виконавців, заснованих на концепції MapReduce (наприклад, на основі Apache Hadoop [17]), через можливість більш гнучкого управління стратегіями матеріалізації. Також через підтримку бінарної серіалізації даних буде спостерігатися і перевага над системами, що використовують формат JSON для внутрішнього представлення інформації.

Надалі, у межах розробки аналізованої системи, плануються такі кроки:

- інтеграція з оптимізатором запитів,
- розробка адаптерів для підтримки відкритих інтерфейсів, таких як ODBC, JDBC, ADO, та ін.,
- пошук та впровадження найбільш оптимальних стратегій для матеріалізації даних [4],
- дослідження алгоритмів виконання нечітких запитів у рамках розширеної реляційної алгебри,
- дослідження можливості використання методів машинного навчання для виконання запитів.

ПЕРЕЛІК ПОСИЛАНЬ

1. U. Dayal, S. Chaudhur, An Overview of Data Warehousing and OLAP Technology, ACM Sigmod Record, March 1997, pages 65-74.

2. N. Dalvi, SK Sanghai, P. Roy, і S. Sudarshan, Pipelining в Multi-Query Optimization, в PODS, Journal of Computer and System Sciences, 2003, pages: 728–762.

3. J. Gantz, D. Reinsel, Big Data, Bigger Digital Shadows, і Biggest Grow th in the Far East, The Digital Universe in 2020, p 1 – 14, URL: https://www.emc.com/collateral/ analyst-reports/idc-digital-universe-united-states.pdf.

4. M. Grund, J. Krueger, M. Kleine, A. Zeier, H. Plattner, Optimal Query Operator Materialization Strategy for Hybrid Databases, DBKDA 2011, The Third International Conference on Advances in Databases, Knowledge, and Data Applications, 2011, pages : 169 - 174.

5. P. Garcia, HF Korth, Pipelined Hash-Join на Multithreaded Architectures, DaMoN '07 Процедури 3-го міжнародного workshop on Data management on new hardware, 2007.

6. Z. Ives: Query execution techniques, CSE 544, Spring 2000, URL: http://courses.cs.washington.edu/courses/cse544/00sp/lectures/ppt/l5.ppt.

7. Y. Ioannidis, Query optimization, ACM Comput. Surv. vol. 28, no. 1, 1996, pages. 121-123.

8. J. Jayashree, C.Ranichandra, Join algoritm for efficient query processing for large datasets, AJCSIT, 2012, pages 31-35.

9. D. Kossmann, State of the Art in Distributed Query Processing, ACM Computing Surveys (CSUR) Surveys Homepage archive, Volume 32 Issue 4, Dec. 2000, Pages 422-469.

10. B. Novikov, A. Vassilieva, A. Yarygina, Querying Big Data. In CompSysTech,12 Процедури 13-ї міжнародної конференції комп'ютерних систем та технологій, сторінок 1-10, 2012.

11. K. Ong, Y. Papakonstantinou, R. Vernoux, The SQL++ Unifying Semi-structured Query Language, і expressiveness Benchmark of SQL-on-Hadoop, NoSQL і NewSQL Databases, cs.DB, 19 Mar 2015 p 1-28.

12. A. Shatdal, C. Kant, JF Naughton, Cache Conscious Algorithms для Relational Query Processing, VLDB '94 Proceedings of 20th International Conference on Very Large, 1994, pages 510-521.

13. N. Saveliev, Implementation of generalized relations algebraic operations with AsterixDB BDMS, New Trends in Database and Information Systems II Advances в Intelligent Systems and Computing Volume 312, 2015, pages: 323-332.

14. A. Wilschut, P. Apers, Pipelining in query execution, PARBASE-90, 1990.

15. H. Zeller, J. Gray, An Adaptive Hash Join Algorithm for Multiuser Environments, VLDB '90 Процедури 16-ї міжнародної конференції на Very Large Data Bases, pages 186-197, 1990.

16. Embedding the Pentaho Reporting Engine, URL: http://www.ambientelivre.com.br/downloads/doc_download/83-embedding-pentaho-reporting-engine.html.

17. Extract, Transform, і Load Big Data with Apache Hadoop, URL: https://software.intel.com/sites/default/files/article/402274/etl-big-data-with-hadoop.pdf.

18. Pentaho Analysis Viewer User Guide, URL: http://www.osbi.fr/wp-content/Pentaho-Analysis-Viewer-User-Guide.pdf.

19. С.В.Скударнов, Система DAMP: уніфікований доступ до гетерогенних даних у розподілених додатках, Системне

програмування. Том 4, вип. 1: Зб. статей/За ред. А.Н.Терехова, Д.Ю.Буличова. - СПб.: Вид-во СПбГУ, 2009, с.:150-170.

20. Риза С., Лезерсон У., Оуэн Ш., Уиллс Д. Spark для профессионалов. Современные паттерны обработки больших данных. Изд: Питер Пресс, 2017-272с.

21. ГОСТ 12.1.003 - 83 *. ССБТ. Шум. Загальні вимоги безпеки. - Введ.01.07.84

22. ГОСТ 12.1.006 - 84.ССБТ. Електромагнітні поля радіочастот. Допустимі рівні на робочих місцях і вимоги до проведення контролю. - Введ.01.01.85.