Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Розробка та дослідження методів оптимізації запитів
в розподілених базах даних
(тема)

Виконав:
студент 2 курсу, групи СШІм-19-2
Зінін Б.І.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту
(повна назва спеціалізації)

Керівник проф. Філатов В.О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Зініну Богдану Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка та дослідження методів оптимізації запитів в розподілених базах даних

затверджена наказом університету від 29 березня 20 21 р. № 390Ст

2. Термін подання студентом роботи до екзаменаційної комісії _____ _____ 20 ___ р.

3. Вихідні дані до роботи документація мови виконання запитів до баз даних SQL, СУБД Oracle, інші Інтернет-джерела та література з вказаної теми

4. Перелік питань, що потрібно опрацювати в роботі

1) Аналіз предметної галузі

2) Моделі та методи підвищення ефективності розподілених баз даних

3) Розробка та дослідження моделі SQL-запиту у реляційних структурах

4) Плани виконання запитів

5) Аналіз користувацьких запитів інформаційної технології оптимізації структури БД

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Рисунок 1 – Побудова плану виконання запиту в методі оптимізації по синтаксису; Рисунок 2 – Вибір найкращого плану; Рисунок 3 – Приклади розподілів даних; Рисунок 4 – Дерево парсингу запиту SQL

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Основна частина	проф. Філатов В.О.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	29.03.2021	виконано
2	Аналіз літератури та Інтернет-джерел	30.03.2021–04.04.2021	виконано
3	Постановка задачі оптимізації запитів, узгодження з	05.04.2021–12.04.2021	виконано
	керівником		
4	Дослідження методів оптимізації запитів у розподіле-	13.04.2021–20.04.2021	виконано
	них базах даних		
5	Розробка рекомендацій щодо написання ефективних	21.04.2021–28.04.2021	виконано
	запитів		
6	Реалізація прикладів оптимізації запитів	29.04.2021–08.05.2021	виконано
7	Оформлення пояснювальної записки	10.05.2021–12.05.2021	виконано
8	Подання роботи на рецензію	13.05.2021	виконано
9	Попередній захист	14.05.2021	виконано
10	Захист роботи		

Дата видачі завдання 29 березня 2021 р.

Студент _____
(підпис)

Керівник роботи _____ _____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 91 с., 4 рис., 2 дод., 39 джерел.

ВАРТІСТЬ, ЗАПИТ, З’ЄДНАННЯ, ІНДЕКС, КЛАСТЕРИЗАЦІЯ, ОПТИМІЗАЦІЯ, СЕЛЕКТИВНІСТЬ, СЕМАНТИКА, СТАТИСТИКА, ХІНТ, COST, FROM, ORACLE, RULE, SELECT, SQL, WHERE.

Об'єктом дослідження є методи оптимізації запитів в розподілених базах даних.

Метою роботи є проведення дослідження методів оптимізації виконання запитів в розподілених системах управління базами даних та розробка рекомендації по написанню ефективних SQL запитів.

Методи дослідження: аналіз літератури та інтернет-джелел.

У даній роботі розглянуті проблеми та рішення в області оптимізації запитів в СКБД. Порушені проблеми перетворення запитів, проблеми семантичної і логічної оптимізації, проблеми вибору і оцінки вартості альтернативних планів виконання запитів. Виконано аналіз існуючих методів оптимізації виконання запитів. Детально розглянуто питання побудови послідовності з'єднання відносин в запитах реляційної бази даних. На практиці продемонстровані результати застосування деяких методів оптимізації. Дано рекомендації для написання ефективних запитів.

РЕФЕРАТ

Пояснительная записка: 91 с., 4 рис., 2 прил., 39 источников.

ЗАПРОС, ИНДЕКС, КЛАСТЕРИЗАЦИЯ, ОПТИМИЗАЦИЯ, СЕЛЕКТИВНОСТЬ, СЕМАНТИКА,СОЕДИНЕНИЕ, СТАТИСТИКА, СТОИМОСТЬ, ХИНТ,COST, FROM, ORACLE, RULE, SELECT, SQL, WHERE.

Объектом исследования являются методы оптимизации запросов в распределенных базах данных.

Целью работы является исследование методов оптимизации выполнения запросов в распределенных системах управления базами данных и разработка рекомендаций по эффективному написанию SQL запросов.

Методы исследования: анализ литературы и интернет-источников.

В данной работе рассмотрены проблемы и решения в области оптимизации запросов в реляционных СУБД. Затронуты проблемы преобразования запросов, проблемы семантической и логической оптимизации, проблемы выбора и оценки стоимости альтернативных планов выполнения запросов. Выполнен анализ существующих методов оптимизации выполнения запросов. Подробно рассмотрен вопрос построения последовательности соединения отношений в запросах реляционной базы данных. На практике продемонстрированы результаты применения некоторых методов оптимизации. Даны рекомендации для написания эффективных запросов.

ABSTRACT

Explanatory note: 91 p., 4 fig., 2 ann., 39 sources.

CLUSTERING, CONNECTION, COST, FROM, INDEX, OPTIMIZATION, ORACLE, QUERY, RULE, SELECT, SELECTIVITY, SEMANTIC, STATISTIC, SQL, WHERE.

Object of research are methods of query optimization in distributed databases.

The aim is to study the optimization methodsof query execution in distributed database management systems, and development of recommendations for writing efficient SQL queries.

In this paper considered the problems and solutions in the area of query optimization in relational databases. The problems of conversion requests, problems of semantic and logical optimization, problem selection and evaluation of the cost of alternative query execution plans were considered. The analysis of existing methods to optimize query performance was performed. Considered in detail the issue of constructing a sequence of connection relationships in a relational database queries. In practice, the results of some optimization methods were demonstrated. Recommendations for writingefficient queries were developed.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ББД – багатовимірна база даних;

БД – база даних;

БКО – багатокритеріальна оптимізація;

ІС – інформаційна система;

КІС – корпоративна інформаційна система;

ЛПР – людина, що приймає рішення;

МАІ – метод аналізу ієрархій;

ОДД – оперативне джерело даних;

ПЗ– програмне забезпечення;

РБД – розподілена база даних;

РКІС– розподілена корпоративна інформаційна система;

СД – сховище даних;

СКБД – система керування базами даних;

SQL – Structured Query Language – мова структурованих запитів.

ВСТУП

Широке використання інформаційних технологій багаторівневих, територіально розосереджених та розподілених корпоративних інформаційних систем (РКІС) на основі баз і сховищ даних, у тому числі розподілених, обумовлює необхідність забезпечення певної автономності роботи користувачів, зниження навантаження на сервери баз даних (БД) та сервіси синхронізації даних шляхом оптимізації структури БД віддаленого вузла корпоративної інформаційної системи (КІС). Під розподіленою базою даних (РБД) розуміємо сукупність взаємопов'язаних баз даних, розподілених у мережі (у тому числі Інтернет). Основна функція розподілених систем керування базами даних (СКБД)– координування спільної роботи багатьох користувачів з розподіленою інформацією. Розв'язання задачі автономності роботи користувачів розподіленої системи створює багато специфічних проблем в організації баз даних, оскільки різні користувачі можуть працювати паралельно з одними й тими самими даними, виконуючи з ними різні перетворення.

Ключовим фактором, що впливає на надійність і доступність БД, є локалізація посилань. Про високий ступінь локалізації посилань свідчить представлення на поточному вузлі тих даних, що викликаються винятково його користувачем. Комбінована стратегія розподілу даних є найбільш привабливою, але при її використанні, окрім задачі синхронізації дубльованої інформації, актуальною постає задача оптимального проєктування структури БД з точки зору приналежності даних до вузла РКІС.

Комбінована стратегія розподілу даних є найбільш виправданою із точки зору можливості поєднання переваг стратегій з та без дублювання. Але при її використанні, окрім задачі синхронізації дубльованої інформації, актуальною постає задача оптимального проєктування структури БД з точки зору приналежності даних до категорії того чи іншого вузла мережі. Крім

того, продуктивність системи напряму буде залежати від прийняття рішення щодо необхідності часткового або повного дублювання даних.

Відношення БД представляються на вузлі РКІС після застосування операцій проєкції та вибірки. Тобто для оптимального представлення даних необхідно використати елементи вертикальної та горизонтальної фрагментації даних. Імітаційні моделі РКІС зазвичай дуже спрощені та не враховують ряд важливих факторів, що впливають на її поведінку. У свою чергу накопичення статистичних даних виконання SQL-запитів до розподіленої БД на реально працюючих серверах дає змогу врахувати всі особливості РКІС.

Наведені питання розглядались у роботах таких вітчизняних вчених, як: Стогній А. О., Пасічник В. В., Шаховська Н. Б., Кунгурцев О. Б., Філатов В. О., Малахов Є. В., Скобцов Ю. А., Зіноватна С. Л., Новохатська С. Ю. та ін. Однак, не дивлячись на певний прогрес в методах та інформаційних технологіях проєктування структур РБД, питання оптимізації структури БД вузлів РКІС шляхом нових підходів до моделювання та статистики SQL-запитів обумовлює актуальність теми даного дослідження.

Так, у роботах Стогнія А. О., Дейта К. Дж., Кодда Е. Ф. висвітлюються питання проєктування автоматизованих систем управління та обробки даних, у роботах Пасічника В. В., Барсегяна А. А. – питання організації сховищ даних та багатовимірних моделей. Невисвітленими залишаються питання побудови моделей при використання комбінованої стратегії розподіленого представлення даних у КІС. Вчені Кунгурцев О. Б., Філатов В. О., Зіноватна С. Л., Новохатська С. Ю. у своїх роботах розглядають питання підвищення продуктивності автоматизованих систем за рахунок використання матеріалізованих представлень, реструктуризації БД та денормалізації відношень. Однак, аспекту оптимальності структури окремого вузла РКІС у проаналізованих дослідженнях приділяється мало уваги. У роботах Шаховської Н. Б. розглянуто підходи до аналізу тексту, а також використання консолідації та просторів даних при роботі із багатовимірними

моделями. Проте, використання багатовимірних моделей при представленні SQL-запитів до реляційної БД не розглядається.

Отже, задача розробки моделей та інформаційної технології оптимізації структури бази даних вузла у корпоративних інформаційних системах є актуальною.

Мета і завдання дослідження

Метою дослідження є підвищення рівня доступності даних та ефективності використання розподілених та територіально розосереджених комп'ютерних систем шляхом реалізації інформаційної технології оптимізації структури БД вузла РКІС на основі статистики SQL-запитів.

Об'єктом дослідження є процес оптимізації структури БД розподілених корпоративних інформаційних систем, в основі яких лежать реляційні БД.

Предмет дослідження – моделі SQL-запитів реляційних БД та методи вибору кращої альтернативи структури вузла РКІС.

1 МОДЕЛІ ТА МЕТОДИ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ РОЗПОДІЛЕНИХ БАЗ ДАНИХ

1.1Технології підвищення швидкості роботи інформаційних систем

Інформаційні системи (ІС), що є взаємозалежною сукупністю засобів, методів і персоналу та мають на меті зберігання, обробку та представлення інформації, звичайно мають справу з великими обсягами інформації, що нерідко має досить складну структуру. Майже будь-яка ІС для збереження даних використовує бази даних (БД), що перебувають під управлінням системи керування базами даних [1].

Система керування базами даних (СКБД) — це комплекс програмних і мовних засобів, необхідних для створення баз даних, підтримки їх в актуальному стані й організації пошуку в них необхідної інформації. Розрізняють файлові та клієнт-серверні СКБД. Специфікою архітектури «клієнт-сервер» є використання спеціальної мови структурованих запитів, або Structured Query Language (SQL) – простого й потужного інструменту для доступу до даних. Запит клієнта на виконання якої-небудь операції з даними провокує на сервері пошук і добування даних. Витягнуті дані транспортуються по мережі від сервера до клієнта [2].

Один із основних методів підвищення ефективності роботи клієнт-серверних ІС полягає у зменшення сумарного часу, необхідного на обробку клієнтського запиту та повернення результату його виконання, більшу частину якого складає, як правило, його виконання на сервері БД.

1.1.1 Оптимізація запитів та індексація даних

Оптимізація запитів є одним із найбільш важливих і цікавих напрямків досліджень і розробок в області баз даних. Існує перелік стандартних підходів, що дозволяють зменшити час виконання SQL-запиту на сервері

БД [3]. Так, у роботах наведено рекомендації використання конкретних імен атрибутів, зведення до мінімуму використання підзапитів, уникання використання оператору «in» із великими підмножинами, використання фільтрації даних на якомога ранніх етапах вибірки, використання правильного порядку поєднання таблиць та ін. Дослідження наголошує на можливих перевагах використання тимчасових таблиць, оператора «join» перед вкладеним підзапитом та уникання конструкцій корельованих підзапитів, якщо це можливо.

У роботі Філатова В. О. [4] розглянуті питання побудови оптимальної послідовності з'єднання відношень в запитах реляційної бази даних, а Кунгурцев А. Б. у [5] використовує ієрархічну модель об'єктів для дослідження запитів до БД.

Всі сучасні СКБД мають у своєму складі спеціальні модулі та підсистеми, що мають спільну назву «оптимізатор запиту». Робота оптимізатора запиту є ключовою у обробці даних, а розуміння принципів його роботи допомагає при побудові швидких запитів. На етапі вибірки даних СКБД намагається найбільш швидким та менш витратним чином використовувати свої ресурси. Цю роботу виконує оптимізатор запитів, і, як результат, видає «план виконання запиту». У більшості випадків оптимізатор запиту виконує перетворення тексту запиту відповідно до загальних рекомендацій підвищення продуктивності його роботи. Відповідно, навіть при реалізації неоптимальних запитів з боку розробників ІС, СКБД частково вирішує цю проблему. Крім того, оптимізація запитів стає можливою тільки при наявності доступу до модифікації їх тексту, що у більшості випадків у користувача ІС відсутній.

Один із ефективних шляхів підвищення швидкості SQL-запитів полягає в упорядкуванні даних за рахунок створення індексів. Дані можуть бути упорядковані двома шляхами: через зміну реального порядку збереження даних на дисковому носії, або через створення додаткової структури, де зберігається поле, за яким виконується сортування, та посилання на рядок

таблиці. У другому випадкувиконується сортування додаткової структури, а не самих даних. Даним типам сортування в реляційних СКБД відповідають кластерний на некластерний типи індексів.

Тобто можна стверджувати, що оптимізатор запитів разом зі статистикою СКБД надає достатній обсяг рекомендацій розробникам ІС та БД щодо створення та модифікації існуючих індексів, які можуть позитивно вплинути на швидкість виконання SQL-запитів. Практика використання ІС показує, що у переважній більшості випадків всі необхідні індекси, що можуть позитивно вплинути на роботу ІС, уже створені і ефективно використовуються [6].

1.1.2 Модифікація структури БД

Іншим підходом оптимізації запитів та підвищення ефективності ІС є модифікація структури БД на базі контрольованої денормалізації. Усунення аномалій даних відповідно до теорії реляційних БД вимагає, щоб будь-яка БД була нормалізована, тобто відповідала вимогам нормальних форм. Відповідність вимогам нормалізації мінімізує надмірність даних в БД і забезпечує відсутність багатьох видів логічних помилок оновлення та вибірки даних. Однак, при виконанні вибірки великих обсягів даних операція з'єднання нормалізованих відношень виконується неприйнятно довго. Внаслідок цього, в ситуаціях, коли продуктивність таких запитів неможливо підвищити іншими засобами, може проводитиськонтрольована денормалізація, що полягає у композиції кількох відношень (таблиць) в одну [7], [8].

За рахунок такого перепроєктування операція сполучення при вибірці даних стає непотрібною, і, як результат, запити вибірки, які раніше вимагали з'єднання, працюють швидше. Слід пам'ятати, що денормалізація завжди виконується за рахунок підвищення ризику порушення цілісності даних при операціях модифікації. Тому денормалізацію слід проводити в крайньому

випадку, якщо інші заходи підвищення продуктивності неможливі, або у випадках, коли денормалізована БД використовується тільки на читання. Крім того, слід враховувати, що прискорення одних запитів на денормалізованій БД може супроводжуватися уповільненням інших запитів, які раніше виконувалися окремо на нормалізованих відношеннях.

1.1.3 Використання матеріалізованих представлень

Окремим випадком модифікації структури БД, та ще одним із шляхів підвищення швидкості виконання запитів є використання матеріалізованих представлень. Матеріалізоване представлення – фізичний об'єкт БД, що містить результат виконання запиту. Так як в матеріалізованих представленнях зберігаються вже заздалегідь обчислені результати запиту, включаючи підсумки і результати з'єднань таблиць (JOIN),отримання даних з них виконується значно швидше, ніж у випадку звичайних представлень. В деяких СКБД, таких як Oracle, вже є штатний механізм для створення матеріалізованих представлень на рівні мови SQL (CREATE MATERIALIZED VIEW). В інших СКБД такий механізм може бути відсутній, але це не означає, що створити матеріалізоване представлення там неможливо [9], [10].

Як випливає з визначення, щоб представлення працювало як матеріалізоване, воно повинно зберігати результати виконання запиту на фізичному рівні. У разі використання СКБД, де не передбачено матеріалізованих представлень явним чином, цього можна домогтися шляхом створення для представлення кластерного індексу. Тоді результати зберігаються фізично в індексі і час звернення до них зменшується. Незважаючи на свої переваги, даний підхід має недолік, спільний із індексацією даних, що полягає у сповільненні виконання операцій модифікації даних. Також відсутність доступу до модифікації коду ІС

обумовлює неможливість їх використання останніми для підвищення ефективності ІС.

1.2 Використання моделей СД, OLAP-технологій та ETL-систем

Ще один підхід підвищення ефективності ІС пов’язаний із поняттям сховища даних (СД). Системи, що вирішують задачі вводу та збереження даних та інформаційно-пошукові системи обробки даних, використовують для обробки даних транзакційний підхід. Розвинений механізм керування транзакціями в сучасних СКБД зробив їх основним засобом побудови OLTP-систем, основною задачею яких є забезпечення виконання операцій із БД.

Практика використання OLTP-систем показала неефективність їх застосування для повноцінного аналізу інформації. Основною причиною невдач є протиріччя у вимогах, що пред’являються до систем OLTP та систем підтримки прийняття рішень (СППР), серед яких різний ступінь деталізації даних, допущення надлишковості, кількість даних, що зберігаються, час обробки звернення до даних та ін. Наявність наведених суперечностей веде до появи концепції сховищ даних, основна ідея якої полягає у розділенні БД для OLTP систем і БД для виконання аналізу і подальшому їх проєктуванні з врахуванням відповідних вимог. При реалізації в СППР концепції СД дані з різних оперативних джерел даних (ОДД) акумулюються в єдиному сховищі, що неминуче приводить до дублювання інформації в ОДД і в СД.

Недоліками концепції СД є необхідність інтеграції даних неоднорідних джерел у розподіленому середовищі, потреба в ефективному зберіганні великих об’ємів інформації, необхідність багаторівневих довідників метаданих та підвищення вимог до безпеки даних. Використання віртуального СД вирішує частину з них, але веде до виникнення ряду інших, серед яких збільшення часу обробки запитів до СД, необхідність постійної доступності всіх ОДД, та неможливість отримання даних за довгий період часу.

Задача перенесення даних із ОДД до СД покладається на окремий клас систем, що отримали назву Extraction Transformation Loading (ETL) системи. Задача асинхронної реплікації даних, що є актуальною для розподілених СКБД, також може бути покладена на ETL-систему.

Процес перенесення, включає етапи витягання, перетворення і завантаження. Серед способів витягання даних виділяються:отримання даних допоміжними програмними засобами безпосередньо із структур зберігання інформації та вивантаження даних засобами OLTP-систем в проміжні структури.

Перший підхід характеризує відсутність необхідності розширення OLTP-системи, та витягання даних з врахуванням потреб процесу перенесення. При цьому структура БД може бути невідомою або відсутні механізми доступу до даних. Крім того, зміна структури БД ОДД веде до необхідності перепроєктування підсистеми витягання даних. Другий характеризує можливість використання засобів OLTP-систем, адаптованих до структур даних та зміни засобів вивантаження разом із змінами OLTP-систем і ОДД [10].

Етап перетворення включає узагальнення, приведення значень та очищення даних. Процедура очищення даних направлена на виявлення і видалення помилок і невідповідностей у даних з метою поліпшення їх якості. Після того, як дані перетворені, здійснюється етап їх завантаження.

1.3 Проблеми використання багаторівневих, розподілених та територіально розосереджених КІС

У межах одного підприємства часто існує необхідність автоматизації різних типів обліку. У якості прикладу можна навести складський, бухгалтерський облік, облік кадрів, розробка інформаційних порталів, систем відеоспостереження, контролю прав доступу до приміщень та ін. Спроба одночасної автоматизаціїрізних видів обліку привело до виникнення так

званих «універсальних» або «комплексних» облікових систем, що створюють єдине облікове середовище організації, та забезпечують доступ до всіх необхідних даних моніторингу, контролю і оцінки ефективності роботи організації, а також для підтримки прийняття управлінських рішень.

Підхід, пов’язаний із використанням такого роду систем, має цілий ряд недоліків. По-перше, реалізація всіх видів обліку у межах однієї ІС веде до перевантаження БД такої системи. Враховуючи специфіку різних типів обліку, структура такої БД є досить складною, відповідно транзакції запитів на вибірку та зміну даних є тривалими, що приводить до виникнення взаємних блокувань користувачів при паралельній роботі. Іншим аспектом є низька відмовостійкість та вразливість такої системи, оскільки тимчасова недоступність центральної БД комплексної ІС приводить до зупинки роботи інформаційної систем всіх структурних підрозділів організації. Також слід зауважити, що жодна з «універсальних» облікових систем не реалізовує у повному обсязі специфіку всіх видів діяльності компанії.

1.3.1 Централізована та розподілена моделі КІС

Наведені недоліки можуть бути мінімізовані за допомогою використання окремих спеціалізованих ІС. Перевагами спеціалізованих облікових систем є більш високий рівень якості реалізації необхідних облікових механізмів та локалізація використання даних кожної ІС у вигляді окремої БД. Тимчасова недоступність БД або серверу застосунків однієї з ІС не впливає на роботу інших. Але даний підхід веде до появи різних платформ СКБД, що потребують подальшої синхронізації.

Холдингова структура об’єкта автоматизації та/або географічна віддаленість філій підприємства також може обумовити необхідність створення та використання БД розподіленої структури. У деяких випадках можливе використання центральної БД із роботою через виділений канал зв’язку або розміщення БД у хмарі. Але при цьому збільшується

навантаження на центральну БД, канали зв’язку, та знижується відмовостійкість системи. Крім того, існують фрагменти даних, що мають бути оперативно доступними 24/7 незалежно від наявності зв’язку, навіть за рахунок втрати їх актуальності.

У розвитку сучасних ІС намітилася тенденція переходу від локальних БД до створення розподілених БД. Це обумовлено як наведеними вище причинами, так і розвитком інформаційних технологій, а саме появою і широким поширенням технічних засобів високошвидкісної передачі даних і вдосконаленням засобів побудови розподілених систем. Відповідно все більше підприємств, що мають територіально розподілену структуру впроваджують і використовують розподілені корпоративні інформаційні системи (РКІС).

У свою чергу розподілені корпоративні інформаційні системи (РКІС) – це один із складних видів комп’ютерних систем, що створюється як правило для крупних підприємств або корпорацій, що об’єднують декілька організацій, у тому числі територіально розподілених. Зв’язок забезпечується за допомогою комп’ютерних мереж, у тому числі із використанням мережі Інтернет, а дані зберігаються у РБД.

1.3.2 Технології представлення даних та використання розподілених транзакцій

Серед загальних моделей та стратегій розподілу даних між вузлами РКІС, без урахування особливостей та обмежень конкретної розподіленої СКБД, виділяють централізовану та розподілену моделі БД. Стратегії представлення даних на базі розподіленої моделі включають наступні класи: розподілена без дублювання; розподілена з дублюванням; та комбінована [11].

Централізована стратегія характеризується розміщенням всіх даних в одному вузлі мережі та наявністю системи управління доступом інших вузлів

до даних. Вона має ряд переваг, серед яких простота реалізації забезпечення цілісності та захисту даних, створення та ведення файлів БД, проєктування структури БД, відсутність необхідності у синхронізації даних між ОДД та можливістю проведення інтегрованого аналізу даних у межах всієї предметної області конкретної БД. Однак також характерно виникання великих черг, та збільшення часу реакції системи, що особливо актуально у високонавантажених системах та/або при використанні довгих транзакцій. Обсяг бази даних обмежений ресурсами одного серверу баз даних. Актуальним є також питання навантаження на мережу при передачі даних на вузол РКІС, особливо у випадку відсутності надійного каналу зв’язку між сервером БД та клієнтським застосунком.

При розподіленій стратегії без дублювання структура РБД проєктується, як неперетинні між собою підмножини даних, розподілені по вузлах РКІС, що є досить складною задачею. Ця стратегія оптимальна у випадку мінімальної кількість логічних посилань взаємозв'язків вузлів одного з одним, коли кожен вузол працює зі своїми даними та майже не використовує дані інших вузлів РКІС. При цьому зменшуються витрати на передавання інформації та забезпечується доступність та швидкість доступу до даних, зменшується вірогідність виникнення черг, збільшується швидкість обробки локальних запитів. Однак наряду із цим виникає необхідність використання розподілених транзакцій для організації доступу до даних на різних вузлах РКІС. Використання розподілених транзакцій у свою чергу передбачає одночасну доступність всіх вузлів, а блокування, накладені на кожному окремому вузлі РКІС будуть зняті лише після повного завершення роботи розподіленої транзакції [12].

За розподіленої стратегії з дублюванням проєктування БД виконується як за централізованого підходу, але із фізичним її дублюванням в кожному вузлі РКІС. Дана стратегія ефективно розв'язує проблеми доступу та вибірки даних з мінімальними витратами часу, а система досить проста при проєктуванні. При цьому цей підхід характеризується складністю

адміністрування та розв'язання проблеми узгодженості файлів БД у різних вузлах РКІС,особливо при виникненні розбіжностей у даних, що вимагає спеціальних механізмів їх узгодження.

Комбінована стратегія поєднує підходи розподілу без дублювання та з дублюванням даних. Ця стратегія поділяє БД на багато логічних фрагментів та дозволяє мати довільну кількість фізичних копій кожного фрагмента. Система, побудована за цією стратегією, допускає реалізацію паралельної обробки даних, що скорочує час відгуку та забезпечує більш високу надійність даних за рахунок їх часткового дублювання. Однак залишається проблема узгодженості копій фрагментів БД у всіх вузлах РКІС [13].

Враховуючи результати аналізу наведених наукових робіт, а також особливості різних стратегій представлення даних в РКІС, комбінована стратегія є найбільш виправданою, однак актуальною постає задача оптимізації структури БД вузла РКІС, а також задача синхронізації даних.

1.3.3 Технології СКБД та ІС щодо інтеграції даних

Синхронізація даних між однорідними та/або різнорідними джерелами даних може бути здійснена за допомогою різних методів. По-перше, у розпорядженні майже бідь-якої сучасної СКБД існують механізми по керуванню розподіленими транзакціями, забезпеченню дзеркального відображення та реплікації БД та їх окремих елементів. Іншим підходом може бути використання інструментів синхронізації сторонніх розробників, що дозволяють із одного середовища взаємодіяти із різними СКБД, наприклад EMS Data Comparer або Akeneo.

Деякі ІС мають у своєму розпорядженні механізми файлового обміну даними із можливістю налаштування автоматичного або напівавтоматичного режиму оновлення, що реалізовано безпосередньо розробниками OLTP-систем та враховують особливості представлення сутностей в інформаційній моделі. Крім того, при відомій структурі БД OLTP-системи завжди існує

можливість розробки власної системи синхронізації, яка в більшості випадків буде поєднувати методи СКБД та ІС для можливості використати переваги кожного з підходів, врахувавши їх недоліки [14].

Реплікація є набором технологій, за допомогою яких дані або об'єкти БД можна копіювати і переносити з однієї БД в іншу, а потім синхронізувати ці БД для забезпечення узгодженості. Виділяють три категорії даних, що беруть участь в процесі реплікації:

– оновлювані тільки на центральному вузлі;

– відфільтровані дані, що оновлюються тільки на одному з віддалених вузлів;

– дані, які можуть оновлюватися декількома користувачами незалежно один від одного.

В останньому випадку при синхронізації даних можуть виникати конфлікти. Процес виявлення і вирішення конфліктів може займати багато часу і ресурсів, тому рекомендовано мінімізувати кількість конфліктів застосунку, створюючи секції даних так, щоб різні вузли РКІС змінювали різні підмножини даних. Одним з основних параметрів є фільтрація даних, що дозволяє фільтрувати стовбці і рядки так, що таблиці міститимуть тільки дані, необхідні для застосунку.

1.4 Методи та технології оптимізації структури БД на вузлах РКІС

Ключовим фактором, який впливає на надійність і доступність БД, є так звана «локалізація посилань». Якщо БД розподілена таким чином, що розміщені на вузлі дані викликаються винятково його користувачами, це свідчить про високий ступінь локалізації посилань. Якщо подібнуфрагментацію даних здійснити неможливо і для виконання запитів потрібно звертатись за інформацією до інших вузлів, то це свідчить про невисокий ступінь локалізації посилань [15].

Комбінована стратегія розподілу даних є найбільш виправданою із точки зору можливості поєднання переваг стратегій з та без дублювання та дозволяє досягти найвищого рівня локалізації посилань при мінімальній збитковості даних. Але при її використанні, окрім задачі синхронізації дубльованої інформації, актуальною постає задача оптимального проєктування структури БД з точки зору приналежності даних до того чи іншого вузла РКІС. Крім того, продуктивність системи напряму буде залежати від прийняття рішення про необхідність часткового або повного дублювання даних.

Враховуючи закритість коду SQL-запитів РКІС, задача оптимального проєктування структури БД полягає у пошуку оптимального розподілу даних по вузлах РКІС [16].

Класичні методи пошуку оптимуму, такі як метод Гоморі, метод гілок та границь не можуть бути застосовані у зв’язку із характером задачі, що вирішується, а також її великою розмірністю. Метод повного перебору, що дозволяє знайти глобальний оптимум, при великих розмірах дискретної задачі пошуку оптимального розподілу даних по вузлах РКІС не може бути застосований через неприпустимі часові витрати на проведення розрахунків.

Однак імітаційні моделі РКІС, що є вкрай складною системою, зазвичай дуже спрощені та не враховують багатьох факторів, що впливають на її поведінку. Не розглядаються особливості транзакційного режиму роботи із даними та роботи на різних рівнях ізоляції, що безпосередньо впливає на кількість блокувань, та взаємний вплив процесу виконання одних запитів на інші. Вкрай важливими також є налаштування серверу БД, від яких напряму залежать як продуктивність окремого запиту, так і їх взаємний вплив один на одного. Ускладнення моделі поглиблюється із уведенням розподілених запитів та розподілених транзакцій з двофазним протоколом їх фіксації та можливою наявністю різних СКБД у випадку гетерогенних джерелданих або СД. Також більшість існуючих підходів оптимізації структури РБД та КІС розглядають оптимальне розташування попередньо визначених фрагментів

даних на вузлах РКІС, але не розглядають оптимізацію самого процесу фрагментації [17].

Відповідно до наведеного, використання імітаційних моделей можна вважати недоцільним. Натомість запропоновано накопичення статистичної вибірки виконання SQL-запитів РКІС до РБД на реально працюючих серверах, що дає змогу врахувати всі вищенаведені особливості РКІС. Парсинг текстів SQL-запитів та розщеплення на окремі підзапити дозволить виявити множини відношень, атрибутів та кортежів БД, задіяних у кожному запиті. Подальший аналіз накопиченої та обробленої статистики в аналітиці доступних вимірів у вигляді програмного застосунку, хоста, користувача та ін. дозволить виявити частоту та характер звернень до окремих відношень та їх атрибутів і кортежів.

1.5 Оптимізація структури БД вузла РКІС за критеріями оптимальності

При формулюванні задачі оптимізації структури БД вузла РКІС запропоновано виконання оцінки оптимальності структури за значеннями відповідних критеріїв оптимальності, а саме:доступності та швидкості отримання даних, незалежності від центрального вузла БД, розміру БД, ступеня достовірності даних, необхідності у подальшій синхронізації. Деякі з критеріїв, що мають спільну природу, об’єднано у спільні групи, після чого отримано три групи критеріїв: «незалежність БД», «розмір БД» та «необхідність синхронізації». Кожен атрибут та кортеж відношення згідно з результатами обробки статистики SQL-запитів маркується відповідним значенням необхідності представлення на вузлі РКІС. Значення критеріїв виражається у вигляді залежності від рівня представленості даних на вузлі РКІС, а задача зводиться до визначення оптимального значення маркеру представленості даних на вузлі РКІС [18].

1.5.1 Формулювання задачі багатокритеріальної оптимізації

Процес прийняття рішень, та методи його підтримки відносяться до теорії прийняття рішень та часто реалізуються у комплексах комп'ютерних програм, що мають загальну назву системи підтримки прийняття рішень (СППР). В процесі прийняття рішення, людина, що приймає рішення (ЛПР) може також виступати у ролі експерта. Прийняття рішення передбачає вибір одного з можливих варіантів дій, що прийнято називати альтернативами. У деяких випадках альтернатив може виявитись забагато для ЛПР, що вимагає залучення додаткових заходів підтримки вибору рішення. До цього випадку потрапляє також і задача визначення оптимального рівня маркеру представленості даних.

Кожне значення рівня маркеру представленості даних характеризується показниками їх привабливості для ЛПР, а саме такими критеріями оптимальності, як незалежність від центрального вузла БД, співвідношення розміру локальної БД до центральної, та показник рівня необхідності синхронізації даних. Всі вони є критеріями вибору рішення. У процесі прийняття рішення нерідко до участі залучаються експерти, але враховуючи конфіденційність даних, самого процесу прийняття рішення, та його результату, для ЛПР це не завжди є прийнятним. Крім того, наявність експертів завжди підвищує вартість системи. Враховуючи це, ставиться на меті реалізація методів, при використанні яких експерти можуть бути залучені, але ЛПР також може виступати в ролі експерта [19].

При визначенні маркеру представленості даних зв'язок між його значенням (альтернативою) та значеннями критеріїв для оцінки цих альтернатив (незалежності локальної БД, необхідності у синхронізації та розміру БД) задається у вигляді математичних моделей. Отже, це дозволяє, обравши один із методів вирішення багатокритеріальної задачі дослідження операцій, перейти до знаходження оптимального рішення.

Традиційний підхід дослідження операцій передбачає наявність єдиного критерію оцінки якості прийнятого рішення. Одним із перших підходів, що використовується для 2-х критеріїв, яким є метод «вартість-результат», що полягає у побудові моделей результативності та вартості, та подальшого синтезу їх оцінок. Однак метод має ряд недоліків, крім того, у нашому випадку при наявності трьох критеріїв, одна з моделей (в залежності від того, зарахувати критерій необхідності синхронізації даних до вартості або до результату) перетворюється на багатокритеріальну [20].

Задача визначення оптимального рівня маркеру представленості даних є добре структурованою, оскільки всі залежності можуть бути представленні у чисельному вигляді. Також критерії оптимальності (рівень незалежності локальної БД, ступінь необхідності у синхронізації даних та розмір локальної БД) є незалежними, і може буди задано напрямок покращення значень кожного з критеріїв. Так, при збільшенні значення маркеру представленості даних, ступінь необхідності у синхронізації зростає або залишається сталою, розмір локальної БД збільшується, рівень незалежності локальної БД також зростає або залишається сталим. Виходячи із наведеного, дану задачу можна віднести до класу задач багатокритеріальної оптимізації (БКО).

У випадку багатокритеріальних задачах частина інформації, необхідна для повного та однозначного визначення вимог до рішення, принципово відсутня. Так, було визначено критерії оптимальності, та встановлено зв’язок між ними та альтернативами (значенням маркеру представленості даних). Але кращі поєднання критеріїв не можуть бути визначені на основі об'єктивної інформації, наявної в розпорядженні дослідника. Враховуючи наведене, дана задача може бути визначена, як слабо структурована.

Наявність декількох критеріїв призводить до зміни характеру розв'язуваної задачі і ролі ЛПР,суб'єктивні переваги якої мають стати основою для вироблення рішення. Рішення хоч і буде суб'єктивним, але в процесі вирішення будуть використані об'єктивні моделі. Використання

багатокритеріальних методів дозволяє ЛПР уникнути поверхневих рішень та схильності до спрощення задачі, внаслідок чого замість декількох критеріїв буде розглянуто лише один, що вбачається ЛПР більш важливим.

Набір рішень, що містить в собі ті рішення, які можна використовувати при пошуку найкращого, прийнято називати простором рішень (далі *W*). Деякі з елементів простору рішень (що задається неоднозначно) неможливо реалізувати з тих чи інших причин. Ті рішення, які можуть бути використані при пошуку найкращого рішення, утворюють підмножину *X* простору рішень *W*. Таку множину $X \subset W$ називаємо множиною допустимих рішень. Нехай задана числова функція *f*, значення якої описують рівень переваги рішень. В цьому випадку природно вважати, що метою є збільшення значення функції *f*.

У даному вигляді рішення трактується як пошук деякого елемента множини *X*, при якому значення *f* максимальне. Таким чином, приходимо до задачі однокритеріальної (скалярної) оптимізації: $f(x) \to max, x \in X$.

Елемент $x^* \in X$ називається рішенням скалярної задачі оптимізації, якщо $f(x^*) > f(x)$ для всіх $x \in X$. Набір з трьох критеріїв вибору рішення є сукупність функцій (f_1, f_2, f_3), заданих на просторі *W*. Поряд з простором рішень *W*, можна представити критеріальний простір *W'*, що включає в себе прямий добуток шкал критеріїв (множин можливих значень окремого критерію). Таким чином, сукупність критеріальних функцій задає відображення $f = (f_1, f_2, f_3)$, що діє з *W* в *W'*.

Важливою умовою є незалежність критеріїв за перевагою, при якій бажані для ЛПР зміни значень кожного з окремих критеріїв при незмінних значеннях інших критеріїв не повинні залежати від конкретних значень інших критеріїв. Для будь-якої пари значень з шкали кожного з критеріїв (скажімо, y_j' и y_j'') ЛПР в змозі визначити, що одне значення краще іншого $y_j' \succ y_j''$, або значення рівноцінні $y_j' \approx y_j''$. При цьому повинна виконуватися вимога несуперечності тверджень про переваги, тобто якщо $y_j' \succ y_j''$ та $y_j'' \succ y_j'''$, то $y_j' \succ y_j'''$, і якщо $y_j' \approx y_j''$ та $y_j'' \approx y_j'''$, то $y_j' \approx y_j'''$.

Таким чином, переваги, сформульовані ЛПР, зручно виразити за допомогою математичного поняття, що характеризує співвідношення між парами об'єктів, тобто через бінарні відношення.

Критерії оптимальності значення маркеру представленості даних, а саме рівень незалежності локальної БД, ступінь необхідності у синхронізації даних та розмір локальної БД є числовими, незалежними за перевагою та ЛПР у більшості випадків може порівняти будь-яку пару значень за шкалою кожного з окремих критеріїв. Критеріальний простір є тривимірним лінійним простором, а окремі критерії не тільки є числовими, але і набори їх значень можуть додаватись та помножатись на число.

Як вже зазначалося, критерії є монотонними за перевагою, що означає збільшення (або зменшення) значення окремого критерію (при постійних значеннях інших критеріїв) для ЛПР. Щоб звести задачу до класичної задачі багатокритеріальної максимізації, необхідно дотримання умови бажаності для ЛПР збільшення значення кожного із окремих критеріїв.

У нашому випадку бажаним є збільшення значення критерію рівня незалежності локальної БД, а ступінь необхідності у синхронізації даних та розмір локальної БД бажано зменшувати. Тому, для зведення задачі до виду багатокритеріальної максимізації, останні два критерія мають бути помножені на «–1». Враховуючи всі зазначені особливості, для порівняння критеріальних векторів у нашому випадку може бути використане алгебраїчне порівняння значень окремих критеріїв.

При відсутності конфлікту між критеріями, можливе рішення задачі може бути представлене у вигляді ідеальної точки. Під ідеальною точкою розуміють такий вектор $y^* \in W'$, компоненти якого є максимумами окремих критеріальних функцій $f(x)$ окремих критеріїв.

Але враховуючи той факт, що у нашому випадку зміна значення рівня маркеру представленості даних (зміна альтернативи) веде до покращення значення одних критеріїв за рахунок погіршення значення інших, ідеальної точки на критеріальному просторі допустимих рішень не існує, а отже не

існує і абсолютно оптимального рішення $x^* \in X$. Отже, для знаходження єдиного оптимального рішення, задача має бути зведена до однокритеріальної.

1.5.2 Аналіз методів розв'язання задачі БКО

При вирішенні задачі БКО основною задачею є перехід до однокритеріальної задачі, або послідовності таких задач, що може бути вирішена одним із методів дослідження операцій. Серед основних підходів слід згадати отримання суперкритерію шляхом згортки критеріїв, методи звуження множини альтернатив, групування критеріїв, а також методи головного критерія, ідеальної точки та послідовних поступок.

Методи звуження множини альтернатив у більшості випадків базуються на визначенні множини рішень, ефективних за Парето (Стейтором, Смейлом). На першому кроці моделювання на віддаленому вузлі БД територіально розосередженої КІС запропоновано створення повної копії БД центрального вузла. Далі із її складу виключаються всі надлишкові дані, до яких не виконується жодних звернень користувацьких SQL-запитів. Всі інші альтернативи, що перебувають між цим станом і станом повної відсутності БД у віддаленому вузлі, входять до множини рішень, ефективних за Парето, оскільки призводять до покращення рівня значень одних критеріїв за рахунок погіршення інших.

Метод головного критерію, що полягає у виборі одного критерію у якості вирішального, та використання двох інших у вигляді обмежень, переважає за рахунок простоти інтерпретації результатів та відсутності додаткових вимог до експертів та обчислювальних засобів. Однак у нашому випадку залучення експертів вважається невиправданим через конфіденційність даних, тому переваги перекриваються цілим рядом суттєвих недоліків, а саме: надмірним спрощенням структури задачі; суб'єктивізмом при виборі головного критерія; можливістю втрати

сукупного впливу двох другорядних критеріїв і, як наслідок, отримання неефективного рішення. Також при деякій комбінації обмежень по двох неосновних критеріях існує ймовірність отримання порожньої множини допустимих рішень.

Повертаючись до методу ідеальної точки та методу послідовних поступок слід звернути увагу на можливість отримання у вигляді рішення неефективних за Парето альтернатив. Крім того, обчислення відстані від ідеальної точки у нашому випадку дає різні результати при використанні різних методів його визначення (Евклідова відстань, манхеттенівська відстань, відстань Чебишева тощо). Визначення відсотку поступки та міри співвідношення отриманого виграшу за неосновним критерієм у порівнянні із програшом за основним також може виявитися для ЛПР складною, а іноді, навіть невизначено складною операцією.

Серед інших методів виділено метод аналізу ієрархій (МАІ), що є загальною методологією розв’язання широкого класу задач прийняття рішень і дозволяє поєднати порівняно простий математичний апарат зі знаннями та досвітом ЛПР. Основою даного метода є представлення процесу рішення у вигляді багаторівневої ієрархії, яка має відображати всі складові задачі, що вирішується.

МАІ дозволяє виконувати такі етапи аналізу багатокритеріальних задач прийняття рішень, як: структурування задачі та формалізація зв’язків між складовими; формування системи переваг ЛПР для критеріїв та альтернатив. Перевагами методу є наочність моделей, простота інтерпретації результатів, можливість оцінки альтернатив по якісним та суб’єктивно-визначеним критеріям, а також стійкість до порушення узгодженості оцінок ЛПР. Основу метода складають принципи декомпозиції, парних порівнянь та ієрархічної композиції. Основними етапами методу є побудова ієрархії, оцінювання значимості та пріоритетів, перевірка узгодженості пріоритетів та синтез рішення.

2 РОЗРОБКА ТА ДОСЛІДЖЕННЯ МОДЕЛІ SQL-ЗАПИТУ У РЕЛЯЦІЙНИХ СТРУКТУРАХ

2.1 Поняття оптимізації обробки запитів

Слово «оптимізація» є математичним терміном і має на увазі вибір найкращого (або близького до нього відповідно до заданого критерію) об'єкта або алгоритму, що належить формально певного класу, відповідно, об'єктів або алгоритмів. Ясно, що такий підхід непридатний ні з теоретичної, ні з практичної точки зору до зв'язці «БД-СКБД» (як і до багатьох інших прикладних систем). Тому під оптимізацією обробки запитів СКБД розуміють стратегії, призначені для підвищення ефективності процедур обробки запитів. Такі стратегії є евристичні методи, доцільність використання яких обґрунтовується статистичними методами, тобто обробкою результатів проведених експериментів.

Поняття «запит», по своїй суті, є вираженням деякого мови і розглядається, по крайней мере, в наступних трьох контекстах:

– як стандартна вимога доступу користувача до даних БД (в цьому випадку оптимізація здійснюється за рахунок програмування відповідних процедур пошуку даних та перетворення входу користувача в результат необхідного формату);

– як транзакція, що здійснює зміна даних БД на підставі їх поточного значення;

– як вираз, яке СКБД використовує для авторизації користувача, забезпечення цілісності даних і синхронізації багато призначеного для користувача доступу до БД.

Для того, щоб говорити про оптимізацію обробки запитів, перш за все потрібно розуміти, що розуміється під вартістю запиту Виділяють наступні складові цього поняття.

Вартість комунікацій. Вартість передачі даних з місця, в якому вони зберігаються, в місця, де виконуються обчислення і представляються результати. Ця вартість складається з вартості комунікаційного каналу, яка зазвичай пов'язана з часом, протягом якого канал відкритий, і вартістю затримок в обробці, що викликаються передачею. Останній компонент, більш важливий для оптимізації запитів, часто покладається лінійною функцією від обсягу переданих даних.

Вартість доступу до вторинної пам'яті. Вартість (або час) завантаження сторінок даних з вторинної пам'яті в основну пам'ять. На цю вартість впливає обираних даних (головним чином, розмір проміжних результатів), кластеризація даних на фізичних сторінках, розмір доступного буферного простору і швидкість використовуваних пристроїв.

Вартість зберігання. Вартість заняття вторинної пам'яті і буферів основний пам'яті. Вартість зберігання доречна лише в тому випадку, коли пам'ять стає вузьким місцем системи, і розмір необхідної пам'яті може змінюватися від запиту до запиту.

Вартість (або час) використання ЦП.

На структуру алгоритмів оптимізації запитів впливають співвідношення між цими компонентами вартості. Вартість доступу до вторинної пам'яті (зазвичай вимірюється числом звернення до сторінок) і вартість використання ЦП (часто вимірюється числом порівнянь, які потрібно зробити). В основі більшості методів, розроблених для скорочення цієї вартості, лежить ряд загальних ідей:

– уникати дублювання зусиль;

– використовувати стандартизовані компоненти;

– заглядати вперед, щоб уникати зайвих операцій;

– вибирати найбільш дешеві способи виконання елементарних операцій;

– вибудовувати їх послідовність оптимальним чином.

Таким чином, оптимізація обробки запитів є нетривіальну многокритериальную завдання.

Існує два базові підходи в оптимізації запитів: спадний і висхідний, які реалізують принципи від загального до приватного і від приватного в загальному відповідно. Висхідний підхід характерний вивченням особливих приватних випадків і їх оптимізацією з подальшим будівництвом з них більших елементів. Цей підхід ефективний в тих випадках, коли стоїть завдання оптимізації відомого статичного набору запитів. Спадний підхід більш глобальний і універсальний, тому що він діє в евристичної манері. Він ефективний для оптимізації роботи системи взагалі, коли статичного набору запитів не існує, а безліч можливих запитів велике. Для пошуку оптимального глобального плану виконання розподіленого запиту ми виберемо саме цей підхід. Протягом останнього 20-річчя XX століття сформувалися такі три напрямки дослідження завдання оптимізація обробки запитів.

Розробка методів оптимізації обробки послідовності запитів. Формування цього напрямку обумовлена спробою зниження вартості обробки запиту за рахунок використання того фактора, що, як правило, в послідовності складових запитів міститься значна кількість загальних подвираженій. Перші методи оптимізації обробки послідовності запитів були засновані на вичерпну пошуку. Ці методи використовували двічі експонентну пам'ять від розміру запиту і були неефективними на практиці. Відчутний прорив стався в зв'язку з розробкою методів оптимізації обробки послідовності запитів, заснованих на використанні подання запитів орієнтованими ациклическими І-АБО графами.

Лексична оптимізація запиту полягає в його перетворенні, направленому на усунення надмірності на основі аналізу обмежень і умов, що містяться в ньому.

Існують наступні два підходи до вирішення цього завдання:

– перетворення запиту, поданого на нереляційних мовою (а при необхідності і даних), до реляційної формі, і застосування до неї методів лексичної оптимізації, розроблених для реляційних СКБД;

– побудова нових алгебр, призначених для подання запитів з урахуванням особливостей нових мов, і розробка методів оптимізації виразів цих алгебр.

Семантична оптимізація запиту є валідацію і перетворення синтаксичного дерева запиту до вигляду, придатного для виконання подальших кроків оптимізації. При цьому здійснюється перетворення запитів в канонічну форму (тобто розкриття уявлень, перетворення підзапитів в сполуки, спуск предикатів), спрощення умов, розподіл предикатів і перетворення дерева умов в шляху вибірки

2.2 Загальна схема обробки запиту

Незважаючи на те, що сучасних реляційних СКБД відрізняються за складністю та принципам створення, всі вони слідують одним і тим основним етапам у виконанні оптимізації запиту.

На першій фазі запит, поданий на мові запитів, піддається лексичного і синтаксичному аналізу. Лексичний аналізатор розбиває запит на лексичні одиниці – лексеми (найменування полів і таблиць, константи, знаки операцій і т.д.). Синтаксичний аналізатор перевіряє синтаксичну правильність запиту. В результаті виробляється внутрішнє уявлення запиту. Воно відображає структуру запиту і містить інформацію, яка характеризує об'єкти бази даних, згадані в запиті (таблиці, поля, константи). Інформація про об'єкти бази даних вибирається зі словника-довідника даних. Внутрішнє представлення запиту використовується і перетворюється на наступних стадіях обробки запиту.

На другій фазі запит в своєму внутрішньому поданні піддається логічній оптимізації. При цьому можуть застосовуватися різні перетворення,

поліпшуючі початкове уявлення запиту. Серед цих перетворень можуть бути еквівалентні перетворення. Після проведення еквівалентних перетворень виходить внутрішнє уявлення, семантично еквівалентний початковому запиту.

Перетворення можуть також використовувати інформацію про обмеження цілісності, існуючих в БД. Такі перетворення є семантичними, тобто вони засновані на семантиці (сенсі) предметної області. В цьому випадку отримується подання не є семантично еквівалентним початкового запиту. Але система гарантує, що результат виконання перетвореного запиту збігається з результатом запиту в початковій формі при дотриманні обмежень цілісності, існуючих в базі даних

У будь-якому випадку після виконання другої фази обробки запиту його внутрішнє подання залишається непроцедурного, хоча і є в певному сенсі більш ефективним, ніж початкова. Це означає, що за цим поданням система зможе побудувати більш ефективний план.

Третій етап обробки запиту полягає у виборі альтернативних процедурних планів виконання даного запиту відповідно до його внутрішнім поданням, отриманим на другій фазі. Для цього оптимізатор використовує інформацію зі словника-довідника даних, в першу чергу, про існуючі шляхи доступу до даних. Єдиний шлях доступу, який можливий в будь-якому випадку, – це послідовне читання (FULL). Можливість використання інших шляхів доступу залежить від способів розміщення даних в пам'яті (наприклад, кластеризація або хешування даних), від наявності індексів і формулювання самого запиту.

Також на третьому етапі для кожного з обраних планів оцінюється передбачувана вартість виконання запиту за цим планом. При оцінках використовується або доступна оптимізаторові статистична інформація про розподіл даних, або інформація про механізми реалізації шляхів доступу. З альтернативних планів вибирається найбільш оптимальний з точки зору деякого (заздалегідь обраного або заданого) критерію.

На четвертому етапі за обраним оптимальному плану виконання запиту формується виконується представлення плану. Яке Виконує представлення плану може бути програмою в машинних кодах, якщо система орієнтована на компіляцію запитів в машинні коди, або бути машинно-незалежним, але більш зручним для інтерпретації, якщо система орієнтована на інтерпретацію запитів. У нашому випадку це не принципово, оскільки четверта фаза обробки запиту вже не пов'язана з оптимізацією.

Нарешті, на останньому, п'ятому етапі обробки запиту відбувається його реальне виконання відповідно до процедурного планом виконання запиту.

2.3 Методи оптимізації

При класичному підході до організації оптимізаторів запитів на етапі логічного оптимізації виробляються еквівалентні перетворення внутрішнього подання запиту, які покращують початкове внутрішнє уявлення відповідно до фіксованих стратегіями оптимізатора. Характер поліпшень пов'язаний зі специфікою загальної організації оптимізатора, зокрема, з особливостями процедури пошуку можливих процедурних планів запитів, виконуваної на третій фазі обробки запиту.

Тому важко привести повну характеристику і класифікацію методів логічної оптимізації. Ми обмежимося декількома прикладами, а також розглянемо один приватний, але важливий клас логічних перетворень, що стосуються складних запитів, виражених мовою SQL.

2.3.1 Логічна оптимізація

Очевидний клас логічних перетворень запиту складають перетворення предикатів, що входять в умову вибірки, до канонічного поданням. Маються на увазі предикати, що містять операції порівняння простих значень. Такий

предикат має вигляд «арифметичний вираз ор арифметичне вираз», де «ор» – операція порівняння, а арифметичні вирази лівої і правої частин в загальному випадку містять імена полів відносин і константи (в мові SQL серед констант можуть зустрічатися і імена змінних осяжний програми, значення яких стають відомими тільки при реальному виконанні запиту).

Канонічні уявлення можуть бути різними для предикатів різних типів. Якщо предикат включає тільки одне ім'я поля, то його канонічне уявлення може, наприклад, мати вигляд «ім'я поля або константне арифметичне вираз» (ця форма предиката – простий предикат селекції дуже корисна при виконанні наступного етапу оптимізації).

Якщо предикат включає в точності два імені поля різних відносин (або двох різних входжень одного відносини), то його канонічне уявлення може мати вигляд «ім'я поля ор арифметичне вираз», де арифметичне вираз в правій частині містить тільки константи і друге ім'я поля.

Нарешті, для розглянутих предикатів більш загального вигляду має сенс приведення предиката до канонічного поданням виду «арифметичний вираз ор константне арифметичне вираз», де вираження правої і лівої частин також приведені до канонічного поданням. Наприклад, в виразах повністю розкриті дужки і вироблено лексикографическое впорядкування. Надалі можна зробити пошук загальних арифметичних виразів в різних предиката запиту. Це виправдано, оскільки при виконанні запиту обчислення арифметичних виразів буде проводитися при вибірці кожного чергового кортежу, тобто потенційно велике число раз.

При приведенні предикатів до канонічного поданням можна обчислювати вирази зі сталими та позбавлятися від логічних заперечень.

Наступний клас логічних перетворень пов'язаний з приведенням до канонічного виду логічного виразу, що задає умову вибірки запиту. Як правило, використовуються або діз'юнктівная, або Кон'юнктивна нормальні форми. Вибір канонічної форми залежить від загальної організації оптимізатора.

При приведенні логічного умови до канонічного поданням можна проводити пошук спільних предикатів (вони можуть існувати спочатку, можуть з'явитися після приведення предикатів до канонічного виду або в процесі нормалізації логічного умови) і спрощувати логічне вираз за рахунок, наприклад, виявлення кон'юнкції взаємно суперечать предикатів.

Фрагмент логічного виразу 1/4 (A> 5) AND (A <5) 1/4 можна замінити на 1 / 4FALSE1 / 4 Можливі й більш «розумні» спрощення. Наприклад, фрагмент логічного виразу 1/4 (A> B) AND (B = 5) 1/4 можна замінити на 1/4 (A> 5) 1/4. Такі спрощення можуть виявитися істотними для подальшої обробки запиту: в запиті з логічним умовою першого виду передбачалося з'єднання відносин; після перетворення запит вже не вимагає з'єднання.

У традиційних оптимізаторів поширені логічні перетворення, пов'язані зі зміною порядку виконання реляційних операцій.

Хоча мало хто реляційні системи мають мови запитів, засновані в чистому вигляді на реляційній алгебрі, правила перетворень алгебраїчних виразів можуть бути корисні і в інших випадках. Досить часто реляційна алгебра використовується в якості основи внутрішнього подання запиту. Природно, що після цього можна виконувати і алгебраїчні перетворення.

Зокрема, існують підходи, пов'язані з перетворенням до алгебраїчної формі запитів на мові SQL. Можна виявити дві основні спонукальні причини перетворень запитів на SQL до алгебраїчної формі. Першою, на наш погляд, менш важливою причиною може бути прагнення до використання реляційної алгебри в якості уніфікованого внутрішнього інтерфейсу реляційної СКБД. Такий підхід поширений при використанні спеціалізованих машин баз даних, на основі яких реалізуються різні інтерфейси доступу до баз даних. Інтерфейс машини баз даних повинен бути уніфікований (наприклад, бути алгебраїчним), а всі інші інтерфейси, включаючи інтерфейс на основі SQL, приводяться до алгебраическому.

Другою причиною, особливо важливою в контексті проблем оптимізації, є те, що реляційна алгебра простіша, ніж мова SQL.

Перетворення запиту до алгебраїчної формі спрощує подальші дії оптимізатора по вибірці оптимальних планів. Взагалі кажучи, розвинений оптимізатор запитів системи, орієнтованої на SQL, повинен виявити всі можливі плани виконання будь-якого запиту, але простір пошуку цих планів в загальному випадку дуже велике; в кожному конкретному оптимізаторі використовуються свої евристики для скорочення простору пошуку. Деякі, можливо, найбільш оптимальні плани ніколи не будуть розглядатися. Розумне перетворення запиту на SQL до алгебраическому поданням скорочує простір пошуку планів виконання запиту з гарантією того, що оптимальні плани втрачені не будуть.

Основною відмінністю мови SQL від мови реляційної алгебри є можливість використовувати в логічному умови вибірки предикати, що містять вкладені підзапити. Глибина вкладеності не обмежується мовою, тобто, взагалі кажучи, може бути довільною. Предикати з вкладеними підзапитах при наявності загального синтаксису можуть мати дуже різної семантикою. Єдиним загальним для всіх можливих семантик вкладених підзапитів алгоритмом виконання запиту є обчислення вкладеного підзапиту щоразу при обчисленні значення предиката. Тому природно прагнути до такого перетворення запиту, що містить предикати зі вкладеними підзапитах, яке зробить семантику підзапиту більш явнХою, надавши тим самим надалі оптимізаторові можливість вибрати спосіб виконання запиту, найбільш точно відповідний семантиці підзапиту.

Нижче R_iпозначає і-е відношення бази даних, C_k – k-е поле (стовпець) відносини.Предикати, допустимі в запитах мови SQL, можна розбити на наступні чотири групи.Прості предикати. Це предикати виду:

$$R_i, C_k \; op \; X, \tag{2.1}$$

де X – константа або список констант;

op – оператор скалярного порівняння (=,>,> =, <, <=) або оператор перевірки входження в безліч (IS IN, IS NOT IN).

Предикати зі вкладеними підзапитах. Це предикати виду:

$$R_i, C_k \; op \; Q, \tag{2.2}$$

де Q – блок запиту, а op може бути таким же, як для простих предикатів.

Предикат може також мати вигляд:

$$Q op \; R_i, C_k. \tag{2.3}$$

У цьому випадку оператор приналежності до безлічі замінюється на EXISTS або NOT EXISTS. Ці дві форми симетричні. Досить розглядати тільки одну.

Предикати з'єднання. Це предикати виду:

$$R_i, C_k \; op \; R_j, C_n, \tag{2.4}$$

де $R_i = R_j op$ – оператор скалярного порівняння.

Предикати ділення. Це предикати виду:

$$Q_i op \; Q_j, \tag{2.5}$$

де Q_iіQ_j – блоки запитів, а op може бути оператором скалярного порівняння або оператором перевірки входження в безліч.

Наведена класифікація є спрощенням реальної ситуації в SQL. Чи не розглядаються предикати з'єднання загального вигляду, що включають арифметичні вирази з полями більш ніж двох відносин.

Канонічним поданням запиту на n відносинах називається запит, який містить n-1 предикат з'єднання і не містить предикатів з вкладеними підзапитах. Фактично, канонічна форма – це алгебраїчне представлення запиту.При використанні в оптимізаторі запитів подібного підходу не обов'язково проводити формальні перетворення запитів. Оптимізатор повинен більшою мірою використовувати семантику оброблюваного запиту, а яким чином вона буде розпізнаватися – це питання техніки.

2.3.2 Оптимізація запитів за допомогою семантики БД

Розглянуті перетворення запитів грунтувалися на семантиці мови запитів, але в них не використовувалася семантика бази даних, до якої адресується запит. Будь-яке перетворення може бути проведено незалежно від того, яка конкретна база даних мається на увазі. Насправді ж, за будь-якої істинно реляційної базі даних зберігається і деяка семантична інформація, яка визначає, наприклад, цілісність бази даних.

Подання бази даних (view) в термінах мов – це іменований каталогізований запит, який представляє собою з точки зору користувачів такої ж об'єкт бази даних, як і ставлення. Уявлення можуть використовуватися в запитах так само, як і таблиці.

Семантика уявлень вимагає, щоб вони були точними: в момент виконання запиту над поданням отримана інформація повинна точно відповідати поточному стану збереженої частини бази даних. Виконання запиту над поданням вимагає його матеріалізації, тобто обчислення відносини, що визначається запитом, який пов'язаний з поданням.

Ідея семантичної оптимізації в принципі не нова. Є і принципова різниця між розглянутими вище перетвореннями запитів і тими, які виробляються при семантичної оптимізації. Основна відмінність полягає в тому, що коли виробляються злиття внутрішньої форми запиту з внутрішньою формою подання або внутрішніми формами обмежень

цілісності, ми зобов'язані повністю і однозначно використовувати зовнішню інформацію, незалежно від того, чи будуть це сприятиме оптимізації запиту: умова вибірки уявлення має бути цілком додано через AND до умові вибірки запиту; то ж відноситься і до набору логічних умов обмежень цілісності. Інакше семантика запиту над поданням або оператора модифікації бази даних буде порушена.

2.3.3 Оптимізація, заснована на правилах

Коли був досягнутий певний прогрес в поліпшенні обробки запитів, були зроблені і зусилля для поліпшення методів доступу до таблиць. Це стосується розробки методів доступу на основі використання індексів і функцій хешування. Однак застосування техніки індексування і хеширования збільшує складність обробки запиту. Наприклад, якщо таблиця має індекси по трьом різним колонках, то будь-який з них може бути використаний для доступу до таблиці (крім послідовного доступу до таблиці в фізичному порядку розташування рядків).

Крім цього, з'явилося багато нових алгоритмів для виконання з'єднання таблиць. Двома найбільш основними алгоритмами виконання з'єднання є:

– з'єднання за допомогою вкладеного циклу (Nested Loop Join). У цьому алгоритмі рядок читається з першої таблиці, званої зовнішньої (outer) таблицею, і потім читається кожен рядок другої таблиці, званої внутрішньої (inner), як кандидат для з'єднання. Потім читається другий рядок першої таблиці і знову кожен рядок з другої, і так до тих пір, поки всі рядки першої таблиці не будуть прочитані. Якщо в першій таблиці знаходиться M рядків, а в другій – N, то читається M x N рядків;

– з'єднання за допомогою об'єднання (Merge Join). Цей метод виконання з'єднання передбачає, що таблиці відсортовані (або проіндексовані) таким чином, що рядки читаються в порядку значень колонки (колонок), за якими вони з'єднуються. Це дозволяє виконувати

з'єднання за допомогою читання рядків з кожної таблиці і порівнювання значень колонок з'єднання до тих пір, поки відповідність цих значень існує. У цьому способі з'єднання завершується за один прохід по кожній таблиці.

Операції з'єднання підкоряються як комутативну, так і асоціативному законам.

Однак різні шляхи доступу, алгоритми з'єднань і порядок виконання з'єднань можуть призводити до значно розрізняється продуктивності. Отже, коли виконується з'єднання декількох таблиць, кожна з яких має кілька індексів, існує кілька сотень різних комбінацій для вибору порядку виконання з'єднань, алгоритмів з'єднань і шляхів доступу здійснення вибірки. Кожна з цих комбінацій виробляє один і той же результат, але з різними характеристиками продуктивності.

Одним з перших підходів на шляху боротьби з комбінаторної складністю виконання з'єднань полягає у встановленні евристичних правил для вибору між шляхами доступу і методами з'єднань, який називається оптимізацією, заснованої на правилах (rule based optimization).

При використанні цього методу план складається на підставі існуючих шляхів доступу і їх рангів. Всі шляхи доступу ранжуються на підставі знань про правила і послідовності здійснення цих шляхів. Ранги шляхів доступу для СКБД Oracle:

1. один рядок по ROWID *;
2. один рядок за кластерним з'єднанню;
3. один рядок по хеш-кластеру з унікальним або первинним ключем;
4. один рядок за унікальним або первинному ключу;
5. кластерний з'єднання;
6. ключ хеш-кластера;
7. ключ індексного кластера;
8. складовою індекс;
9. індекс по одиночному колонки (по умові рівності);
10. індексний пошук по закритому інтервалу;

11. індексний пошук по відкритому інтервалу;

12. сортування-об'єднання;

13. MAX і MIN по індексованих стовпцю;

14. ORDER BYпо індексувати одну;

15. повний перегляд таблиці.

Ранг шляху доступу визначається на підставі знань про послідовність реалізації цього шляху. Наприклад, найшвидший спосіб доступу – це читання по ROWID якщо він відомий, то це одна фізична читання.

Метод оптимізації по синтаксису враховує ранги шляхів доступу. Якщо для будь-якої таблиці існує більше одного шляху доступу, то вибирається той шлях, чий ранг вище, тому що за інших рівних умов він виконується швидше, ніж шлях з більш низьким рангом.

План виконання запиту в методі оптимізації по синтаксису (рисунок 2.1) формується з обраних шляхів доступу з максимальними рангами.

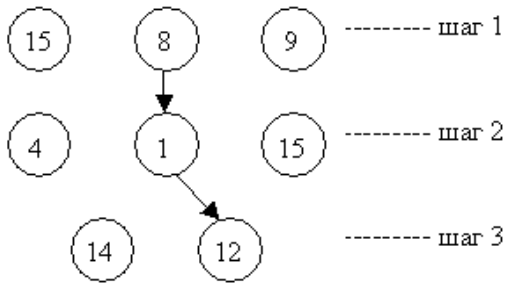

Рисунок 2.1 – Побудова плану виконання запиту в методі оптимізації по синтаксису

Оптимізація, заснована на правилах, забезпечує задовільну продуктивність системи в тих ситуаціях, коли евристики є точними. Однак часто загальновизнані правила не є точними. Для виявлення таких ситуацій оптимізатор запитів повинен розглядати характеристики даних, такі як:

– число рядків в таблиці;

– інтервал і розподіл значень даної колонки;

– довжина рядка і, відповідно, число рядків на фізичної сторінці диска;

– висота індексу;

– число термінальних (leaf) сторінок в індексі.

Ці характеристики даних можуть сильно впливати на ефективність обробки запиту. Використання таких характеристик призводить до наступного типу оптимізації.

2.3.4 Метод оптимізації, заснований на вартості

При використанні цього методу оптимізатор спочатку будує кілька можливих планів виконання запиту (зазвичай, 2–3 плани). Для вибору найбільш перспективних планів він застосовує деякі евристики, тобто правила, отримані дослідним шляхом. Ці правила дозволяють звузити простір пошуку оптимального плану завдяки тому, що неефективні плани відкидаються на самому початку і не розглядаються. Прикладом евристики може послужити таке правило: хорошим є такий план, в якому селекція проводиться на ранньому етапі виконання запиту. Евристики часто засновані на законах перетворення операцій реляційної алгебри.

Для кожного з побудованих планів розраховується його вартість. Вартість – це оцінка очікуваного часу виконання запиту з використанням конкретного плану виконання. При розрахунку вартості оптимізатор може враховувати такі параметри, як кількість необхідних ресурсів пам'яті, час операцій дискового введення-виведення, час процесора.

З безлічі можливих планів виконання запиту оптимізатор відповідно до критерію вибирає найкращий план (рисунок 2.2).

В якості критерію оптимізації може виступати:

– найкраща загальна продуктивність системи. Взагалі кажучи, її можна досягти, якщо з усіх планів вибирати ті, які вимагають найменше ресурсів (пам'яті і центрального процесора). Це дозволить збільшити ступінь

паралельності роботи системи і підвищити загальну продуктивність, хоча при цьому час виконання окремих запитів збільшиться;

– мінімальний час реакції – час, необхідний для обробки і видачі першого рядка. Цей критерій призначений для роботи в інтерактивному режимі, але може використовуватися тільки для запитів, які не містять агрегуються функцій і не вимагають сортування даних результату;

– мінімальні витрати часу на обробку всіх рядків, до яких звертається дана команда. Цей критерій використовується при роботі в пакетному режимі або в ситуації, коли неможливо видавати результат по частинах (наприклад, при використанні агрегуються функцій).

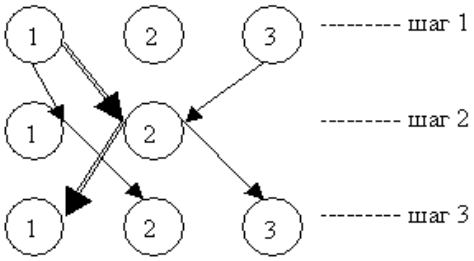

Рисунок 2.2 – Вибір найкращого плану

Вартість плану виконання запиту визначається на підставі відомостей про розподіл даних в таблицях, до яких звертається команда, і пов'язаних з ними кластерів і індексів. Ці відомості про розподіл значень даних називаються статистикою і зберігаються в словнику-довіднику даних.

Для таблиці статистика може включати в себе:

– загальна кількість блоків даних (сторінок пам'яті), виділенихтаблиці;

– кількість порожніх блоків даних (сторінок пам'яті);

– кількість записів в таблиці;

– середню довжину запису в таблиці;

– середня кількість записів на блок (сторінку) пам'яті.

Для кожного індексу статистика може містити такі дані, як:

– загальна кількість проіндексованих записів (воно може бутименше, ніж кількість записів в таблиці, тому що null-значення не індексуються);

– мінімальне і максимальне індексовані значення;

– кількість різних індексованих значень.

Розподіл значень в стовпці може бути відображено за допомогою гістограми, яка також входить в статистику. Для цього все безліч значень стовпця упорядковується і розбивається на N інтервалів. Приклади розподілів даних наведені на рисунку 2.3.

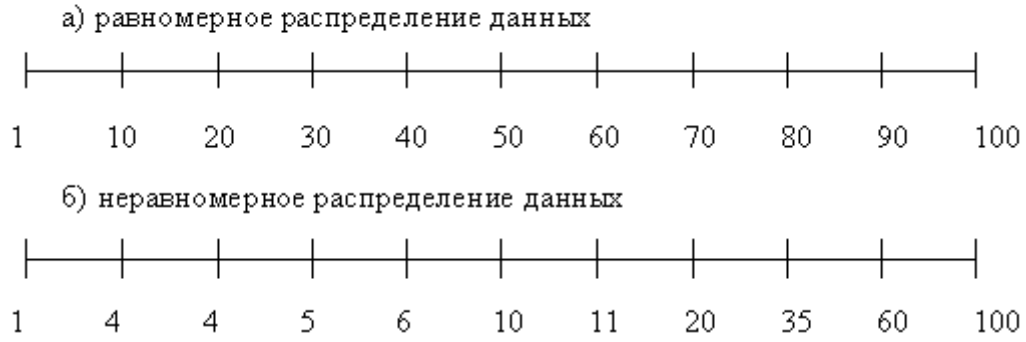

Рисунок 2.3 – Приклади розподілів даних

На рисунку 2.3а наведено розбиття на N = 10 інтервалів безлічі значень деякого стовпця F. Для рівномірного розподілу це означає, що в перших 10% записів це поле має значення від 1 до 10, в наступних 10% записів – від 11 до 20 і т.д.

Гістограма допомагає оцінити обсяг даних, які відповідають умові запиту. На рисунку 2.3б представлено нерівномірний розподіл даних для деякого стовпця F. До словника-довідник даних записуються отримані значення (1, 4, 4, 5, 6, 10, 11, 20, 35, 60, 100). При аналізі запиту, наприклад, з умовою (F <= 5) система зможе по цій гістограмі визначити, що через індекс доведеться вибрати не менше 30% записів таблиці.

Гістограми корисні тільки в тих випадках, коли вони відображають актуальне розподіл даних. Якщо розподіл змінюється при завантаженні або модифікації даних, гистограмму потрібно оновлювати.

Побудова гістограми марно в наступних випадках:

– значення стовпця розподілені рівномірно;

– стовпець не використовується в предиката запитів;

– значення стовпця унікальні і використовуються тільки в предиката еквівалентності.

Існують різні підходи до порядку збору статистики. Деякі СКБД постійно збирають статистичну інформацію, але це може зменшити швидкодію системи. Інші дозволяють здійснювати збір статистики періодично, наприклад, в період мінімального завантаження системи. Треті пропонують адміністратору спеціальні засоби для збору статистики, які запускаються інтерактивно по його команді. В останньому випадку в обов'язок адміністратора (адміністратора даних або додатків) входить вибір таблиць для аналізу і періодичне оновлення статистики даних.

Одним з найважливіших питань при роботі з оптимізатором є вибір режиму його роботи. Описані вище види оптимізації можуть бути задані на рівні примірника, сесії або SQL-вирази. Вказівка режиму оптимізації на рівні примірника здійснюють адміністратори баз даних за допомогою параметрів ініціалізації екземпляра.

Заданий таким чином режим оптимізації буде використовуватися тільки для даної сесії, при початку інший його потрібно вказати знову.

Як режим оптимізатора може бути задано одне з наступних значень.

CHOOSE – при вказівці цього значення буде обрана вартісна оптимізація, заснована на аналізі витрат. В іншому випадку буде використана оптимізація, заснована на аналізі правил.

RULE – при вказівці цього значення буде використана оптимізація, заснована на аналізі правил. Слід зазначити, що при вказівці тільки цього

значення (або відповідної підказки) буде використаний оптимізатор за правилами.

FIRST_ROWS – це значення використовується для мінімізації часу відгуку, тобто для зведення до мінімуму тимчасового інтервалу між початком виконання запиту і появою результатів на екрані. Це значення слід використовувати в системах, де критичним є час реакції. Оптимізатор ігнорує цю підказку для пропозицій DELETE і UPDATE, а також в тих пропозиціях SELECT, які містять хоча б одну конструкцію виду UNION, INTERSECT, MINUS, UNION ALL, GROUP BY, FOR UPDATE, DISTINCT або групові функції. Як вже зазначалося раніше, при обчисленні цих виразів неявно проводиться сортування і буде витягнуто весь набір даних.

ALL_ROWS – при вказівці цього значення буде використана оптимізація, заснована на аналізі витрат (при наявності відповідних статистичних даних) для мінімізації загальної кількості рядків, що обробляються системою за одиницю часу (в транзакціях за секунду). Це значення слід використовувати в високопродуктивних системах пакетної обробки. Наприклад, розглянемо пропозицію з'єднання таблиць, яке може бути виконане або операцією вкладених циклів, або операцією сортування злиття. Операція сортування–злиття швидше поверне весь результат запиту, тоді як операція вкладених циклів може швидше повернути перший рядок. Тому, якщо метою є найкраща пропускна здатність, оптимізатор швидше вибере з'єднання через сортування-злиття.

Якщо в запиті зазначено підказка ALL_ROWS або FIRST_ROWS, але словник даних не містить статистики для таблиць, перелічених у фразі FROM, то оптимізатор скористається відомостями про параметри зберігання цих таблиць, щоб оцінити статистику і на цій основі вибрати план виконання. Для завдання режиму оптимізації на рівні вираження слід використовувати ключові слова методів доступів, або використовувати одну з наведених вище значень (CHOOSE, RULE, FIRST_ROWS, ALL_ROWS). При використанні в SQL – вираженні будь-підказки, відмінною від RULE, здійснюється

автоматичний вибір оптимізатора за вартістю. Підказки, специфицирующие метод доступу:

– ROWID – використання ідентифікатора;

– CLUSTER – сканування ключа кластер;

– HASH – сканування хеш-індексу;

– INDEX – сканування індексу;

– INDEX_ASC – сканування індексу в порядку зростання;

– INDEX_DESC – сканування індексу в порядку убування;

– INDEX FFS – швидке повне сканування індексу;

– AND_EQUAL – використання декількох індексів зі злиттям результатів;

– FULL – повне сканування таблиці.

2.4 Оптимізація запитів в розподілених базах даних

Розподілені бази даних з'явилися як наслідок розвитку розподілених систем різного призначення і, в першу чергу, інформаційно-пошукових систем (ІПС), інтегруючих, як правило, безліч баз даних, орієнтованих на різні предметні області. Проектування (і в тому числі вибір) таких БД являє собою досить складну задачу, оскільки, крім проблем власне інтеграції та проектування локальних класичних БД, з'явилися нові проблеми, характерні саме для розподілених БД . Це проблеми:

– оптимізації розміщення розподілених БД по вузлах мережевої структури системи, що не підтримує в даний час сучасними CASE – системами (Computer Aided System Engineering);

– мережевий масштабованості БД для географічно розподілених систем;

– проектування та реалізації (використання) комунікаційного середовища системи пам'яті, як основи для занурення розподіленої БД;

– оптимізації розміщення даних в розподілених БД для забезпечення балансу завантаження;

– синхронізації доступу до даних і паралельної обробки запитів для забезпечення необхідної продуктивності системи;

– підтримання копій даних в декількох вузлах мережі для скорочення пересилань при виконанні розподілених запитів;

– фрагментації об'єктів БД і розширення простору пошуку варіантів виконання запитів;

– управління транзакціями (наприклад, коректне завершення розподілених транзакцій), багаторівневого захисту даних в БД і реалізації можливості отримання різноманітної за структурою і змістом інформації від різних типів джерел.

Відповідно до цього сформувалися нові вимоги до розподілених БД, з'явилися нові технології їх проектування.

Логічно єдина БД розділяється на фрагменти, кожен з яких зберігається на одному комп'ютері, а всі комп'ютери з'єднані лініями зв'язку. Кожен з цих фрагментів працює під управлінням своєї СКБД.

Основні функції ті ж, що і у СКБД. Додаткові функції:

– розширені служби установки з'єднань для передачі даних між вузлами мережі;

– розширені засоби ведення каталогу, що дозволяють зберегти відомості про розподіл об'єктів БД в мережі;

– засоби обробки розподілених запитів, що включають механізми оптимізації та організації віддаленого доступу;

– функції, що підтримують цілісність реплицируемой даних.

– розширені функції відновлення, що враховують можливості відмов в інших вузлах мережі або лініях зв'язку.

Існують різні методи підтримки розподіленості:

– фрагментація – розбиття БД або таблиці на кілька частин і зберігання цих частин на різних вузлах РБД;

– реплікація – створення і зберігання копій одних і тих же даних на різних вузлах РБД.

– розподілені обмеження цілісності – обмеження, для перевірки виконання яких потрібно звернення до іншого вузла РБД;

– розподілені запити – це запити на читання, які звертаються більш ніж до одного вузла РБД;

– розподілені транзакції – команди на зміну даних, які звертаються більш ніж до одного вузла РБД.

База даних фізично розподіляється по вузлах даних на основі фрагментації і реплікації даних. При наявності схеми реляційної бази даних кожне відношення фрагментируется на горизонтальні або вертикальні розділи. Горизонтальна фрагментаціяреалізуется за допомогою операції селекції, яка направляє кожен кортеж відносини в один з розділів, керуючись предикатом фрагментації. Наприклад, для відносини Employee можлива фрагментація відповідно до розташування робочих місць службовців. При вертикальній фрагментації ставлення ділиться на розділи за допомогою операції проекції. За рахунок фрагментації дані наближаються до місця їх найбільш інтенсивного використання, що потенційно знижує витрати на пересилання; зменшуються також розміри відносин, що беруть участь в призначених для користувача запити.

Розподіленим називається запит, який звертається до двох і більше вузлів РБД, але не оновлює на них дані. Запитувач вузол повинен визначити, що в запиті йде звернення до даних на іншому вузлі, виділити підзапит до віддаленого вузла і перенаправити його із сайтом.

Для централізованої СКБД весь процес складається зазвичай з двох кроків: декомпозиції запиту (query decomposition) і оптимізації запиту. Декомпозиція запиту – це трансляція його з мови SQL в вираз реляційної алгебри. В ході декомпозиції запит піддається семантичному аналізу; при цьому некоректні запити відкидаються, а коректні спрощуються. Спрощення полягає, зокрема, у виключенні надлишкових предикатів, які могли бути

привнесені за рахунок використання уявлень, а також виходячи з обмежень безпеки і семантичної цілісності. Спрощений запит перетворюється в алгебраїчну форму.

Для заданого SQL-запиту існує більш ніж одне алгебраїчне уявлення, причому деякі з них можуть бути краще за інших. Якість алгебраїчного виразу визначається виходячи з обсягу витрат, необхідних для його обчислення. Традиційна процедура полягає в тому, щоб спочатку оттранслировать SQL-запит в яке-небудь вираз, а потім, застосовуючи правила еквівалентних алгебраїчних перетворень, отримувати з нього інші алгебраїчні перетворення, поки не буде знайдено оптимальне. При пошуку найкращого вираження використовується функція вартості, відповідно до якої обчислюється сума витрат, необхідних для виконання запиту. Цей процес і називається оптимізацією запитів.

У розподіленої СКБД між кроками декомпозиції і оптимізації запиту включаються ще два дії: локалізація даних (data localization) і глобальна оптимізація запиту (global query optimization).

Вихідною інформацією для локалізації даних служить вихідне вираз, отримане на кроці декомпозиції запиту. У вихідному алгебраическом вираженні фігурують глобальні відносини без урахування їх фрагментації або розподілу. Основна роль локалізації даних полягає в тому, щоб локалізувати беруть участь в запиті дані, використовуючи інформацію про їх розподіл. На цьому кроці виявляються фрагменти, реально беруть участь в запиті, і запит перетворюється до форми, де операції застосовуються вже не до глобальних відносин, а до фрагментів.

Як зазначалося вище, правила фрагментації виражаються за допомогою реляційних операцій (селекції для горизонтальної фрагментації і проекції для вертикальної). Розподілені відносини реконструюються шляхом застосування інверсії правил фрагментації. Це називається програмою локалізації.

Програма локалізації для горизонтально (вертикально) фрагментированного відносини є об'єднанням (union) (з'єднання (join))

відповідних фрагментів. Таким чином, на етапі локалізації даних кожне глобальне ставлення запит замінюється його програмою локалізації, а потім результуючий Фрагментний запит спрощується і реструктурується з метою отримання іншого хорошого запиту. Для спрощення та реструктуризації можуть використовуватися ті ж правила, що і на кроці декомпозиції. Як і на етапі декомпозиції, остаточний запит над фрагментами може бути ще далекий від оптимального; даний процес лише виключає погані алгебраїчні запити.

Вихідною інформацією для третього кроку є фрагментний запит. Мета глобальної оптимізації – знайти стратегію виконання запиту, близьку до оптимальної. Стратегію виконання розподіленого запиту можна виразити в термінах операцій реляційної алгебри і комунікаційних примітивів (операцій send / receive), використовуваних для пересилання даних між вузлами. На попередніх етапах запит вже був до певної міри оптимізований, зокрема, за рахунок видалення надлишкових виразів.

Однак проведена оптимізація не залежала від характеристик фрагментів, наприклад їх потужності. Крім того, на попередніх кроках ще не враховувалися комунікаційні операції. Шляхом зміни порядку операцій всередині одного фрагментного запиту можна отримати багато еквівалентних планів його виконання. Оптимізація запиту полягає в знаходженні найкращого плану з безлічі можливих планів, досліджуваних оптимізатором.

Оптимізатор запитів зазвичай представляється у вигляді трьох компонентів: простір пошуку, модель вартості і стратегія пошуку. Простір пошуку – це безліч альтернативних планів виконання початкового запиту. Ці плани еквівалентні в тому сенсі, що вони дають один і той же результат, але розрізняються порядком і способами виконання окремих операцій.

Модель вартості – це спосіб оцінити вартість даного плану виконання запиту. Для досягнення точності модель вартості повинна грунтуватися на точних знаннях про середовищі паралельних обчислень.

Стратегія пошуку – це спосіб обходу простору пошуку та вибору найкращого плану. Вона визначає, які плани і в якому порядку слід вибирати і оцінювати.

У розподіленої середовищі функція вартості, часто обумовлена в одиницях часу, оцінює витрати обчислювальних ресурсів, таких як дисковий простір, число обмінів з дисками, час центрального процесора, комунікації і т.д. Зазвичай це деяка зважена сума витрат введення-виведення, центрального процесора і комунікацій. У розподілених СКБД застосовується спрощений підхід, коли в якості найбільш значущих розглядаються лише комунікаційні витрати.

Це справедливо для глобальних мереж, де через обмежену пропускну спроможність ліній зв'язку пересилання даних обходяться значно дорожче, ніж при локальній обробці. Щоб визначити порядок виконання операцій, необхідно оцінити вартості виконання планів з іншим порядком операцій. Визначення вартості виконання до реального виконання запиту (статична оптимізація) засновано на статистиці фрагментів і формулах для оцінки потужності результатів реляційних операцій. Таким чином, рішення, що приймаються в ході оптимізації, залежать від наявної статистики фрагментів.

Важливим аспектом оптимізації запитів є порядок виконання з'єднань, оскільки його зміна може призвести до прискорення на декількох порядків. Базовий метод оптимізації послідовності розподілених операцій з'єднання полягає в застосуванні операції напівз'єднання (semijoin). Основна перевага напівз'єднання в розподіленої системі – це скорочення розмірів операндів, що беруть участь в з'єднаннях, і, отже, комунікаційних витрат. Однак в більш сучасних методах, які враховують, поряд з комунікаційним витратами, також і витрати на локальну обробку, напівз'єднання не використовуються, оскільки вони призводять до збільшення обсягу локальної обробки. Результатом роботи глобального оптимізатора є оптимізоване алгебраїчне вираз, що включає комунікаційні операції над фрагментами.

2.5 Побудова оптимальної послідовності з'єднання відносин в запитах реляційної бази даних

Одним з важливих властивостей, що впливають на ефективність запиту, є послідовність обчислення з'єднань в ланцюжку операцій реляційної алгебри. З'єднання – це визначальна операція, так як час її виконання пропорційно часу виконання операції об'єднання відносин. Як критерій складності структури запиту можна вибрати оцінку кількості проміжних операцій з'єднання і порядок виконання комутуючих і асоціативних операцій.

При детальному розгляді операції з'єднання, можна відзначити важливість способу її обчислення. Обраний спосіб може визначатися або тим, як зберігається відношення, чи є пов'язані додаткові структури. При цьому процедура обчислення може проводитися різними методами, наприклад, послідовним або індексним доступом, з урахуванням або без урахування дублікатів тощо

Для вирішення питання ефективної реалізації запиту необхідно розглянути техніку управління файлами, взаємозв'язок між зверненнями до зовнішньої пам'яті і операціями реляційної алгебри, а також деякі технічні характеристики обчислювальної техніки.

Розглянуті способи реалізації запиту використовують властивості операцій алгебри і теорії множин. Подальші викладки будуть грунтуватися на властивостях операції з'єднання і правилах, що дозволяють здійснювати алгебраїчні перетворення.

Алгоритм перебору повинен вибрати оптимальний план виконання запиту на основі дослідження простору пошуку. Алгоритми перебору багатьох існуючих систем розраховані на вибір тільки оптимального порядку лінійних з'єднань.

На практиці бажано мати такий алгоритм перебору, який міг би легко пристосовуватися до змін простору пошуку через додавання нових перетворень, додавання нових фізичних операцій (наприклад, нових

реалізацій з'єднань) і до змін методів оцінки вартості. Сучасні архітектури оптимізації побудовані на основі цієї парадигми і називаються розширюваними оптимізаторами.

Побудова расширяемого оптимізатора – складне завдання, оскільки необхідно не тільки наявність поліпшеного алгоритму перебору, але і забезпечення інфраструктури для розвитку техніки оптимізації. Однак спільність архітектури повинна бути збалансована з потребою ефективного перебору. Таким чином, однією з важливих складових частин оптимізатора є наявність ефективного методу перебору, що визначає послідовність виконання операцій запиту.

Одним з критеріїв підвищення ефективності виконання запиту є зменшення числа кортежів у відносинах при багаторазовому з'єднанні. За умови, що операція з'єднання передбачає звернення до кожного кортежу з'єднуються відносин, виникає завдання відшукання такої послідовності, яка гарантує найменше сумарне число звернень до кортежів при послідовному з'єднанні.

3 ПЛАНИ ВИКОНАННЯ ЗАПИТІВ

3.1 Вибір альтернативних планів виконання запитів

Все оптимізують перетворення запитів, які ми розглядали в попередніх підрозділах, залишали внутрішнє уявлення запиту непроцедурного. Навіть якщо говорити про процедурності на рівні виконання реляційних операторів в виразах реляційної алгебри (для тих випадків, коли внутрішнім поданням є вираз реляційної алгебри), то ця процедурность дуже умовна, оскільки, наприклад, порядок виконання операцій реляційного з'єднання зовсім не зобов'язаний збігатися з порядком, в якому вони зустрічаються в вираженні. Більш того, при реальному виконанні запиту ту ж операцію з'єднання можна виконати багатьма способами.

Процедурним поданням або планом виконання запиту називається таке його уявлення, в якому в деталізованої формі відображений порядок виконання операцій доступу до бази даних фізичного рівня. Як правило, в СКБД, орієнтованих на використання традиційної апаратури без використання спеціалізованих процесорів або пристроїв зовнішньої пам'яті, виділяється підсистема управління даними на фізичному рівні. Така підсистема називається RSS (Relational Storage System) і забезпечує простий інтерфейс, що дозволяє послідовно переглядати кортежі відносин, що задовольняють заданим умовам на значення полів, з використанням індексів або простим скануванням сторінок бази даних. Крім того, RSS дозволяє виробляти відсортовані тимчасові файли і заносити, видаляти і модифікувати індивідуальні кортежі. Аналогічні підсистеми явно або неявно виділяються у всіх подібних СКБД.

Природно, що в цих умовах до виконання запиту необхідно виробити його процедурне подання. Крім того, оскільки воно, взагалі кажучи, вибирається неоднозначно, необхідно вибрати серед альтернативних планів запиту один, найбільш задовільний відповідно до деякими критеріями. Як

правило, критерієм вибору плану виконання запиту є мінімізація вартості виконання.

Тим самим, при обробці запиту на стадії, наступної за логічної оптимізацією, вирішуються два завдання.

Перше завдання: виходячи з одержуваного після проведення логічної оптимізації внутрішнього подання запиту та інформації, що характеризує керуючі структури бази даних (наприклад, індекси), перейти до групи потенційно можливих планів виконання даного запиту. Друге завдання: оцінити вартість виконання запиту відповідно до кожного альтернативним планом і вибрати план з найменшою вартістю.

При традиційному підході до організації оптимізаторів обидва завдання вирішуються на основі фіксованих вбудованих в оптимізатор алгоритмів. Наприклад, оптимізатор може бути розрахований на те, що обмеження будь-якого відношення відповідно до заданого предикатом може бути виконано шляхом послідовного перегляду відносини з використанням будь-якого з існуючих для нього індексів і прямим послідовним переглядом.Компонент оптимізатора, який відає породженням безлічі альтернативних планів виконання запиту, базується на стратегіях декомпозиції запиту на елементарні складові і стратегіях виконання елементарних складових.

Перша група стратегій забезпечує простір пошуку оптимального плану виконання запиту, друга спрямована на те, щоб в цьому просторі дійсно знаходилися ефективні плани виконання запиту. Очевидно, що ключем до забезпечення ефективного виконання складного запиту є наявність ефективних стратегій виконання елементарних складових. Це виключно важливе питання, якому приділяється величезна увага в теорії і практиці реляційних баз даних. Особливо багато робіт присвячується виробленню ефективних алгоритмів реалізації з'єднань відносин і аналогічних операцій.

Якщо оптимізаторові вдалося згенерувати безліч планів виконання запиту на основі розумних стратегій декомпозиції і ефективних стратегій виконання елементарних операцій, то цього ще мало – потрібно зуміти

вибрати один план, відповідно до якого буде відбуватися реальне виконання запиту. Якщо цей вибір буде невірним, то запит буде виконаний неефективно. Перш за все необхідно визначити, що розуміється під ефективність виконання запиту.

Взагалі кажучи, це поняття неоднозначно і залежить від специфіки операційного середовища СКБД. В одних умовах можна вважати, що ефективність виконання запиту визначається часом його виконання, тобто реактивністю системи по відношенню до оброблюваних нею запитам. В інших умовах визначальним є загальна пропускна здатність системи по відношенню до суміші паралельно виконуваних запитів.

В інших умовах визначальним є загальна пропускна здатність системи по відношенню до суміші паралельно виконуваних запитів. Тоді мірою ефективності запиту можна вважати кількість системних ресурсів, необхідних для його виконання: чим менше ресурсів потрібно, тим запит ефективніше.

Якщо користувачі СКБД працюють в умовах бюджетної системи, то розумною мірою ефективності може бути бюджетна вартість виконання запиту і т.д. Зауважимо, що в будь-якому випадку оцінка ефективності виконання запиту спирається на ресурсні характеристики відповідного плану; при використанні різних заходів вони просто по-різному беруть участь в оцінці.

3.2 Генерація планів

При традиційному підході до організації оптимізаторів обидва завдання вирішуються на основі фіксованих вбудованих в оптимізатор алгоритмів. Оптимізатор може бути розрахований на те, що обмеження будь-якого відношення відповідно до заданого предикатом може бути виконано шляхом деякого послідовного перегляду відносини. Так, запит

```
SELECT empName
FROM EMP
WHERE departmentNo = 6
AND salary> 15000
```

може виконуватися:

– послідовним скануванням відносини EMP з виборомкортежів з DEPT # = 6 і SALARY> 15.000;

– скануванням відносини через індекс I1, певний на поле DEPT #, з умовою доступу до індексу DEPT # = 6 і умовою вибірки кортежу SALARY> 15.000;

– скануванням відносини через індекс I2, певний на поле SALARY, з умовою доступу до індексу SALARY> 15.000 і умовою вибірки кортежу DEPT # = 6.

Аналогічно, фіксовані і стратегії виконання більш складних операцій – реляційних з'єднань відносин, обчислення агрегатних функцій на групах кортежів відносини і т.д.

Компонент оптимізатора, що генерує виконуються плани запитів, має досить складну організацію. Генерація плану виконання складного запиту – це багатоетапний процес, в ході якого враховуються властивості створюваних при виконанні запиту за даним планом тимчасових об'єктів бази даних.

Наприклад, нехай запит заданий над трьома відносинами і в ньому є два предиката з'єднання:

```
SELECT R1.C1, R2.C2, R3.C3
FROM R1, R2, R3
WHERE R1.C4 = R2.C5
AND R2.C5 = R3.C6
```

Тоді, якщо в плані запиту вибирається порядок виконання з'єднань спочатку R1 з R2, а потім отриманого тимчасового відносини – з R3, і при цьому для виконання першого з'єднання вибирається метод угруповань зі злиттям, то тимчасове ставлення буде свідомо відсортовано за C5, і одна

сортування не буде потрібно, якщо і друге з'єднання буде виконуватися тим же методом.

Компонент оптимізатора, який відає породженням безлічі альтернативних планів виконання запиту, базується на стратегіях декомпозиції запиту на елементарні складові і стратегіях виконання елементарних складових. Перша група стратегій визначає простір пошуку оптимального плану виконання запиту, друга спрямована на те, щоб в цьому просторі дійсно знаходилися ефективні плани виконання запиту. Ключем до забезпечення ефективного виконання складного запиту є наявність ефективних стратегій виконання елементарних складових. Це дуже важливе питання, але тут ми його не торкаємося: оптимізатор запитів користується заданими стратегіями. Розглянемо більш актуальну для оптимізатора проблему – обгрунтований вибір плану виконання запиту з безлічі альтернативних планів.

3.3 Оцінка вартості плану запиту

Після генерації безлічі планів виконання запиту на основі розумних стратегій декомпозиції і ефективних стратегій виконання елементарних операцій потрібно вибрати один план, відповідно до якого буде відбуватися реальне виконання запиту. При неправильному виборі запит буде виконаний неефективно. Перш за все необхідно визначити, що розуміється під ефективністю виконання запиту. Це поняття неоднозначно і залежить від специфіки операційного середовища СКБД. В одних умовах можна вважати, що ефективність виконання запиту визначається часом його виконання, тобто реактивністю системи по відношенню до оброблюваних нею запитам. В інших умовах визначальною є загальна пропускна здатність системи по відношенню до суміші паралельно виконуваних запитів. Тоді мірою ефективності запиту можна вважати кількість системних ресурсів, необхідних для його виконання.

Дотримуючись прийнятої термінології, ми будемо говорити про вартість плану виконання запиту, яка визначається ресурсами процесора і пристроїв зовнішньої пам'яті, які витрачаються при виконанні запиту.

У традиційних реляційних СКБД виділяється підсистема управління доступом до даних на зовнішній пам'яті). Дані на зовнішній пам'яті традиційно зберігаються в блокованому вигляді; база даних займає деякий набір блоків одного або декількох дискових томів. Передбачається, що кошти доступу до блоків зовнішньої пам'яті породжують незрівнянно менші накладні витрати.

Як правило, підсистема управління доступом до даних на зовнішній пам'яті здійснює буферизацію блоків бази даних в оперативній пам'яті. Кожен блок бази даних, прочитаний в оперативну пам'ять для виконання запиту, зберігається в одному з буферів буферного пулу СКБД, поки не буде витіснений з нього іншим блоком бази даних. Ця особливість СКБД істотна для підвищення загальної ефективності системи, але не враховується (за винятком приватного, але важливого випадку) при оцінках вартостей планів виконання запиту. У будь-якому випадку компонент вартості виконання запиту, пов'язаний з ресурсами пристроїв зовнішньої пам'яті, монотонно залежить від числа блоків зовнішньої пам'яті, доступ до яких буде потрібно при виконанні запиту.

З іншого боку, число блоків зовнішньої пам'яті, доступ до яких потрібно при виконанні запиту, монотонно залежить від числа кортежів, яких торкається запитом.

З цього випливає важливість показника предиката обмеження, що характеризує частку кортежів відносини, які задовольняють даному предикату, і званого ступенем селективності предиката. Точність оцінок ступенів селективності визначає точність загальних оцінок і правильність вибору оптимального плану запиту.

Ступінь селективності залежить від виду операції порівняння, значення константи і розподілу значень поля відносини в базі даних. Перші два

параметри селективності можуть бути відомі під час проведення оцінок, але справжні розподілу значень полів відносин, як правило, невідомі. Існує ряд підходів до наближених визначень розподілів на основі статистичної інформації.

Програмні засоби, що дозволяють отримати плани виконання запитів, можна розділити на 2 групи:

– кошти, що дозволяють отримати передбачуваний план виконання запиту;

– кошти, що дозволяють отримати реальний план виконання запиту.

До засобів, що дозволяє отримати передбачуваний план виконання запиту, відносяться Toad, SQL Navigator, PL / SQL Developer і ін. Це важливий момент, оскільки треба враховувати, що реальний план виконання може відрізнятися від того, що показують ці програмні засоби. Вони видають ряд показників ресурсоємності запиту, серед яких основними є:

– cost – вартість виконання;

– cardinality (або Rows) – кардинальність.

Чим більше значення цих показників, тим менш ефективний запит.

Разом з тим, багаторічний досвід оптимізації показує, що якісний аналіз ефективності запиту вимагає, крім Cost і Cardinality, розгляду інших додаткових показників:

– CPU Cost – процесорна вартість виконання;

– IO Cost – вартість введення-виведення;

– TempSpace – показник використання дискового простору.

Якщо дисковий простір використовується (при нестачі оперативної пам'яті для виконання запиту, як правило, для проведення угруповань, угруповань і т.д.), то з великою ймовірністю можна говорити про неефективність запиту. Зазначені додаткові параметри з відповідною настроюванням можна побачити в PL / SQL Developer і при їх відповідній настройці. Для PL / SQL Developer в вікні з планом виконання треба вибрати зображення гайкового ключа, увійти в вікно Preferensec додати додаткові

параметри в SelectColumn, після чого і натиснути OK. У Toad в плані виконання по правій кнопці миші вибирається директива DisplayMode, а далі Graphic, після чого з'являється дерево, в якому по кожному аркушу натисканням мишки можна побачити додаткові параметри: CPU Cost, IO Cost, Cardinality. Структура плану запиту, зазначеного вище, у вигляді дерева приведена нижче.

Передбачуваний план виконання запиту з Cost і Cardinality можна також отримати, виконавши після аналізованого запиту інший запит, формує план виконання:

Select * from table (dbms_xplan.display_cursor ());

Додатково в плані виконання запиту видається значення SQL_ID запиту, який можна використовувати для отримання реального плану виконання запиту з набором як основних (Cost, Cardinality), так і додаткових показників через запит:

Select * from v $ sql_plan where sql_id = 'SQL_ID';

Реальний план виконання запиту і вказаний вище перелік характеристик для аналізу ресурсоемкого запиту дають динамічні уявлення Oracle: V $ SQL_PLAN і V $ SQL_PLAN_MONITOR.

Файлі трасування це ще один засіб отримання реального плану виконання. Це досить сильний засіб діагностики і оптимізації запиту. Результат пишеться в відповідну директорію сервера, яка знаходитися із запиту.

Файлі трасування для зручності читання розшифровується утилітою Tkprof (при певному навику аналізувати можна без розшифровки, в цьому випадку маємо більш детальну інформацію).

Ще одним із засобів отримання реального плану виконання запиту з отриманням рекомендацій щодо його оптимізації є засіб Oracle SQLTUNE.

Для аналізу запиту запускається PL / SQL блок (наприклад, в Toad або PL / SQL Developer), в якому є стандартні рядки і аналізований запит.

Для роботи SQLTUNE необхідно як мінімум з під SYSTEM видати права на роботу з SQLTUNE схемою, в якій запускається PL / SQL блок. Наприклад, для видачі прав на схему HIST видається GRANT ADVISOR TO HIST;

В результаті роботи SQLTUNE видає рекомендації (якщо Oracle вважатиме, що є що рекомендувати). Рекомендаціями можуть бути: зібрати статистику, побудувати індекс, запустити команду створення нового ефективного плану і т.д.

4 АНАЛІЗ КОРИСТУВАЦЬКИХ ЗАПИТІВ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ОПТИМІЗАЦІЇ СТРУКТУРИ БД

4.1 Функціональне моделювання інформаційної технології

Задача аналізу тексту та результату SQL-запиту полягає у накопиченні статистики та виділенні із тексту SQL-запитів таких аналітичних характеристик, як перелік відношень, атрибутів, та кортежів, до яких виконується звернення. Вхідними даними виступають SQL-запити та БД КІС. Ресурсами виступають системи профайлінгу та реляційні СКБД, керуючими впливами – стандарти мови SQL та реляційна модель даних, результатом є набір аналітичних характеристик SQL-запиту.

Задача обліку SQL-запитів КІС до БД включає у себе розробку структури та наповнення даними реляційної та багатовимірної БД для отримання необхідних зрізів даних за набором аналітичних характеристик запиту (застосунку, місця розташування, ролі користувача, та ін.). Вхідними даними виступає БД КІС та набір SQL-запитів із аналітичними характеристиками. Ресурсами є системи управління реляційними та багатовимірними БД, та ЛПР, що виконує заповнення рівня маркеру представленості для елементів вимірів ББД. Керуючими впливами є стандарти та вимоги до реляційної та багатовимірної моделей даних. Виходом є підмножина SQL-запитів, що є результатом операцій зрізу та консолідації за набором вимірів відношення–кортеж або відношення–атрибут.

За виходом, отриманим із попередньої задачі, виконується розрахунок рівня маркеру представленості даних для атрибутів та кортежів відношення. При цьому, керуючим впливом є багатовимірна модель, а ресурсом – система OLAP. Результатом є конкретні значення рівня маркеру представленості даних для кожного кортежу та атрибуту таблиці.

Останньою є задача визначення оптимального граничного значення маркеру представленості даних на вузлі КІС, для якої входом є БД КІС, зрізи даних SQL-запитів та рівень маркеру представленості атрибутів та кортежів відношення; керуючим впливом – методи БКО, а ресурсом – ЛПР (при заповненні матриці попарних порівнянь для розрахунку векторів відносних та глобальних пріоритетів альтернатив). Виходом є рішення про представлення або непредставлення відповідних даних на вузлі РКІС.

Перші дві із наведених задач є предметом технології визначення аналітичних характеристик SQL-запитів, їх обліку та аналізу, і детально розглянуті у даному розділі.

4.2 Задача парсингу SQL-запитів

Задача аналізу тексту та результату SQL-запиту з метою виявлення його аналітичних характеристик у вигляді набору відношень, атрибутів та кортежів, представлена у вигляді функціональної моделі IDEF0 [21].

Виділено задачі накопичення статистичних даних SQL-запитів до БД КІС, створення граматики мови SQL, парсинг тексту запиту та визначення переліку кортежів відношення, що задіяні у межах запиту. Перша із них має вхідним потоком множину SQL-запитів КІС, керується реляційною моделлю даних, та використовує реляційні СКБД і механізми профайлінгу.

Зауважимо, що на даному етапі механізми профайлінгу вже дозволяють визначити частину необхідних аналітичних характеристик запиту. Задача створення граматики керується синтаксисом мови SQL та дає на виході класи лексора та парсера тексту SQL-команд, які використовуються задачею парсингу для отримання ієрархічного дерева відношень та їх атрибутів, що використовуються у межах запиту [22], [23], [24].

Окремою паралельною задачею є визначення переліку кортежів відношення, вхідними потоками якої виступають SQL-запити, структура та дані БД КІС, у межах якої запит розбивається на множину підзапитів,

кількість елементів якої відповідає кількості задіяних відношень. Виходи задач накопичення статистики запитів, парсингу та визначення діапазону ключів об’єднуються, і дають за виході повний перелік необхідних аналітичних характеристик SQL-запиту. Більшість сучасних СКБД мають у своєму розпорядженні розвинені механізми з відслідковування користувацької активності при роботі з БД [25], [26].

Також на ринку ПЗ присутні програмні продукти сторонніх розробників, що дозволяють фіксувати сеанси взаємодії із сервером та відображають список запитів із прив’язкою до робочої станції, програмного забезпечення та користувача. Крім того, переважна більшість ІС мають у своєму розпорядженні власні інструменти профайлінгу користувацьких запитів. Вибір того чи іншого інструменту залежить від СКБД, що використовується для роботи із БД, наявності в ІС власних механізмів та від глибини необхідного аналізу отриманих даних.

В рамках даного дослідження обрано програмний продукт SQL Profiler, що входить до стандартного інсталяційного пакету SQL Server 2019. Даний вибір обумовлено достатнім переліком аналітичних властивостей, що надає функціонал ПЗ та сумісністю із версією СКБД, що використовується на підприємстві – базі автоматизації. При зміні вхідних умов вибір може бути змінено на користь іншого ПЗ.

Створивши новий сеанс профайлінгу, є можливість автоматичного створення таблиці, у яку записуються результати спостереження за користувацькою активністю. У ході даного дослідження було обрано перелік полів, що можна побачитиу таблиці profilingResultSet, який наводиться під час розгляду логічної та фізичної моделей БД збереження даних SQL-запитів. Даний перелік дозволяє отримати необхідні для проведення подальшого аналізу дані [27].

Задачі створення граматики мови T-SQL та парсинту SQL-запитів мають на меті визначити набір відношень, їх атрибутів, а також множину вкладених підзапитів, що задіяні у межах кожного SQL-запиту. Вхідними

даними виступає список запитів, що є результатом розв’язання задачі накопичення статистичних даних, при створенні граматики керуючим впливом єстандартами мови SQL та на виході отримано перелік відношень та атрибутів, задіяних у межах запиту. Також розв’язання задачі створення граматики робить можливим генерацію класів лексора та парсера SQL-запиту для виконання подальшого парсингу тексту [28].

Кожна мова програмування має правила, які визначають синтаксичну структуру програмного коду. Синтаксис конструкцій мови програмування може бути описаний за допомогою контекстно-вільних граматик або нотації БНФ (Backus-Naur Form, форм Бекуса-Наура). До переваг граматик відносять: точність специфікації; структурованість; гнучкість; автоматична побудова синтаксичних аналізаторів.

Формально граматика визначається, як

$$G = (V_n, V_t, S, P), \tag{4.1}$$

де V_n і V_t– відповідно множини нетермінальних і термінальних символів, що не перетинаються;

S– виділений символ у V_n, що звичайно називають вихідним або початковим символом;

P– кінцева множина продукцій (правил), за якими нетермінальні символи визначаються як упорядкована послідовність термінальних та/або нетермінальних символів.

Об'єднання множин V_n і V_t називають словником граматики.

Найбільш ефективні спадні й висхідні методи граматичного розбору працюють тільки з підкласами граматик, однак деякі із цих підкласів, такі як LL-граматики і LR-граматики, досить виразні для опису більшості синтаксичних конструкцій мов програмування. Синтаксичні аналізатори для класу LR-граматик, як правило, створюються за допомогою автоматизованих генераторів (YACC, BIZON, ANTLR та інші).

SQL – декларативна мова програмування для взаємодії користувача з БД, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних та її модифікації, системи контролю за доступом до бази даних. SQL складається з наступних підрозділів: Data Definition Language (DDL) – робота зі структурою бази; Data Manipulation Language (DML) – робота з даними; Data Control Language (DCL) – робота з правами; Transaction Control Language (TCL)– робота з транзакціями. DML – це сімейство комп'ютерних мов, що використовуються в комп'ютерних програмах або користувачами баз даних для отримання, вставки, видалення або зміни даних в базах даних. Зараз найпопулярнішою DML є SQL, що використовується для отримання і маніпулювання даними в реляційній БД [29-31].

Нижче наведено спрощений синтаксис команди DML «SELECT»– оператора мови SQL, що повертає результуючий набір рядків однієї чи багатьох відношень. Синтаксис оператора «SELECT» є досить розгалуженим, проте в спрощеному вигляді його можна описати так (лістинг 4.1):

Лістинг 4.1 – Синтаксис оператора «SELECT»

```
SELECT <список_вибірки>
[ INTO <нова_таблиця> ]
FROM <таблиця>
[ WHERE <умови_пошуку> ]
[ GROUP BY <групувати_по_умові> ]
[ HAVING <умови_пошуку> ]
[ ORDER BY <сортувати_по_умові> [ ASC | DESC ] ]
```

У більшості застосунків, команда «SELECT» зустрічається найчастіше. Реальний рівень складності команди «SELECT» значно вищий, наряду із цим задача ускладняється наявністю різних діалектів мови SQL, що використовується при роботі із різними СКБД. Відповідно, працюючи із певною СКБД, маємо необхідність у створенні специфічного механізму

лінгвістичного та синтаксичного аналізу тесту запиту. У якості шляхурозв’язання задачі запропоновано використання програмного продукту ANTLR на базі проекту Java 1.8 для парсингу користувацького запиту із попереднім створенням граматики для відповідного діалекту SQL [32].

Надалі створення граматики розглядається на прикладі мови T-SQL для СКБД SQL Server. Даний вибір обумовлено СКБД системи обліку користувацьких запитів, а також приналежність основної БД підприємства – тестової бази автоматизації до сімейства SQL Server.

При використанні даної технології для парсингу запитів інших діалектів SQL виникає необхідність у розробці окремого файлу граматикиіз подальшою генерацією за допомогою ANTLR відповідних класів лексору та парсеру. Використання основних положень шаблонного проектування (Design Pattern) при реалізації інформаційної технології дозволяє виконувати дану операцію без необхідності зміни коду класів верхнього рівня. Даний факт свідчить про високий рівень адаптованості технології під різні діалекти SQL.

Для створення граматики, виконується формальний опис всіх використовуваних у користувацьких запитах SQL-команд. При реалізації даного завдання для T-SQL було використано документацією Microsoft SQL Server Language References, а саме Transact-SQL Reference (Database Engine) [33].

Так, команда «SELECT» в T-SQL формалізується наступним чином (лістинг 4.2), але наведений фрагмент не є завершеною граматикою для команди «SELECT», та потребує подальшого визначення таких токенів, як «common_table_expression», «order_by_expression», «column_position», «FOR Clause», «query_hint» та ін.

Лістинг 4.2 – Команда «SELECT» в T-SQL

```
<SELECT statement> ::=
    [ WITH { [ XMLNAMESPACES ,] [ <common_table_expression>
[,...n] ] } ]
<query_expression>
    [ ORDER BY { order_by_expression | column_position [ ASC
| DESC ] }
  [ ,...n ] ]
    [ <FOR Clause>]
    [ OPTION ( <query_hint> [ ,...n ] ) ]

<query_expression> ::=
    { <query_specification> | ( <query_expression> ) }
    [  { UNION [ ALL ] | EXCEPT | INTERSECT }
<query_specification> | ( <query_expression> ) [...n ] ]

<query_specification> ::=
SELECT [ ALL | DISTINCT ]
    [TOP ( expression ) [PERCENT] [ WITH TIES ] ]
<select_list>
    [ INTO new_table ]
    [ FROM { <table_source> } [ ,...n ] ]
    [ WHERE <search_condition> ]
    [ <GROUP BY> ]
    [ HAVING < search_condition > ]
```

Генератор парсерів ANTLR є LL (*),існує вже понад 25 років, зараз його розробка ведеться на GitHub. В даний момент він дозволяє генерувати парсери на різних мовах, серед яких Java, C#, Python та JavaScript. Головна частина проєкту – генератор парсерів, що є консольною програмою, якій на вхід подається файл з описом граматики кінцевого компілятора. Генератор читає цей файл, і у відповідь генерує вихідні коди компілятора або помилки, якщо граматика порушена. Фактично ANTLR є інструментом, який

перетворює граматику на парсер/лексор в Java (або іншої мови програмування із переліку, що підтримуються).

Використання ПЗ ANTLR у режимі командного рядку має свої переваги, однак є досить незручним при написанні великих граматик, тестуванні отриманих результатів та пошуку помилок. Існує ряд розширень, розроблених під більшість популярних інтегрованих середовищ розробки підтримуваних ANTLR мов програмування, що дозволяють інтегрувати інструменти ANTLR із середовищем розробки та зробити процес створення, відлагодження та інтеграції граматик із основним проектом більш зручним та швидким.

Правила граматики, що представлені БНФ, можна представити у графічній формі, що робить граматику більш наглядною. Аналіз сполучень різноманітних елементів рядка для визначення того, чи є або не є даний рядок символів реченням мови, називається граматичним розбором. Для синтаксично правильного речення за допомогою створеної граматики можна побудувати дерево граматичного розбору.

Нарисунку 4.1 наведено приклад такого дерева для запиту «select (select number from groups where id=students.id), name, age from students», що звертається до декількох атрибутів і має у своєму складі один вкладений запит.

Далі кожен запит представляється у вигляді об’єкту, що має наступні атрибути: робочу станцію, користувача та програмне забезпечення, від яких надійшов запит; сам текст запиту; колекцію таблиць, до яких звертається запит; та колекцію вкладених запитів, якщо такі мають місце.

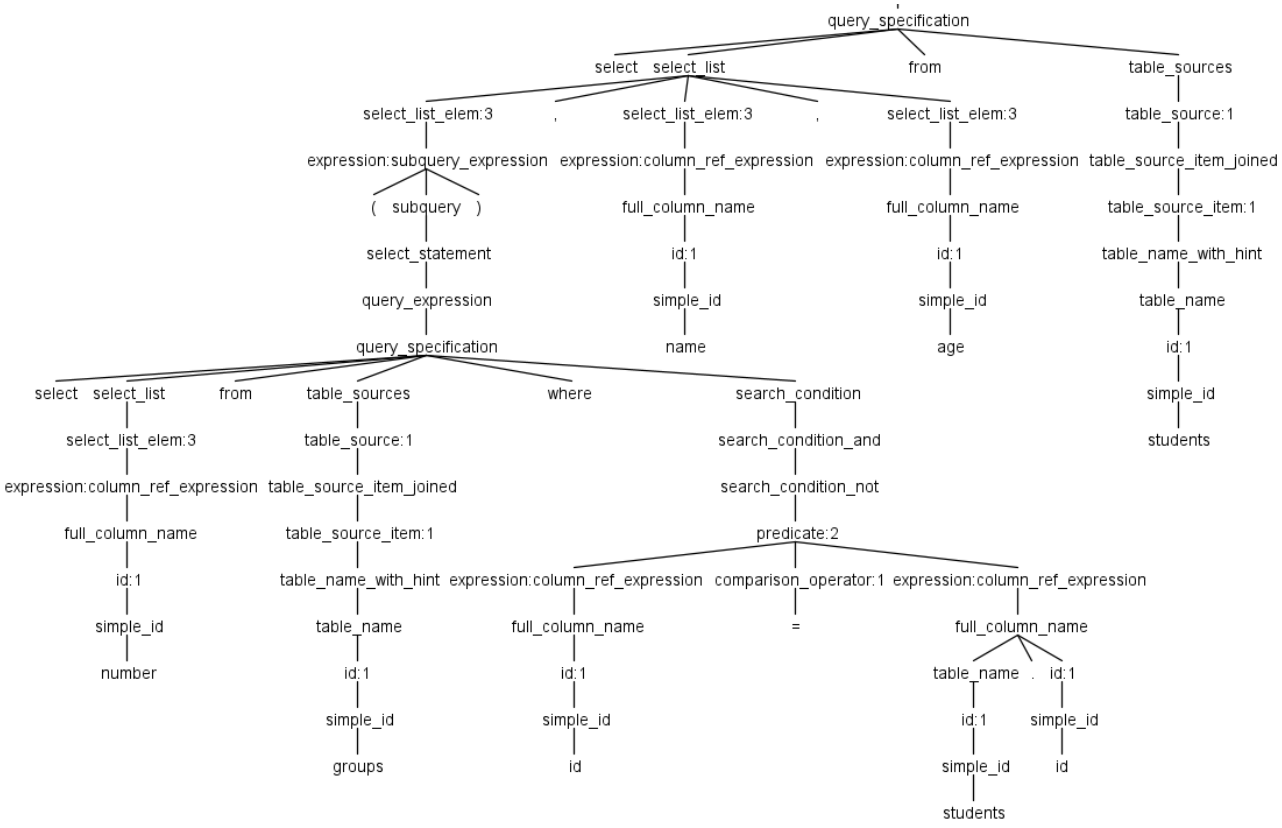

Рисунок 4.1 – Дерево парсингу запиту SQL

4.3 Проблеми з хінтами в запиті

Неефективний хинт може привести до істотного зниження продуктивності. Причини виникнення неефективності хинтов:

– хинт був написаний, коли БД працювала на 9-му Oracle, при переході на Oracle 10g і вище хинт став гальмом (це можуть бути хинти Rule, Leading і ін.). Leading потужний хинт, але при переході на іншу версію Oracle в деяких випадках призводить до різкого зниження продуктивності і перед застосування цих хинтов необхідно враховувати ймовірність зміни з часом статистики системи і її об'єктів (таблиць і індексів), які використовуються в запиті;

– в хинти USE_NL міститься не повний перелік алиасов;

– в складеному хинти використовується неправильний порядок проходження хинтов, в результаті чого хинти блокую ефективну роботу один одного;

– хинт написаний давно, після чого була модифікація запиту (наприклад, відсутня або змінився індекс, вказаний в хинти).

У запиті може отсутствивалось хинт, який би підвищив ефективність роботи запиту. У ряді випадком наявність хинта підвищує ефективність запиту і забезпечує стабілізацію планів виконання (наприклад, при зміні статистики).

При створенні хинтов в запиті є ряд рекомендацій:

–в хинти INDEX можуть бути перераховані кілька індексів. Оптимізатор сам вибере відповідний індекс або поставити хинт NO_INDEX, якщо необхідно заблокувати використання якогось індексу;

– при наявності Distinct в запиті Distinct ставитися після хинта (тобто хинт завжди йде після Select);

– найбільш ефективні і часто використовуваними є хинти: Ordered, Leading, Index, No_Index, Index_FFS, Index_Join, Use_NL, Use_Hash, Use_Merge, First_Rows (n), Parallel, Use_Concat, And_Equal, Hash_Aj і інші. При цьому, наприклад, індекс Index_FFS крім швидкого повного сканування індексу дозволяє йому виконуватися паралельно, в силу чого можна отримати суттєвий виграш в продуктивності [34].

У деяких випадках, коли хинт неефективний, але замінити його оперативно в запиті не представляється можливим (наприклад, чужа розробка), є можливість, не змінюючи робочий запит в програмному модулі, замінити хинт (хинти) в запиті, а також в його підзапитах, на ефективний хинт (хинти). Це прийом – підміна хинтов (який відомий, як використання збережених шаблонів Stored Outlines). Але така підміна повинна бути тимчасовим рішенням до моменту коригування запиту, оскільки постійна підміна хинта може привести до деякого зниження продуктивності запиту.

4.4 Неефективно написаний запит

Причин неефективності запиту може бути кілька:

– неефективне з'єднання таблиць;

– використання NOT і NOT IN в умови where;

– блокування індексу в силу використання неправильних функцій до колонку, за яким побудований індекс;

– велика вкладеність запиту або більша його довжина;

– великий обсяг обираних даних, що вимагають підключення в роботу дисків, в тому числі для виконання агрегованих функцій (order by, group by і т.д.);

– неефективні процедури, що зберігаються, використовуються в запиті і ін.

Серед причин неефективності особливу увагу слід приділити неефективного з'єднанню таблиць (наявність HASH або MERGE з'єднань там, де краще NESTED LOOP – про що сказано вище). Крім того ефективність з'єднання може залежати від порядку таблиць у фразі FROM. Щоб оптимізатор працював з таблицями в тому порядку, в якому вони знаходяться у фразі From використовується хинт Ordered.

Ефективність з'єднання залежить від повноти зв'язку у фразі WHERE між таблицями. При недостатній зв'язці в плані виконання з'являється MERGE JOIN CAPTESIAN (про що було сказано вище).

Особлива увага при модифікації запиту слід приділити фразі NOT IN в умови where. Як варіант звільнення від NOT IN можна використовувати прийом, при якому пишеться перший запит без NOT IN, а за ним після MINUS пишеться той же запит з IN (віднімання з повного числа рядків рядка, одержувані після використання умови IN, який працює швидше, ніж NOT IN).

З метою прискорення роботи запиту використовувати (там, де це можна) замість UNION фразу UNION ALL (UNION операція більш повільна, тому що здійснюється шляхом сортування).

Рекомендується зменшувати число таблиць у фразі FROM. Це дозволить зробити план виконання прозорим для оптимізатора та його аналізу. В першу чергу прибрати з FROM таблиці, стовпці яких не використовуються після фрази Select. В цьому випадку можна використовувати вкладені запити з цими таблицями після Select або у фразі where.

Завдання діапазону дат, починаючи з 01.01.2001, призводить до неефективного плану виконання. Треба зробити мінімальну межу дати, тобто якомога ближче до реальної дати.

З метою підвищення продуктивності запиту не робити довгі запити, тому що довгий запит збільшує час розбору запиту оптимізатором, час передачі по каналах і займає надлишкову пам'ять.

З метою підвищення продуктивності роботи запиту ширше використовувати кешування всіх видів: послідовностей, таблиць, результатів виконання запитів. Кешування результатів виконання запитів з'явилося в Oracle 11g і дозволяє витягувати результат першого виконання запиту з оперативної пам'яті. Це особливо ефективно при великому числі виконання запиту і відсутність в момент багаторазового виконання запиту операцій DML над таблицею [35], [36], [37].

4.5 Аналіз запитів з метою підвищення швидкості їх виконання

Зміна SQL-виразів на основі знань про дані, індексах, зв'язках таблиць для підвищення ефективності їх виконання, називається корекцією запитів (query rewriting). Зміна пропозицій SQL відрізняється від написання нових пропозицій. Для того щоб ефективно переписувати запити, необхідно протягом деякого часу накопичити знання про систему. Сюди відносяться

відомості про те, які пропозиції SQL потребують переписуванні в зв'язку з їх частим використанням або використанням ними значних ресурсів, які дані ними обробляються, які характеристики і розподіл цих даних, які логічні умови в виразах можна прибрати або трансформувати в зв'язку з логікою функціонування системи. При вирішенні завдань оптимізації проблемних запитів необхідно дотримуватися наступних рекомендацій [38].

По-перше, при необхідності доступу до значної частини рядків будь-якої таблиці повне сканування (full scan) є більш ефективним, ніж використання індексів. Кордон застосування даних методів доступу в загальному випадку становить 5–10% записів таблиці, до яких звертається запит. Справа в тому, що для сканування індексу і вилучення рядка потрібні, принаймні, дві операції читання для кожного рядка (одна для читання індексу, інша для читання даних з таблиці). А при повному скануванні таблиці для вилучення рядка потрібно тільки одна операція читання. При доступі до великої кількості рядків стає очевидною неефективність використання індексу в порівнянні з повним скануванням таблиці, при якому рядки зчитуються безпосередньо з таблиці. Для невеликих таблиць повне сканування практично завжди виявляється ефективніше використання індексу.

По-друге, на різних етапах виконання запитів слід максимально використовувати результати попередніх етапів. Наприклад, якщо результуючий набір даних потрібно впорядкувати за значенням деякого стовпчика, то при виконанні операції з'єднання таблиць можна вказати спосіб виконання цієї операції, при якому буде проведено сортування цих значень. Отримані результати будуть використані при остаточній сортуванні [39].

По-третє, при використанні різних видів підзапитів на основі знань про дані слід враховувати особливості обчислення спеціальних предикатів і застосування операторів теоретико-множинних операцій. Наприклад, оператор MINUS може виконуватися набагато швидше, ніж запити з WHERE NOT IN (SELECT) або WHERE NOT EXISTS.

Крім таких, досить очевидних, способів поліпшення якості запитів можна використовувати інші. Як правило, на основі досвіду роботи з конкретною базою даних у кожного користувача формується свій стиль написання оптимальних SQL-виразів. При цьому існують об'єктивні засоби визначення якості запиту. В першу чергу до них відноситься трасування.

ВИСНОВКИ

На основі виконаних теоретичних досліджень вирішено актуальну науково-технічну задачу створення моделей та інформаційної технології оптимізації структури БД вузла у КІС.У ході виконання дослідженняв иконано аналіз особливостей задач використання інтегрованого комплексу територіально розосереджених інформаційних систем та методів їх оптимізації.Побудовано модель SQL-запиту та формалізовано критерії оптимальності структури БД віддаленого вузла КІС.

Розроблено інформаційну технологію парсингу та подальшого обліку SQL-запитів до реляційної БД, а також обчислення значень критеріїв оптимальності відповідно до результатів операцій зрізу та консолідації.Розроблено інформаційну технологію обчислення вектору глобальних пріоритетів альтернатив із представленням результату у вигляді вектору нечітких чисел та приведенням до чіткого значення.

Удосконалено модель SQL-запиту, яка, на відміну від інших, представлена моделлю багатовимірної БД в аналітиці множини атрибутів та кортежів відношення, а також інших характеристик запиту, що дозволяє проведення оперативно-аналітичного аналізу зважаючи на множинність вимірів з метою розрахунку рівня представленості атрибутів та кортежів відношення БД на вузлі РКІС.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Дейт К. Дж. SQL и реляционная теория. Как грамотно писать код на SQL. Пер. с англ. СПб.: Символ-Плюс, 2010. 480 с.

2. Дворецький М. Л., Боровльова С. Ю., Дворецька С. В. WEB-застосунок складського обліку в неавтоматизованих торгових точках. Наукові праці: Науково-методичний журнал Чорноморського національного університету ім. П. Могили. *Комп'ютерні технології*. 2018. Вип. 308 Т. 320. С. 45-52.

3. Как оптимизировать запросы в SQL? URL: https://vc.ru/newtechaudit/113408-kak-optimizirovat-zaprosy-v-sql (дата звернення: 01.04.2021).

4. Пономаренко Л. А., Танянский С. С., Филатов В. А. Построение оптимальной последовательности соединений отношений в запросах реляционной базы данных. *Системні дослідження та інформаційні технології*. 2003. №2. С. 53-58

5. Кунгурцев О.Б., Зіноватна С.Л. Порівняння вартості виконання запитів до й після денормалізації реляційної БД. *ВІСНИК ЖДТУ Одеського нац. політехнічного університету: Технічні науки*. 2006. No 4 (39). С. 207-212.

6. Оптимизация в распределенных системах управления базами даннях.URL: http://citforum.ru/database/articles/art_26_10.shtml (дата звернення: 03.04.2021).

7. ЧухраевИ. В., ЖуковаИ. В. Оптимизация работы с информацией в базах данных. *Международный научный журнал «Инновационная наука»*. 2016. №4. С. 206-208.

8. Oracle Database Documentation. URL: https://docs.oracle.com/en/database/oracle/oracle-database/index.html (дата звернення: 03.04.2021).

9. Базы данных. Оптимизация запросов. Оптимизация структуры даннях. URL: https://www.youtube.com/watch?v=9yWZ-LIsAII (дата звернення: 04.04.2021).

10. Кунгурцев А. Б., Зиноватная С. Л. Иерархическая модель объектов для исследования запросов к базе данных. Труды Одесского политехнического университета. 2008. № 2. С. 130-134.

11. Филатов В. А., Танянский С. С. Сравнительная характеристика показателей сложности выполнения запросов в реляционных СУБД. *Системи обробки інформації*. 2004. Вип. 2. С. 91-95.

12. Дворецький М. Л., Кулаковська І. В. Порівняльний ABC-XYZ аналіз на базі різних факторів із використанням ієрархічних даних. *Проблеми інформаційних технологій Херсонського нац. технічного університету*. 2016. № 19. С. 200–209.

13. Markus Winand. SQL performance. DGS Druck u. Graphikservice GmbH Wien Austria, 2014. 197 p.

14. Дворецький М.Л. Підходи щодо підвищення швидкості роботи SQL-запитів при використанні індексів БД. *Ольвійський форум: стратегії країн Причорноморського регіону в геополітичному просторі*: матеріали XII міжнар. наук.-практ. конф., 7-10 червня 2018 р. Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2018. С. 30-32.

15. Дворецький М. Л., Дворецька С. В., Давиденко Є. О. Розробка системи управління знаннями організації на базі CMS WORDPRESS. *Проблеми інформаційних технологій Херсонського національного технічного університету*. 2018. №1 (023). С. 173–180.

16. Гарсиа-Молина Г., Ульман Дж.Д., Уидом Дж. Системы баз данных. Полный курс. Пер. с англ. М.: Изд. дом «Вильямс», 2003. 1088 с.

17. Крёнке Д. Теорія і практика проектування баз даних. 8-е изд. Пер. с англ. СПб: Питер, 2003. 800 с.

18. Дейт К. Дж. Введение в системы баз данных. 8-е изд. Пер. с англ. М.: Изд. дом «Вильямс», 2005. 1328 с.

19. Новиков Б. А., Рогова Е. В. Основы технологий баз данных: учеб. пособие. М.: ДМК Пресс, 2019. 240 с.

20. Денормализация как средство повышения производительности. Реализация денормализации. URL: https://intellect.icu/denormalizatsiya-kak-sredstvo-povysheniya-proizvoditelnosti-realizatsiya-denormalizatsii-7877 (дата звернення: 04.04.2021).

21. Дворецький М. Л. Інтелектуальний аналіз даних в 1С:8.0. *Наукові праці: Науково-методичний журнал Миколаївського державного гуманітарного університету ім. П. Могили.* Сер. Комп'ютерні технології. 2007. Вип. 55. Т. 68. С. 141–149.

22. Кунгурцев А. Б., Возовиков Ю.Н. Поиск закономерностей в распределении запросов для управления материализованными представлениями. *Труды Одесского политехнического университета.* 2008. 2(30). С. 135–140.

23. Методы и модели анализа данных: OLAP и Data Mining / А. А. Барсегян та ін. Петербург, 2004. 336 с.

24. Пасічник В. В., Шаховська Н. Б., Сховища та простори даних: монографія. Львів : Вид-во Львівської політехніки, 2009. 244 с.

25. Пасічник В. В., Шаховська Н. Б., Сховища даних: Навчальний посібник. Львів : «Магнолія, 2006», 2008. 492 с.

26. Малахов Е. В., Востров Г. Н., Мороз В. В. Проблемы создания баз данных и информационных хранилищ. URL: http://www.nbuv.gov.ua/Articles/OSPU/opu_97_2/1_19.htm (дата звернення: 04.04.2021).

27. Малахов Е. В., Иванченко О.В. Организация переноса информации из баз данных в информационные хранилища. *Тр. Одес. политехн. ун-та*. Одесса. 1998. No 2(6). С. 5254.

28. Філатов В. О. Мультіагентні технології інтеграції гетерогенних інформаційних систем і розподілених баз даних : автореф. дис. … д-ра техн. наук : 05.13.06. Харків, 2005. 32 с.

29. Фісун М.Т., Дворецький М.Л., Дворецька С.В. Побудова моделей для оптимізації структури бази даних вузла у корпоративних інформаційних системах. *Інформаційні технології та комп'ютерна інженерія: Міжнародний науково-технічний журнал Вінницького національно-технічного університету.* vol 48, № 2. 2020. С. 52-60.

30. Дворецький М.Л. Інтеграція підсистем обліку та оперативно-аналітичної обробки даних в інформаційній системі супермаркету. *Комп'ютерні науки: освіта, наука, практика.* Миколаїв. Національний університет кораблебудування. 2014. С. 58-60.

31. Малахов Є. В., Блажко О. А., Глава М. Г. Проектування баз даних та їх реалізація засобами стандартного SQL та PostgreSQL: Навчальний посібник для студентів вищих навчальних закладів. Одеса : ВМВ, 2012. 248 с.

32. Моделирование и оптимизация распределенных информационных систем / Ю. О. Скобцов та ін. Донецк : Изд-во «Нуолидж», 2012. 300 с.

33. Savchuk T., Kozachuk A. Development of cloud application efficiency evaluation criterion. *EasternEuropean Journal of Enterprise Technologies*. 2015. № 2 (77). P. 20–26. ISSN 1729-3774

34. Филатов В. А., Семенец Р. В. Методы и средства проектирования информационных систем и распределенных баз данных. *Вестник Херсонского национального технического университета*. 2007. No 4(27). С. 203–207.

35. Иванов А.Ю. Мобильные распределенные базы данных втоматизированных информационно-управляющих систем МЧС России: монография. СПб.: Санкт-Петербургский ун-т ГПС МЧС России. 2008. 152 с.

36. Пасічник В. В., Резніченко В. А. Організація баз даних та знань. К.: Видавнича група BHV, 2006. 384 с.

37. Ковалюк О. А. Интеграция программных компонентов распределенной информационной системы. *Информационные технологии и компьютерная техника*. 2011. No 4. С. 95–96.

38. Касаткіна Н. В., Пономаренко Л. А., Філатов В. О. Роль і місце інтегрованих баз даних у розподіленій інформаційній системі ВАК України. *Проблеми системного підходу в економіці: Збірник наук. праць*. 2009. №29. С. 3–7.

39. Фісун М. Т., Дворецький М. Л. Синхронізаця оновлення даних в гетерогенних інформаційних системах. *Наукові праці: Науково-методичний журнал Миколаївського державного гуманітарного університету ім. П. Могили*. Сер. Комп'ютерні технології. 2006. Вип. 44. Т. 57. С. 61–66.