НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра автоматизованих систем обробки інформації і управління

«На правах рукопису»

«До захисту допущено»

В.о. завідувача кафедри

УДК _____

_____ *О.А.Павлов*

(підпис) (ініціали, прізвище)

Магістерська дисертація

зі спеціальності *121 «Інженерія програмного забезпечення»*

на тему: « Математичне та програмне забезпечення методів обробки даних в розподілених базах на платформі .NET »

Виконала: студентка VI курсу, групи *ІП-з81мп*

Шушакова Яна Анатоліївна _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник *ст. вик. Олійник Юрій Олександрович* _____
(посада, науковий ступінь, , вчене звання, прізвище та ініціали) (підпис)

Консультант *доц., к.т.н., Ліщук К.І.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент *доц. каф. АУТС, к.т.н., доц. Букасов М.М.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2019 року

РЕФЕРАТ

Магістерська дисертація: 68 с., 22 рис, 35 таб., 2 додатки, 16 джерел.

Актуальність теми: на сьогодні на платформі .NET не існує повного рішення для роботи з розподіленими базами даних. Проте розробники потребують таку можливість, як робота з розподіленими базами, існуючі бібліотеки лише надають можливості аналізу BigData.

Мета дослідження: покращення можливостей програмного забезпечення для роботи з розподіленими базами даних на платформі .NET

Для реалізації поставленої мети були сформульовані **наступні завдання**:

- створення програмної архітектури розподіленої бази даних;
- розробка методів додавання, простої вибірки та вибірки зі з'єднанням даних;
- реалізація методів додавання, простої вибірки та вибірки зі з'єднанням даних;
- дослідження ефективності розроблених методів.

Об'єкт дослідження: розподілені бази даних.

Предмет дослідження: реалізація доступу до даних в розподілених базах на платформі .NET.

Методи дослідження: методи join data в розподілених базах даних.

Наукова новизна: найбільш суттєвими науковими результатами магістерської дисертації є:

- розробка адаптованого методу поєднання даних в розподілених базах.

Практичне значення отриманих результатів визначається тим, що запропонований модифікований алгоритм ефективно виконує операцію поєднання розподілених даних.

Зв'язок роботи з науковими програмами, планами, темами: робота виконувалась на кафедрі автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський

політехнічний інститут ім. Ігоря Сікорського» в рамках теми «Методи та технології високопродуктивних обчислень та обробки надвеликих масивів даних». Державний реєстраційний номер 0117U000924.

Апробація: Основні положення роботи доповідались і обговорювались на III всеукраїнській науково-практичній конференції молодих вчених та студентів «Інформаційні системи та технології управління» (ІСТУ-2019)

Публікації: Наукові положення дисертації опубліковані в Шушакова Я.А. Огляд підходів виконання операцій в розподілених базах даних/ Шушакова Я.А., Ю.О. Олійник // Матеріали III всеукраїнської науково-практичної конференції молодих вчених та студентів «Інформаційні системи та технології управління» (ІСТУ-2019) – м. Київ: НТУУ «КПІ ім. Ігоря Сікорського», 20-22 листопада 2019 р.

Ключові слова: РОЗПОДІЛЕНІ БАЗИ ДАНИХ, ОБРОБКА ЗАПИТІВ, ОБ’ЄДНАННЯ ДАНИХ, ОПТИМІЗАЦІЯ.

ABSTRACT

Master's dissertation consists 68 pages, 22 images, 35 tables, 16 referring sources.

Topicality: Today, there is no general solution for dealing with distributed databases in the .NET platform. However, developers need the ability to work with distributed databases; existing libraries only provide BigData analysis capabilities.

The aim of the study: To enhance software for distributed databases on the .NET platform.

To achieve this goal, the **following tasks** were formulated:

- creation of software architecture of distributed database.
- develop methods for adding, selecting and joining data.
- implementation of methods of adding, selecting and joining data.
- investigation of the effectiveness of the developed methods.

Object of study: distributed databases.

Subject of research: implementation of access to data in distributed databases on the .NET platform.

Research methods: join data methods in distributed databases.

Scientific novelty: The most significant scientific results of the master's thesis are:

- development of an adapted method for joining data in distributed databases.

The practical value of the obtained results is determined by the fact that the proposed modified algorithm effectively performs the operation of joining distributed data.

Relationship with working with scientific programs, plans, topics: work was performed at the Department of Automated Information Processing and Management Systems of the National Technical University of Ukraine «Igor Sikorsky Kyiv Polytechnic Institute» within the topic «Methods and technologies of high-performance computing and processing of large data sets». State Registration Number 0117U000924.

Testing: The main points of the work were reported and discussed at the Third All-Ukrainian Scientific and Practical Conference of Young Scientists and Students "Information Systems and Management Technologies".

Publications: Scientific provisions of the dissertation published in Y.A. Shushakova Overview of join operations in distributed systems / Y.A. Shushakova, Y.O. Oliynyk // Proceedings of the Third All-Ukrainian Scientific and Practical Conference of Young Scientists and Students "Information Systems and Management Technologies" (ISTU2019) - Kyiv: NTUU “ Igor Sikorsky Kyiv Polytechnic Institute”, November 20-22, 2019.

Keywords: DISTRIBUTED DATA BASE, QUERY PROCESSING, JOIN DATA, OPTIMIZATION.

ЗМІСТ

ВСТУП

Кожного дня у світі публікується та генерується величезна кількість інформації. Людина вже не в змозі оброблювати таку кількість інформацію. Тому сьогодні однією з найактуальніших тем на ринку є програмні засоби, що можуть оброблювати великі масиви даних, що постійно оновлюється.

У зв’язку з ростом великої кількості даних постає питання розміщення даних. Наразі, найпопулярнішим рішенням є розподілене зберігіння даних, що вимагає нових підходів для ефективної обробки даних в розподілених базах даних.

Однією з найпопулярніших мов програмування є C# і платформа .NET. Проте на сьогодні на платформі .NET не існує повного рішення для роботи з розподіленими базами даних. Розробники потребують таку можливість, як робота з розподіленими базами, існуючі бібліотеки лише надають можливості аналізу BigData.

Тому головною **метою дослідження є** *покращення можливостей програмного забезпечення для роботи з розподіленими базами даних на платформі .NET.*

До **об’єкту дослідження** відносяться *розподілені бази даних*, а до **предмету дослідження** відноситься *реалізація доступу до даних в розподілених базах на платформі .NET.*

Наукова новизною є адаптований метод поєднання даних в розподілених базах.

1 ТЕОРЕТИЧНІ ОСНОВИ

1.1.Розподілена база даних

Розподілені бази даних (РБД) - сукупність логічно взаємопов'язаних баз даних, розподілених у комп'ютерній мережі.

РБД складається з набору вузлів, пов'язаних комунікаційною мережею, в якій:

а) кожен вузол - це повноцінна СКБД сама по собі;

б) вузли взаємодіють між собою таким чином, що користувач будь-якого з них може отримати доступ до будь-яких даних в мережі так, як ніби вони знаходяться на його власному вузлі.

Кожен вузол сам по собі є системою бази даних. Будь-який користувач може виконати операції над даними на своєму локальному вузлі точно так само, як якщо б цей вузол зовсім не входив в розподілену систему. Розподілену систему баз даних можна розглядати як партнерство між окремими локальними СКБД на окремих локальних вузлах.

Фундаментальний принцип створення розподілених баз даних («правило 0»): Для користувача розподілена система повинна виглядати так само, як нерозподілена система.

Формалізація концептуальної схеми даних спричинила за собою можливість класифікації моделей представлення даних на ієрархічні, мережеві і реляційні. Це відбилося в понятті архітектури систем керування базами даних (СКБД) і технології обробки. Для обробки даних, розміщених на віддалених комп'ютерах, розроблені мережеві СКБД, а сама база даних називається розподіленою[1].

Розподілена обробка і розподілена база даних не є синонімами. Якщо при розподіленій обробці проводиться робота з базою, то мається на увазі, що подання даних, змістовна обробка даних бази виконуються на комп'ютері клієнта, а підтримка бази в актуальному стані на файл-сервері. Розподілена база даних може розміщуватися на декількох серверах і для доступу до віддалених даних

треба використовувати мережеву СКБД. Якщо мережева СКБД не використовується, то реалізується розподілена обробка даних.

При розподіленій обробці клієнт може надіслати запит до власної локальної базі або віддаленої. Віддалений запит ~ це одноразовий запит до одного сервера. Кілька віддалених запитів до одного сервера об'єднуються в віддалену транзакцію. Якщо окремі запити транзакції обробляються різними серверами, то транзакція називається розподіленою. При цьому запит визначення транзакції обробляється одним сервером[1]. Якщо запит транзакції обробляється декількома серверами, він називається розподіленим.

Тільки обробка розподіленого запиту підтримує концепцію розподіленої бази даних.

Розподілена база даних складається з декількох, можливо, що перетинаються або навіть дублюючих один одного частин, які зберігаються в різних ЕОМ обчислювальної мережі. Однак користувач розподіленої бази даних не зобов'язаний знати, яким чином її компоненти розміщені у вузлах мережі, і уявляє собі цю базу даних як єдине ціле (дану властивість називають прозорістю). Робота з такою базою даних здійснюється за допомогою системи управління розподіленою базою даних (СКРБД). Частини розподіленої бази даних, розміщені на окремих ЕОМ мережі, можуть управлятися власними (локальними) СКБД. Локальні СКБД не обов'язково повинні бути однаковими в різних вузлах мережі. Об'єднання неоднорідних локальних баз даних в єдину розподілену базу даних є складною науково-технічною проблемою. Її рішення вимагало проведення великого комплексу наукових досліджень і експериментальних розробок.

Програмне забезпечення систем керування розподіленими базами даних (СКРБД) зазвичай має багаторівневу архітектуру (рисунок 1.1).

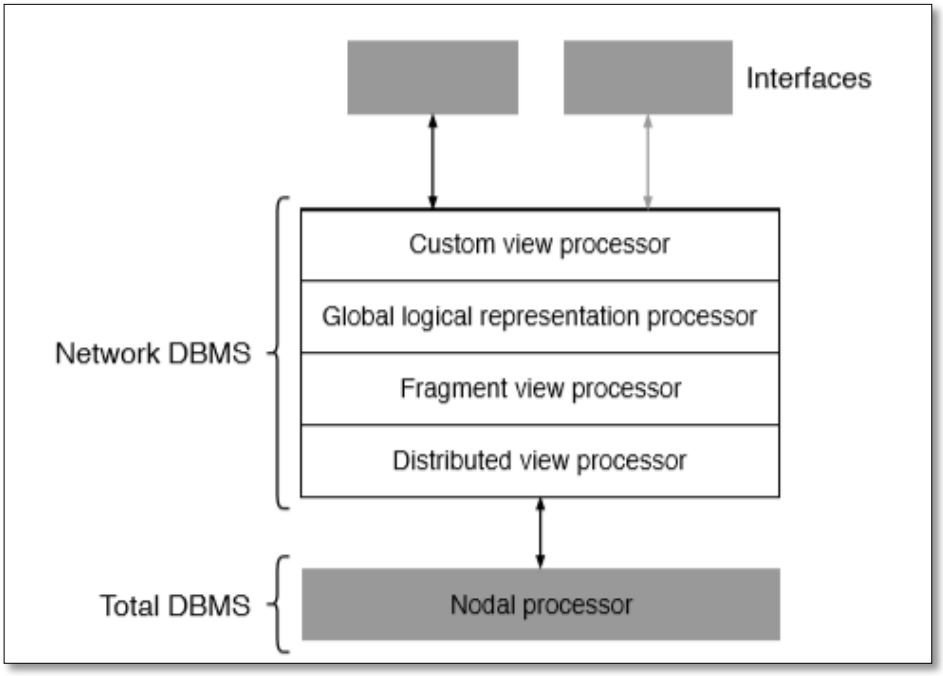

Рисунок 1.1 - Рівні розподіленої БД

Слід розрізняти роботу з розподіленою БД і роботу з віддаленими БД. У другому випадку користувач явно з'єднується з джерелом даних.

1.1.1. Основні принципи розподіленої обробки.

Локальна автономія. Даний принцип означає, що операції на даному вузлі керуються ним же, не потрібно очікування від інших вузлів, хоча в реальних системах автономія неповна, тому що є багато ситуацій, коли потрібно злагоджена робота вузлів.

Незалежність від центрального вузла. Принцип означає, що всі вузли виступають як рівні, інакше при пошкодженні центру може вийти з ладу вся система.

Безперервне функціонування. Принцип означає, що системи повинні бути високо надійні і дані доступні. Надійність - це ймовірність того, що система справна і працює в будь-який заданий момент часу. Системи можуть підтримувати весь спектр методів підвищення надійності (дзеркальні диски, резервні сервери, багатомашинні кластери і т.д.)

Незалежність від розташування. Принцип означає прозорість розташування даних.

Незалежність від фрагментації. Таблиця розбивається на групи, які зберігаються на різних дискових розділах (дисках). Фрагментація бажана для підвищення продуктивності системи, тому що частини таблиць читаються одночасно. Підвищується доступність таблиці, навіть якщо її частини пошкоджені, знижується конкурентність операцій. Дані краще зберігати там, де їх частіше використовують. Існують два основних види фрагментації: вертикальна і горизонтальна (фактично це операції проекції і вибірки) [2].

Незалежність від реплікації. Реплікація корисна з двох причин: для підвищення продуктивності і збільшення доступності. Асинхронна асиметрична - це реплікація, коли один вузол - власник основний майстер-копії таблиці з можливістю внесення змін до неї і автоматичної підтримкою необмеженого числа копій в інших вузлах, з доступом тільки читання. Асинхронна симетрична - це реплікація даних, коли дані доступні для зміни в будь-якому вузлі і автоматично поширюються на всі копії.

Обробка розподілених запитів. У розподілених системах є складний оптимізатор, тому що надзвичайно важливо знайти найбільш ефективну стратегію виконання запиту. Можливі використання паралельних операцій. Оптимальне пересилання даних між вузлами. Підключення індексів.

Управління розподіленими транзакціями. Базовою конструкцією, яка дозволяє фіксувати «правильний» стан БД, є транзакція. Під транзакцією розуміють таку логічну одиницю роботи з БД, яка не приводить її в несуперечливий стан. Вона може включати сотні операцій (наприклад, UPDATE-запитів) і в момент роботи БД може перебувати в суперечливому стані. Але по завершенню БД повинна бути в узгодженому стані. Іншими словами транзакція – це виконання в якості атомарної (неподільної) дії однієї і більше операцій над БД, що не приводить до порушення цілісності БД.

Двофазна фіксація транзакції - коли транзакція виконується під управлінням одного сервера-координатора. Перша фаза: координатор, отримавши інструкцію COMMIT, розсилає іншим серверам повідомлення про підготовку до

фіксації. Сервери відповідають про можливість фіксації. Друга фаза: координатор, отримавши всі докази, приймає рішення про фіксацію або про скасування транзакції. Несуперечливість даних може підтримуватися за допомогою тригерів і збережених процедур. Тригери - спосіб автоматизації дій над БД. Тригер - це спеціальна процедура, яка спрацьовує при певній умові, операції і т.п. Процедура зберігається в відкомпільованому і оптимізованому вигляді на сервері, в результаті частина обчислень переноситься з машин-клієнтів на сервер, скорочується мережевий трафік[2].

Розподілений словник даних. Словник складається не тільки із звичайної для нього інформації, але і зберігає всю інформацію про розміщення, фрагментації, реплікації. Сам словник може перебувати в одному вузлі, бути повністю або частково реплікованим.

Багатомасштабна і многоплатформна. Даний принцип означає незалежність від апаратури, незалежність від операційної системи, незалежність від мережі.

Незалежність від СКБД. Принцип означає відхід від однорідності. Всі СКБД повинні підтримувати один і той же інтерфейс. У неоднорідних системах потрібні спеціальні програми - шлюзи для організації прозорого обміну між різними СКБД.

1.1.2. Однорідні і неоднорідні бази даних

Розподілені системи часто будуються шляхом «інтеграції» різнорідних апаратних і програмних засобів. Отже, повинен бути зроблений вибір між однорідною і неоднорідною обчислювальної системою. У разі однорідних СКБД немає проблем ні з моделями даних, ні з мовами запитів, ні з іншими засобами. Все це збігається з тим, що підтримується кілька СКБД. Для неоднорідних СКБД питання ускладнюються. Використання неоднорідних СКБД зазвичай є наслідком формування розподіленої БД з ряду дотеперішніх автономних баз даних. Завдання, поставлене перед розробниками, мета - досягти прозорого доступу, що є чимось більшим, ніж просте забезпечення доступу до віддалених СКБД і їх баз даних.

Однорідні розподілені системи баз даних мають в своїй основі один продукт СКБД, зазвичай з єдиною мовою баз даних (наприклад, SQL з розширеннями для управління розподіленими даними). СКБД з підтримкою рівномірного розподілу є сильно пов'язаними системами, їх вбудовані засоби пошуку даних і засоби обробки запитів оптимізовані і налаштовані для досягнення максимальної продуктивності і пропускної здатності. На рисунку 1.2 зображена структура типового однорідного середовища розподіленої бази даних.

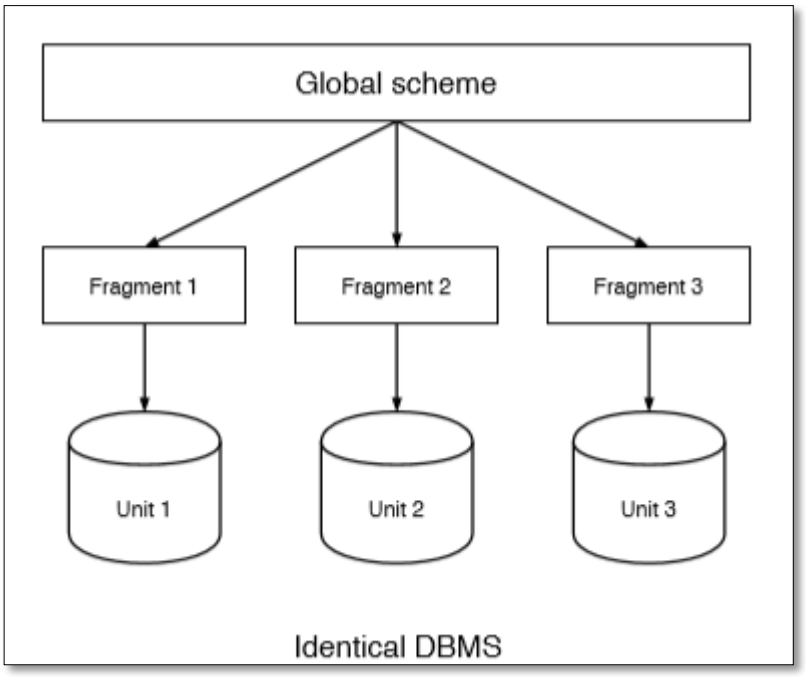

Рисунок 1.2 - Архітектура однорідної розподіленої бази даних

Існує безліч варіантів однорідної системи РБД. Так, на деякому вузлі може існувати одна глобально доступна "головна машина СКБД", з якою пов'язані компоненти для доступу до даних локальних баз даних, розміщення спільно з цими базами даних в межах всієї компанії (або окремого її підрозділу в залежності від масштабу розподілу). Більш складні моделі можуть допускати розподіленість самої СКБД, коли кожен її компонент на рівних правах має доступ до даних будь-якого іншого вузла. Однак щодо власне управління даними ми маємо тут ідентичні моделі зберігання, структури індексування і формати даних в рамках всього розподіленого середовища.

1.1.3. Методи побудови розподілених баз даних "зверху вниз" і "знизу вгору"

Розглянемо спочатку процес побудови розподілених баз даних методом "зверху вниз", оскільки він концептуально найбільш простий для розуміння. Проектування "зверху вниз" здійснюється в цілому аналогічно проектуванню централізованих баз даних[3]. В ідеалі воно проводиться за допомогою однієї з формальних методологій, які включають створення концептуальної моделі бази даних, відображення її в логічну модель даних і, нарешті, створення (і налаштування) специфічних для конкретної СКБД структур (наприклад, таблиць бази даних системи).

Однак при проектуванні методом "зверху вниз" передбачається, що її об'єкти не будуть зосереджені в одному місці, а розподіляться по кільком обчислювальним системам (рисунок 1.3). Розподіл проводиться шляхом фрагментації або тиражування.

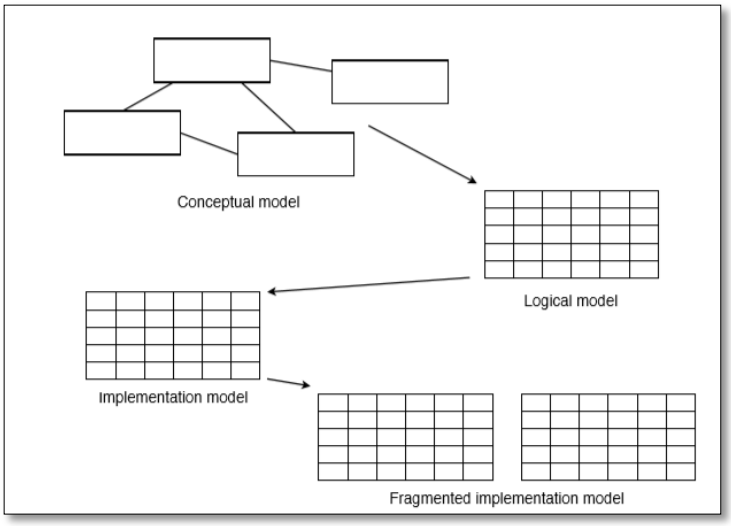

Рисунок 1.3 - Побудова розподіленої бази даних методом «зверху вниз»

Фрагментація означає декомпозицію об'єктів бази даних, таких, як реляційні таблиці, на дві або більше частин, які розміщуються на різних комп'ютерних системах. Класичний приклад, який зазвичай використовують для ілюстрації цього поняття, - таблиця з даними про співробітників або про замовлення на продаж, розділена на фрагменти за географічною чи іншою характеристическому ознакою.

На рисунку 1.4 показана горизонтальна фрагментація, коли в таблиці робляться горизонтальні "зрізи" відповідно до значення, скажімо, будь-якого стовпця таблиці. Рядки даних про співробітників можуть розбиватися на підмножини, відповідні філіям. Дані про продажі фрагментируются по магазинах, де ці продажі проводилися.

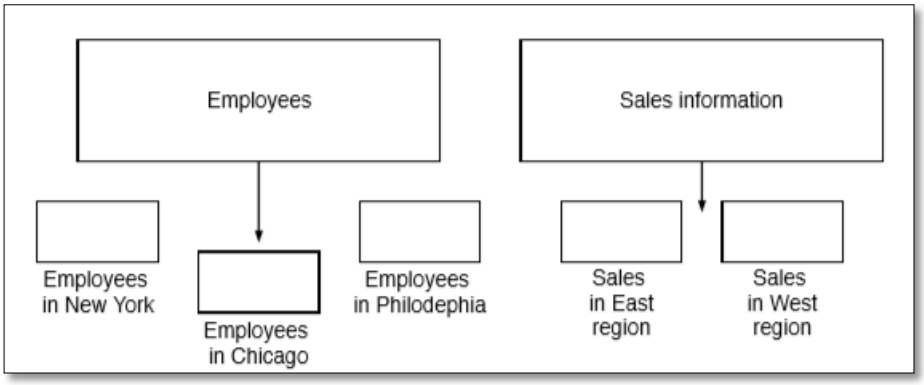

Рисунок 1.4 – Горизонтальна фрагментація

Альтернативна модель фрагментації - вертикальна - означає розбиття таблиці не по рядках, а по стовпцях (рисунок 1.5). В цьому випадку деяка частина інформації про кожного співробітника зберігається в одному місці, а інша частина (що відноситься до тієї ж таблиці) - в іншому.

Незалежно від того, застосовується горизонтальна або вертикальна фрагментація, підтримується глобальна схема, що дозволяє відтворити з наявних фрагментів логічно централізовану таблицю або іншу структуру бази даних[3].

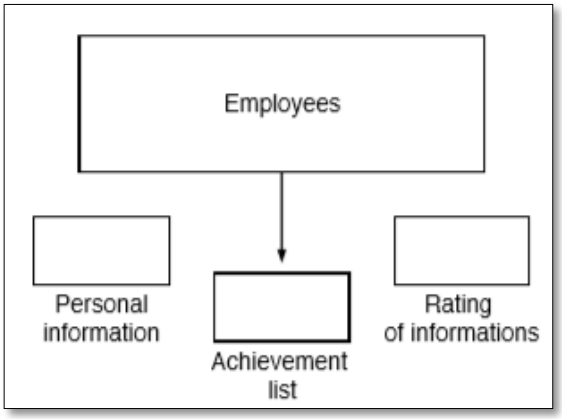

Рисунок 1.5 – Вертикальна фрагментація

Тиражування (або реплікація) означає створення дублікатів даних.

Реплікати – це безліч різних фізичних копій деякого об'єкта бази даних (зазвичай таблиці), для яких відповідно до визначених в базі даних правилами підтримується синхронізація (ідентичність) з деякою "головною" копією.

Теоретично значення всіх даних в тиражованих об'єктах повинні автоматично та негайно синхронізуватися між собою (на практиці це правило зазвичай дещо послаблюється).

У деяких системах копії використовуються виключно в режимі читання і оновлюються відповідно до заданого розкладу. В інших середовищах допускається модифікація окремих значень в копіях, і ці зміни поширюються відповідно до процедур планування та координації. На рисунку 1.6 показані різні моделі тиражування.

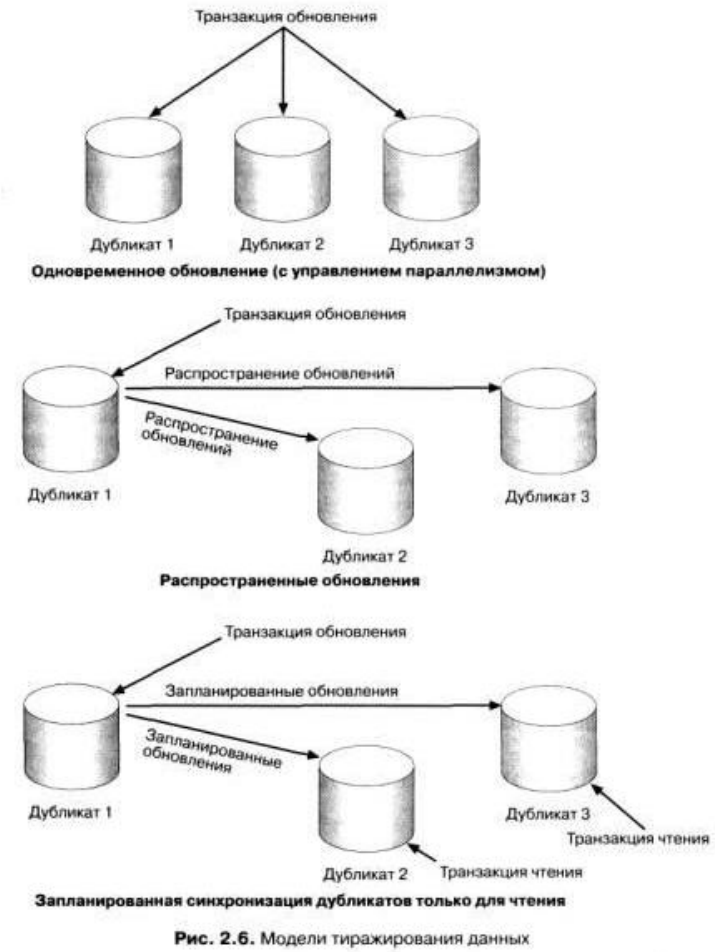

Рис. 2.6. Модели тиражирования данных

Рисунок 1.6 – Моделі тиражованих даних

Ідеологія побудови розподілених баз даних за принципом "зверху вниз" можна застосувати тільки до однорідних, для яких спочатку визначається глобальна схема, а потім проводиться розподіл об'єктів бази даних. Такий підхід виправданий при створенні нових додатків, але набагато ймовірніше, що доведеться вирішувати задачу створення інтегрованого середовища шляхом об'єднання існуючих баз даних і відповідних інформаційних менеджерів, можливо, на додаток до деяких знову проектованим компонентів баз даних. В цьому випадку розробники не можуть дозволити собі "розкіш" проектування "зверху вниз". Тут доводиться вдаватися до проектування "знизу вгору", де основною проблемою стає об'єднання схем вже існуючих баз даних, щоб надати як новим, так і колишнім програмам доступ і до нових, і до старих ресурсів даних (рисунок 1.7) [3].

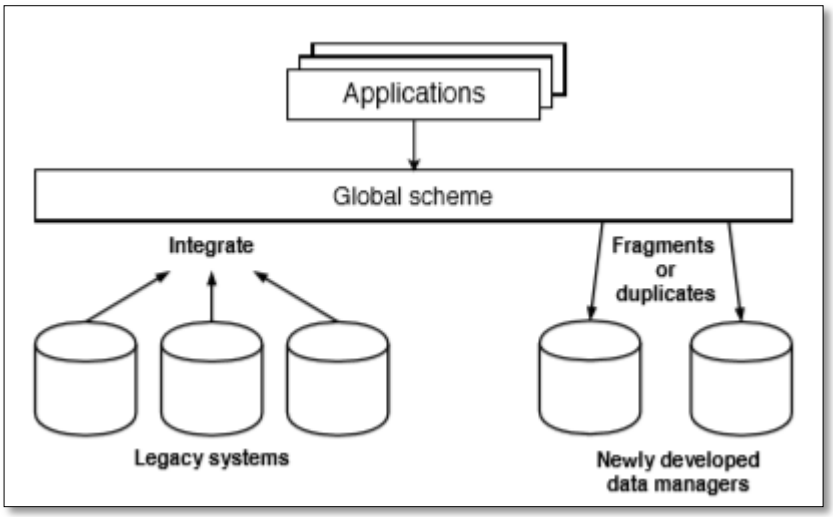

Рисунок 1.7 – Інтеграція розподіленої бази даних методом «знизу вгору»

Серед багатьох складних технічних проблем, які доводиться при цьому вирішувати, відзначимо наступні:

- взаємне відображення різних моделей даних (тобто наявність деякого способу глобального доступу до множинних форм представлення даних: до плоских файлів, до ієрархічних, реляційних, об'єктно-орієнтованих баз даних);
- управління метаданими;

- дозвіл невідповідностей типів даних в різних БД (елемент MOVIE_TYPE - тип фільму - в одній базі даних має числове уявлення, а в іншій - символьне).

1.1.4. Стратегії розподілу даних

Стратегії розподілу даних по вузлах мережі ЕОМ можуть класифікуватися в залежності від кількості вузлів, що містять дані, і наявності дублювання інформації. Допустимі стратегії визначаються архітектурою системи і програмним забезпеченням системи управління бази даних.

Особливості реалізації стратегій розподілу даних визначаються зазвичай в процесі проектування бази даних. За способом реплікації стратегії розподілу даних бувають:

– централізація;

– розчленування;

– дублювання;

– змішана.

Централізація баз даних. Централізований, або метод отримання даних вручну (рисунок 1.8), є найпростішим для реалізації способом. На одному сервері знаходиться єдина копія бази даних. Всі операції з базою даних забезпечуються цим сервером. Доступ до даних виконується за допомогою віддаленого запиту або видаленої транзакції[4].

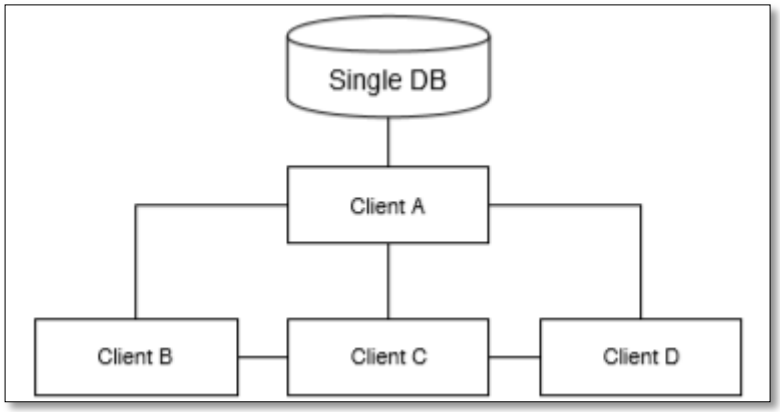

Рисунок 1.8 – Централізоване розподілення

Перевагою такого способу є легка підтримка бази даних в актуальному стані.

Недоліком є те, що розмір бази може перевищувати обсяг зовнішньої пам'яті, всі запити направляються до єдиного серверу за відповідними витратами на вартість зв'язку і тимчасову затримку. Звідси - обмеження на паралельну обробку. База може бути недоступною для віддалених користувачів при появі помилок зв'язку і повністю виходить з ладу при відмові центрального сервера.

Розчленування розподіленої бази даних. При цій стратегії існує єдина копія бази даних, а локальні бази даних розподілені по окремих вузлах. Обсяг розподіленої бази даних обмежується необхідним обсягом вторинної пам'яті, наявної вже в усій інформаційно-обчислювальної мережі. Ефективність стратегії розчленування тим вище, чим вище ступінь локалізації посилань, тобто чим більше число запитів користувачів реалізується в базах даних відповідних локальних інформаційних систем (рисунок 1.9).

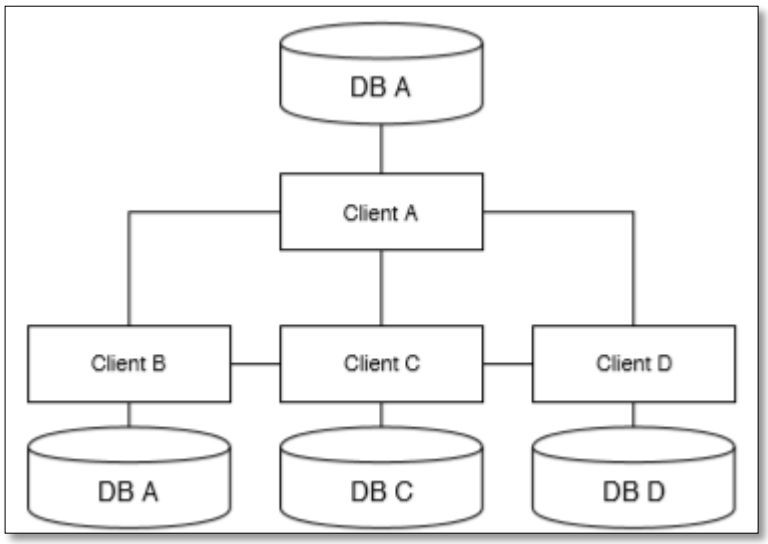

Рисунок 1.9 – Метод розчленування

Перевагами даного методу є збільшення обсягу бази даних; більшість запитів задовольняється локальними базами, що скорочує час відповіді; збільшення доступності і надійності; зниження вартості запитів на вибірку та оновлення в порівнянні з централізованим розподілом; система залишиться частково працездатною, якщо вийде з ладу один сервер.

До недоліків методу відноситься те, що частина вилучених запитів або транзакцій можуть зажадати доступ до всіх серверів, що збільшує час очікування і ціну; необхідно мати відомості про розміщення даних в БД. Однак доступність і надійність збільшуються. Розчленовані бази даних найбільш підходять до випадку спільного використання локальних і глобальних мереж ЕОМ[4].

Дублювання розподілених баз даних. При використанні методу дублювання (рисунок 1.10) в кожному сервері мережі ЕОМ розміщується повна база даних. При цій стратегії організовується кілька копій бази даних; повна копія всіх даних розташовується в кожному логічному вузлі.

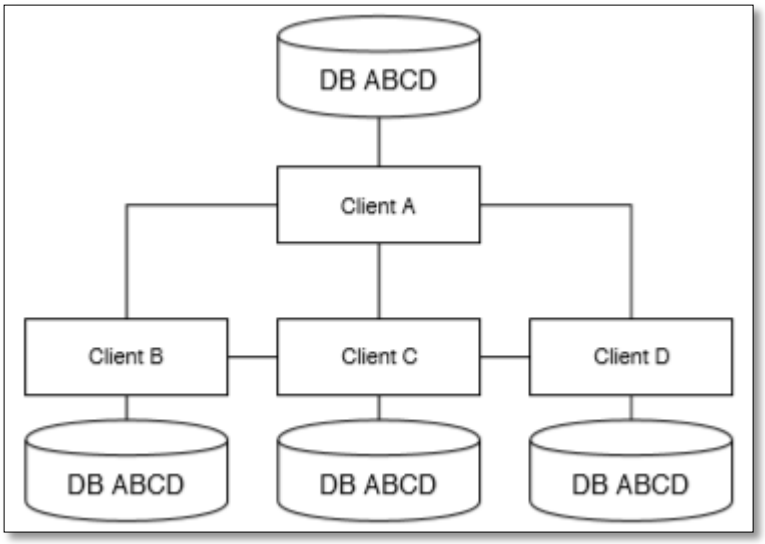

Рисунок 1.10 – Метод дублювання

Основна перевага даної стратегії полягають у високій надійності інформаційної бази до простоти її відновлення. Цей метод дає найбільш надійний спосіб зберігання даних.

Недоліками методу є підвищені вимоги до обсягу зовнішньої пам'яті; ускладнення коригування баз, так як потрібна синхронізація з метою узгодження копій. До переваг методу відноситься те, що всі запити виконуються локально, завдяки чому забезпечується швидкий доступ. Цей метод використовується, коли фактор надійності є критичним, база - невелика, а інтенсивність відновлення невелика[4].

Змішана стратегія. У методі змішаного розподілу об'єднані два способи розподілу даних: дублювання і розчленування (рисунок 1.11). При цьому придбані як переваги, так і недоліки обох способів. З'явилася необхідність зберігати інформацію про те, де знаходяться дані в мережі. Головна перевага цього методу - гнучкість цієї системи, так як можна встановити компроміс між обсягом пам'яті під базу в цілому і під базу в кожному сервері, щоб забезпечити надійність і ефективність роботи. У цій стратегії легко реалізується паралельна обробка, а саме обслуговування розподіленого запиту або транзакції. Недоліки методу: залишається проблема взаємозалежності факторів, що впливають на продуктивність системи, її надійність, підвищуються вимоги до пам'яті. Змішану стратегію використовують при наявності мережевої СКБД, яка забезпечує реалізацію розподіленої бази даних.

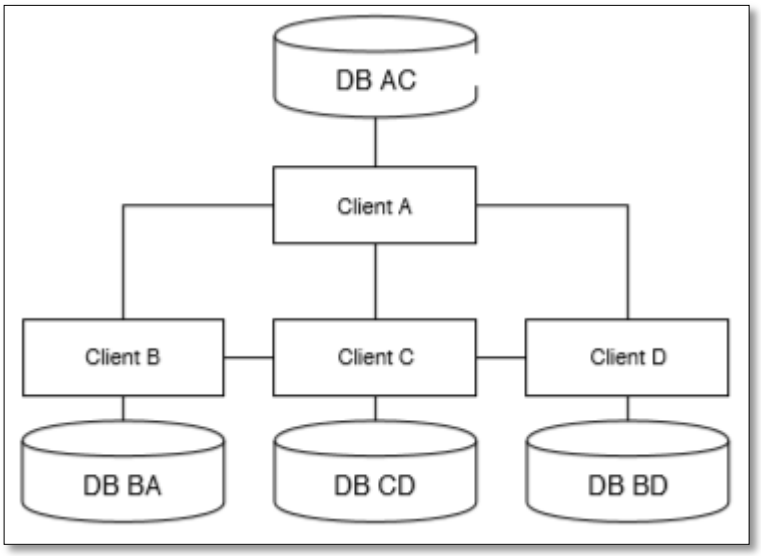

Рисунок 1.11 – Змішане розподілення

За однорідностю використання апаратних і програмних засобів системи підрозділяються на однорідні (гомогенні) і різнорідні (гетерогенні). Якщо ж розподілена база даних підтримується неоднорідними СКБД, то питання ускладнюються. Використання неоднорідних СКБД зазвичай є наслідком формування розподіленої бази даних з ряду дотеперішніх автономних баз даних. Завдання, поставлене перед розробниками, мета - досягти прозорості доступу, що

є чимось більшим, ніж просте забезпечення доступу до віддалених СКБД і їх базами даних.

У розподілених системах баз даних логічно цілісна база даних може бути фрагментована і широко розподілена по мережі з метою поліпшення продуктивності системи. Фрагментація і розподіл бази даних без уважного централізованого планування часто призводять до безладдя і неузгодженості при використанні бази даних. Поетапне проектування розподіленої бази даних враховує цей важливий факт.

1.1.5. Типи розподілених баз даних

Мультибази даних з глобальної схемою. Система мультибаз даних - це розподілена система, яка служить зовнішнім інтерфейсом для доступу до безлічі локальних СКБД або структурується, як глобальний рівень над локальними СКБД.

Федеративні бази даних. На відміну від мультибаз не мають глобальної схеми, до якої звертаються всі програми. Замість цього підтримується локальна схема імпорту-експорту даних. На кожному вузлі підтримується часткова глобальна схема, що описує інформацію тих віддалених джерел, дані з яких необхідні для функціонування.

Мультибази зі спільною мовою доступу – розподілені середовища управління з технологією «клієнт-сервер» [5].

Інтероперабельні системи – це системи, в яких самі додатки, які виконуються в середовищі тієї чи іншої СКБД, відповідальні за інтерфейси між різними середовищами додатки, незалежно від того, є вони однорідними або неоднорідними. Системи орієнтовані головним чином на обмін даними. Подальший розвиток цих систем є об'єктно-орієнтовані БД.

1.1.6. Методи розподіленої обробки баз даних

У системах обробки розподілених баз даних база даних розподілена по безлічі комп'ютерів. На рисунку 1.12 база даних (або частина її) зберігається на

всіх n комп'ютерах. Як показано на рисунку, комп'ютери 1, 2 і n обробляють і додатки, і базу даних, а комп'ютер 3 обробляє тільки базу даних.

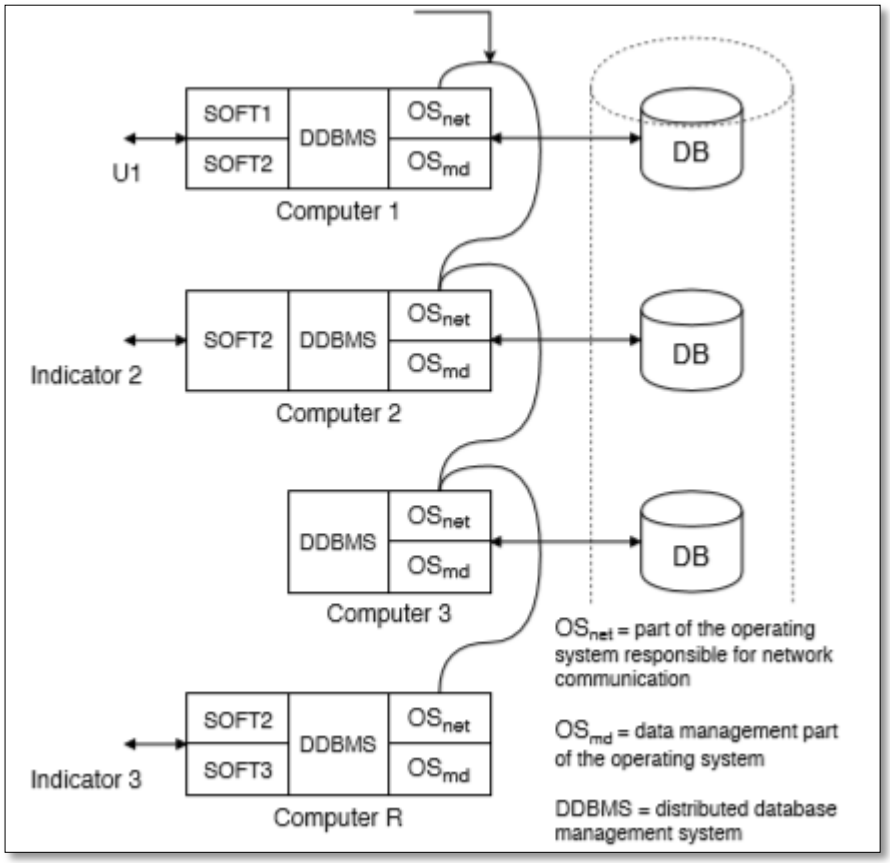

Рисунок 1.12 – Архітектура розподіленої бази даних

Пунктирна лінія, якою обведені файли, означає, що база даних включає в себе всі сегменти, що зберігаються на всіх n комп'ютерах. Ці комп'ютери можуть фізично розміщуватися в одному приміщенні, а можуть - в різних кінцях планети.

Система зі спільним використанням файлів, клієнт-серверна система система обробки розподілених баз даних мють одну важливу відмінність від системи віддаленої обробки даних: у всіх них для обробки додатків або СКБД використовується більш одного комп'ютера[5].

Слід звернути увагу, однак, що сама база даних є розподіленою тільки в архітектурі на рисунку 1.12. Ні в клієнт-серверній системи, ні в системі зі спільним використанням файлів база даних не розподілена по множині комп'ютерів.

Є кілька типів розподілених баз даних. На рисунку 1.13а зображена нерозподілений база даних, що складається з чотирьох частин - W, X, Y і Z. Всі ці сегмента розташовані в одному і тому ж місці, і дублювання даних відсутнє.

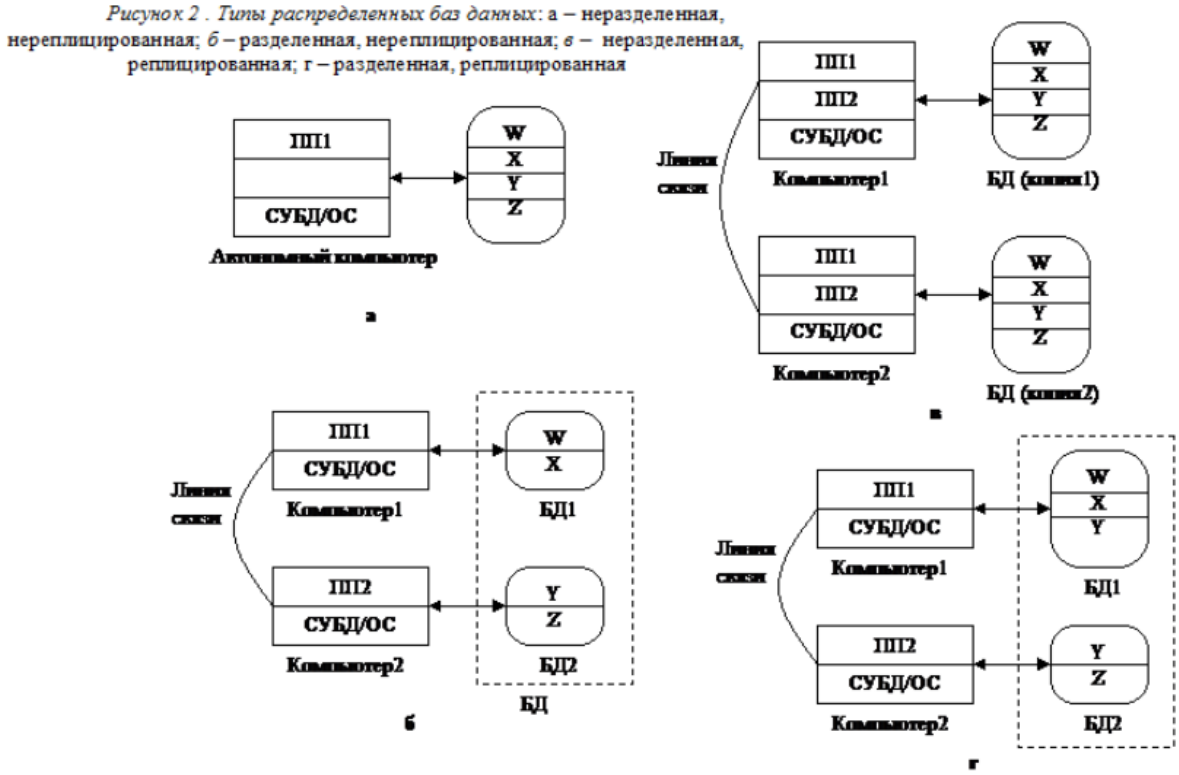

Рисунок 1.13 – Типи розподілених баз даних

На рисунках 1.13б-г зображені розподілені бази даних. На рисунку 1.13б зображений перший тип розподіленої бази даних, де база даних розташована на двох комп'ютерах - сегменти W і X зберігаються на комп'ютері 1, а сегменти Y і Z - на комп'ютері 2. На рисунку 1.13в вся база даних продубльована (реплікована) на двох комп'ютерах. На рисунку 1.13г база даних розділена, а її сегмент Y продубльований.

По відношенню до поділу баз даних іноді вживаються два терміни. Вертикальним розділом або вертикальним фрагментом називається таблиця, розділена на дві або більше сукупності стовпців. Наприклад, таблиця R (C1, C2, C3, C4) може бути розділена на два вертикальних фрагмента: P1 (C1, C2) і P2 (C3, C4). В залежності від програми і від причини, по якій проводиться поділ, ключ відносини R, швидше за все, буде поміщений в відношення P2, яке набуде вигляду P2 (C1, C3, C4).

Горизонтальний розділ або горизонтальний фрагмент - це фрагмент таблиці, що представляє собою сукупність рядків. Наприклад, якщо перші 1000 рядків відносини R поміщаються в відношення R1 (О, С2, С3, С4), а рядки що залишилися поміщаються в відношення R2 (С1, С2, С3, С4), то відносини R1 і R2 утворюють два горизонтальних фрагмента. Іноді база даних розбивається і на горизонтальні, і на вертикальні розділи, і результат такого поділу називається змішаним розділом.

Розділи бази даних зберігаються на двох або більше комп'ютерах, а нерозділена база даних цілком дублюється на двох або більше комп'ютерах. Найбільшим недоліком розподілу є складність управління і обумовлена цим потенційна небезпека втрати цілісності даних. Розглянемо архітектуру бази даних на малюнку 1.13г. Користувач, який сидить за комп'ютером 1, може читати і оновлювати елемент даних розділу Y, розташований на комп'ютері 1, одночасно з тим, як цей же елемент розділу Y, розташований на комп'ютері 2, буде читатися і оновлюватися користувачем, що сидить за комп'ютером 2.

У різних СКБД застосовуються різні способи розподіленої обробки. У Oracle і SQL Server підтримка розподіленої обробки здійснюється подібним чином, але одні й ті ж речі називаються по-різному, і навпаки - різні речі мають однакові назви. У інших виробників також є своя термінологія. Тому зосередимося на основних ідеях.

Найпростішим способом обробки розподіленої бази даних є завантаження даних тільки для читання. В цьому випадку оновленням всіх даних в базі займається тільки один комп'ютер, але копії даних тільки для читання можуть надсилатися на безліч (можливо, тисячі) комп'ютерів. У Oracle такі копії, призначені тільки для читання, називаються матеріалізовані уявленнями. У SQL Serverеті копії називаються миттєвими знімками.

При більш складному способі розподіленої обробки запити на оновлення даних можуть приходити з безлічі комп'ютерів, але на обробку вони передаються одному спеціалізованому комп'ютеру. Наприклад, комп'ютер А може бути

визначений як єдиний комп'ютер, який може оновлювати таблицю СПІВРОБІТНИК (і засновані на ній уявлення), а комп'ютер Б може бути визначений як єдиний комп'ютер, якому дозволено оновлювати таблицю КЛІЄНТ (і її уявлення). Час від часу поновлення повинні передаватися назад на всі комп'ютери в розподіленої мережі, і бази даних повинні синхронізуватися.

Найбільш складний спосіб полягає в тому, щоб дозволити множинне оновлення одних і тих самих даних в різних місцях. В цьому випадку можуть виникнути три види конфліктів розподілених оновлень. По-перше, може бути порушена унікальність. У базі даних галереї View Ridge два різних комп'ютера можуть створити в таблиці ROW рядок з однаковими значеннями стовпців Copy, Title і ArtistlD. Інша можливість нагадує проблему втраченого поновлення: на двох комп'ютерах може оновлюватися одна і той самий рядок. Третій конфлікт виникає в ситуації, коли на одному комп'ютері оновлюється рядок, віддалена на іншому комп'ютері.

Для вирішення конфліктів оновлень виділяється спеціальний комп'ютер. Він стежить за всіма оновленнями, і виникаючі конфлікти вирішуються або власними коштами СКБД, певною програмою, подібно до того, як це робиться в тригерах. У крайніх випадках робиться запис про конфлікт в журналі, і він вирішується вручну. Останній спосіб не рекомендується, оскільки до усунення конфлікту безліч рядків в працюючих базах даних можуть зависнути в невизначеному становищі, що призведе до неприйнятного зниження пропускної здатності інформаційної системи.

Жоден з цих способів не вирішує проблеми забезпечення атомарности транзакцій в розподіленої базі даних. Ця проблема стає особливо важливою, якщо це призводить до конфліктів оновлень. В який момент роботу з базою даних можна вважати виконаною? Якщо під час вирішення конфлікту розподілених оновлень повинен відбутися відкат оновлень, то потенційно можливо, що розподілена транзакція не буде виконана протягом декількох годин або навіть діб. Очевидно, що така затримка не є прийнятною.

Якщо не брати до уваги конфлікти розподілених оновлень, залишається ще одне складне питання - координація розподілених транзакцій. Щоб транзакція була атомарною, жодне оновлення в розподіленої транзакції не повинно бути збережено, поки не будуть збережені всі дії транзакції. Це означає, що кожен комп'ютер повинен записати свої поновлення умовно і очікувати від диспетчера розподілених транзакцій повідомлення про те, що дії всіх інших комп'ютерів також записані. Для цієї мети використовується алгоритм, званий двофазної фіксацією[5].

1.1.7. Поетапне проектування розподіленої бази

Основні етапи послідовності проектування розподіленої бази даних показані на рисунку 1.14.

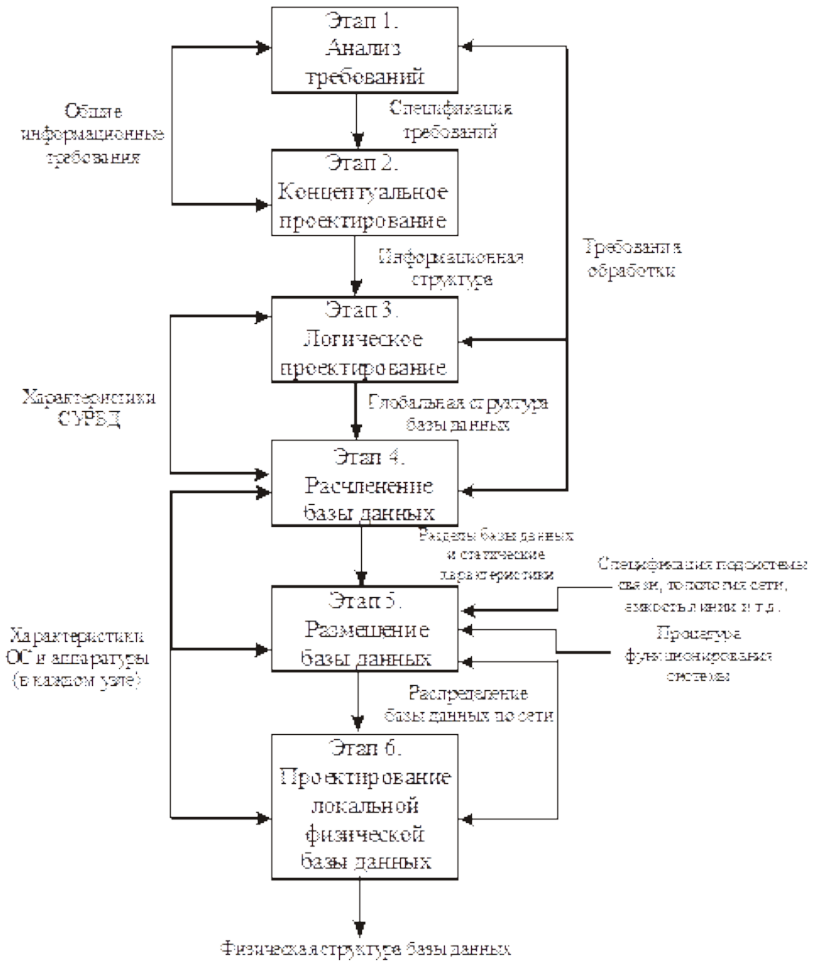

Рисунок 1.14 – Етапи проектування розподіленої баз даних

Відзначимо, що етапи 1, 2, 3 і 6 подібні етапах при проектуванні централізованої бази даних. Тому розглянемо тільки етапи 4 і 5 (етап розчленування бази даних і етап розміщення).

Кожен підфайл в розчленованої базі даних повинен вибиратися як неподільна одиниця розміщення даних.

Етап розчленування бази даних пов'язаний з розчленуванням глобальної бази даних і синтезом різних додатків на основі моделі.

На цьому етапі проектування вихідна глобальна база даних розчленовується на безліч подфайлов. Потрібно, щоб розчленовані підфайли містили в точності всі відомості, що були в глобальній базі даних. Крім вимоги про збереження інформації часто потрібна сумісність обмежень на розділи бази даних[6]. Це допустимий розмір і продуктивність.

При оцінці продуктивності повинні враховуватися також два основні чинники: час відгуку і надійність системи. Необхідно об'єднувати в підфайл такі часто використовувані спільно записи, щоб він (будучи розміщений з урахуванням частоти використання) поліпшував характеристики відгуку системи. Необхідно намагатися отримати необхідний рівень надійності, використовуючи за можливості меншу кратність дублювання.

На цьому етапі проводиться аналіз того, як додатки бази даних використовують можливі розділи бази даних. Моделі додатків можуть бути класифіковані в такий спосіб:

Програми, що використовують єдиний файл.

Програми, що використовують кілька файлів:

─ додатки, що допускають незалежну паралельну обробку;

─ додатки, що допускають синхронізовану обробку.

Після того як обрані стратегія розчленування і конкретна система управління базою даних, постає завдання раціонального розміщення даних по вузлах мережі. Для того щоб отримати кращу структуру, цей етап повторюється для кожного можливого варіанту розчленування. Цей процес триває до тих пір,

поки не буде отриманий задовільний результат чи будуть вичерпані всі допустимі розчленування.

При цьому необхідно мати інформацію про дані, вузлах мережі, додатках, їхні зв'язки, а також знати вимоги, що пред'являються до часу відгуку і надійності, і обмеження, що накладаються апаратними та програмними засобами мережі.

1.1.8. Архітектура розподілених БД

«Клієнт-сервер» - це вид розподіленої системи, в якій є сервер, який виконує запити клієнта, причому сервер і клієнт спілкуються між собою з використанням того чи іншого протоколу. Під клієнтом розуміється програма, яка використовує ресурси, а під сервером програма, яка обслуговує запити клієнтів на отримання ресурсів певного виду. Настільки широке визначення включає в себе практично будь-яку програмну технологію, в якій беруть участь більше однієї програми, функції між якими розподілені асиметрично [7].

Існує два підходи до розподілу обов'язків між сервером і клієнтом при виконанні запитів до баз даних. При використанні технології файлового сервера клієнт робить запит, сервер передає йому необхідні для виконання запиту файли, після чого клієнт вибирає з цих файлів потрібну інформацію. В цьому випадку левова частка роботи лягає на клієнта, по каналах зв'язку передається велика кількість даних, велика частина з яких насправді не з'являється у відповіді на запит.

Технологія «клієнт-сервер» передбачає, що відбір даних для відповіді на запит робиться сервером, а клієнту передається тільки результат - ті дані, які були запитані. На рисунку 1.15 показаний процес обміну інформацією між сервером і клієнтом при виконанні одного із запитів. При роботі з файл-серверною версією вся відповідальність за збереження і цілісність бази даних лежить на програмі і мережевий операційній системі. Обробка всіх даних відбувається на робочих місцях, а сервер використовується тільки як розділяється накопичувач. Кожен

користувач безпосередньо використовує інформацію і вносить зміни в файли даних і в індексні файли [7].

Рисунок 1.15 – Порівняння технологій файлового сервера и «клієнт-сервера»

При великих обсягах даних і роботі в багатокористувацькому режимі істотно знижується швидкодія - адже чим більше користувачів, тим вище вимоги до поділу даних. Крім того, може виникнути пошкодження баз даних. Наприклад, в момент запису в файл може виникнути збій мережі. В цьому випадку, комп'ютер перериває роботу і база даних може виявитися пошкодженою, а індексний файл - зруйнованим. Переіндексація, яку необхідно провести після подібних збоїв, може тривати кілька годин [6].

Клієнт-серверна версія дозволяє обійти ці проблеми, так як вся робота з базою даних відбувається на сервері, не проходить по дротах і не залежить від збоїв на робочих станціях. Всі запити на запис в файл перехоплюються сервером. У файл зміни вносяться тільки після того, як сервер отримає повідомлення про те, що коригування файлу завершено. Це виключає ушкодження індексних файлів і істотно підвищує швидкодію системи.

Крім високої швидкодії і надійності, архітектура «клієнт-сервер» дає багато переваг і в частині технічного забезпечення. По-перше, сервер оптимізує виконання функцій обробки даних, що позбавляє від необхідності оптимізації

робочих станцій. Робоча станція може бути укомплектована не надто швидким процесором, і, тим не менш, сервер дозволить швидко отримати результати обробки запиту. По-друге, оскільки робочі станції не обробляють всі проміжні дані, істотно знижується навантаження на мережу. З'являється можливість ведення журналу операцій, в якому автоматично реєструються всі минулі транзакції що, в свою чергу, допоможе швидкому відновленню системи під час апаратних збоїв [6].

Розрізняють дворівневу і трирівневу модель «клієнт-сервер». На рисунку 1.16 представлена дворівнева модель «клієнт-сервер». Ця модель, можливо, найбільш загальна, оскільки вона подібна до схеми розробки локальних баз даних. Багато системи «клієнт-сервер», що використовуються сьогодні, розвинулися з існуючих локальних додатків бази даних, які зберігають свої дані у файлі на сервері. Перенесення систем здійснюється для підвищення ефективності роботи, захищеності і надійності бази даних.

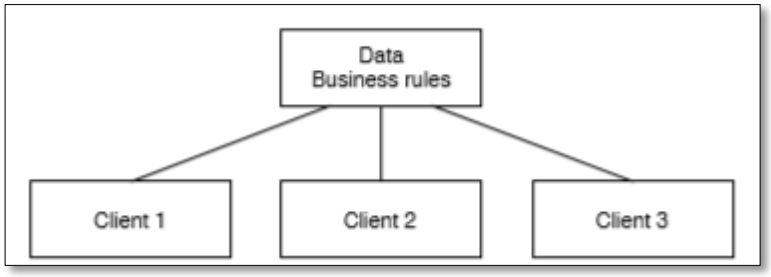

Рисунок 1.16 – Дворівнева модель «клієнт-сервер»

Переваги моделі: уможливлює, в більшості випадків, розподіл функцій обчислювальної системи між декількома незалежними комп'ютерами. Це дозволяє спростити обслуговування обчислювальної системи. Зокрема, заміна, ремонт, модернізація або переміщення сервера не зачіпають клієнтів. Всі дані зберігаються на сервері, який, як правило, захищений набагато краще за більшість клієнтів. На сервері простіше забезпечити контроль повноважень, щоб вирішувати доступ до даних тільки клієнтам з відповідними правами доступу; дозволяє об'єднати різні клієнти. Використовувати ресурси одного сервера часто можуть клієнти з різними апаратними платформами, операційними системами.

Недоліки моделі: непрацездатність сервера може зробити непрацездатною всю обчислювальну мережу; підтримка роботи даної системи вимагає окремого фахівця - системного адміністратора; висока вартість обладнання.

У такій моделі дані постійно знаходяться на сервері, а клієнтські додатки на своєму комп'ютері, або на сайті. Бізнес-правила (сервер) при цьому можуть розташовуватися на будь-якому з комп'ютерів (або навіть на обох одночасно).

Трирівнева архітектура клієнт-сервер - різновид моделі «клієнт-сервер», в якій функція обробки даних винесена на один або кілька окремих серверів. Це дозволяє розділити функції зберігання, обробки і представлення даних для більш ефективного використання можливостей серверів і клієнтів.

Переваги: масштабованість; конфігурованість - ізольованість рівнів один від одного дозволяє (при правильному розгортанні архітектури) швидко і простими засобами переконфігурувати систему при виникненні збоїв або при плановому обслуговуванні на одному з рівнів; висока безпека; висока надійність; низькі вимоги до швидкості каналу (мережі) між терміналами і сервером додатків; низькі вимоги до продуктивності і технічним характеристикам терміналів, як наслідок зниження їх вартості[6].

Недоліки випливають з переваг. У порівнянні с клієнт-серверної або файл-серверної архітектурою можна виділити наступні недоліки трирівневої архітектури: більш висока складність створення додатків; складніше в розгортанні і адмініструванні; високі вимоги до продуктивності серверів додатків і сервера бази даних, а, значить, і висока вартість серверного обладнання; високі вимоги до швидкості каналу (мережі) між сервером бази даних і серверами додатків.

На рисунку 1.17 показана трирівнева модель «клієнт-сервер». Тут клієнт це призначений для користувача інтерфейс до даних, а дані знаходяться на віддаленому сервері. Клиентной додаток робить запити для отримання доступу або зміни даних через сервер. Якщо клієнт, сервер і бізнес-правила розподілені по

окремих комп'ютерів, розробник може оптимізувати доступ до даних і підтримувати їх цілісність у всій системі.

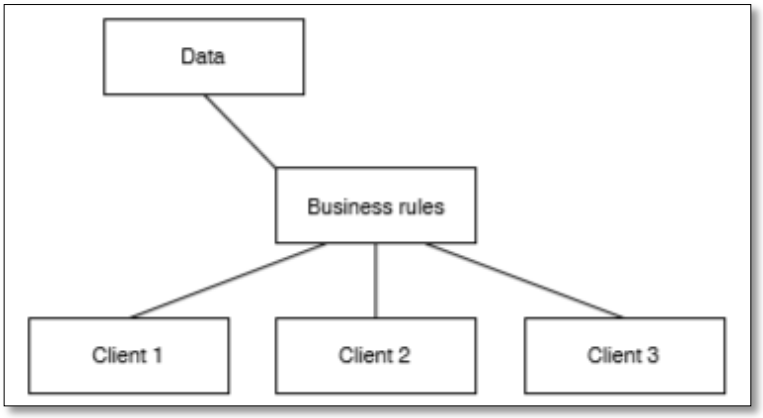

Рисунок 1.17 – Трирівнева модель «клієнт-сервер»

1.2.Постановка завдання

В рамках дослідження мають бути вирішені наступні завдання:

– створити програмну архітектуру розподіленої бази даних;

– розробити методи додавання, простої вибірки та вибірки зі з’єднанням даних;

– реалізувати методи додавання, простої вибірки та вибірки зі з’єднанням даних;

– дослідити ефективність розроблених методів.

Висновки до розділу

У розділі розглянуто основні принципи розподідених баз даних, методи побудови та різні підходи до виконання операцій в розподілених базах даних. Наведено принципи тиражування і фрагментації даних. Розглянуто поетапне проектування розподіленої бази даних, а також архітектруру розподіленої бази дани.

Отож обробка даних в розподіленій базі даних буде повністю залежати від принципу розміщення даних та обраної архітектури програмного забезпечення.

2 МАТЕМАТИЧНА СКЛАДОВА

2.1.Метод з'єднання даних

2.1.1. Вкладені цикли (Nested Loops Join)

Дано два набори даних R і S, алгоритм приєднання до вкладених циклів (R join S).

Це один з найдавніших Join алгоритмів і є одним з найпростіших. Він здатний об'єднати два набори даних на основі будь-якої умови приєднання. Він не страждає від недоліку наступних двох алгоритмів які можуть виконувати тільки equi-приєднання.

```
for all tuples r ε R do
  for all tuples s ε S do
    if JoinCondition then {JoinCondition can be like r.a == s.b}
      add (r,s) to the result
    end if
  end for
end for
```

Використовуючи цей алгоритм, будуть взяті дві таблиці, що містять 100 000 і 50 000 рядків. Приблизно півтори години потрібно, щоб виконати рівне приєднання, якщо кожен I/O займає 10 мс, вводи кластеризовані, а I/O робиться по одній сторінці за раз [7]

2.1.2. Сортування і об'єднання (Sort-Merge Join)

Дано два набори даних P і Q, алгоритм Sort-Merge Join сортує обидва набори даних по атрибуту приєднання, а потім шукає кваліфіковані кортежі p ∈ P і q ∈ Q по суті об'єднання двох наборів даних [7]. Крок сортування групує всі кортежі з однаковим значенням у стовпчику приєднання разом і тим самим полегшує ідентифікацію розділів або груп кортежів з однаковим значенням у стовпчику приєднання (join column). Цей розділ експлуатується порівнюванням P кортежі в розділі з лише Q кортежами в тому ж розділі (а не з усіма кортежами Q), тим самим уникаючи перерахування чи перехрестя P і Q. Цей підхід на основі розділів працює лише для умов приєднання рівності(==).

Має бути очевидним, що цей алгоритм дасть хороші показники, якщо вхідні набори даних вже відсортовані.

Алгоритм Reduce-Side Join багато в чому є розподіленою версією Sort-Merge Join. У алгоритмі Reduce-Side Join, кожен відсортований розділ надсилається до функції зменшення (reduce function) для об'єднання (merging) [7].

```
p ∈ P; q ∈ Q; gq ∈ Q
while more tuples in inputs do

  while p.a < gq.b do
    advance p
  end while
  while p.a > gq.b do
    advance gq {a group might begin here}
  end while
  while p.a == gq.b do
    q = gq {mark group beginning}
    while p.a == q.b do
      Add ⟨p, q⟩ to the result
      Advance q
    end while
    Advance p {move forward}
  end while
  gq = q {candidate to begin next group}
end while
```

2.1.3. Хешування (Hash Join)

Алгоритм приєднання Hash складається з фази "збірки"(‘build’) та фази "зондування"(‘probe’). У своєму найпростішому варіанті, менший набір даних завантажується в хеш-таблицю пам'яті у фазі збірки.

На етапі "зондування" більший набір даних сканується та з'єднується з відповідними кортежами дивлячись в хеш-таблицю. Такий алгоритм, як і алгоритм Sort-Merge Join працює лише для equi-приєднань[7].

Цей алгоритм, як правило, швидший, ніж сортування-об’єднання, але додає значну кількість навантаження на пам'ять для зберігання хеш-таблиці.

Розглянемо два набори даних P і Q. Алгоритм простого хеш-з'єднання виглядатиме так

```
for all p ∈ P do
   Load p into in memory hash table H
end for
for all q ∈ Q do

   if H contains p matching with q then
      add ⟨p,q⟩ to the result
   end if
end for
```

Алгоритм Broadcast Join є розподіленим варіантом алгоритму Hash Join. Менший набір даних надсилається кожному вузлу в розподіленому кластері. Потім він завантажується в хеш-таблицю в пам'яті. Кожна map function йде через виділений фрагмент вхідного набору даних і зондує цю хеш-таблицю для пошуку відповідності кортежів[8].

2.1.4. Reduce-Side Join

Процес, коли операція з'єднання виконується у фазі редуктора. В основному, зменшення бокового з'єднання відбувається наступним чином:

- Mapper зчитує вхідні дані, які слід об'єднати на основі загального стовпця або ключа з'єднання.

- Маппер обробляє вхід і додає тег до входу, щоб відрізнити вхід, що належить з різних джерел або наборів даних або баз даних.

- Маппер виводить проміжну пару ключ-значення, де ключ є не що інше, як ключ з'єднання.

- Після фази сортування та переміщення створюється ключ та список значень для редуктора.

- Тепер редуктор з'єднує значення, присутні в списку, з ключем, щоб дати остаточний сукупний вихід[9].

Принцип роботи Reduce-Side Join алгоритму подано на рисунку 2.1.

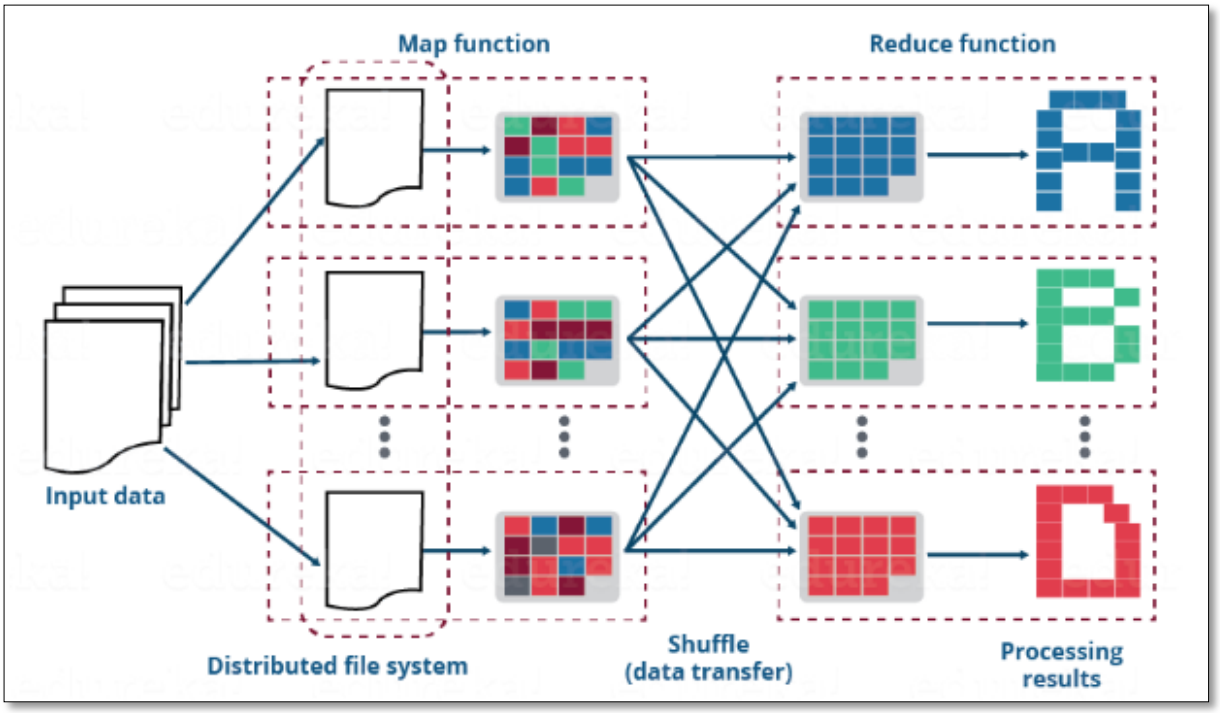

Рисунок 2.1 – Алгоритм поєднання даних Reduce-Side Join

2.1.5. Map-Side Join

Припустимо, що у нас є дві таблиці, з яких одна з них - невелика таблиця. Коли ми подаємо завдання зменшити карту, перед початковою задачею приєднати до карт зменшення картки буде створено завдання Зменшити карту, яке буде читати дані невеликої таблиці та зберігати їх у хеш-таблиці пам'яті. Після читання він серіалізує хеш-таблицю в пам'яті у файл таблиці хеш-книг.

На наступному етапі, коли виконується оригінальна задача об’єднання Map Reduce, вона переміщує дані у файлі хеш-таблиці до кешованого розподіленого кеша, який заповнює ці файли на локальний диск кожного картографа. Таким чином, всі картографи можуть завантажити цей стійкий файл хеш-таблиці назад у пам’ять і виконувати роботу приєднання, як і раніше. Після оптимізації маленьку таблицю потрібно прочитати лише один раз. Крім того, якщо на одній машині працює кілька картографічних операцій, розподіленому кешу потрібно лише надіслати на цю машину одну копію файлу хеш-таблиці[10].

Переваги використання з'єднання на карті:

– Об’єднання на стороні карти допомагає мінімізувати витрати, які виникають під час сортування та об'єднання в перетасуванні та зменшення етапів.

– Приєднання на карті також допомагає покращити виконання завдання за рахунок скорочення часу на виконання завдання.

Недоліки з'єднання на карті:

– Об'єднання сторони карти є адекватним лише тоді, коли одна з таблиць, за якою ви виконуєте операцію з'єднання на карті, є достатньо малою, щоб вміститись у пам'яті. Отже, не підходить для виконання об'єднання на карті на таблицях, які обидві мають величезні дані.

Принцип роботи Map-Side Join алгоритму подано на рисунку 2.2.

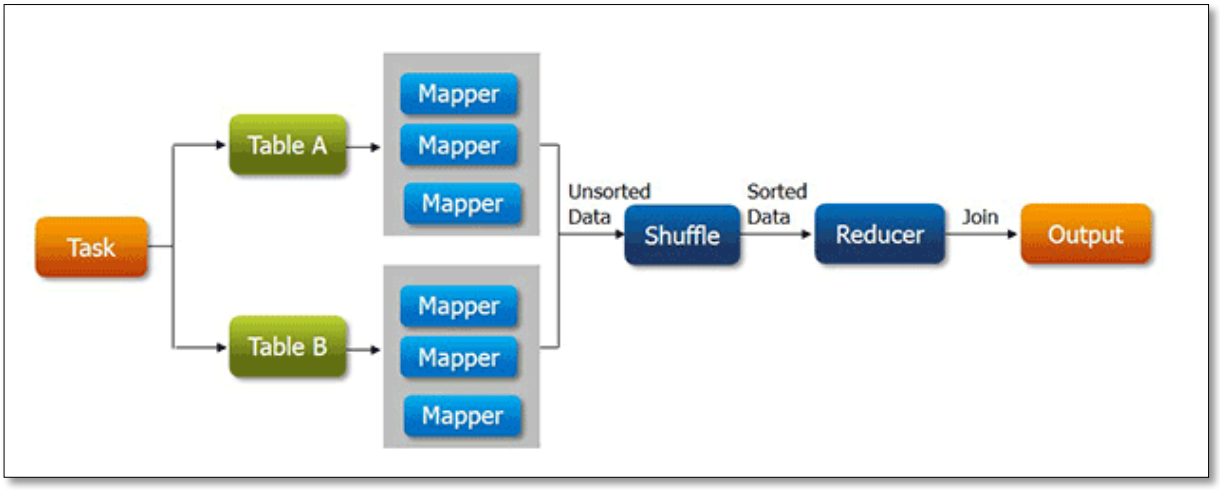

Рисунок 2.2 – Алгоритм поєднання даних Map-Side Join

2.1.6. Broadcast Join

Якщо один із наборів даних дуже маленький, так що він може помістится в пам'яті, оптимізація може бути використана, щоб уникнути накладних витрат на передачу даних, які беруть участь у перенесенні значень з Mappers на Редуктори. Такий сценарій часто зустрічається в реальних програмах. Наприклад, малу базу даних користувачів може знадобитися з'єднати з великим журналом. Цей невеликий набір даних може бути просто тиражуються на всіх машинах. Цього можна досягти просто за допомогою –файлів або -архіва директива надсилати файл на кожну машину під час виклику завдання Hadoop.

Broadcast Join - це лише Map-алгоритм. Перевага від того, що він використовує невелике "локальне" сховище в окремі вузли. Це дає можливість завантажити весь набір даних в хеш-таблицю пам'яті, доступ до якої дуже швидкий[11]. Але з іншої сторони, це буде виникати проблеми, коли обидва набори даних є великими, і жоден з них не може зберігатися локально на окремих вузлах.

Висновки до розділу

У розділі розглянуто різні підходи до виконання операцій в розподілених базах даних. Розглянуто проблеми виконання розподілених запитів, а також методи оптимізації в розподіленій обробці запитів; основні алгоритми виконання операції поєднання даних.

В результаті для ефективної вибірки даних в розподілених базах потрібно враховувати кількість фрагментів даних, їх розмір, при аналізі цих показників обрати найдоречніший алгоритм поєднання даних, шляхом перенесення фрагментів на інші сайти або перенесенням готових поєднань.

Простим в реалізації є алгоритм Sort-Merge Join. Хоча простий має ряд переваг, кожний рядок таблиці зчитується тільки раз на відміну від Nested Loops Join (хоча і вимагає додаткових зусиль сортування по стовпцям перед поєднанням, де доцільно використати алгоритм паралельного сортування). Цей алгоритм краще ніж Hash Join впорається з велекими таблицями, а також допускає не тільки умови рівності.

Цікавим в реалізації є алгоритм Reduce-Side Join, який чудово підходить для поєднання двох великих таблиць, які не можуть розміститися в пам’яті. Частина роботи сортування проходить на етапі сортування та переміщення на етапі MapReduce, який поєднує значення одного ключа та надсилає його на той же редуктор. А отже на стороні Reduce відсутній жорсткий формат даних, що дає можливість виконувати операцію приєднання на неструктурованих даних.

3 АЛГОРИТМ ТА МЕТОД ОБРОБКИ ДАНИХ

3.1.Архітектура програмного забезпечення

Програмне забезпечення виконано на мові програмування C# .NET Core [12,13], для доступу до бази даних використано Entity Framework Core [14], бази даних Microsoft SQL Server [16].

Додати схему архітектури.

3.2.Компоненти програмного забезпечення

Програмне забезпечення включає в себе один додаток з конфігурацією для підключення до кількох баз даних.

3.3.Алгоритм поєднання даних

навести схему алгоритму

Алгоритм виконання поєднання даних скдається з трьох основних етапів: розбір і валідація надісланого запиту, аналіз таблиць зазначених у запиті та вибір стратегії, виконання обраного алгоритму для опреції поєднання даних. Кожен етап включає в себе кілька кроків виконання. Далі детально описано кожен етап з усіма функціональними кроками.

3.3.1. Розбір і валідація запиту

Программа очікує на вхід SQL запит у вигляді строки, відповідно до стандарту T-SQL. Після отримання SQL строки проводиться валідація запиту:

– SQL синтакс правильний – підримуються ключові слова select, from, join, on, where.

– Таблиці існують – перевірка чи існуть зазначені в запиті таблиці хоча б у одній базі даних.

– Колонки існують – перевірка чи існуть зазначені колонки у таблицях наведених у запиті, можна також використовувати символ * для вибору всіх колонок, але слід зменшити кількість таких запитів, адже кількість колонок для виведення напряму впливає на швидкість виконання запиту.

3.3.2. Аналіз таблиць і вибір стратегії

Після розбору і валідації SQL запиту переходимо до аналізу таблиць, щоб підібрати найкращу стратегію поєднання даних. Результат попереднього кроку дозволяє зрозуміти які таблиці використовуються у поєднанні даних. Отож, аналіз таблиць включає в себе наступні кроки:

– Розміщення таблиць – перевірка в якій базі даних знаходится таблиця.

– Розподіл таблиць – перевірка чи розподілена таблиця в кількох базах даних.

– Розмір таблиць – перевірка розміру таблиці за допомогою стандартної команди [exec sp_spaceused 'ім’я таблиці'], що повертає значення кількості рядків в таблиці, та розміру даних в КВ.

Отож, розглянемо пошук стратегії поєднання даних для двох таблиць.

Якщо таблиця1 і таблиця2 знаходяться в одній базі – використати стандартний механізм поєдняння даних на сторні бази даних.

Якщо таблиця1 горизонтально розподілена і таблиця2 невеликого розміру (може розміститися в пам’яті), то надіслати таблицю2 в іншу базу даних для виконання приєднання [15].

Якщо таблиця1 і таблиця2 горизонтально розподілені і невеликого розміру (можуть розміститися в пам’яті), то виконати вибірку даних в базі, а операцію поєднання на строні серверу за прикладом Map-Side Join.

Якщо таблиця1 і таблиця2 горизонтально розподілені і великого розміру (не можуть розміститися в пам’яті), то виконати об’єднання даних згідно алгоритму Reduce-Side Join.

3.3.3. Виконання оперції поєднання

Виконная опереції поєднання відбувається згідно результатів попереднього кроку.

3.4.Алгоритм додавання даних

навести схему алгоритму.

Алгоритм додавання (вставки) даних скдається з трьох основних етапів: розбір і валідація надісланого запиту, аналіз таблиць зазначених у запиті та вибір стратегії, безпоседньо виконання опреції вставки даних. Кожен етап включає в себе кілька кроків виконання. Далі детально описано кожен етап з усіма функціональними кроками.

3.4.1. Розбір і валідація запиту

Программа очікує на вхід SQL запит у вигляді строки, відповідно до стандарту T-SQL. Після отримання SQL строки проводиться валідація запиту:

– SQL синтакс правильний – підримуються ключові слова insert, into, values, where.

– Таблиці існують – перевірка чи існуть зазначені в запиті таблиці хоча б у одній базі даних.

– Колонки існують – перевірка чи існуть зазначені колонки у таблицях наведених у запиті, чи всі обов’язкові колонки вказані.

3.4.2. Аналіз таблиці і вибір стратегії

– Після розбору і валідації SQL запиту переходимо до аналізу таблиць, щоб підібрати найкращу стратегію вставки даних. Результат попереднього кроку дозволяє зрозуміти в яку таблицю необхідно додати дані. Отож, аналіз таблиць включає в себе наступні кроки:

– Розміщення таблиці – перевірка в якій базі даних знаходится таблиця.

– Розподіл таблиці – перевірка чи розподілена таблиця в кількох базах даних.

– Розмір таблиці – перевірка розміру таблиці за допомогою стандартної команди [exec sp_spaceused 'ім’я таблиці'], що повертає значення кількості рядків в таблиці, та розміру даних в KB.

Отож, розглянемо пошук стратегії додавання даних у таблицю.

Якщо таблиця1 знаходиться в одній базі – використати стандартний механізм вставки даних на сторні бази даних.

Якщо таблиця1 горизонтально розподілена і встановлено базу з привілегією збереження – використати стандартний механізм вставки даних на сторні бази даних з привілегією.

Якщо таблиця1 горизонтально розподілена – перевірити розмір таблиць в кожній базі і надати перевагу меншій за об’ємом – використати стандартний механізм вставки даних на сторні бази даних з меншим об’ємом.

3.4.3. Виконання оперції вставки

Виконная опереції вставки відбувається згідно результатів попереднього кроку.

Висновки до розділу

У розділі роглянуто архітектуру розробленого програмного забезпечення, компоненти і технології використанні при проектуванні. Також докладно розглянуто реалізацію алгоритмів поєднання даних з розподілених таблиць і додовання (вставки) даних з обранням найдоречнішої таблиці для випадку горизонтально розподілених таблиць.

Виявлено, що стандартні методи для роботи з кількома базами в одному контексті в одному програмному забезпечені відсутні, отож сформовано окремий контест для кожної бази і підбран алгоритм для найефективнішого способу поєдняння даних враховуюси розмір та розподіленість таблиць. Порівняння ефективності різних алгоритмів на різних наборах даних наведено в наступному розділі.

4 ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ

4.1. Апаратне забезпечення для проведення експерименту

Для тестування використовувався персональний комп'ютер:

– Процесор: Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz;

– RAM: 16.0 Gb Kingston DDR4 SDRAM.

4.2. Опис експерименту

Для дослідження ефективності роботи розробленого алгоритму порівняємо швидкість виконання вибірки і поєднання даних на різних розмірах таблиць, що знаходяться в одній або двох базах даних.

Для проведення експерименту створено кілька таблиць з різною кількістю рядків у двох базах даних. Перевіримо кілька сценаріїв.

4.3. Результати експерименту

Експеримент 1: проста вибірка з таблиці, яка знаходиться в одній базі даних.

Результати експерименту подані в таблиці 4.1 і на рисунку 4.1.

Таблиця 4.1 – Результат експерименту 1

Кількість рядків в таблиці, шт	*Стандарний підхід, мс*	*Розроблений алгоритм, мс*
10	2	5
100	13	12
1000	32	30
10000	280	305

Рисунок 4.1 – Результат експерименту 1

Експеримент 2: проста вибірка з таблиці, яка знаходиться у двох базах даних.

Результати експерименту подані в таблиці 4.2 і на рисунку 4.2.

Таблиця 4.2 – Результат експерименту 2

Кількість рядків в таблиці, шт	*Стандарний підхід, мс*	*Розроблений алгоритм, мс*
10	4	4
100	19	18
1000	42	40
10000	350	357

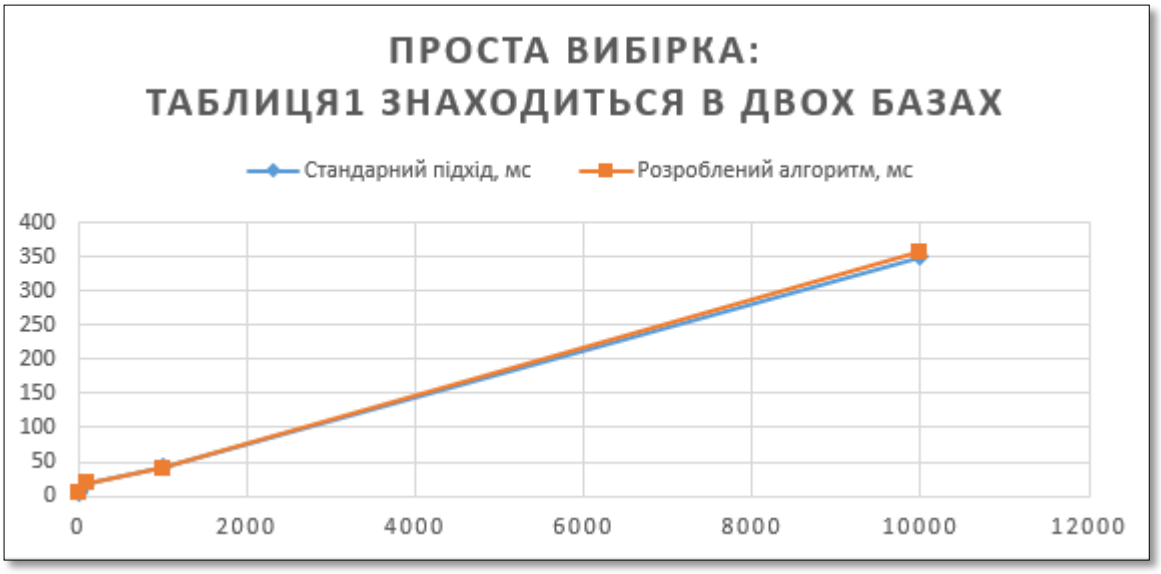

Рисунок 4.2 – Результат експерименту 2

Експеримент 3: вибірка поєднання з двох таблиць, які знаходиться у одній базі даних.

Результати експерименту подані в таблиці 4.3 і на рисунку 4.3.

Таблиця 4.3 – Результат експерименту 3

Кількість рядків в таблиці, шт	*Стандарний підхід, мс*	*Розроблений алгоритм, мс*
10	5	5
100	16	15
1000	40	38
10000	310	305

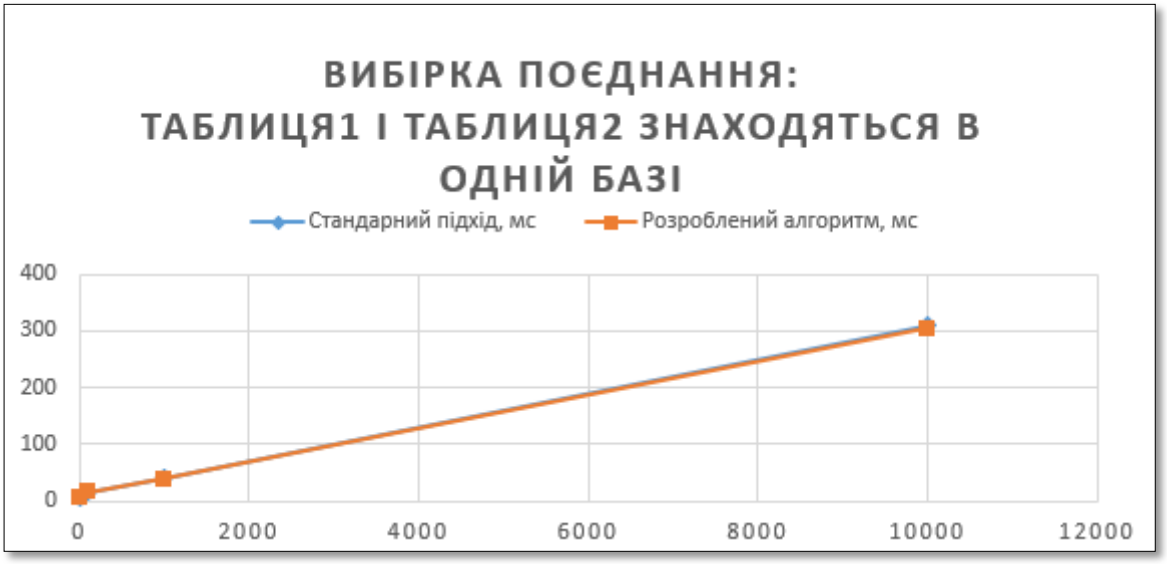

Рисунок 4.3 – Результат експерименту 3

Експеримент 4: вибірка поєднання з двох таблиць, які знаходиться у різних базах даних.

Результати експерименту подані в таблиці 4.4 і на рисунку 4.4.

Таблиця 4.4 – Результат експерименту 4

Кількість рядків в таблиці, шт	*Стандарний підхід, мс*	*Розроблений алгоритм, мс*
10	9	5
100	34	17
1000	71	42
10000	680	409

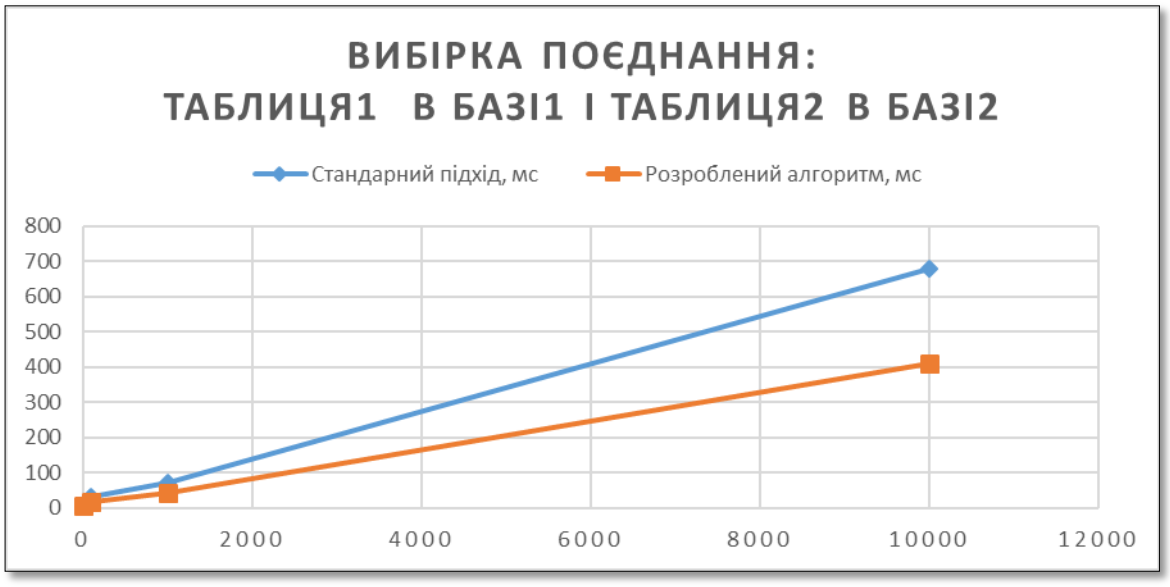

Рисунок 4.1 – Результат експерименту 4

Експеримент 5: вибірка поєднання з двох таблиць, перша знаходиться в обох базах даних, друга - в одній.

Результати експерименту подані в таблиці 4.5 і на рисунку 4.5.

Таблиця 4.5 – Результат експерименту 5

Кількість рядків в таблиці, шт	*Стандарний підхід, мс*	*Розроблений алгоритм, мс*
10	10	6
100	42	21
1000	79	45
10000	720	427

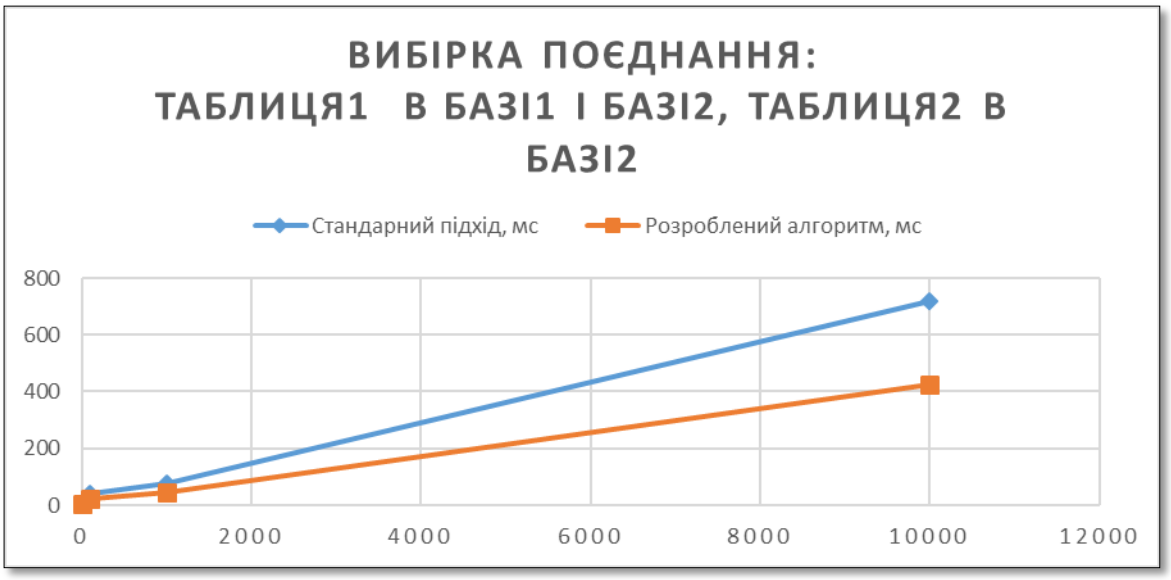

Рисунок 4.5 – Результат експерименту 5

Висновки до розділу

У розділі проведено тестування ефективності розробленого агоритму у порівнянні з існуючим підходом вибірки і поєднання даних, що знаходяться в розподілених базах.

Розроблений алгоритм виконує вибірку і поєднання даних зі швидкістю аналогічно існуючиму підходу. Проте, коли таблиці розміщені в різних базах даних, або таблиці горизонтально розподілені розроблений алгоритм дає 40% кращий час виконання запиту на 10000 тис рядків.

З експериментальних даних можна зробити висновок, що алгоритм є ефективним для поєднання даних з розподілених таблиць.

5 РОЗРОБЛЕННЯ СТАРТАП ПРОЕКТУ

ВИСНОВКИ

В рамках магістерської дисертації розроблено математичне та програмне забезпечення методів обробки даних в розподілених базах на платформі .NET. Програмний продукт виконаний на мові програмування C# з використанням .NET Core, для доступу до бази даних використано Entity Framework Core, підключені бази даних Microsoft SQL Server.

У першому розділі розглянуто основні принципи розподідених баз даних, методи побудови та різні підходи до виконання операцій в розподілених базах даних. Наведено принципи тиражування і фрагментації даних. Розглянуто поетапне проектування розподіленої бази даних, а також архітектруру розподіленої бази дани. Виведено, що обробка даних в розподіленій базі даних повністю залежить від принципу розміщення даних та обраної архітектури програмного забезпечення.

У другому розділі розглянуто різні підходи до виконання операцій в розподілених базах даних. Розглянуто проблеми виконання розподілених запитів, а також методи оптимізації в розподіленій обробці запитів; основні алгоритми виконання операції поєднання даних. Виведено, що для ефективної вибірки даних в розподілених базах потрібно враховувати кількість фрагментів даних, їх розмір, при аналізі цих показників обрати найдоречніший алгоритм поєднання даних, шляхом перенесення фрагментів на інші сайти або перенесенням готових поєднань.

У третьому розділі роглянуто архітектуру розробленого програмного забезпечення, компоненти і технології використанні при проектуванні. Також докладно розглянуто реалізацію алгоритмів поєднання даних з розподілених таблиць і додування (вставки) даних з обранням найдоречнішої таблиці для випадку горизонтально розподілених таблиць. Виявлено, що стандартні методи для роботи з кількома базами в одному контексті в одному програмному забезпечені відсутні, отож сформовано окремий контест для кожної бази і підбран

алгоритм для найефективнішого способу поєдняння даних враховуюси розмір та розподіленість таблиць.

Щоб визначити ефективність роботи алгоритму, у четвертому розділі проведено тестування ефективності розробленого агоритму у порівнянні з існуючим підходом вибірки і поєднання даних, що знаходяться в розподілених базах. Експериментально перевірено, що розроблений алгоритм виконує вибірку і поєднання даних зі швидкістю аналогічно існуючиму підходу. Проте, коли таблиці розміщені в різних базах даних, або таблиці горизонтально розподілені розроблений алгоритм дає на 40% кращий час виконання запиту при тестуванні на 10 000 тис рядках.

З експериментальних даних можна зробити висновок, що алгоритм є ефективним для поєднання даних з розподілених таблиць.

Наукова новизна розробки полягає у розробці модифікованого методу поєднання даних в розподілених базах.

Усі поставленні задачі наукової роботи були виконані. Було створено програмну архітектуру розподіленої бази даних, розроблено і реалізовано методи додавання, вибірки та з’єднання даних, а також досліджено ефективність розроблених методів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Andrew S. Tanenbaum, Maarten van Steen, 2016. Distributed Systems: Principles and Paradigms 2nd Edition, pp.427-438
2) Andrew S. Tanenbaum, Maarten van Steen, 2017. Distributed Systems 3rd Edition, pp.324-328
3) Особливості проектування розподіленої бази даних [Електронний ресурс] – Режим доступу: https://studopedia.su/19_167104_osobennosti-proektirovaniya-raspredelennoy-bazi-dannih.html
4) Яковлев Ю.С., 2017. О концепции построения и выбора распределенных баз данных информационно-поисковых систем [Электронный ресурс] – Режим доступа:http://dspace.nbuv.gov.ua/bitstream/handle/123456789/727/6-Yakovlev.pdf
5) Distributed Database Architecture [Electronic Resource] – Mode of access: https://docs.oracle.com/html/E25494_01/ds_concepts001.htm
6) Проектування для розподілених баз даних [Електронний ресурс] – Режим доступу: http://bourabai.kz/dbt/servers/12.html
7) M. Tamer Ozsu, Patrick Valduriez. Distributed and parallel database systems [Electronic Resource] – Mode of access: http://citforum.ru/database/classics/distr_and_paral_sdb/
8) S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian, 2010. A comparison of join algorithms for log processing in mapreduce. In SIGMOD ’10: Proceedings of the 2010 international conference on Management of data, pp.975–986.
9) R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou, 2008. Scope: easy and efficient parallel processing of massive data sets, pp.1265–1276.
10) S. Viglas, 2010. Advanced databases [Electronic Resource] – Mode of access: http://www.inf.ed. ac.uk/teaching/courses/adbs.

11) J. C. Olamendy, 2010. Distributed database technologies to share data between database systems [Electronic Resource] – Mode of access: https://www.c-sharpcorner.com/uploadfile/john_charles/distributed-database-technologies-to-share-data-between-database-systems/
12) C# documentation [Electronic Resource] – Mode of access: https://docs.microsoft.com/en-us/dotnet/csharp/
13) ASP.NET Core [Electronic Resource] – Mode of access: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-3.1
14) Entity Framework Core [Electronic Resource] – Mode of access: https://docs.microsoft.com/en-us/ef/core/
15) Raw SQL Queries [Electronic Resource] – Mode of access: https://docs.microsoft.com/en-us/ef/core/querying/raw-sql
16) Microsoft SQL Server 2019 [Electronic Resource] – Mode of access: https://www.microsoft.com/en-us/sql-server/sql-server-2019