Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра програмних систем і технологій

УДК 004.942

На правах рукопису

ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Тема: "Дослідження архітектури та управління систем з розподіленими базами даних"

(назва згідно з наказом ректора)

Спеціальність – 121 "Інженерія програмного забезпечення"

ПОЯСНЮВАЛЬНА ЗАПИСКА

БР.ІПЗ - 33.00.00.000 ІПЗ

(позначення)

Студент

ІПЗ-43_____/Денис КАРАНДАСЬ/

(шифр групи) (підпис) (дата) (розшифровка підпису)

Науковий керівник

к.т.н., асист._____/Максим ТКАЧЕНКО/

(посада) (підпис) (дата) (розшифровка підпису)

Консультант

з питань нормоконтролю

фахівець _____/Тамара ЧАПОВСЬКА/

(посада) (підпис) (дата) (розшифровка підпису)

Допускається до захисту

з питань нормоконтролю

Завідувач кафедри

д.т.н., проф._____/Олексій БИЧКОВ/

(посада) (підпис) (дата) (розшифровка підпису)

Київ – 2021

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра програмних систем і технологій

Освітньо-кваліфікаційний рівень бакалавр

Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖЕНО

Зав. кафедри програмних систем і технологій

_____ (Олексій БИЧКОВ)

(підпис) (розшифровка підпису)

“__”_____2021р

ЗАВДАННЯ

НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Карандасю Денису Дмитровичу

(прізвище, ім’я, по-батькові)

1. **Тема бакалаврської роботи** “Дослідження архітектури та управління систем з розподіленими базами даних”
   керівник проекту (роботи) Ткаченко Максим Васильович, к.т.н., асистент
   затверджені наказом вищого навчального закладу від “ 11 ” листопада 2020 р. № 6
2. **Строк подання студентом роботи**_____
3. **Вихідні дані до проекту (роботи)** Система управління розподіленими базами даних, модель програмного застосунку
4. **Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)**
   1. Аналіз різновидів архітектури систем управління розподіленими базами даних.
   2. Аналіз способів розподілу даних, зберігання та взаємодія з ними.
   3. Дотримання правил та концепцій роботи з базами даних.
   4. Переваги та недоліки використання розподілених систем баз даних.

5. Розробка моделі та програмної реалізації системи розподіленого доступу до баз даних.

6. Аналіз та порівняння результатів

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

1. Архітектура однорангової мережі (рис. 1.1, ст. 16)
2. Горизонтальна фрагментація (рис. 1.2, ст. 21)
3. Вертикальна фрагментація (рис. 1.3, ст. 22)
4. Архітектура програмного забезпечення (рис. 2.1, ст. 27)
5. Діаграма прецедентів (рис. 2.2, ст. 29)
6. Діаграма розгортання (рис. 2.3, ст. 30)
7. Діаграма послідовносні створення нової таблиці (рис. 2.4, ст. 31)
8. Модуль управління процесом (рис. 2.5, ст. 35)
9. Процес взаємодії з вузлом мережі (рис. 3.1, ст. 41)
10. Список вузлів користувача (рис. 3.2, ст. 43)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основна частина	к.т.н., асист., Ткаченко М.В.		

7. Дата видачі завдання _____

Керівник _____ /Максим ТКАЧЕНКО/
(підпис) *(розшифровка підпису)*

Завдання прийняв до виконання _____ /Денис КАРАНДАСЬ/
(підпис) *(розшифровка підпису)*

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Ознайомлення з літературою	08.12.2020 - 28.12.2020	виконано
2	Аналіз архітектурних моделей	12.01.2021 - 18.01.2021	виконано

3	Аналіз концепцій розподілу даних	20.01.2021 - 27.01.2021	виконано
4	Дослідження існуючих рішень	02.02.2021 - 08.02.2021	виконано
5	Вибір архітектури та проектування моделі системи	14.02.2021 - 12.03.2021	виконано
6	Вибір засобів та технологій	20.03.2021 - 21.03.2021	виконано
7	Розробка програмного застоснку	24.03.2021 - 21.04.2021	виконано
8	Тестування та вдосконалення програмного забезпечення	26.04.2021 - 06.05.2021	виконано
9	Підготовка звіту та презентації	12.05.2021 - 25.05.2021	виконано
10	Перевірка виконаної роботи	26.05.2021 - 27.05.2021	виконано

Студент – бакалавр _____ /Денис КАРАНДАСЬ/
(підпис) *(розшифровка підпису)*

Керівник роботи _____ /Максим ТКАЧЕНКО/
(підпис) *(розшифровка підпису)*

АНОТАЦІЯ

Випускна кваліфікаційна бакалаврська робота: 62 сторінки, 21 рисунок, 3 додатки, 13 джерел.

Тема: Дослідження архітектури та управління систем з розподіленими базами даних.

Об'єкт дослідження: дані, що описують роботу з великими об'ємами інформації у базах даних.

Мета роботи: дослідити різновиди архітектури, розробити модель та реалізувати систему управління розподіленими базами даних.

Предмет дослідження: системи управління розподіленими базами даних, досліджуються різновиди архітектури та принципи організації і взаємодії.

Результати дослідження:

Досліджено різновиди архітектури та реалізовано систему упраління розподіленеми базами даних. Досліджені концепції та способи розподілу даних. Проаналізовані переваги та недоліки використання даного підходу і запропонована власна програмна реалізація.

Висновок

В результаті роботи було досліджено різновиди архітектури, побудавана модель та реалізована система упраління розподіленеми базами даних.

БАЗИ ДАНИХ, СИСТЕМИ РОЗПОДІЛЕНИХ БАЗ ДАНИХ, ФРАГМЕНТАЦІЯ, ТРАНЗАКЦІЇ, АРХІТЕКТУРА СИСТЕМ УПРАВЛІННЯ, ВЕБ-ТЕХНОЛОГІЇ.

АННОТАЦИЯ

Выпускная квалификационная бакалаврская работа: 62 страницы, 21 рисунок, 3 дополнения, 13 источников.

Тема: Исследование архитектуры и управления систем с распределенными базами данных.

Объект исследования: данные, описывающие работу с большими объемами информации в базах данных.

Цель работы: исследовать разновидности архитектуры, разработать модель и реализовать систему управления распределенными базами данных.

Предмет исследования: системы управления распределенными базами данных, исследуются разновидности архитектуры и принципы организации и взаимодействия.

Результаты исследования:

Исследовано разновидности архитектуры и реализована система упраления распределенными базами данных. Исследованы концепции и способы распределения данных. Проанализированы преимущества и недостатки использования данного подхода и предложена собственная программная реализация.

Вывод

В результате работы были исследованы разновидности архитектуры, побудавана модель и реализована система упраления розподиленемы базами данных.

БАЗЫ ДАННЫХ, СИСТЕМЫ РАСПРЕДЕЛЕННЫХ БАЗ ДАННЫХ, ФРАГМЕНТАЦИЯ, ТРАНЗАКЦИИ, АРХИТЕКТУРА СИСТЕМ УПРАВЛЕНИЯ, ВЕБ-ТЕХНОЛОГИИ.

ANNOTATION

Final qualifying bachelor's thesis: 62 pages, 21 images, 3 additions, 13 sources.

Topic: Research of architecture and management of systems with distributed databases.

Object of research: data describing the work with large amounts of information in databases.

Purpose: explore the types of architecture, develop a model and implement a distributed database management system.

Subject of research: distributed database management systems, the types of architecture and principles of organization and interaction are studied

Research results:

Varieties of architecture are researched and the system of management of distributed databases is realized. Concepts and methods of data distribution are investigated. The advantages and disadvantages of using this approach are analyzed and the own software implementation is offered.

Conclusion

As a result, the types of architecture were studied, a model was built and a distributed database management system was implemented.

DATABASES, DISTRIBUTED DATABASE SYSTEMS, FRAGMENTATION, TRANSACTIONS, MANAGEMENT SYSTEMS ARCHITECTURE, WEB TECHNOLOGIES.

ЗМІСТ

Стр.

РОЗДІЛ 3

РОЗДІЛ 4

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД	-	база даних
СУБД	-	система управління базами даних
РБД	-	розподілена база даних
ПЗ	-	програмне забезпечення
API	-	Application Programming Interface
UML	-	Unified Modeling Language
HTTP	-	HyperText Transfer Protocol
HTTPS	-	HyperText Transfer Protocol Secure
JWT	-	JSON Web Token
JSON	-	JavaScript Object Notation
SQL	-	Structured Query Language
REST	-	Representational State Transfer
DRDA	-	Distributed Relational Database Architecture

ВСТУП

На даному етапі розвитку і використання баз даних вони стали невід'ємною частиною будь-якого програмного засобу, що працює з даними. Існує багато різних варіантів організації, зберігання та взаємодії з великими об'ємами інформації, але з часом, коли даних стає надвичайно багато, системи починають працювати повільніше і виникає потреба пошуку альтернативних рішень. Тому, досить поширеною стала тема дослідження концепцій розподілу даних.

Розподіл даних у базах даних дозволяє зменшити об'єм інформації за рахунок її розподілу по різним частинам мережі. Кожен вузол мережі може мати свій набір даних і доступ до них відбуватиметься по необхідності. Таким чином, мы запобігаємо взаємодії з налишковим набором даних і збільшуємо швидкість відповіді серверів. Для реалізації такого підходу використовуються спеціальні системи управління розподіленими базами даних, які демонструють користувачу ілюзію роботи з повноцінною БД.

Користувачі можуть використовувати системи управління сховищами даних для власних потреб або бізнес рішень. В перспективі дана технологія може досить добре зарекомендувати себе у якості веб-сервісу або платформи, що оптимізує процеси роботи з даними. Тому за останні роки досить великої уваги набули дослідження способів налаштування і покращення взаємодії з розподіленими базами даних.

Метою бакалаврської роботи є дослідження архітектури та управління розподіленеми базами даних з обгрунтуванням перспектив розвитку технології розподілу даних. Реалізація власної моделі системи на основі існуючої архітектури з можливістю масштабування та покращення ефективністі роботи. Розробка власного програмного застосунку на основі розглянутого теоретичного метріалу, підходів побудови систем управління РБД і розглянутих сучасних аналогів. При цьому, доречним буде організувати більш спрощенну модель системи, але найбільш схожу

до сучаних програмних комплексів, що надають управління розподіленими базами даних.

Задачі роботи:

- Аналіз предметної області.
- Огляд існуючих рішень.
- Реалізація архітектури та проектування моделі системи.
- Формування основних компонентів системи.
- Розробка програмного застосунку управління РБД.
- Порівняння результатів з існуючими аналогами.
- Підведення підсумків.

Об’єктом дослідження є системи, що працюють з великими об’ємами інформації у базах даних та надають необхідний функціонал для доступу та взаємодії з даними.

Предметом дослідження є моделі, що описують структуру програмного застосунку, та самі системи управління розподіленими базами даних, що здійснюють управління даними у різних вузлах мережі.

Методи дослідження – пошук та ознайомлення з інформацією, аналіз даних, дослідження архітектури та існуючих рішень, огляд та вивчення літератури, перегляд лекцій та відео матеріалів, порівняння систем управління розподілених баз даних на основі досліджень сучасних аналогів.

Новизна одержаних результатів – ввдосконалено модель систем управління РБД та запропоновані шляхи покращення, що дозволять розробляти масштабовані та більш оптимізовані застосунки.

Практичне значення одержаних результатів полягає у тому, що розроблений програмний застосунок можливо використовувати для вирішення ряду проблем пов’язаних з оптимазацією процесу взаємодії та навантаженням на системи. Також

він дозволяє самостійно розподіляти дані між базами даних і досить легко керувати усіма вузлами мережі.

Особистий внесок:

- Розроблена модель системи управління розподіленими базами даних, що має можливість гнучкого масштабування, доступність, за рахунок розгортання веб-застосунку, і зручність у використанні.
- Запропоновані пропозиції покращення існуючих рінешь.

Публікації: підготовлені тези для сьомої Східно-Європейської конференції на тему «Дослідження архітектури та управління розподіленими базами даних», що надруковані у збірнику матеріалів сьомої Східно-Європейської конференції, розділу «Математичні та програмні технології Internet of Everything» (22-23.12.2020, Київ).

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Кожен день мільйони користувачів інтернету взаємодіють з даними, щось додають, оновлюють, видаляють і тд. Коли ми заходимо у соціальну мережу, спілкуємося у чаті, переглядаємо новини або здійснюємо покупки у інтернеті. Всі ці дані мають десь зберігатись для подальшого використання і саме для цього використовують бази даних.

Бази даних представляють собою сховища, що зберігаються дані відповідно до моделі організації даних та описують взаємодію елементів, їх характеристики та правила доступу до інформації[1]. Зараз бази даних використовуються майже у будь-яких системах, оскільки вони здатні зберігати надзвичайно великі об’єми інформації та мають безліч засобів управління.

Використання однієї бази даних, може стати проблемною для швидкозростаючого бізнесу, оскільки кількість інформації буде зростати, ефективність взаємодії зменшуватись, а навантаження збільшуватись. Тоді використання централізованої системи зберігання даних стає невиправданим і гарним виходом у цій ситуації буде розгляд розподілених баз даних та їх систем управління.

1.1 Дослідження розподілених баз даних

Розподілені бази даних – це сукупність логічно взаємопов’язаних баз даних, розподілених у комп’ютерній мережі. Для управління такою системою використовують систему управління розподіленою базою даних, яка створює ілюзію роботи з цілісною БД, ніби дані зберігаються в одному місці.

Розподіл даних базується на основі механізмів фрагметації, що розбиває дані по фрагментам, або реплікації, що синхронізує дані у кількох базах даних. Для дотримання успішного виконання всіх команд як єдиного цілого, використовують механізми транзакцій. Загалом процес роботи з розподіленими системами є

складнішим, оскільки потребує наявність програмного забпечення, що доволить працювати з усіма вузлами і коректно виконувати операції з різними частинами даних.

1.1.1 Системи управління

Системи управління базами даних (СУБД) – це комплексне рішення представлене у вигляді програми, що здійснюють взаємодіють з базою даних та дозволяє керувати даними через спеціальний інтрефейс. Використання таких систем може бути представленим у вигляді будь-якого окремого інтрефейсу або як консольний застосунок. При цьому, важливо щоб користувач мав повноту інформаційного забезпечення і міг виконувати весь ряд дій з даними.

Для керування розподіленою структурою було розроблено програмне рішення, яке називають системами управління розподіленими базами даних. Такі системи поділяють на однорідні (гомогенні), якщо на кожному вузлі мережі використовуються однові СУБД та неоднорідні (гетерогенні), що є протилежними до представлених. За ранунок використання стандартизованих механізмів доступу до БД відмінності між представленими типами стають не значними[2]. На ринку розробників баз даних вже активно розвиваються і поширюються рішення, що пов’язані зі створення проміжного етапу між сервісами БД для транляції запитів.

1.1.1.1 Архітектура

Системи розподілених СУБД мають ряд різних архітектурних рішень, що демонструють різні способи організації складових компонентів системи. Розглянемо різновиди архітектури розподілених СУБД[2]:

- Архітектура однорангової мережі

  Дана архітектура передбачає що всіх комп’ютерах мережі, що являють собою сервера, розміщені СУБД і база даних, при цьому з кожного пристрою можливо відправити запит на інший для отпримання даних (рис. 1.1).

- Архітектура з багатьма незалежними серверами

  Ця архітектура потребує щоб у користувачів системи зберігалась інформація про дані та розташування їх серверів. Також має бути присутній спеціальний додаток що надасть можливість здійснювати розділені запити та збирати їх у одну відповіть.

- Архітектура із взаємодіючими серверами

  Дана архітектура передбачає що кожен сервер містить всю інформацію про дані та розташування їх серверів у мережі, і може оброблювати розподілені запити.

- Клієнт-серверна архітектура

  Не передбачає мережевого розподілу даних, при цьому, існує тільки один сервер і довільна кількість користувачів, які взаємодіють з серверною (централізованою) базою даних через спеціальні додатки та мають віддалену СУБД на сервері.

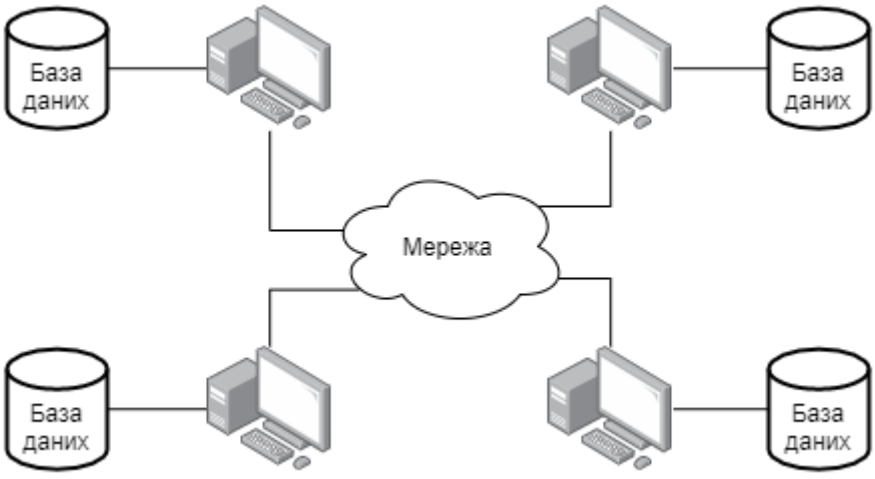

Рис. 1.1 Архітектура однорангової мережі

1.1.1.2 Переваги та недоліки

Системи управління розроділеними базами даних володіють рядом переваг перед централізованими рішеннями, але також мають і свої недоліки. Почнемо з розгляду переваг до яких відносяться[3]:

- Відображення структури організації

Багато компаній та організацій мають свої філіали або підрозділи, що можуть розташовуватись як у межах країни, так і за кордоном. Отже кожен підрозділ буде мати доступ до свого вузла і виконувати локальні запити. При цьому, керівництво зможе слідкувати і здійснювати глобальні запити при необхідності. При такій організації бізнесу, розподілені системи управління стають зручнішими у порівнянні з централізованими рішеннями.

- Роздільність і локальна автономність

Користувачі, що зареєстровані на одному сайті або сервісі, зможуть отримувати доступ до даних, що зберігаються на інших сайтах або сервісах, і встановлювати локальні обмеження на їх використання.

- Підвищення доступності даних

Системи управління РБД проектуються з урахуванням забезпечення відмовостійкості. Коли ми працюємо з централізованою СУБД, відмова головного серверу одразу припинить роботу усієї системи управління. Однак, при використанні систем управління РБД припинення роботи одного з серверів приведе тільки до втрати доступу до частини даних і забезпечить подальше функціонуваннння системи.

- Підвищення надійності

Системи, що реалізують реплікацію даних, можуть забезпечити повноцінне функціонування системи, навіть в ситаціях, коли один з вузлів припинив працювати. Запит перенаправляється на інший сервері дані, що були скопійовані з вимкненого вузла, будуть повернені користувачеві.

- Підвищення продуктивності

Система управління здатна працювати з навантаженим середовищем та великим рівнем паралельності. Це долволяє підвищити швидкість доступу до даних БД за рахунок отримання частин даних і запобіганням роботи з усією вибіркою.

- Економічна вигода

Практика показала, що використання невеликих вузьконаправлених серверів та робочих станцій є значно дешевним варіантом ніж робота з великими серверами. Це дозволить зменшити витрати на купівлю або оренду дорогих серверів.

- Модульність системи

Відмінною особливісню системи управління РБД від централізованого рішення є те, що її набагато легче розширювати. Додання нового вузла у мережу не впливає на роботу інших серверів і функціонування системи в цілому. При збільшенні навантаження можливо без будь-яких проблем додати нових обчислювальних потужностей і пристроїв дискової пам'яті.

Отже, тепер розглянемо ряд неділоків[4]:

- Підвищення складності

Дані системи мають безліч переваг, але, при цьому, є більш складними програмними комплексами. Також слід зазначити що дані можуть бути репліковані, що збільшує об’єм інформації у два рази і портебує постійної синхронізації даних між вузлами мережі. У випадку, якщо реплікація даних не буде підтримуватися на необхідному рівні, доступність, повнота та надійність даних системи знизиться.

- Підвищення ціни

Чим складніша система управління, тим більші затрати будуть очікувати. Організація системи, її розгортання та покупка обладнання потребують значної фінансової спроможності. Навантаження на трафік буде зростати за рахунок масштабування та постійної передачі даних по каналам мережі. Також необхідно

здійснювати оплату роботи працівників, що покращують, оптимізують та перевіряють цу систему.

- Проблеми захисту

Доступ до даних у централізованих систем досить легко контролювати, що не можна сказати про розподілені системи управління. Звертуючись до певного набору даних, що розташовані у різних вузлах, ми маємо забезпечити можливість захищенного доступу до серверів по всім каналам мережі.

- Ускладнення контролю за цілісністю даних

Під поняттям цілісності бази даних розуміється узгодженість та коректність зберігаємих у ній даних. Загалом забезпечення цілісності передбачає дотримання певних обмежень, що гарантує захист інформації в БД. Вони потребують доступ до великих об’ємів даних, що передбачає навантаження на канали мережі. А це вплине на вартість трафіку, що перешкоджає організації ефективного захисту.

1.1.2 Розподіл даних за допомогою фрагментації

Фрагментація призначена для ділення об’єкту на логічні фрагменти (також їх можуть називати сегментами) та їх розподіл і зберігання на певних вузлах мережі[4].

Основними причинами необхідності проведення фрагментації є:

- Ефективність

  Дані які використовуються частіше будуть зберігатись в більш доступних місцях, що надає можливість ефективнішого доступу до них.

- Паралельність

  Забезпечую розпаралелювання запиту і надає можливість отримати дані тільки с необхідних вузлів системи.

- Захищеність

  Отримання певних фрагментів залежить від можливості доступу до них. Якщо відповідний користувач не володіє правами, тоді він не зможе отримати необхідні дані.

Існує три види фрагментації даних:

- Горизонтальна
- Вертикальна
- Змішана

Горизонтальна фрагментація

Розбиття таблиці відбувається за рахунок виокремлення унікальних груп рядків по первній умові або набору умов. Загалов відбувається перенесення певної логічної групи рядків в таблицю на інший вузол системи. При цьому кожен фрагмент системи знаходиться на окремому вузлі та має набір унікальних значень, і всі рядки таблиці зберігають свою структуру, тобто набір стовпців.

Розглянемо приклад горизонтального фрагментування на таблиці продуктів (рис. 1.2). Існує таблиця з набором товарів та список правил за якими відбувається фрагментація. Після успішного виконання ми отримуємо три таблиці, кожна з яких знаходиться на окремому вузлі мережі і представлена в окремій БД.

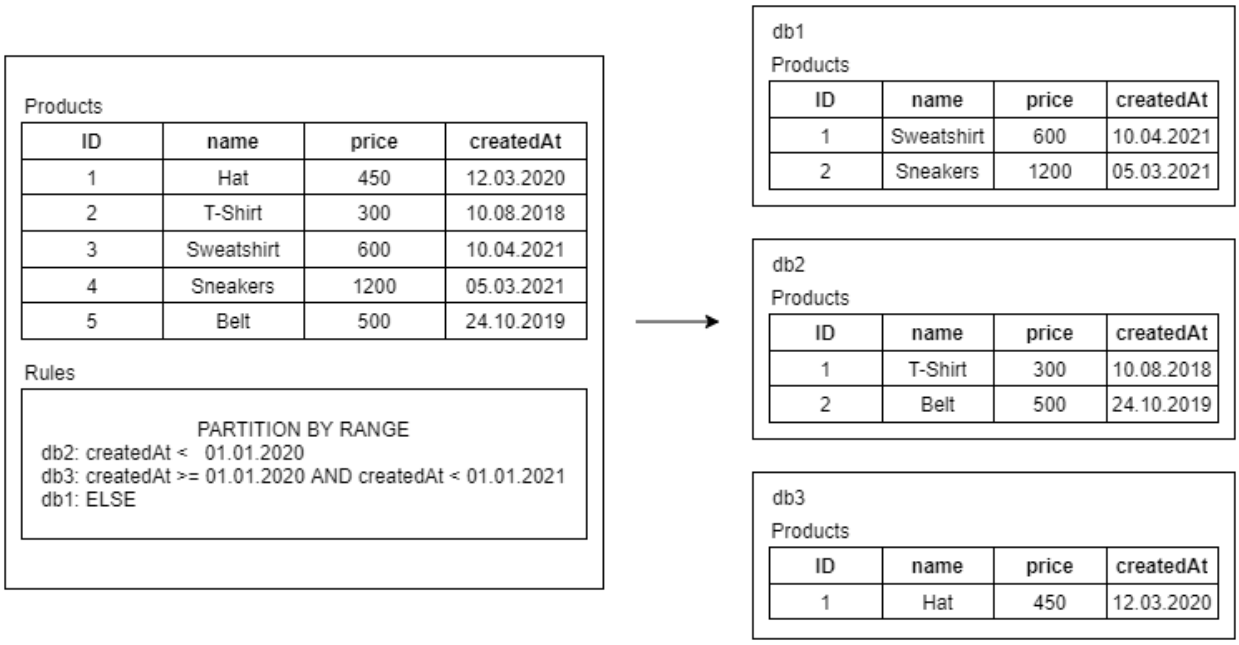

Рис. 1.2 Горизонтальна фрагментація

Вертикальна фрагментація

Розбиття таблиці відбувається за рахунок виокремлення підмножини стовпців. Таким чином набір стовців з даними розділяється між вузлами системи і представляє унікальні значення за виключенням ключового стовпця, який є у всіх фрагментах.

Розглянемо приклад вертикального фрагментування на таблиці продуктів (рис. 1.3). Отже, запропонуємо варіант компанії у якій існує два відділи, один займається організацією товарі, а інший формуванням цін. У такому випадку кожному з відділів не потріна надмірна інформація, тому досить зручно було би зберігати дані про товар у різних таблицях. Це допоможе швидше здійснювати пошук та оптимізує процес роботи з БД.

В результаті було проведо вертикальну фрагментацію яка допомогла розділити таблицю продуктів за сформованим набором правил. До таблці продуктів першої БД потрапили атрибути ID (ключовий атрибут), name та createAt. У свою чергу, в другу потрапили атрибути ID і пов’язані з ціною товару price та discount.

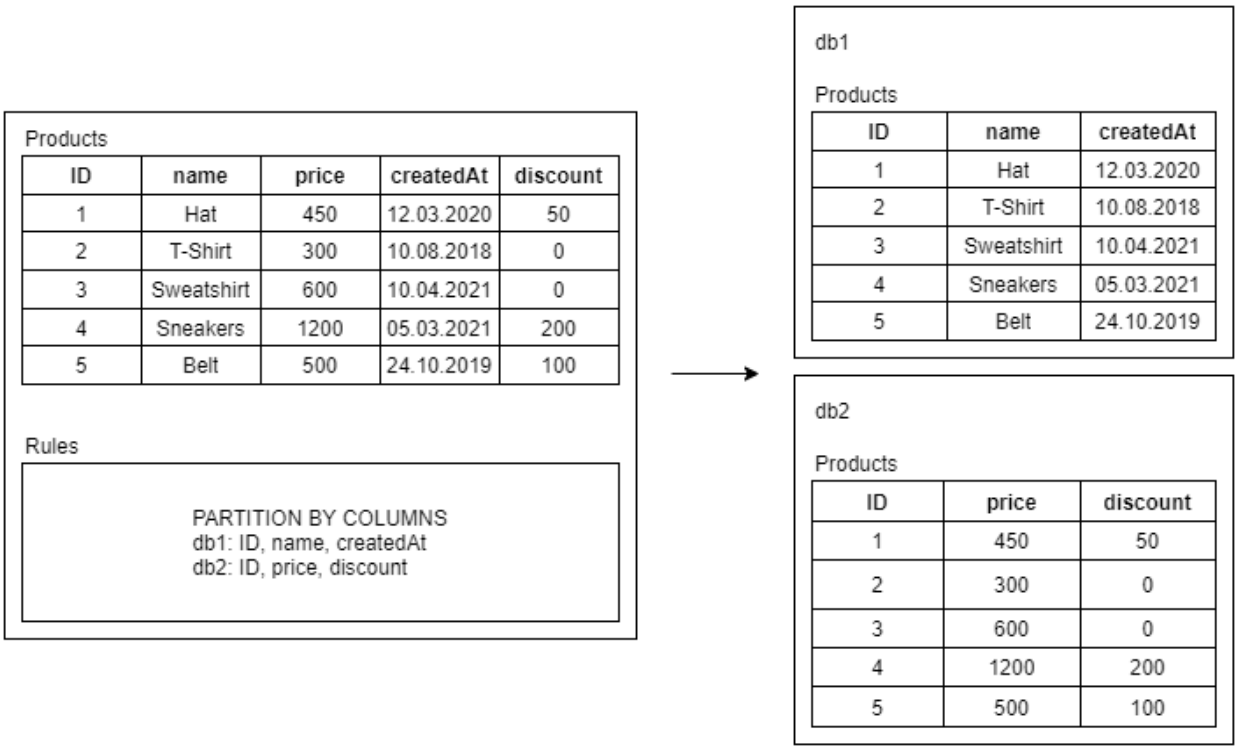

Рис. 1.3 Вертикальна фрагментація

Змішана фрагментація

Змішана фрагментація представляє поєднання вертикальної і горизонтальної фрагментацій. Тобто таблицю можливо поділити на кілька вертикальних стовпців, а потом на довільну кількість атрибутів, в залежності від поставлених задач.

1.1.3 Синхронізація доступу до даних через транзакції

Транзакція – це група впорядкованих операцій з БД, яка є логічною одиницею роботи з даними, що може бути цілком підтверджена або скасована. Вона є надзвичайно важливим інструментом регулювання одночасного доступу до даних під час запитів. Для роботи з транзакціями використовують спеціальні транзакційні системи, які в процесі роботи ведуть історію транзакцій.

Транзакції використовують для таких цілей:

- Забезпечення надійності роботи сиситеми і можливості відновлення усіх складових у разі виникнення проблеми або наявності незавершеності операцій з БД.
- Для забезпечення ізольованості та роздільного доступу до БД.

Транзакції і транзакційні системи мають набір вимог ACID. Вони були сформульовані Джимом Ґреєм наприкінці 70-х років, і до них відносять[5]:

- Атомарність (Atomicity)
  Атомарність гарантує, що ніяка транзакція не буде зафіксована в системі частково. Будуть або виконані всі її складові операції, або не виконано жодної.
- Узгодженість (Consistency)
  У відповідності до цієї вимоги, система знаходиться в узгодженому стані до початку транзакції і повинна залишитись в узгодженому стані після завершення транзакції.
- Ізольованість (Isolation)
  Під час виконання певної транзакції, транзакції які відбуваються паралельно, не повинні впливати на її результат.
- Надійність (Durability)
  Незалежно від проблем на нижніх рівнях, зміни, зроблені транзакцією, яка успішно завершена, повинні залишитись збереженими після повернення системи до роботи.

1.2 Огляд існуючих рішень

Зараз багато лідерів на ринку СУБД пропонують для системи управління розподіленими базами даних, але всі ці рішення частково підтримують побудову масштабних неоднорідних систем. Розглянемо рішення що є частинами великих проектів, так і особистими засобами упраління РБД.

У 70-80-х роках серед науково-дослідницьких робіт того часу потрібно згадати про систему SDD-1, що була розроблена фірмою Computer Corporation of America, систему R*, що була розроблена фірмою IBM і є розподіленою версією продукту System R, а також, систему Distributed INGRES, що була розроблена у Каліфорнійському університеті в Берклі і є розподіленою версією продукту INGRES[6].

Зараз більшість реляційних систем володіють різними видами підтрики використання РБД і рівнями функціональної інтеграції. Розглянемо найбільш відомі приклади:

- Система Oracle компанії Oracle Corporation.

Розподілена система баз даних дозволяє програмам отримувати доступ до даних з локальних та віддалених баз даних. В однорідній системі розподілених баз даних кожна база даних є базою даних Oracle, а у гетерогенній системі принаймні одна з баз даних не є базою даних Oracle. Розподілені бази даних використовують архітектуру клієнт/сервер для обробки інформаційних запитів[7].

У версії системі управління БД Oracle 7 додано безліч функцій для роботи з РБД: інструмент читання та оновлення даних декількох вузлів в межах однієї транзакції, підтримку більшості мереживих протоколів, засіб оптимізації розподілених запитів, а також можливість обміну даними з іншими СУБД.

- Система INGRES/STAR компанії The ASK Group Inc..

Систему передбачає можливість включення кількох апаратних та операційних систем та системи управління базами даних. За допомогою цього продукту можна об’єднати безліч окремих БД в єдине представлення даних, до яких можна отримати доступ так само, як і до будь-якої окремої локальної бази даних[8].

До переваг системи INGRES/STAR водносяться: можливість роботи і виконання ряду операцій в межаї однієї транзакції, оптимізація виконання

розподілених запитів та можливість видалення даних одразу з декількох вузлів мережі.

- Модуль DB2 фірми IBM.

IBM постійно розширює та вдосконалює свій продукт і додавання функцій розподіленої бази даних не стало виключенням. Компанія представила свій підхід архітектури розподіленої реляційної бази даних (DRDA) у версії 12.10. Вона предсталяє набір протоколів, що забезпечують зв'язок між програмами та системами баз даних на різних платформах, а також дозволяє розподіляти реляційні дані між кількома платформами[9].

1.2.1 Шляхи вдосконалення

Виходячи з розглянутих рішень, можливо підкреслити потребу у простому засобі управління РБД, що не буде представляти частину складної ситеми як пакет чи модуль. Користувач зможе швидко налаштувати засіб під власні потреби і мати можливість розподілу та зберігання даних у кілька кроків. Застосунок має володіти набором базових операцій для роботи з даними і забезпечувати надійність та стабільність роботи усіх складових компонентів системи.

Ще однією пропозицією є розробка масштабованої моделі, де сервери баз даних будуть мати одну або кілька конфігурацій розгортання, і користувачі зможуть налаштовувати вузли на основі підготовлених шаблонів з конфігурацією і рядом можливостей. Таким чином, сервер не буде мати надлишкового функціоналу, що зможе підвищити швидкість роботи системи.

1.3 Висновки до розділу

Підсумовуючи, був проведений аналіз предметної області і теоретичних відомостей, що дозволило сформувати проблематику і визначити напрямок розвитку даної теми. У процесі аналізу були розглянуті базові поняття баз даних, систем

управління та концепції розподілу даних. Ці ключові поняття стали основою теми для дослідження розподілених систем баз даних, їх управління та різновидів архітектури. Розглянуті способи розподілу даних через механізм фрагметації та синхронізація доступу до даних через транзакції. Це дозволило розібратись, яким чином має функціонувати засіб і як ми можемо розподіляти дані, щоб надалі було зручно з ними працювати.

Проаналізовані існуючі аналоги систем управління розподіленими базами даних, такі як система Oracle, INGRES/STAR та модуль DB2 компанії IBM, та були запропоновані шляхи вдосконалення.

На основі розглянутих даних була сформована абстрактна модель системи, що включає реалізацію архітектури з багатьма незалежними серверами.

РОЗДІЛ 2
ПРОЕКТУВАННЯ МОДЕЛІ СИСТЕМИ

2.1 Архітектура та компоненти системи

Основною метою роботи є розробка власної моделі системи, що реалізує архітектуру з багатьма незалежними серверами. Таким чином, була сформована модель системи, що складається з трьох основних компонентів: серверу управління запитами, серверів баз даних та клієнтського інтерфейсу (рис. 2.1).

Сервер управління запитами надає доступ до особистої панелі користувача і здійснює керування розподіленою системою БД через API. Кожен користувач має можливість самостійно створювати та розподіляти дані між існуючими серверами баз даних.

Сервери баз даних представляють набір незалежних вузлів мережі, що володіють API, яке дозволяє створювати, видаляти та здійснювати взаємодію з таблицями баз даних що підключені до них.

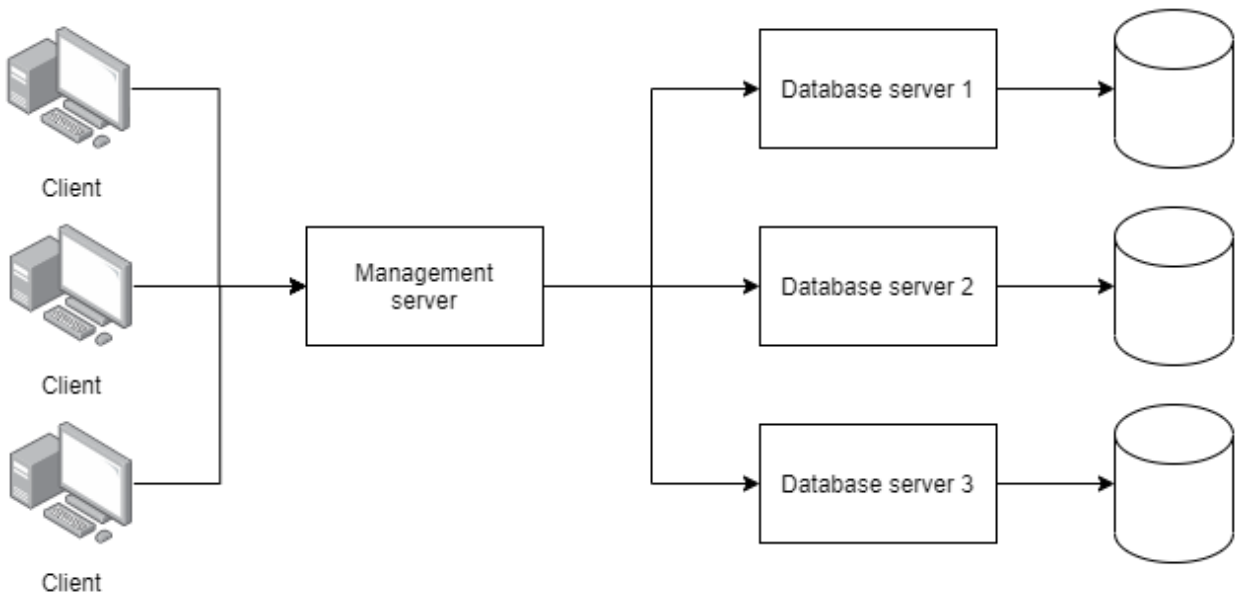

Рис. 2.1 Архітектура програмного забезпечення

Такий підхід побудови системи забезпечує можливость її масштабування за рахунок можливості додавати нові вузли. Зробити це можливо на сторінці списку вузлів, на якій знаходиться посилання на форму підключення нового вузла. Коли додається новий вузол, система перевіряє статус його акивності і відображає інформацію про новий сервер бази даних, у разі якщо він запущений, або ні, у іншому випадку.

Один вузол представляє окремий сервер, що підключений до однієї конкретної БД. Він може розміщуватись у будь-якій частині мережі, але має володіти необхідною конфігурацією і дотримуватись ряду обмежень, що визначені системою. До таких обмежень належить дотримання принципів REST архітектури та шаблонів запитів при роботі з API.

Після того, як користавач додав всі необхідні вузли, він може створити нову таблицю та працювати з нею. Кожна таблиця володіє своїм набором правил, що описуюсь розташування її даних у розподіленій мережі.

2.1.1 Діаграми прецедентів, розгортання та послідоності

Для більш детального опису моделі системи стане доречним використати мову UML та побудувати ряд діаграм: розгортання, компонентів та прецедентів.

UML – це уніфікована мова моделювання, що є складовою частиною процесу розробки ПЗ. UML використовується для візуалізації, проектування та документування програмних складових системи. Використання даної мови допомагає у формуванні абстрактної моделі системи, яку прийнято називати UML-модель[10].

Почнемо за огляду системи з точки зору користувача і для цього буде використана діаграма прецедентів або, які її ще називають, діаграма варіантів використання (рис. 2.2). Діаграма прецедентів – це UML діаграма, що описує відношення між акторами та прецедентами системи[10].

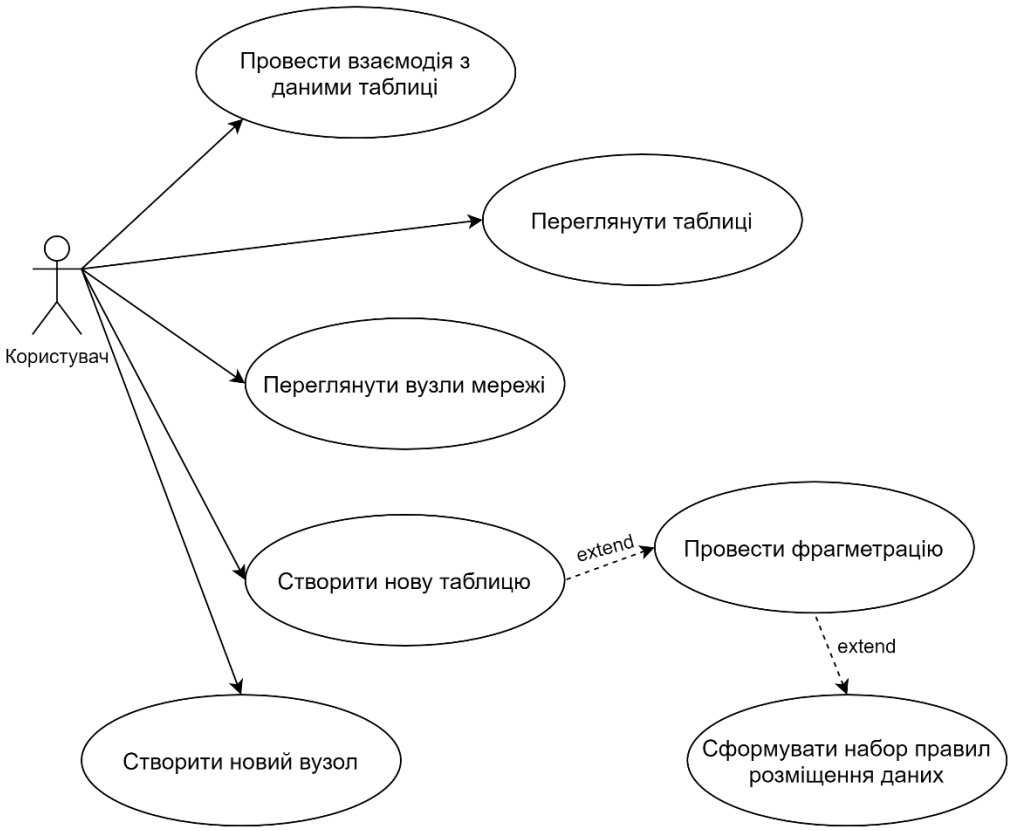

Рис. 2.2 Діаграма прецедентів

На діаграмі зображений один суб’єкт, що представляє користувача системи управління, який може виконувати ряд таких подій: переглядати таблиці, переглядати вузли мережі, проводити взаємодію з даними таблиці і створити нові таблиці та вули.

Для розгляду складових компонентів системи, їх зв’язків та внутрішніх об’єктів, візуалізуємо цю модель у вигляід діаграми розгортання (рис. 2.3). Діаграма розгортання – це UML діаграма, що відображає обчислювальні вузли під час роботи програми, їх компоненти та об’єкти[10].

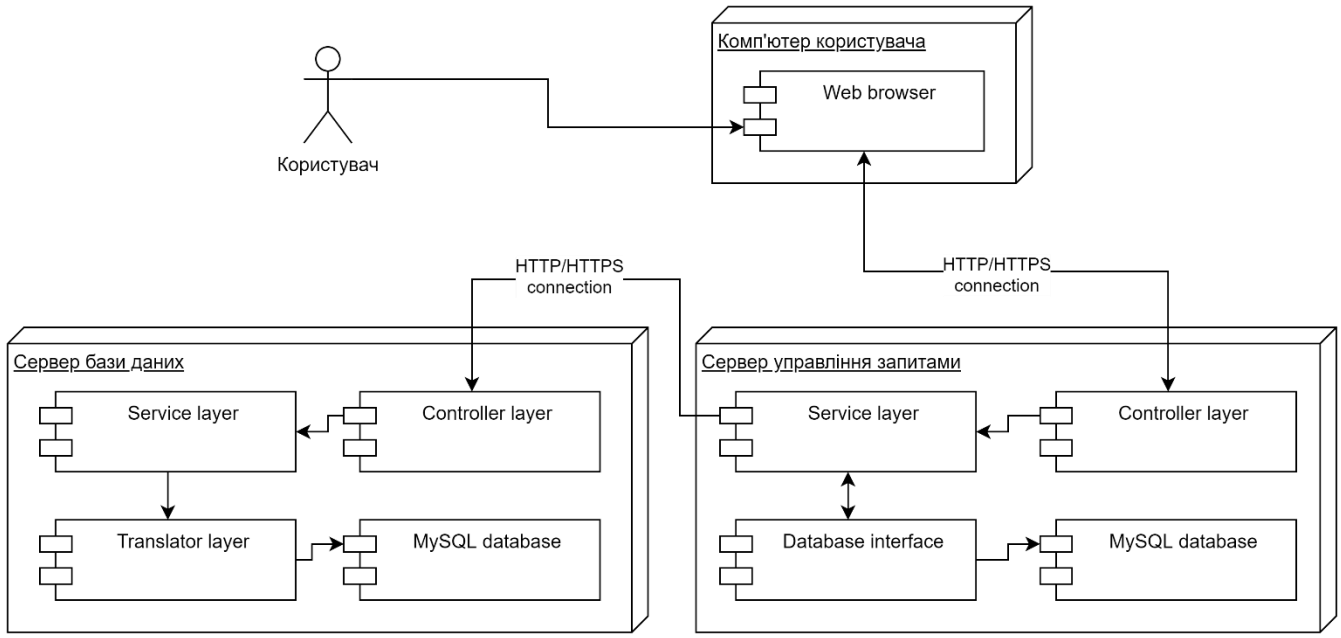

Рисунок 2.3 Діаграма розгортання

Використовуючи веб-браузер користувач здійснює HTTP/HTTPS запит на сервер управління запитами та отримає сторінку застосунку. Сервер складається з чотирьох об’єктів: рівню контролера (Controller layer), сервісного рівню (Service layer), інтерфейсу бази даних (Database interface) та БД MySQL (MySQL Database).

Для доступу до даних вузлів, рівень сервісу виконує HTTP/HTTPS запити до серверів БД, які містять наступні об’єкти: рівень контролера (Controller layer), сервісний рівнень (Service layer), рівень транслятора (Translator layer) та БД MySQL (MySQL Database).

На даному етапі, формування моделі, ми вже маємо абстрактне представлення структури системи і розуміння, які дії може виконувати користувач. Розглянемо послідовність дій для створення нової таблиці за допомогою діаграми послідовності (рис. 2.4). Діаграма послідовності – це UML діаграма, що відображає взаємодію об’єктів впорядковану на проміжку за часом виконання[10].

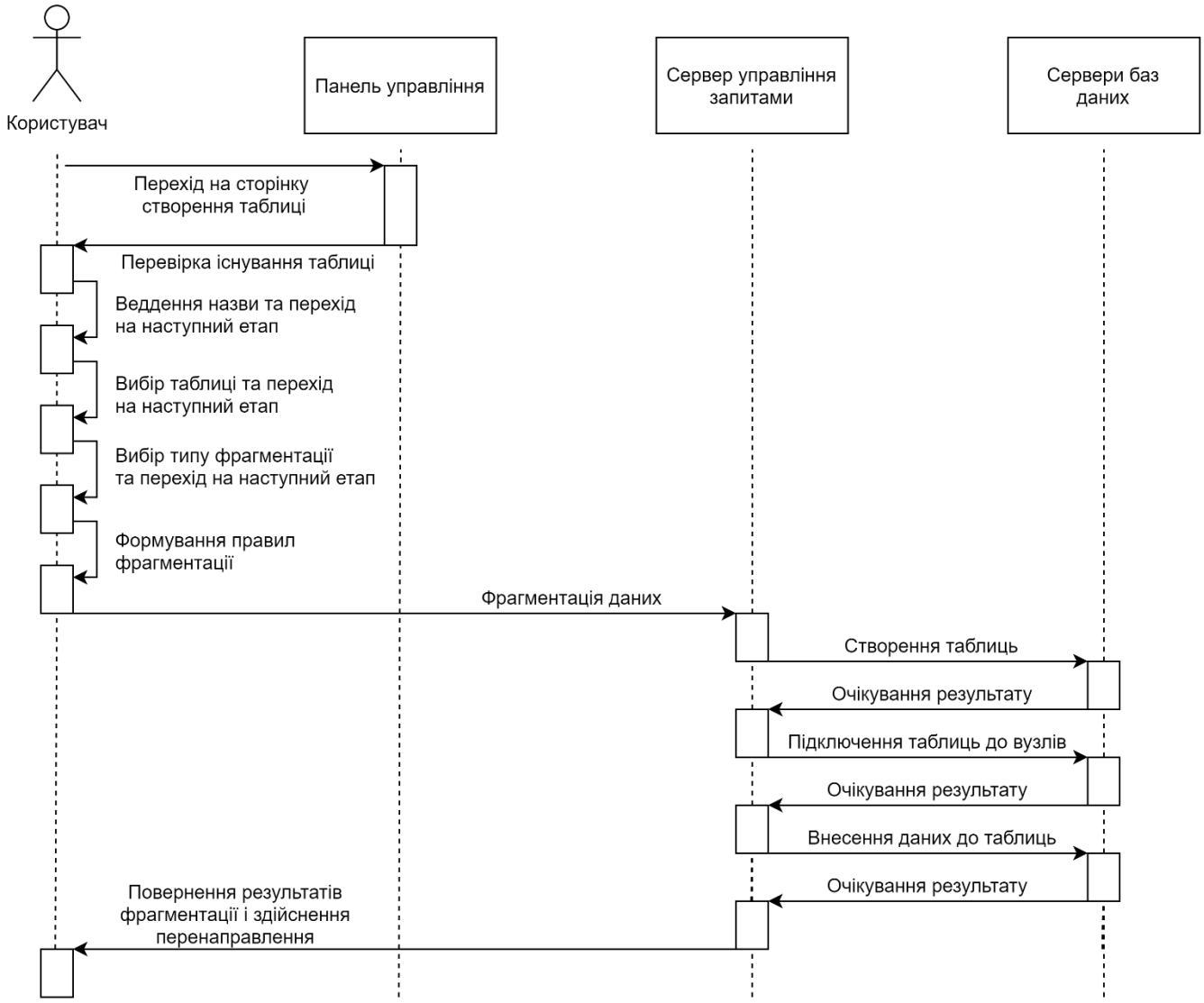

Рисунок 2.4 Діаграма послідовносні створення нової таблиці

Як ми бачимо з ринунку 2.4, діаграма розбита на чотири окремі часові проміжки, кожен з яких характеризує певну складову частину системи управління РБД. Створення таблиці починається з переходу на відповідну сторінку і містить кілька кроків виконання. У разі, якщо створення було перервано і користувач покинув сторінку, він може повернутись до кроку на якому запунився одразу після переходу на сторінку створення. Цей видапок був передбачений для забезпечення зручного використання системи та уникнення проблем.

2.2 Функціональні та нефункціональні вимоги

Систума управління РБД повинна задовольняти наступним функціональним вимогам:

- Здійснення заємодії з РБД.
- Створення нових вузлів та таблиць.
- Здійснення фрагментації даних.
- Обрабка помилок при виконанні подій.
- Виведення звіту у журналі подій.
- Взаємодія з таблицями.

До нефункціональних вимог відносяться:

- Надійність – забезпечення надійного обміну даними під час здійснення запитів у мережі.
- Цілісність – забезпечення стабільного доступу до всіх елементів системи.
- Масштабованість – можливість розширити систему без потреби її зупинки.
- Ефективність – забезпечення швидкого обміну даними та відсуності навантажешь.
- Швидкість – швидке виконання всіх процесів між складовими компонентами системи.
- Зручність – зрозумілість всіх елементів інтерфейсу при роботі з системою управління.
- Функціонально повнота – надання користувачуві всіх доступих функцій для зручної роботи.

2.3 Фрагментація та правила розміщення даних

Фрагментування використовується для того, щоб розділити дані таблиці, яку ми створюємо, між доступними вузлами мережі. Модель системи передбачає реалізацію двух видів фрагментації: вертикальну та горизонтальну. Коли користувач обирає вид фрагментації, він потрапляє на відповідну сторінку з таблицею та формує правила, за якими відбуватиметься розподіл даних.

У базі даних серверу управління запитами створюємо дві таблиці, кожна я яких буде зберігати правила фрагментації для кожної РБД. Таблиця правил вертикальної фрагментації містить такі поля:

- id – унікальний ідентифікатор;
- node_table_id – ідентифікатор, що вказує на таблицю у певному вузлі;
- attribute – атрибут, який має бути розміщений на даному вузлі.

Таблиця правил горизонтальної фрагметації містить такі поля:

- id – унікальний ідентифікатор;
- node_table_id – ідентифікатор, що вказує на таблицю у певному вузлі;
- attribute – атрибут, який описує правило;
- operator – умова поділу по даному атрибуту;
- value – значення поділу по даному атрибуту та оператору.

Розглянемо формування правил для кожного з видів фрагментації:

- Вертикальна фрагметація

У даному виді таблиця розбивається по атрибутам, тому сервер має отримати список об’єктів, у якому кожен об’єкт містить ключ вузла та назву самого атрибуту.

- Горизонтальна фрагментація

У даному виді таблиця розбивається по умовам атрибутів на групи записів. Наприклад, формуються списки правил для кожного вузла, до яких користувач має

змогу додавати власні правила, після чого йому пропонується обрати атрибут, умову та значення умови.

Для візуального відображення результатів розбиття, після кожного додавання правила, відбувається оновлення таблиці і поля, які підпадають під правила, позначаються відповідним кольором вузла. Таким чином на сервер відправляється об’єкт, який містить списки правил по кожному вузлу. Після цього, відповідно до правил, відбувається створення таблиць та збереження даних на вузлах. Кожен крок виконання супроводжується відмітками у журналі подій, щоб користувач міг перевірити коректність виконання та дізнатись на якому кроці виникла помилка, у разі проблем.

2.4 Оброка помилок та відкат системи

Процес фрагментації та взаємодія з розподіленою системою баз даних передбачає роботу в багатьма вузлами у мережі і їх базами даних. У такому випадку досить великої уваги набуває обробка помилкових ситуацій та можливість відновлення попереднього стану системи.

Орієнтуючись на теоретичні відомості, виникла ідея розробити власний модуль (рис. 2.5), який реалізує частовий функціонал транзакцій і дозволяє роботи відкат у всіх вузлах, під час роботи з таблицею. Основна ідея полягає у створенні стеку подій, в який будуть записуватись функції, які оброблюють обернені події до тих, що ми виконуємо.

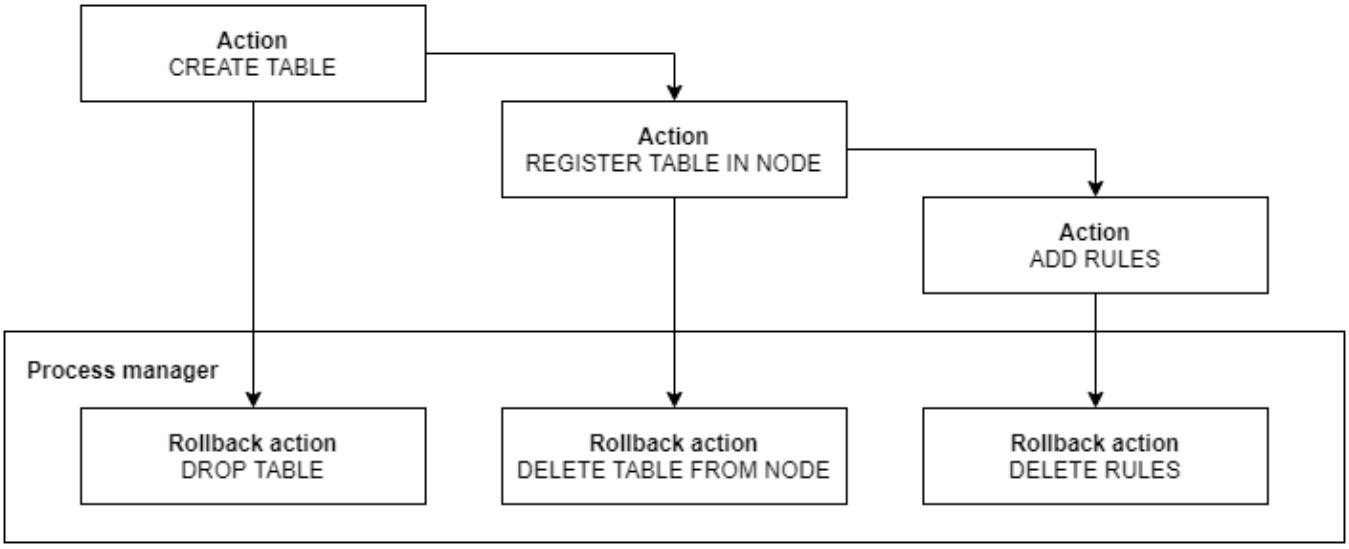

Рис. 2.5 Модуль управління процесом

Список операцій відкату може вміщувати безліч подій, які представляють собою функції. Початок виконання та завершення роботи модуля управління процесом має бути викликаний відповідними функціями до та після частини коду, що реєструють та виконують відкат системи.

Коли у процесі виконнання програми відбувається збій, що може бути спричинений як проблемами доступу до серверу бази даних, так і іншими помилками, модуль управління процесом запускатиме відкат системи і виконуватимуться усі операції відкату зі стеку. Таким чином вдастся запобігти проблем, коли користувач намагається оновити таблицю, чи провести фрагментацію, тоді цілісність синхронізації доступу до вузлів має бути повноцінною і не відбуватиметься створення нових даних на частині серверів мережі.

Менеджер процесу використовується у всіх ключових точках роботи з даними таблиць або вузлів, до таких відносять обробники муршрутів, сервіси роботи з даними, моделі даних та окремі модулі.

2.5 Структура бази даних

Сервер управління запитами підключений до головної бази даних, що містить інформацію про кожного користувача, його вузли у мережі та таблиці. Розглянемо кожну таблицю та її атрибути.

Таблиця user представляє список доступних користувачі системи, які можуть входити у власну панель та здійнювати роботу з таблицями через розподілену систему БД. Таблиця містить наступні атрибути:

- id – унікальний ідентифікатор користувача;
- firstName – ім’я користувача;
- lastName – прізвище користувача;
- email – поштова адреса користувача;
- password – пороль користувача (у зашифрованому форматі).

Таблиця table містить набір всіх таблиць користувачів розподіленої системи БД. Таблиця містить наступні атрибути:

- id – унікальний ідентифікатор таблиці;
- name – унікальна назва таблиці;
- label – назва таблиці, що вводиться користувачем і доступна при перегляді списку таблиць;
- status – статус таблиці;
- type – тип таблиці (может бути завантажена користувачем або обрана зі списку існуючих);
- fragmentationType – тип фрагментації (вертикальна або горизонтальна);
- user_id – ідентифікатор користувача, якому належить таблиця.

Таблиця node представляє список вузлів, до яких під’єднані користувачі системи. Таблиця містить наступні атрибути:

- key – унікальний ключ вузла;

- name – назва вузла;
- host – адреса сервера БД;
- port – порт сервера БД;
- user_id – ідентифікатор користувача, що підключений до вузла.

Наступна таблиця node_table встановляє зв’язок один до багатьох між таблицею та вузлами на яких вона знаходиться. Використовується у таблицях правил фрагментації, щоб вказати за допомогою яких правил дані були розміщені у відповідній таблиці на певному вузлі. Таблиця містить наступні атрибути:

- id – унікальний ідентифікатор таблиці та вузла;
- node_key – унікальний ключ вузла;
- table_id – унікальний ключ таблиці;

Остання таблиця log представляє список подій, що відбулись у системі при створенні або взаємодії з таблицями. Таблиця містить наступні атрибути:

- id – унікальний ідентифікатор події;
- name – назва події;
- createdAt – дата створення події;
- status – статус події (може приймати три значення: успіх, помилка, відкат).
- user_id – ідентифікатор користувача, який викликав подію.

База даних представлена у реляційній моделі, структури зберігання даних у яких приведені до нормальних форм, відповідно до перших трьох правил нормалізації. Дані не мають надлишкових залежностей, підтримують повну атомарність та володіють первинними і зовнішніми ключами. Повна структура БД продемонстрована у додатку А.

2.6 Висновки до розділу

У розділі було проведено проектування моделі системи, що використовує архітектуру з багатьма незалежними серверами. Були сформовані ключові компоненти системи і розроблений план функціонування кожної складової частини. Модель передбачає розподіл застосунку на три складові компоненти: клієнтська частина, сервер управління запитами та сервери баз даних.

Продемонстровані UML діаграми для опису подій, об’єктів та послідовностей виконання операцій, що дозволяють деталніше ознайомитись з функціонуванням кожного компонента на різних рівнях роботи системи. Окреслений план проведення розподілу даних, обробки помилок і визначена структура бази даних. Всі ці складові проаналізовані і представлені з розрахунком дотримання функціональних та нефункціональних вимог.

Тепер ми маємо уявлення про складові елементи системи, їх процеси та взаємодію з ними. Ці матеріали допоможуть нам почати розробку програмного застосунку і визначити набір технологій, за допомогою яких він буде створюватись. Засіб має здійснювати управління розподіленою базою даних і надавати користувачу можливість взаємодії з даними через інтерфейс, що створює ілюзію роботи з цілісною БД, ніби дані зберігаються в одному місці**.**

Отже, були сформована модель системи і підготовлені матеріали для розробки власного програмного застосунку.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ УПРАВЛІННЯ РБД

3.1 Засоби та інструменти розробки

Для розробки програмного забезпечення була обрана мова програмування JavaScript. Вона є досить поширеною і продовжує набирати популярність серед розробників. Має великий набір програмних рішень та інструментів що можуть допомоти у розробці будь-якого застосунку. Веб-інтерфейс для системи був обраний з розрахунком можливості кросплатформного доступу. Таким чином легше здійснювати розробку та орієнтуватись на ринок користувачів.

Для розробки клієнтського інтерфейсу була використана бібліотека React, яка дозволяє досить легко та зручно будувати компоненту структуру сторінки. Перевагою даного засобу є те, що він є досить швидким, простим, масштабованим і вирішує проблему часткового оновлення веб-сторінок[11].

Для реалізації серверної частини обрана платформа Node.js. Загалом це програмна платформа з відкритим вихідним кодом, що дає можливість виконувати скрипти написані на мові JavaScript на стороні сервера, за допомгою ядра V8 що вміє транслювати код написаний на JS у машинний код[12]. Її вибір зумовлений зручністю розгортання та досить високою продуктивністю.

У якості СУБД обрана MySQL, і її реалізація для JavaScript у вигляді модуля mysql2. Загалом це вільна система керування реляційними базами даних з відкритим вихідним кодом[13]. Її вибір зумовлений особливою організацією даних та наявністю вбудованих можливостей для реалізації процесу розподілення баз даних.

Для ведення розробки обрано інтегроване середовище WebStorm, фукціональний пакет інструментів DataGrip для взаємодії з базами даних та Postman для тестування запитів та розробки API.

3.2 Сервер бази даних

3.2.1 Підключення до бази даних

Відповідно до моделі системи, кожен вузол взаємодіє лише з однією базою даних. У якості БД використовується MySQL, а саме пакет мови JavaScript mysql2. Для підключення формується об’єкт конфігурації, який повинен мати такі обов’язкові дані:

- Адресу БД
- Назву БД
- Логін користувача
- Пароль користувача

Сам процес підключення представляє асинхрону операцію, що відбувається на сервері під чай його запуску. У параметрах запуску прописуються дані конфігурації БД та зчитуються модулем аналізу CLI. Якщо виникає помилка, тоді сервер переходить до обробника помилок, виводить помилку у консолі та запиняє процес роботи сервера.

3.2.2 Реалізація API

Сервер бази даних має набір кінцевих точок доступу по яким сервер управління взаємодіє з ним. Усі точки можна роділити на дві частини, перша здійснює роботу з таблицями та їх даними, а друга надає серверну інформацію та доступні дані конфігурації даного вузла, що продемонстровані у таблиці 3.1.

Таблиця 3.1

Доступні кінцеві точки

№	Назва	Метод	Запит
1	Вибірка даних таблиці	GET	/api/db/select
2	Створення таблиці	POST	/api/db/create-database
3	Вставка даних	POST	/api/db/insert-into
4	Оновлення даних	PUT	/api/db/update
5	Видалення даних	DELETE	/api/db/delete-from
6	Видалення таблиці	DELETE	/api/db/drop-table
7	Статус сервера	GET	/api/server/status
8	Конфігурація сервера	GET	/api/server/info

Кожна кінцева точка також має набір додаткових параметрів, що передаються у самому запиті або у його тілі. Отже, усі запити, що пов’язані з роботою з табляцями, мають отримати параметр tableName, що ідентифікує з якою таблицею буде відбуватись взаємодія. Якщо ми виконуємо операції створення чи оновлення, сервер БД має отримати модель даних у вигляді об’єкту зі всіма атрибутами. Також передчабена фільтрація по списку умов where для таких операцій як читання, оновлення та видалення даних.

3.2.3 Організація процесів

Організація процесів відбувається за рахунок прописаного набору кінцевих точок API. Для запиту до серверу використовується сформований набір шаблонів, які представляють JSON об’єкти з даними та умовами, які відправляються на вузлол, проходять через транслятор і формують SQL запит (рис. 3.1).

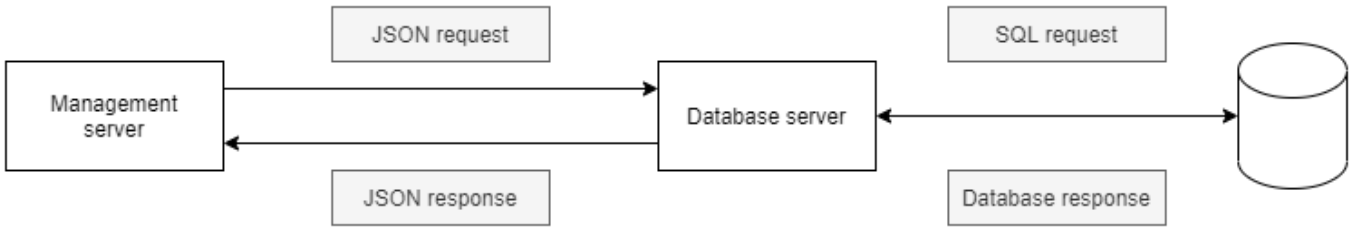

Рис. 3.1 Процес взаємодії з вузлом мережі

3.3 Сервер управління запитами

3.3.1 Роль та ораганізація процесів

Серевер виступає посердником у архітектурній моделі системи між клієнтом та вузлами мережі. Розглянемо кілька прикладів організації різних процесів.

Коли користувач заходить у власну панель, він отримує інформації від серверу управління, по мірі необхідності. Прямого доступу до даних таблиць користувача серевер не має, тому, щоб отримати всю необхідну інформацію, сервер управління запитами здійнює взаємодії з вузлами мережі через API. Після цього, сервер упраління робить злиття даних відповідно до правил фрагментації даної таблиці і повертає повноцінну версію таблиці користувачу. Таким чином відбувається органазація процесу отримання даних таблиці у розподіленій системі БД.

Також сервер забезпечує безпечний доступ до даних за рахунок використання підходу взаємодії з сервером через ключ доступу. Реалізовано за допомогою стандарту JWT, що передбачає генерацію ключів доступу та оновлення. При успішному вході у систему, з серверу відправляється ключ, що зберігається у локальному сховищі браузера та, при подальшому оновлені сторінки, відправляється на сервер для авторизації користувача. Таким чином відбувається організація процесу підтримання сессії користувачів системи.

3.3.2 Керування вузлами

Керування вузлами системи здійснюється на сторінці зі списком вузлів у особистій панелі користувача (рис. 3.2). Список представлений у вигляді набору об’єктів веб-інтерфейсу, що містить інформацію про вузли та дані БД до яких вони підключені.

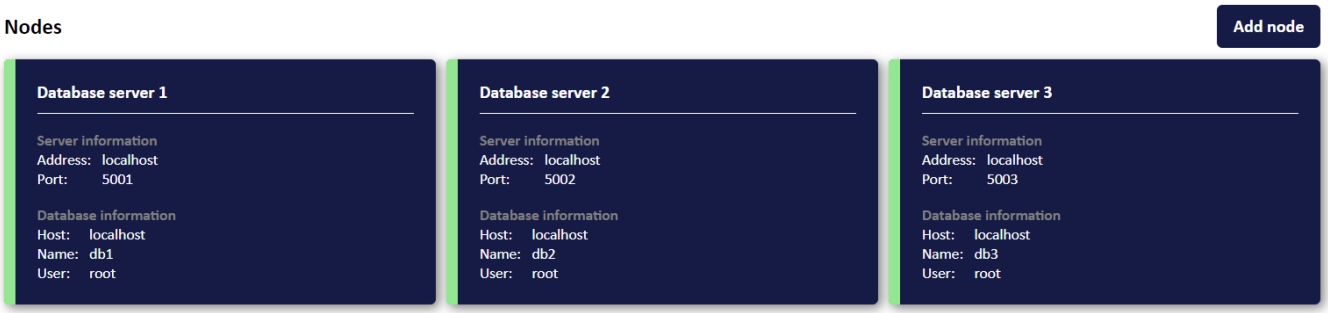

Рис. 3.2 Список вузлів користувача

Взаємодія з вузлами мережі передбачає створення, видалення та оновлення налаштувань. У разі виникнення проблем з сервером, система має попередити користувача про стан функціональної цілісності. Сервер може перебувати у трьох можливих станах:

- Активний – сервер готовий до роботи.
- Неактивний – сервер зупинений або виникла помилка.
- Налаштовується – сервер здійснює перевірку конфігурації.

Якщо вузол, до якого підключена таблиця, є некативних (рис. 3.3), доступу до даних цієї таблиці блокується (рис. 3.4) і користувач не зможе потрапити на її сторінку.

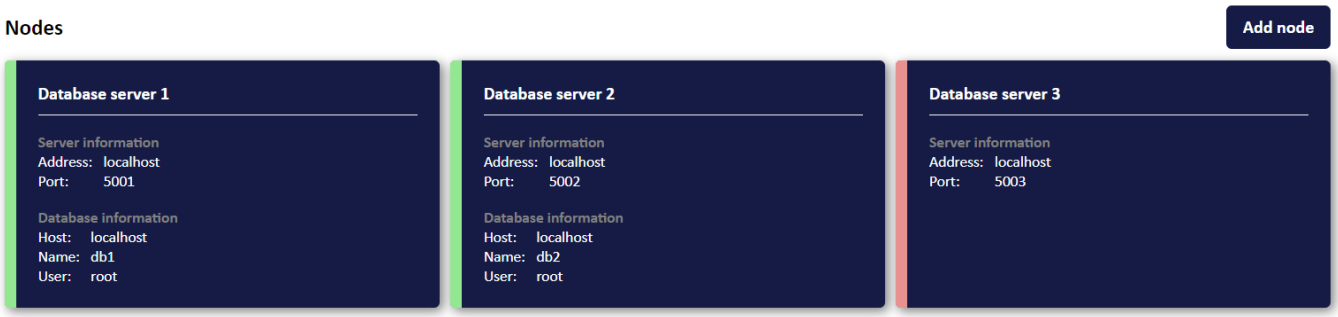

Рис. 3.3 Вимкнення одного з вузлів

Кожен вузол запускається з набором зазначених параметрів конфігурації до яких входять дані підключення до БД. Оскільки север знаходиться у неактивному стані, параметри підключення до бази даних відсутні. Такий підхід має як свої недоліки, так і переваги. Недоліком є те, що користувач не сможе зрозуміти до якої БД підключений сервер, поки він не стане активним, з іншого боку, це надасть можливість масштабування системи, оскільки всі сервера мають однакову конфігурацію і можуть підключатись та працювати з будь-якою БД.

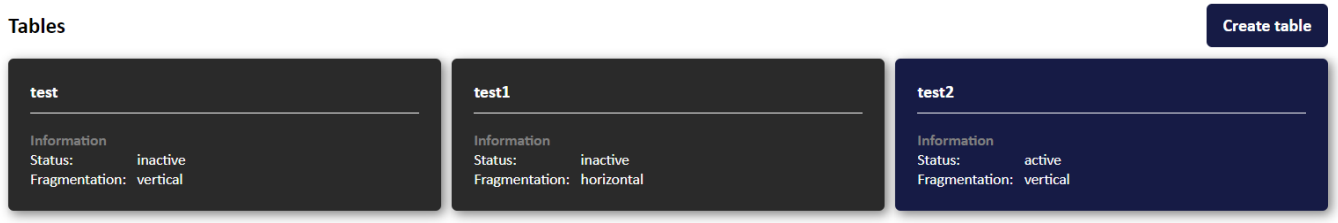

Рис. 3.4 Список активних та неактивних таблиць

3.3.3 Керування таблицями

Таблиці є однією з найважливіших одиниць системи. Модель системи базується на розподілі таблиць та їх даних по різним частинам мережі. Загалом передбачається що користувач зможе виконувати базовий набір команд взаємодії з таблицями, до таких відносяться:

- Читання запису

- Створення нового запису
- Оновлення існуючого запису
- Видалення запису

3.3.3.1 Створення нової таблиці

Створення таблиці відбувається на окремій сторінці, на яку користувач може потрапити зі списку таблиць, і де він має пройти наступні етапи:

- Введення назви таблиці (рис. 3.5)

Назва таблиці має бути унікальною у рамках таблиць користувача, тому присутня валідація даних. У разі виникнення помилок, користувачу виведеться повідомлення під полем вводу.

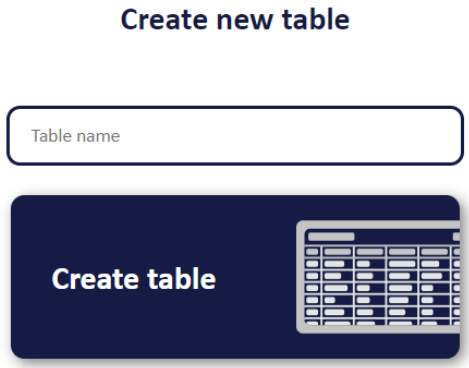

Рис. 3.5 Етап введення назви таблиці

- Вибір таблиці

Вибір передбачає завантаження власного файлу таблиці з розширенням .sql або вибір одного з існуючих шаблонів (рис. 3.6). Якщо користувач обирає завантаження файлу, то перед цим йому потрібно змінити назву таблиці на шаблон «[[tableName]] », оскільки система автоматично згенерує ключ таблиці.

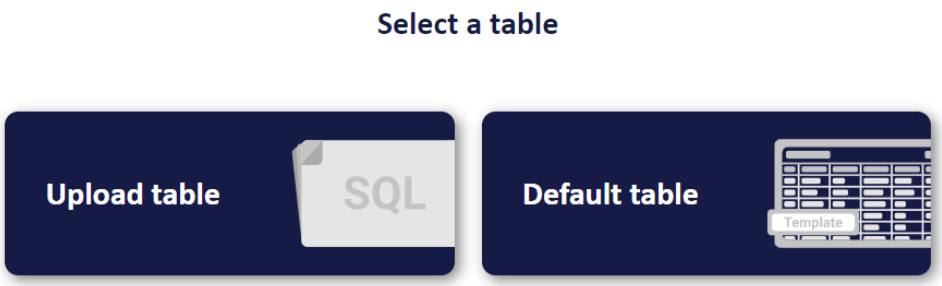

Рис. 3.6 Етап вибору таблиці

Серед існуючих шаблонів представлена таблиця продуктів, таблиця блогу та таблиця новин (рис. 3.7). Такі шаблони зазвичай можеть знадобитись для перевірки роботи системи, коли користувач не має готової таблиці для завантаження, але бажає спробувати попрацювати з панелю управління.

Рис. 3.7 Вибір існуючого шаблону таблиці

- Вибір типу фрагментації

Наступним етапом є вибір типу фрагментації. Як вже було зазначено, модель системи надає можливість вибору між вертикальною та горизонтальної фрагментаціями (рис. 3.8). В залежності від вибору, далі користувач потрапляє на одну зі сторінок, де передбачена панель інтсрументів для здійснення певного типу фрагментації.

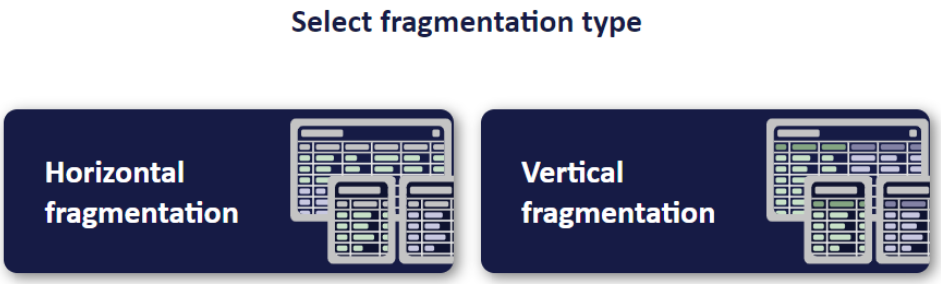

Рис. 3.8 Вибір типу фрагментації

- Фрагментація

Останнім етапом є здійснення розподілу даних шляхом проведення фрагментації. Розглянемо панель інструментів для кожного типу.

Отже, оскільки вертикальна фрагментація передбачає розподіл даних по атрибутам, тому для кожного атрибута у таблиці були додані списки северів БД (рис. 3.9), у яких користувач може обрати вузлок, на якому даний атрибут таблиці буде зберігатись. Обов’язковою умовою є використання двух або більше вузлів при розподілі, у іншому випадку користувач не зможе завешити створення таблиці.

id	name	createdAt	price	discount
	Database server 1	Database server 1	Database server 2	Database server 3
1	T-shirt	1619689268	200	0
2	Hat	1619659268	120	0
3	Sweatshirt	1619644268	250	62.5
4	Jeans	1619685268	400	0
5	Scarf	1619639268	180	54
6	Jacket	1619687268	500	50
7	Cap	1619689168	150	0
8	Backpack	1619791268	400	40
9	Wallet	1619659268	120	0
10	Sneakers	1619684268	450	90

< Confirm

Рис. 3.9 Сторінка вертикальної фрагментації

При горизонтальній фрагментації користувачу виводиться таблиця та панель зі списками правил, де виводяться активні вузли позначені унікальним кольором (рис. 3.10). При цьому, дані мають розділятись на групи по первних умовах атрибутів, тому була додана можливість створення правил фрагментації.

Рис. 3.10 Сторінка горизонтальної фрагментації

Коритувач може створити нове правило, натиснувши на плюс біля назви вузла, і йому відкриється модальне відко, у якому він має обрати атритуб, умовний оператор та значення умови розподілу (рис. 3.11). Після цього воно видеться у списку правил, а таблиця оновиться і відмітить відповідним кольором усі рядки, що підпадають під дане правило.

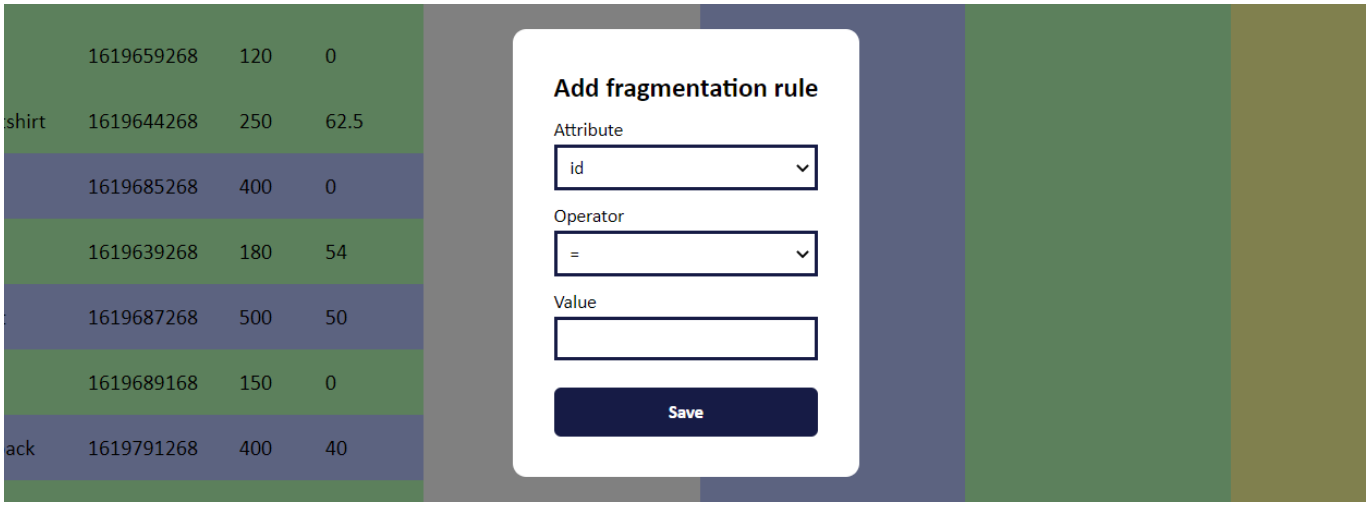

Рис. 3.11 Вікно створення нового правила фрагментації

Після виконання всіх кроків, користувач має побачити новий елемент у списку таблиць.

3.3.3.2 Реалізація взаємодії з даними

Після створення таблиці, користувач може переглядати та взаємодіяти з нею (рис. 3.12). У верхній частині знаходиться дві кнопки, перша «Create» відкриває модальне вікно, у якому пропонується заповнити форму та додати новий елемент до списку. Друга «Drop table» відкриває вінко, у якому просить користувача підтвердити видалення таблиці. Операції оновлення та видалення даних пов’язані безпосередньо з записами і розміщуються у кінці кожного рядка.

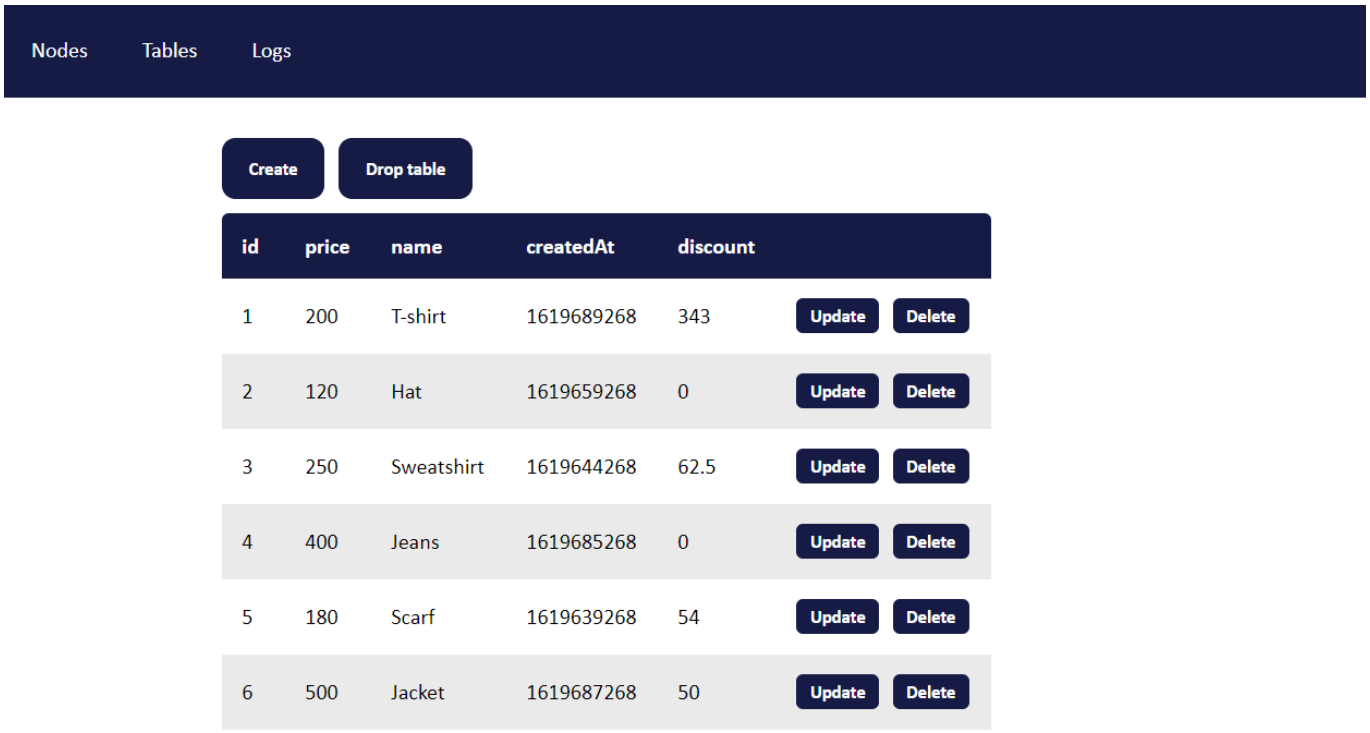

Рис. 3.12 Сторінка таблиці

3.3.4 Обробка помилок

Систему управління розподіленими базами даних мають складну структуру, тому їх розробка є досить скадною, і у просеці роботи системи можуть виникати проблеми, які потребують забезпечення коректного завершення операцій, що відбуваються у процесі взаємодії з даними.

Для обробки помилок був розроблений модуль менеджера процесу, що містить стек подій, які виконуються при виникенні проблеми у системі. Наприклад, на рисунку 3.13 продемонстрований журнал подій, у якому описаний кожен крок фрагментації таблиці. Коли, під час цього процесу виникла помикла, був запущений відкат через менеджер процесу і стан системи був повернений до початкового стану. Програмний код модуля представлений у додатку Б.

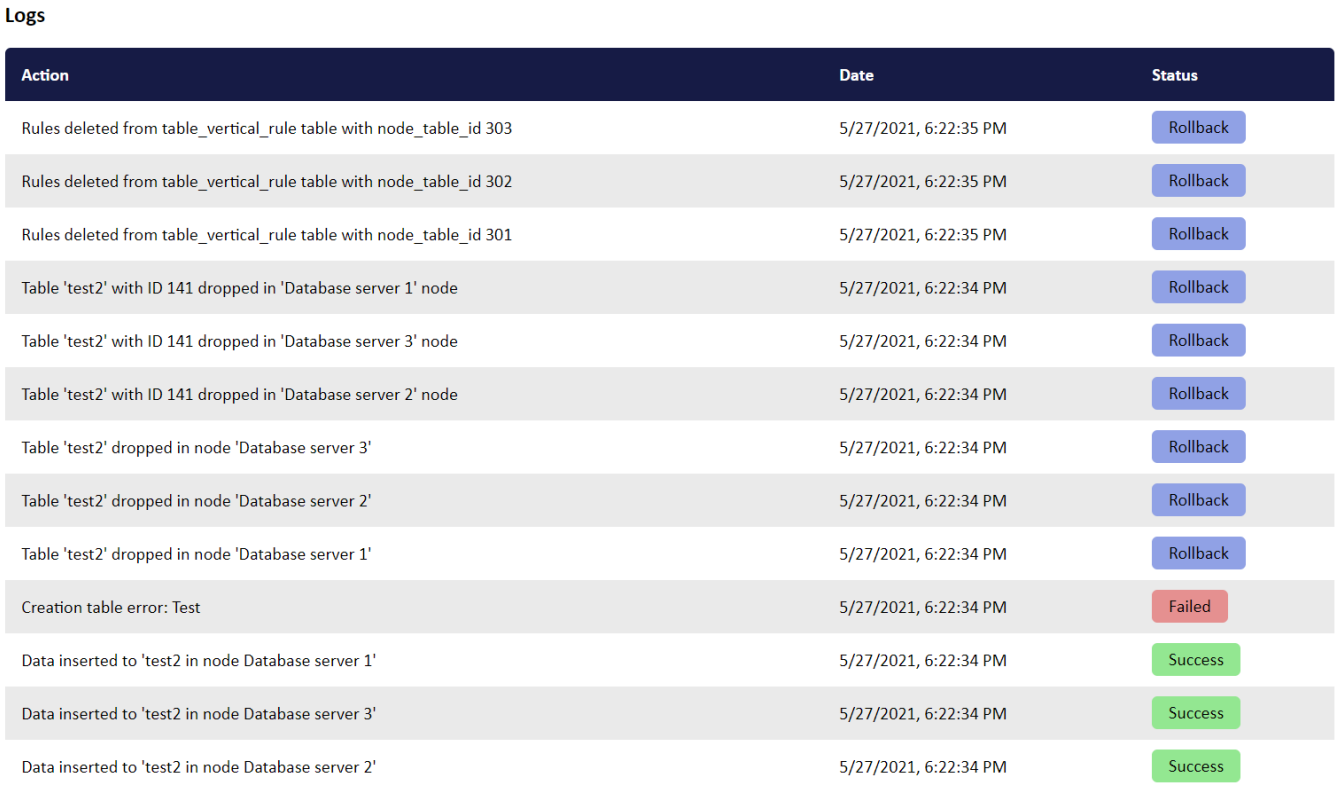

Рис. 3.13 Відкат системи при помилці

У прикладі вище (див. рис. 3.13) представлений випадок, коли на кроці завершення фрагментації відбулась помилка і система почала здійснювати операції відкату одна за одною витягуючи їх зі стеку.

3.4 Висновки до розділу

Проведений опис розробленого програмного застосунку для управління розпобіленими базами даним, що включає детальний розбір роботи кожного елементу системи. Реалізовані усі складові компоненти програмного комплексу, а саме, клієнтську панель управління РБД, сервер управління запитами та шаблон серверу бази даних.

У розділі розглянуті основні об’єкти, з якими працює користувач, а сами вузли та таблиці, їх реалізацію та представлення у панелі управління. Структура застосунку не є досить складною, що є її перевагою, тому вона може досить швидко виконувати весь необхідний функціонал і легко розширюватись в залежності від потреб клієнтів.

Проведено запуск та тестування роботи ситеми, що продемонструвала успішне виконання всіх складових елементів, серед яких важливо виокремити менеджер управління процесом, що показав коректність обробки помилок з інформуванням користувача у таблиці подій і забезпечив стабільність та цілісність виконання всіх процесів системи.

Отже, було успішно проведено розробку власного програмного застосунку і продемонстровано роботу усіх складових частин системи. У результаті ми отримали повноцінно функціонуючий засіб, що працює відповідно до спроектованої моделі системи і має можливість розподілу та взаємодії з даних у мережі.

РОЗДІЛ 4
АНАЛІЗ РЕЗУЛЬТАТІВ ТА ПІДВЕДЕННЯ ПІДСУМКІВ

4.1 Повірняння програмного застоснку з аналогами

Результатом робзробки стала система управління розподіленими базами даних, що надає користувачеві повноцінний функціонал створення нових таблиць з можливістю самостійного налаштування фрагментації даних.

Якщо порівнювати даний програмний застосунок з існуючими аналогами, то можно одразу сказати, що він є досить простим і легким у використанні, оскільки сучасні системи управління РБД представляють великі програмні рішення, що працюють на різних рівнях систем. Важливо зазначити, що сучасні рішення є частинами вже існуючих комплексів управління БД або домоміжними модуля, як наприклад модуль DB2 фірми IBM. Такий підхід не завжди є зручним, тому розроблена система є власним сервісом, що орієнтується виключно на роботу з система управління РБД.

Масштабування та підхід до реалзація є відмінним від існуючих сервісів. Зазвичай засоби управління інтегрують нові вузли як окремний готовий модуль. У свою чергу, розроблений застосунок працює з шаблонами серверів баз даних (заготовлені сервера для розміщення у якості вузлів), тому вони можуть розгортатись окремо, не мати прив’язки до системи управління і бути окремим програмними засобами. Шаблони доступні для вдосконалення під потреби користувача, але потребують доримання ряду правил.

Ще однією ключовою особливістю розробленої системи є те, що вона передає запити до сервера БД у вигляді об’єтів спеціально формату, а на вузлах транслює у SQL запити для взаємодії з БД. Як вже було зазначено, існуючі аналоги працють на різних рівнях органіції системи, тому передача запитів може відбуватись по різному. Деякі можуть надсилати їх на окремому рівні протоколів, як наприклад продукти

Oracle або IBM, або в особливому форматі відправки. Це описує організацію передачі даних між складовими компонентами системи РБД.

4.2 Шляхи покращення існуючої моделі

Розглянемо кілька шляхів покращення існснуючої моделі системи управління РБД:

- Додання можливості реплікації даних.

На даний момент система має можливість проводити тільки вертикальну та горизонтальну фрагментацію даних. Надання можливості синхронізації кількох копій даних може знадобитись у системах, які потребують максимальної цілісності зберігання інофрмації, тому додання цієї можливості дозводить значно підвищити функціональність і залучити нових користувачів.

- Вдосконалення концепції роботи з шаблонами баз даних.

Як вже було зазначено, шаблони використовуються для розгортання вузлів серверів баз даних і налаштування конфігурації системи. Додавши нові можливості до існуючого API можливо розширити функціональність функціональні можливості застосунку.

- Доповнення функціональності взаємодії з таблицями.

Зараз таблиці здатні працювати з даними на рівні базових операцій: читання, створення новго запису, оновлення запису та видалення. В перспективі можливо додати роботу з групами записів, модифікацією по критеріям, створення кількох записів і т.д.

- Організація створення та керування базами даних у панелі користувача.

Система керування підтримує роботу з таблицями, але нема можливості розділити їх на бази даних, оскільки не передбачений механізм, що реалізував би

подібні функції. Тому можливо додати новий об’єкт баз даних і розподіляти таблиці по по різним елементам БД.

- Доопрацювання механізму транзакцій.

Модель реалізує частковий функціонал транзакцій, а саме, вміє здійснювати відкат системи у разі виникнення проблем через модуль менеджеру процесу. Додавши можливість фіксації набору змін і котролю за синхронізацією доступу до баз даних можна значна вдосконалити систему управління.

4.3 Висновки до розділу

У розділі було підведено підсумки виконаних робіт з розробки системами управління розподіленими базами даних. У результаті був розроблений застосунок, що реалізує весь необхідний функціонал відповідно до спроектованої моделі системи. Він має ряд переваг перед існуючими аналогами і може задовольнити потреби будь-якого сегменту ринку.

Проведено порівняння існуючих аналогів систем з розробленим програмним застосунком, що дозволило виокремити особливості реалізації та надати нові погляди до проектування даних систем. Це надасть можливість легше орієнтуватись під час розробки нових засобів управління і поліпшить роботу з системами управління РБД.

Також були проаналізовані та надані рекомендації щодо покращення розробленої моделі, що можуть підвищити ефективність процесів та розширити можливості системи.

ВИСНОВКИ

В даній роботі було проведено дослідження систем управління розподіленими базами даних, їх архітектурних різновидів, переваг та недоліків використання. Розглянуті способи розподілу даних за допомогою механізму фрагментації і сихранізаціїї доступу до даних через інструмент транзакцій. Ознайомлено з такими продуктами систем управління, як Oracle System, INGRES/STAR та модулем DB2 компанії IBM.

Проблеми пов’язані з навантаженням та великими об’ємами інформації у базах даних є актальними для невеликих підприємств та бізнесів, оскільки зазвичай перевага надається централізованим рішенням. Розподіл даних та використання системи управління розподіленими базами даних дозволить вирішити ці проблем і може стати у нагоді для швидкозростаючого бізнесу. Тому, дослідження в даній області може допомогти відкрити нові шляхи для розвитку та покращення систем управління.

Проаналізувавши та дослідивши предметну область і теоретичні відомості, було спроектовано власну модель системи, що складається з тьох основних копонентів: клієнтського інтерфейсу, серверу управління запитами та сервервів баз даних. Кожен компонент відіграє свою роль і володіє обмеженим функціоналом в раммах своїх можливостей. Компоненти організують чіткий зв’язок для обміну даними між складовими частинами системи.

На основі спроектованої моделі системи, що реалізує архітектуру з багатьма незалежними серверами, було розроблено програмний застосунок управління розподіленими базами даних. Він представляє собою веб-застосунок, доступ до якого можна отримати з будь-якого пристрою, що має вихід у мережу. Відповідає всім функціональним та нефункціональним вимог і забезпечує взаємодію з розподіленими даними. Має можливість масштабування в залежності від побажань користувача.

На відміну від існуючих рішень, застосунок є досить легким у використанні, має можливість гнучкого масштабування, використовуючи шаблони вузлів мережі, як спосіб оранізації нових серверів баз даних, та представляє вузьконаправлений продукт, що здатен управляти розподіленими даними і не є частиною складного комплексу управління базами даних. Для підвищення ефективності процесів та розширення можливостей системи у майбутньому планується здійснити ряд запропонованих вдосконалень і надати користувачам можливість більш гнучкої взаємодії з програмним застосунком.

Проектне рішення моделі системи демонструє як легко можна ораганізувати засіб управління розподіленими базами даних, що реалізує весь необхідний функціонал розподілу та взаємодії з даними.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Організація баз даних: практичний курс: Навч. посіб. для студ. / А. Ю. Берко, О. М. Верес; Нац. ун-т «Львів. політехніка». – Л., 2003. – 149 с.

2. Fundamentals of database systems [Електронний ресурс]. – Режим доступу: http://www.uoitc.edu.iq/images/documents/informatics-institute/Competitive_exam/Database_Systems.pdf.

3. Коннолл Т., Бегг К., Страчан А. Бази даних: проектування, реалізація і супровід. – 2003. – 822-825 с.

4. Бази даних. Мови запитів. Управління транзакціями. Розподілена обробка даних [Електронний ресурс]. – Режим доступу: https://web.posibnyky.vntu.edu.ua/fitki/11petuh_bazdanyh_movy_zalitiv/32.htm.

5. Gray, Jim. The Transaction Concept: Virtues and Limitations. Proceedings of the 7th International Conference on Very Large Databases. – 1981. – 144-154 с.

6. Дейт К. Дж. Введення в системи баз даних: Пер. з англ. - 6-е вид. - Київ: Діалектика, – 1998. – 824-825 с.

7. Oracle Distributed Database Concepts [Електронний ресурс]. – Режим доступу: https://docs.oracle.com/cd/B10501_01/server.920/a96521/ds_concepts.htm#20409.

8. Introducing Ingres Star [Електронний ресурс]. – Режим доступу: https://docs.actian.com/ingres/10.2/index.html#page/Star/1._Introducing_Ingres_Star.htm

9. IBM Overview of Distributed Relational Database Architecture (DRDA) [Електронний ресурс]. – Режим доступу: https://www.ibm.com/docs/en/informix-servers/12.10?topic=communications-overview-drda.

10. Буч Г., Рамбо Дж., Джекобсон А. Язык UML. Руководство пользователя. – 2006. – 29-33 с., 120-123 с.

11. М. Кантелон , М. Хартер, Т. Головайчук, Н. Райлих. Node.js в действии. – 2014 – 9-10 с.

12. React. A JavaScript library for building user interfaces [Електронний ресурс]. – Режим доступу: https://reactjs.org/.

13. MySQL. MySQL Database Service [Електронний ресурс]. – Режим доступу: https://www.oracle.com/mysql/.

Додаток А

Структура бази даних

Структура бази даних серверу управління запитами складається з семи основних таблиць (див. рис. А.1): користувачів, таблиць, вузлів, подій, зв’язків між таблицями та вузлами, і правил горизонтальної та вертикальної фрагментацій.

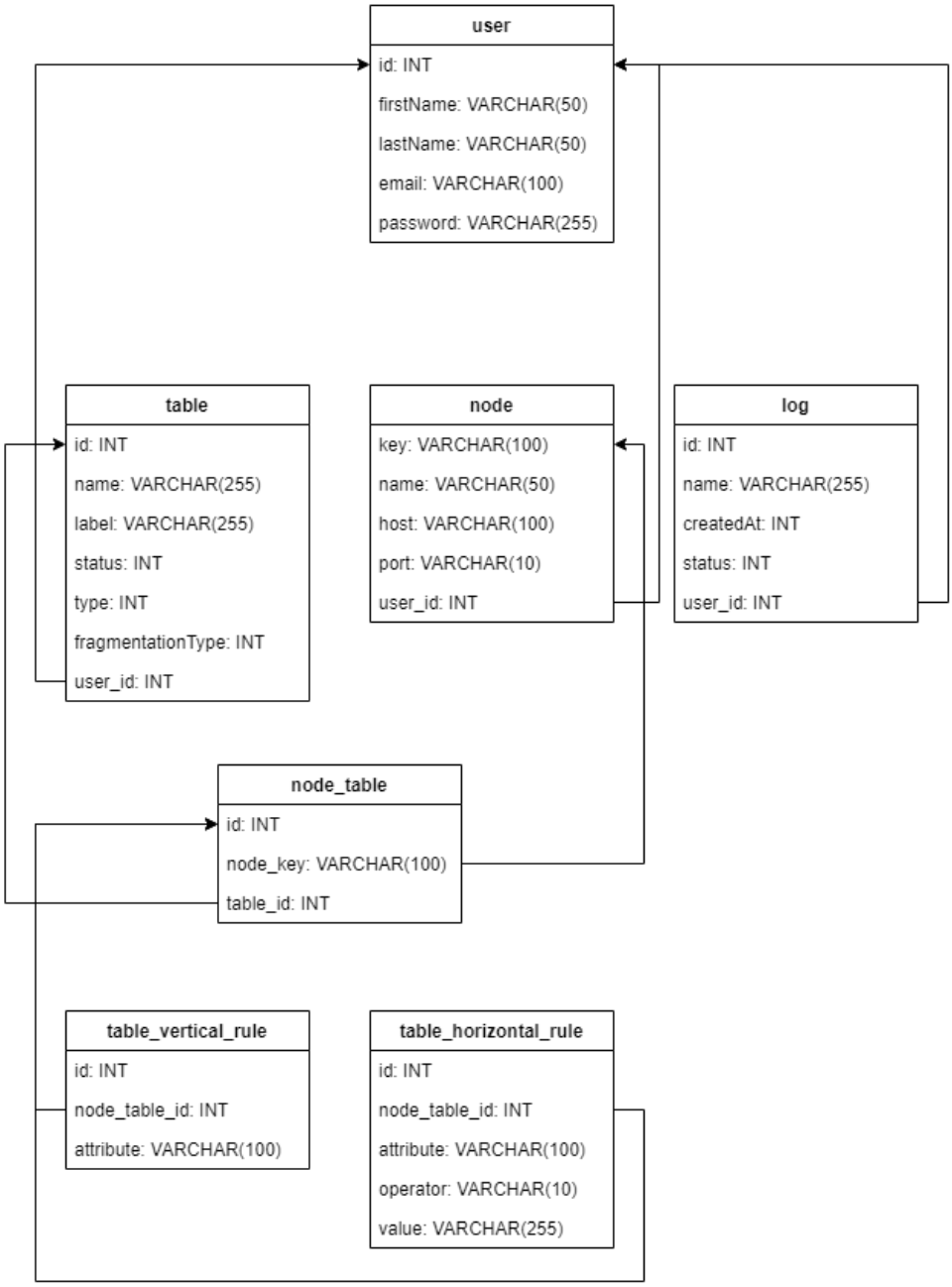

Рисунок А.1 Структура бази даних серверу управління запитами

Додаток Б

Код модуля менеджеру процесу

```
const STATUS = {
    ACTIVE: 'ACTIVE',
    INACTIVE: 'INACTIVE'
}

let state = {
    status: STATUS.INACTIVE,
    rollbackActions: []
}

const setState = newState => state = newState;

const start = () => {
    setState({
        ...state,
        status: STATUS.ACTIVE
    });
}

const addRollbackAction = action => {
    if (state.status === STATUS.INACTIVE) {
        throw new Error('Process not started')
    }

    setState({
        ...state,
        rollbackActions: [
            action,
            ...state.rollbackActions
        ]
    })
}

const rollback = async (user_id) => {
    if (state.status === STATUS.INACTIVE) {
        throw new Error('Process not started')
    }

    for (const action of state.rollbackActions.reverse()) {
        await action()
    }

    setState({
        status: STATUS.INACTIVE,
        rollbackActions: []
```

```
    });
}

const stop = () => {
    if (state.status === STATUS.INACTIVE) {
        throw new Error('Process not started')
    }

    setState({
        status: STATUS.INACTIVE,
        rollbackActions: []
    });
}

module.exports = { start, addRollbackAction, rollback, stop }
```

Додаток В

Software Architecture Document (SAD)

Taras Shevchenko National University of Kyiv

Research of architecture and management of systems with distributed databases

Software Architecture Document (SAD)

CONTENT OWNER: Karandas Denys

DOCUMENT NUMBER:	RELEASE/REVISION:	RELEASE/REVISION DATE:
1	1.0	May 2021

Table of Contents

1. Documentation Roadmap

1.1. Document Management and Configuration Control Information

- Revision Number: 1
- Revision Release Date: 21/05/2021
- Purpose of Revision: check the correctness and completeness of the created document

1.2. Purpose and Scope

Purpose: explore the types of architecture, develop a model and implement a distributed database management system.

Scope: the tool is used to store and interact with data. Users can use data warehouse management systems for their own needs or business decisions. In the long run, this technology can prove itself quite well as a web service or platform that optimizes data processing.

1.3. Viewpoint Definitions

1.3.1. Management system model Viewpoint Definitions

1.3.1.1. Abstract

Views on the model and processes of control systems are considered. These include entry points, action processes, event handlers, components, modules, and exit points. Views characterize the point of view on a particular object of the system or sequence of actions.

1.3.1.2. Stakeholders and Their Concerns Addressed

Stakeholders and their concerns include:

- Customers who are interested in the correct operation of the system as a whole.
- Developers who are interested in this topic and carry out activities related to it.
- Lifecycle and process management managers who need to know how the software is designed.
- Testers that test components and identify system performance issues.
- Organizers and leaders, for whom it is important to control all components of the software package.

1.3.1.3. Elements, Relations, Properties, and Constraints

Elements are part of a software package that are created to solve a specific problem. The elements in the system are interconnected and perform a number of actions at different levels of software implementation. They define consecutive and parallel types of bonds that can be described in different constructions. Such constructions include cyclic, conditional and program-defined processes. The properties of the elements depend on the goals of the component in which the solutions and problems of this element are written.

Therefore, the model of the management system of distributed databases has its own set of elements defined within the components in which they are implemented. The main components of the system include: client interface, query management server and database servers.

Problems can be formed on the basis of the elements themselves and their connections, so there are a number of limitations and requirements:

- Integrity. The system model should provide stable access to all elements of the system, which provides the ability to access any module without problems.
- Reliability. Ensuring reliable data exchange during network requests, which involves the implementation of modules and error handling processes.
- Scalability. The model should be able to expand the system without the need to stop it, which will add new nodes during the operation of the system.
- Efficiency. The system module should provide fast data exchange and no load.

- Convenience. Provides clarity of all elements of the interface when working with the control system.
- Functionally complete. The model needs to provide the user with all available functions for convenient operation.

1.3.1.4. Language(s) to Model/Represent Conforming Views

Views can be formed using such representations:

- Plain text that can be presented in a variety of formats, such as lists, forms, or regular paragraphs.
- Representation in the form of uml diagrams representing a unified modeling language is used in the paradigm of object-oriented programming, and allow you to correctly describe the data in graphical form.
- Software based on the representation of the system model in the form of code, algorithmic sequences and constructions of programming languages.

2. Architecture Background

2.1. Problem Background

2.1.1. System Overview

The system is a fully functional tool that works according to the designed model and has the ability to distribute and interact with data in the network.

The main purposes of the system include:

- Distributed data storage. The system allows you to store large amounts of information on the nodes of the Internet, which optimizes the processes of access to information and allows you to interact more effectively with it.
- Data fragmentation. Fragmentation is designed to divide an object into logical fragments (also known as segments) and their distribution and storage on certain network nodes.
- Distributed system management. The system provides the user with the full functionality of creating new tables with the ability to customize data fragmentation.

2.1.2. Goals and Context

The system model is based on the use of an architecture with rich independent servers. This architecture requires that system users store information about the data and location of their servers. There should also be a special application that will allow you to make split requests and collect them into one answer.

The architecture provides the ability to implement flexible scaling with optimization of processes that will occur between distributed network nodes. Each node of the system has its own template for organizing software processes, entry and exit points, so other tools must understand how to interact with it through a set of instructions provided during application design.

The system development life cycle determines the processes, starting from the stage of formation of the design model and ending with the presentation of a fully functioning management system for distributed databases. In this case, the design of the model, based on the architecture, is a key step that determines the main goals and objectives of the system. Different approaches and methods of building elements of the software complex, parts of the abstract model and algorithmic solutions are determined, which requires a detailed analysis and research of this area.

2.1.3. Significant Driving Requirements

The Architecture Tradeoff Analysis Method is a method for evaluating software architectures relative to quality attribute goals. Method evaluations expose architectural risks that potentially inhibit the achievement of an organization’s business goals.

Attributes define a set of characteristics that the system must have. Considering this topic, you need to focus on non-functional requirements for the system and the quality of the system as a result. On the other hand, if we consider the scenarios, then it is worth looking at the functional requirements and compare with the results of the components of each component of the system.

Depending on the stage of the life cycle and the tasks set on it, the attributes can be placed in order of importance. For example, if we organize a model of node placement in the network and their interaction, it will be appropriate to consider the attributes of reliability, integrity, communication and organization of data exchange.

Quality attribute requirements are the means by which a system is designed to achieve its business objectives. These requirements must be recognized and taken into account early in the life cycle, and the architecture of the system and software must be designed to match their performance characteristics.

2.2. Solution Background

2.2.1. Architectural Approaches

Software architecture refers to the basic structure of a software system and the discipline to create such a structure. Each structure consists of program objects, the properties they possess, and the relationships.

During software development, an architectural approach using many independent servers in a distributed database management system was considered. The structure of the tool was based on the implementation of a component approach to development, which was chosen on the basis of the creation of reusable components to facilitate work with large amounts of data. This approach makes it easier to set up connections between system elements and allows for flexible configuration of the software.

The software model implements several different software design patterns that facilitate work with the system itself and optimize the software elements of the application. In software engineering, software schemes are a common solution for reusable problems that typically arise in a particular software design context. This is not a complete design that can be converted directly to source code or machine code. Design templates are formalized best practices that programmers can use to solve common problems in developing a program or system. In a distributed database management system, the query management server implements an architectural template (pattern) MVC, which allows you to organize a convenient structure of processes and distribute responsibilities between the constituent objects of the server.

To implement the concept of data sharing, it was decided to implement the fragmentation mechanism used to divide the data of the table we create between the available network nodes. The system model involves the implementation of two types of fragmentation: vertical and horizontal. Fragmentation is used to divide an object into logical fragments and store it on certain network nodes.

And another very important approach in the management system is the use of data exchange templates in the network, representing objects of a special kind. Use is due to the need for convenient and fast exchange of requests to obtain data from network nodes.

2.2.2. Analysis Results

After analyzing and researching the subject area and theoretical information, we designed our own model of the system, consisting of the following main components: the client interface, the query management server and database servers. Each component plays its role and has limited functionality within its capabilities. The components organize a clear connection for the exchange of data between the components of the system.

Based on the designed model of the system, which implements the architecture with many independent servers, a software application for distributed database management was developed. It is a web application that can be accessed from any device that has access to the network. Meets all functional and non-functional requirements and provides interaction with distributed data. Has the ability to scale depending on the user's wishes.

The design solution of the system model demonstrates how easy it is to organize a tool for managing distributed databases, which implements all the necessary functionality of distribution and interaction with data.

2.2.3. Requirements Coverage

An important condition for working with the system is the availability of access to the network and the ability to open a web application.

Development Requirements:

- WebStorm IDEA or other development environment;
- Programing language JavaScript;
- React;
- Node.js;
- MySQL;

3. Referenced Materials

IEEE 1471	ANSI/IEEE-1471-2000, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, 21 September 2000.
Barbacci 2003	Barbacci, M.; Ellison, R.; Lattanze, A.; Stafford, J.; Weinstock, C.; & Wood, W. Quality Attribute Workshops (QAWs), Third Edition (CMU/SEI-2003-TR-016). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. <http://www.sei.cmu.edu/publications/documents/03.reports/03tr016.html>.
Bass 2003	Bass, Clements, Kazman, Software Architecture in Practice, second edition, Addison Wesley Longman, 2003.
Carlos Coronel 2018	Database Systems: Design, Implementation, & Management, 13th edition, Carlos Coronel, 1 January 2018.
Saeed K. Rahimi, Frank S. Haug 2010	Distributed Database Management Systems: A Practical Approach, Saeed K. Rahimi, Frank S. Haug, August 2010.
Alex Petrov 2019	Database Internals: A Deep Dive into How Distributed Data Systems Work, 1st edition, Alex Petrov, 22 October 2019.
Brendan Burns 2018	Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services, 1st edition, Brendan Burns, 20 March 2018.

4. Directory

4.1. Glossary

Term	Definition
Software architecture	The structure or structures of that system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible" properties refer to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.
Software design pattern	In software engineering, a software design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design. It is not a finished design that can be transformed directly into source or machine code. Rather, it is a description or template for how to solve a problem that can be used in many different situations. Design patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system.
Distributed database	Is a set of logically interconnected databases distributed on a computer network. To manage such a system, use a distributed database management system, which creates the illusion of working with a single database, as if the data is stored in one place.
React	Is an open-source front-end JavaScript library for building user interfaces or UI components. It is maintained by Facebook and a community of individual developers and companies. React can be used as a base in the development of single-page or mobile applications. However, React is only concerned with state management and rendering that state to the DOM, so creating React applications usually requires the use of additional libraries for routing, as well as certain client-side functionality.
Node.js	Is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web-application development around a single programming language, rather than different languages for server-side and client-side scripts.
MySQL	Is an open-source relational database management system. A relational database organizes data into one or more data tables in which data types may be related to each other; these relations help structure the data.

4.2. Acronym List

SAD	Software Architecture Document
MVC	Model-View-Controller
DB	Database
DBMS	Database Management System
DDBMS	Distributed Database Management System
UML	Unified Modeling Language
IEEE	Institute of Electrical and Electronics Engineers