Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

Створення та використання розподілених баз даних

Текстова частина до курсової роботи
за спеціальністю «Інженерія програмного забезпечення»

Керівник курсової роботи
ас. Яремко С. А.
_____
(підпис)
“____” _____ 2024 р.
Виконав студент 3-го курсу
Ахмадов О. С.
“____” _____ 2024 р.

Київ 2024

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри інформатики,

проф., д.ф.-м.н.

_____М. М. Глибовець

(підпис)

„____”_____2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Ахмадову Олексію Сергійовичу факультету інформатики 3 курсу

ТЕМА Створення та використання розподілених баз даних

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Календарний план

Вступ

1 Основні концепції та типи розподілених баз даних

2 Архітектура розподілених баз даних

3 Методи реплікації та розвитку розподілених баз даних

4 Безпека в розподілених базах даних

5 Розробка власного застосунку

Висновки

Список літератури

Перелік прийнятих скорочень

Дата видачі „___” _____ 2024 р. Керівник _____(підпис)

Завдання отримав _____(підпис)

Тема: Створення та використання розподілених баз даних

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	24.09.2023	
2.	Огляд технічної літератури за темою роботи.	01.11.2023	
3.	Вибір концепції додатку.	10.01.2024	
4.	Написання текстової частини курсової.	20.01.2024	
5.	Розробка додатку.	01.03.2024	
6.	Оформлення презентації.	10.05.2024	
7.	Здача роботи на перевірку.	16.05.2024	
8.	Захист курсової роботи.	24.05.2024	

Студент Ахмадов О. С.

Керівник Яремко С. А.

“_____”_____

Зміст

Анотація

У цій курсовій роботі розглянемо створення та використання розподілених баз даних. Основна увага приділяється MySQL базі даних, де буде створено розподілена система у конфігурації master-slave, де головна БД обробляє всі запити, а резервна БД синхронізується з нею для забезпечення резервного копіювання.

Робота містить огляд основних концепцій РБД, відмінностей між Homogeneous та Heterogeneous системами, аналіз типів конфігурацій (active-passive, active-active) та архітектурних рішень. Розглянуто стратегії реплікації, фрагментації та шардування даних, а також заходи безпеки.

У практичній частині реалізовано застосунок на Python, що використовує бібліотеку pymysql для підключення до баз даних, та налагоджено автоматичне переключення на резервну базу в разі відмови головної. Використано тригери та EVENTS для автоматизації процесу реплікації.

Вступ

В 21 столітті основне питання, де зберігати данні, стає все більш актуальним. Все більше людей купують смарт холодильники, колонки, телевізори, які з'єднуються в одну систему та зв'язані між собою через інтернет. Такі пристрої відомі, як: **Internet of things**(IoT).

А тепер уявіть, що з кожним днем все більше і більше людей починають використовувати IoT. Мільярди пристроїв генерують величезні обсяги даних, зростає кількість запитів, якоїсь корисної інформації, яку треба десь зберігати, в якихось сховищах. Усім відомі централізовані бази даних вже не можуть впоратися та потребують інших можливостей, які могли б покрити головні проблеми, а саме: масштабованості та гнучкості.

Рішення з'явилось в середині 1960 роках, яке назвали: розподілені бази даних(РБД), які змогли вирішити проблеми масштабованості, продуктивності та доступності, з якими централізовані БД не могли впоратися через надмірне навантаження. Головна ідея РБД розподілення даних на декілька серверів чи комп'ютерів. Це дозволяє обробляти запити швидше, бути стійкими до відмови одного сервера, бо, в цей же момент, його замінить інший та мати майже нескінченну можливість масштабування.

Може здатися, що РБД найкращий варіант та чому не використовувати їх скрізь? Але в кожному підході та рішеннях існують свої недоліки, для розподілених БД це будуть: складність управління, складність розгортання такої системи, мережеві затримки. Але окремо треба сказати про головну та найважливішу проблему усіх систем та ПЗ – це безпека. Розглянувши ці питання докладніше у наступних розділах, отримаємо глибше розуміння переваг та недоліків РБД з точки зору розгортання, управління та налаштування безпеки.

Мета курсової роботи розкрити та повністю зрозуміти актуальність розподілених баз даних. Головною частиною роботи буде чотири розділи, де ми розкриємо основні концепції РБД, вивчимо основні найпоширеніші архітектури, дізнаємось про методи реплікації в розподілених системах та про безпеку.

Зрозуміємо, де і коли треба використовувати розподілені бази даних, які переваги та недоліки можуть бути в різних задачах, що і буде ключем для ефективного вирішення проблем в галузі обробки даних.

Основні концепції та типи розподілених баз даних

Визначення розподіленою бази даних

Розподілені бази даних є важливою складовою сучасних інформаційних систем, що використовуються в різних сферах праці, від малих бізнесів до великих компаній, яким іноді потрібно створювати свої РБД та інструменти для подальшого розвитку.

Сама назва вже говорить нам про розподіленість систем баз даних, а саме – яка працює, обробляється на декількох чи навіть більше комп'ютерах чи серверах водночас та іноді навпаки, по одинці, коли не має прямого зв'язку між системами.

РБД відрізняються від централізованих, де вся інформація зберігається на одному носії, що може стати недоліком у майбутньому для розширення системи, бо одна з головних переваг – це зручність в масштабованості.

В основі РБД лежать три основні концепції: реплікація, фрагментація та шардування.

- Реплікація – робить копію даних.
- Фрагментація – розбиває таблиці, які зберігають конкретні дані, на частини для полегшення їх використання.
- Шардування – це процес розподілу даних, зазвичай після виконання фрагментації, на різні фізичні вузли, такі як: сервера, комп'ютери.

Більш детальне пояснення цих концепцій та їх роль у розподілі та перенесенні корисної інформації буде розглянуто у наступних розділах роботи.

Завдяки основним перевагам РБД, компанії можуть забезпечити себе нескінченними можливостями розширення та масштабованістю інформації,

ефективного використання та зберігання даних, швидкою відповіддю клієнту від серверу, де б він не знаходився.

Але головне розуміти, що система з великою кількістю переваг не може бути без недоліків.

Переваги та недоліки розподілених систем

Трішки згадаємо традиційні централізовані бази даних(ЦБД), щоб мати точне розуміння переваг розподілених баз даних та їх недоліків у порівнянні.

Як вже було сказано, ЦБД зберігають усі данні на одному фізичному носії. Це може бути звичайний комп'ютер чи цілий сервер. Їх чудово створювати для вирішення малих задач, де маєш точне уявлення, як система буде прогресувати у майбутньому.

Зрозуміло, що ЦБД мають свої недоліки та переваги, але головне розуміти:

- ЦБД зазвичай створюються, коли точно розумієш потреби системи та можеш передбачити обсяг даних, який буде оброблятися.
- Вони відомі простотою створення та використання таких БД.
- Забезпечують швидку відповідь клієнту при умові, що навантаження на сервер не велике.
- Мають складність в масштабуванні даних. Тут мається на увазі, коли настає час покращувати систему, іноді потрібно робити дуже велику роботу для перебудови, іноді навіть, перероблювати повністю дизайн БД.
- Також, централізовані бази даних неефективні або зовсім неможливі у реалізації, якщо потрібно зберігати дані в різних географічних регіонах світу та забезпечувати ефективно швидку відповідь користувачу.

І тому, в середині 1960 роках почалось використання розподілених баз даних, де більшість недоліків ЦБД було вирішено та широко використовуються по цей час.

Основні переваги та недоліки РБД:

Переваги:

- Головна перевага розподілених систем, що вони можуть зручно масштабуватися незалежно від зростаючого обсягу даних, що зростає з кожним днем.
- Дозволяють зберігати дані на різних серверах, що можуть знаходитись в різних місцях та країнах. Це забезпечує швидку відповідь до клієнту та можливість робити реплікації, тобто резервні копії даних, щоб забезпечити надійність в разі відмови одного з серверів.
- Можна розподіляти навантаження між серверами, що допоможе збільшити продуктивність під час великих навантажень.

Недоліки:

- Розподілені бази даних складніше проектувати та обслуговувати ніж ЦБД.
- Дорожчі, обслуговування та необхідне обладнання буде значніше дорожчим, особливо, коли серверів багато.
- Складність транзакцій, через те, що дані можуть бути розміщені на декількох вузлах.
- Безпека є однією з головних недоліків. РБД більш вразливі до атак ззовні. Система може складатися з багатьох пристроїв, які зв’язані між собою для передачі та отримання різних запитів, повідомлення в мережі можуть переглядатися або перехоплюватися.

Основні принципи розподілу даних

РБД є важливою складовою сучасник інформаційних систем. Одним із ключових аспектів є їхня архітектура та конфігурація. Типи розподілених баз даних та їх конфігурація є основними принципами розподілених систем.

Типи розподілених баз даних

Централізовані бази даних зазвичай використовують однакове програмне забезпечення та структуру даних, такий тип БД називають Homogeneous або омогенний. Використання в РБД омогенного підходу не завжди є доцільним.

У великих організаціях і підприємствах може бути низка існуючих систем, які використовують різне ПЗ, різні формати даних. І тому треба використовувати інший тип бази даних, а саме: Heterogeneous або гетерогенні. Використання гетерогенного підходу дозволяє простіше інтегрувати нову розподілену базу даних до існуючої системи без необхідності уніфікації всіх даних.

Homogeneous Distributed Databases(Омогенні розподілені бази даних) – тип БД, я яких всі вузли розподіленої системи мають однакове ПЗ та структуру даних.

Основні переваги таких систем це: стандартизація, спрощеність масштабування та розгортання. Усі вузли використовують однакове програмне забезпечення, що спрощує управління такої системи. У порівнянні з гетерогенним підходом, таку БД простіше розгорнути та масштабувати, якщо потрібно підключити до основної системи додаткові ресурси. Але, якщо РБД використовує різні формати даних та різні застосунки, краще розробляти БД з гетерогенним підходом.

Heterogeneous Distributed Databases(Гетерогенні розподілені бази даних) – тип систем, де в кожному вузлі використовується різне програмне забезпечення та структури даних.

Розподілені бази даних, які використовують гетерогенний підхід мають велику кількість переваг, які, в свою чергу, тягнуть за собою деякі недоліки.

Здатність використовувати різне ПЗ, дозволяє адаптувати кожний вузол до конкретних потреб або вимог. Це дозволяє вибрати найбільш підходящий інструмент для вирішення конкретних задач. Крім того, цей підхід надає додаткової гнучкості у роботі та використання системи, обробляти та використовувати різні типи даних, розгортання додаткових серверів з різним ПЗ.

Основні недоліки гетерогенного підходу:

- Коли використовується різні типи даних на різних вузлах, може з’явитись проблема з синхронізацією та обмінами даних. Різні формати та програмне забезпечення може призвести до складнощів передачі даних між різними частинами системи.

- Іншим недоліком гетерогенного типу є складність управління, у гетерогенної системи, де використовується різне ПЗ, ускладнений процес управління та підтримки. Що потребує додаткової кваліфікації для адміністрування таких БД.

- Потреба у використанні сервера-посередника, який діє як міст між різними компонентами системи, що мають різне ПЗ чи тип даних. Основна роль такого сервера, конвертація даних одного формату в інший. Це дозволяє спростити взаємодію між різними серверами з різними типами даних та ПЗ.

Гетерогенні розподілені бази даних відкривають широкі можливості для адаптації кожного вузла системи до конкретних вимог та потреб. Цей

підхід надає гнучкість у виборі інструментів та обробці різних типів даних, що забезпечує ефективність та високу функціональність розподіленої системи. Однак, необхідно враховувати й недоліки такого підходу, такі як складність синхронізації та управління даними.

Конфігурація розподілених баз даних

Одна з головних задач розподілених баз даних – це висока доступність даних від серверу до клієнту. Виникає задача, правильно налаштувати БД таким чином, щоб забезпечити цілодобовий доступ до неї і даних, котрі вона зберігає. Це може бути досягнуто за допомогою налаштування конфігурації бази даних.

Конфігурація визначається, як розподілення та організація різних компонентів БД у великій системі. Це впливає на розміщення даних на різних вузлах, фрагментацію, визначення ролей та взаємозв’язки серверів між собою. Коли бази даних розподілена, її данні реплікуються на кілька фізичних вузлів і є кілька способів налаштування цих реплік:

Active-passive:

Найпростіший тип конфігурації баз даних є Active-passive конфігурація. В цьому режимі весь трафік проходить через активний сервер та потім усі зміни, які відбулися на активному, реплікуються на інші “пасивні” сервера для резервного копіювання. Основна мета, мати можливість відновлення даних у випадку відмови активного сервера.

Основні характеристики:

- **Активний сервер:** Один основний сервер, через який проходить усі запити та трафік, він обробляє усі запити та повертає потрібні данні.

- **Пасивний сервер:** Допоміжний сервер чи сервери, які працюють пасивно, зазвичай їх використовують для зберігання даних активного серверу, які регулярно оновлюються. У випадку, якщо головний вузол перестане працювати, його швидко замінить один з пасивних.

Цей підхід є простим, але він може призвести до потенційних проблем та недоліків:

- **Реплікація даних:** Немає ніяких гарантій, що не буде ніякого збою в процесі реплікації даних між серверами і усі данні будуть відображені на пасивному вузлі. Це може призвести до втрати даних і до проблем з консистентністю.
- **Консистентність:** Якщо данні реплікувати синхронно, і якщо один з пасивних серверів не зможе записати копію даних активного, це може призвести до порушення консистентності, коли відбувається конфлікт даних між активним та пасивним сервером.
- **Втрата продуктивності:** Для Active-passive конфігурації використовуються пасивні сервера, які потрібні для збереження даних активного серверу. Вони не використовуються для обробки запитів і тому вони завжди знаходяться в пасивному стані.

Таким чином, Active-passive конфігурація є найпростішим типом, який може використовуватися в малих задачах, коли треба розгорнути невелику реляційну РБД.

Active-active:

Конфігурація Active-active вже більш складніша та не має недоліків з продуктивністю. На відміну від Active-passive, конфігурація Active-active передбачає, що всі сервери будуть в активному стані. Запити можуть бути спрямовані до будь якого активного вузла. В цій системі навантаження розподіляється рівномірно, що дозволяє підвищити продуктивність системи, забезпечити високий рівень доступності та мати можливість обробляти запити паралельно.

Основні характеристики:

- **Висока доступність:** Через те, що всі сервери знаходяться в активному стані, у разі відмови одного з них, інші активні сервери можуть продовжувати обробляти запити користувачів без затримок. Це забезпечує неперервну роботу БД та уникнення втрат продуктивності.
- **Балансування навантаження:** Навантаження рівномірно розподіляється між усіма вузлами, що дозволяє ефективно використовувати ресурси системи, що додатково забезпечить швидку відповідь від сервера до клієнту.

Недоліками Active-active будуть: складне налаштування для більшості задач. Також часто виникають проблеми з узгодженням даних між серверами.

Уявимо два активних сервери A та B. Вузол A отримує запит на оновлення даних в той самий час, що і вузол B. Якщо A виконує оновлення даних, відправляючи запит для синхронізації з вузлом B, а він в свою чергу робить теж саме, синхронізуючись із першим сервером, то це може спричинити неузгодженість між серверами та призвести до проблем цілісності даних.

Тому треба пам’ятати, при створенні РБД, про неузгодженість між серверами, які можуть виникнути в результаті асинхронності або різниці у стані даних. Це призведе до конфліктів даних, втрати цілісності інформації або навіть втрати даних.

Але все ж таке налаштування, коли кожен сервер активний, є найбільш надійним та ефективним в розподіленій базі даних. Оскільки забезпечує високу доступність, продуктивність та швидку роботу серверів.

Архітектура розподілених баз даних

Архітектура розподілених баз даних відіграє важливо роль при створення та проектування розподілених БД. Вибір правильної архітектури впливає на оптимізацію системи, спосіб передавання даних, надійність БД та доступність. Це важливо для створення максимально оптимізованої системи**.** Що дозволить зробити систему ефективною та гнучкою, для швидкого обробляння поточних завдань.

Основні архітектури, які найчастіше використовуються при створенні розподілених баз даних:

- **Client−Server Architecture**
- **Peer−to−Peer Architecture**
- **Multi - DBMS Architectures**

Вибір кожної архітектури вплине на оптимізацію, ефективність обробки даних та загальну продуктивність інформаційної системи. Кожна з перерахованих архітектур має свої особливості та відмінності у способі організації взаємодії між клієнтами та серверами, а також у розподіленні та обробці даних.

Client - Server Architecture:

Клієнт-серверна архітектура в розподілених баз даних є однією з найпоширеніших та важливих моделей. Вона базується на концепції розділення функціоналу між клієнтом та сервером.

В цій системі клієнт відправляє різноманітні запити, а сервер, у свою чергу, обробляє отриманні данні та повертає потрібну інформацію назад до користувача. В цій концепції клієнт-серверна архітектура передбачає, що всі

данні зберігаються на сервері, а клієнт, використовуючи різні пристрої, через програмний інтерфейс, може звертатися до них.

Клієнти повинні знати про доступні сервери, хоча і не можуть мати уявлення про існування інших клієнтів.

Така модель є найпоширенішою та зрозумілою, тому її найчастіше використовують. Проте треба зазначити, що вона має свої переваги та недоліки.

Головна перевага цієї архітектури є централізоване керування, яке спрощує керування та безпеку системи. Крім того, така модель дозволяє зручно та ефективно масштабувати систему, додаючи нові сервери. Клієнт-серверна архітектура також сприяє зменшенню навантаження на мережу, оскільки всі дані обробляються на сервері, що зменшує навантаження на комп’ютери користувачів.

Основні недоліки, які можуть зустрітися – це проблеми з безпекою та точка відмови сервера.

Існує головна проблема безпеки, пов’язана з конфіденційністю та цілісністю даних під час передачі між клієнтами та сервером через мережу. Недостатня захищеність може призвести до витоку конфіденційних даних.

Щодо точки відмови, коли сервер перестає працювати, це може привезти до повної дієздатності системи. Користувачі не зможуть відправляти та обробляти свої запити до сервера. Це може виникнути через технічні несправності, програмні помилки.

У висновку, клієнт-серверна архітектура в РБД є однією з найбільш поширених та важливих моделей. Вона ґрунтується на концепції розділення функціоналу між клієнтом та сервером. Ця модель є простою та зрозумілою, проте має свої недоліки та переваги, які треба враховувати при виборі моделі, забезпечуючи необхідний рівень безпеки та доступності для користувачів.

Peer−to−Peer Architecture:

В розподілених баз даних Peer−to−Peer архітектура є ще одною основною моделлю. В цій системі кожен вузол може виконувати роль як клієнта, так і сервера. Навіть якщо один з вузлів перестане працювати, інші зможуть виконувати свою роботу незалежно від інших серверів.

У Peer−to−Peer архітектурі співпрацюють один з одним, обмінюючи даними без централізованого керування, як в Client - Server моделі, іншими словами: децентралізоване керування. Це робить P2P більш стійкою до відмов з ладу окремих серверів. Вона є більш масштабованою, ніж Client – Server, оскільки кількість вузлів може додаватися без обмежень.

Автономність є вродженою якістю P2P мереж, де кожен вузол функціонує незалежно. Це робить їх по справжньому демократичними, оскільки жоден окремий вузол не має повного контролю над усією мережею.

Однак, Peer−to−Peer архітектура має свої недоліки. Наприклад, через відсутність централізованого контролю може призвести до складності управління та до проблем з безпекою. P2P-системи більш схильні до кібератак, таких як DDoS-атаки та атаки з метою крадіжки даних. Швидкість передачі запитів між вузлами теж не завжди однакова. Кожен вузол може мати свою швидкість мережі, що знижує ефективність обміну даних.

При виборі архітектури для РБД важливо ретельно зважити переваги та недоліки P2P-архітектури. P2P може бути доцільним вибором для випадків, коли необхідна масштабованість, доступність та стійкість до відмов. Однак важливо враховувати складність, проблеми з безпекою та потенційні проблеми з координацією вузлів, які можуть виникнути при використанні P2P.

Multi - DBMS Architectures:

У сучасному часі, коли великі обсяги даних керують світом, розподілені бази даних є найбільш ефективними для забезпечення доступу до інформації. Проте, з розвитком технологій виникають нові проблеми інтеграції та управлінням РБД. Серед різноманітних та поширених архітектур, таких як: Client – Server та Peer−to−Peer Architectures, Multi-DBMS архітектура відіграє важливу роль в розподілених системах.

Multi-DBMS архітектура представляє собою підхід, який об'єднує декілька автономних баз даних в єдину, логічну зв'язану структуру. У цій моделі кожна БД незалежна та зберігає свої дані, але використовуючи посередника, вони можуть бути легко доступні, як одна об’єднана система. Такий підхід є дуже зручний для компаній, які мають розподілені структури даних, оскільки це дозволяє об’єднати усі дані в єдину систему без необхідності перенесення всіх даних в одну централізовану базу. Це полегшує використання різних типів даних, навіть якщо використовується різні БД.

Multi-DBMS архітектуру можна розділити на 2 підходи:

Федеративна архітектура:

У цій архітектурі різні бази даних залишаються розділеними та незалежними, але вони можуть співпрацювати між собою через спеціальний шар абстракції, відомий як федерація. Федерація дозволяє об'єднувати дані з різних джерел та виконувати запити, що охоплюють декілька баз даних, як одне об'єднане джерело. Це дозволяє забезпечити доступ до даних з різних носіїв через єдиний інтерфейс та спрощує обробку даних з різних джерел.

Реплікаційна архітектура:

У цій архітектурі дані розміщуються на кількох серверах, і зміни в одній базі даних автоматично реплікуються на інші бази даних. Основна мета реплікації - забезпечити доступність та надійність даних. Це означає, що якщо один сервер вийде з ладу, інші сервери можуть продовжувати роботу і

забезпечувати доступ до даних. Реплікаційна архітектура дозволяє підвищити доступність та швидкодію обробки запитів за рахунок розподілення навантаження та резервного копіювання даних. Іншими словами, робиться копія усіх даних на кожний сервер, у випадку недієздатності одного, його місце займе інший, з усією інформацією.

Основні типи реплікації

- Повна реплікація
- Часткова реплікація
- Транзакційна реплікація

Основні переваги Multi-DBMS архітектури включають: можливість доступу до розподілених даних без необхідності їх фізичного об'єднання, що зменшує навантаження на мережу та спрощує управління даними; масштабованість, федеративна архітектура може бути легко розширена шляхом додавання нових автономних БД до системи; гнучкість: федеративна архітектура дозволяє використовувати різні типи БД, що дає можливість вибирати найкраще рішення для кожного набору даних**.**

Однак, Multi-DBMS архітектура може стикатися з певними викликами:

- **Складність**: Multi-DBMS архітектура може бути складною у проектуванні, розробці та управлінні.
- **Продуктивність**: Доступ до даних в федеративній системі може бути трохи повільнішим, ніж у централізованій БД, через необхідність координації між декількома БД.
- **Безпека**: Multi-DBMS архітектура може бути більш вразливою до атак, ніж централізована БД, через те, що дані розподілені по декількох БД.

Multi-DBMS архітектура є потужним інструментом для об'єднання декількох автономних БД в єдину віртуальну систему. Її можна

використовувати для покращення масштабованості, гнучкості, доступності та безпеки БД.

Методи реплікації та розвитку РБД

Фрагментація даних

Фрагментація в понятті БД означає розділення чи розбиття баз даних або таблиць на менші логічні частини. Це робиться за критеріями, такими як рядки (горизонтальна фрагментація) або стовпці (вертикальна фрагментація) таблиці, або комбінацією попередніх методів (гібридна фрагментація).

Способи Фрагментації:

- Під час **горизонтальної фрагментації**, дані розбиваються на рядки за певним критерієм. Наприклад, якщо у вас є таблиця клієнтів і ви розділяєте їх за регіонами, то кожен регіон може бути окремим фрагментом даних. А також горизонтальна фрагментація дозволяє рівномірно розподілити обсяг даних між повними серверами або логічними частинами системи.

- У **вертикальній фрагментації** таблиці розбиваються на стовпці за певними атрибутами або характеристиками. Наприклад, великі таблиці з великою кількістю стовпців можна розділити на частини, де кожна частина містить основні атрибути, такі як ім’я клієнта або номер телефону. Цей підхід дозволяє оптимізувати доступ до даних.

- **Гібридна фрагментація** – це комбінація горизонтальної та вертикальної фрагментації для оптимального розподілу даних. Наприклад, ви можете спочатку горизонтально розділити дані для регіонів, а потім вертикально розділити кожен регіон, використовуючи менші атрибути. Гібридна фрагментація дозволяє гнучко досягти структури даних залежно від конкретних потреб системи.

Основна мета фрагментації даних відбувається в покращенні ефективності обробки даних та забезпеченні швидкого доступу до них. Розподіл бази даних на частини дозволяє розподіляти завантаження між ефективними серверами або областями систем, знижуючи час відповіді та підвищуючи продуктивність.

Фрагментація також може бути корисною для забезпечення безпеки даних, тому вона дозволяє обмежувати доступ до певних частин інформації лише вповноваженим користувачам або системам.

Плюси фрагментації даних:

- **Покращення продуктивності:** Фрагментація даних дозволяє оптимізувати доступ до інформації, зменшуючи час відповіді на запити та підвищуючи швидкість обробки даних.

- **Ефективність використання ресурсів:** Розподіл даних між ефективними серверами або частинами системи дозволяє рівномірно розподіляти навантаження, що зменшує перевантаження системи та підвищує її ефективність роботи.

- **Безпека даних:** через фрагментацію можна обмежувати доступ до певних частин інформації, що забезпечує конфіденційність та цільність даних та знижує ризик несанкціонованого доступу.

Мінуси фрагментації даних:

- **Управління складністю:** підтримка фрагментованих даних вимагає більше зусиль з боку адміністраторів баз даних, необхідно керувати розміщенням і розподілом даних.

- **Можливість дублювання:** фрагментація даних може привести до дублювання інформації, коли одні й ті самі дані зберігаються в кількох

місцях одночасно. Це означає, що для збереження і синхронізації цих дублікатів даних необхідні додаткові ресурси, такі як місце на диску та час управління цими даними.

- **Складність запитів:** якщо у вас є фрагментовані дані, тобто дані розділені на частини, це може зробити складнішим написання запитів до бази даних. Потрібно враховувати, як саме розподіляються дані та як вони пов'язані між собою між різними частинами баз даних.

Враховуючи ці плюси та мінуси, важливо збалансувати використання фрагментації даних із врахуванням конкретних потреб системи, обсягу та чутливості даних, а також ресурсів, доступних для управління базою даних.

Узагальнюючи, фрагментація даних є важливим інструментом для оптимізації роботи баз даних, забезпечення надійності та безпеки інформації, а також підвищення продуктивності системи управління даними.

Реплікація даних

Реплікація - це процес створення та підтримки копій даних або БД на кількох серверах або віддалених місцях із метою забезпечення доступності, надійності та швидкості обробки даних. Він також забезпечує надмірність і стійкість до відмов, гарантуючи, що дані реплікуються на кілька вузлів синхронно або асинхронно.

На реплікацію даних впливають нижче зазначені фактори:

- Розмір самої БД
- Частота використання БД
- Витрати, (ефективність, непродуктивні витрати ПЗ та управління) спричинені синхронізацією транзакції та їх частин при забезпеченні належної відмовостійкості, пов’язаної з реплікацією даних

Цілі реплікації даних включають забезпечення надійності системи, що дозволяє продовжувати роботу при виході з ладу одного сервера або у випадку збою, забезпечення доступності інформації для користувачів навіть при проблемах з однією з реплік, покращення швидкості доступу до даних через близькість реплік до користувачів та їх менше навантаження, забезпечення безпеки за допомогою резервних копій на репліках для відновлення даних у разі втрати або пошкодження, а також можливість масштабування системи шляхом додавання нових серверів та реплік для обробки зростаючого обсягу даних чи збільшення навантаження.

Основні типи реплікації:

1. **Повна реплікація (Full Replication):**
    - У цьому типі реплікації всі дані з основної бази даних копіюються на інші сервери. Це означає, що всі таблиці та рядки дублюються на кожній репліці, що забезпечує повний набір даних на кожному сервері.
    - Повна реплікація є надійним методом, оскільки вся інформація завжди доступна на кожній репліці. Однак це вимагає більше місця для зберігання даних та більше ресурсів для синхронізації.
2. **Часткова реплікація (Partial Replication):**
    - У цьому випадку лише певні частини даних копіюються на інші сервери. Наприклад, ви можете вибрати певні таблиці або фрагменти таблиць для реплікації.
    - Часткова реплікація може бути корисною, якщо не всі дані необхідні на кожній репліці. Вона зменшує обсяг даних для синхронізації, але потребує уваги до вибору даних для реплікації.
3. **Транзакційна реплікація (Transactional Replication):**
    - Цей тип реплікації передбачає автоматичну реплікацію змін у базі даних, що відбуваються у межах транзакцій. Це означає, що зміни,

які відбуваються на основному сервері, автоматично передаються на репліки.

- Транзакційна реплікація дозволяє забезпечити консистентність даних між репліками, оскільки зміни відбуваються атомарно в межах транзакцій.

Ці типи реплікації мають свої властивості та використання, і вибір між ними залежить від конкретних потреб системи, обсягу даних, доступних ресурсів та вимог до надійності і швидкості доступу до даних.

Шардинг даних

В міру збільшення кількості користувачів, обсяг даних, який зберігає БД також зростає. І ми стикаємось з проблемою, коли БД становиться занадто малою, щоб умістити в собі всі необхідні дані. Це впливає на швидкість та продуктивність роботи з БД. Шардинг є одним з варіантів вирішення цієї проблеми, оскільки він надає можливість паралельно обробляти менші набори даних у шардах.

Шардинг бази даних — це процес зберігання великої бази даних на кількох серверах або комп’ютерах. Один комп’ютер або сервер бази даних може зберігати й обробляти лише обмежену кількість даних. Шардинг бази даних долає це обмеження, розділяючи логічні таблиці БД на менші фізичні одиниці, які називаються шардами, і зберігаючи їх на кількох серверах баз даних. Усі сервери баз даних зазвичай мають однакові основні технології, і вони працюють разом, щоб зберігати та обробляти великі обсяги даних.

Також важливо зазначити, що існують різні методи шардингу, наприклад горизонтальне (поділ рядків) і вертикальне (розбиття стовпців).

Перевагами шардування є краща продуктивність, оскільки користувачі можуть отримувати доступ лише до тих даних, які їм потрібні, і масштабованість бази даних, оскільки шардування дозволяє збільшити

кількість збережених даних, завдяки тому, що вони будуть зберігатися в різних місцях. Однак недоліки шардингу включають підвищену складність керування базою даних і можливість резервування даних, коли ті самі дані зберігаються в кількох шардах.

Безпека в розподілених базах даних

Основні загрози

Розподілені бази даних все більш широко використовуються компаніями та організаціями, ніж централізовані БД, завдяки їх масштабованості, доступності та стійкості до відмов. Безпека в РБД є однією з найважливіших аспектів. Дані в БД повинні гарантовано безпечно зберігатися. Інформація не може бути вилучена без дозволу або бути загубленою в процесі виконання транзакції чи операції.

Деякі основні загрози безпеки в РБД:

- **Втрата даних**. В наслідок технічних проблем, які можуть статися через катастрофу чи зловмисних дій ззовні, можлива втрата чи повна їх недоступність.
- **Неавторизований доступ**. Отримання даних зловмисниками, використовуючи різні методи для їх отримання, наприклад: крадіжка паролів, використання вразливих ПЗ.
- **Компрометація даних**. Це може статися під час передачі даних через мережу, недостатня захищеність каналів зв’язку може призвести до вразливостей, таких як: несанкціонований доступ, зміни або перехоплення даних.

Безпека даних підтримуються комплексом стандартних заходів таких як: шифрування даних, резервне копіювання, аутентифікація та авторизація, моніторинг та аудит. Кожен з цих заходів створює різні шари безпеки для надійного зберігання даних.

Методи забезпечення безпеки

Аутентифікація та авторизація:

Забезпечення ідентифікації користувачів та перевірка прав доступу до інформації в БД. Зазвичай, використовується найпоширеніший засіб авторизація, використання логіна та пароля для входу у систему. Будь-який користувач повинен бути зареєстрований в системі для отримання доступу до БД під унікальним логіном та паролем. Спочатку користувач не має доступу до баз даних і тому йому потрібно ввести свій логін та пароль для отримання повноважень.

Повноваження можуть бути різними. Можуть бути різні розподіли прав. Тому є таке поняття, як роль. Роль – сукупність прав та повноважень, які можуть бути надані користувачам чи прикладним програмам, які роблять запити до БД.

Шифрування даних:

Використання вже готових рішень шифрування, чи створення своїх, є одним з основних заходів безпеки даних від зловмисників. Забезпечує захист від несанкціонованого доступу до інформації.

Шифрування перетворює звичайний текст чи набір даних в незрозумілий формат, який створений алгоритмом шифруванням. Такий формат може бути розшифрований тільки з використанням відповідного ключа. Іншими словами, зловмисник, який вкрав данні, не може їх розшифрувати без ключа.

Шифрування може використовуватися до різних типів даних: файли, картинки, повідомлення, текст. Загалом, шифрування даних є одним за найпоширеніших способів безпеки даних, забезпечення їх цілісності та конфіденційності.

Моніторинг та аудит:

Моніторинг та аудит є ключовими аспектами систем безпеки даних. Вони допомагають виявляти та відслідковувати події пов’язані з безпекою.

Моніторинг та аудит – процеси збору та аналізу даних про безпеку з метою виявлення порушень та підозрілих дій. Основні аспекти моніторингу та аудиту системи:

- **Аналіз журналів подій.** Журнали подій зберігають інформацію про всі події, які відбулися в системі. Це охоплює спроби входу користувачів до системи, зміни даних в БД, зміни налаштування в системі та інші дії.
- **Відслідковування активності користувачів.** Моніторинг дозволяє відслідковувати активності користувачів, як часто хто намагається зайти у систему, доступ до закритих даних. Це все може свідчити про потенційні загрози, які можуть бути заздалегідь виявлені.
- **Аудит безпеки.** Аудит безпеки включає в себе систематичну перевірку та оцінку безпеки системи на предмет відповідних стандартів захисту.
- **Реагування на події.** Моніторинг дозволяє збирати усі дані та робити аналіз, який допомагає заздалегідь виявити потенційну небезпеку та прийняти рішення проти неї та прийняття заходів для запобіганню подальших атакам.

Моніторинг та аудит безпеки - це важливі інструменти, які можуть допомогти організаціям захистити свою інформацію від зовнішніх загроз, записуючи усі події та роблячи їх аналіз для виявлення потенційних небезпек для системи.

Резервне копіювання:

Резервне копіювання – процес створення копій інформації з метою забезпечення безпеки та відновлення у випадку втрати основних даних.

Регулярне створення резервної копії даних запобігає втраті інформації внаслідок технічних збоїв чи повної її втрати.

Основні принципи резервного копіювання даних:

- **Повнота**. Копії даних повинні бути повними та містити усю інформацію для відновлення іншого сервера.
- **Регулярність**. Backup повинен здійснюватися регулярно, чим частіше ти краще, але це залежить від ресурсів системи. Це забезпечить актуальність копії інформації.
- **Зберігання копій в різних місцях.** Реплікації даних повинні зберігатися на різних носіях. Щоб максимально знизити шанс втрати корисної інформації, навіть якщо деякі копії зникнуть, все рівно залишиться носії, які несуть потрібну інформацію.

Найчастіше використовується схема master-slave:

Master – основний сервер БД, де записуються всі дані. Усі зміни в даних повинні відбуватися на головному сервері.

Slave – це додатковий сервер БД, який в пасивному стані робить весь час копію головного сервера, master.

Коли система виявляє, що master недоступний, наприклад, через втрату зв’язку, вона автоматично переключає slave з пасивного стану в активний і він замінює master. Після того, як slave-сервер переключився на активний режим, він приймає на себе всі запити та продовжує обслуговувати клієнтів, як і master-сервер раніше.

Резервне копіювання може бути здійснене як локально так і на інших серверах. Це простий та ефективний спосіб захисту даних системи від можливості втрати. Резервне копіювання дозволяє бути впевненим в захисті даних, це дозволяє мати можливість тестувати свій застосунок, з можливістю

відновлення даних. Головне треба робити реплікації регулярно, щоб можна було відновити систему найактуальнішою інформацією.

Розробка власного застосунку

У даному розділі розглядається створення мого застосунку, який реалізовує розподілені бази даних, використовуючи MySQL базу даних з конфігурацією master-slave.

Для свого застосунку використовується база даних MySQL та мова програмування Python, для тестування створення та використання розподілених баз даних**.**

Прикладна область РБД – зберігання категорій та товарів, які розподілені по категоріям.

Мета:

Метою даного проекту є розробка та тестування розподіленої бази даних, яка забезпечує високу надійність та доступність даних за рахунок використання реплікації та резервного копіювання. Для цього я створив дві бази даних, які реплікуються, створюють резервні копії, і в разі відмови головної бази даних (master), автоматично переходять до резервної бази даних (slave).

Архітектура застосунку

Основні компоненти:

Використовується стратегії master-slave.

Master: головна база даних(shop_main), яка обробляє всі запити на запис.

Slave: резервна база даних(shop_backup), яка синхронізується з master для забезпечення резервного копіювання (рисунок 2.0).

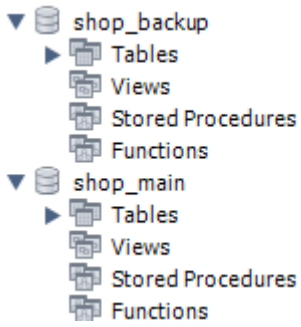

Рисунок 2.0 master та slave

Клієнт: Python-застосунок, який надсилає запити на сервер бази даних.

Client-Server Architecture:

Використовується Клієнт-серверна архітектура для створення застосунку. У цій архітектурі клієнт взаємодіє із сервером бази даних (MySQL), надсилаючи запити на запис до головної бази даних (shop_main) та отримуючи відповіді, використовуючи резервну базу даних (shop_backup) для резервного копіювання.

Реплікація даних:

Реплікація даних для копіювання усієї бази з master робить на основі тригерів та подій. Використовується транзакційна повна реплікація, це потрібно для забезпечення узгодженості актуальності даних між master та slave.

Використання тригерів забезпечує перенесення даних між головною БД до резервної (рисунок 2.1). Після кожної зміни даних, використовуючи CRUD операції, наприклад INSERT, UPDATE, DELETE, запускається тригер, який після зміни робить таку саму зміну на резервному носії(slave) (рисунки 2.2 та 2.3).

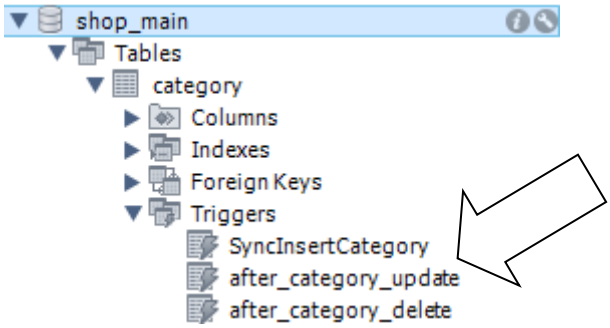

Рисунок 2.1 Створення основних тригерів

```
CREATE TRIGGER SyncInsertCategory
AFTER INSERT ON shop_main.category
FOR EACH ROW
BEGIN
    INSERT INTO shop_backup.category (id_category, name)
    VALUES (NEW.id_category, NEW.name);
END //
```

Рисунок 2.2 Використання тригера після INSERT

```
CREATE TRIGGER after_category_delete
AFTER DELETE ON category
FOR EACH ROW
BEGIN
    DELETE FROM shop_backup.category
    WHERE id_category = OLD.id_category;
END //
```

Рисунок 2.3 Використання тригера після DELETE

Якщо під час роботи застосунку зникає зв’язок з master, його замінює slave. Який в цей час отримує нові данні та зміни даних, яку зберігає. Slave зберігає усі зміни та нові данні у додаткові таблиці (рисунок 2.4), щоб, коли з’явиться зв’язок з master, переписати вже нові данні на нього.

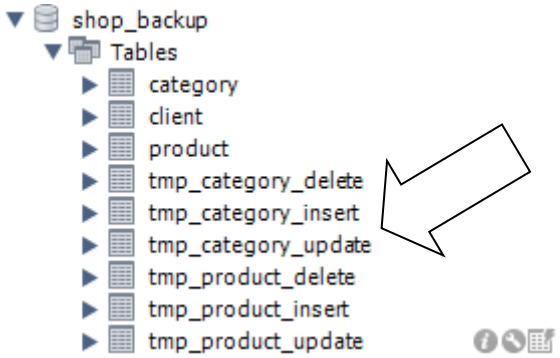

Рисунок 2.4 Створення додаткових таблиць

Для оновлення новими даними головного сервера, використовуються події(EVENTS), які кожні 10 секунд активуються, якщо master з’явився – оновлюють його дані, якщо його нема – продовжують чекати з ним зв’язок (рисунок 2.5).

```
CREATE EVENT category_update_event
ON SCHEDULE EVERY 10 SECOND
DO
BEGIN
    REPLACE INTO shop_main.category (id_category, name)
    SELECT id_category, name
    FROM tmp_category_update;

    DELETE FROM tmp_category_update;
END //
```

Рисунок 2.5 Використання події кожні 10 секунд

Реалізація на Python:

Виконується тестування РБД та створення застосунку, використовуючи MySQL та Python. Підключення до БД MySQL виконується за допомогою бібліотеки pymysql (рисунок 2.6).

```
import pymysql
```

Рисунок 2.6

Створюється підключення до через pymysql до master та slave баз даних (рисунок 2.7). Master – shop_main. Slave - shop_backup

```
def __init__(self):
    self.primary_db_config = {
        'host': 'localhost',
        'database': 'shop_main',
        'user': 'root',
        'password': '0995188581'
    }

    self.backup_db_config = {
        'host': 'localhost',
        'database': 'shop_backup',
        'user': 'root',
        'password': '0995188581'
    }
```

Рисунок 2.7 Підключення до баз даних

Створений простий застосунок, який перевіряє перемикання Master на Slave у разі його відключення (рисунок 2.8).

Підключення відбувається кожен раз, коли йде якась операція з БД, після неї вона закриває підключення (рисунок 2.9).

```
Successfully connected to master database
Connection closed
```

Рисунок 2.9

У випадку, якщо зв’язок з Master зникає, ми можемо повністю працювати з застосунком, буде виконано зразу підключення до Slave БД (рисунок 3.0).

```
Error connecting to master database: (1049, "Unknown database 'shop_main
Successfully connected to slave database
Connection closed
```

Рисунок 3.0

Висновки

У цій курсовій розглянуто концепції та особливості розподілених баз даних, чим вони відрізняються від звичайних централізованих. Розглянуті основні переваги та недоліки РБД.

Розподілені бази даних пропонують значне перевагу у вигляді підвищеної надійності та доступності даних. Було розглянуто, як різні типи розподілених систем, зокрема Homogeneous та Heterogeneous, використовують у різних сценаріях, надаючи унікальні переваги та можливості.

Робота містила аналіз різних типів конфігурацій та стратегій РБД, таких як active-passive та active-active. Було розглянуто різноманітні архітектури, які найчастіше застосовуються при створенні та проектуванні РБД, надаючи глибоке розуміння їх функціонування.

Було детально розглянуто стратегії реплікації, типи фрагментації та процес шардування, що є критично важливими для ефективного управління розподіленими базами даних. Аналіз цих аспектів дозволив зрозуміти, як забезпечити узгодженість та цілісність даних у розподіленій системі.

Було розглянуто потенційні небезпеки при створенні та експлуатації РБД, а також заходи щодо їх запобігання. Використання методів аутентифікації, авторизації та резервного копіювання були визначені як ключові заходи для забезпечення безпеки даних.

Проектом було реалізовано приклад використання MySQL з конфігурацією master-slave, де дві бази даних реплікуються, забезпечуючи резервне копіювання. Це дозволяє автоматично переключатися на резервну базу даних у разі відмови головної бази, забезпечуючи безперебійну роботу системи.

Використовуючи транзакційну реплікацію, виконується безперервне оновлення даних з slave до master.

Ця робота надала глибоке розуміння концепцій розподілених баз даних, їх архітектурних рішень, методів забезпечення надійності та безпеки. Реалізація прикладного проекту дозволила на практиці оцінити переваги РБД і зрозуміти їх значення у сучасних інформаційних системах.

Додаток А

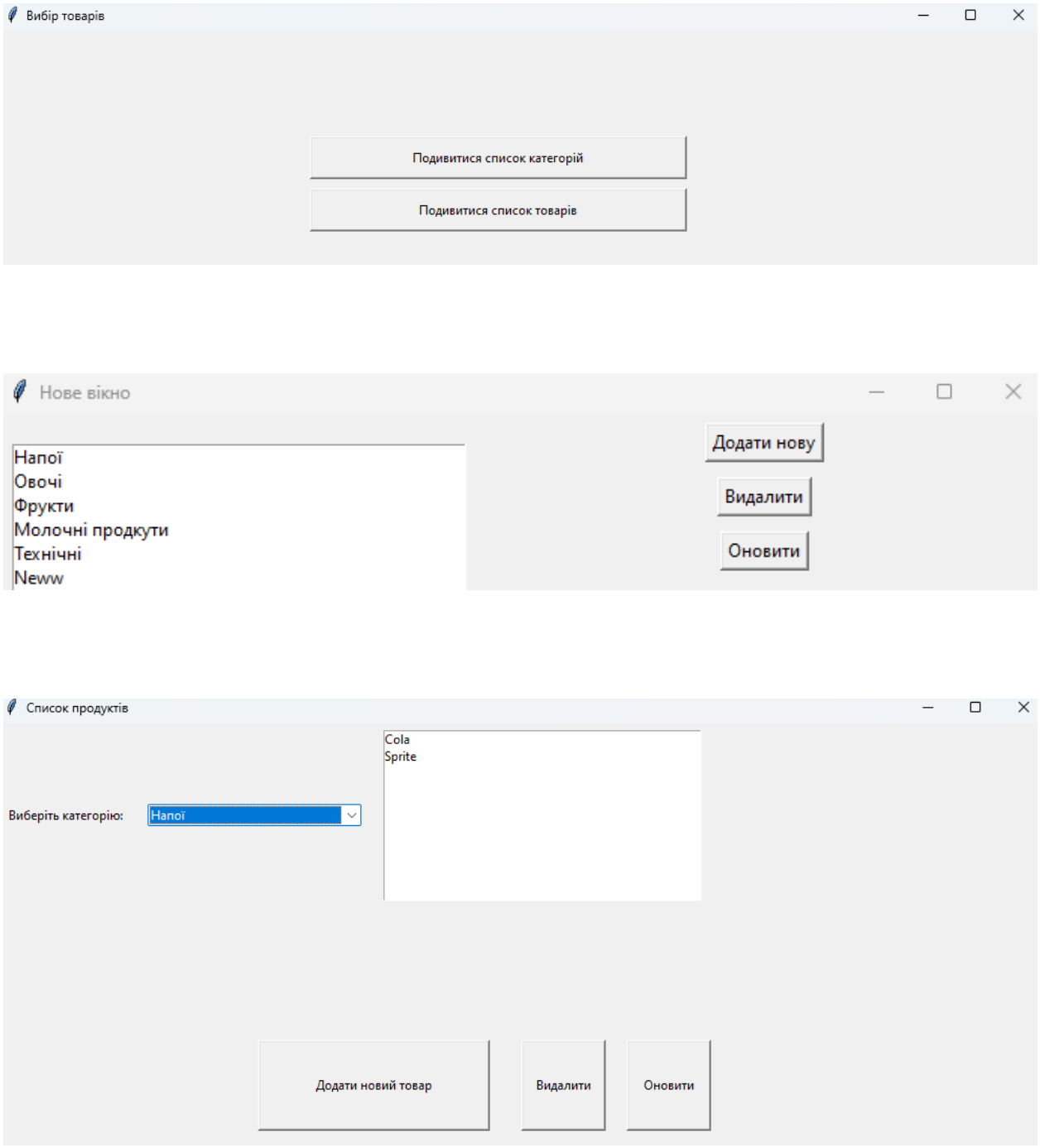

Рисунок 2.8 Приклад інтерфейсу застосунку

Список використаної літератури

1. https://studfile.net/preview/5203877/page:4/
2. https://www.cockroachlabs.com/blog/what-is-a-distributed-database/#:~:text=A%20distributed%20database%20is%20a,servers%20on%20a%20computer%20network.
3. https://phoenixnap.com/kb/distributed-database#ftoc-heading-3
4. https://www.tutorialspoint.com/distributed_dbms/distributed_dbms_databases.htm
5. https://elearning.sumdu.edu.ua/free_content/lectured:89b3d175c06a6b137e410cb14821d0e94549ad5a/20151030211833/44700/index.html
6. https://web.posibnyky.vntu.edu.ua/fitki/11petuh_bazdanyh_movy_zalitiv/32.htm
7. https://aws.amazon.com/what-is/database-sharding/?nc1=h_ls
8. https://www.tutorialspoint.com/distributed_dbms/distributed_dbms_database_environments.htm

Перелік прийнятих скорочень

IoT – Internet of Things

БД – бази даних

ПЗ – програмне забезпечення

РБД – розподілені бази даних

ЦБД – централізовані бази даних

P2P – Peer−to−Peer

DDoS – Distributed Denial of Service