Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Пояснювальна записка

до магістерської кваліфікаційної роботи

магістр

на тему: Програмна система оптимізації обміну даних у паралельних обчисленнях при обробці баз даних

Виконав: студент II курсу, групи 2ПІ-19м

спеціальності

121 – Інженерія програмного забезпечення

(шифр і назва напряму підготовки, спеціальності)

Брюханов В.С.

(прізвище та ініціали)

Керівник: к.т.н., доц. Рейда О. М.

(прізвище та ініціали)

Рецензент: д.т.н., проф. Василевський О. М

(прізвище та ініціали)

Вінниця – 2021

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Ступінь вищої освіти – магістр
Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Зав. каф. ПЗ, проф., д.т.н.
_____О. Н. Романюк
”09”березня 2021р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Брюханову Володимиру Сергійовичу

1. Тема роботи: Програмна система оптимізації обміну даних у паралельних обчисленнях при обробці баз даних.

Керівник роботи: Рейда Олександр Миколайович, к.т.н., доцент, затверджені наказом вищого навчального закладу від “09”березня 2021 року №64

2. Строк подання студентом роботи:

3. Вихідні дані до роботи: дані, засоби обробки запитів, методи оптимізації запитів, реляційна база даних; система управління базами даних; операційна система – Windows; інтегроване середовище розробки – SQL Server Management Studio; мова програмування – SQL.

4. Зміст розрахунково-пояснювальної записки: вступ; обґрунтування вибору методу розробки та постановка задачі дослідження; розробка програмної архітектури програмно-інформаційної системи для; опис структури та методів обробки даних; тестування; економічна частина; висновки.

5. Перелік графічного матеріалу: актуальність обраної теми; основні задачі дослідження; наукова новизна одержаних результатів; порівняльний аналіз аналогів;архітектура інформаційної системи; лістинг.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Рейда О. М к.т.н., доц. кафедри ПЗ	17/02/2021	01/06/2021
5	Глущенко Л.Д., к.е.н., доцент кафедри ЕПВМ	17/02/2021	01/06/2021

7. Дата видачі завдання: 17 лютого 2021 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів проекту (роботи)	Примітка
1	Обґрунтування вибору методу розробки та постановка задачі дослідження	20.02.2021_– 12.03.2021	Вик.
2	Розробка програмної архітектури програмно-інформаційної системи для роботи з базою даних	13.03.2021– 31.03.2021	Вик.
3	Опис структури та методів обробки даних	01.04.2021– 25.04.2021	Вик.
4	Тестування роботи та аналіз результатів	26.04.2021– 30.04.2021__	Вик.
5	Економічне обґрунтування доцільності розробки	01.05.2021– 16.05.2021	Вик.
6	Оформлення матеріалів до захисту МДР	17.05.2021 – 31.05.2021	Вик.

Студент _____ **Брюханов В.С.**
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Рейда О.М.**
(підпис) (прізвище та ініціали)

Рецензент магістерської кваліфікаційної роботи _____ **д.т.н., проф. каф. КН Василевський О.М.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

В магістерській дипломній роботі проведено аналіз методів організації паралельних запитів у великих потоках даних для систем доступу до баз даних. Сформульовано мету досліджень - підвищення взаємодії користувача з великими масивами даних за допомогою паралельної обробки запитів.

У ході роботи було проведено опис структур даних та визначині засоби і методи тестування проектованої системи. Запропоновано метод адаптивної оптимізації ресурсів при виконанні паралельних запитів великих баз даних та метод паралельної організації запитів великих баз даних.

У проведених експериментах використовуються статичне та динамічне тестування. Тестування підтвердило ефективність і правильність функціонування запропонованих рішень.

У роботі проведено економічні розрахунки витрат та прибутків від впровадження розробки, терміни окупності та визначено її комерційний потенціал.

Отримані в магістерській кваліфікаційній роботі наукові та практичні результати можна використати для збільшення швидкодії роботи з базою даних у наявних застосунках.

ANNOTATION

In the master's thesis the analysis of methods of the organization of parallel inquiries in large data streams for systems of access to databases was made. The purpose of the research was formulated - to increase the user's interaction with large data sets by means of parallel request processing.

During the process of work on thesis the description of data structures was carried out and means and methods of testing of the designed system were defined. A method of adaptive optimization of resources when performing parallel queries of large databases and a method of parallel organization of queries of large databases were proposed.

Static and dynamic testing are used in the conducted experiments. Testing has confirmed the effectiveness and correctness of the proposed solutions.

The economic calculations of costs and profits from the implementation of development were made, payback period and its commercial potential was determined.

The scientific and practical results obtained in the master's qualification work can be used to increase the speed of work with the database in existing applications.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. В сучасному світі базу даних (БД) можна визначити як уніфіковану сукупність даних, спільно використовувану різними завданнями в рамках деякої єдиної автоматизованої інформаційної системи (ІС).

Бази даних організовуються і управляються на основі реляційної моделі мови SQL. Прикладні програми на SQL, як правило, є комбінаціями звичайних програм і операторів SQL. Програми взаємодіють з клієнтами, відображають дані і забезпечують високорівневий напрямок потоку даних. Така модель запропонована з метою збільшення продуктивності баз даних. Додатковою перевагою є незалежність даних методу обробки запиту. Використання SQL дозволяє керувати базою даних при зміні логічних і фізичних схем. Паралельні системи баз даних мають пріоритет над традиційними, так як дозволяють оперувати з великими базами даних в режимі, що підтримує транзакції.

Використання паралельної обробки запитів (Parallel Data Query, PDQ) – це технологія, яка дозволяє розподілити обробку складного запиту на процесори, для використання максимально доступних системних ресурсів, у багато разів скорочуючи час отримання результату. Основні типи завдань, на яких проявляється ефект технології PDQ: обробка складних запитів, що включають сканування великих таблиць, сортування, з’єднання, групування, масові вставки; побудова індексів; збереження і відновлення даних; завантаження, вивантаження даних, реорганізація баз даних; масові операції вставки, видалення, модифікації даних.

Практично це дозволяє зменшити час обробки даних для організації оперативної реакції на терміновий запит, зменшується кількість проблем, пов’язаних з обробкою і обслуговуванням (архівуванням, копіюванням) великих таблиць – завдяки фрагментації, паралельній обробці і можливостям виконання адміністративних дій в оперативному режимі. В результаті розширюється клас потенційних додатків, і відповідно, коло користувачів.

Таким чином актуальність роботи безпосередньо пов’язана із підвищення продуктивності роботи із великими базами даних , що значно підвищує швидкодію обробки і якість отриманих даних.

Мета та завдання дослідження. Метою роботи є підвищення взаємодії користувача з великими масивами даних за допомогою паралельної обробки запитів.

Основними задачами дослідження є:

- провести аналіз методів розробки автоматизованих систем паралельної обробки баз даних;
- зробити порівняльний аналіз аналогів та методів розв’язання поставленої задачі системи паралельної обробки баз даних;
- провести аналіз методів організації паралельних запитів у великих потоках даних для систем доступу до баз даних та виконати програмну реалізацію інформаційної системи швидкісної обробки бази даних;
- запропонувати нові:
    - метод паралельної організації запитів великих баз даних
    - метод адаптивної оптимізації ресурсів при виконанні паралельних запитів великих баз даних
- провести опис структур даних та визначити засоби і методи тестування проектованої системи;
- розробити програмні компоненти та систему на основі запропонованих методів;
- провести експериментальні дослідження розроблених засобів.

Об’єкт дослідження – процес паралельної обробки даних для підвищення швидкодії використання великих бази даних у програмно-інформаційній системі.

Предмет дослідження – методи та засоби реалізації процесу паралельної обробки даних для підвищення швидкодії використання великих бази даних у програмно-інформаційній системі.

Методи дослідження. У процесі досліджень використовувались: методи паралельних обчислень і методи синхронізації даних, методи фільтрації, математичне моделювання і прогнозування, комп'ютерне моделювання для аналізу та перевірки отриманих теоретичних положень, методи порівняння, методи проектування.

Наукова новизна отриманих результатів.

На основі виконаних теоретичних і експериментальних досліджень вирішено прикладну проблему розробки теоретичних та практичних засад для паралельної обробки баз даних, а саме:

- приведено класифікацію методів для підвищення швидкодії роботи з великими обсягами даних

- подальшого розвитку отримав метод паралельної організації запитів великих баз даних

- запропоновано метод адаптивної оптимізації ресурсів при виконанні паралельних запитів великих баз даних

Практична цінність отриманих результатів. полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень запропоновано методи та розроблено програмні засоби паралельної обробки баз даних.

Впровадження.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Особистий внесок здобувача. Здобувачем проведено аналіз наукової літератури, розроблено метод паралельної організації запитів великих баз даних, запропоновано метод адаптивної оптимізації ресурсів при виконанні

паралельних запитів великих баз даних, проведено розробку та тестування. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто.

Апробація матеріалів магістерської кваліфікаційної роботи. Основні положення магістерської кваліфікаційної роботи доповідалися та обговорювалися на міжнародних та всеукраїнських конференціях: Міжнародна науково-практична конференція «Електронні інформаційні ресурси: створення, використання, доступ» (Вінниця, 2020), XII Міжнародна науково-практична конференція “Advancing in research and education”, 07-10 грудня 2020 Франція, Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2021).

Публікації. Основні результати досліджень опубліковано в 3 наукових працях, у тому числі 3 – у матеріалах конференцій.

Структура та обсяг роботи. Відповідно до мети і завдань дослідження структура роботи складається зі вступу, п’яти розділів, висновків та списку використаної літератури. За час роботи опрацьовано 46 літературних джерел. Зміст роботи викладено на 99 сторінках машинописного тексту.

Джерелами інформації для вирішення перерахованих вище завдань є збірники наукових праць, монографії, періодична література, підручники та довідники, періодичні фахові журнали.

1. ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1. Опис методів розробки систем паралельної обробки баз даних

Паралельні системи (ПС) характеризуються паралельним функціонуванням і взаємодією складових об’єктів. У ПС з динамічно змінюваною структурою об’єкти можуть динамічно створюватися й знищуватися, а зв’язки між ними – динамічно перекомутовуватися. Для таких систем необхідні програмні засоби адекватного опису динамічного паралелізму. У мовах паралельного програмування, реалізованих на трансп’ютерних комплексах (OCCAM, Parallel C. Parallel Fortran і т.д.), таких засобів практично немає. Основний акцент у них зроблений на розв’язанні задач із статичним паралелізмом. Найпростіші засоби динамічного паралелізму в цих мовах підтримуються апаратом паралельних галузей (Parallel Execution Threads) і процесорних ферм (Processor Farms).

Для підвищення продуктивності збільшується швидкодія послідовних архітектур, шляхом застосування більш сучасної технології на основі, наприклад, мініатюризації. Але цей підхід нереалістичний, якщо не буде здійснено науковий прорив, так як технологія досягла тієї межі, коли невелике підвищення продуктивності досягається великими затратами, що ілюструють сучасні суперкомп’ютери. Архітектура з одним процесором має межу, яка визначається швидкістю поширення електричного сигналу по фізичним лініям зв’язку між компонентами.

Альтернативою послідовним процесорам є архітектури, розробка яких базується на паралелізмі: коли дані та алгоритми розподіляються між процесорами. Концепція керуючих просторів та система програмування ПАРКС (Паралельні Асинхронні Рекурсивно Керовані Системи) створені спеціально для використання в мультипроцесорних системах. У них продуктивність лінійно збільшується при збільшенні кількості процесорів, що підключені. Для цього в

роботі використовується концепція керуючого простору, розроблена на початку 80-х років в роботах Глушкова В. М. та Анісімова А. В.

Вона дозволяє описувати архітектуру будь-якої обчислювальної системи, що складається з довільної динамічної множини паралельно асинхронно діючих процесів. Застосування керуючого простору в роботі дозволило побудувати якісну, ефективну модель системи паралельних обчислень, що дозволяє спростити розв'язання багатьох актуальних задач проектування сучасних обчислювальних систем. Віртуальний паралелізм (virtual parallelism) - це максимальна кількість логічно-незалежних процесів, які задача може породити. В той самий час фізичний паралелізм (physical parallelism) задачі – це реальна кількість процесів, що виконуються незалежно один від одного на конкретній паралельній платформі.

Паралельні обчислювальні системи можна класифікувати за типами потоків команд та даних у центральній частині обчислювальної системи. З точки зору потоку команд розрізняють чотири класи обчислювальних систем, які були наведені Флінном: SISD – Single Instruction Single Data(Одиночний потік команд, Одиночний потік даних), SIMD – Single Instruction Multiple Data(Одиночний потік команд, Множинний потік даних), MISD – Multiple Instruction Single Data(Множинний потік команд, Одиночний потік даних), MIMD – Multiple Instruction Multiple Data(Множинний потік команд, Множинний потік даних), Перший клас систем – це SISD, класичні послідовні ЕОМ з одним процесором. За принципом Дж. фон Неймана вони виконують операцію, що подана на шину команд, над операндами, що знаходяться на шинах даних. Інший важливий клас паралельних систем – це SIMD, векторні й матричні ЕОМ. В цих системах кожний процесор виконує один потік інструкцій, але розповсюджує ці інструкції для виконання процесорами даних.

Підмножини процесорних модулів можуть пропускати виконання команд, що визначається за допомогою операцій маскування. В системах такого типу реальна швидкість обробки інформації залежить від можливості навантажити

процесорні модулі. Вони складаються з тисяч процесорних елементів, що можуть виконати деяку спільну інструкцію над даними, що в них містяться. Дуже часто виникають ситуації, коли треба виконати якусь операцію над багатьма даними (наприклад, скласти дві матриці великої розмірності); у таких випадках SIMD система дає суттєвий приріст у швидкості. У системах класу MISD – процес обробки розбивається на кілька етапів, кожному з яких відповідає один з процесорних модулів. Ці модулі складають у сукупності магістраль обробки – конвеєр процесорів. Реальна швидкість обробки залежить від можливості заповнення магістралі [3].

Найбільш висока швидкість обробки досягається при виконанні великих ланок програм з однорідними операціями. При частих припиненнях лінійних ланок програм командами розгалуження швидкість обробки сповільнюється. Ці системи виконують кілька операцій над даними: наприклад, скласти два числа, зсунути результат на один розряд праворуч та помножити на третє число. Такі системи дозволяють отримати деякий виграш у виконанні, але це є досить спеціалізованими системами та не отримали значного поширення. Деякі підходи до таких систем використовуються при конвеєризації SISD структур. Системи четвертого класу MIMD – багатопроцесорні ЕОМ – є найбільш розповсюдженими і саме вони будуть досліджуватися в даній роботі. Ці системи є цілком паралельними, тобто кілька керуючих пристроїв одночасно керують виконанням кількох ланок програми. MIMD-системи представляють перспективний напрямок розвитку обчислювальної техніки: вони складаються з кількох процесорів, кожен з яких функціонує за власною програмою.

Така структура призначена для серверних завдань (адже дозволяє кільком користувачам одночасно підключатися до системи та запускати на виконання кожен власну програму), складних задач, коли потрібне розпаралелювання частин програми або для використання її як SIMD-системи. З розвитком обчислювальної техніки з метою підвищення потужності найбільш розповсюджених серед звичайних користувачів SISD-систем до них починають

використовувати підходи конвеєризації та розпаралелювання, що характерні для MIMD-систем [7].

Нові системи складаються з кількох (найчастіше – двох) процесорів та забезпечуються швидкодіючими механізмами внутрішньої комунікації між системними компонентами (пам’ять, накопичувачі, пристрої вводу / виводу). Але все ж самі системи SIMD та MIMD з тисячами процесорів мають значно більшу обчислювальну потужність та находяться поза межами конкуренції у складних наукових розрахунках. Системи SIMD та MIMD за типом зв’язку між процесорами поділяють на два підкласи: MIMD – тісний зв’язок через загальну пам’ять – багатопроцесорна ЕОМ; – розподілена обчислювальна система з нетісним зв’язком через мережу комутацій та обміном повідомленнями. SIMD – векторна ЕОМ без безпосереднього зв’язку між процесорними елементами; – матрична ЕОМ зі зв’язком між елементами через комутаційну мережу.

1.2. Порівняльний аналіз архітектур систем для роботи з базами даних

Паралельна обробка запитів (Parallel Data Query, PDQ) – це технологія, яка дозволяє розподілити обробку одного складного запиту на декількох процесорах, мобілізувати для його виконання максимально доступні системні ресурси, що значно скорочує час отримання результатів. Основні типи завдань, на яких проявляється ефект технологій PDQ такі: обробка складних запитів, що включає сканування великих таблиць, сортування, з’єднання, групування, масові вставки; побудова індексів; зберігання та відновлення даних; завантаження, вигрузка даних, реорганізація бази даних; масові операції вставки, видалення, модифікація даних [6].

Звіт або відповідь на складний запит, від якого залежить прийняття відповідального рішення, можна отримати не з плином певного тривалого часу (після нової обробки), а безпосередньо під час звичайної оперативної щоденної роботи. У зв’язку з цим, знімаються проблеми, пов’язані з обробкою та обслуговуванням (архівуванням, копіруванням) дуже великих таблиць – завдяки

фрагментаціям, паралельній обробці та можливостям виконання адміністративних дій в оперативному режимі. У результаті розширення класів потенційних додатків стає більш гнучким режим роботи ПС, причому все це досягається не на вузькоспеціалізованих, а на звичайних широко поширених апаратних платформах. Таким чином, можна говорити про нову якість, яка привноситься за допомогою технології PDQ.

Максимальна перевага результату даної технології є на багатопроцесорних платформах в умовах застосування фрагментаційних таблиць, де час виконання запитів скорочується в десятки разів; однак виграш у продуктивності досягається на однопроцесорних машинах та нефрагментованих таблицях за рахунок того, що доступ до дисків здійснюється паралельно з іншими видами обробки та за рахунок максимально повного використання пам’яті.

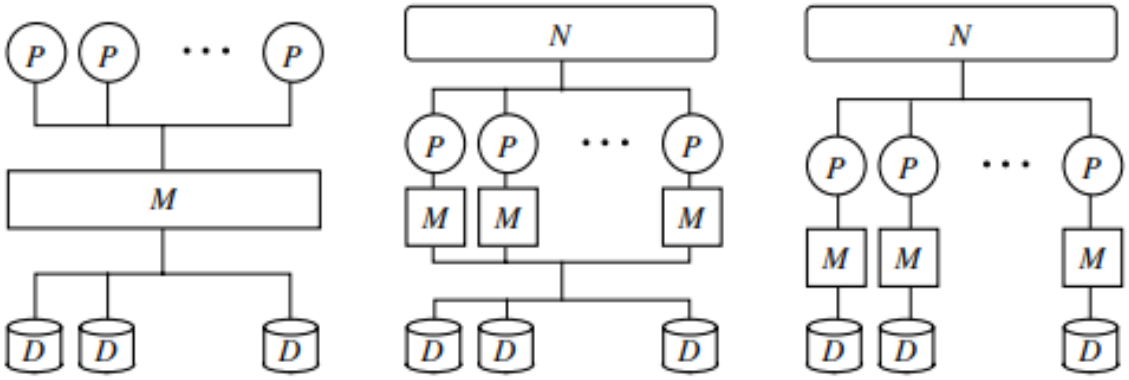
Паралельне програмування представляє додаткові джерела складності – необхідно явно керувати роботою тисяч процесорів, координувати мільйони міжпроцесорних взаємодій. Для того вирішити задачу на паралельному комп’ютері, необхідно розподілити обчислення між процесорами системи так, щоб кожен процесор був зайнятий вирішенням частини задачі. Крім того, бажано, щоб як можна менший обсяг даних пересилався між процесорами, оскільки такі комунікації значно сповільнюють операції, ніж обчислення. При цьому, часто виникають конфлікти між ступенем розпаралелювання й об’ємом комунікацій, тобто між чим більшою кількістю процесорів розподілена задача, тим більший обсяг даних необхідно пересилати між ними. Середовище паралельного програмування повинне забезпечувати адекватне керування розподілом і комунікаціями даних [9].

Порівняльний аналіз SE, SD та SN архітектури було зроблено Stonebraker. Цей аналіз показав що, з позиції масштабної високої продуктивності систем баз даних, архітектура SN є найбільш оптимальною з цих трьох архітектур.

Класифікація архітектур паралельних систем баз даних служить методологічною основою для багатьох дослідження, пов’язані з базами даних.

Донедавна використовувався таксономічний підхід, запропонований М. Стоунбракером для цих цілей і яка є найпопулярнішим підходом до класифікації паралельних систем баз даних.

Класифікація Stonebraker схематично показана на рис. 1.1.



a) *b)* *c)*

Рис. 1.1. Класифікація Stonebraker:

a) SE (Shared-everything) - спільна пам'ять та диски

b) SD (Shared-disk) – спільні диски

c) SN (Shared-nothing) - без спільних ресурсів

P позначає процесори, а M означає модуль оперативної пам'яті, D – це дисковий пристрій, і N – мережа зв'язку.

Архітектура SE системи баз даних, до яких кожен процесор має доступ будь-якого диску (з однаковим часом доступу) через спільний доступ пам'яті (див. рис. 1.1. a). Міжпроцесорні комунікації в ДП системі реалізовані за допомогою спільної пам'яті. Доступ до дисків у SE системи реалізується зазвичай через загальний буферний пул. Слід підкреслити, що кожен процесор в системі SE має свою кеш-пам'ять. Існує безліч паралельних систем баз даних із SE архітектура. Фактично, всі провідні комерційні СУБД мають реалізації, засновані на архітектурі SE.

Архітектура SD (див. рис. 1.1. b) включає системи баз даних, в яких кожен процесор має доступ до будь-якого диска; однак кожен процесор має власну

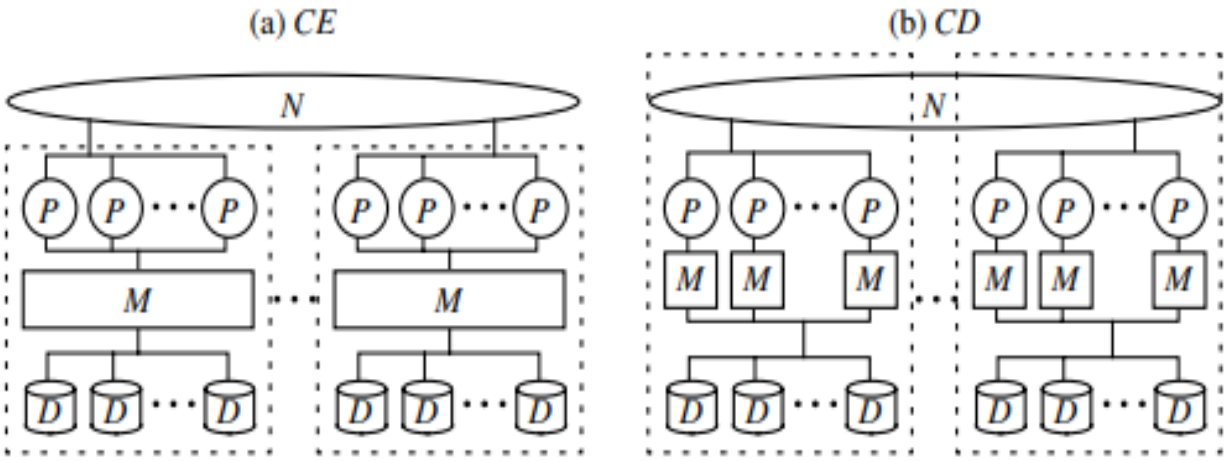
приватну пам’ять. Процесори в таких системах підключені один з одним за допомогою високошвидкісної мережі можна передавати дані. Приклади паралельної бази даних системи з архітектурою SD – це IBM IMS, Oracle Parallel Server на nCUBE [20] та VAXclusters, IBM Parallel Sysplex.

В архітектурі SN (див. рис. 1.1. с) кожен процесор має власну пам’ять і власний диск. Як і в системах SD, процесори пов’язані між собою через високошвидкісна мережа, що дозволяє організувати обмін повідомленнями між процесорами [2]. В даний час існує багато систем-прототипів та кілька комерційних систем з архітектурою SN, які використовують розділений паралелізм. На прикладах прототипу SN систем, ми можемо навести наступне ті: ARBRE, BUBBA, EDS, GAMMA, KARDAMOM [7] та PRISMA. Приклади комерційних систем з архітектурою SN це NonStop SQL, Informix PDQ [3], NCR / Teradata DBC [1], IBM DB2 PE.

Розширення класифікації Stonebraker. Класифікація Стоунбракера широко застосовувалася для аналізу архітектур паралельних систем баз даних. Однак ця класифікація в даний час розглядається як застаріла та неадекватна, адже класифікація Stonebraker не охоплює всіх різноманітність існуючих архітектур, а також ця класифікація на основі спільного використання апаратних засобів ресурси не підходить для класифікації архітектур сучасних паралельних систем баз даних.

Перший аргумент заснований на тому, що там з’явилися багатопроцесорні системи, що поєднують функції як SE, так і SN архітектур. Коупленд та Келлер запропонували розширити класифікацію Стоунбракера із введенням двох додаткових класів архітектури паралельних систем баз даних (див. рис. 1.2.): архітектура CE з SE кластерами, що об’єднані на основі принципу SN; архітектура CD із кластерами SD, об’єднані на основі принципу SN. Межі кластерів SD поширюються на загальну (глобальну) мережу зв’язку, оскільки вони можуть включати власні (локальні) мережі зв’язку, ці архітектури також називаються ієрархічними архітектурами.

Прикладами таких систем є багатопроцесорні системи серії МВС–100 / 1000 [6], багатопроцесорні системи SP2 [4] виробництва IBM, комп'ютери на основі технології ServerNet від Tandem.



a) *b)*

Рис. 1.2. Розширення класифікації Stonebraker:

a) CE (cluster everything)

b) CD (cluster disk)

Архітектура системи полягає в тому, що кластери SD на верхньому рівні ієрархії розглядаються як SN системи. Це проявляється що для кожного процесорного вузла призначається окремий диск. Такий підхід дозволяє уникнути труднощів пов'язаних з реалізацією глобальної таблиці блокування та підтримкою когерентності кешу, які є типовими для систем SD, і одночасно скористатися можливостями архітектури SD для балансування навантаження. Подібний підхід був використовується при розробці паралельної системи баз даних NonStop SQL / MP. Проведемо аналіз основних критеріїв що висуваються до використовуваних архітектур. Оцінка ефективності кожної структури буде оцінено за чотирибальною оцінкою: 0 («незадовільно»), 1 («задовільно»), 2 («добре») і 3 («відмінно»). Класифікація Stonebraker має спеціально розроблену для паралельних систем базу даних, але в даний час вона також не є цілком

адекватною. Це пояснюється тим, що існуючі підходи до класифікації базуються на відображенні архітектури паралельної системи баз даних безпосередньо до апаратної архітектури багатопроцесорної системи, як показано в таблиці 1.1.

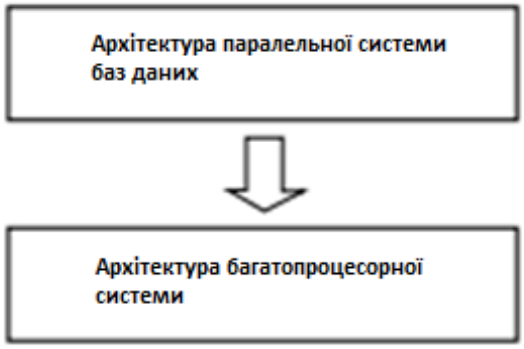


Рис. 1.3. Схематичне зображення архітектур паралельних систем баз даних на основі класифікації апаратної архітектури.

Зазначена вище проблема класифікації може бути вирішена шляхом введення додаткового рівня абстракції на основі поняття віртуальної паралельної бази даних. Архітектура паралельної системи баз даних відображається на архітектурі віртуальної паралелі баз даних, яка, у свою чергу, відображається на одній або інша апаратна архітектура мультипроцесора система. Цей підхід ми використовували для класифікації сучасних архітектур паралельних баз даних.

Таблиця 1.1 - Порівняння архітектур

	SN	CE	CD	CDN
Масштабованість	2	3	3	3
Наявність даних	2	1	3	3
Балансування навантаження	0	2	1	1
Інтерпроцесор	0	2	1	1
комунікації	3	2	0	3
Когерентність кешу	3	2	0	3
Контроль паралельності	10	12	8	14

На підставі вищенаведеного аналізу та методу визначення оцінок за різними критеріями можна зробити висновок, що архітектура CD у чистому вигляді не підходить. CE архітектура виглядає більш привабливою, ніж архітектура SN. Однак, якщо взяти до уваги всю сукупність вимог до паралельних систем баз даних ми можемо побачити, що CDN архітектура – найкраща.

1.3. Аналіз методів розв'язання поставленої задачі розробки системи паралельної обробки баз даних

Загалом виділяють тринадцять методів обчислень, виконання яких можна організувати у паралельному режимі для підвищення ефективності вирішення наукових та інженерних задач. Ці методи використовують класи, де класові взаємозв'язки визначаються за допомогою схожості в обчисленнях та переміщенні даних (табл. 1.2). Вони описуються на вищому рівні абстракції, що дозволяє об'єднати пов'язані, але досить різноманітні обчислювальні методи. Алгоритми розглядаються незалежно від розміру та типів даних.

Програма складається з міні моделей, кожна з яких обчислює певну частину даних. Тому продуктивність великої програми залежить не тільки від ефективності кожної міні моделі, а також від того, як ці моделі об'єднуються разом на платформі. Набір міні моделей, який включає прикладна програма, може бути розподілений на багатопроцесорній платформі двома шляхами:

- з розподілом часу на загальному наборі процесорів.
- з розподілом простору, при якому кожна модель однозначно займає один або більше процесорів.

Вибір часового або просторового розподілення частково залежить від структури зв'язків між моделями. Наприклад, деякі прикладні програми побудовані як кількість послідовних стадій, де кожна стадія є міні моделлю, яка повинна бути завершена перед початком наступної. В цьому випадку слід

використовувати часове мультиплексування для призначення повного набору процесорів для кожної стадії. Інші прикладні програми можуть бути побудовані як мережа суміжних міні моделей, які працюють одночасно, в такому випадку природнім є просторове розподілення моделей серед наявних процесорів. Можливим є використання двох форм розподілення у ієрархічному режимі. Наприклад, міні модель може бути реалізована як конвеєр, де обчислення для вхідних даних розбиваються на стадії, при цьому кожна стадія працює на його власному просторовому розподіленню процесорів. Кожна стадія обробляє послідовні вхідні дані з використанням часового мультиплексування.

Проведемо аналіз головних типів паралельності бази даних. Паралельність використовується для підтримки прискорення, коли запити виконуються швидше, оскільки надається більше ресурсів, таких як процесори та диски. Паралелізм також використовується для забезпечення масштабування, де зростаючі робочі навантаження управляються без збільшення часу відгуку через збільшення ступеня паралельності. Різні архітектури для паралельних систем баз даних – це спільна пам’ять, спільний диск, спільне ніщо та ієрархічні структури. На даний час розрізняють такі типи паралелізму:

1. Горизонтальний паралелізм: Це означає, що база даних розділена на кілька дисків, і паралельна обробка відбувається в межах певного завдання (тобто сканування таблиці), яка виконується одночасно на різних процесорах проти різних наборів даних.
2. Вертикальний паралелізм: він зустрічається серед різних завдань. Всі операції запиту компонентів (тобто сканування, об’єднання та сортування) виконуються паралельно конвеєрним способом. Іншими словами, вихід від однієї функції (наприклад, об’єднання), як тільки записи стають доступними.
3. Внутрішньозапитний паралелізм. Паралелізм внутрішнього запиту визначає паралельне виконання одного запиту на декількох процесорах і дисках. Використання паралелізму внутрішнього запиту має важливе

значення для прискорення тривалих запитів. Паралелізм запитів не допомагає в цій функції, оскільки кожен запит виконується послідовно. Щоб покращити ситуацію, багато постачальників СУБД розробили версії своїх продуктів, які використовували паралелізм внутрішнього запиту. Ця програма паралелізму розкладає послідовний SQL, запит на операції нижчого рівня, такі як сканування, об'єднання, сортування та агрегування. Ці операції нижчого рівня виконуються одночасно, паралельно.

4. Міжзапитний паралелізм. У паралелізмі запитів різні запити або транзакції виконуються паралельно один одному. Ця форма паралелізму може збільшити пропускну здатність транзакцій. Час відгуку на окремі транзакції не швидший, ніж був би, якщо би транзакції виконувались ізольовано.

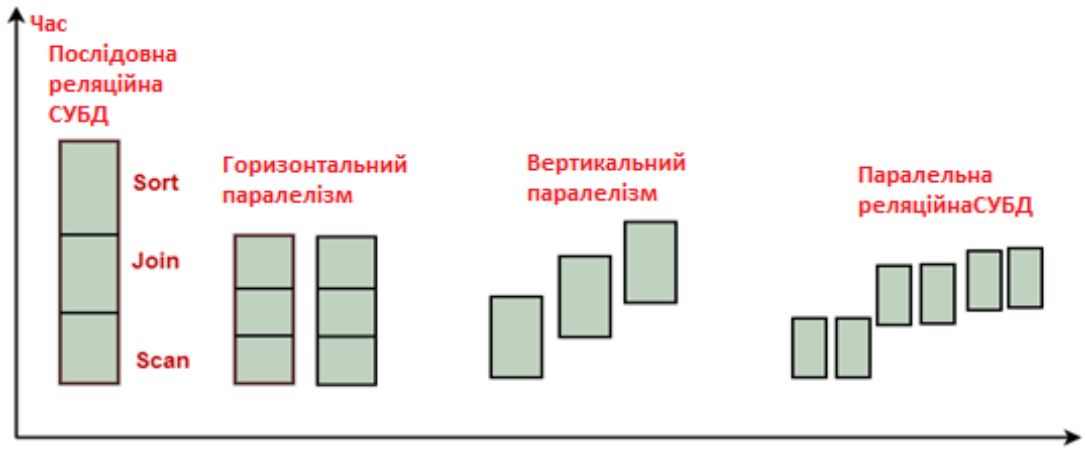


Рис. 1.4. Основні типи паралельності бази даних

Таким чином, основним використанням паралелізму запитів є розширення системи обробки транзакцій для підтримки більш значної кількості транзакцій в секунду. Постачальники баз даних почали користуватися перевагами паралельних апаратних архітектур, впроваджуючи багатосерверні та багатопотокові системи, що призначені для ефективної обробки великої кількості запитів клієнтів.

Цей підхід природно призвів до паралелізму запитів, при якому різні потоки сервера (або процеси) обробляють кілька запитів одночасно. Паралелізм запитів був успішно реалізований в системах SMP, де він збільшив пропускну здатність і дозволив підтримку більш одночасних користувачів.

Проаналізуємо основні типи апаратних архітектур.

Архітектура спільного диска. Архітектура спільного диска реалізує концепцію спільного володіння всією базою даних між серверами СУБД, кожен з яких працює на вузлі системи розподіленої пам’яті. Кожен сервер СУБД може читати, писати, оновлювати та видаляти інформацію з тієї самої спільної бази даних, якій потрібна система для реалізації форми розподіленого менеджера блокування (DLM). Компоненти DLM можна знайти в апаратному забезпеченні, операційній системі та окремому програмному рівні, все залежно від постачальника системи.

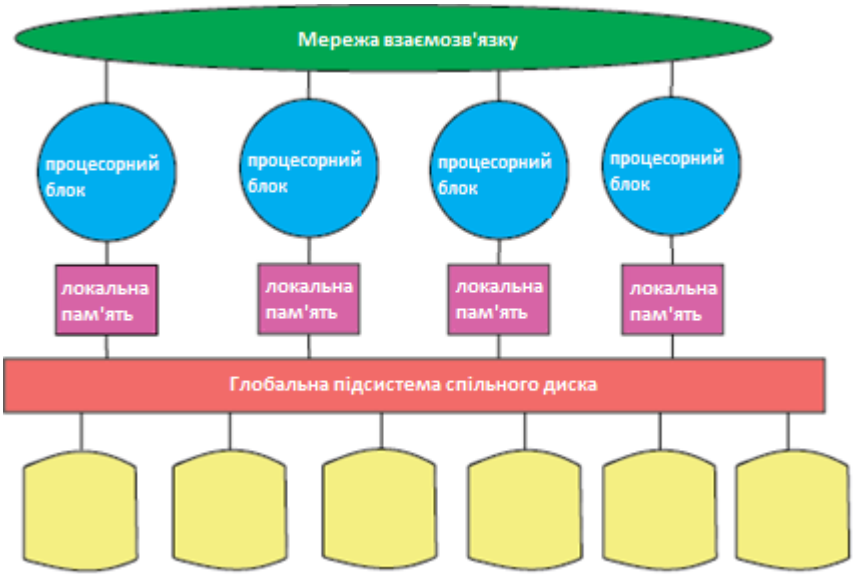


Рис. 1.5 Схематичне зображення архітектури спільного диска

Позитивним є те, що архітектури спільних дисків можуть зменшити вузькі місця в продуктивності, що виникають внаслідок перекосу даних (нерівномірний розподіл даних), і можуть значно збільшити доступність системи. Дизайн розподіленої пам’яті на спільному диску усуває вузькі місця до доступу до

пам’яті, як правило, великих SMP-систем, і допомагає зменшити залежність СУБД від розділення даних.

Стиль спільної пам’яті або спільного використання – традиційний підхід реалізації СУБД на апаратному забезпеченні SMP. Його відносно просто реалізувати, і він був дуже успішним до того моменту, коли він стикається з обмеженнями масштабованості архітектури спільного використання. Ключовим моментом цієї методики є те, що єдиний сервер СУБД може, ймовірно, застосувати всі процесори, отримати доступ до всієї пам’яті та отримати доступ до всієї бази даних, забезпечуючи таким чином клієнту єдиний образ системи.

Типи паралельності бази даних. У SMP-системах із спільною пам’яттю СУБД вважає, що множинні компоненти бази даних, що виконують оператори SQL, взаємодіють між собою, обмінюючись повідомленнями та інформацією через спільну пам’ять. Усі процесори мають доступ до всіх даних, які розділені на локальні диски.

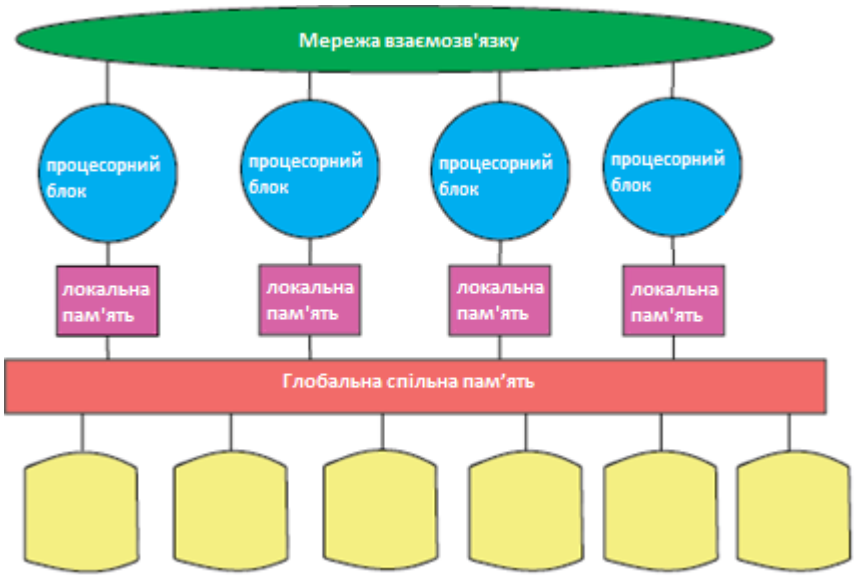


Рис. 1.6. Схематичне зображення архітектури спільної пам’яті

У розподіленому середовищі розподіленої пам’яті дані розподіляються по всіх дисках, а СУБД «розділяється» на кілька спільних серверів, кожен з яких

знаходиться на окремих вузлах паралельної системи і має право власності на свій диск, а отже його розділ бази даних.

СУБД спільного використання не паралельно виконує запит SQL на кількох вузлах обробки. Кожен процесор має свою пам’ять і диск взаємодії з іншими процесорами, обмінюючись повідомленнями та даними через мережу взаємозв’язку. Ця архітектура оптимізована спеціально для MPP та кластерних систем.

Така архітектура пропонує майже лінійну масштабованість. Кількість процесорних вузлів обмежена тільки можливостями самої апаратної платформи (і фінансовими можливостями), і кожен вузол сам може бути потужною системою SMP.

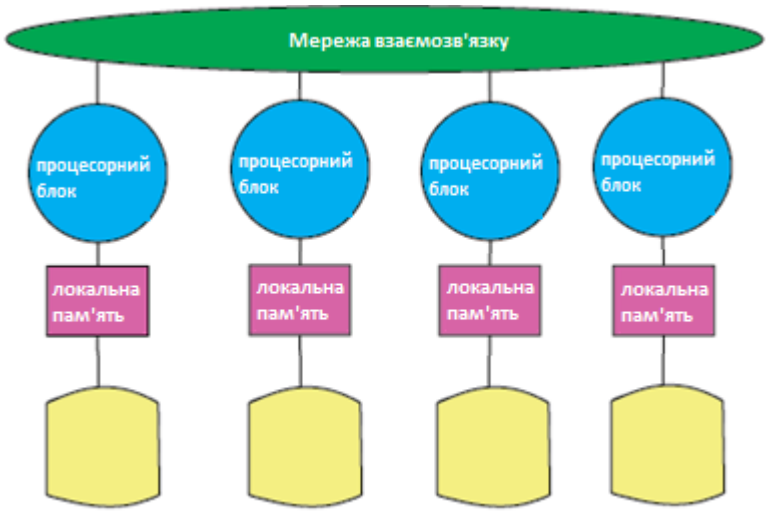


Рис. 1.7. Схематичне зображення архітектури Shared-Nothing

На підставі проведеного аналізу різних підходів до розробки паралельних програм для обчислювальних кластерів і мереж ЕОМ (MPI, HPF, OpenMP,) можна зробити наступні висновки:

1. У простоті розробки паралельних програм та їхнього повторного використання явну перевагу мають підходи HPF, OpenMP, DVM та mpC. Для оцінки складності програмування цілком підходять дані про

співвідношення кількості додаткових операторів при розпаралелювання тестів NPB 2.3. Слід зазначити при цьому, що додаткові оператори DVM-програми є простими спецкоментарями, що не залежать від розмірів масивів і числа процесорів. Додатковий код MPI-програми являє собою складну систему програм керування передачею повідомлень, що залежать від розмірів масивів і кількості процесорів. Дані для HPF й OpenMP повинні бути близькі по цьому показнику до DVM-підходу.

2. По ефективності виконання програм HPF помітно відстає від інших підходів. OpenMP обмежує можливість переносу програм, але є білш простим у використанні. Гібридні підходи OpenMP+MPI або MPICH+MPI, як самої MPI, не може забезпечити ефективного виконання програм на неоднорідних кластерах і мережах ЕОМ.
3. Найбільш гнучким, що може переноситися, і загальноприйнятим інтерфейсом програмування є MPI (інтерфейс передачі повідомлень). Однак модель передачі повідомлень по-перше недостатньо ефективна на SMP-системах; по-друге відносно складна в опануванні, тому що вимагає мислення в «специфічних» термінах.

Таким чином, жоден із підходів (MPI, HPF, OpenMP, mpC) не може розглядатися, як той, що цілком підходить для розробки паралельних програм для обчислювальних кластерів і мереж ЕОМ. Цю точку зору розділяють багато спеціалістів в області паралельних обчислень.

1.4. Постановка задачі для розробки програмно-інформаційної системи для роботи з базою даних

Перенесення послідовної програми на паралельну електронно-обчислювальну машину (ЕОМ) без її суттєвої переробки, не призводить до прискорення обчислень. Зусилля витрачені на переробку залежать від вирішуємих задач. При наявності в алгоритмі властивості паралелізму даних, одна операція може виконуватися відразу над усіма елементами масиву даних. У

цьому випадку різні фрагменти масиву можуть оброблятися незалежно на різних процесорах. Для алгоритмів цього типу розподіл даних між процесорами зазвичай здійснюється до виконання завдання на ЕОМ. Побудова алгоритму, що володіє властивістю паралелізму даних, та підбір необхідної архітектури комп'ютеру для нього можуть виконуватися з використанням досить простих методик, що не вимагають застосування складного математичного апарату [5].

В цілому технологія підготовки паралельних прикладних програм містить такі етапи:

1) аналіз та декомпозиція задачі на підзадачі, які можуть бути реалізовані в значній мірі незалежно одна від одної;
2) виділення для сформованого набору підзадач інформаційних залежностей, які мають здійснюватися в ході розв'язання поставленого завдання;
3) масштабування підзадач, визначення кількості процесорів;
4) визначення архітектури системи, закріплення підзадач за процесорами, побудова розкладу обчислень.

Загальний підхід до організації паралельних обчислень, який описаний вище, також може застосовуватися для алгоритму, який заснований на функціональному паралелізмі. В даному випадку обчислювальна задача розбивається на декілька відносно самостійних підзадач, кожна з яких завантажується на «свій» процесор. Кожна підзадача реалізується незалежно, але використовує спільні дані та обмінюється результатами своєї роботи з іншими підзадачами. Для реалізації такого алгоритму на багатопроцесорній системі необхідно виявляти незалежні підзадачі, які можуть виконуватися паралельно. Часто це виявляється далеко не очевидним та дуже важким завданням [8].

Для написання ефективного коду реалізації обчислювального процесу з функціональним паралелізмом спочатку проводиться часовий аналіз, який витрачається різними частинами програми з метою виявлення найбільш ресурсномістких частин. Це можна реалізувати за допомогою формальним моделей, наприклад, просторово-часових діаграм. Проте сучасні досліджувані

обчислювальні процеси можуть мати складний та незрозумілий характер. Для того, щоб написати правильний та ефективний код обчислювального процесу, необхідно спочатку окремо провести математичний аналіз поставленої задачі та побудувати модель опису алгоритму, що є зрозумілою як математику, так і програмісту.

Процес створення паралельних прикладних програм може бути організований більш ефективно, якщо розбити попередні етапи на ряд окремих менших підетапів.

Етап 1. Математична постановка задачі. На даному етапі задаються вихідні параметри обчислювального процесу та описуються припущення, які можуть бути присутніми при вирішенні поставленої задачі. Також аналізу підлягає оцінка обсягу даних, яка має оброблятися.

Етап 2. Побудова ациклічного орієнтованого графу обчислювального процесу для опису множини операцій алгоритму та існуючих між ними інформаційних залежностей. Модель графу алгоритму описується як:

$$G = (V, E), \qquad (1.1)$$

де V = {1,...,|V|} – множина вершин графу операцій алгоритму

E – множина ребер логічних послідовностей виконання операцій.

Ребро Ei, j = (i, j) належить графу тільки в тому випадку, якщо операція j використовує результати виконання операції i. Якщо ребрам графу присвоїти деякі висові коефіцієнти, що відображають інтенсивність інформаційного обміну між двома сусідніми вершинами, то такий граф є зваженим. Властивість ациклічності графу алгоритму полягає в тому, що ніяка величина не може знаходитися через саму себе. В загальному випадку грав обчислювального процесу може бути мультиграфом, коли будь-які дві вершини можуть бути пов’язані декількома ребрами.

Етап 3. Приведення графу до строго паралельної форми. Цей процес можна виконати двома шляхами. По-перше, з використанням алгоритму перебору вершин з інцидентними ребрами. Його суть полягає в наступному. Одиницею помічається будь-яка кількість вершин графу, які не мають попередніх вершин. Потім ці помічені вершини видаляються разом з інцидентними ребрами. Далі помічаються двійкою наступні вершини, які не мають попередніх та видаляють разом із інцидентними ребрами. Цей процес продовжується до кінця обробки усього графу. Оскільки на кожному етапі помічається лише не менш однієї вершини, число індексів не перевищує число вершин графу. Група вершин з однаковими індексами називається ярусом, число вершин у групі – шириною ярусу, а число ярусів – висотою паралельної форми [17].

Якщо граф обчислювального процесу складається з великої кількості вершин можна застосувати інший метод приведення графу до строго паралельної форми. Він заснований на використані матриці суміжності:

1) будується матриця суміжності;
2) в черговий ярус вибираються вершини з нульовою напівступеню заходу, вони відповідають нульовим стовпцям, рядки, які відповідають обраним стовпцям, також дорівнюють нулю;
3) другий етап повторюється до тих пір, поки всі вершини не будуть упорядковані.

Етап 4. Аналіз отриманого графу з використанням відомих алгоритмів. Граф, отриманий в результаті попередніх перетворень, може бути використаний для отримання кількісних характеристик, перш за все пов’язаних з часом, які є важливими при подальшій реалізації алгоритму. Наприклад, доцільним може бути розрахунок критичного шляху в графі. Даний алгоритм схожий на алгоритм Дейкстрі з поправкою того, що шукається шлях не мінімальної, а максимальної довжини. Також для зручності дослідження графу можливим є побудова матриці слідування, з подальшим її перетворенням, а саме доповненням транзитивними

зв’язками для виявлення гілок зв’язаних операторів, які не можуть виконуватися паралельно, та виявлення контурів у графі.

Етап 5. Якщо архітектура обчислювальної системи для вирішення поставленої задачі визначена, то будується розклад обчислювального процесу. Задається наступна множина:

$$Hn = \{(i, Pi, ti): i \in V\} \qquad (1.2)$$

Кожній операції $i \in V$ відповідає номер процесору Pi, який використовується для її виконання, а також час початку виконання операції ti. Для того, щоб розклад міг реалізовуватися, при завдані множини Hn необхідно дотримуватися наступних вимог:

1) один і той самий процесор не може бути призначений різним операціям в один і той же момент часу:

$$\forall i, i \in V: ti = tj \Longrightarrow Pi \neq Pj \qquad (1.3)$$

2) до призначеного моменту виконання операції усі необхідні дані вже повинні бути визначеними:

$$\forall (i, j) \in R \; ti \geq ti + \tau \qquad (1.4)$$

де τ час виконання однієї операції

Граф обчислювальної схеми алгоритму (1.1) разом з розкладом (1.2) може розглядатися як модель паралельного алгоритму An (G, Hn), який реалізується з використанням n процесорів.

Етап 6. На основі отриманої моделі пишеться програмний паралельний код з можливим використанням таких технологій як OpenMP, MPI в залежності від архітектури обчислювальної системи та виконується налагодження програми. Під час математичного аналізу обчислювального алгоритму важливим етапом може бути оцінка характеристик продуктивності, яка виконується за допомогою існуючих моделей, що залежать від обраної архітектурної реалізації.

У мовах паралельного програмування, реалізованих на трансп'ютерних комплексах (OCCAM, Parallel C. Parallel Fortran), таких засобів практично немає. Основний наголос в них зроблений на розв'язання задач зі статичним паралелізмом. Найпростіші засоби динамічного паралелізму в цих мовах підтримуються апаратом паралельних галузей (Parallel Execution Threads) і процесорних ферм (Processor Farms). Система програмування ПАРКС, як і система паралельного програмування на трансп'ютерах це сукупність програмних засобів, що підтримують процес розробки і реалізації алгоритмів паралельної обробки інформації за рахунок розширення базових алгоритмічних мов (JAVA, C, Pascal, Fortran, Modula – 2).

Відмінність системи програмування ПАРКС полягає в тому, що пропоновані нею програмні засоби сполучать найбільш могутні алгоритмічні концепції – рекурсію і паралельність – і призначені для опису 26 взаємодій зі змінною комутацією зв'язків і рекурсивно-паралельного розвитку процесів, ядра яких описуються на основі конструкцій базової мови. Акцент робиться на програмні засоби опису динамічного паралелізму. Для CSP-моделі паралельних обчислень (яка лежить в основі мов OCCAM і Ada) ПАРКС – засоби забезпечують динамічне створення і знищення паралельних процесів (з використанням рекурсії по даним і рекурсії по керуванню), динамічну перекомутацію зв'язків між процесами. Іншим важливим критерієм класифікації ПС. крім статичності / динамічності, на нашу думку, варто вважати відкритість або закритість програмного забезпечення.

Ряд розробок і технологій (наприклад, Ада, ОССАМ, МАЯК) пропонують як базу заново розроблену мову (як правило, дуже складну), змушуючи користувачів її освоювати. Це позиція закритих систем і технологій. Сильною стороною такого підходу є можливість досягнення високої ефективності виконання для визначених класів задач. Альтернативні відкриті системи і технології (наприклад, LINDA, ПАРКС, Parallel Fortran) надають користувачу можливість працювати в найбільш придатному для його задач середовищі, доповнюючи його тими або іншими засобами паралелізму.

Крім комфортності для користувача у відкритих системах часто спрощуються і семантичні проблеми, тому що вони концентруються винятково на паралелізмі.

Моделі керуючого простору та топології мереж ПАРКС – модель визначається в такий спосіб: а) виділяються кінцеві набори базових АМ, що описують способи розпаралелювання алгоритмів на різних рівнях деталізації (операцій, операторів, процедур, процесів і т. п.); б) визначаються правила створення, знищення й взаємодії АМ на всіх рівнях деталізації; в) уточнюються поняття рекурсії за даними й рекурсії по керуванню для заданої базової алгоритмічної мови. ПАРКС – модель, фактично, і задається відповідною моделлю КП.

В свою чергу, модель КП утворює топологію певної мережі. Кроком підвищення обчислювальних потужностей ЕОМ є їх об’єднання в паралельні комплекси, мережі. На сьогоднішньому етапі розвитку електронно-обчислювальних систем велика увага приділяється створенню програмного забезпечення не тільки для однопроцесорних ЕОМ, а й для мереж великого розміру. Серед них можна виділити мережі безпосереднього зв’язку, багатостанові розподілені мережі. Для великої кількості існуючих топологій мереж розробляються алгоритми для дослідження задач стійкості, маршрутизації [2, с. 31], надійності та коректності їх роботи. Поширеною

задачею є також пошук сприятливої топології, на якій можна отримати розв’язок певної задачі.

Швидкість багатопроцесорного комплексу та ефективність паралельної обробки даних повністю залежить від методів комутації елементів мережі. Схема повного з’єднання є найкращою для комунікації, оскільки для передачі повідомлення між двома довільними процесорними елементами потрібен лише один такт. Але непрактичність цієї топології полягає у її вимірності: для сполучення n процесорних елементів необхідно використати n 2 зв’язків. Тому схему повного з’єднання для великих мереж не використовують. Схема сполучення зіркою має перевагу в тому, що вона має малу кількість зв’язків і для передачі інформаційного пакета досить використати два з’єднання, але вада її полягає в існуванні вузького горла – центрального процесора, який буде задіяний майже при будь-якій трансакції. Вузьке горло має і деревоподібна мережа, хоча вона має сприятливу структуру для вирішення багатьох задач. Деякі з наведених мережевих архітектур, реалізовані у вигляді моделей КП за допомоги мови ПАРКС-JAVA і розглядаються у п’ятому розділі цієї роботи

1.5. Висновки

Сьогодні, у конкурентному світі, підприємства всіх видів використовують і залежать від своєчасності доступна, актуальна інформація. Обсяги інформації зростають на 25–35% року, і прогнозується, що традиційний курс транзакцій зросте в кілька разів протягом наступних п’яти років – вдвічі більше, ніж нинішня тенденція зростання мейнфреймів [19].

Таким чином, складні запити, такі як макроси, що генеруються системами підтримки прийняття рішень або генеровані системою, як у виробничому контролі, збільшаться до значного попиту пропускна здатність з прийнятним часом відгуку. Крім того, дуже складні запити на дуже великі бази даних, створені кваліфікованим персоналом або експертними системами, можуть

пошкодити пропускну здатність, вимагаючи належного часу відгуку. З точки зору бази даних, проблема полягає у створенні бази даних сервери, які ефективно підтримують усі ці типи запитів на, можливо, дуже великих онлайнові бази даних. Однак вражаючі вдосконалення технології кремнію один не може йти в ногу з цими зростаючими вимогами.

Тому рішення полягає у використанні широкомасштабного паралелізму для збільшення вихідної потужності окремих компонентів, інтегруючи їх у цілісну систему разом із відповідне програмне забезпечення для паралельних баз даних. Використання стандартних апаратних компонентів є важливо використовувати постійне вдосконалення технології з мінімальними затримками. Тоді програмне забезпечення баз даних може використовувати три форми паралелізму, властиві при інтенсивних робочих навантаженнях додатків. Паралелізм запитів уможливлює паралельність виконання декількох запитів, породжених одночасними транзакціями. Інтраквест паралелізм робить паралельне виконання декількох незалежних операцій (наприклад, вибрати операції), можливо в межах одного запиту. І запит, і запит паралелізм можна отримати за допомогою розділення даних. Нарешті, з інтраопераціею паралелізм, одну і ту ж операцію можна виконати, використовуючи стільки субопераций функціональне розділення на додаток до розділення даних. Орієнтований на множини режим мови баз даних (наприклад, SQL) надає багато можливостей для внутрішньої операції паралелізм. Наприклад, продуктивність операції з’єднання може бути збільшена суттєво паралелізмом.

В даному розділі нами встановлено, що етапи проектування представляють собою ієрархію віртуальних машини, такі, що реалізує кожна поточна машина попередньо. Для класифікації та порівняння архітектури сучасних паралельних систем баз даних, ми проаналізували запропоновану класифікацію Стоунбракера в середині 1980-х. Доведено, що ця класифікація залишається найбільш підходящою для систем баз даних; однак вона потребує певного вдосконалення та розширення. Однак слід відмітити, що можливе вдосконалення та розширення

класифікації Стоунбракера за рахунок віртуальної паралелі абстракція машини баз даних та введення додаткових класів ієрархічних архітектурних кластерів.

На основі розширеної Stonebraker класифікації, проведено порівняльний аналіз сучасних архітектур паралельних систем баз даних. Такий аналіз показав, що гібридна архітектура CDN має найкращі показники.

2. РОЗРОБКА ПРОГРАМНОЇ АРХІТЕКТУРИ ПРОГРАМНО-ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ РОБОТИ З БАЗОЮ ДАНИХ

2.1. Специфіка паралельної обробки великих потоків даних

Паралельна система баз даних – це система управління базами даних (СУБД), реалізована на багатопроцесорній системі з високим ступенем зв'язку. Багатопроцесорна система з високим ступенем підключення – це система, що містить безліч процесорів і дисків, в яких процесори пов'язані між собою за допомогою такої мережі зв'язку, як мережа передачі даних. Час обміну порівнянний з часом даних обмін з диском. Це визначення виключає з розглянуті розподілені СУБД, реалізовані на декількох незалежних комп'ютерах, підключених через локальний або / та глобальних мереж. Останні системи мають свої власні специфічні особливості (наприклад, такі, що пов'язані з різними географічними розташуваннями комп'ютерів, локальні автономність та неоднорідність програмного та апаратного забезпечення [1]), а також проблеми, пов'язані з великою кількістю процесорних вузлів зазвичай не розглядаються для таких системи. Однак існує широкий спектр систем – починаючи від традиційних однопроцесорних СУБД перенесений на симетричні багатопроцесорні системи, які використовують лише паралелізм між транзакціями, який закінчується складними паралельними системами, реалізованими на кластерах або мультипроцесорах з масивним паралелізмом, що використовують секціонований паралелізм [2] – що відповідає наведеному вище визначенню.

Перелічимо основні вимоги до паралельних систем баз даних. Критерії, що використовуються при порівнянні архітектур паралельних систем баз даних покладаються на такий набір вимоги як: (1) достатня масштабованість, (2) висока доступність даних, (3) ефективне вирівнювання навантаження, (4) низька вартість міжпроцесорних бірж, (5) низькі накладні витрати на забезпечення узгодженості кеш-пам'яті, (6) ефективна організація контролю паралельності.

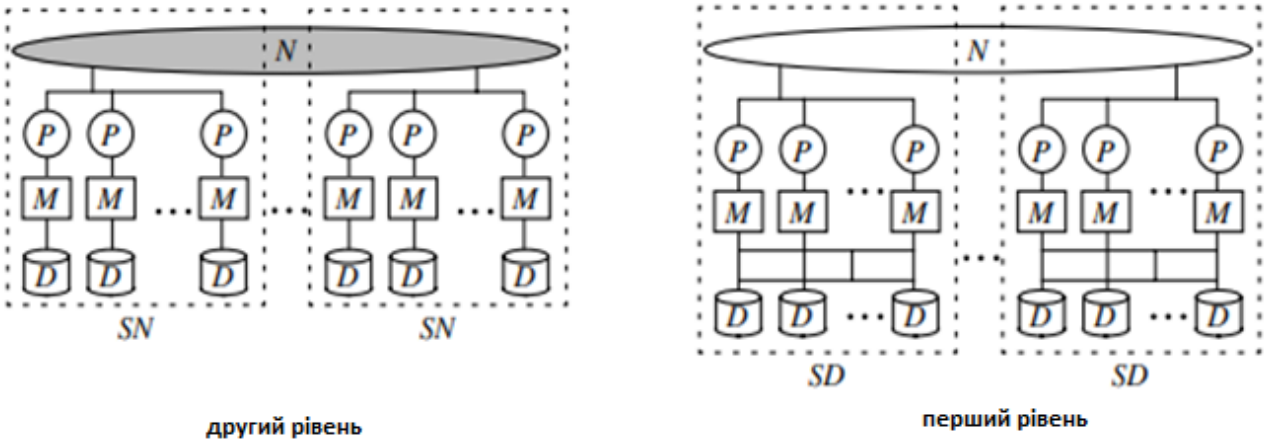


Рис. 2.1. Гібридна архітектура CDN.

Розглянемо зазначені критерії більш детально.

Масштабованість. Можливість динамічного накопичення адаптації до зростаючого розміру бази даних або збільшення вимог до продуктивності і це є важливою властивістю паралелі платформи [3]. Ця особливість досягається поступовим включенням додаткових процесорів, модулів пам’яті, дисків та інших апаратних компонентів системи. Цей процес називається масштабуванням системи. Якщо апаратна потужність системи подвоюється, то очікується також подвоєння її продуктивності, однак на практиці реальне збільшення продуктивності значно більше. Наприклад, масштабованість систем ПЕ є обмежено 20–30 процесорами [5]. З подальшим вдосконалення системи SE, її продуктивність зростає дуже повільно або навіть починає падати [3]. Це пояснюється тим, що процесори проводять багато часу в очікуванні для доступу до спільних ресурсів. Отже, масштабованість будь-якої багатопроцесорної системи визначається ефективністю розпаралелювання.

Ефективність паралелізації описується термінами двох основних якісних характеристик: прискорення та масштабування. Архітектура багатопроцесорної системи вважається добре масштабованою, якщо вона демонструє майже лінійне масштабування та пришвидшення. Лінійне масштабування передбачає, що час, витрачений системою на розв’язання задачі рівний витраченому подвійною

системою для розв’язування подвійної задачі. Лінійне прискорення означає, що подвійна система вирішує проблему вдвічі швидше, ніж оригінальна система. Основним фактором, що погіршує масштабованість систем, є наслідки недоліків, які пов’язані з одночасним доступом до спільних ресурсів процесорами.

Наявність даних. Одна з критичних характеристик паралельних систем баз даних – це здатність системи забезпечувати високий ступінь доступності даних в рамках стан відмов деяких апаратних компонентів. В даний час існує кілька підходів до класифікації паралельних обчислювальних систем.

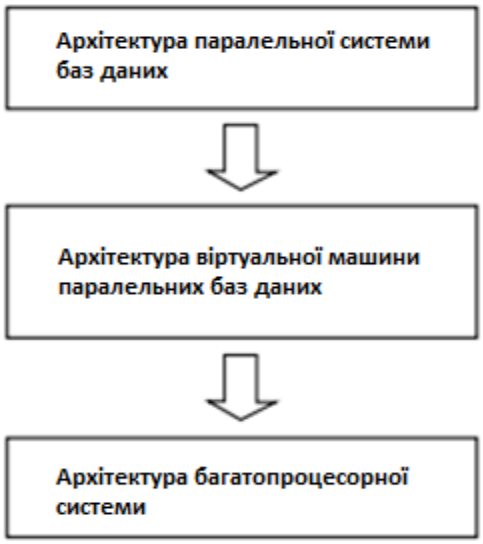


Рис. 2.2. Класифікація заснована на понятті віртуальної машини паралельних баз даних.

На основі побудованої класифікації проведено якісний порівняльний аналіз різних архітектур.

Віртуальні паралельні бази даних. Побудова віртуальної паралельної машини баз даних відбувається з таких віртуальних пристроїв: віртуальні процесори, модулі віртуальної пам’яті, віртуальні диски та віртуальні мережа зв’язку.

Віртуальний процесор – це віртуальний пристрій, що використовується для виконання окремого завдання, визначеного як процес бази даних. Типовим

прикладом процесу баз даних є запит або агент запиту (якщо використовується розділений паралелізм). В реальній системі, віртуальний процесор може бути представлений мікропроцесор або модуль процесора. Якщо кілька процесів баз даних виконуються на одному фізичному процесорі в режимі розподілу часу, то кажуть, що цей фізичний процесор реалізує кілька віртуальних процесів.

Модуль віртуальної пам’яті – це віртуальний пристрій, що використовується для буферизації об’єктів бази даних. Типовим прикладом об’єкту реляційної бази даних є відношення до його фрагменту (якщо використовується розділений паралелізм). Віртуальний процесор може отримати доступ до об’єкту бази даних тільки через його образ завантаження в якийсь модуль віртуальної пам’яті, що доступний для цього процесора. Відповідно до цього, кількість модулів віртуальної пам’яті у віртуальній паралелі баз даних не може перевищувати кількість віртуальних процесорів. У реальній системі модулі віртуальної пам’яті зазвичай реалізуються як фізичні модулі оперативної пам’яті. При цьому, один фізичний модуль пам’яті може бути представлений кількома модулями віртуальної пам’яті і, навпаки, кілька фізичних модулів пам’яті можна розглядати як один віртуальний модуль пам’яті.

Віртуальний диск – це віртуальний пристрій, що використовується для зберігання об’єктів бази даних. У реальній системі віртуальний диск зазвичай реалізується як фізичний дисковий пристрій або масив дисків [11].

Віртуальна мережа зв’язку – це віртуальний пристрій забезпечення передачі даних з одного модуля віртуальної пам’яті на інший. Передачі здійснюються тільки засоби комунікативних дій відповідних віртуальних процесорів. Без втрати загальності ми можемо припустимо, що віртуальна машина паралельних баз даних не має більше однієї віртуальної комунікаційної мережі. Віртуальна паралельна машина баз даних визначається як пов’язаний граф, вузли якого відповідають різним віртуальним пристроям до потоків даних. Приклад конфігурація віртуальної паралельної машини баз даних показано на рис. 2.3.

Слід зазначити, що в даному контексті машина віртуальних баз даних є тільки певним рівнем абстракції в системній ієрархії програмних модулів СУБД та операційної системи, що реалізує систему баз даних на певній апаратній платформі.

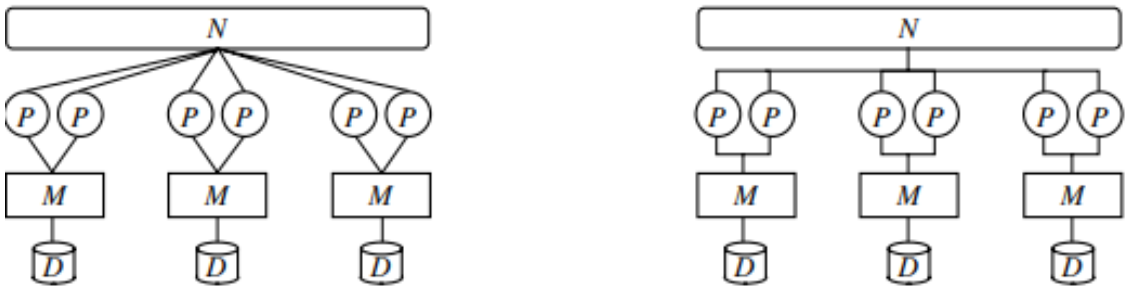


Рис. 2.3. Приклад конфігурації віртуальної машини паралельних баз даних. P – віртуальний процесор, M – віртуальна пам'ять модуля, D – віртуальний диск, N – віртуальна мережа зв'язку.

В рамках однієї системи ми можемо розглянути кілька рівнів абстракції цього виду. У віртуальній машині кожен поточний рівень реалізований на основі функцій та послуг, що реалізує віртуальна машина на попередньому рівні. Щоб краще пояснити це, розглянемо наочний приклад на рис. 2.4.

Використовуючи віртуальну машину V0, можемо створити програмний комплекс I1, що реалізує віртуальний диск із зберігання, що рівний вільному дисковому простору всіх дисків фізичної системи. У цьому випадку, I1 реалізує віртуальна машина V1, в якій кожен процесор має свою пам'ять, але всі процесори мають спільний дисковий простір.

Іншими словами, I1 визначає відображення V0 на V1. Подібним чином, у середовищі V1 можемо створити програму I2, що реалізує загальну віртуальну пам'ять розміром, що рівна загальному вільному адресному простору всіх фізичних модулів пам'яті. В результаті ми будуємо відображення від конфігурації V1 до конфігурації V2 в які всі процесори мають спільну пам'ять і спільний простір на диску. Зробивши це, можемо завершити реалізацію СУБД в

контексті конфігурації V2. Отримана система баз даних має гібрид архітектура в наступному розумінні: на фізичному (нульовому) рівні, це система без спільних ресурсів; на логічному (першому) рівні це система із спільними дисками; і, далі на віртуальному (другому) рівні ми маємо систему із загальною пам'яттю та дисків. Звичайно, конфігурація зображене на рис. 2.4 є нежиттєздатною через труднощі, які пов'язані з неефективністю доступу до такої віртуальної пам'яті та віртуального диска.

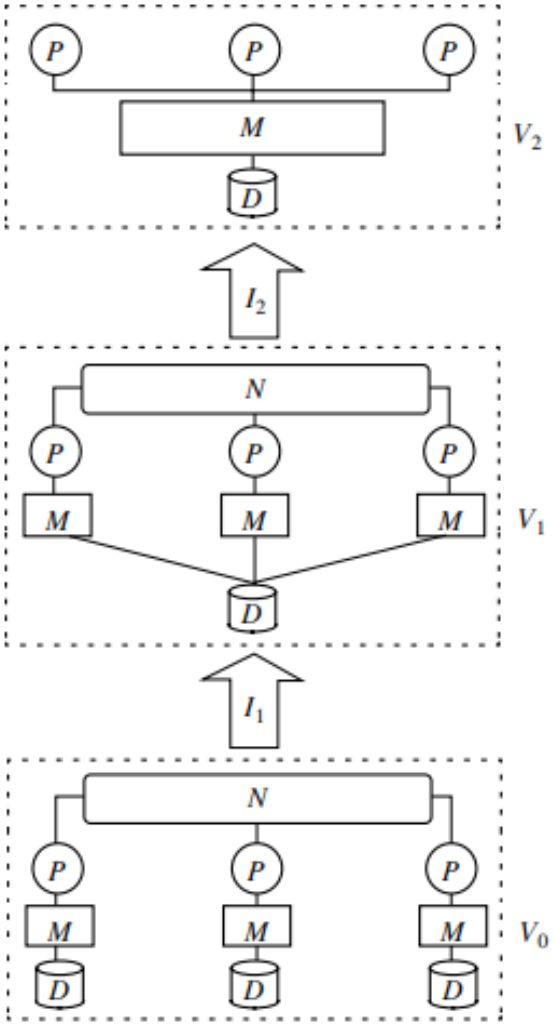


Рис. 2.4. Приклад, що ілюструє ієрархію машин віртуальних баз даних. P – віртуальний процесор, M – віртуальна пам'ять модуля, D – віртуальний диск, N – віртуальна мережа зв'язку.

2.2. Методика паралельної обробки великих потоків даних в базах даних

Методика побудови паралельних методів розв’язання складних завдань є однією з основних у паралельному програмуванні і широко використовується на практиці.

Існують різні форми паралелізму. На рис. 2.5 показано чотири простих види паралелізму графічно. Приведемо кілька властивостей паралелізму для вирішення проблеми інтенсивної обробки даних:

- Розділення проблем. Поділ на самостійні підзадачі дає незалежний паралелізм. Встановлені відображення природно адаптуються до незалежного паралелізму (та сама інструкція застосовується до кожного елементу набору), тоді як відображення потоків адаптуються до паралельності трубопроводу (деякі інструкції послідовно застосовуються до кожного елементу потоку). Таким чином, набори та потоки пропонують певний формат для задання паралельності;
- Обчислення згідно постулату «розділяй і володарюй» можна представити, комбінуючи їх типи паралелізму. «Розділення» проблеми представляють вузли, що розширюються в графі, що збирають результати у набір (із незалежним паралелізмом), потік (із конвеєрним паралелізмом) та сукупністю.

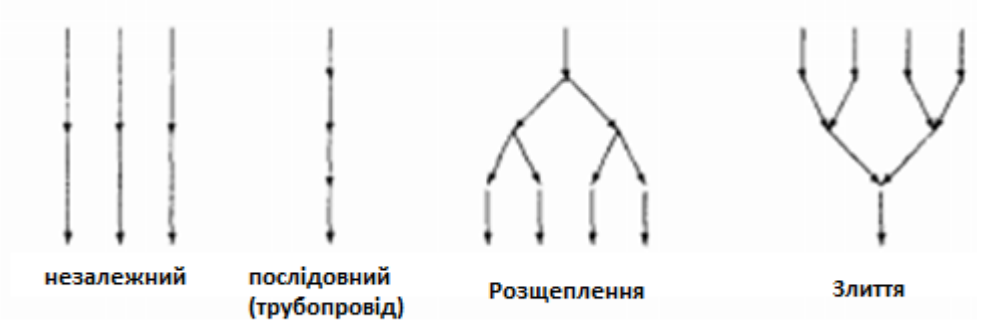


Рис. 2.5. Основні типи паралелізму

Оператори реляційної алгебри часто можна виразити як обчислення «розділяй і володарюй». Перша проблема полягає в тому, що реляційна модель не дає можливості говорити про порядок серед даних (наприклад, відсортовані відносини або упорядковані кортежі). Реляційні мови тому є недостатніми для вказівки «потокової обробки», в якій послідовності даних обробляються послідовно [8]. Отже, потоків бути не може. Паралельність трубопроводу зазвичай використовується, прозоро для користувача, на нижчому рівні мови, що реалізують реляційну алгебру (наприклад, PLERA [9] або PFAD [10]).

Друга проблема полягає в тому, що паралельна обробка даних вимагає ефективних можливостей розподілу даних. Як правило, реляційний запит (вираз select-project-join) перекладається у форму низького рівня реляційної алгебри з явною (низькорівневою) паралельністю конструкції. Розбиття даних використовується для поширення обчислення операторів реляційної алгебри серед паралельних процесорів. Це розділення є типовим і визначається під час фізичного проектування бази даних, а потім використовується компілятором.

Здебільшого для розподіленого обчислення процесори обмінюються проміжними результатами для обчислення кінцевого результату. В ідеалі розділення даних має бути вираженим програмістом або компілятором у паралельній мові бази даних. Це важливо для автоматичного вилучення паралелізму і має призвести до ефективних реалізацій на паралельних системах баз даних.

Проведення паралельних обчислень відносно зв’язків часто вимагає зазначення способу передачі даних, згідно якого буде здійснено розподіл (паралелізм фан-аут) та спосіб розподілу результатів які будуть зібрані. Моделі баз даних дозволяють виражати як упорядкування між кортежами, так і розділення даних. Мова FAD Bubba має оператори, що виражають різні форми фан-аут і паралелізм. FAD – це орієнтований набір мовою бази даних на основі функціонального програмування та реляційної алгебри. Це забезпечує фіксований набір функцій вищого порядку для сукупних функцій, таких як

параметризований сукупний оператор та оператор групування. Оператор насоса застосовує уніарну функцію до кожного елемента набору, створюючи проміжний набір, який потім «зводиться» до одного даного за допомогою двійкової функції, що поєднує проміжні набори елементів. Оператор групи дозволяє встановити розділення.

Модель SVP йде на крок далі, дозволяючи набори, потоки та паралелізм, який буде зафіксовано в єдиній системі, що формалізує розуміння постулату «розділяй та володарюй». Існує колекції моделей SVP, спеціалізація яких веде до наборів або потоків, як послідовно-паралельної графіки, що полегшує вираження паралельної обробки даних. Перетворювачі, узагальнюють сукупні операції та набір або потокові операції. Вони призводять до високого рівня специфікації незалежних даних. Таким чином, SVP є можливою формальною основою для подальшого розвитку дослідження паралельних мов програмування даних.

Опишемо порядок організації паралелізму на основі поділу даних. При побудові паралельних способів виконання матричного множення поряд з розглядом матриць у вигляді наборів рядків і стовпців широко використовується блочне представлення матриць. Виконаємо більш докладний розгляд даного способу організації обчислень.

Нехай кількість процесорів складає $p = k^2$, а кількість рядків і стовпців матриці є кратним величині $k = \sqrt{p}$, тобто $n = mk$. Уявімо початкові матриці A, B і результуючу матрицю C у вигляді наборів прямокутних блоків розміру $m \times m$. Тоді операцію матричного множення матриць A і B в блоковому вигляді можна представити таким чином:

$$A11A12…A1k…Ak1Ak2…Akk×B11B12…B1k…Bk1Bk2…Bkk=C11C12…C1k…C$$

де кожен блок C_{ij}матриці C визначається відповідним виразом

1

Інформаційний граф алгоритму множення при подібному представленні матриць зображений на рис. 2.6. При аналізі цього графа можна звернути увагу на взаємну незалежність обчислень блоків C_{ij} матриці C. Як результат, можливий підхід для паралельного виконання обчислень може складатися у виділенні для розрахунків, пов'язаних з отриманням окремих блоків C_{ij}, різних процесорів. Застосування подібного підходу дозволяє отримати багато ефективні паралельні методи множення блочно-представлених матриць; один з алгоритмів даного класу розглядається нижче.

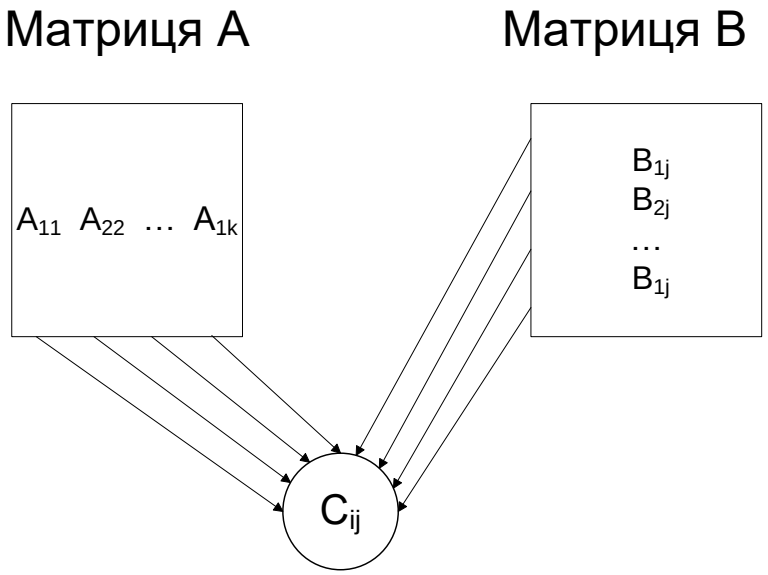


Рис. 2.6. Інформаційний граф матричного множення при блочному поданні матриць

Для організації паралельних обчислень припустимо, що процесори утворюють логічну прямокутну решітку розміром $k \times k$.

Для пояснення наведених правил паралельного наведено стан блоків на кожному процесорі в ході виконання ітерацій етапу обчислень (для процесорної решітки 2×2). Наведений паралельний метод матричного множення приводить до рівномірного розподілу обчислювального навантаження між процесорами

$$T_p = \frac{2n^3}{p} \qquad (2.2)$$

$$S_p = \frac{2n^3}{\frac{p(2n^3)}{p}} = 1 \qquad (2.3)$$

Разом з тим, блочне представлення матриць призводить до деякого підвищення обсягу пересилаються між процесорами даних – на етапі ініціалізації і на кожній ітерації етапу обчислень на кожен процесор передається 2 блоку даних загальним обсягом. Враховуючи виконується кількість ітерацій методу, обсяг даних, що пересилаються для кожного процесора становить величину

$$V_{ij} = \frac{2n^2}{\sqrt{p}}\left(1 + \frac{1}{\sqrt{p}}\right) \qquad (2.4)$$

Обсяг даних, що пересилаються може бути знижений, наприклад, при використанні строкового (для A) і стовпового (для B) розбиття матриць, при якому справедлива оцінка.

$$V'_{ij} = \frac{2n^2}{\sqrt{p}} \qquad (2.5)$$

Дані оцінки показують, що різниця обсягів даних, що пересилаються:

$$\frac{V_{ij}}{V'_{ij}} = \frac{\frac{2n^2}{\sqrt{p}}\left(1+\frac{1}{\sqrt{p}}\right)}{\frac{2n^2}{\sqrt{p}}} = 1 + \frac{1}{\sqrt{p}} \qquad (2.6)$$

є не настільки істотною і дана відмінність зменшується при збільшенні кількості використовуваних процесорів.

З іншого боку, використання блокового подання матриць приводить до ряду позитивних моментів. Так, при даному способі організації обчислень пересилання даних виявляється розподіленої за часом і це може дозволити

поєднання процесів передачі та обробки даних; блокова структура може бути використана для створення високоефективних методів слабо заповнених (розріджених) матриць. І головне – даний метод є прикладом широко поширеного способу організації паралельних обчислень, що складається в розподілі між процесорами оброблюваних даних з урахуванням близькості їх розташування в змістовних постановках вирішуваних завдань. Подібна ідея, часто називається в літературі геометричним принципом розпаралелювання, широко використовується при розробці паралельних методів вирішення складних завдань, оскільки в багатьох випадках може призводити до значного зниження потоків даних за рахунок локалізації на процесорах істотно інформаційно-залежних частин алгоритмів (зокрема, подібний ефект може бути досягнутий при чисельному рішенні диференціальних рівнянь в часткових похідних).

2.3. Аналіз інструментальних засобів для роботи з базами даних

Під інструментальним засобом для моделювання баз даних розуміється комп’ютерна програмна реалізація (програмний додаток), що реалізує одну певну або безліч нотацій представлення структур даних і зв’язків між ними в рамках деякої методології проектування.

На ринку інформаційних продуктів для моделювання баз даних існує досить велика кількість спеціалізованих інструментальних засобів, серед яких можна виділити наступні:

- CA ERWin Data Modeler – засіб, орієнтоване на розробки логічної і фізичної моделей даних з виконанням верифікації по нотації IDEF 1x і формуванням звітів по сформованим моделям даних (виробник: З А Technologies (США));
- IBM InfoSphere Data Architect – засіб, спрямоване на управління даними на логічному і фізичному рівнях з можливістю інтеграції з СУБД (виробник: IBM (США)).

Крім зазначених програмних засобів на ринку представляється ряд інших програмних засобів, серед яких: MySQL Designer, PowerDesigner і ін. Також кошти моделювання часто представляються як інструмент СУБД, як, наприклад, це реалізовано в SQLServer або Oracle Database. Такі інструментальні засоби, орієнтовані переважно на роботу з самою базою даних, дають тільки базові можливості моделювання, але бувають досить корисні, коли необхідно працювати з елементами бази даних, використовуючи візуальні механізми і інструменти.

Для вирішення завдань розробки баз даних, реалізуючи етапи моделювання на концептуальному, логічному і фізичному рівнях, розробники застосовують саме спеціалізовані інструментальні засоби. Склад цих кроків обумовлений тим, що складноорганізовані структури даних не завжди доцільно представляти в рамках єдиного простору моделі і його потрібно розділяти на окремі робочі області. В результаті, на першому кроці розробник формує склад робочих областей (діаграм), відповідних виділеним функціям роботи з даними в моделях бізнес-процесів предметної області. Подальше моделювання бази даних виконується в рамках єдиного простору моделі, але на рівні окремих робочих областей (діаграм).

Сам процес моделювання передбачає попереднє побудова логічної моделі бази даних з подальшою її нормалізацією і переходом до фізичному уявленню моделі, на основі якої виробляють додаткове уточнення окремих параметрів майбутньої бази даних, з огляду на логічне і фізичне уявлення структур даних. В результаті всього моделювання розробником формується для бази даних програмний код створення необхідних структур даних (таблиць) і інших об’єктів бази даних, який може бути реалізований у відповідній вибраній системі управління базами даних.

Розробка моделей бази даних за допомогою спеціалізованого програмного засобу ERWin є невід’ємною частиною роботи розробника по створенню бази даних, особливо в складноорганізованних інформаційних системах, де база

даних буде використовувати не тільки багато даних, представлених екземплярами інформаційних об’єктів, а й багато таблиць, що представляють інформаційні структури предметної області на фізичному рівні. Використання даного інструментального засобу дозволяє розробнику уявити спроектовану модель бази даних і забезпечити переклад цієї моделі в фізичну базу даних.

Інструментальне засіб ERWin є програмним продуктом, що надає необхідні для розробника функціональні можливості по моделюванню і документування моделей бази даних з подальшою трансформацією фізичної моделі в базу даних.

При застосуванні до більш складних доменів додатків, таких як інженерія, офісні інформаційні системи та експертні системи, реляційні дані моделі мають обмеження з точки зору управління правилами, системи типів та комплексу підтримка об’єкта. Для вирішення цих питань використовується технологія – KBMS.

2.4. Розробка методу паралельного опрацювання баз даних з використання колоночних індексів

На даний час розроблено кілька паралельних архітектур. Спільна пам’ять, спільний диск та спільне використання – це основні архітектури паралельних систем баз даних. Кожна архітектура має деякі переваги та недоліки. Гібридні архітектури використовуються, щоб поєднати достоїнства архітектур та подолати їхні індивідуальні недоліки. Для порівняння паралельних систем баз даних потрібно визначити критерії порівняння. Далі ми обговоримо індекси продуктивності різних архітектурам паралельних систем баз даних [12].

Ефективність системи баз даних може бути оцінена кількома вимірами. Час відгуку та пропускна здатність є найбільш популярними показниками. Це тому, що їх легко виміряти, і вони безпосередньо впливають на користувачів системи баз даних. Час відповіді можна визначити як час, необхідний для виконання операції, поданої користувачем системи.

Використання паралелізму в базі даних може значно підвищити продуктивність системи. Багато паралельних систем баз даних забезпечують майже лінійне прискорення та масштабування. Пришвидшення досягається, використанням паралелізму внутрішнього запиту. Такий паралелізм можна досягти шляхом розділення даних та використання існуючих послідовних підпрограм або написання нових спеціальних паралельних підпрограм для запитів. Складні запити можна розділити на підзавдання, які можуть працювати паралельно.

Масштабування можна отримати за допомогою паралелізму між запитами, при якому кілька запитів можуть виконуватися незалежно, збільшуючи загальну пропускну здатність системи.

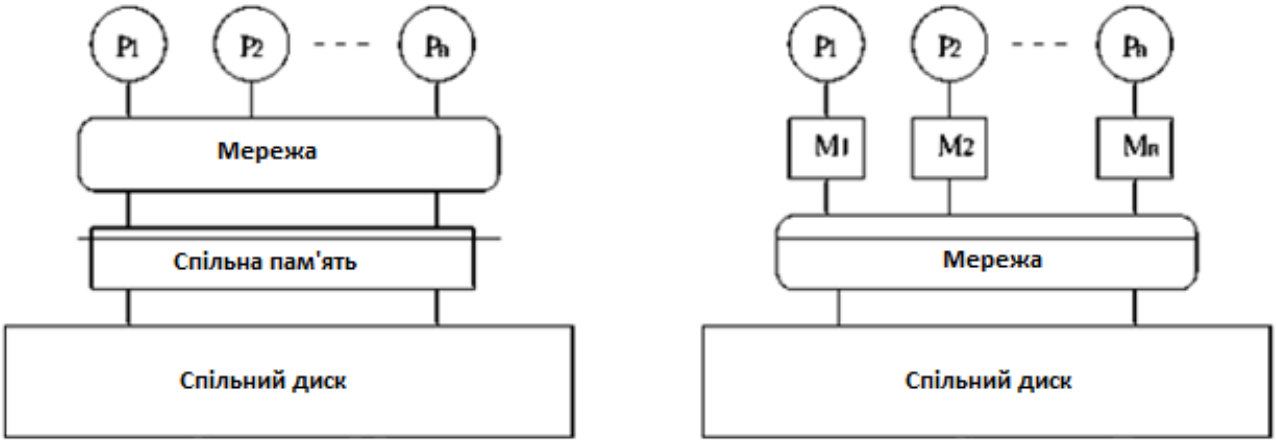


Рис. 2.8. Схематичне зображення архітектури спільної пам'яті та спільного диску

P – віртуальний процесор, M – віртуальна пам'ять модуля

В архітектурі спільної пам'яті, яка також називається спільною системою, система складається з ряду процесорів, усіх підключених до однієї спільної пам'яті та одного логічного спільного диска (рис. 2.8). Зв'язок між процесорами надається безкоштовно за допомогою спільної пам'яті. Системи, що слідують цій архітектурі, характеризуються чудовим балансом навантаження. Основним

недоліком цих систем є обмежена масштабованість та доступність. Обмежена масштабованість обумовлена тим, що всі процесори змагаються за використання спільної пам’яті. Це ставить верхню межу кількості процесорів. Доступність також може бути обмежена, наприклад, якщо загальна пам’ять вийшла з ладу, вся система не працює.

Архітектура спільних дисків забезпечує кожен процесор власною приватною пам’яттю з одним глобальним (логічним) спільним диском (рис. 2.8). Процесори більше не змагаються за спільну пам’ять. Однак вони змагаються за доступ до спільного диска. Такі системи все ще мають хороший баланс навантаження. Доступність набагато краще, ніж спільна пам’ять, з тим фактом, що збій спільного диска означає збій всієї системи. У практичних системах спільний диск фізично являє собою кілька дискових модулів, підключених через мережу до всіх елементів процесора. Це забезпечує високу надійність диска в цілому.

2.5. Висновки

В даному розділі проведено розробка програмної архітектури програмно-інформаційної системи для роботи з базою даних. Розглянуто специфіку паралельної обробки великих потоків даних та проаналізовано методику їх розгляду в базах даних

Проведено аналіз інструментальних засобів для роботи з базами даних та розробку методу паралельного опрацювання баз даних з використання колоночних індексів.

Наголошено, що архітектура CDN має найбільші перспективи в прототипі паралельної системи баз даних і з точки зору подальших досліджень, найбільший інтерес представляють такі проблеми:

1) експериментальні дослідження різних конфігурацій кластера в системах з архітектурою CDN;

2) розробка алгоритмів, що виконують реляційні операції, що враховують специфічні особливості архітектури CDN певним оптимальним чином;

3) розробка методів оптимізації паралельної роботи запитів, що розроблені для систем баз даних із CDN архітектурою.

3. ОПИС СТРУКТУРИ ТА МЕТОДІВ ОБРОБКИ ДАНИХ

3.1. Програмна реалізація інформаційної системи швидкісної обробки бази даних

Реляційна модель визначає кілька операторів, які поєднують і порівнюють два або більше відносин. Вона забезпечує звичайний набір операторів об'єднання, перетину, різниці та деякі більш екзотичні такі, як приєднання та поділ. Обговорення тут буде зосереджено на операторі equi-join (тут називається приєднатися). Оператор приєднання складає два відношення, A і B, за деяким атрибутом, щоб отримати третій відношення. Для кожного кортежу ta, в A, об’єднання знаходить усі кортежі tb, у B, значення яких атрибутів дорівнює ta. Для кожної пари відповідних кортежів оператор з’єднання вставляє у вихідні дані запарити кортеж, побудований об'єднанням пари.

У класичній роботі Кодд показав, що реляційна модель даних може представляти будь-яку форму даних, і що такі оператори повні [CODD70]. Сьогодні додатки SQL, як правило, є поєднання звичайних програм та операторів SQL. Програми взаємодіють з клієнтами, виконувати відображення даних і забезпечувати високий рівень напрямку потоку даних SQL. Спочатку модель даних SQL була запропонована для підвищення продуктивності програмістів на пропонування непроцедурної мови баз даних. Незалежність даних була додатковою перевагою; оскільки програми не визначають, як виконувати запит, програми SQL продовжують працювати у міру розвитку логічної та фізичної схеми бази даних.

Паралельність - це непередбачувана перевага реляційної моделі. Оскільки реляційні запити насправді просто реляційні оператори, що застосовуються до дуже великих наборів даних, то вони пропонують багато можливості паралелізму. Оскільки запити подаються непроцедурною мовою, вони пропонують значну свободу у виконанні запитів. Реляційні запити можуть виконуватися як графік потоку даних. Як згадувалося в Вступ, ці графіки можуть використовувати як конвеєрний паралелізм, так і секціонований паралелізм. Якщо один оператор надсилає свої результати іншому, два оператори можуть виконувати паралельно, надаючи потенціал прискорення двох.

Паралельність всередині реляційних операторів розбиття даних - це перший крок у розподіленому виконанні реляційних графіків потоку даних. Основна ідея полягає у використанні паралельних потоків даних замість написання нових паралельних операторів (програми). Такий підхід дозволяє використовувати немодифіковані існуючі послідовні підпрограми для виконання паралельнореляційних операторів. Кожен реляційний оператор має набір вхідних портів, на яких надходять вхідні кортежі та вихідний порт, на який надсилається вихідний потік оператора. Паралельний потік даних працює шляхом розділення (розпаралелювання) та об’єднання потоків даних у ці порти. Такий підхід дозволяє використовувати існуючі послідовні реляційні оператори для паралельного виконання.

Розглянемо приклад сканування відношення A, яке було розділене на три диски фрагменти A0, A1 та A2. Це сканування можуть бути реалізовані як трьома операторами сканування, які надсилають свої дані до загального оператора злиття. Оператор злиття виробляє єдиний вихідний потік даних до додатку або до наступного реляційного оператора. Паралельний виконавець запиту створює три процеси сканування, що показані на рис. 3.1 і спрямовує їх брати свої вхідні дані з трьох різних послідовних вхідних потоків (A0, A1, A2). Це також спрямовує їх надсилати свої результати на загальний вузол злиття. Кожне сканування може працювати на незалежному процесорі та диску. Отже перший основний оператор паралелізації - це злиття, яке може об’єднати кілька паралельних потоків даних в один послідовний потік.

```
insert into C
  select    *
  from      A, B
  where     A.x = B.y;
```

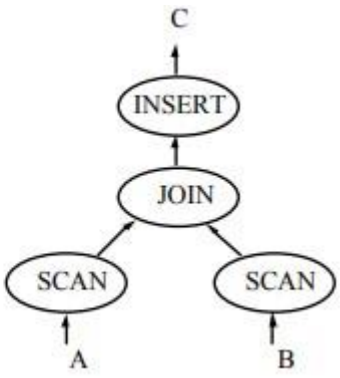


Рис. 3.1. Простий запит SQL та відповідний графік реляційних запитів.

Запит визначає, що а об’єднання має виконуватися між відносинами A і B, порівнюючи атрибут x кожного кортежу з A відношення зі значенням атрибута y кожного кортежу відношення B. Для кожної пари кортежів, які задовольняють предикат, результат кортежу формується з усіх атрибутів обох кортежів. Потім цей кортеж результату додається до відношення результату C. Пов’язаний графік логічного запиту (який може бути створений оптимізатором запитів) показує дерево операторів, один для об’єднання, один для вставки та один для сканування кожного відношення вводу [22, с. 719].

Розглянемо більш детально доменно-колоночну модель реалізації інформаційної системи швидкісної обробки бази даних. Для позначення реляційних операцій використали підхід, в якому під $\pi_{*\setminus A}(R)$ розуміється проекція на всі атрибути відношення R, за винятком атрибута A. З допомогою символу «o» позначаємо операцію конкатенації двох кортежів:

$$(x_1, \ldots, x_u) \text{ o } (y_1 \ldots, y_u) = (x_1, \ldots, x_u, y_1 \ldots, y_u) \qquad (3.1)$$

Під R (A, B_1,..., B_u) розуміємо відношення R з сурогатним ключем A (ідентифікатором цілочислового типу, що однозначно визначає кортеж) і атрибутами B_1,..., B_u, що представляє собою безліч кортежів довжини u + 1 виду (a, b_1,..., b_u), де $a \in Z_{\geq 0}$. Тут D_{Bj} – домен атрибуту Bj. через r. Bj позначає значення атрибута Bj, через r. A – значення сурогатного ключа кортежу r: r = (rA, r. B_1,..., r. B_u). Під адресом кортежу r розуміємо значення сурогатного ключа цього кортежу. Для отримання кортежу відношення R на його адресу використовуємо функцію розіменування.

Далі розглядаємо відношення як множини, а не як мультимножини. Це означає, що якщо при виконанні деякої операції вийшло певне відношення з дублікатами, то до цього відношення за замовчуванням використовується операція видалення дублікатів.

Нехай задано відношення R (A, B,...), T (R) = n. Нехай на множині $\mathfrak{D}_B$ задано відношення лінійного порядку. Колоночним індексом $I_{R.B}$ атрибута B

відношення R будемо називати впорядковане ставлення / , яке задовольняє наступним властивостям:

$$\begin{aligned} &T(I_{R.B}) = n, \pi_A(I_{R.B}) = \pi_A(R); \\ &\forall x_1, x_2 \in I_{R.B}\,(x_1 \le x_2 \Leftrightarrow x_1.B \le x_2.B); \\ &\forall r \in R(\forall x \in I_{R.B} \\ &\qquad (r.A = x.A \Rightarrow r.B = x.B)). \end{aligned} \qquad (3.2)$$

Умова першого рівняння означає, що множини значень сурогатних ключів (адрес) індексу і індексовані відносини збігаються. Умова другого рівняння означає, що елементи індексу впорядковані в порядку зростання значень атрибута B. Умови третього рівняння означають, що атрибут A елемента індексу містить адресу кортежу відносини R, що має таке ж значення атрибута B, як і у даного елементу колоночного індексу. Із змістовної точки зору колоночний індекс $I_{R.B.}$ є таблицею з двох колонок з іменами A і B. Кількість рядків в колоночному індексі збігається з кількістю рядків у індексованій таблиці. Колонка B індексу $I_{R.\ B}$ включає в себе всі значення колонки B таблиці R (з урахуванням значень, що повторюються), які відсортовані в порядку зростання. Кожен рядок x індексу $I_{R.B.}$ містить в колонці A сурогатний ключ (адреса) рядки r в таблиці R, що має таке ж значення в колонці B що і x. На рис. 3.2 представлені приклади двох різних колоночних індексів для одного і того ж відношення.

Нехай на безлічі значень домену D_B задано відношення лінійного порядку. Розіб'ємо множину D_B на k> 0 неперетинаємих інтервалів:

$$\left.\begin{aligned} &V_0 = [v_0; v_1], V_1 = (v_1; v_2], \ldots, \\ &V_{k-1} = (v_{k-1}; v_k]; \\ &v_0 < v_1 < \ldots < v_k; \\ &\mathfrak{D}_B = \bigcup_{i=0}^{k-1} V_i. \end{aligned}\right\} \qquad (3.3)$$

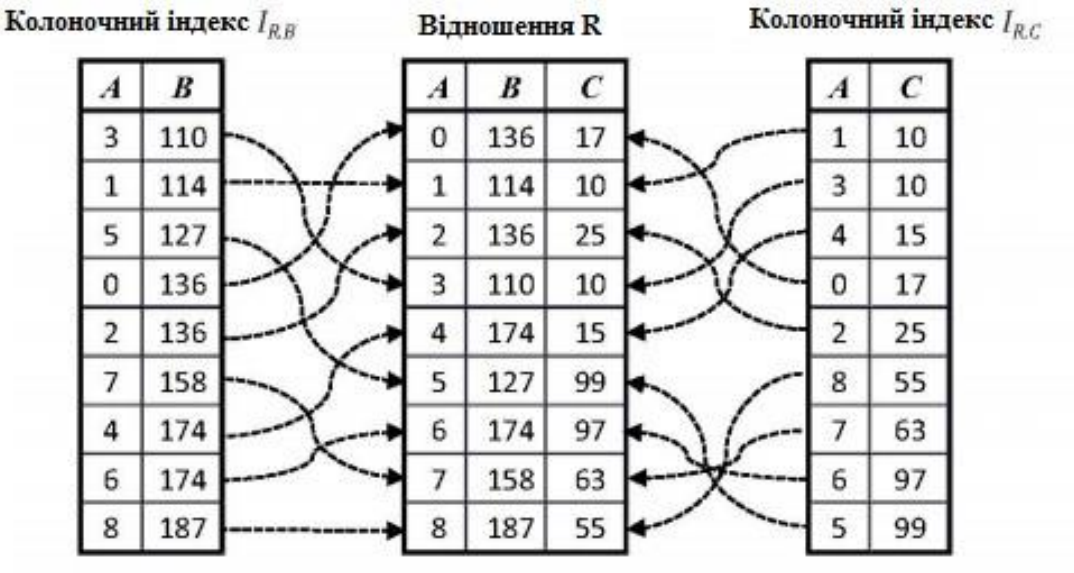


Рис. 3.2. Колоночний індекс

Відзначимо, що в випадку D_B = R матимемо $v_0 = -\infty$. Функція $\varphi_{\mathfrak{D}_B} : \mathfrak{D}_B \to \{0, \ldots, k-1\}$ називається доменною функцією фрагментації для / , якщо вона задовольняє наступній умові:

$$\forall i \in \{0, \ldots, k-1\} \quad (\forall b \in \mathfrak{D}_B \, (\varphi_{\mathfrak{D}_B}(b) = i \Leftrightarrow b \in V_i)) \tag{3.4}$$

Іншими словами, доменна функція фрагментації відповідає значенням b – номер інтервалу, якому це значення належить.

Нехай заданий колоночний індекс $I_{R.B.}$ для відношення R (A, B,...) з атрибутом B над доменом D_B та доменна функція фрагментації φD_B функція:

$$\varphi_{I_{R.B}} : I_{R.B} \to \{0, \ldots, k-1\}, \tag{3.5}$$

визначена за правилом:

$$\forall x \in I_{R.B} \, (\varphi_{I_{R.B}}(x) = \varphi_{\mathfrak{D}_B}(x.B)) \tag{3.6}$$

називається доменно-інтервальною функцією фрагментації для індексу / . Іншими словами, функція фрагментації $\varphi I_{R.B.}$: $I_{R.B.}$ зіставляє кожному кортежу x з $I_{R.B.}$ номер доменного інтервалу, якому належить значенням x. B. Визначимо r перший фрагмент (i = 0,..., k – 1) індексу $I_{R.B.}$ наступним чином:

$$I^i_{R.B} = \{x | x \in I_{R.B};\ \varphi_{I_{R.B}}(x) = i\} \qquad (3.7)$$

Це означає, що в і-тий фрагмент потрапляють кортежі, у яких значення атрибута B належать і-тому доменному інтервалу. Будемо називати таку фрагментацію доменно-інтервального. Кількість фрагментів k будемо називати ступенем фрагментації.Доменно-інтервальна фрагментація має наступні фундаментальними властивостями, що випливають безпосередньо з її визначення:

$$I_{R.B} = \bigcup_{i=0}^{k-1} I^i_{R.B};$$
$$\forall i, j \in \{0, \ldots, k-1\} \qquad (3.8)$$
$$\left(i \neq j \Rightarrow I^i_{R.B} \cap I^j_{R.B} = \emptyset\right)$$

На рис. 3.3 схематично зображено фрагментація колоночного індексу, що має ступінь k = 3.

Нехай для відношення R (A, B, C,...) задані колоночні індекси $I_{R.B.}$ та $I_{R.C.}$ транзитивних фрагментацією індексу $I_{R.C.}$ щодо індексу $I_{R.B.}$ називається фрагментацією, що задається функцією, що задовольняє умові: $\forall x \in I_{R.C}$

$$\ddot{\varphi}_{I_{R.C}}(x) = \varphi_{I_{R.B}}(\sigma_{A=x.A}(I_{R.B})) \qquad (3.9)$$

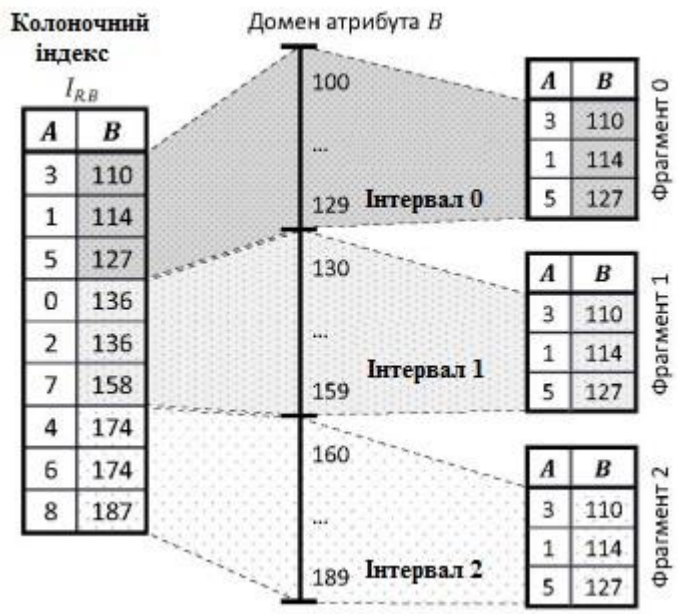


Рис. 3.3. Фрагмент колоночного індексу

Транзитивна фрагментація дозволяє розмістити на одному і тому ж вузлі елементи колоночних індексів, що відповідають одному кортежу індексованого відношення.

3.2. Обґрунтування засобів обробки даних

Ідея того, що системи для виконання безперервних запитів над потоковими даними можна будувати на основі реляційних систем управління базами даних, досить логічна – входять події, які можна представити у вигляді суворо типізованих кортежів, аналогічних збереженим в базі даних, а функціональність в мовах безперервних запитів і мов запитів до БД багато в чому перетинається [33, с. 92].

Для прикладу, розглянемо мову SQL: функціональність, що надається основними операторами select, where, group by, що відповідають, відповідно, за вибірку, фільтрацію і угруповання даних, необхідна в тому ж вигляді і при виконанні безперервних запитів. Виняток становить те, що у випадку з системою керування базами даних, всі вхідні кортежі вже відомі, і можна відразу виконати

запит над усіма даними, а у випадку з безперервним запитом, необхідно очікувати надходження нових кортежів і працювати з ними в міру появи.

Ще одна важлива відмінність РСУБД від систем виконанню безперервних запитів на їх основі полягає в тому, що деякі оператори SQL, такі як order by, join. Вони не можуть використовуватися в незмінному вигляді для виконання безперервних запитів – їх результат визначений тільки в тих випадках, коли відомий повний набір вхідних даних, що неможливо при постійному надходженні нових подій. Але їх функціональність також затребувана, тому потрібна підтримка віконної семантики. Наприклад, на вхід оператора order by раз в п’ять хвилин можуть подаватися всі події, отримані за ці п’ять хвилин – і тоді ми періодично будемо отримувати відсортоване результат. Зауважимо, що для підтримки віконної семантики в мові запитів, спочатку розробленому для СУБД, потрібно додавання спеціальних конструкцій.

Якщо торкатися архітектурної частини, то наступні компоненти РСУБД можуть бути використані з невеликими змінами при побудові систем виконання безперервних запитів:

– Аналізатор запитів. Відповідає за розбір запиту відповідно до цих слів і побудова плану його виконання. Зміни полягають у розширенні мови запитів віконної семантики.
– Оптимізатор запитів. З класичного вартісного оптимізатора повинні бути виключені частини, що стосуються оптимізації методів доступу до збережених на диску даними, так як при виконання безперервних запитів такі операції більше не використовуються.
– Виконавець запитів. Класичний виконавець запитів, який реалізує обхід за планом виконання запиту може залишитися практично незмінним, зміни будуть полягати в реалізації операцій, що вимагають віконної семантики [45, с. 291].

На базі описаної вище доменно-колоночної моделі представлення даних і методів декомпозиції реляційних операцій (розпаралелення) розроблена

програмна система «колоночний співпроцесор (Ксп)» (Columnar COProcessor сварки) для кластерних обчислювальних систем. В даному розділі описується архітектура і реалізації колоночного співпроцесора Ксп.

Рис. 3.4. Схематичне зображення взаємодії SQL-сервера з Ксп (з розпаралелюванням).

Стовпчик співпроцесор Ксп – це програмна система, призначена для управління розподіленими колоночнимио індексами, розміщеними в оперативній пам’яті кластерної обчислювальної системи. Призначення Ксп – обчислювати таблиці попередніх обчислень (ТПО) для ресурсномістких реляційних операцій за запитом СУБД. Загальна схема взаємодії СУБД і Ксп зображена на рис. 3.4. Ксп включає в себе програму «Координатор», що запускається на вузлі обчислювального кластера з номером О, та програму «Виконавець», що запускається на всіх інших вузлах, виділених для роботи Ксп.

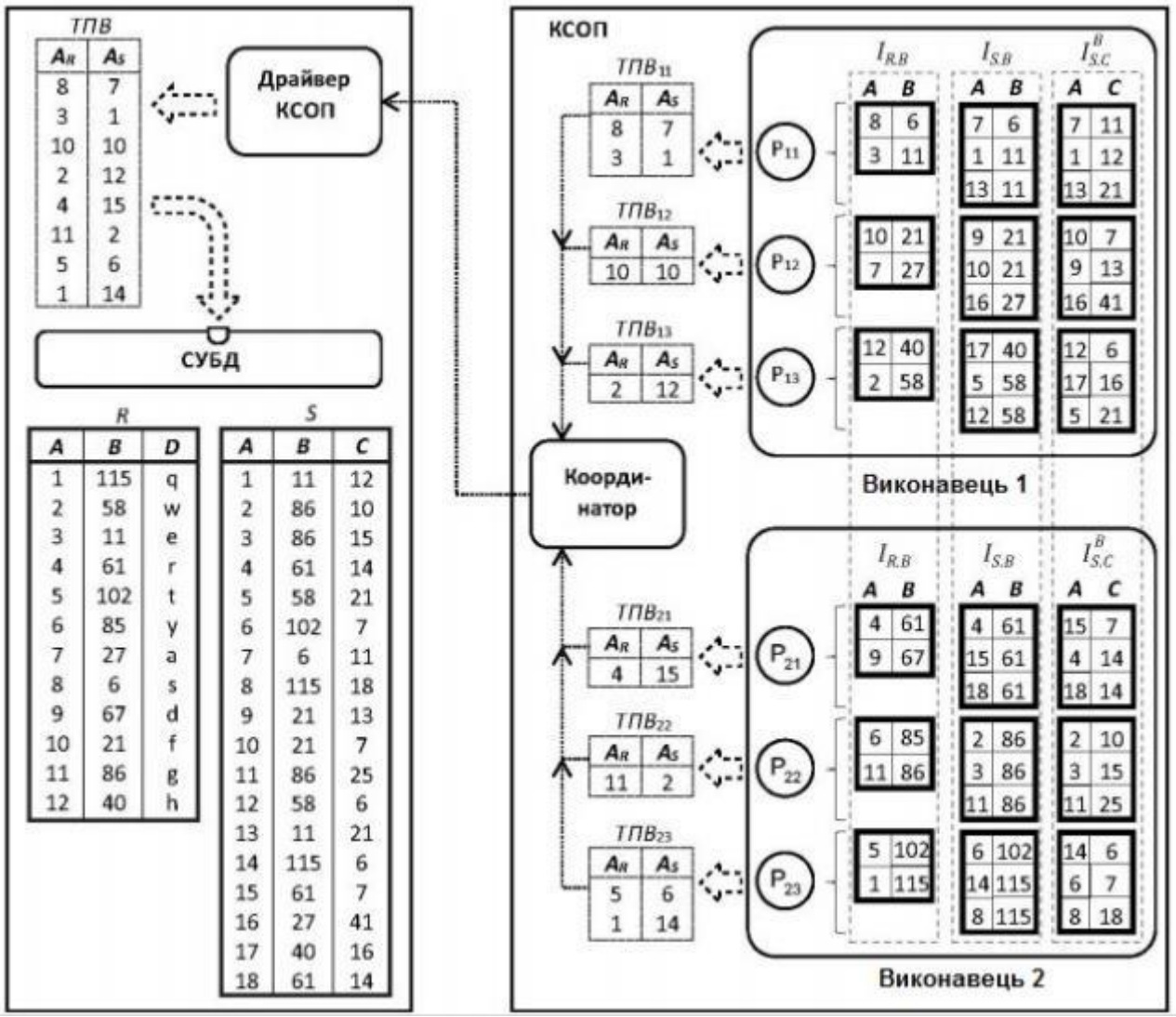


Рис. 3.5. Схематичне зображення обчислення ТПО з використанням Ксп

На SQL-сервері встановлюється спеціальна програма «Драйвер Ксп», що забезпечує взаємодію з координатором Ксп по протоколу TCP / IP. Ксп працює тільки з даними цілих типів 32 або 64 байта. При створенні колоночного індексів для атрибутів інших типів, їх значення кодується у вигляді цілого числа або вектора цілих чисел. Ксп підтримує наступні основні операції, доступні СУБД через інтерфейс драйвера Ксп: CreateColumnIndex (створення розподіленого колоночного індексу), Execute (виконання запиту на обчислення ТПВ), Insert (додавання в стовпчик індекс нового кортежу), TransitiveInsert (додавання в стовпчик індекс нового кортежу по транзитивності значенням), Delete (видалення з колоночного індексу кортежу), TransitiveDelete (видалення з

колоночного індексу кортежу по транзитивності значенням).Пояснимо загальну логіку роботи Ксп на простому прикладі. Нехай є база даних з двох відносин R (A, B, D) і S (A, B, C), що зберігаються на SQL сервері (див. рис. 3.5).

Припустимо, що Ксп має тільки два вузли-виконавці і на кожному вузлі є три процесорних ядра (процесорні ядра на рис. 3.5 промарковані позначками P_{11},..., P_{23}) – Покладемо, що атрибути R. B і S. B визначені на домені цілих чисел з інтервалу [0; 120). Сегментні інтервали для R. B і S. B визначимо наступним чином: [0; 20), [20; 40), [40; 60), [60; 80), [80; 100), [100; 120). В якості фрагментних інтервалів для R. B і S. B зафіксуємо: [0; 59] і [60,119]. Нехай атрибут S. визначено на домені цілих чисел з інтервалу [0; 25]. Спочатку адміністратор бази даних за допомогою драйвера Ксп створює для атрибутів R. B і S. B розподілені колоночні індекси $I_{R.\ B}$ і $I_{S.\ B.}$ Потім для атрибута S. створюється розподілений стовпчик індекс $I^{B}_{S.\ C}$, який фрагментується і сегментується транзитивно щодо індексу $I_{S.\ B}$ Розподілені колоночного індекси $I_{R.\ B}$, $I_{S.\ B}$ та $I^{B}_{S.\ C}$ зберігаються в оперативній пам’яті вузлів-виконавців. Таким чином, ми отримуємо розподіл даних всередині Ксп, наведене на рис. 3.5. При надходженні SQL запиту він перетвориться драйвером Ксп в план, який визначається наступним виразом реляційної алгебри:

$$\pi_{I_{R.B}.A \to A_R, I_{S.B}.A \to A_S}(I_{R.B} \underset{I_{R.B}.B = I_{S.B}.B}{\bowtie} (I_{S.B} \bowtie \sigma_{C<13}(I^{B}_{S.C}))) \qquad (3.10)$$

При виконанні драйвером операції Execute вказаний запит передасться координатору Ксп у вигляді оператора CCOPQL в форматі JSON. Запит виконується незалежно процесорними ядрами вузлів-виконавців над відповідними групами сегментів. При цьому за рахунок доменної фрагментації і сегментації не потрібні обміни даними як між вузлами-виконавцями, так і між процесорними ядрами одного вузла. Кожне процесорний ядро обчислює свою частині ТПВ, яка пересилається на вузол-координатор. Координатор об’єднує

фрагменти ТПВ в єдину таблицю і пересилає її драйверу, який виконує матеріалізацію цієї таблиці у вигляді відносини в базі даних, що зберігається на SQL сервері. Після цього SQL сервер замість вихідного SQL оператора, виконує наступний оператор:

```
SELECT D, C
FROM
R INNER JOIN (
ТПО INNER JOIN S ON (S. A = ТПО. AS)
) ON (R. A = ТПО. AR).
```

При цьому використовуються звичайні кластеризовані індекси у вигляді B-дерев, що заздалегідь побудовані для атрибутів R. A і S. A.

Колоночний співпроцесор Ксп був реалізований з використанням апаратно-незалежних паралельних технологій MPI та OpenMP.

3.3. Опис методів та алгоритмів

Декомпозиція полягає в розбитті алгоритму виконання операції на окремі підзадачі (розпаралелювання) для подальшого обміну даними.

Декомпозиція природного з'єднання. Розглянемо декомпозицію операції природного з'єднання. Нехай задані два відношення R (A, B_1,..., B_u, C_1,..., C_v) та S (A, B_1,..., B_u, D_1,..., D_w). Нехай є два набори колоночних індексів але атрибутам B_1,..., B_u $I_{R.B1}$,..., $I_{R.Bu}$ $I_{S.B1}$,..., $I_{S.Bu}$. Нехай для всіх цих індексів задана доменно-інтервальна фрагментація ступеня k:

$$I_{R.B_j} = \bigcup_{i=0}^{k-1} I^i_{R.B_j};\ I_{S.B_j} = \bigcup_{i=0}^{k-1} I^i_{S.B_j}. \qquad (3.11)$$

Припустимо

$$P_j^i = \pi_{I^i_{R.B_j}.A \to A_R,\ I^i_{S.B_j}.A \to A_S}(I^i_{R.B_j} \underset{I^i_{R.B_j}.B_j = I^i_{S.B_j}.B_j}{\bowtie} I^i_{S.B_j}) \quad (3.12)$$

для всіх i = 0,..., k – 1. Визначимо:

$$P = \bigcup_{i=0}^{k-1} P^i. \quad (3.13)$$

побудуємо відношення:

$$Q\ (B_1,..., B_u, C_1,..., C_v, D_1,..., D_w) \quad (3.14)$$

наступним чином:

$$Q = \{(\&_R(p.A_R).B_1, \ldots, \&_R(p.A_R).B_u, \&_B(p.A_B).C_1, \ldots, \&_B(p.A_B).C_v, \&_S(p.A_S).D_1, \ldots, \&_S(p.A_S).D_w) | p \in P\}. \quad (3.15)$$

Приклад розрахунку операції природного сполучення двох відношень з використанням розподілених колоночних індексів перевірено також рядом інших науковців і отримані результати засвідчують їх правдивість.

Декомпозиція операції угруповання. Розглянемо декомпозицію операції угрупування. Нехай задано відношення

$$R\,(A, B, C_1, \ldots, C_u, D_1, \ldots, D_w, \ldots) \quad (3.16)$$

з сурогатним ключем A. Нехай для атрибутів D_1,..., D_w задана агрегуюча функція agrf. Нехай є колоночний індекс $I_{R.\,B}$. Нехай також є колоночні індекси:

$I_{R.\,C1},\ldots, I_{R.\,Cu}$; $I_{R.\,D1},\ldots, I_{R.\,Dw}$. Нехай для індексу $I_{R.\,B}$ задана доменно-інтервальна фрагментація ступеня k:

$$I_{R.B} = \bigcup_{i=0}^{k-1} I^i_{R.B} \qquad (3.17)$$

Нехай для індексів $I_{R.\,C1},\ldots, I_{R.\,Cu}$ та $I_{R.\,D1},\ldots, I_{R.\,Dw}$ задана транзитивній відносно $I_{R.\,B}$ фрагментація:

$$\forall j \in \{1,\ldots,u\} \left(I_{R.C_j} = \bigcup_{i=0}^{k-1} I^i_{R.C_j} \right) \qquad (3.18)$$

$$\forall j \in \{1,\ldots,w\} \left(I_{R.D_j} = \bigcup_{i=0}^{k-1} I^i_{R.D_j} \right) \qquad (3.19)$$

Припустимо:

$$\begin{aligned} P_i = \pi_{A,\,F}(& \\ & \gamma_{\min(A)\to A,B,C_1,\ldots,C_u,\mathrm{agrf}(D_1,\ldots,D_w)\to F}(\\ & \quad I^i_{R.B} \bowtie I^i_{R.C_1} \bowtie \ldots \bowtie I^i_{R.C_u} \bowtie \\ & \quad \bowtie I^i_{R.D_1} \bowtie \ldots \bowtie I^i_{R.D_w})) \end{aligned} \qquad (3.20)$$

для всіх i = 0,..., k – 1. Визначимо / . Побудуємо відношення Q (B, C_1,..., C_u, F) наступним чином:

$$\begin{aligned} Q = \{ & (\&_{R}(P.A).B, \&_{R}(P.A).C_1,\ldots, \\ & \&_{R}(P.A).C_U, {}_{P.}F) \; I_P \in P\}. \end{aligned} \qquad (3.21)$$

Доведення коректності описаної декомпозиції операції угруповання наводиться також в роботі інших науковців.

Декомпозиція операції перетину. Розглянемо декомпозицію операції перетину. Нехай задані дві відносини R (A, B_1,..., B_u) та S (A, B_1,..., B_u), що мають однаковий набір атрибутів. Нехай є два набори колоночних індексів по атрибутам B_1,..., B_u: $I_{R.\,B1}$,..., $I_{R.\,Bp}$, $I_{S.\,B1}$,..., $I_{s.\,Bu}$. Нехай для всіх цих індексів задана доменно-інтервальною фрагментацією ступеня k:

$$I_{R.B_j} = \bigcup_{i=0}^{k-1} I^i_{R.B_j}; I_{S.B_j} = \bigcup_{i=0}^{k-1} I^i_{S.B_j}. \qquad (3.22)$$

Припустимо:

$$P^i_j = \pi_{I^i_{R.B_j}.A \to A_R,\ I^i_{S.B_j}.A \to A_S}(I^i_{R.B_j} \underset{(I^i_{R.B_j}.B_j = I^i_{S.B_j}.B_j)}{\bowtie} I^i_{S.B_j}) \qquad (3.23)$$

для всіх i = 0,..., k – 1 і j = 1,..., u. Побудуємо відношення Q (A, B_1,..., B_u) наступним чином:

$$Q = \{r | r \in R \wedge r.A \in \pi_{A_R}(P)\} \qquad (3.24)$$

Доведення коректності описаної декомпозиції операції перетину наведено в наукових дослідженнях інших науковців.

З використанням описаної методики ми можемо виконувати декомпозицію операцій проекції, вибору, вилучення дублікатів та об’єднання баз даних шляхом розпаралелювання.

3.4 Висновки

В розділі проведено опис структури та методів обробки даних. Проаналізовано програмна реалізація інформаційної системи швидкісної обробки бази даних та сказано, що додатки SQL, як правило, є поєднання звичайних програм та операторів SQL, адже програми взаємодіють з клієнтами, виконувати відображення даних і забезпечувати високий рівень напрямку потоку даних SQL. Оскільки реляційні запити насправді просто реляційні оператори, що застосовуються до дуже великих наборів даних, то вони пропонують багато можливості паралелізму. Паралельність всередині реляційних операторів розбиття даних – це перший крок у розподіленому виконанні реляційних графіків потоку даних. Основна ідея полягає у використанні паралельних потоків даних замість написання нових паралельних операторів (програми).

Розглянуто та описано доменно-колоночну модель реалізації інформаційної системи швидкісної обробки бази даних.

Проведено обґрунтування використовуваних засобів обробки даних та окремо описано інтерфейс таких драйверів, як CreateColumnlndex (створення розподіленого колоночного індексу), Execute (виконання запиту на обчислення ТПВ), Insert (додавання в стовпчик індекс нового кортежу), Transitivelnsert (додавання в стовпчик індекс нового кортежу по транзитивності значенням), Delete (видалення з колоночного індексу кортежу), TransitiveDelete (видалення з колоночного індексу кортежу по транзитивності значенням).

Деталізовано порядок виконання різних варіантів операції декомпозиції.

4 ТЕСТУВАННЯ

Тестування програмного забезпечення (ПЗ) – це процес, що використовується для виміру якості розроблюваного програмного забезпечення, а саме відповідність специфікації, технічному завданню, або вимогам замовника ПЗ.

Практичний підхід до тестування ПЗ особливу увагу приділяє процесам тестування на фоні стрімкого прискорення процесу розробки ПЗ.

Цей підхід орієнтований на використання спеціалістами з тестування ПЗ тестових робіт. Швидкість і ефективність розробки ПЗ залежить від того наскількипроцес тестування вписується в загальний життєвий цикл розробки ПЗ і від ефективності використання технології тестування.

Тестування - це одна з технік контролю якості, що включає в себе діяльністьз планування робіт (Test Management), проектуванню тестів (Test Design), виконанню тестування (Test Execution) і аналізу отриманих результатів (Test Analysis).

Необхідними умовами для тестування є наявність :

- об'єкта тестування, доступного для проведення іспитів;
- виконавця(ів) (залежно від виду проведених іспитів їм може бути як людина,так і машина або комбінація людина + машина).

Достатніми умовами для тестування є наявність:

- об'єкта тестування, доступного для проведення іспитів;
- виконавця(ів) (залежно від виду діяльності на різних фазах їм може бути яклюдина, так і машина або комбінація людина + машина);
- плану тестування;
- тест кейсів / тестів;
- звіту, що підтверджує виконання задач і досягнення цілей, по тестуваннюоб'єкта.

4.1 Аналіз методів тестування

Методології тестування програмного забезпечення - це різні стратегії або підходи, що використовуються для тестування програми для забезпечення її передбаченої роботи. Вони охоплюють всі аспекти тестування програми, включаючи модульне, інтеграційне та системне тестування.

Тестування чорної скриньки (black box)

Методика тестування без будь-яких знань про внутрішню роботу додатка називається «чорним ящиком». При тестуванні чорного ящика тестувальник має доступ до ПЗ тільки через ті ж інтерфейси, що й замовник або користувач, або через зовнішні інтерфейси, що дозволяють іншому комп'ютеру або іншому процесу підключитися до системи для тестування.

У таблиці 4.1 наведено переваги та недоліки тестування методом чорної скриньки (black box).

Таблиця 4.1 – Переваги та недоліки тестування методом чорної скриньки(black box)

Переваги	Недоліки
1	2
Добре підходить і ефективно для великих сегментів коду.	Обмежене покриття, оскільки насправді виконується тільки вибрану кількість тестових сценаріїв.
Кодовий доступ не потрібно.	Тестові приклади важко розробити.
Чіткий поділ перспективи користувача з точки зору розробника за допомогою явно певних ролей.	Сліпий охоплення, оскільки тестувальник неможе орієнтуватися на певні сегменти коду

Продовження табл. 4.1.

1	2
Велика кількість тестувальників може протестувати додаток без будь-яких знань про реалізацію, мовою програмування або операційних системах.	Неефективне тестування, через те, що тестувальник тільки має обмежені знання пропрограму.

Тестування білої скриньки (white box)

Тестування білої скриньки (white box) – це докладне дослідження внутрішньої логіки і структури коду. Під час тестування білого ящика (англ. white-box testing, також - прозорого ящика), розробник тесту має доступ до вихідного коду програм і може писати код, який пов'язаний з бібліотеками ПЗ, що тестується. Це типово для юніт-тестування (unit testing), при якому тестуються тільки окремі частини системи. Воно забезпечує те, що компоненти конструкції - працездатні й стійкі до певного ступеня. Під час тестування білого ящика використовуються метрики покриття коду.У таблиці 4.2 наведено переваги та недоліки тестування методом білоїскриньки (white box).

Освоєння домену системи завжди дає тестувальнику перевагу над кимось з обмеженими знаннями домену. На відміну від тестування чорного ящика, де тестувальник тестує тільки призначений для користувача інтерфейс програми; при тестуванні в сірому польоті тестувальник має доступ до проектної документації та базі даних. Для тестування сірих скриньок потрібні як високорівневі, так і детальні документи, що описують заявку, яку вони збирають для визначення тестових випадків.

Таблиця 4.2 – Переваги та недоліки тестування методом білої скриньки (white box)

Переваги	Недоліки
Оскільки тестувальник знає вихідний код, стаєдуже легко дізнатися, який тип даних може допомогти в ефективному тестуванні програми.	У зв'язку з тим, що для тестування білих ящиків потрібно кваліфікований тестувальник,витрати збільшуються.
Це допомагає в оптимізації коду.	Іноді неможливо заглянути в кожен куточок і кут, щоб виявити приховані помилки, які можуть створювати проблеми.
Додаткові рядки коду можуть бути видалені, що може привести до прихованих дефектів.	Важко підтримувати тестування білих ящиків,для цього потрібні спеціалізовані інструменти, такі як аналізатори коду і інструменти. налагодження.
Завдяки знанням тестувальника про код, максимальне охоплення досягаєтьсяпри написанні сценарію сценарію.	

Тестування сірої скриньки (grey box)

Тестування сірої скриньки (grey box) – це метод тестування програми з обмеженим знанням внутрішньої роботи програми. При тестуванні програмного забезпечення фраза, чим більше ви знаєте, тим краще переносить масу при тестуванні програми.

Маючи ці знання, тестувальник може підготувати кращі тестові дані і сценарії тестування при складанні плану тестування.

У таблиці 4.3 наведено переваги та недоліки тестування методом сірої скриньки (grey box).

Таблиця 4.3 – Переваги та недоліки тестування методом сірої скриньки(grey box)

Переваги	Недоліки
Пропонує комбіновані переваги тестування чорного ящика і білого ящика, де це можливо.	Оскільки доступ до вихідного коду недоступний, можливість пройти через код і зону тестування обмежена.
Тестувальники сірого ящика не покладаються на вихідний код; замість цього вони покладаються на визначення інтерфейсу і функціональні специфікації.	Тести можуть бути зайвими, якщо розробник програмного забезпечення вже виконав тестовий приклад.
Грунтуючись на наявної обмеженої інформації, тестувальник сірого ящика може розробити відмінні сценарії тестування, особливо щодо протоколів зв'язку та обробки даних.	Тестування всіх можливих вхідних потоків нереально, оскільки для цього потрібно необгрунтоване кількість часу, тому багато програмні шляху будуть неперевірені.

У таблиці 4.4 наведено відмінності тестування «чорного ящика», «сірогоящика» і «білого ящика».

Описані вище техніки припускають, що код виконується, і різниця полягає лише в тій інформації, якою володіє тестувальник. В обох випадках це динамічне тестування.

Таблиця 4.4 – Відмінності методів тестування «чорного ящика», «сірогоящика» і «білого ящика»

Тестування методом чорної скриньки	Тестування методом сірої скриньки	Тестування методом білої скриньки
Не підходить для тестування алгоритмів.	Не підходить для тестування алгоритмів.	Підходить для тестування алгоритмів.
Він є вичерпним інайменш трудомістким.	Частково трудомісткий і вичерпний.	Самий вичерпний і трудомісткий тип тестування.
Також відомий як тестування з закритим ящиком, тестування з використанням даних або функціональне тестування.	Також відомий як прозоре тестування, оскільки тестувальник має обмежене знання внутрішніх аспектів програми.	Також відомий як прозоре тестування, структурне тестування або тестування на основі коду.
Виконується кінцевими користувачами, а також тестувальниками і розробниками.	Виконується кінцевими користувачами, а також тестувальниками і розробниками.	Зазвичай виконуються тестувальниками і розробниками.
Тестування засноване на зовнішніх очікуваннях. Внутрішнє поведінку додатки невідомо.	Тестування виконується на основі діаграм бази даних високого рівня і діаграм потоків даних.	Внутрішні роботи повністю відомі, і тестувальник може відповідним чином створювати тестові дані.
Це можна зробити тільки методом проб і помилок.	Домени даних і внутрішні кордони можуть бути перевірені, якщо вони відомі.	Домени даних і внутрішні кордони можуть бути краще перевірені.

Динамічне тестування програмного забезпечення – це тип тестування, який передбачає запуск програмного коду. Таким чином, аналізується поведінка програми під час її роботи.

Для виконання динамічного тестування необхідно щоб тестований програмний код був написаний , скомпільований і запущений . При цьому, може виконуватися перевірка зовнішніх параметрів роботи програми: завантаження процесора, використання пам'яті, час відгуку і т.д. – тобто, її продуктивність.

Динамічне тестування є частиною процесу валідації програмного забезпечення. Крім того динамічне тестування може включати різні підвиди, коженз яких залежить від:

– доступу до коду (тестування методом чорної скриньки (black box), тестування методом білої скриньки (white box) і тестування методом сірої скриньки (grey box));
– рівня тестування (модульне інтеграційне, системне, і приймальне тестування);

сфери використання програми (функціональне тестування, навантажувальнетестування, тестування безпеки)

При статичному тестуванні програмний код не виконується - аналіз програми відбувається на основі вихідного коду, який вираховується вручну, або аналізується спеціальними інструментами. У деяких випадках, аналізується не вихідний, а проміжний код (такий як байт-код або код на MSIL).

Статичне тестування починається на ранніх етапах життєвого циклу ПЗ і є, відповідно, частиною процесу верифікації.

Більшість статичних технік можуть бути використані для «тестування» будь-яких форм документації, включаючи вичитку коду, інспекцію проектної документації, функціональної специфікації і вимог.

Статичне тестування також може бути автоматизовано – так наприклад, можна використовувати автоматичні засоби перевірки синтаксису програмного коду.

Види статичного тестування:

– аналіз вихідного коду програми;
– перевірка вимог.

4.2 Особливості тестування системи

Для проведення тестування запропонованих алгоритмів паралельної обробки баз даних та метод оптимізації ресурсів паралельної обробки баз даних було проведено 2 види тестування; тестування на підготовленій базі даних, та тестування на користувацькій базі сайту.

Тестування проводилось за допомогою SQL Server 2019, середовище розробки - SQL Server Management Studio. Для відслідковування плана запросу та швидкості його виконання було задіяно профілювальник SQL Server Profiler

Для першого тестування було підготовлено базу з набором зв’язаних даних отриманих методом випадкової генерації та проведено запуск тестів на різній кількості записів: 100, 1000, 10000,100000, 1000000 записів. Кожен з цих тестів запускався двічі, до та після впровадження алгоритму паралельної обробки.

З іншого боку, для тестування користувацької бази було визначено два сайти(середовища) з даними користувачів системи, і протестовано кожен з них двічі.

4.3 Тестування системи

Тестування відповідно до вхідних даних, отриманих методом випадкової генерації.

Тест проводився для набору даних сгенерованих по заданій ієрархічній структурі. Було проведено тестування перерахунку порядку елементів у своїх гілках за після перемішування та підрахунок їх значень(значення батьківського елемента це сума значень нащадків), перед впровадженням алгоритму паралельної обробки.

Були проведені тести для різної кількості даних. Наприклад:

1) 100 строк;
2) 1000 строк;
3) 10000 строк;
4) 100000 строк;
5) 1000000 строк.

Результати тесту наведені в таблиці 4.5, і відображають інформацію про кількість оброблених елементів, часу на перерахунок порядку, і часу на підрахунок нових значень, які вони отримали внаслідок перемішування.

Після цього було застосовано алгоритму паралельної обробки даних до файлів бази даних та повторне тестування для кожного набору даних. Швидкість обробки збільшується з збільшенням кількості даних які можна розпаралелити. У таблиці 4.6 приведено результати вимірювань, на основі отриманих даних можна зробити висновок, що використання процесорного часу, після застосування алгоритму паралельної обробки даних значно зменшилось.

Таблиця 4.5 – Результати тестування до впровадження алгоритму паралельної обробки даних

Кількість елементів	Час на перерахунок порядку (сек.)	Час на підрахунок нових значень (сек.)
1	2	3
100	0,007	0,003
1000	0,05	0,036

Продовження табл. 4.5.

1	2	3
10000	0,3	0,2
100000	2,7	1,8
1000000	11,36	9,4

Проаналізувавши результати тестування без та з алгоритмом паралельної обробки даних, можна зробити висновок, що використання алгоритму значно пришвидшить час виконання запиту, оскільки він оброблюється в декількох потоках. Велика економія часу виконання запиту відбувається на великих обсягах даних. Невелика різниця часу до і після застосування алгоритму на малих обсягах даних пояснюється тим, що пришвидшення обробки даних було нівельовано додатковими затратами часу на розпаралелювання.

Таблиця 4.6 – Результати проведення тестування після впровадження алгоритму паралельної обробки даних

Кількість елементів	Час на перерахунок порядку (сек.)	Час на підрахунок нових значень (сек.)
100	0,006	0,003
1000	0,04	0,029
10000	0,23	0,09
100000	1,9	1,1
1000000	7,3	5,4

Тестування відповідно до вхідних даних, отриманих від користувачів

Для експерименту на реальних даних було проведено тестування, на двух сайтах с даними від користувачів. Вони відрізняються від сгенерованих меншою зв’язністю. Для кожного сайту було проведено два тести, до та після впровадження алгоритму паралельної обробки даних. У таблиці 4.7 наведено результати тестування до впровадження алгоритму.

Таблиця 4.7 – Результати проведення динамічного тестування до впровадження алгоритму паралельної обробки даних

Середовище	Час на перерахунок порядку (сек.)	Час на підрахунок нових значень (сек.)
stage	5,4	3,1
prod	15,6	12,3

Після застосування алгоритму паралельної обробки даних було проведено ще два тести для обох сайтів. У таблиці 4.8 наведено результати тестування з використанням алгоритму паралельної обробки даних.

Таблиця 4.8 – Результати проведення динамічного тестування з використанням алгоритму паралельної обробки даних

Середовище	Час на перерахунок порядку (сек.)	Час на підрахунок нових значень (сек.)
stage	4,6	2,5
prod	10,9	9

З попередніх експериментів можна зробити висновок, що алгоритм паралельної обробки даних забезпечує значну економію часу виконання запиту на великих обсягах даних.

4.4 Висновки

Одним із важливих етапів розробки додатку є тестування, так як воно є гарантією якості розроблюваного додатку.

В ході створення даного розділу було розглянуто різні стратегії тестування, та визначено, що для запропонованого алгоритму найоптимальнішим є метод тестування «білого ящика».

Проведення статичного та динамічного тестування довело доцільність використання алгоритму паралельної обробки даних для економії ресурсів системи та пришвидшення її роботи. Найефективніше алгоритм паралельної обробки даних показує себе при великій вибірці. На малих вибірках ефективність його роботи нівелюється затратами на розпаралелювання.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1. Оцінювання комерційного потенціалу розробки.

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Результатом магістерської кваліфікаційної роботи є програмно-інформаційна система паралельної обробки баз даних. Для проведення технологічного аудиту залучено трьох незалежних експертів: Рейда О.М. (к.т.н., доцент каф. ПЗ ВНТУ), Катєльніков Д.І. (к.т.н., доцент каф. ПЗ ВНТУ) та Майданюк В.П. (к.т.н., доцент каф. ПЗ ВНТУ).

Оцінювання комерційного потенціалу буде здійснене за критеріями, що наведені в таблиці 5.1. [46]

Таблиця 5.1 - Критерії оцінювання комерційного потенціалу розробки бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-тер.	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження таблиці 5.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-тер.	0	1	2	3	4
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використ. у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Продовження таблиці 5.1

11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів	Необхідно отримання великої кількості дозвільних документів, що вимагає значних коштів та часу	Процедура отримання дозвільних документів у вимагає незначних коштів та часу	Необхідно тільки пові-домлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання комерційного потенціалу експертами розробки зведено в таблицю 5.2.

Таблиця 5.2 - Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	Рейда О.М.	Катєльніков Д.І.	Майданюк В. П.
	Бали, виставлені експертами:		
1	3	3	4
Ринкові переваги (недоліки):			
2	3	3	2
3	3	4	3
4	3	3	4
5	3	4	3
Ринкові перспективи			
6	3	2	4
7	3	3	3
Практична здійсненність			
8	4	3	4
9	3	4	3
10	4	4	4
11	4	4	3
12	4	3	3
Сума балів	$СБ_1$=40	$СБ_2$=40	$СБ_3$=40
Середньоарифметична сума балів $\overline{СБ}$	41		

За даними таблиці 5.3 можна зробити висновок, щодо рівня комерційного потенціалу розробки. Зважимо на результат й порівняємо його з рівнями комерційного потенціалу розробки, що представлено в таблиці 5.3.

Таблиця 5.3 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів $\overline{СБ}$, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 – 10	Низький
11 – 20	Нижче середнього
21 – 30	Середній
31 – 40	Вище середнього
41 – 48	Високий

Рівень комерційного потенціалу розробки, становить 41 бал, що відповідає рівню «високий».

Сьогодні відомо більше двох десятків серверних систем управління базами даних. Найпопулярнішими є DB2, Oracle, Microsoft SQL Server, Informix. У таблиці 5.4 наведені основні технічні показники аналогів і нового програмного продукту.

Таблиця 5.4 - Основні технічні показники аналогів і нового програмного продукту.

Функціональність програми	Oracle	Застосунок
Підтримка бази даних MySql (хмарових технологій)	Крім ліцензії потрібно доплатити окремо за доповнення додатку «перевід бази в SQL»	Підтримка бази даних Sql, додатково можливе підключення до веб додатку
Історія (архів) запитів	+	+
Смс оповіщення щодо типу запитів	(крім ліцензії потрібно окремо доплатити за додаток)	Для того, щоб користуватися функцією смс оповіщення потрібно додатково поповняти рахунок в аккаунті
Розподіл запитів за допомогою пріоритетів	Відсутнє	Наявне
Статуси запитів	Не повністю функціонують всі види запитів до бази даних	всі запити відстежуються на кожному етапі роботи за допомогою статусів, статуси можна довільно редагувати, додавати або видаляти.
Облік кількості запитів до бази даних	Обмежено кіл-ть запитів до бази даних не більше 100000	Без обмежень

З таблиці 5.4. видно, що розроблений продукт по обміну інформацією з баз даних має ряд переваг, в першу чергу перевагу в тому що, він не обмежений, як її існуючі аналоги, де кожні додаткові модулі потрібно докупити, не зважаючи на те що продукт є платним. Розроблений продукт по обміну інформацією з баз даних є завершеним і призначений для оптимізації роботи з віддаленими джерелами інформації. Дана система не є вузькоспеціалізованою системою, адже вона може бути використана в практичній діяльності інтернет-магазинів, сервісних центрів та інших установах, де потрібен швидкісний обмін даними.

Дана система зацікавить користувачів своєю дешевизною, оскільки завдяки цій системі швидкість обміну даними буде збільшена, а це вплине на якість отримання необхідної інформації. На даному етапі розробки продукт не має високого рівня новизни, проте його технічна реалізація може стояти на рівні з існуючими аналогами на ринку, що підтверджує його конкурентоспроможність на ринку програмного забезпечення.

Надійність розробки перевірено на різних операційних системах, з різним тест кейсами.

В даній розробці зацікавлене широке коло споживачів. Розробка є актуальною в сферах бізнесу, науки, освіти тощо.

Вона є перспективною, оскільки використовує передові технології, які включають в себе переваги сучасних програмних рішень (простота, гнучкість, ефективність, швидкодія). Із недоліків можна відмітити необхідність доповнення функціоналу.

Розробка на даний момент знаходиться у ході доопрацювання, але в подальшому її можна застосувати на практиці, що підтверджено відповідною довідкою про можливість впровадження.

Комерціалізація розробки знаходиться на початковому етапі, розпочато процес виведення продукту на інформаційний ринок, проводиться підготовка презентації для інвесторів. Програму можливо подавати безкоштовно, але

отримання заробітку можливе за ту саму роботу програмного продукту після випробувального терміну.

Серед методів розкрутки нової розробки, які дозволяють отримати віддачу від витрачених зусиль на розробку та для швидкого просування на ринок доцільно обрати: створення офіційної сторінки у Facebook, Instagram, Twitter та Google+, створення каналу на YouTube, використання блог-платформ Livejournal.com, Блоги i.ua, тощо.

5.2 Прогнозування витрат на виконання науково-дослідної, дослідно-конструкторської) та конструкторсько-технологічної роботи.

Прогнозування витрат на виконання науково-дослідної, дослідно-конструкторської та конструкторсько-технологічної робіт складається з таких етапів:

1) розрахунок витрат, які безпосередньо стосуються виконавців даного розділу роботи;

2) розрахунок загальних витрат на виконання даної роботи;

3) прогнозування загальних витрат на виконання та впровадження результатів даної роботи.

Виконаємо розрахунок витрат, які безпосередньо стосуються виконавця даного розділу роботи, приймаючи до уваги те, що для розробки програми було залучено одного розробника.

Основна заробітна плата розробника (дослідника) $З_о$:

$$З_о = \frac{М}{Т_р} \cdot t \ [грн], \qquad (5.1)$$

де М – місячний посадовий оклад розробника, 11000,00 грн.

$Т_р$ – число робочих днів в місяці; приблизно $Т_р$ = (22) дні;

t – число робочих днів роботи розробника (дослідника) – 66 днів.

$$З_о = \frac{11000}{22} \cdot 66 = 33000{,}00 \text{ (грн).}$$

Додаткова заробітна плата $З_д$ розробника, розраховується як 10 % від суми основної заробітної, тобто:

$$З_д = (0{,}1...0{,}12) \cdot З_о \text{ [грн].} \quad (5.2)$$

$$З_д = 0{,}12 \cdot 33000 = 3960{,}00 \text{ (грн).}$$

Нарахування на заробітну плату $Н_{зп}$ розробника становлять 22 % і розраховуються за формулою:

$$Н_{зп} = (З_о + З_о) \cdot \frac{\beta}{100} \text{ [грн],} \quad (5.3)$$

де $З_о$ – основна заробітна плата розробників, грн;

$З_д$ – додаткова заробітна плата розробника, грн;

β – ставка єдиного соціального внеску, 22%.

$$Н_{зп} = (33000{,}00+3960{,}00) \cdot 0{,}22 = 8131{,}20 \text{ (грн).}$$

У спрощеному вигляді амортизаційні відрахування розраховуємо за формулою:

$$А = (Ц \cdot Т)/(12 \cdot Тв) \text{ [грн],} \quad (5.4)$$

де Ц – загальна балансова вартість обладнання, приміщення тощо, грн;

Т – фактична тривалість використання, міс;

$Т_В$ – термін використання обладнання, приміщень тощо, роки.

Зроблені розрахунки зведено до таблиці 5.5.

Таблиця 5.5 – Амортизаційні відрахування

Найменування	Балансова вартість, грн	Термін використання, роки	Фактична трив. використання, міс.	Величина амортизаційних відрахувань, грн
Приміщення	200000	25	3	2000,00
Ноутбук	20000	5	3	1000,00
Всього				3000,00

Під час розробки продукту використовувались лише безкоштовні програмні засоби.

Крім того, за три місяці було сплачено 450 грн на послуги Інтернет (150 грн/міс).

Витрати на силову електроенергію $В_е$ розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п \text{ грн}, \quad (5.5)$$

де В – вартість 1 кВт-год. В = 2,4 грн./кВт;

П – установлена потужність обладнання – 0,3 кВт;

Ф – фактична кількість годин роботи обладнання – 528 годин,

$К_п$ – коефіцієнт використання потужності.

$$В_е = 1,9 \cdot 0,3 \cdot 528 \cdot 0,4 = 120,38 \text{ (грн)}.$$

Інші витрати $І_в$, як 50% від суми основної заробітної плати розробника тобто:

$$В_{ін} = 100\% \cdot З_о \textit{ [грн]}. \quad (5.6)$$

$$В_{ін} = 1 \cdot 33000,00 = 33000,00 \text{ (грн).}$$

Сума всіх попередніх статей витрат дає витрати на виконання даної частини (розділу, етапу) роботи – В:

$$В = 33000+3960,00+8131,20+3000+450+120,38+33000 = 81661,58 \text{ (грн).}$$

Загальна вартість всієї наукової роботи ($В_{заг}$) визначається за формулою:

$$B_{заг} = B/\alpha \text{ [грн],} \quad (5.7)$$

де α – частка витрат, які безпосередньо здійснює виконавець даного етапу роботи, у відносних одиницях.

$$В_{заг} = 81661,58/1 = 81661,58 \text{ (грн).}$$

Проведемо прогнозування загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи за формулою:

$$ЗВ = \frac{В_{заг}}{\beta} \text{ [грн],} \quad (5.8)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи. Так, як розробка знаходиться на дослідного зразка, то $\beta \approx 0,9$.

$В_{заг}$ - загальна вартість всієї наукової роботи – 81661,58грн.

$$ЗВ = \frac{81661,58}{0,9} = 90735,09 \text{ (грн).}$$

Отже, прогноз загальних витрат на виконання та впровадження результатів становить 90735,09 грн.

5.3 Прогнозування комерційних ефектів від реалізації результатів розробки.

Спробуємо кількісно спрогнозувати вигоду яку можна отримати у майбутньому від впровадження результатів виконаної наукової роботи. Зрозуміло, що всі зроблені тут розрахунки будуть приблизними і не передбачають деталізації.

Виконання даної наукової роботи та впровадження її результатів складає приблизно 3 місяці. Позитивні результати від впровадження розробки очікуються вже в перший рік впровадження.

Проведемо прогнозування позитивних результатів та кількісне їх оцінювання по роках.

Обчислимо збільшення чистого прибутку підприємства $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

$$\Delta П_i = \sum_1^n (\Delta П_я \cdot N + П_я \cdot \Delta N)_i \ [грн], \quad (5.9)$$

де $\Delta П_я$ – покращення основного якісного показника від впровадження результатів розробки у даному році;

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ – основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

Функціональні переваги нового продукту дають можливість збільшити чистий прибуток підприємства на 1000 грн, а кількість одиниць реалізованої

послуги збільшиться: протягом першого року – на 200 од., протягом другого року – ще на 300 од., протягом третього року – ще на 400 од.

Орієнтовно: реалізація до впровадження результатів наукової розробки складала 1 од., а прибуток, що його отримувало підприємство на одиницю послуги до впровадження результатів наукової розробки – 200 грн.

Спрогнозуємо збільшення чистого прибутку підприємства від впровадження результатів наукової розробки у кожному році відносно базового.

Збільшення чистого прибутку підприємства $\Delta П_1$ протягом першого року:

$$\Delta П_1=200\cdot1+(200+1000)\cdot200=240200{,}00 \text{ (грн)}.$$

Збільшення чистого прибутку підприємства $\Delta П_2$ протягом другого року:

$$\Delta П_2=200\cdot1+(200+1000)\cdot(200+300)=600200{,}00 \text{ (грн)}.$$

Збільшення чистого прибутку підприємства $\Delta П_3$ протягом третього року:

$$\Delta П_3=200\cdot1+(200+1000)\cdot(200+300+400)=1080200{,}00 \text{ (грн)}.$$

Отже, комерційний ефект від впровадження розробки виражається у щорічному збільшенні чистого прибутку підприємства протягом трьох років.

5.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності.

Щоб оцінити доцільність фінансування проекту, необхідно провести розрахунки ефективності вкладених інвестицій. Основними показниками є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності.

На першому етапі розрахуємо теперішню вартість інвестицій PV, що вкладаються в наукову розробку.

Такою вартістю ми можемо вважати прогнозовану величину загальних витрат ЗВ на виконання та впровадження результатів НДДКР. Будемо вважати, що ЗВ = PV=90735,09 (грн).

На другому етапі розраховуємо очікуване збільшення прибутку $\Delta П_i$, що його отримає підприємство від впровадження результатів наукової розробки, для кожного із років, починаючи з першого року впровадження. Таке збільшення прибутку було розраховане раніше. Результати вкладених у наукову розробку інвестицій виявляться за перший рік після провадження у тому, що у першому році підприємство отримає збільшення чистого прибутку на 240200,00 грн відносно базового року, у другому році – збільшення чистого прибутку на 600200,00 грн (відносно базового року), у третьому році – збільшення чистого прибутку на 1080200,00 грн (відносно базового року).

На третьому етапі для спрощення подальших розрахунків будуємо вісь часу, на яку наносимо всі платежі (інвестиції та прибутки), що мають місце під час виконання науково-дослідної роботи та впровадження її результатів. Рисунок, що характеризує рух платежів (інвестицій та додаткових прибутків) буде мати вигляд, наведений на рис. 5.1.

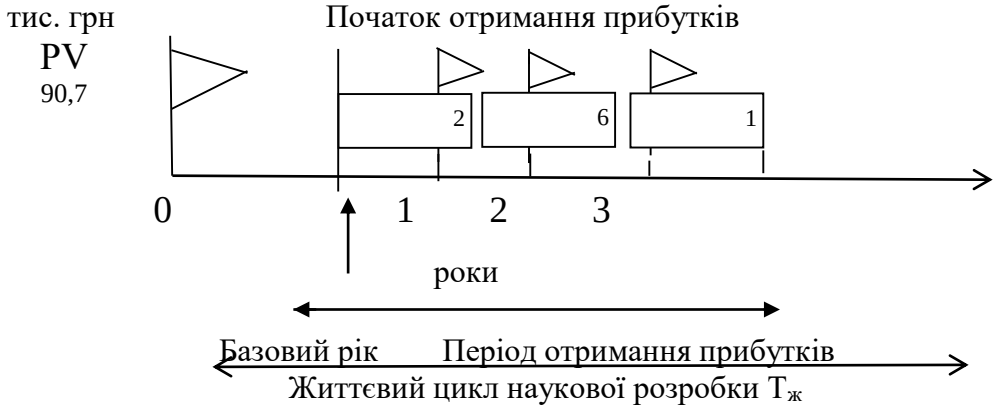


Рисунок 5.1 – Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

На четвертому етапі розраховуємо абсолютну ефективність вкладених інвестицій $Е_{абс}$ за формулою:

$$Е_{абс} = (ПП – PV) \text{ [грн]}, \qquad (5.10)$$

де ПП – приведена вартість всіх чистих прибутків, що їх отримає підприємство (організація) від реалізації результатів наукової розробки, грн;

PV – теперішня вартість інвестицій PV = ЗВ, грн.

Приведена вартість всіх чистих прибутків ПП розраховується за формулою:

$$ПП = \sum_{1}^{т} \frac{\Delta П_i}{(1+\tau)^t} \text{ [грн]}, \quad (5.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні - 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки „0”;

$$ПП = \frac{240200{,}00}{(1+0{,}1)^1} + \frac{600200{,}00}{(1+0{,}1)^2} + \frac{1080200{,}00}{(1+0{,}1)^3} = 1525966{,}93 \text{ (грн)}.$$

$$Е_{абс} = 1525966{,}93 - 90735{,}09 = 1435231{,}84 \text{ (грн)}.$$

Оскільки $Е_{абс} > 0$, результат від проведення наукових досліджень щодо розробки програмного продукту та їх впровадження принесе прибуток, тобто є доцільним, але це ще не свідчить про те, що інвестор буде зацікавлений у фінансуванні даної програми.

На п’ятому етапі розраховують відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$ за формулою:

$$E_в = \sqrt[T_ж]{1+\frac{E_{абс}}{PV}} - 1, \quad (5.12)$$

де $E_{абс}$ – абсолютна ефективність вкладених інвестицій, грн;

PV – теперішня вартість інвестицій PV = ЗВ, грн;

$T_ж$ – життєвий цикл наукової розробки, роки.

$$E_в = \sqrt[3]{1 + \frac{1435231,84}{90735,09}} - 1 = \sqrt[3]{16,81} - 1 = 1,56 \text{ або } 156\%$$

Порівняємо $E_в$ з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть. Спрогнозуємо величину $\tau_{мін.}$ У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою:

$$\tau = d + f, \quad (5.13)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; d = 0,2;

f – показник, що характеризує ризикованість вкладень; величина f = 0,5.

$$\tau = 0,2 + 0,5 = 0,7$$

Оскільки $E_в$ = 156% > $\tau_{мін}$ = 70%, то у інвестора є потенційна зацікавленість у фінансуванні даної наукової розробки.

На шостому етапі розраховуємо термін окупності вкладених у реалізацію наукового проекту інвестицій $T_{ок}$ за формулою:

$$T_{ок} = \frac{1}{E_в} \text{ [року]}. \quad (5.14)$$

$$T_{ОК} = \frac{1}{1{,}56} = 0{,}64 \text{ (року).}$$

Оскільки термін окупності вкладених у реалізацію наукового проекту інвестицій менше трьох років ($T_{ок} < 3$ років), то фінансування нової розробки є доцільним.

5.5 Висновок.

В даному розділі було здійснено оцінювання комерційного потенціалу розробки. Проведено технологічний аудит з залученням трьох експертів.

Аналіз експертних даних показав, що рівень комерційного потенціалу розробки є високим. Дослідження комерційного потенціалу розробки показав, що продукт за своїми характеристиками випереджає аналогічні програмні продукти, що робить його конкурентоспроможним на ринку. Існуючі переваги нової розробки дозволять швидко її поширити на ринку.

Згідно із розрахунками всіх статей витрат на виконання науково-дослідної, дослідно-конструкторської та конструкторсько-технологічної роботи загальна вартість витрат на розробку і впровадження складає 90735,09 грн.

Абсолютна ефективність вкладених інвестицій в сумі 1435231,84грн свідчить про отримання прибутку інвестором від впровадження програмного продукту.

Щорічна ефективність вкладених в наукову розробку інвестицій складає 156 %, що набагато вище за мінімальну бар'єрну ставку дисконтування, яка складає 70%. Це означає потенційну зацікавленість інвесторів у фінансуванні розробки.

Термін окупності складає 0,64 роки, що також свідчить про доцільність фінансування.

ВИСНОВКИ

У магістерській роботі розроблено програмну систему оптимізації обміну даних у паралельних обчисленнях при обробці баз даних, для підвищення ефективності взаємодії користувача з великими масивами.

У процесі виконання роботи розвязано такі задачі:

– проведено аналіз методів розробки автоматизованих систем паралельної обробки баз даних;

– проведено порівняльний аналіз аналогів та методів розв’язання поставленої задачі системи паралельної обробки баз даних;

– проведено аналіз методів організації паралельних запитів у великих потоках даних для систем доступу до баз даних;

– запропоновано нові методи: паралельної організації запитів великих баз даних, адаптивної оптимізації ресурсів при виконанні паралельних запитів великих баз даних;

– проведено опис структур даних та визначено засоби і методи тестування проектованої системи;

– розроблено програмні компоненти та систему на основі запропонованих методів;

– проведено експериментальні дослідження розроблених засобів.

У процесі досліджень використовано методи паралельних обчислень і методи синхронізації даних, методи фільтрації, математичне моделювання і прогнозування, комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень, методи порівняння, методи проектування.

На основі виконаних теоретичних і експериментальних досліджень приведено класифікацію методів для підвищення швидкодії роботи з великими обсягами даних, подальшого розвитку отримав метод паралельної організації запитів великих баз даних, запропоновано метод адаптивної оптимізації ресурсів при виконанні паралельних запитів великих баз даних

Для розробки моделі системи було проаналізовано предметну область дослідження та проведено аналіз аналогів архітектур (спільний диск, спільна пам’ять та інші), визначено їх переваги та недоліки.. Проведено варіантний аналіз засобів реалізації і обґрунтовано вибір мови програмування та інтегрованого середовища розробки.

Проведено тестування, доведено ефективність і правильність функціонування та відповідність до технічного завдання.

ПЕРЕЛІК ПОСИЛАНЬ

1. Азаров О. Д. Комп'ютерні мережі: навчальний посібник / О. Д. Азаров, С. М. Захарченко, О. В. Кадук. Вінниця: Вінницький Національний Технічний Університет, 2013. 371 с.
2. Щербаков О. В. Алгоритми та структури даних / уклад. О. В. Щербаков, Ю. Е. Парфьонов, В. М. Федорченко. Харків: ХНЕУ ім. С. Кузнеця, 2017. 58 с.
3. Кормен Х. Алгоритмы: построение и анализ [Текст] / Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л Ривест, Клиффорд Штайн. 3-е изд.: Пер. с англ. М.: ООО «И. Д. Вильямс», 2013. 1328 с.
4. Алгоритм Дейкстри [Електронний документ]. – Режим доступу: URL: http: // uk. wikipedia. org / wiki / Алгоритм_Дейкстри. Дата звернення – 1.12.2020.
5. Богач І. В. Алгоритми розв'язання задач з програмування. Решебник./ І. В., Богач, С. М. Довгалець, В. М. Дубовой Вінниця: ВНТУ, 2017 119 с.
6. Біячуєв Т. А. Безпека корпоративних мереж / Т. А. Біячуєв. М.: 2014. 481 с.
7. Васильєв О. Програмування на С++ в прикладах і задачах: навч. Посібник К.: Ліра-К, 2017. 258 с.
8. Волков С. Л. Требования к цифровым сигнальным процессорам, реализующим алгоритмы БПФ / С. Л. Волков *Наукові записки УНДІЗ*. №1. 2008. С. 25-31.
9. Ганжела С. І. Основи інформатики з елементами програмування та сучасні інформаційні технології навчання. Ч. ІІ. Елементи програмування / С. І. Ганжела, С. О. Шлянчак. Кропивницький: РВВ ЦДПУ ім. В. Винниченка, 2017. 61 с.
10. Гриб'юк О. О. Педагогічне проектування комп'ютерно орієнтованого середовища навчання дисциплін природничо-математичного циклу. / Гриб'юк О. О. // *Наукові записки. Випуск 7. Серія: Проблеми методики фізико-математичної і технологічної освіти*. Частина 3. Кіровоград.: РВВ КДПУ ім. В. Винниченка, 2015. С. 38–50.

11. Григорович В. Г. Методичні вказівки до виконання лабораторних і практичних завдань з курсу «Алгоритмізація та програмування». Частина 1 : навч. посібник / В. Г. Григорович. Дрогобич: Редакційно-видавничий відділ Дрогобицького державного педагогічного університету імені Івана Франка, 2016. 1400 с.
12. Додонов А. Г. Розпізнавання інформаційних операцій / А. Г. Додонов, Д. В Ландэ. К.: ООО «Инжиниринг», 2017. 284 с.
13. Морзе Н.В. Інформатика: підруч. для 11 кл. загальноосвіт. навч. закл.: рівень стандарту / Н. В. Морзе, О. В. Барна, В. П. Вембер [та ін.]. К.: Школяр, 2015. 304 с.
14. Иванова Е. В. Параллельная декомпозиция реляционных операций на основе распределенных колоночных индексов / Е. В. Иванова, Л. Б. Соколинский *Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика.* 2015. Т. 4, № 4. С. 80–100.
15. Иванов М. А. Теория, применение и оценка качества генераторов псевдослучайных последовательностей. / М. А. Иванов, И. В. Чугунков : КУДИЦ-ОБРАЗ, 2003. 240 с.
16. Караванова Т. П. Інформатика: методи побудови алгоритмів та їх аналіз. Необчислювальні алгоритми: Навч. посіб. для 9–10 кл. із поглибл. вивч. інформатики. / Т. П. Караванова Генеза. 2007. 216 с.
17. Клакович Л. М. Теорія алгоритмів: навч. посібник – 2-е вид., доп. / Л. М. Клакович Львів: ЛНУ, 2015. – 310 с.
18. Клейнберг Дж. Алгоритмы: разработка и применение. Классика Computers Science / Дж .Клейнберг ., Е. Тардос Пер. с англ. Е. Матвеева. СПб.: Питер, 2016. 800 с.
19. Кормен Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн; пер. с англ. под ред. А. Шеня. М.: ООО «И. Д. Вильямс», 2013. 1398 с.

20. Кравець П. Об’єктно-орієнтоване програмування: навч. посібник / П. О. Кравець. Львів: Видавництво Львівської політехніки, 2012. 624 с.
21. Кузьменко Н. Г. Комп’ютерні мережі та мережеві технології / Н. Г. Кузьменко. СПб.: Наука і техніка, 2013. 368 с.
22. Куроуз Д. Комп’ютерні мережі. Спадний підхід / Д. Куроуз, К. Росс. М.: Ексмо, 2016. 912 с.
23. Лобанов Л. М. Применение современных информационных технологий для решения задач автоматизации технологических процессов / Л. М. Лобанов, Е. М. Шаповалов, В. А. Коляда. // *Техническая диагностика и неразрушающий контроль.* 2014. №4. С. 52–56.
24. Матвієнко М. П. Алгоритми та структури даних: навч. посіб. / М. П. Матвієнко. К.: Видавництво «Ліра-К», 2014. 340 с.
25. Пономаренко В. С. Методи та моделі розроблення комп’ютерних систем і мереж: монографія / В. С. Пономаренко, С. В. Мінухін, С. В. Кавун, Пономаренка В. С. Х.: Вид. ХНЕУ, 2016. 316 с.
26. Оліфер В. Г. Мережеві операційні системи / В. Г. Оліфер. СПб.: Пітер, 2016. 544 с.
27. Олифер В. Г. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. / В. Г. Олифер, Н. А. Олифер СПб.: Питер, 2010. 672 с.
28. Олифер В. Г. Новые технологии и оборудование IP сетей [Текст] / В. Г. Олифер, Н. А. Олифер. СПб.: БХВ, 2012. 512 с.
29. Виснадул Б. Д. Основы компьютерных сетей: учеб. пособие / Б. Д. Виснадул, С. А. Лупин. С. В. Сидоров, П. Ю. Чумаченко / Под ред. Л. Г. Гагариной. М.: ИД «Форум»: ИНФРА-М. 2007. 272 с.
30. Панфілов К. В. Аналіз систем моніторингу мережевого обладнання мережі передачі даних / К. В. Панфілов. Самара: ПГУТИ, 2019. 96 с.
31. Пекарський Б. Г. Основи програмування: навчальний посібник./ Б. Г. Пекарський Рек. МОН. К.: Кондор, 2018. 270 с.

32. Прончев Г. Б. Комп’ютерні комунікації. Найпростіші обчислювальні мережі: Навчальний посібник / Г. Б. Прончев. М.: КДУ, 2009. 64 с.
33. Степанишин Ю. В. Інформаційні технології та комп’ютерна інженерія. / Ю. В. Степанишин, М. Л. Гаєвський 2007. № 3 С. 86–93.
34. Столлингс В. Современные компьютерные сети: пер. с англ. / В. Столлингс. Санкт-Петербург: Питер, 2003. 783 с.
35. Таненбаум Э. Компьютерные сети: пер. с англ. / Э. Таненбаум. Санкт-Петербург: Питер, 2003. 992 с.
36. Филимонов А. Ю. Построение мультисервисных сетей Ethernet: учебное пособие / А. Ю. Филимонов. СПб.: БХВ–Петербург, 2015. 248 с.
37. Фурашев В. М. Інформаційні операції крізь призму системи моніторингу та інтеграції Інтернет-ресурсів / В. М. Фурашев, Д. В. Ланде // *Правова інформатика*. 2009. № 2 (22). С. 49–57.
38. Хоменко В. Г. Комп’ютерні мережі: Навчальний посібник / В. Г. Хоменко, М. П. Павленко. Донецьк: ЛАНДОН-ХХІ, 2011. 316 с.
39. Шаньгин В. Защита информации в компьютерных системах и сетях / Шаньгин В. // *ДМК Пресс*. 2013. С. 65.
40. Шиндер Л. Д. Основи комп’ютерних мереж / Л. Д. Шиндер. М.: 2015. 152 с.
41. Руденко В. Д. Базовий курс інформатики / В. Д. Руденко [Навч. посіб.]. К.: Вид. група BHV. Кн. 1: Основи інформатики. 2015. 320 с.
42. Юрченко І. В. Інформатика та програмування. Частина 1. Навчальний посібник. / І. В. Юрченко Чернівці: Книги-ХХІ, 2015. 203 с.
43. Юрченко І. В. Інформатика та програмування. Частина 2. / І. В. Юрченко, В. С. Сікора Чернівці: Видавець Яворський С. Н., 2015. 210 с.
44. Назаров А. В. Эксплуатация объектов сетевой инфраструктуры: учебник для студ. учреждений сред. проф. образования [Текст] / А. В. Назаров, В. П. Мельников; под ред. А. В. Назарова. М.: Издательский центр «Академия», 2014. 538 с.

45. Kucherov D. P. Control System Objects with Multiple Stream of Information / D. P. Kucherov, A. N. Kozub // *Proceedings 2015 IEEE 3rd International Conference «Actual Problems of Unmanned Aerial Vehicles Developments (APUAVD)*», October 13–15, 2015. P. 290–293.
46. Методичні вказівки до виконання студентами-магістрантами економічної частини магістерських кваліфікаційних робіт / Уклад. В. О. Козловський – Вінниця: ВНТУ, 2012. – 22 с.

ДОДАТОК А. Технічне завдання

ВНТУ

ЗАТВЕРДЖУЮ
Завідувач
кафедри ПЗд.т.н.,
проф. О.Н. Романюк
“09” березня 2021р.

ТЕХНІЧНЕ ЗАВДАННЯ
на магістерську кваліфікаційну роботу зі спеціальності 121 – Інженерія програмного забезпечення
студенту групи 2ПІ-19м Брюханову Володимиру Сергійовичу

Керівник магістерської кваліфікаційної роботи:

к. т. н., доц., О. М. Рейда

"___" _____20___р.

Виконав:

студент гр. 2ПІ-19м В.С.Брюханов

"___" _____20___р.

Вінниця ВНТУ 2021

1.1 Найменування та галузь застосування

Програмна система оптимізації обміну даних у паралельних обчисленнях при обробці баз даних. Згідно отриманого завдання кінцевий програмний продукт може використовуватись кінцевими користувачами.

1.2 Підстава для проведення робіт

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол № 15 від 17.02.2021.

1.3 Мета та призначення роботи

Мета роботи полягає у підвищенні взаємодії користувача з великими масивами даних за допомогою паралельної обробки запитів.

Основними задачами дослідження є:

– провести аналіз методів розробки автоматизованих систем паралельної обробки баз даних;
– запропонувати нові:
  - метод паралельної організації запитів великих баз даних
  - метод адаптивної оптимізації ресурсів при виконанні паралельних запитів великих баз даних
– розробити програмні компоненти та систему на основі запропонованих методів;
– провести експериментальні дослідження розроблених засобів.

1.4 Технічні вимоги

− операційна система – Windows;
− мова програмування – SQL;
− середовище розробки – SQL Server Management Studio;
− тип резервування – інкрементне.

1.5 Перелік технічної документації, що пред’являється по закінченню робіт:

– пояснювальна записка;
– лістинги програми.

1.6 Економічні показники

– абсолютна ефективність вкладених інвестицій – 1435231,84грн;

– загальна вартість витрат на розробку і впровадження – 90735,09 грн;

– щорічна ефективність інвестицій – 156%;

– термін окупності – 0,64 роки.

1.7 Стадії і етапи розробки

Завдання на проектування видане 17 лютого 2021 року. Проектування та дослідження повинно бути завершеним до 1 червня 2021 року.

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів проекту (роботи)
1	Обґрунтування вибору методу розробки та постановка задачі дослідження	20.02.2021_– 12.03.2021
2	Розробка програмної архітектури програмно-інформаційної системи для роботи з базою даних	13.03.2021– 31.03.2021
3	Опис структури та методів обробки даних	01.04.2021– 25.04.2021
4	Тестування роботи та аналіз результатів	26.04.2021– 30.04.2021__
5	Економічне обґрунтування доцільності розробки	01.05.2021– 16.05.2021
6	Оформлення матеріалів до захисту МДР	17.05.2021 – 31.05.2021

1.7 Порядок контролю і приймання

Порядок контролю і приймання роботи регламентується відповідними документами ВНТУ і державними стандартами.

ДОДАТОК Б. Ілюстративний матеріал

ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДО ЗАХИСТУ МАГІСТЕРСЬКОЇКВАЛІФІКАЦІЙНОЇ РОБОТИ

Завідувач кафедри ПЗ, д.т.н., професор _____О. Н. Романюк

Науковий керівник, к.т.н., доц. кафедри ПЗ _____О. М. Рейда

Рецензент, д.т.н., проф. каф. КН _____О. М Василевський

Нормоконтроль, к.т.н., доц. кафедри ПЗ _____О. М. Рейда

Виконавець, студент групи 2ПІ-19м _____В. С. Брюханов

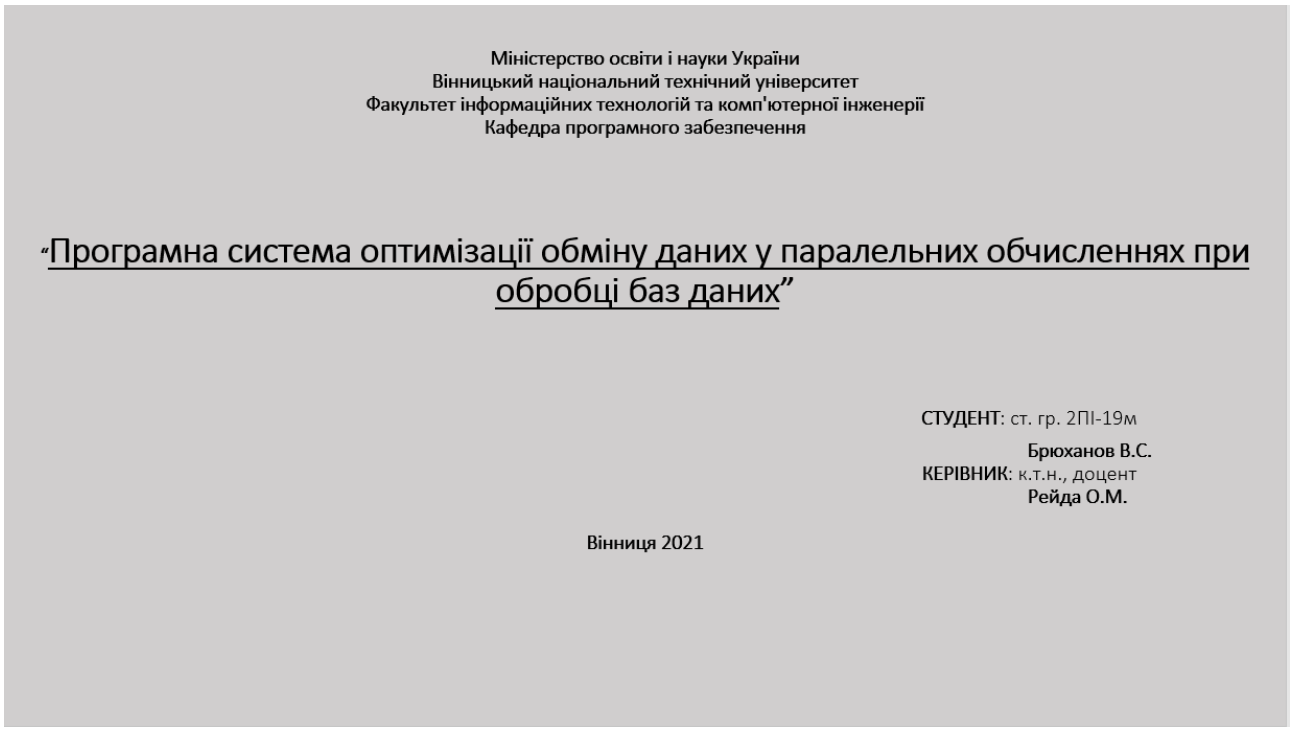


Рисунок Б.1. Плакат 1

Актуальність. Актуальність роботи безпосередньо пов’язана із підвищення продуктивності роботи із великими базами даних , що значно підвищує швидкодію обробки і якість отриманих даних.

Мета і завдання дослідження. Метою роботи є підвищення взаємодії користувача з великими масивами даних за допомогою паралельної обробки запитів.

Предмет дослідження –методи та засоби реалізації процесу паралельної обробки даних для підвищення швидкодії використання великих бази даних у програмно-інформаційній системі.

Об’єктом дослідження - процес паралельної обробки даних для підвищення швидкодії використання великих бази даних у програмно-інформаційній системі.

Рисунок Б.2. Плакат 2

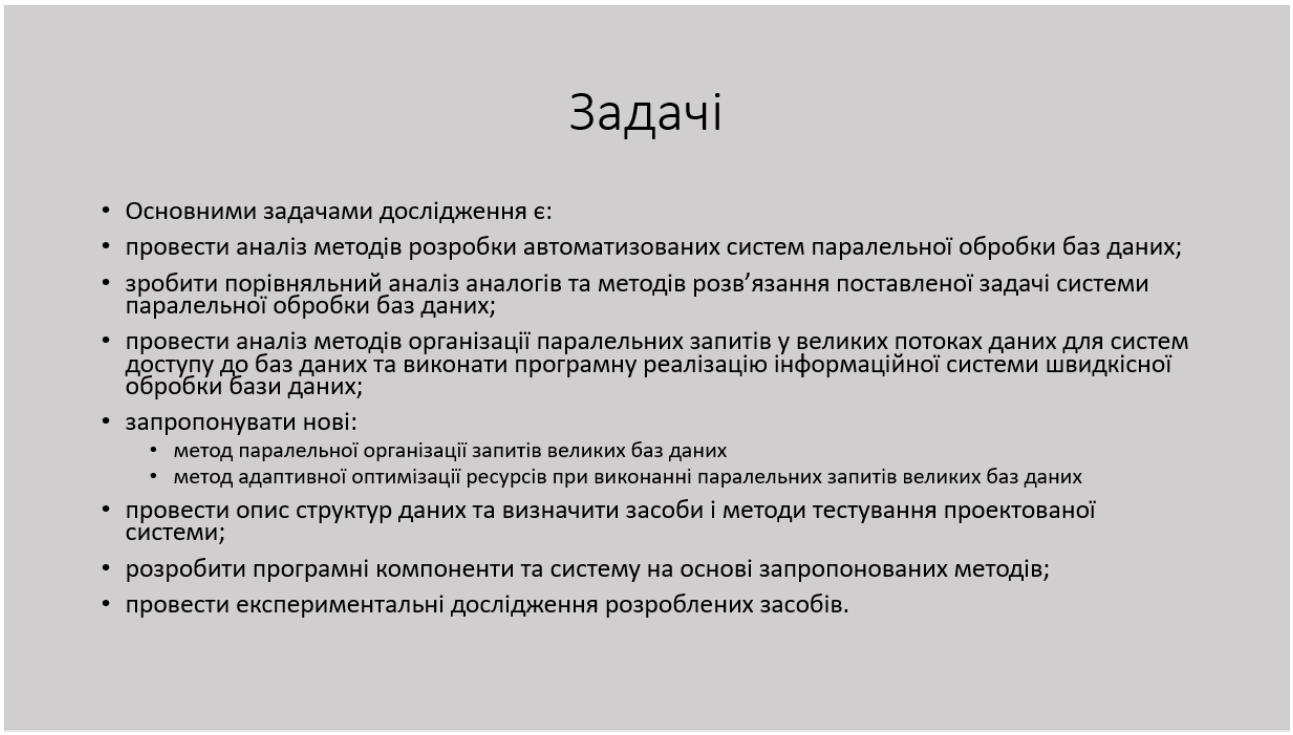


Рисунок Б.3. Плакат 3

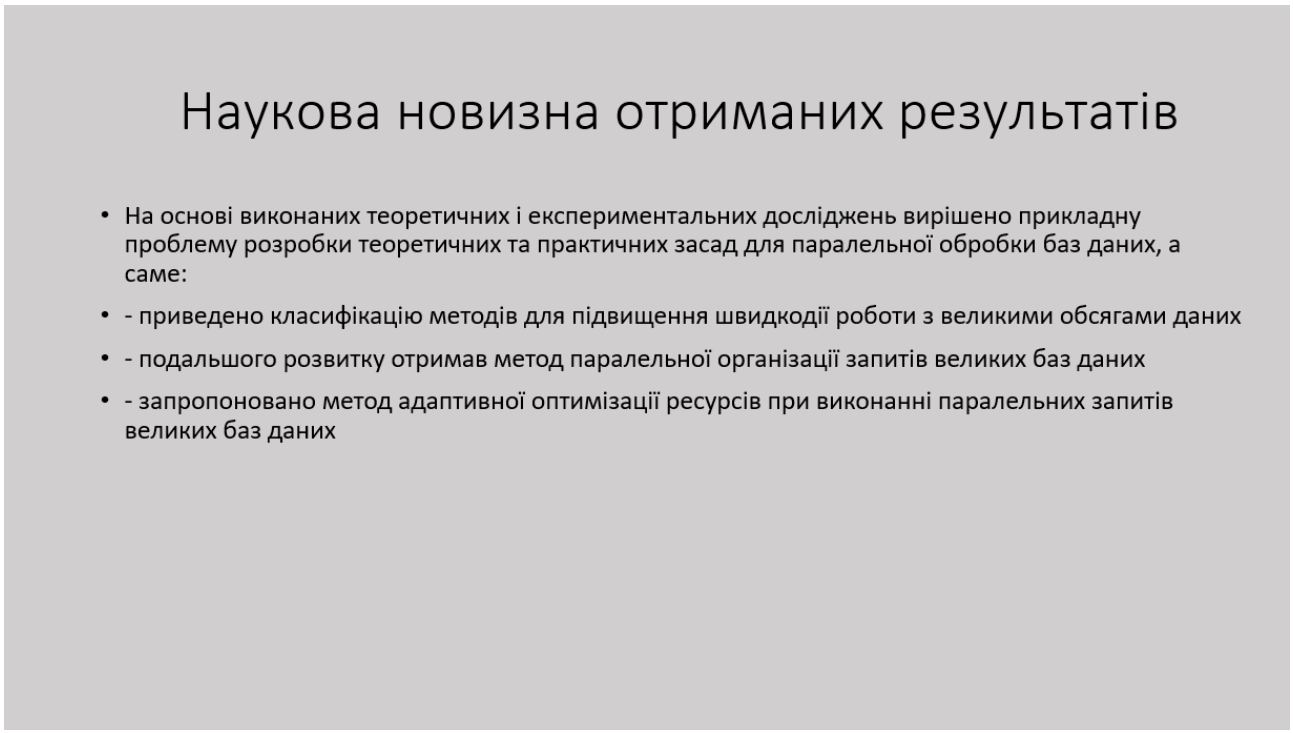


Рисунок Б.4. Плакат 4

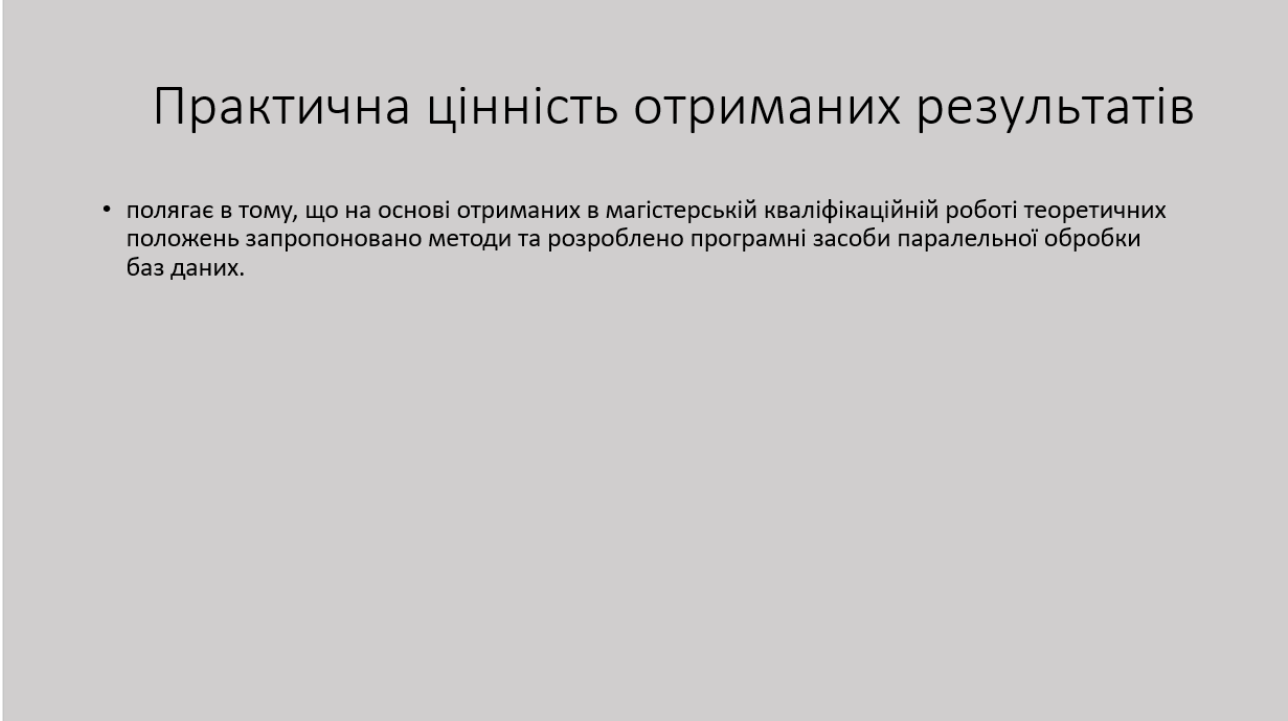


Рисунок Б.5. Плакат 5

	SN	CE	CD	CDN
Масштабованість	2	3	3	3
Наявність даних	2	1	3	3
Балансування навантаження	0	2	1	1
Інтерпроцесор комунікації	0	2	1	1
Когерентність кешу	3	2	0	3
Контроль паралельності	3	2	0	3
	10	12	8	14

Рисунок Б.6. Плакат 6

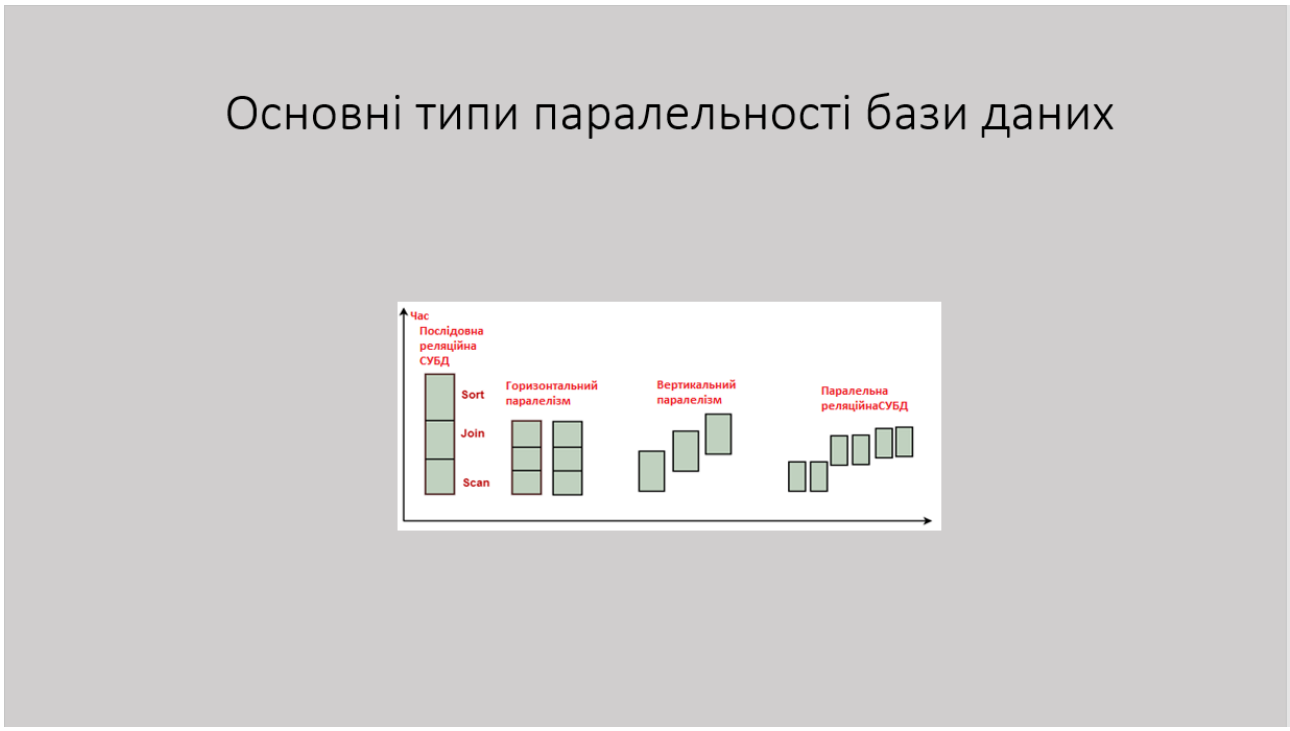


Рисунок Б.7. Плакат 7

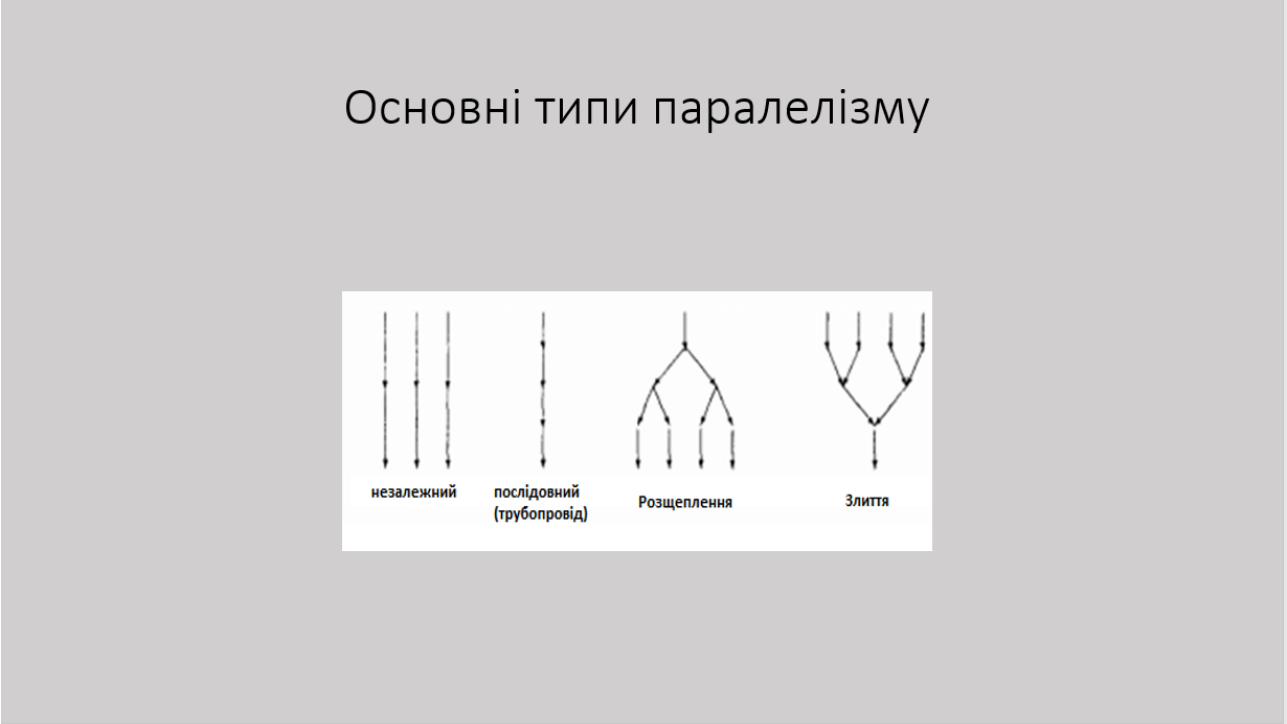


Рисунок Б.8. Плакат 8

Схематичне зображення взаємодії SQL-сервера з Ксп (з розпаралелюванням).

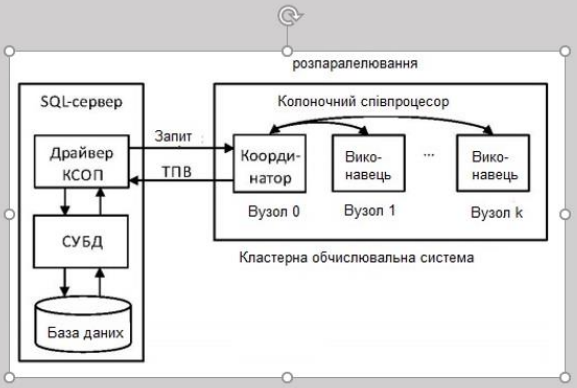


Рисунок Б.9. Плакат 9

Рисунок Б.10. Плакат 10

Результати тестування

Результати тестування до впровадження алгоритму паралельної обробки даних

Кількість елементів	Час на перерахунок порядку (сек.)	Час на підрахунок нових значень (сек.)
100	0,007	0,003
1000	0,05	0,036
10000	0,3	0,2
100000	2,7	1,8
1000000	11,36	9,4

Результати тестування після впровадження алгоритму паралельної обробки даних

Кількість елементів	Час на перерахунок порядку (сек.)	Час на підрахунок нових значень (сек.)
100	0,006	0,003
1000	0,04	0,029
10000	0,23	0,09
100000	1,9	1,1
1000000	7,3	5,4

Рисунок Б.11. Плакат 11

Результати тестування(на даних користувачах)

Результати тестування до впровадження алгоритму паралельної обробки даних

Середовище	Час на перерахунок порядку (сек.)	Час на підрахунок нових значень (сек.)
stage	5,4	3,1
prod	15,6	12,3

Результати тестування після впровадження алгоритму паралельної обробки даних

Середовище	Час на перерахунок порядку (сек.)	Час на підрахунок нових значень (сек.)
stage	4,6	2,5
prod	10,9	9

Рисунок Б.12. Плакат 12

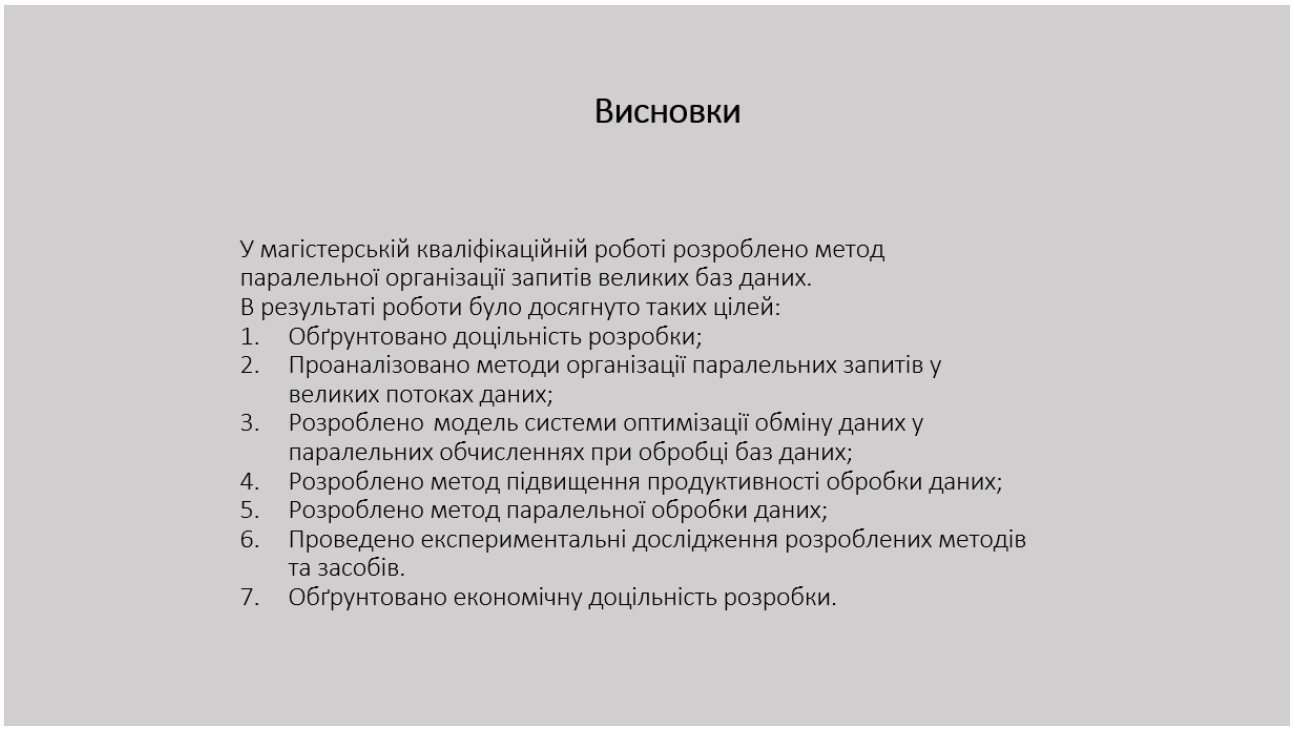


Рисунок Б.13. Плакат 13

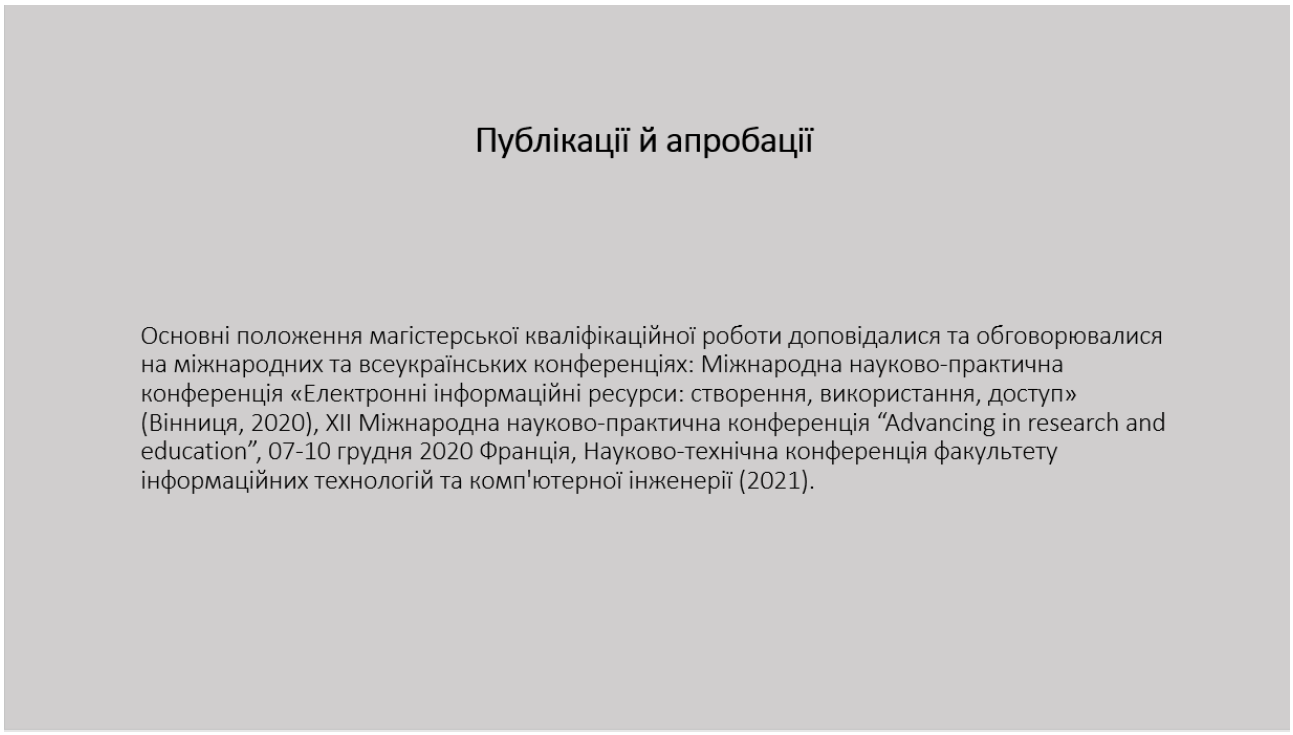


Рисунок Б.14. Плакат 14

ДОДАТОК В. Лістинг програми

```
CREATE TABLE Koristyvach
(
Name VARCHAR2 (200) NOT NULL,
Familia VARCHAR2 (200) NOT NULL,
God_Rogdenia DATE NULL,
adres_progivania NUMBER (30) NULL,
stati VARCHAR2 (200) NULL,
kod_firn NUMBER (30) NOT NULL
);
CREATE UNIQUE INDEX XPKКористувач ON Koristyvach
(Name ASC, Familia ASC, kod_firn ASC);
ALTER TABLE Koristyvach
ADD CONSTRAINT XPKКористувач PRIMARY KEY (Name, Familia,
kod_firn);
CREATE TABLE Korzina
(
kod_tovary NUMBER (30) NULL,
name_tovary VARCHAR2 (200) NULL,
name_firm VARCHAR2 (200) NULL,
kod_firm NUMBER (30) NOT NULL,
Price CHAR (18) NULL
);
CREATE UNIQUE INDEX XPKКорзина ON Korzina
(kod_firm ASC);
ALTER TABLE Korzina
ADD CONSTRAINT XPKКорзина PRIMARY KEY (kod_firm);
CREATE TABLE Tovari
(
name_firm VARCHAR2 (200) NULL,
name_models VARCHAR2 (200) NULL,
kod_models NUMBER (30) NOT NULL,
name_tovaru VARCHAR2 (200) NULL,
kod_tovaru NUMBER (30) NOT NULL,
Price CHAR (18) NULL,
kod_firm NUMBER (30) NOT NULL,
kod_obladnanya NUMBER (30) NOT NULL
);
CREATE UNIQUE INDEX XPKТовари ON Tovari
(kod_firm ASC, kod_models ASC, kod_tovaru ASC, kod_obladnanya
ASC);
ALTER TABLE Tovari
ADD CONSTRAINT XPKТовари PRIMARY KEY (kod_firm, kod_models,
kod_tovaru, kod_obladnanya);
CREATE TABLE Firm
(
name_firm VARCHAR2 (200) NOT NULL,
kod_modeli NUMBER (30) NOT NULL,
adres_firm VARCHAR2 (200) NULL,
kod_firm NUMBER (30) NOT NULL,
```

```
     kod_tovaru NUMBER (30) NOT NULL,
     kod_obladnya NUMBER (30) NOT NULL,
     name VARCHAR2 (200) NOT NULL,
     familia VARCHAR2 (200) NOT NULL,
     kod_tovary NUMBER (30) NOT NULL,
     name_models VARCHAR2 (200) NOT NULL
     );
     CREATE UNIQUE INDEX XPKФірма ON Firm
     (kod_firm ASC, kod_modeli ASC, kod_tovaru ASC, kod_obladnya
ASC, name ASC, familia ASC, name_firm ASC, name_models ASC);
     ALTER TABLE Firm
     ADD CONSTRAINT XPKФірма PRIMARY KEY (kod_firm, kod_modeli,
kod_tovaru, kod_obladnya, name, familia, name_firm, name_models);
     CREATE TABLE Models
     (
     kod_modely NUMBER (30) NOT NULL,
     Name_Firm VARCHAR2 (200) NULL,
     name_models VARCHAR2 (200) NULL,
     Price CHAR (18) NULL,
     kod_firm NUMBER (30) NOT NULL,
     kod_komplektuechgo NUMBER (30) NOT NULL,
     kod_obladnya NUMBER (30) NOT NULL,
     Characteristics VARCHAR2 (2500) NULL
     );
     CREATE UNIQUE INDEX XPKМодель ON Models
     (kod_modely ASC, kod_firm ASC, kod_komplektuechgo ASC,
kod_obladnya ASC);
     ALTER TABLE Models
     ADD CONSTRAINT XPKМодель PRIMARY KEY (kod_modely, kod_firm,
kod_komplektuechgo, kod_obladnya);
     CREATE TABLE News
     (
     Name_firm VARCHAR2 (200) NOT NULL,
     Name_models VARCHAR2 (200) NOT NULL,
     Data INTERVAL YEAR TO MONTH NULL,
     inPrice CHAR (18) NULL
     );
     CREATE UNIQUE INDEX XPKНовинки ON News
     (Name_firm ASC, Name_models ASC);
     ALTER TABLE News
     ADD CONSTRAINT XPKНовинки PRIMARY KEY (Name_firm, Name_models);
     CREATE TABLE Obladnanya
     (
     kod_obladnanya NUMBER (30) NOT NULL,
     Name_obladnanya VARCHAR2 (200) NULL,
     Price CHAR (18) NULL
     );
     CREATE UNIQUE INDEX XPKОбладнання ON Obladnanya
     (kod_obladnanya ASC);
     ALTER TABLE Obladnanya
     ADD CONSTRAINT XPKОбладнання PRIMARY KEY (kod_obladnanya);
     ALTER TABLE Obladnanya
```

```
     ADD CONSTRAINT Инв._номер UNIQUE (kod_obladnanya);
     CREATE TABLE Shogi_tovary
     (
     Name_Firm VARCHAR2 (200) NOT NULL,
     Name_tovary VARCHAR2 (200) NULL,
     kod_firm NUMBER (30) NOT NULL,
     kod_tovary NUMBER (30) NOT NULL,
     Price CHAR (18) NULL,
     Name_models VARCHAR2 (200) NOT NULL
     );
     CREATE UNIQUE INDEX XPKСхожі_Товари ON Shogi_tovary
     (Name_Firm ASC, kod_firm ASC, kod_tovary ASC, Name_models ASC);
     ALTER TABLE Shogi_tovary
     ADD CONSTRAINT XPKСхожі_Товари PRIMARY KEY (Name_Firm,
kod_firm, kod_tovary, Name_models);
     ALTER TABLE Firm
     ADD (CONSTRAINT R_1 FOREIGN KEY (kod_modeli, kod_firm,
kod_tovaru, kod_obladnya) REFERENCES Models (kod_modely, kod_firm,
kod_komplektuechgo, kod_obladnya));
     ALTER TABLE Firm
     ADD (CONSTRAINT R_5 FOREIGN KEY (name, familia, kod_firm)
REFERENCES Koristyvach (Name, Familia, kod_firn));
     ALTER TABLE Firm
     ADD (CONSTRAINT R_7 FOREIGN KEY (name_firm, kod_firm,
kod_tovary, name_models) REFERENCES Shogi_tovary (Name_Firm,
kod_firm, kod_tovary, Name_models));
     ALTER TABLE Tovari
     ADD (CONSTRAINT R_3 FOREIGN KEY (kod_obladnanya) REFERENCES
Obladnanya (kod_obladnanya))
     ALTER TABLE Koristyvach
     ADD (CONSTRAINT R_4 FOREIGN KEY (kod_firn) REFERENCES Korzina
(kod_firm));
     ALTER TABLE Models
     ADD (CONSTRAINT R_2 FOREIGN KEY (kod_firm, kod_modely,
kod_komplektuechgo, kod_obladnya) REFERENCES Tovari (kod_firm,
kod_models, kod_tovaru, kod_obladnanya));
     ALTER TABLE Shogi_tovary
     ADD (CONSTRAINT R_6 FOREIGN KEY (Name_Firm, Name_models)
REFERENCES News (Name_firm, Name_models));
     CREATE TRIGGER tI_Firm BEFORE INSERT ON Firm for each row
     - ERwin Builtin Trigger
     - INSERT trigger on Firm
     DECLARE NUMROWS INTEGER;
     BEGIN
     /· ERwin Builtin Trigger ·/
     /· Models Firm on child insert restrict ·/
     /·   ERWIN_RELATION:   CHECKSUM=«0003b9bb»,   PARENT_OWNER=««,
PARENT_TABLE=«Models»
     CHILD_OWNER=««, CHILD_TABLE=«Firm»
     P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
     FK_CONSTRAINT=«R_1»,     FK_COLUMNS=«kod_modeli»     «kod_firm»
«kod_tovaru» «kod_obladnya» ·/
```

```
     SELECT count(·) INTO NUMROWS
     FROM Models
     WHERE
     /·%JoinFKPK(:%New, Models,» = «,» AND») ·/
     :new. kod_modeli = Models. kod_modely AND
     :new. kod_firm = Models. kod_firm AND
     :new. kod_tovaru = Models. kod_komplektuechgo AND
     :new. kod_obladnya = Models. kod_obladnya;
     IF (
     /·%NotnullFK(:%New,» IS NOT NULL AND») ·/
     NUMROWS = 0
     )
     THEN
     raise_application_error(
     -20002,
     'Cannot insert Firm because Models does not exist.'
     );
     END IF;
     /· ERwin Builtin Trigger ·/
     /· Koristyvach Firm on child insert restrict ·/
     /·   ERWIN_RELATION:   CHECKSUM=«00000000»,   PARENT_OWNER=««,
PARENT_TABLE=«Koristyvach»
     CHILD_OWNER=««, CHILD_TABLE=«Firm»
     P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
     FK_CONSTRAINT=«R_5», FK_COLUMNS=«name» «familia» «kod_firm» ·/
     SELECT count(·) INTO NUMROWS
     FROM Koristyvach
     WHERE
     /·%JoinFKPK(:%New, Koristyvach,» = «,» AND») ·/
     :new. name = Koristyvach. Name AND
     :new. familia = Koristyvach. Familia AND
     :new. kod_firm = Koristyvach. kod_firn;
     IF (
     /·%NotnullFK(:%New,» IS NOT NULL AND») ·/
     NUMROWS = 0
     )
     THEN
     raise_application_error(
     -20002,
     'Cannot insert Firm because Koristyvach does not exist.'
     );
     END IF;
     /· ERwin Builtin Trigger ·/
     /· Shogi_tovary Firm on child insert restrict ·/
     /·   ERWIN_RELATION:   CHECKSUM=«00000000»,   PARENT_OWNER=««,
PARENT_TABLE=«Shogi_tovary»
     CHILD_OWNER=««, CHILD_TABLE=«Firm»
     P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
     FK_CONSTRAINT=«R_7»,     FK_COLUMNS=«name_firm»     «kod_firm»
«kod_tovary» «name_models» ·/
     SELECT count(·) INTO NUMROWS
     FROM Shogi_tovary
```

```
    WHERE
    /·%JoinFKPK(:%New, Shogi_tovary,» = «,» AND») ·/
    :new. name_firm = Shogi_tovary. Name_Firm AND
    :new. kod_firm = Shogi_tovary. kod_firm AND
    :new. kod_tovary = Shogi_tovary. kod_tovary AND
    :new. name_models = Shogi_tovary. Name_models;
    IF (
    /·%NotnullFK(:%New,» IS NOT NULL AND») ·/
    NUMROWS = 0
    )
    THEN
    raise_application_error(
    -20002,
    'Cannot insert Firm because Shogi_tovary does not exist.'
    );
    END IF;
    - ERwin Builtin Trigger
    END;
    /
    CREATE TRIGGER tU_Firm AFTER UPDATE ON Firm for each row
    - ERwin Builtin Trigger
    - UPDATE trigger on Firm
    DECLARE NUMROWS INTEGER;
    BEGIN
    /· ERwin Builtin Trigger ·/
    /· Models Firm on child update restrict ·/
    /·   ERWIN_RELATION:   CHECKSUM=«0003a835»,   PARENT_OWNER=««,
PARENT_TABLE=«Models»
    CHILD_OWNER=««, CHILD_TABLE=«Firm»
    P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
    FK_CONSTRAINT=«R_1»,    FK_COLUMNS=«kod_modeli»    «kod_firm»
«kod_tovaru» «kod_obladnya» ·/
    SELECT count(·) INTO NUMROWS
    FROM Models
    WHERE
    /·%JoinFKPK(:%New, Models,» = «,» AND») ·/
    :new. kod_modeli = Models. kod_modely AND
    :new. kod_firm = Models. kod_firm AND
    :new. kod_tovaru = Models. kod_komplektuechgo AND
    :new. kod_obladnya = Models. kod_obladnya;
    IF (
    /·%NotnullFK(:%New,» IS NOT NULL AND») ·/
    NUMROWS = 0
    )
    THEN
    raise_application_error(
    -20007,
    'Cannot update Firm because Models does not exist.'
    );
    END IF;
    /· ERwin Builtin Trigger ·/
    /· Koristyvach Firm on child update restrict ·/
```

```
     /·   ERWIN_RELATION:   CHECKSUM=«00000000»,   PARENT_OWNER=««,
PARENT_TABLE=«Koristyvach»
     CHILD_OWNER=««, CHILD_TABLE=«Firm»
     P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
     FK_CONSTRAINT=«R_5», FK_COLUMNS=«name» «familia» «kod_firm» ·/
     SELECT count(·) INTO NUMROWS
     FROM Koristyvach
     WHERE
     /·%JoinFKPK(:%New, Koristyvach,» = «,» AND») ·/
     :new. name = Koristyvach. Name AND
     :new. familia = Koristyvach. Familia AND
     :new. kod_firm = Koristyvach. kod_firn;
     IF (
     /·%NotnullFK(:%New,» IS NOT NULL AND») ·/
     NUMROWS = 0
     )
     THEN
     raise_application_error(
     -20007,
     'Cannot update Firm because Koristyvach does not exist.'
     );
     END IF;
     /· ERwin Builtin Trigger ·/
     /· Shogi_tovary Firm on child update restrict ·/
     /·   ERWIN_RELATION:   CHECKSUM=«00000000»,   PARENT_OWNER=««,
PARENT_TABLE=«Shogi_tovary»
     CHILD_OWNER=««, CHILD_TABLE=«Firm»
     P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
     FK_CONSTRAINT=«R_7»,     FK_COLUMNS=«name_firm»     «kod_firm»
«kod_tovary» «name_models» ·/
     SELECT count(·) INTO NUMROWS
     FROM Shogi_tovary
     WHERE
     /·%JoinFKPK(:%New, Shogi_tovary,» = «,» AND») ·/
     :new. name_firm = Shogi_tovary. Name_Firm AND
     :new. kod_firm = Shogi_tovary. kod_firm AND
     :new. kod_tovary = Shogi_tovary. kod_tovary AND
     :new. name_models = Shogi_tovary. Name_models;
     IF (
     /·%NotnullFK(:%New,» IS NOT NULL AND») ·/
     NUMROWS = 0
     )
     THEN
     raise_application_error(
     -20007,
     'Cannot update Firm because Shogi_tovary does not exist.'
     );
     END IF;
     - ERwin Builtin Trigger
     END;
     /
     CREATE TRIGGER tI_Tovari BEFORE INSERT ON Tovari for each row
```

```
     - ERwin Builtin Trigger
     - INSERT trigger on Tovari
     DECLARE NUMROWS INTEGER;
     BEGIN
     /· ERwin Builtin Trigger ·/
     /· Obladnanya Tovari on child insert restrict ·/
     /·   ERWIN_RELATION:   CHECKSUM=«000104c6»,   PARENT_OWNER=««,
PARENT_TABLE=«Obladnanya»
     CHILD_OWNER=««, CHILD_TABLE=«Tovari»
     P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
     FK_CONSTRAINT=«R_3», FK_COLUMNS=«kod_obladnanya» ·/
     SELECT count(·) INTO NUMROWS
     FROM Obladnanya
     WHERE
     /·%JoinFKPK(:%New, Obladnanya,» = «,» AND») ·/
     :new. kod_obladnanya = Obladnanya. kod_obladnanya;
     IF (
     /·%NotnullFK(:%New,» IS NOT NULL AND») ·/
     NUMROWS = 0
     )
     THEN
     raise_application_error(
     -20002,
     'Cannot insert Tovari because Obladnanya does not exist.'
     );
     END IF;
     - ERwin Builtin Trigger
     END;
     /
     CREATE TRIGGER tD_Tovari AFTER DELETE ON Tovari for each row
     - ERwin Builtin Trigger
     - DELETE trigger on Tovari
     DECLARE NUMROWS INTEGER;
     BEGIN
     /· ERwin Builtin Trigger ·/
     /· Tovari Models on parent delete restrict ·/
     /·   ERWIN_RELATION:   CHECKSUM=«000124b7»,   PARENT_OWNER=««,
PARENT_TABLE=«Tovari»
     CHILD_OWNER=««, CHILD_TABLE=«Models»
     P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
     FK_CONSTRAINT=«R_2»,     FK_COLUMNS=«kod_firm»     «kod_modely»
«kod_komplektuechgo» «kod_obladnya» ·/
     SELECT count(·) INTO NUMROWS
     FROM Models
     WHERE
     /·%JoinFKPK (Models,:%Old,» = «,» AND») ·/
     Models. kod_firm =:old. kod_firm AND
     Models. kod_modely =:old. kod_models AND
     Models. kod_komplektuechgo =:old. kod_tovaru AND
     Models. kod_obladnya =:old. kod_obladnanya;
     IF (NUMROWS > 0)
     THEN
```

```
raise_application_error(
-20001,
'Cannot delete Tovari because Models exists.'
);
END IF;
- ERwin Builtin Trigger
END;
/
CREATE TRIGGER tU_Tovari AFTER UPDATE ON Tovari for each row
- ERwin Builtin Trigger
- UPDATE trigger on Tovari
DECLARE NUMROWS INTEGER;
BEGIN
/· ERwin Builtin Trigger ·/
/· Tovari Models on parent update restrict ·/
/·   ERWIN_RELATION:   CHECKSUM=«0002b5fc»,   PARENT_OWNER=««,
PARENT_TABLE=«Tovari»
CHILD_OWNER=««, CHILD_TABLE=«Models»
P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
FK_CONSTRAINT=«R_2»,     FK_COLUMNS=«kod_firm»     «kod_modely»
«kod_komplektuechgo» «kod_obladnya» ·/
IF
/·%JoinPKPK(:%Old,:%New,» <> «,» OR «) ·/
:old. kod_firm <>:new. kod_firm OR
:old. kod_models <>:new. kod_models OR
:old. kod_tovaru <>:new. kod_tovaru OR
:old. kod_obladnanya <>:new. kod_obladnanya
THEN
SELECT count(·) INTO NUMROWS
FROM Models
WHERE
/·%JoinFKPK (Models,:%Old,» = «,» AND») ·/
Models. kod_firm =:old. kod_firm AND
Models. kod_modely =:old. kod_models AND
Models. kod_komplektuechgo =:old. kod_tovaru AND
Models. kod_obladnya =:old. kod_obladnanya;
IF (NUMROWS > 0)
THEN
raise_application_error(
-20005,
'Cannot update Tovari because Models exists.'
);
END IF;
END IF;
/· ERwin Builtin Trigger ·/
/· Obladnanya Tovari on child update restrict ·/
/·   ERWIN_RELATION:   CHECKSUM=«00000000»,   PARENT_OWNER=««,
PARENT_TABLE=«Obladnanya»
CHILD_OWNER=««, CHILD_TABLE=«Tovari»
P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
FK_CONSTRAINT=«R_3», FK_COLUMNS=«kod_obladnanya» ·/
SELECT count(·) INTO NUMROWS
```

```
     FROM Obladnanya
     WHERE
     /·%JoinFKPK(:%New, Obladnanya,» = «,» AND») ·/
     :new. kod_obladnanya = Obladnanya. kod_obladnanya;
     IF (
     /·%NotnullFK(:%New,» IS NOT NULL AND») ·/
     NUMROWS = 0
     )
     THEN
     raise_application_error(
     -20007,
     'Cannot update Tovari because Obladnanya does not exist.'
     );
     END IF;
     - ERwin Builtin Trigger
     END;
     /
     CREATE TRIGGER tI_Koristyvach BEFORE INSERT ON Koristyvach for
each row
     - ERwin Builtin Trigger
     - INSERT trigger on Koristyvach
     DECLARE NUMROWS INTEGER;
     BEGIN
     /· ERwin Builtin Trigger ·/
     /· Korzina Koristyvach on child insert restrict ·/
     /·   ERWIN_RELATION:   CHECKSUM=«0000e9b6»,   PARENT_OWNER=««,
PARENT_TABLE=«Korzina»
     CHILD_OWNER=««, CHILD_TABLE=«Koristyvach»
     P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
     FK_CONSTRAINT=«R_4», FK_COLUMNS=«kod_firn» ·/
     SELECT count(·) INTO NUMROWS
     FROM Korzina
     WHERE
     /·%JoinFKPK(:%New, Korzina,» = «,» AND») ·/
     :new. kod_firn = Korzina. kod_firm;
     IF (
     /·%NotnullFK(:%New,» IS NOT NULL AND») ·/
     NUMROWS = 0
     )
     THEN
     raise_application_error(
     -20002,
     'Cannot insert Koristyvach because Korzina does not exist.'
     );
     END IF;
     - ERwin Builtin Trigger
     END;
     /
     CREATE TRIGGER tD_Koristyvach AFTER DELETE ON Koristyvach for
each row
     - ERwin Builtin Trigger
     - DELETE trigger on Koristyvach
```

```
     DECLARE NUMROWS INTEGER;
     BEGIN
     /· ERwin Builtin Trigger ·/
     /· Koristyvach Firm on parent delete restrict ·/
     /·   ERWIN_RELATION:   CHECKSUM=«0000ec92»,   PARENT_OWNER=««,
PARENT_TABLE=«Koristyvach»
     CHILD_OWNER=««, CHILD_TABLE=«Firm»
     P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
     FK_CONSTRAINT=«R_5», FK_COLUMNS=«name» «familia» «kod_firm» ·/
     SELECT count(·) INTO NUMROWS
     FROM Firm
     WHERE
     /·%JoinFKPK (Firm,:%Old,» = «,» AND») ·/
     Firm. name =:old. Name AND
     Firm. familia =:old. Familia AND
     Firm. kod_firm =:old. kod_firn;
     IF (NUMROWS > 0)
     THEN
     raise_application_error(
     -20001,
     'Cannot delete Koristyvach because Firm exists.'
     );
     END IF;
     - ERwin Builtin Trigger
     END;
     /
     CREATE TRIGGER tU_Koristyvach AFTER UPDATE ON Koristyvach for
each row
     - ERwin Builtin Trigger
     - UPDATE trigger on Koristyvach
     DECLARE NUMROWS INTEGER;
     BEGIN
     /· ERwin Builtin Trigger ·/
     /· Koristyvach Firm on parent update restrict ·/
     /·   ERWIN_RELATION:   CHECKSUM=«00023bc0»,   PARENT_OWNER=««,
PARENT_TABLE=«Koristyvach»
     CHILD_OWNER=««, CHILD_TABLE=«Firm»
     P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
     FK_CONSTRAINT=«R_5», FK_COLUMNS=«name» «familia» «kod_firm» ·/
     IF
     /·%JoinPKPK(:%Old,:%New,» <> «,» OR «) ·/
     :old. Name <>:new. Name OR
     :old. Familia <>:new. Familia OR
     :old. kod_firn <>:new. kod_firn
     THEN
     SELECT count(·) INTO NUMROWS
     FROM Firm
     WHERE
     /·%JoinFKPK (Firm,:%Old,» = «,» AND») ·/
     Firm. name =:old. Name AND
     Firm. familia =:old. Familia AND
     Firm. kod_firm =:old. kod_firn;
```

```
     IF (NUMROWS > 0)
     THEN
     raise_application_error(
     -20005,
     'Cannot update Koristyvach because Firm exists.'
     );
     END IF;
     END IF;
     /· ERwin Builtin Trigger ·/
     /· Korzina Koristyvach on child update restrict ·/
     /·   ERWIN_RELATION:   CHECKSUM=«00000000»,   PARENT_OWNER=««,
PARENT_TABLE=«Korzina»
     CHILD_OWNER=««, CHILD_TABLE=«Koristyvach»
     P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
     FK_CONSTRAINT=«R_4», FK_COLUMNS=«kod_firn» ·/
     SELECT count(·) INTO NUMROWS
     FROM Korzina
     WHERE
     /·%JoinFKPK(:%New, Korzina,» = «,» AND») ·/
     :new. kod_firn = Korzina. kod_firm;
     IF (
     /·%NotnullFK(:%New,» IS NOT NULL AND») ·/
     NUMROWS = 0
     )
     THEN
     raise_application_error(
     -20007,
     'Cannot update Koristyvach because Korzina does not exist.'
     );
     END IF;
     - ERwin Builtin Trigger
     END;
     /
     CREATE TRIGGER tD_Korzina AFTER DELETE ON Korzina for each row
     - ERwin Builtin Trigger
     - DELETE trigger on Korzina
     DECLARE NUMROWS INTEGER;
     BEGIN
     /· ERwin Builtin Trigger ·/
     /· Korzina Koristyvach on parent delete restrict ·/
     /·   ERWIN_RELATION:   CHECKSUM=«0000d7be»,   PARENT_OWNER=««,
PARENT_TABLE=«Korzina»
     CHILD_OWNER=««, CHILD_TABLE=«Koristyvach»
     P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
     FK_CONSTRAINT=«R_4», FK_COLUMNS=«kod_firn» ·/
     SELECT count(·) INTO NUMROWS
     FROM Koristyvach
     WHERE
     /·%JoinFKPK (Koristyvach,:%Old,» = «,» AND») ·/
     Koristyvach. kod_firn =:old. kod_firm;
     IF (NUMROWS > 0)
     THEN
```

```
raise_application_error(
-20001,
'Cannot delete Korzina because Koristyvach exists.'
);
END IF;
- ERwin Builtin Trigger
END;
/
CREATE TRIGGER tU_Korzina AFTER UPDATE ON Korzina for each row
- ERwin Builtin Trigger
- UPDATE trigger on Korzina
DECLARE NUMROWS INTEGER;
BEGIN
/· ERwin Builtin Trigger ·/
/· Korzina Koristyvach on parent update restrict ·/
/·   ERWIN_RELATION:   CHECKSUM=«0001065b»,   PARENT_OWNER=««,
PARENT_TABLE=«Korzina»
CHILD_OWNER=««, CHILD_TABLE=«Koristyvach»
P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
FK_CONSTRAINT=«R_4», FK_COLUMNS=«kod_firn» ·/
IF
/·%JoinPKPK(:%Old,:%New,» <> «,» OR «) ·/
:old. kod_firm <>:new. kod_firm
THEN
SELECT count(·) INTO NUMROWS
FROM Koristyvach
WHERE
/·%JoinFKPK (Koristyvach,:%Old,» = «,» AND») ·/
Koristyvach. kod_firn =:old. kod_firm;
IF (NUMROWS > 0)
THEN
raise_application_error(
-20005,
'Cannot update Korzina because Koristyvach exists.'
);
END IF;
END IF;
- ERwin Builtin Trigger
END;
/ CREATE TRIGGER tI_Models BEFORE INSERT ON Models for each row
- ERwin Builtin Trigger
- INSERT trigger on Models
DECLARE NUMROWS INTEGER;
BEGIN
/· ERwin Builtin Trigger ·/
/· Tovari Models on child insert restrict ·/
/·   ERWIN_RELATION:   CHECKSUM=«00015c81»,   PARENT_OWNER=««,
PARENT_TABLE=«Tovari»
CHILD_OWNER=««, CHILD_TABLE=«Models»
P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
FK_CONSTRAINT=«R_2»,     FK_COLUMNS=«kod_firm»     «kod_modely»
«kod_komplektuechgo» «kod_obladnya» ·/
```

```
     SELECT count(·) INTO NUMROWS
     FROM Tovari
     WHERE
     /·%JoinFKPK(:%New, Tovari,» = «,» AND») ·/
     :new. kod_firm = Tovari. kod_firm AND
     :new. kod_modely = Tovari. kod_models AND
     :new. kod_komplektuechgo = Tovari. kod_tovaru AND
     :new. kod_obladnya = Tovari. kod_obladnanya;
     IF (
     /·%NotnullFK(:%New,» IS NOT NULL AND») ·/
     NUMROWS = 0
     )THEN
     raise_application_error(
     -20002,
     'Cannot insert Models because Tovari does not exist.'
     );
     END IF;
     - ERwin Builtin Trigger
     END;
     /
     CREATE TRIGGER tD_Models AFTER DELETE ON Models for each row
     - ERwin Builtin Trigger
     - DELETE trigger on Models
     DECLARE NUMROWS INTEGER;
     BEGIN
     /· ERwin Builtin Trigger ·/
     /· Models Firm on parent delete restrict ·/
     /·   ERWIN_RELATION:   CHECKSUM=«00012069»,   PARENT_OWNER=««,
PARENT_TABLE=«Models»
     CHILD_OWNER=««, CHILD_TABLE=«Firm»
     P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
     FK_CONSTRAINT=«R_1»,     FK_COLUMNS=«kod_modeli»     «kod_firm»
«kod_tovaru» «kod_obladnya» ·/
     SELECT count(·) INTO NUMROWS
     FROM Firm
     WHERE
     /·%JoinFKPK (Firm,:%Old,» = «,» AND») ·/
     Firm. kod_modeli =:old. kod_modely AND
     Firm. kod_firm =:old. kod_firm AND
     Firm. kod_tovaru =:old. kod_komplektuechgo AND
     Firm. kod_obladnya =:old. kod_obladnya;
     IF (NUMROWS > 0)
     THEN
     raise_application_error(
     -20001,
     'Cannot delete Models because Firm exists.'
     );
     END IF;
     - ERwin Builtin Trigger
     END;
     /CREATE TRIGGER tU_Models AFTER UPDATE ON Models for each row
     - ERwin Builtin Trigger
```

```
     – UPDATE trigger on Models
     DECLARE NUMROWS INTEGER;
     BEGIN
     /· ERwin Builtin Trigger ·/
     /· Models Firm on parent update restrict ·/
     /·   ERWIN_RELATION:   CHECKSUM=«0002e12e»,   PARENT_OWNER=««,
PARENT_TABLE=«Models»
     CHILD_OWNER=««, CHILD_TABLE=«Firm»
     P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
     FK_CONSTRAINT=«R_1»,     FK_COLUMNS=«kod_modeli»     «kod_firm»
«kod_tovaru» «kod_obladnya» ·/
     IF /·%JoinPKPK(:%Old,:%New,» <> «,» OR «) ·/
     :old. kod_modely <>:new. kod_modely OR
     :old. kod_firm <>:new. kod_firm OR
     :old. kod_komplektuechgo <>:new. kod_komplektuechgo OR
     :old. kod_obladnya <>:new. kod_obladnya
     THEN
     SELECT count(·) INTO NUMROWS
     FROM Firm
     WHERE/·%JoinFKPK (Firm,:%Old,» = «,» AND») ·/
     Firm. kod_modeli =:old. kod_modely AND
     Firm. kod_firm =:old. kod_firm AND
     Firm. kod_tovaru =:old. kod_komplektuechgo AND
     Firm. kod_obladnya =:old. kod_obladnya;
     IF (NUMROWS > 0)
     THEN
     raise_application_error(
     -20005,
     'Cannot update Models because Firm exists.'
     );
     END IF;
     END IF;
     /· ERwin Builtin Trigger ·/
     /· Tovari Models on child update restrict ·/
     /·   ERWIN_RELATION:   CHECKSUM=«00000000»,   PARENT_OWNER=««,
PARENT_TABLE=«Tovari»
     CHILD_OWNER=««, CHILD_TABLE=«Models»
     P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
     FK_CONSTRAINT=«R_2»,     FK_COLUMNS=«kod_firm»     «kod_modely»
«kod_komplektuechgo» «kod_obladnya» ·/
     SELECT count(·) INTO NUMROWS
     FROM Tovari
     WHERE
     /·%JoinFKPK(:%New, Tovari,» = «,» AND») ·/
     :new. kod_firm = Tovari. kod_firm AND
     :new. kod_modely = Tovari. kod_models AND
     :new. kod_komplektuechgo = Tovari. kod_tovaru AND
     :new. kod_obladnya = Tovari. kod_obladnanya;
     IF (
     /·%NotnullFK(:%New,» IS NOT NULL AND») ·/
     NUMROWS = 0
     )
```

```
     THEN
     raise_application_error(
     -20007,
     'Cannot update Models because Tovari does not exist.'
     );
     END IF;
     - ERwin Builtin Trigger
     END;
     /CREATE TRIGGER tD_News AFTER DELETE ON News for each row
     - ERwin Builtin Trigger
     - DELETE trigger on News
     DECLARE NUMROWS INTEGER;
     BEGIN
     /· ERwin Builtin Trigger ·/
     /· News Shogi_tovary on parent delete restrict ·/
     /·   ERWIN_RELATION:   CHECKSUM=«0000f5bf»,   PARENT_OWNER=««,
PARENT_TABLE=«News»
     CHILD_OWNER=««, CHILD_TABLE=«Shogi_tovary»
     P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
     FK_CONSTRAINT=«R_6», FK_COLUMNS=«Name_Firm» «Name_models» ·/
     SELECT count(·) INTO NUMROWS
     FROM Shogi_tovary
     WHERE
     /·%JoinFKPK (Shogi_tovary,:%Old,» = «,» AND») ·/
     Shogi_tovary. Name_Firm =:old. Name_firm AND
     Shogi_tovary. Name_models =:old. Name_models;
     IF (NUMROWS > 0)
     THEN
     raise_application_error(
     -20001,
     'Cannot delete News because Shogi_tovary exists.'
     );
     END IF;
     - ERwin Builtin Trigger
     END;
     /CREATE TRIGGER tU_News AFTER UPDATE ON News for each row
     - ERwin Builtin Trigger
     - UPDATE trigger on News
     DECLARE NUMROWS INTEGER;
     BEGIN
     /· ERwin Builtin Trigger ·/
     /· News Shogi_tovary on parent update restrict ·/
     /·   ERWIN_RELATION:   CHECKSUM=«00012959»,   PARENT_OWNER=««,
PARENT_TABLE=«News»
     CHILD_OWNER=««, CHILD_TABLE=«Shogi_tovary»
     P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
     FK_CONSTRAINT=«R_6», FK_COLUMNS=«Name_Firm» «Name_models» ·/
     IF
     /·%JoinPKPK(:%Old,:%New,» <> «,» OR «) ·/
     :old. Name_firm <>:new. Name_firm OR
     :old. Name_models <>:new. Name_models
     THEN
```

```
    SELECT count(·) INTO NUMROWS
    FROM Shogi_tovary
    WHERE
    /·%JoinFKPK (Shogi_tovary,:%Old,» = «,» AND») ·/
    Shogi_tovary. Name_Firm =:old. Name_firm AND
    Shogi_tovary. Name_models =:old. Name_models;
    IF (NUMROWS > 0)
    THEN
    raise_application_error(
    -20005,
    'Cannot update News because Shogi_tovary exists.'
    );
    END IF;
    END IF;
    - ERwin Builtin Trigger
    END;
    /
    CREATE TRIGGER tD_Obladnanya AFTER DELETE ON Obladnanya for
each row
    - ERwin Builtin Trigger
    - DELETE trigger on Obladnanya
    DECLARE NUMROWS INTEGER;
    BEGIN
    /· ERwin Builtin Trigger ·/
    /· Obladnanya Tovari on parent delete restrict ·/
    /·   ERWIN_RELATION:   CHECKSUM=«0000f377»,   PARENT_OWNER=««,
PARENT_TABLE=«Obladnanya»
    CHILD_OWNER=««, CHILD_TABLE=«Tovari»
    P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
    FK_CONSTRAINT=«R_3», FK_COLUMNS=«kod_obladnanya» ·/
    SELECT count(·) INTO NUMROWS
    FROM Tovari
    WHERE
    /·%JoinFKPK (Tovari,:%Old,» = «,» AND») ·/
    Tovari. kod_obladnanya =:old. kod_obladnanya;
    IF (NUMROWS > 0)
    THEN
    raise_application_error(
    -20001,
    'Cannot delete Obladnanya because Tovari exists.'
    );
    END IF;
    - ERwin Builtin Trigger
    END;
    /
    CREATE TRIGGER tU_Obladnanya AFTER UPDATE ON Obladnanya for
each row
    - ERwin Builtin Trigger
    - UPDATE trigger on Obladnanya
    DECLARE NUMROWS INTEGER;
    BEGIN
    /· ERwin Builtin Trigger ·/
```

```
     /· Obladnanya Tovari on parent update restrict ·/
     /·   ERWIN_RELATION:   CHECKSUM=«00011a77»,   PARENT_OWNER=««,
PARENT_TABLE=«Obladnanya»
     CHILD_OWNER=««, CHILD_TABLE=«Tovari»
     P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
     FK_CONSTRAINT=«R_3», FK_COLUMNS=«kod_obladnanya» ·/
     IF
     /·%JoinPKPK(:%Old,:%New,» <> «,» OR «) ·/
     :old. kod_obladnanya <>:new. kod_obladnanya
     THEN
     SELECT count(·) INTO NUMROWS
     FROM Tovari
     WHERE
     /·%JoinFKPK (Tovari,:%Old,» = «,» AND») ·/
     Tovari. kod_obladnanya =:old. kod_obladnanya;
     IF (NUMROWS > 0)
     THEN
     raise_application_error(
     -20005,
     'Cannot update Obladnanya because Tovari exists.'
     );
     END IF;
     END IF;
     - ERwin Builtin Trigger
     END;
     /
     CREATE TRIGGER tI_Shogi_tovary BEFORE INSERT ON Shogi_tovary
for each row
     - ERwin Builtin Trigger
     - INSERT trigger on Shogi_tovary
     DECLARE NUMROWS INTEGER;
     BEGIN
     /· ERwin Builtin Trigger ·/
     /· News Shogi_tovary on child insert restrict ·/
     /·   ERWIN_RELATION:   CHECKSUM=«0000fd9a»,   PARENT_OWNER=««,
PARENT_TABLE=«News»
     CHILD_OWNER=««, CHILD_TABLE=«Shogi_tovary»
     P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
     FK_CONSTRAINT=«R_6», FK_COLUMNS=«Name_Firm» «Name_models» ·/
     SELECT count(·) INTO NUMROWS
     FROM News
     WHERE
     /·%JoinFKPK(:%New, News,» = «,» AND») ·/
     :new. Name_Firm = News. Name_firm AND
     :new. Name_models = News. Name_models;
     IF (
     /·%NotnullFK(:%New,» IS NOT NULL AND») ·/
     NUMROWS = 0
     )
     THEN
     raise_application_error(
     -20002,
```

```
     'Cannot insert Shogi_tovary because News does not exist.'
     );
     END IF;
     - ERwin Builtin Trigger
     END;
     /
     CREATE TRIGGER tD_Shogi_tovary AFTER DELETE ON Shogi_tovary for
each row
     - ERwin Builtin Trigger
     - DELETE trigger on Shogi_tovary
     DECLARE NUMROWS INTEGER;
     BEGIN
     /· ERwin Builtin Trigger ·/
     /· Shogi_tovary Firm on parent delete restrict ·/
     /·   ERWIN_RELATION:   CHECKSUM=«0001114b»,   PARENT_OWNER=««,
PARENT_TABLE=«Shogi_tovary»
     CHILD_OWNER=««, CHILD_TABLE=«Firm»
     P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
     FK_CONSTRAINT=«R_7»,     FK_COLUMNS=«name_firm»     «kod_firm»
«kod_tovary» «name_models» ·/
     SELECT count(·) INTO NUMROWS
     FROM Firm
     WHERE
     /·%JoinFKPK (Firm,:%Old,» = «,» AND») ·/
     Firm. name_firm =:old. Name_Firm AND
     Firm. kod_firm =:old. kod_firm AND
     Firm. kod_tovary =:old. kod_tovary AND
     Firm. name_models =:old. Name_models;
     IF (NUMROWS > 0)
     THEN
     raise_application_error(
     -20001,
     'Cannot delete Shogi_tovary because Firm exists.'
     );
     END IF;
     - ERwin Builtin Trigger
     END;
     / CREATE TRIGGER tU_Shogi_tovary AFTER UPDATE ON Shogi_tovary
for each row
     - ERwin Builtin Trigger
     - UPDATE trigger on Shogi_tovary
     DECLARE NUMROWS INTEGER;
     BEGIN
     /· ERwin Builtin Trigger ·/
     /· Shogi_tovary Firm on parent update restrict ·/
     /·   ERWIN_RELATION:   CHECKSUM=«00028ee1»,   PARENT_OWNER=««,
PARENT_TABLE=«Shogi_tovary»
     CHILD_OWNER=««, CHILD_TABLE=«Firm»
     P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
     FK_CONSTRAINT=«R_7»,     FK_COLUMNS=«name_firm»     «kod_firm»
«kod_tovary» «name_models» ·/
     IF /·%JoinPKPK(:%Old,:%New,» <> «,» OR «) ·/
```

```
      :old. Name_Firm <>:new. Name_Firm OR
      :old. kod_firm <>:new. kod_firm OR
      :old. kod_tovary <>:new. kod_tovary OR
      :old. Name_models <>:new. Name_models
      THEN
      SELECT count(·) INTO NUMROWS
      FROM Firm
      WHERE
      /·%JoinFKPK (Firm,:%Old,» = «,» AND») ·/
      Firm. name_firm =:old. Name_Firm AND
      Firm. kod_firm =:old. kod_firm AND
      Firm. kod_tovary =:old. kod_tovary AND
      Firm. name_models =:old. Name_models;
      IF (NUMROWS > 0)
      THEN
      raise_application_error(
      -20005,
      'Cannot update Shogi_tovary because Firm exists.'
      );
      END IF;
      END IF;
      /· ERwin Builtin Trigger ·/
      /· News Shogi_tovary on child update restrict ·/
      /·   ERWIN_RELATION:   CHECKSUM=«00000000»,   PARENT_OWNER=««,
PARENT_TABLE=«News»
      CHILD_OWNER=««, CHILD_TABLE=«Shogi_tovary»
      P2C_VERB_PHRASE=««, C2P_VERB_PHRASE=««,
      FK_CONSTRAINT=«R_6», FK_COLUMNS=«Name_Firm» «Name_models» ·/
      SELECT count(·) INTO NUMROWS
      FROM News
      WHERE
      /·%JoinFKPK(:%New, News,» = «,» AND») ·/
      :new. Name_Firm = News. Name_firm AND
      :new. Name_models = News. Name_models;
      IF (
      /·%NotnullFK(:%New,» IS NOT NULL AND») ·/
      NUMROWS = 0
      )
      THEN
      raise_application_error(
      -20007,
      'Cannot update Shogi_tovary because News does not exist.'
      );
      END IF;
      - ERwin Builtin Trigger
      END;/
```