Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____комп'ютерних наук_____
(повна назва)

Кафедра _____Програмної інженерії_____
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

_____другий (магістерський)_____
(рівень вищої освіти)

Дослідження методів аналізу продуктивності СКБД

(тема)

Виконав: студент 2 курсу, групи ІПЗм-18-2__
спеціальності _121 – Інженерія програмного забезпечення_____
(код і повна назва спеціальності)
освітньо-наукової програми _Інженерія програмного_забезпечення_ _____
(повна назва освітньої програми)
_____Чачанідзе С.С._____
(прізвище, ініціали)

Керівник _____проф. Лєсна Н.С._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ З.В.Дудар

2020 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук_____

Кафедра програмної інженерії_____

Рівень вищої освіти _____другий (магістерський)_____

Спеціальність _121 – Інженерія програмного забезпечення_____
(код і повна назва)

Освітньо-наукова програма Інженерія програмного забезпечення _____
(повна назва)

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

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

Студентові _Чачанідзе Сергію Сергійовичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи _Дослідження методів аналізу продуктивності СКБД_____

затверджена наказом по університету від «__»_____2020 р № ____

2. Термін подання студентом роботи до екзаменаційної комісії «11» травня 2020 р.

3. Вихідні дані до роботи *Алгоритми обробки великих обсягів даних, алгоритми зберігання даних, методи керування базами даних. Використовувати ОС Windows, середовище об'єктно-орієнтованого проектування.*_____
_____

4. Перелік питань, що потрібно опрацювати в роботі _*мета роботи, аналіз проблемної галузі і постановка задачі, методи пошуку корисних даних, опис об'єктних моделей, використовувані. методи та алгоритми, архітектура програмної системи, опис розробленої програмної системи. , результати тестування програмної системи*_____
_____

5. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	проф. Лєсна Н.С.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1.	Аналіз предметної галузі	25 березня 2020 р.	
2.	Огляд існуючих методів	31 березня 2020 р.	
3.	Методи швидкого детектування відрізків	15 квітня 2020 р.	
4.	Підготовка пояснювальної записки	20 квітня 2020 р.	
5.	Спецчастина	28 квітня 2020 р.	
6.	Підготовка презентації та доповіді	03 травня 2020 р.	
7.	Попередній захист	05 травня 2020 р.	
8.	Нормоконтроль, рецензування	07 травня 2020 р.	
9.	Занесення диплома в електронний архів	08 травня 2020 р.	
10.	Допуск до захисту в зав. кафедри	10 травня 2020 р.	

Дата видачі завдання _ «____»_ _____2020 р.

Студент _____
(підпис)

Керівник роботи _____ __проф. Лєсна Н.С._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної роботи: 79 с., 8 табл., 33 рис., 3 дод., 27 джерел.

ПРОДУКТИВНІСТЬ СКБД, ПОРІВНЯННЯ КЛАСТЕРНИХ ІНДЕКСІВ НА ПІДСТАВІ РІЗНИХ СТРУКРУР, ДЕРЕВА, ДЕРЕВА ВАН ЕМДЕ БОАСА

Об'єктом дослідження є процес підвищення продуктивності роботи СКБД з використанням кластерних індексів.

Метою роботи є збільшення продуктивності кластерних індексів у СКБД.

Метод дослідження – математичний апарат теорії автоматів, нечітких множин, методи структурної та прикладної лінгвістики, методи представлення знань в штучному інтелекті.

В результаті проведена ідентифікація й вибір рівнів факторів, обгрунтована загальна схема організації експериментальних досліджень, спроектовано прототип ПЗ.

DBMS PRODUCTIVITY, COMPARISON OF CLUSTER INDICES BASED ON VARIOUS STRUCTURES, TREES VAN EMDE BOAS

The object of study is the process of semantic analysis of the text of technical documentation during certification of software systems.

The purpose is to synthesize an ontological model of text to automate the semantic analysis of technical documentation,

As a result, the basic methods of ontological model construction were used, which were used in the construction of this model. The ontological model of the text of the technical documentation and its program model is designed.

ЗМІСТ

ВСТУП

Бази даних завжди були найважливішою темою при вивченні інформаційних систем. Однак в останні роки сплеск популярності Інтернету й бурхливий розвиток нових технологій для Інтернету зробили знання технології баз даних для багатьох одним з актуальних шляхів кар'єри IT-фахівця. Технології баз даних повели ІнтернетрДодатка далеко від простих брошурних публікацій, які характеризували ранні додатки. У той же час ІнтернетеТехнологія забезпечує користувачам стандартизовані й доступні засоби публікації вмісту баз даних. Правда, жодна із цих нових розробок не скасовує необхідності в класичних додатках баз даних, які з'явилися ще до розвитку Інтернету для потреб бізнесу. Це тільки розширює важливість знання баз даних.

При проектуванні бази даних необхідно добитися, щоб усі важливі операції виконувалися правильно й швидко. Деякі проблеми продуктивності можна розв'язати після того, як база даних буде поміщена у виробниче середовище. Проте, причиною багатьох проблем можуть стати помилки при проектуванні бази даних, які можна виправити тільки шляхом зміни її структури.

При проектуванні й реалізації бази даних слід виділити більші таблиці й найбільш складні процеси. При їхній розробці необхідно приділити особливої увагу продуктивності. Крім того, слід визначити, який вплив на продуктивність буде виявляти кількість, що збільшується, користувачів, що звертаються до таблиць.

Керування більшими й надвеликими базами даних, проектування й розробка додатків для них має свої особливості. Тому, щоб не допустити погіршення характеристик як окремих додатків, так і всієї системи в цілому, потрібне використання спеціальних методів, що підвищують швидкість доступу до даних.

Як правило, використовується комплексний підхід. Під цим розуміється оптимізація всіх ланок системи – серверної, клієнтської й мережної частини. Існує

цілий ряд способів настроювання продуктивності: настроювання робочих станцій клієнтів, мережного транспорту, оптимізація клієнтських додатків, оптимізація серверного PL/ SqlкКоду й Sql-запитів.

Таблиці в базі даних можуть мати велика кількість рядків, які зберігаються в довільному порядку, і їх прямій пошук за заданим критерієм шляхом послідовного перегляду таблиці рядок за рядком займав би багато часу. Для підвищення продуктивності пошуку даних використовуються такі об'єкти бази даних, як індекси.

Індексування – один з найбільш вірних способів нарощування продуктивності бази даних. Крім того, він входить до числа основних механізмів бази даних, якому звичайно приділяється незаслужено мало уваги. Як правило, рядка бази даних зберігаються в тому порядку, у якому створюються. Для витягу із запису бази даних деякої довільної величини потрібне послідовне сканування відповідних рядків бази даних. Індекс створює окрема безліч рядків, упорядкованих відповідно до обраного індексу й утримуючих покажчики на вихідні рядки. Індексована база даних проглядається значно швидше, чим неіндексовані таблиці. Однак індексування "з'їдає" додатковий дисковий простір. Крім того, на модифікацію індексованої таблиці потрібно більше часу, оскільки всі застосовувані індекси теж доводиться коректувати.

У магістерській роботі досліджується ефективність застосування небінарних дерев ван Едме Боаса для представлення кластерного індексу в порівнянні із традиційними методами з погляду продуктивності.

Об'єктом дослідження є процес підвищення продуктивності роботи СКБД із використанням кластерних індексів.

Предметом дослідження є дерева ван Едме Боаса в порівнянні з існуючими структурами, використовувані в кластерних індексах.

Метою роботи є збільшення продуктивності кластерних індексів у СКБД.

Це передбачає рішення наступних завдань:

- аналіз структури різних кластерних індексів у СКБД і їх вибір для дослідження й порівняння;

- описати застосування дерева ван Едме Боаса до завдання побудови кластерних індексів;
- визначення шляхів збільшення продуктивності кластерних індексів у СКБД;
- визначення факторів, що впливають на продуктивність СКБД;
- вибір методу проведення експерименту;
- побудова схеми, плану експерименту, визначення набору досвідів і кількості паралельних досвідів;
- провести алгоритмізацію застосування дерева ван Едме Боаса для побудови кластерних індексів;
- розробити ПЗ, що дозволяє провести порівняльний аналіз продуктивності кластерних індексів при використанні дерева ван Едме Боаса;
- проведення програмного експерименту згідно зі складеним планом за допомогою допоміжного ПЗ;
- визначення відгуків експерименту по аналізі продуктивності БД для збору статистичної інформації;
- статистичний аналіз отриманих результатів;
- оцінити ефективність застосування досліджуваного дерева з погляду продуктивності;
- вказівка рекомендацій після аналізу отриманих результатів.

Для проведення експерименту по підвищенню продуктивності СКБД із використанням кластерних індексів будуть використані методи планування експерименту, статистичний аналіз.

В роботі зроблена заміна структури кластерного індексу, використовуючи дерева ван Едме Боаса. Результати проведеного експерименту можуть бути використані при рішенні проблеми збільшення продуктивності кластерних індексів у СКБД, використовуючи дерева ван Едме Боаса.

1 АНАЛІЗ МЕТОДІВ ЗБІЛЬШЕННЯ ПРОДУКТИВНОСТІ КЛАСТЕРНИХ ІНДЕКСІВ

1.1 Загальні теоретичні відомості

Одна з найбільш важливих завдань при роботі з базою даних – це побудова оптимального індексу, що дозволяє підвищити продуктивність системи. Більшість основних баз даних надають інструменти для перегляду плану виконання запиту й допомагають набудовувати й оптимізувати індекси [1].

Індекси – це головний інструмент підвищення продуктивності будь-якої бази даних, але важливо створити правильні індекси, інакше, замість прискорення ваших операцій, неправильний індекс може суттєво сповільнити виконання Ваших запитів[2].

У більшості випадків, правильність індексу визначають дві характеристики: архітектура бази даних і бізнес правила або логіка використання вашої бази даних.

Можна скористатися наступними правилами для побудови правильних індексів:

- практично всі таблиці повинні мати первинний ключ, який будується на індексі, особливо якщо цей ключ часто використовується;
- маленькі таблиці (менше 100 рядків) не мають потреби в індексах. Процес індексації маленької таблиці досить швидкий, так що Ви одержуєте досить високу продуктивність не зберігаючи такий індекс;
- варто індексувати поля, які часто використовується в запитах, і не варто індексувати стовпці, які використовуються рідко;
- будь-яке поле, використовуване в функції, що агрегує (сума, агрегат і т.ін.), яка містить пропозиції GROUP BY або ORDER BY і використовується в JOIN, повинне розглядатися як кандидат на індекс. Справа в тому, що движок бази даних для цих операцій використовує індекси [3].

Гарна практика не індексувати поля, у яких дані постійно змінюються. Індекс вимагає додаткового дискового простору, буде сильно фрагментований, а операції вставки, зміни й переіндексації будуть вимагати більших витрат системних ресурсів, чого ми прагнемо уникнути.

Також, уникніть індексування довгих полів даних, але індексуйте будь-який стовпчик, який має велику кількість унікальних значень і часто використовується в запитах [3]–[4].

Індекси погіршують продуктивність системи під час змін записи. У будь-який час при виконанні запиту на зміну даних у таблиці індекс повинен також змінюватися. Для вибору оптимальної кількості індексів необхідне тестування бази даних і спостереження за її продуктивністю. Статичні системи, де бази даних використовуються в основному для витягу даних, наприклад, для побудови звітів, дозволяють містити більша кількість індексів для підтримки запитів тільки на читання. Бази даних з більшою кількістю транзакцій для зміни даних будуть потребувати невеликої кількості індексів для забезпечення більш високої пропускної здатності.

Індекси займають додаткове місце на диску й в оперативній пам'яті. Точний розмір буде залежати від кількості записів у таблиці, також як і від кількості й розміру стовпчиків в індексі. У більшості випадків це не є основною проблемою, тому що дисковим простором зараз легко пожертвувати для кращої продуктивності [5].

Індекс – це спосіб бази даних попередньо сортувати дані в різноманітних порядках, реалізованих одночасно. Індекс формується зі значень одного або декількох стовпців таблиці й покажчиків на відповідні рядки таблиці й, таким чином, дозволяє шукати рядка, що задовольняють критерію пошуку [6]. Прискорення роботи з використанням індексів досягається в першу чергу за рахунок того, що індекс має структуру, оптимізовану під пошук – наприклад, збалансованого дерева.

Для оптимальної продуктивності запитів індекси звичайно створюються на тих стовпцях таблиці, які часто використовуються в запитах. Для однієї таблиці

може бути створено кілька індексів. Однак збільшення числа індексів сповільнює операції додавання, відновлення, видалення рядків таблиці, оскільки при цьому доводиться обновляти самі індекси [7]. Крім того, індекси займають додатковий обсяг пам'яті, тому перед створенням індексу слід переконатися, що планований виграш у продуктивності запитів перевищить додаткову витрату ресурсів комп'ютера на супровід індексу.

Існує два типи індексів: кластерні й некластерні. При наявності кластерного індексу рядка таблиці впорядковані за значенням ключа цього індексу. Якщо в таблиці немає кластерного індексу, таблиця називається купою. Некластерний індекс, створений для такої таблиці, містить тільки покажчики на записі таблиці. Кластерний індекс може бути тільки одним для кожної таблиці, але кожна таблиця може мати декілька різних некластерних індексів, кожний з яких визначає свій власний порядок проходження записів. Послідовність, у якій стовпці представлені в складеному індексі, досить важлива. Справа в тому, що одержати набір даних по запиту, що зачіпає тільки перший із проіндексованих стовпців, можна [8]–[9]. Однак у більшості СКБД неможливо або неефективне одержання даних тільки по другому й далі проіндексованих стовпцях (без обмежень на перший стовпець).

Наприклад, телефонний довідник, що відсортований спочатку по місту, потім по прізвищу, і потім по імені. Якщо користувач знає місто, то можна легко знайти всі телефони цього міста. Однак у такому довіднику буде досить трудомістко знайти всі телефони, що записані на певне прізвище – для цього необхідно подивитися в секцію кожного міста й пошукати там потрібне прізвище. Деякі СКБД виконують цю роботу, інші ж просто не використовують такий індекс.

Кластерний індекс відрізняється від некластерного тим, що в листах цього індексу втримуються не посилання на записі в таблиці, а самі записи. Таким чином, при наявності кластерного індексу запису в таблиці вибудовуються в порядку ключів такого індексу (строго говорячи, це не зовсім так, але в першім наближенні вірно). По очевидних причинах кластерний індекс може бути тільки

один на таблицю. Ідентифікація конкретного запису в таблиці також перебуває в прямої залежності від наявності кластерного індексу. Якщо кластерного індексу ні, то запис перебуває по унікальному ідентифікатору запису – RID, що однозначно визначає її положення у файлі даних. Якщо ж кластерний індекс у таблиці присутня, то фізичне місце цього запису не постійно, а, отже, використовувати RID не дуже практично, оскільки його довелося б обновляти у всіх індексах при кожній зміні. Тому запис ідентифікується по ключу кластерного індексу [9]–[10].

Індекси можуть бути реалізовані різними структурами, головна властивість у них – упорядкованість. У цей час найбільш часте застосовуються B-дерева.

1.2 Огляд існуючих методів підвищення продуктивності СКБД із використанням кластерних індексів

Багато баз даних мають один спеціальний індекс до таблиці, де всі дані з рядка втримуватися в індексі. В SQL сервері такий індекс називається кластерним (кластеризованным). Кластерний індекс можна зрівняти з телефонним довідником, тому як кожний елемент індексу містить усю інформацію, яка вам потрібна й не містить посилань для одержання додаткових даних.

Є загальне правило – кожна нетривіальна таблиця повинна мати кластерный індекс. Якщо можливо створити тільки один індекс до таблиці, зробіть його кластерним. В SQL сервері при створенні первинного ключа буде автоматично створений кластерний індекс (якщо він ще не втримується), використовуючи стовпець із первинним ключем, як ключ для індексування. Кластерний індекс – найбільш ефективний (якщо він використовується, то покриває весь запит) і в багатьох СКБД такий індекс сприяє ефективному управлінню простором, запитуваним для зберігання таблиць, тому що а якщо ні, то (без побудови

кластерного індексу) рядка таблиць зберігаються в неупорядкованій структурі, яку називають купою [11].

При виборі стовпців для кластерного індексу треба бути обережним. Якщо ви змінили запис і поміняєте значення стовпця в кластерному індексі, база даних буде змушена перешикувати елементи індексу (щоб тримати їх у відсортованому порядку). Елементи індексу для кластерного індексу містити всі значення стовпців, таким чином, зміна значення стовпця порівнянне з виконанням інструкції Delete і наступної за нею інструкцією Insert, що очевидно викличе проблеми із продуктивністю, якщо робити це часто. Із цієї причини, кластерні індекси часто складаються зі стовпців первинного ключа й зовнішнього ключа. Значення ключів якщо міняються, то дуже рідко.

1.3 Використання B+ дерев для побудови індексів

Дерева являють собою структури даних, у яких реалізовані операції над динамічними множинами. З таких операцій можна виділити пошук елемента, пошук мінімального (максимального) елемента, вставка, видалення, перехід до батька, перехід до дитини. Таким чином, дерево може використовуватися і як звичайний словник, і як черга із пріоритетами [12].

Основні операції в деревах виконуються за час пропорційне його висоті. Збалансовані дерева мінімізують свою висоту (приміром, висота бінарного збалансованого дерева з n вузлами рівна log n).

При розгляді величезної бази даних, що представлена у вигляді одного зі збалансованих дерев, як «червоно-чорне дерево», «AVL-дерево», «Декартово дерево», то вочевідь, що не можливо зберігати все це дерево в оперативній пам'яті, тому у ній зберігається лише частина інформації, решта ж зберігається на сторонньому носії (на жорсткому диску, швидкість доступу до якого набагато повільніша). Такі дерева як червоно-чорне або Декартове будуть вимагати при

користуванні log n звертань до стороннього носія. При більших n це дуже багато. Саме цю проблему й покликані розв'язати B-дерева.

B-дерева також являють собою збалансовані дерева, тому час виконання стандартних операцій у них пропорційно висоті. Але, на відміну від інших дерев, вони створені спеціально для ефективної роботи з дисковою пам'яттю (у попередньому прикладі – стороннім носієм), а точніше – вони мінімізують обіги типу вводу-висновку.

При побудові B-дерева застосовується фактор t, який називається мінімальним ступенем. Кожний вузол, крім кореневого, повинен мати, як мінімум t – 1, і не більш 2t – 1 ключів. Позначається n[x] – кількість ключів у вузлі x.

Ключі у вузлі зберігаються в неубутному порядку. Якщо x не є листом, то він має n[x] + 1 дітей. Якщо занумерувати ключі у вузлі x, як k[i], а дітей c[i], то для будь-якого ключа в під-дереві з коренем c[i] (нехай k1), виконується наступна нерівність – k[i-1] <=k1<=k[i] (для c[0]: k[i-1] = -∞, а для c[n[x]]: k[i] = +∞). Таким чином, ключі вузла задають діапазон для ключів їх дітей (рис. 1.1).

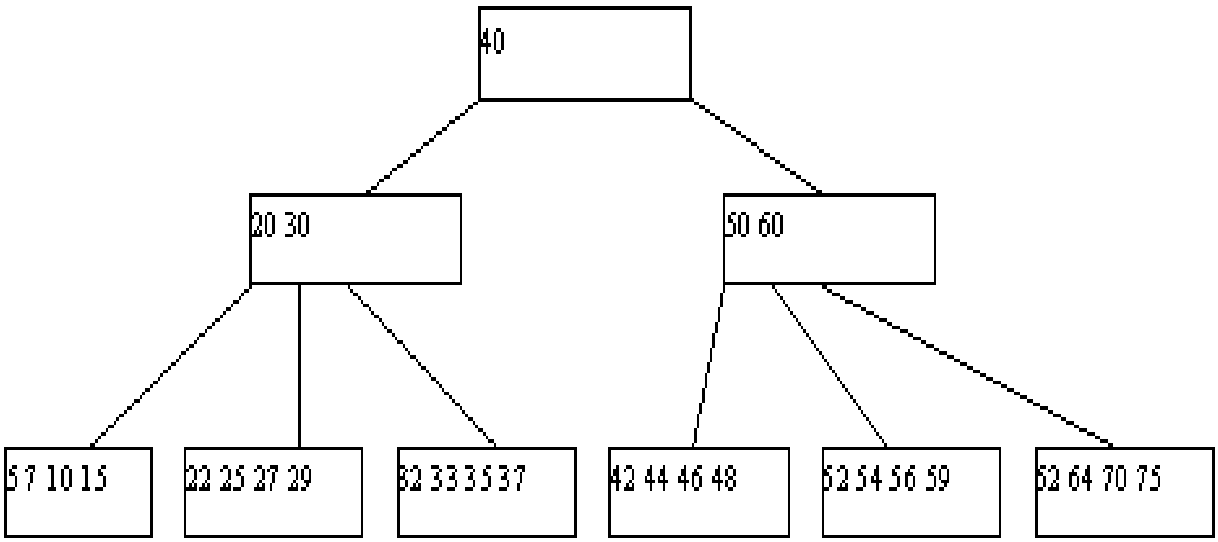

Рисунок 1.1 – Приклад будови B-дерева

Усі листи B-дерева повинні бути розташовані на одній висоті, яка і є висотою дерева. Висота B-дерева з n >= 1 вузлами й мінімальним ступенем t >= 2 не перевищує logt(n+1).

Сімейство B-Tree індексів – це найбільш часто використовуваний тип індексів, організованих як збалансоване дерево, упорядкованих ключів. Вони підтримуються практично всіма СКБД як реляційними, так нереляційними, і практично для всіх типів даних [13].

B+ дерево – структура даних, являє собою збалансоване дерево пошуку. Є модифікацією B-дерева, дійсні значення ключів якого втримуються тільки в листах, а у внутрішніх вузлах – ключілроздільники, що містять діапазон зміни ключів для піддерев.

При побудові B+ дерева, його часом доводиться перебудовувати. Це пов'язане з тим, що кількість ключів у кожному вузлі (крім кореня) повинне бути від k до 2k, де k – ступінь дерева. При спробі вставити у вузол (2k+1)-й ключ виникає необхідність розділити цей вузол. У якості ключа-роздільника сформованих галузей виступає (k+1)-й ключ, який міститься на сусідній ярус дерева. Особливим же случаємо є поділ кореня, тому що в цьому випадку збільшується число ярусів дерева. Особливістю поділу аркуша B+ дерева є те, що він ділиться на нерівні частини. При поділі внутрішнього вузла або кореня виникають вузли з рівним числом ключів k. Поділ листа може викликати «ланцюгову реакцію» розподілу вузлів, що закінчується в корені [14].

Властивості:

- у B+ дереві легко реалізується незалежність програми від структури інформаційного запису;
- пошук обов'язково закінчується в листі;
- видалення ключа має перевагу – видалення завжди походить із листа;
- інші операції виконуються аналогічно B-деревам;
- B+ дерева вимагають більше пам'яті для представлення, ніж B-дерева;
- B+ дерева мають можливість послідовного доступу до ключів.

B*-дерево – різновид B-дерева, у якій кожний вузол дерева заповнений не менш чому на 2/3 (на відміну від B-дерева, де цей показник становить 1/2). B+ дерево, що задовольняє таким вимогам називається B+*-деревом.

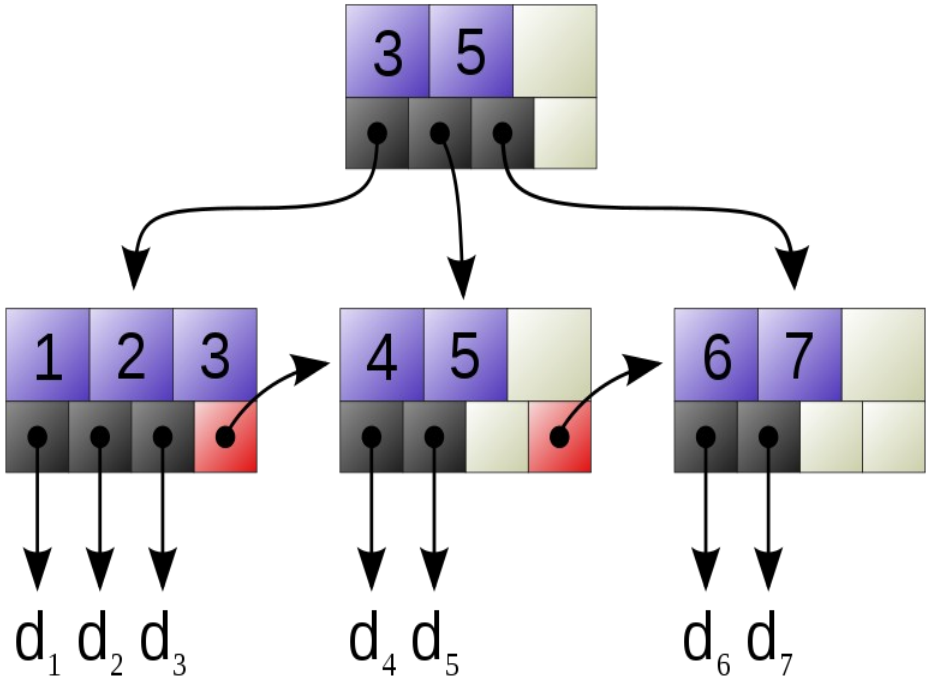

Рисунок 1.2 – Будова B+ дерева

B*-дерева запропонували Р. Бэйер і Е. Маккрейт, що вивчали проблему компактності B-дерев. B*-дерево відносно компактніше, тому що кожний вузол використовується повніше. В іншому ж цей вид дерев не відрізняється від простого B-дерева.

Для виконання вимоги (заповнювання вузла не менш 2/3), доводиться відмовлятися від простої процедури поділу переповненого вузла. Замість цього відбувається «переливання» у сусідній вузол. Якщо ж і сусідній вузол заповнений, то ключі приблизно нарівно розділяються на 3 нових вузла [15].

R-дерево – деревоподібна структура даних (дерево), запропонована в 1984 році А. Гуттманом. Воно подібно B-дереву, але використовується для організації доступу до просторових даних, тобто для індексації багатомірної інформації, такий, наприклад, як географічні дані із двовимірними координатами (широтою й довготою).

Ця структура даних розбиває простір на множину ієрархічно вкладених і прямокутників, що можливо перетинаються (для двовимірного простору). У випадку тривимірного або багатомірного простору це будуть прямокутні паралелепіпеди (кубоїди) або паралелотопи.

Алгоритми вставки й видалення використовують ці обмежуючі прямокутники для забезпечення того, щоб об'єкти, що «близько росташовані» були поміщені в одну листову вершину. Зокрема, новий об'єкт потрапить у ту листову вершину, для якої буде потрібно найменше розширення її обмежуючого прямокутника. Кожний елемент листової вершини зберігає два поля даних: спосіб ідентифікації даних, що описують об'єкт, (або самі ці дані) і обмежуючий прямокутник цього об'єкта [16].

Аналогічно, алгоритми пошуку (наприклад, перетинання, включення, околиці) використовують обмежуючі прямокутники для ухвалення рішення про необхідність пошуку в дочірній вершині. Таким чином, більшість вершин ніколи не зачіпаються в ході пошуку. Як і у випадку з B-деревами, ця властивість R-Дерев обумовлює їхню застосовність для баз даних, де вершини можуть вивантажуватися на диск у міру необхідності.

Для розщеплення переповнених вершин можуть застосовуватися різні алгоритми, що породжує розподіл R-дерев на підтипи: квадратичні й лінійні.

Споконвічно R-дерева не гарантували гарних характеристик для найгіршого випадку, хоча добре працювали на реальних даних. Однак, в 2004-м року був опублікований новий алгоритм, що визначає пріоритетні R-дерева. Стверджується, що цей алгоритм ефективний, як і найбільш ефективні сучасні методи, і в той же час є оптимальним для найгіршого випадку [17].

Також існують методи використання «хеш-таблиць» – це структура даних, що реалізує інтерфейс асоціативного масиву, а саме, вона дозволяє зберігати пари (ключ, значення) і виконувати три операції: операцію додавання нової пари, операцію пошуку й операцію видалення пари по ключу.

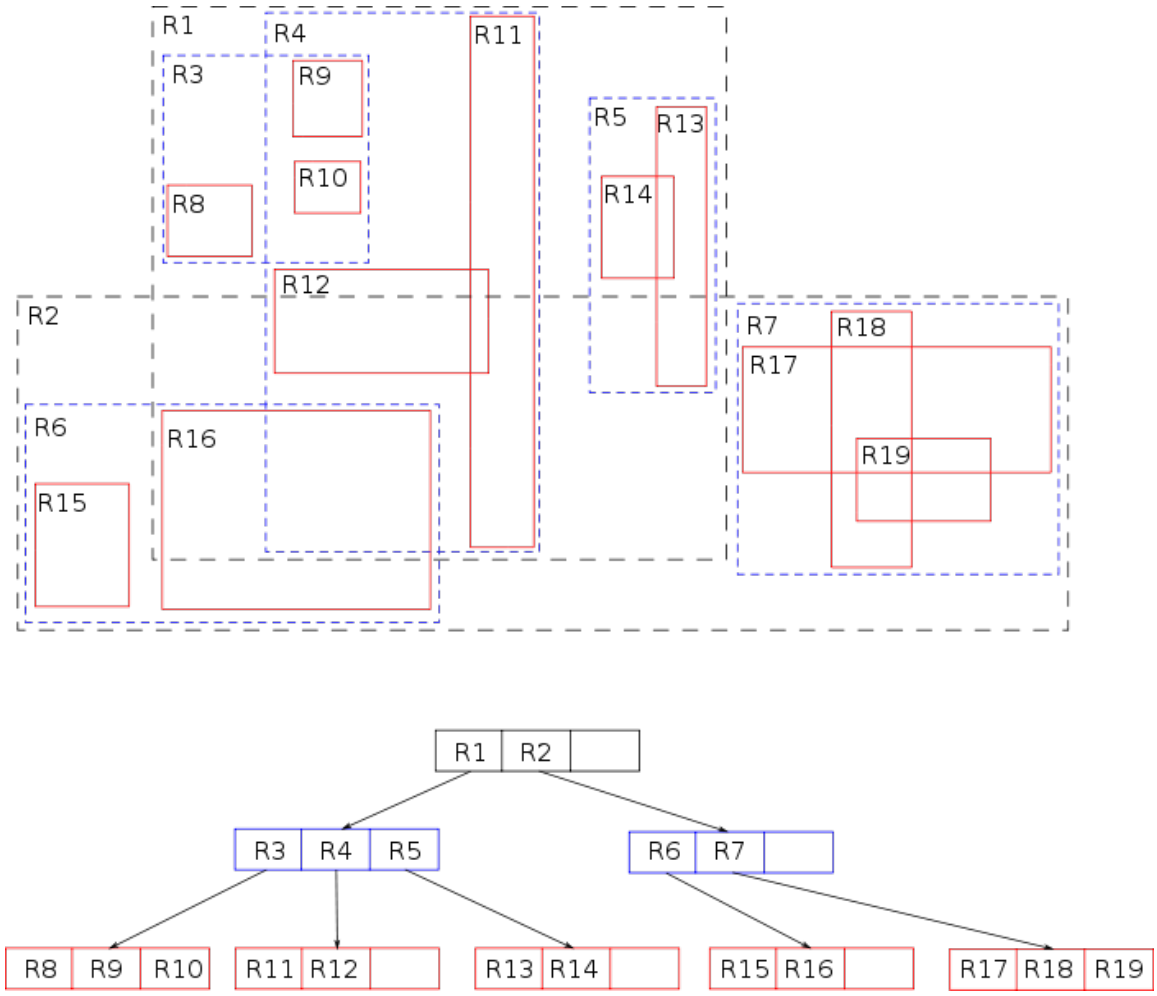

Рисунок 1.3 – Будова R-дерева

1.4 Використання хеш-таблиць для побудови індексів

Існують два основні варіанти хеш-таблиць: з ланцюжками й відкритою адресацією. Хеш-таблиця містить деякий масив, елементами якого є пари (хеш-таблиця з відкритою адресацією) або списки пар (хеш-таблиця з ланцюжками).

Виконання операції в хеш-таблиці починається з обчислення хеш-функції від ключа. Що виходить хеш-значення відіграє роль індексу в масиві. Потім виконувана операція (додавання, видалення або пошук) перенаправляється об'єкту, який зберігається у відповідній чарунці масиву.

Ситуація, коли для різних ключів виходить те саме хеш-значення, називається колізією. Такі події не так уже й рідкі – наприклад, при вставці в хеш-таблицю розміром 365 чарунок усього лише 23-х елементів імовірність колізії вже перевищить 50% (якщо кожний елемент може рівновероятно потрапити в будь-яку чарунку). Тому механізм вирішення колізій – важлива складова будь-який хеш-таблиці [18]– [19].

У деяких спеціальних випадках вдається уникнути колізій взагалі, наприклад, якщо всі ключі елементів відомі заздалегідь (або дуже рідко змінюються), то для них можна знайти деяку зроблену хеш-функцію, яка розподілить їх по чарунках хеш-таблиці без колізій. Хеш-таблиці, що використовують подібні хеш-функції, не мають потреби в механізмі вирішення колізій, і називаються хеш-таблицями із прямою адресацією.

Число збережених елементів, ділене на розмір масиву (число можливих значень хеш-функції), називається коефіцієнтом заповнення хеш-таблиці (load factor) і є важливим параметром, від якого залежить середній час виконання операцій.

Важлива властивість хеш-таблиць полягає в тому, що, при деяких розумних допущеннях, усі три операції (пошук, вставка, видалення елементів) у середньому виконуються за час . Але при цьому не гарантується, що час виконання окремої операції мало. Це пов'язане з тим, що при досягненні деякого значення коефіцієнта заповнення необхідно здійснювати перебудову індексу хеш-таблиці: збільшити значення розміру масиву й заново додати в порожню хеш-таблицю всі пари.

Хешування – перетворення по детермінованому алгоритму вхідного масиву даних довільної довжини у вихідний бітовий рядок фіксованої довжини. Такі перетворення також називаються хеш-функціями або функціями згортки, а їх результати називають хешем, хешокодом або зведенням повідомлення (англ. message digest). Якщо у двох рядків хеш-коди різні, рядки гарантовано різняться, якщо однакові – рядка, імовірно, збігаються.

Хешування застосовується для побудови асоціативних масивів, індексів, пошуку дублікатів у серіях наборів даних, побудови досить унікальних ідентифікаторів для наборів даних, контрольного підсумовування з метою виявлення випадкових або навмисних помилок при зберіганні або передачі, для зберігання паролів у системах захисту (у цьому випадку доступ до області пам'яті, де перебувають паролі, не дозволяє відновити сам пароль), при виробленні електронного підпису (на практиці часто підписується не саме повідомлення, а його хеш-образ) [20].

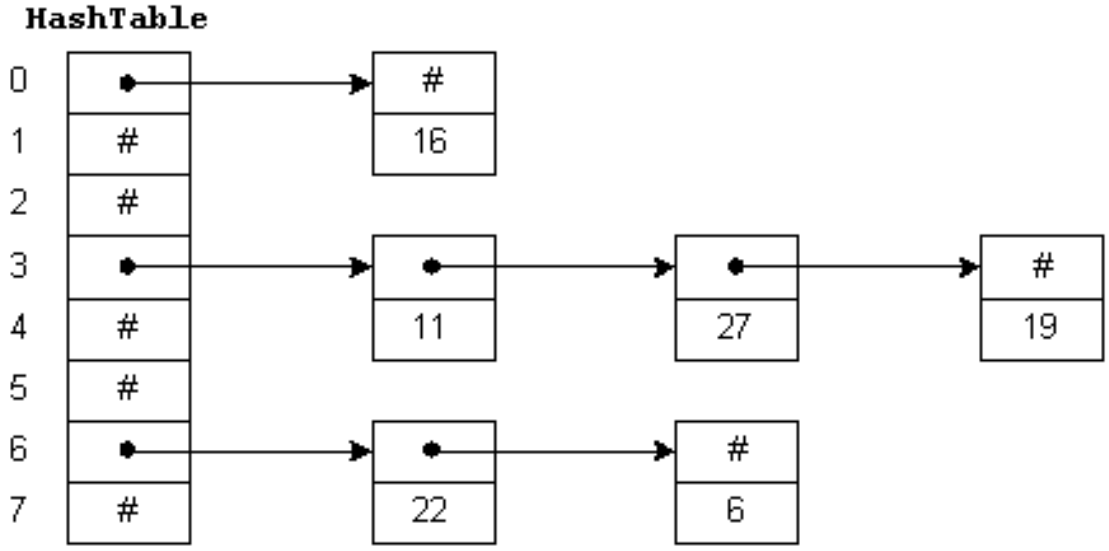

Рисунок 1.4 – Структура хеш-таблиці

У багатьох випадках наведені дерева мають незадовільну продуктивність при пошуку, вставці, зміні або видаленні даних. Заміна структури індексу може дати підвищення продуктивності роботи СКБД. Тому досить обґрунтованим і актуальним бачиться застосування дерева ван Едме Боаса для реалізації кластерних індексів у СКБД.

Дерево ван Едме Боаса (van Emde Boas tree) – асоціативний масив, який дозволяє зберігати цілі числа в діапазоні [0; U), де U = 2k, простіше говорячи, числа, що полягають не більш ніж з k біт.

Головна особливість цієї структури – виконання всіх операцій за час O(log(log(U))) незалежно від кількості елементів, що зберігаються в ній. Властиво, от невеликий список підтримуючих операцій [21]:

- Insert(x) – вставка числа в дерево;
- Remove(x) – Видалення числа;
- Getmin(), Getmax()– знаходження мінімуму й максимуму в дереві;
- Find(x) – пошук числа в дереві;
- Findnext(x) – пошук наступного числа після x, яке втримується в дереві;
- Findprevious(x) – пошук попереднього x числа.

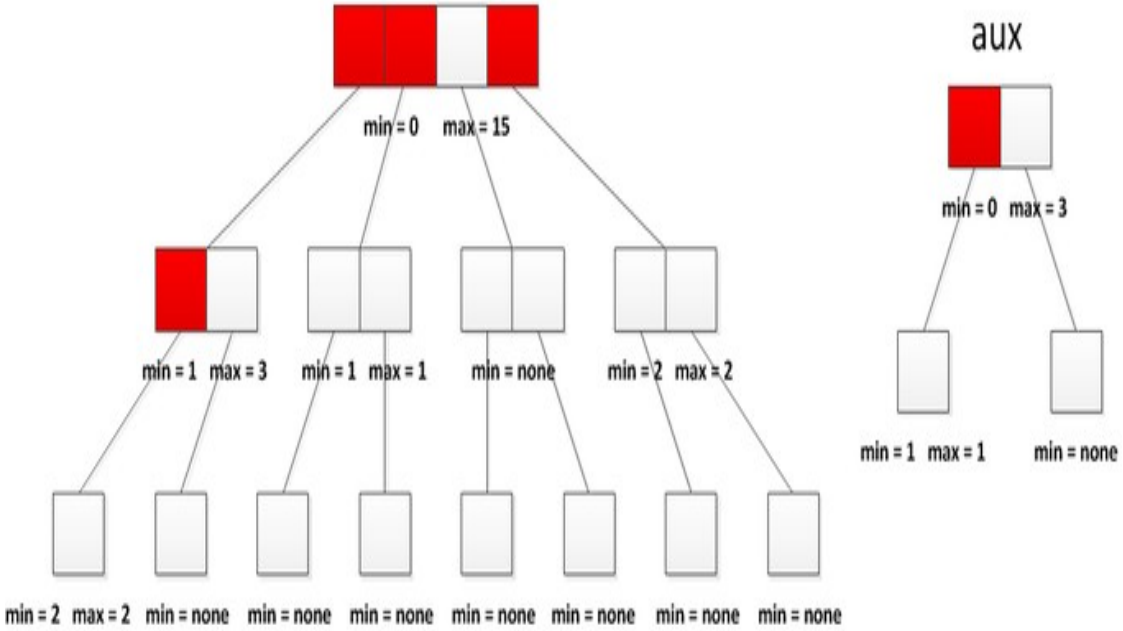

Рисунок 1.5 – Структура дерева ван Едме Боаса

1.5 Постановка завдань дослідження

Мета роботи – збільшення продуктивності кластерних індексів у СКБД.

Це передбачає рішення наступних завдань:

- аналіз структури різних кластерних індексів у СКБД і їх вибір для дослідження й порівняння;
- описати застосування дерева ван Едме Боаса до завдання побудови кластерних індексів;
- визначення шляхів збільшення продуктивності кластерних індексів у СКБД;

- визначення факторів, що впливають на продуктивність СКБД;
- вибір методу проведення експерименту;
- побудова схеми, плану експерименту, визначення набору досвідів і кількості паралельних досвідів;
- провести алгоритмізацію застосування дерева ван Едме Боаса для побудови кластерних індексів;
- розробити ПЗ, що дозволяє провести порівняльний аналіз продуктивності кластерних індексів при використанні дерева ван Едме Боаса;
- проведення експерименту згідно зі складеним планом за допомогою допоміжного ПЗ;
- визначення відгуків експерименту по аналізі продуктивності БД для збору статистичної інформації;
- статистичний аналіз отриманих результатів;
- оцінити ефективність застосування досліджуваного дерева з погляду продуктивності;
- опис рекомендацій після аналізу отриманих результатів;

2 АНАЛІЗ ПОКАЗНИКІВ ПРОДУКТИВНОСТІ СКБД ІЗ РІЗНИМИ СТРУКТУРАМИ КЛАСТЕРНИХ ІНДЕКСІВ

2.1 Аналіз методів програмних експериментів

Теорія планування експериментів охоплює практично всі дослідження, що зустрічаються на практиці варіанти, об'єктів. Надалі будуть розглянуті наступні типові завдання експериментального дослідження:

- пошук значень параметрів системи, що забезпечують досягнення оптимального значення показника якості досліджуваного об'єкта при відомих обмеженнях на значення цих параметрів. Перебір усіх припустимих комбінацій значень параметрів системи з метою пошуку оптимального варіанта нераціональний по витратах ресурсів. Для рішення зазначеного завдання теорія планування експериментів пропонує таку послідовність проведення досвідів, яка дозволяє застосувати градієнтні методи пошуку при апріорно невідомій функції, що зв'язує показник якості з параметрами системи;

- наближений аналітичний опис функціонального зв'язку показників якості з параметрами системи за результатами проведеного експерименту. Традиційні методики проведення експериментів через залежність компонентів відновлюваного аналітичного опису не дозволяють визначити роздільний вплив кожного фактора на результуючий показник, тобто ці методики забезпечують одержання аналітичних залежностей, придатних лише для рішення інтерполяційних завдань. На відміну від них ТПЕ дає можливість оцінити внесок кожного параметра в значення показника, тобто приблизно відновити закон функціонування об'єкта за експериментальним даними. Отриманий аналітичний опис об'єкта можна використовувати для попереднього дослідження варіантів побудови системи або в інтересах побудови моделі старшої системи, що включає даний об'єкт на правах елемента;

- оцінка диференціального впливу рівнів параметрів системи на показник якості. Таке завдання виникає у випадку, коли параметри системи є по своїй

природі якісними або коли кількісні параметри можуть ухвалювати невелике число різних значень.

Крім зазначених, існують і інших завдання, розв'язувані за допомогою ТПЕ, наприклад:

– випробування зразків техніки. Планування повинне дозволити оцінити ступінь відповідності показників якості зразків заданим вимогам при мінімальному обсязі випробувань;

– експерименти, що відсівають. Призначено виявити параметри, що незначно впливають на показник якості системи. Відповідні плани застосовують на початкових етапах дослідження, коли немає конкретних відомостей про вплив тих або інших параметрів. Відсівання несуттєвих факторів знижує трудомісткість рішення завдань оптимізації або наближеного аналітичного опису системи;

– адаптивне планування. Застосовується в умовах керування технологічним процесом, коли система керування увесь час повинна пристосовуватися до конкретних умов функціонування, а можливо, і пророкувати подальший розвиток процесу.

Рішення завдань із застосуванням ТПЕ передбачає використання апріорної інформації про досліджуваний процес для вибору загальної послідовності керування експериментами, яка уточнюється після чергового етапу проведення досліджень на основі знов отриманих відомостей. Тим самим досягається можливість раціонального керування експериментами при неповному первіснім знанні характеристик досліджуваного об'єкта. Доцільність застосування ТПЕ тем вище, чим складніше досліджувана система.

У ТПЕ досліджуваний об'єкт (реальний об'єкт, модель об'єкта) розглядається як «чорна скринька», що має входи v (керовані незалежні параметри) і виходи y [22]–[23].

Змінні v прийнято називати «факторами». Теорія ПЕ вивчає тільки активний тип експериментів, коли є можливість незалежно й цілеспрямовано міняти значення факторів v у всім необхідному діапазоні. Фактори в експерименті бувають якісними й кількісними. Якісні фактори можна квантифіцировати або

приписати їм числові позначення, тим самим перейти до кількісних значень. Надалі будемо вважати, що всі фактори є кількісними й представлені безперервними величинами (якщо інше не застережене особливо). Змінним *v* можна зіставити геометричне поняття «факторного простору» – простору, координатні осі якого відповідають значенням факторів. Сукупність конкретних значень усіх факторів утворює крапку в багатомірному факторному просторі. Прикладами факторів є: інтенсивність потоку запитів до бази даних, швидкість передачі даних по каналу, обсяг запам'ятовувального пристроїв. Крім того, на об'єкт впливають фактори, що обурюють, вони є випадковими й не піддаються керуванню.

Область планування завдається інтервалами можливої зміни факторів $vi_{,min} < vi < vi_{,max}$ для $i = 1, 2, \ldots, k$, де k – кількість факторів. У теорії ПЕ часто використовують нормалізацію факторів, тобто перетворення натуральних значень факторів у безрозмірні (кодовані) величини. Перехід до безрозмірних значень *xi* задається перетворенням

$$xi = (vi - vi_0)/\Delta\varpi\iota, \tag{2.1}$$

де *vi* – натуральне значення фактора,

vi_0 – натуральне значення основного рівня фактора, відповідне до нуля в безрозмірній шкалі,

Dvi – інтервал варіювання.

Сукупність основних рівнів усіх факторів являє собою крапку в просторі параметрів, називану центральною крапкою плану або центром експерименту. З геометричної точки зору нормалізація факторів рівноцінна лінійному перетворенню простору факторів, при якім проводяться дві операції: перенос початку координат у крапку, відповідну до значень основних рівнів факторів; стискання-розтягання простору в напрямку координатних осей.

Активний експеримент включає: систему впливів, при яких відтворюється функціонування об'єкта; реєстрацію відгуку об'єкта. План експерименту задає сукупність даних, що визначають кількість, умови й порядок реалізації досвідів. Досвід становить елементарну частину експерименту й передбачає

відтворення досліджуваного явища в конкретних умовах з наступною реєстрацією результату. В умовах випадковості в тих самих умовах проводяться паралельні (повторні) досвіди в інтересах одержання статистично стійких результатів. Досвід *и* припускає завдання конкретних значень факторам $\mathbf{v}_u = v_{1u}, v_{2u}, \ldots, vku$, а сукупність значень факторів у всіх N крапках плану експерименту утворює матрицю плану

$$\begin{array}{l} v_{11}, v_{21}, \ldots, vk_1 \\ v_{12}, v_{22}, \ldots, vk_2 \\ \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \\ v_{1N}, v_{2N}, \ldots, vkn \end{array} \qquad (2.2)$$

Рядки матриці відповідають досвідам, стовпці – факторам, елемент матриці *viz* задає значення *z-го* фактора в *і-м*у досвіді.

Вектор y називається відгуком. У ТПЕ зазвичай вивчається ситуація, у якій вектор відгуку y складається з одного елемента y. При наявності декількох складових вектора **у**, кожну з них можна досліджувати окремо. Залежність відгуку від факторів зветься функції відгуку, а геометрична вистава функції відгуку – поверхні відгуку. Функція відгуку розглядається як показник якості або ефективності об'єкта. Цей показник є функцією від параметрів – факторів. На практиці широке поширення одержали прості функції виду

$$\mathrm{M}\{y'\} = \mathrm{bf}(v),$$

де $\mathrm{b}=(b_0, b_1, \ldots, bh)$ – вектор невідомих параметрів моделі розмірності $h+1$,

$\mathrm{f}(v)=(f_0(v), f_1(v), \ldots, fh(v))$ – вектор заданих базисних функцій,

$\mathrm{M}\{y'\}$ – математичне очікування функції відгуку.

Таке представлення функції відгуку відповідає лінійної по параметрах моделі регресійного аналізу, тобто функція відгуку є лінійна комбінація базисних функцій від факторів.

Внаслідок впливу на результати експериментів випадкових впливів дійсні значення коефіцієнтів можна визначити тільки приблизно. Оцінку $\beta = (\beta_0, \beta_1, \ldots, \beta\eta)$ вектора невідомих параметрів b знаходять за результатами експериментів,

у ході яких одержують значення *yu* при заданих значеннях факторів **vu**. Ці оцінки звичайно розраховуються за допомогою методу найменших квадратів (МНК) на основі вибірок значень факторів і відгуків системи на впливи [24]. У якості оцінки β вектора b вибирається таке значення, яке мінімізує

$$\frac{1}{N}\sum_{u=1}^{N}(y'_u - y_u)^2,$$

де y'_u – обчислене на моделі значення функції відгуку в u-й точці факторного простору.

Дорівнюючи нулю частки похідні від даної квадратичної форми, узяті по змінним $b_0, b_1, \ldots, b_h$, можна одержати систему рівнянь виду

$$\frac{1}{N}\sum_{u=1}^{N}(y'_u - y_u)f(v_i) = 0,$$

де $i = 0, 1, 2, \ldots, h$.

Значення b знаходять шляхом рішення цієї системи рівнянь. Рішення системи можливо при лінійній незалежності базисних функцій.

Якщо не ухвалювати спеціальних заходів, то оцінки коефіцієнтів β стануть взаємозалежними, і отриманий вираз для функції відгуку можна розглядати тільки як інтерполяційну формулу, що утруднює її фізичну інтерпретацію й наступні розрахунки. Однак, формуючи спеціальним образом матрицю плану, можна одержати незалежні значення β. І ці величини будуть характеризувати внесок кожного фактора в значення функції відгуку.

Отже, завдання полягає у визначенні загальної форми запису функції відгуку y'. У більшості випадків вид цієї функції, одержуваний з теоретичних міркувань, є складним для практичного застосування, а при неповному знанні об'єкта взагалі невідомий. По даних причинах функцію доцільно представити в універсальному, зручному для практичного застосування виді, чому відповідає вистава у вигляді полінома. Тоді системою базисних функцій є сукупність статечних функцій із цілими ненегативними значеннями показників ступені. Поліноміальна форма вистави функції відгуку прийме вид

$$y' = \beta_0 + \beta_1 x_1 + \ldots + \beta_к \xi_к + \beta_{12} x_1 x_2 + \beta_{13} x_1 x_3 + \ldots \qquad (2.3)$$

$$+\beta_{к-1,k} x k_{-1} x k + +\beta_{11} x^{21} + \ldots + \beta_{кк} \xi^2_k + \ldots + \varepsilon,$$

де ε – випадкова величина, що характеризує помилку досвіду.

Така функція відгуку лінійна щодо невідомих коефіцієнтів і буде повністю визначена, якщо задані ступінь полінома й коефіцієнти. Ступінь полінома звичайно задається дослідником апріорно й уточнюється в ході дослідження. На практиці найбільше поширення одержали поліноми першого й другого порядку, відповідно лінійні й квадратичні моделі. Коефіцієнти полінома прийнято називати ефектами факторів.

Іноді функцію відгуку доцільно представити в іншому виді, наприклад, у вигляді статечної функції, тому що досягнення заданої точності вимагає застосування полінома високого порядку. Однак використання функцій, нелінійних щодо невідомих параметрів, ускладнює обчислення, утрудняє оцінку їх властивостей. У деяких випадках завдання можна спростити шляхом штучного перетворення нелінійної функції в лінійну. При цьому потрібне відповідне перетворення й результатів експериментів.

Застосування ТПЕ засновано на ряді допущень[25]–[26].

Функція відгуку містить у своєму складі невипадкову й випадкову складову. Багато показників якості автоматизованих систем обробки інформації носять випадковий характер. Це вимагає багаторазового повторення досвідів у тих самих умовах з метою одержання статистично стійких результатів, а одержувані оцінки показників повинні мати властивості заможності, ефективності, несмещенности й достатності. Оцінки типових показників формуються шляхом усереднення результатів спостережень. Тому при досить великій кількості спостережень можна вважати, що випадкова складова е розподілена за нормальним законом з нульовим математичним очікуванням, що дозволяє одержати незміщену оцінку математичного очікування функції відгуку в конкретній крапці плану. Будемо також уважати, що величина е має дисперсію, що не залежить від значень факторів. Інакше кажучи, результати, отримані

шляхом усереднення повторних досвідів у кожній крапці плану, являють собою незалежні, нормально розподілені випадкові величини.

2.2 Аналіз методів визначення основних факторів

Фактори $v_1, v_2, \ldots, v_k$ виміряються із дуже малою помилкою в порівнянні з помилкою у визначенні величини y (облік перешкод у завданні факторів приводить до важко розв'язних проблем в оцінці коефіцієнтів функції відгуку). Помилка у визначенні значення функції відгуку пояснюється не стільки погрішністю вимірів, скільки впливом на результат роботи системи неврахованих або випадкових факторів, наприклад відмінностями у формованій послідовності випадкових чисел при статистичному моделюванні.

Дисперсії середнього значення функції відгуку в різних крапках рівні один одному (вибіркові оцінки дисперсії однорідні). Це означає, що при багаторазових повторних спостереженнях над величиною yu при деякому наборі значень v_{1u}, v_{2u}, …, vku, одержувана оцінка дисперсії середнього значення не буде відрізнятися від оцінки дисперсії, отриманої при багаторазових спостереженнях для будь-якого іншого набору значень незалежних змінних v_{1s}, v_{2s}, …, v_{ks}.

Зазначені допущення дозволяють використовувати для розрахунків коефіцієнтів полінома МНК, який дає ефективні й незміщені оцінки коефіцієнтів і забезпечує простоту проведення самих розрахунків. Застосування МНК, загалом кажучи, не вимагає дотримання нормального розподілу результатів спостереження. Цей метод у кожному разі дає рішення, що мінімізує суму квадратів відхилень результатів спостереження від значень функції відгуку. Допущення про нормальний розподіл використовується при проведенні різного роду перевірок, наприклад, при перевірці адекватності функції відгуку й експериментальних даних. Природно, що точність оцінок коефіцієнтів функції

відгуку підвищується зі збільшенням числа досвідів, по яких обчислюються коефіцієнти.

У термінології планування експериментів вхідні змінні й структурні допущення, що становлять модель, називаються факторами, а вихідні показники роботи – відгуками. Рішення про те, які параметри й структурні допущення вважати фіксованими показниками моделі, а які експериментальними факторами, залежить від цілей дослідження.

У багатьох випадках фактори можуть носити не тільки кількісний, але і якісний характер. Тому значення факторів звичайно називають рівнями. Якщо при проведенні експерименту можна змінювати рівень фактора, то експеримент називається активним, а якщо ні, то пасивним.

План експерименту будується щодо одного (основного) вихідного скалярного параметра Y, який називається спостережуваної змінної. Якщо моделювання використовується як інструмент ухвалення рішення, то в ролі спостережуваній змінної виступає показник ефективності.

При цьому передбачається, що значення спостережуваної змінної, отримане в ході експерименту, складається з 2-х складових:

$$y = f(x) + e(x) \tag{2.4}$$

Перша складова f(x) – функція відгуку (невипадкова функція факторів), друга складова в(x) – помилка експерименту (випадкова величина).

В обох випадках x – крапка у факторному просторі (певна комбінація рівнів факторів). Очевидно, що *y* є випадкової змінної, тому що залежить від випадкової величини e(x).

Пошук факторів проводиться в так званім факторному просторі.

Факторний простір – це безліч зовнішніх і внутрішніх параметрів моделі, значення яких дослідник може контролювати в ході підготовки проведення модельного експерименту.

Існує два основні варіанти постановки завдання планування імітаційного експерименту:

− із усіх припустимих потрібно вибрати такий план, який дозволив би одержати найбільш достовірне значення функції відгуку f(x) при фіксованім числі досвідів;

− із усіх припустимих потрібно вибрати такий план, при якім статистична оцінка функції відгуку може бути отримана із заданою точністю при мінімальному обсязі випробувань.

Рішення завдання планування в першій постановці називається стратегічним плануванням експерименту, у другий – тактичним плануванням.

2.3 Схема програмних досліджень

Наведено загальні відмості про організацію досліджень, але не розглядаються ті інструментальні засоби, які будуть використовуватися для одержання й оцінки результатів експерименту.

Проведений експеримент являє собою спостереження за поведінкою кластерних індексів з різними структурами в СКБД під впливом вхідних впливів. У результаті буде отриманий набір експериментальних даних, для обробки яких необхідно використовувати статистичні методи обробки. Саме тому виміру при кожній комбінації керованих факторів необхідно робити серіями по 10 штук, щоб визначити основні статистичні показники.

У результаті проведення експерименту й наступного його аналізу повинні бути отримані показники, що характеризують показники продуктивності кластерних індексів, заснованих на різних деревах, а також дані відповіді на наступні питання:

− чи існує залежність між видом кластерних індексів і видами запитів (вставка, читання, редагування, видалення)

− чи існує залежність між витратами процесорного часу й кількістю переданих або запитуваних даних із БД

− при яких видах запитів і методах їх організації можна провести специфікацію структури індексу.

2.4 Аналіз методів виміру кластерних індексів у СКБД

Для виконання загального плану експерименту знадобиться змінити структуру кластерного індексу в існуючої СКБД на дерево ван Едме Боаса. Використання відкритого програмного забезпечення дозволить впровадити у вже існуючу базу даних зміна структури кластерного індексу й провести порівняння з незміненою версією БД. Виконання запитів до БД буде проводитися через розроблену утиліту – програмне забезпечення, що дозволяє виконувати запити до бази даних і вести підрахунок даних про час виконанні запиту й використанні пам'яті.

Метою математичної обробки результатів експерименту є не знаходження дійсного характеру залежності між змінними або абсолютної величини якої-небудь константи, а вистава результатів спостережень у вигляді найбільш простої формули з оцінкою можливої погрішності її використання.

У результаті проведених експериментів будуть отримані серії по 10 вимірів. Обробка буде складатися з наступних етапів.

Розрахунок незміщеної оцінки генеральної середньої (математичного очікування):

$$a = \frac{1}{n} \sum_{i=1}^{n} a_i ,$$

де: a_i – результат виміру, n – кількість вимірів;

Розрахунок незміщеної оцінки дисперсії

$$D = \frac{\sum_{i=1}^{n} (\Delta x_i)^2}{n - 1} ,$$

де $\Delta x_i = a - a_i$.

Дисперсія є показником розсіювання випадкової величини стосовно її середнього значення. Чим більше дисперсія, тим більше ймовірність, що випадкова величина буде ухвалювати значення, далекі від середнього;

Розрахунок середньоквадратичного відхилення:

$$s = \sqrt{D}.$$

Середнє квадратичне відхилення є кращим критерієм точності, тому що не відбувається компенсації позитивних і негативних помилок Δx_i і сильніше враховується дія великих помилок

Побудова гістограми доцільно із двох причин: для оцінки закону розподілу й виявлення промахів. Якщо в ряді вимірів зустрічаються результати, що різко відрізняються від більшої частини ряду, то виникає питання приналежності «, що вискакують» значень цьому ряду вимірів. Більші помилки мають малу ймовірність виникнення. Тому слід об'єктивно оцінити, чи є даний вимір промахом (тоді його виключають із ряду(, або ж це результат випадкового, але закономірного відхилення. Можна вважати кожний вимір промахом, якщо ймовірність випадкової появи такого значення є досить малою.

Таким чином, виконані ідентифікація й вибір рівнів факторів, обґрунтована загальна схема організації експериментальних досліджень, доведена необхідність проведення серій послідовних експериментів.

У результаті проведення експерименту й наступного його аналізу повинні бути отримані дані, що характеризують показники продуктивності різних СКБД, а також дані відповіді на запитання. За яких інтенсивностях, видах запитів і методах їх організації можна провести специфікацію структури кластерного індексу.

3 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕНЬ

3.1 Розробка алгоритмів для заміни структури кластерного індексу в БД

Алгоритмізацію можна визначити як процес спрямованої дії, необхідний для розробки алгоритмів, достатніх для реалізації створюваного об'єкта (системи), що задовольняє заданим вимогам [11]. Завершальним етапом алгоритмізації є випуск набору алгоритмів, що відображає рішення, прийняті проектувальником у необхідній формі. Для отамання результатів в атестаційній роботі розроблено клас алгоритмів з проведення експерименту по зміні структури кластерного індексу в скбд і аналіз отриманих результатів

Алгоритм виконання операції вставки елемента *x* складається з декількох частин: Якщо дерево порожнє або в ньому втримується єдиний елемент (*min*=*max*), то полям *min* і *max* присвоєно відповідні значення. Записана інформація в *min* і *max* повністю описує стан поточного дерева й задовольняє структурі нашого дерева. Інакше: Якщо елемент *x* більше *max* або менше *min* поточного дерева, то обновимо відповідне значення мінімуму або максимуму, а старий мінімум або максимум додамо в дерево.

Вставка в допоміжне дерево aux число high(x), якщо відповідне поддерево children[high(x)] до цього було порожнє.

Вставка числа low(x) у поддерево children[high(x)], за винятком ситуації поточного дерева, – це 1-дерево, і подальша вставка не потрібна (рис. 3.1).

Дана операція працює за час $O(\log\ k)$. На кожному рівні дерева виконується $O(1)$ операцій. Після цього можливі 2 випадки: піддерево *children*[*high*(*x*)] порожньо, і потрібно робити подальшу вставку й у нього, і в допоміжне дерево *aux*, або ж піддерево не порожньо, і проведений спуск на рівень нижче. Але якщо піддерево *children*[*high*(*x*)] порожньо, то вставка в нього буде виконана за $O(1)$, тому що буде оновлено поля *min* і *max*. Усі інші операції будуть виконуватися вже з допоміжним деревом *aux*, висота якого на 1 рівень менше, ніж висота поточного.

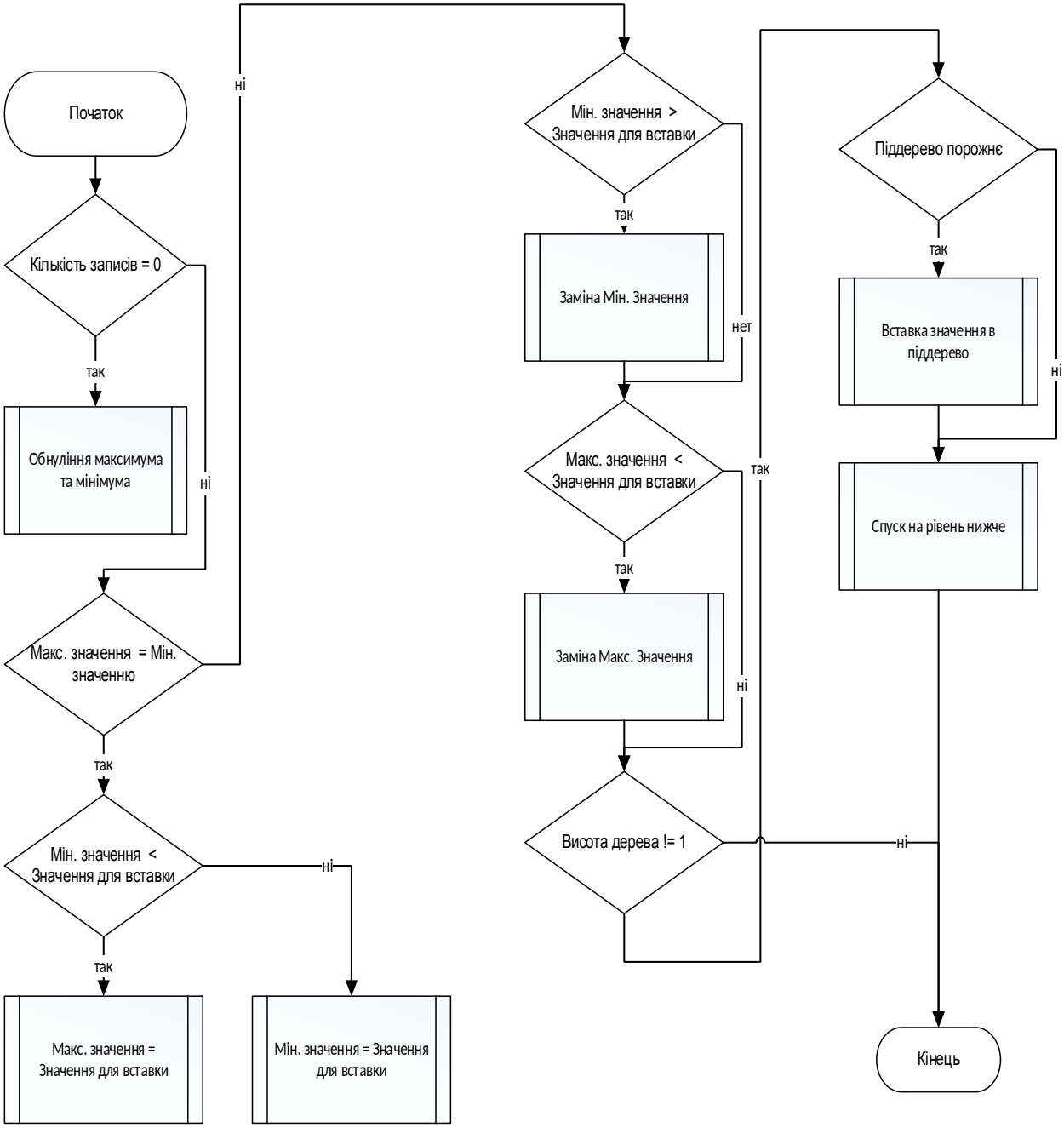

Рисунок 3.1 – Схема додавання нового запису

Якщо ж піддерево *children*[*high*(*x*)] не порожньо, то перехід до вставки елемента в це піддерево, висота якого так само на 1 менш, ніж у поточного.

У підсумку, щораз, виконавши $O(1)$ операцій, перехід до дерева, висота якого на 1 менше, чим у поточного. Отже, кількість операцій пропорційно висоті дерева, яка, як уже було показано, $O(1)$. Тобто операція вставки займе $O(\log k)$ часу.

3.2 Алгоритм видалення

Видалення з дерева також ділиться на декілька підзадач (рис. 3.2).

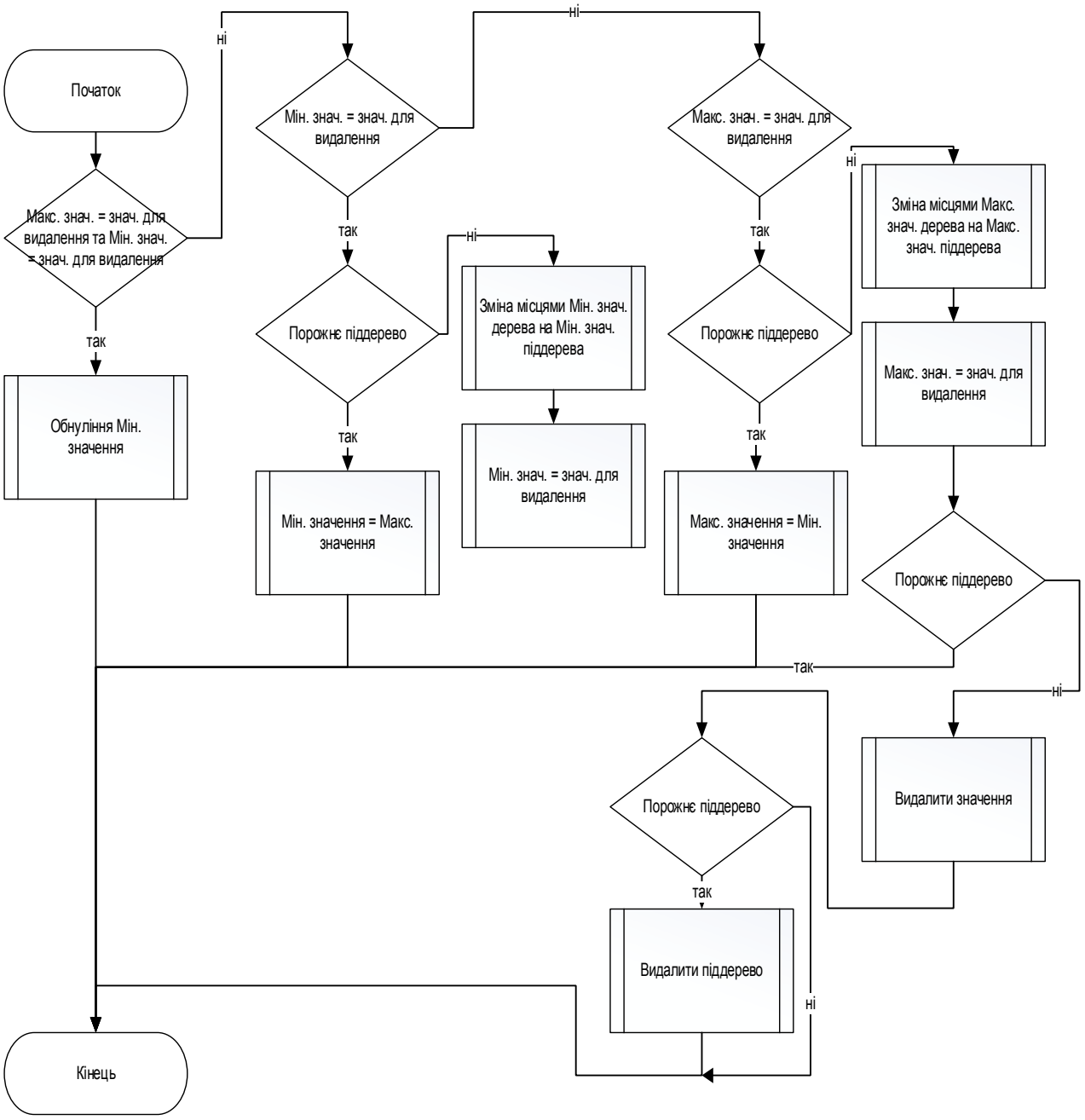

Рисунок 3.2 –Схема видалення записи

Якщо $min = max = x$, це обозначає у дереві один елемент, вилучення якого й відзначатиме, що дерево порожнє.

Якщо $x = min$, то ми повинні знайти наступний мінімальний елемент у цьому дереві, привласнити *min* значення другого мінімального елемента й вилучити його з того місця, де він зберігається. Другий мінімум – це або *max*, або *children*[*aux.min*].*min* (для випадку $x = max$ дії аналогічні).

Якщо ж $x <> min$ і $x <> max$, то ми повинні вилучити *low*(x) з поддерева *children*[*high*(x)].

Тому що в піддеревах зберігаються не всі біти вихідних елементів, а тільки частина їх, то для відновлення вихідного числа, по наявних старших і молодших битах, потрібно використовувати функцію *merge*. Також не можна забувати, що якщо буде видалено останнє входження x, то алгоритм повинен вилучити *high*(x) з допоміжного дерева.

Оцінка часу роботи операції *remove* така ж, як і в операції *insert*. На кожному рівні дерева ми проводиться $O(1)$ операцій і перехід до видалення елементів максимум у двох деревах (в одному піддереві й у допоміжному дереві), чиї висоти на один менше поточної. Але якщо ми робимо операцію видалення з допоміжного дерева, значить видалення з поддерева зажадало $O(1)$ операцій, тому що воно містило всього один елемент. У підсумку, кількість операцій пропорційно висоті дерева, тобто $O(\log k)$.

Алгоритм знаходження максимуму й мінімуму – тому що в дереві зберігається мінімальне й максимальне значення, то дані операції не вимагають нічого, крім виводу значення поля *min* або *max* відповідно до запиту. Час виконання даних операцій відповідно $O(1)$.

Алгоритм пошуку витікає з вищеописаної структури:

- якщо дерево порожнє, то число не втримується в нашій структурі;
- якщо число дорівнює полю *min* або *max*, то число в дереві є;
- інакше шукаємо число *low*(x) у піддереві *children*[*high*(x)].

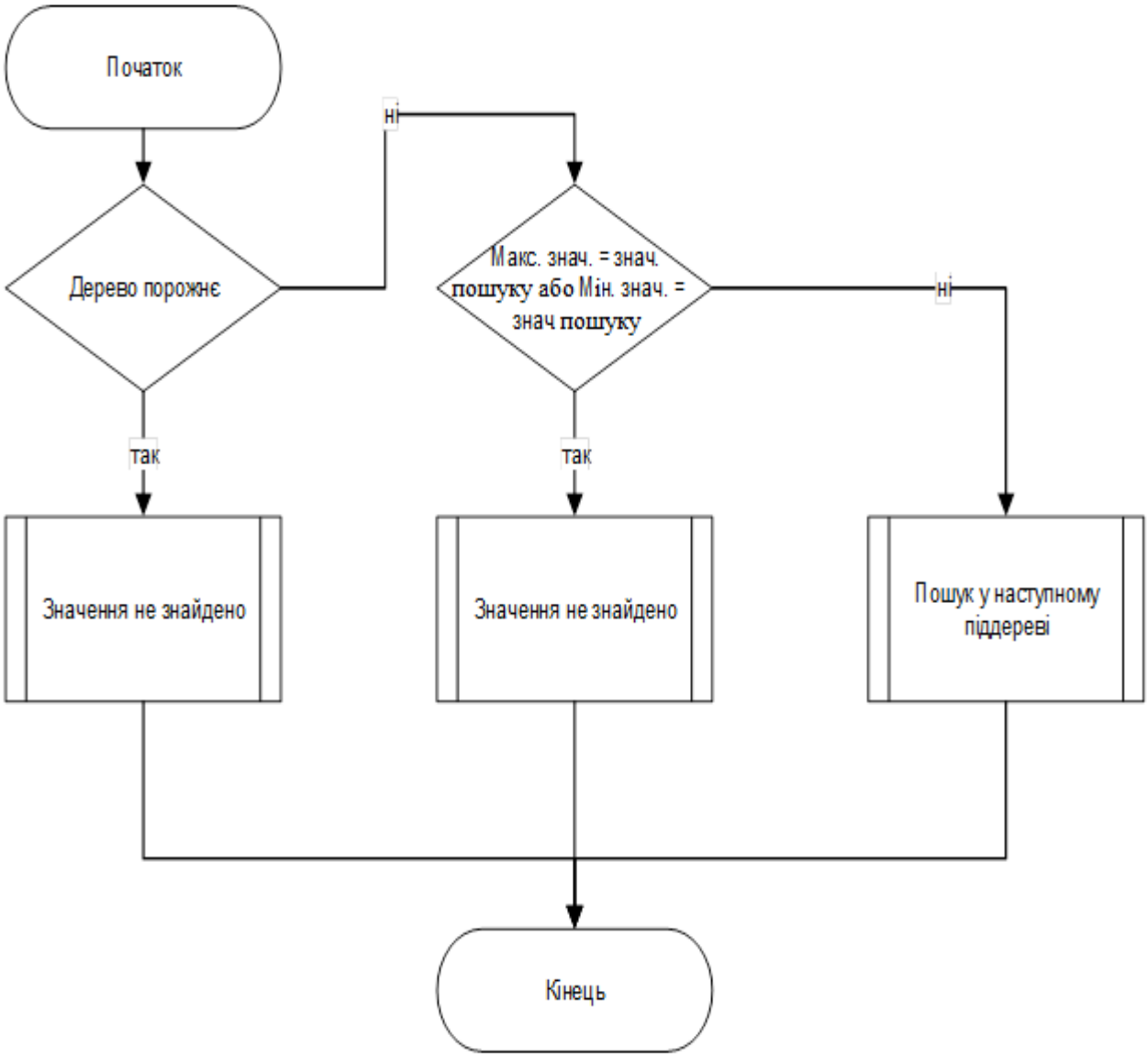

Рисунок 3.3 – Схема пошуку запису

Операція вставки елемента *x* складається з декількох частин:

Якщо дерево порожнє або в ньому міститься єдиний елемент (*min* = *max*), то виконується присвоєння полям *min* і *max* відповідні значення. Інформація записана в *min* і *max* повністю описує стан поточного дерева й задовольняє структурі нашого дерева.

Інакше: якщо елемент *x* більш ніж *max* або менше *min* поточного дерева, то обновляється відповідне значення мінімуму або максимуму, а старий мінімум або максимум додається в дерево.

Вставити в допоміжне дерево *aux* число *high*(*x*), якщо відповідне поддерево *children*[*high*(*x*)] до цього було порожнє.

Вставити число *low*(*x*) у піддерево *children*[*high*(*x*)]., за винятком ситуації дерево, що коли тече, – це 1вдерево, і подальша вставка не потрібна.

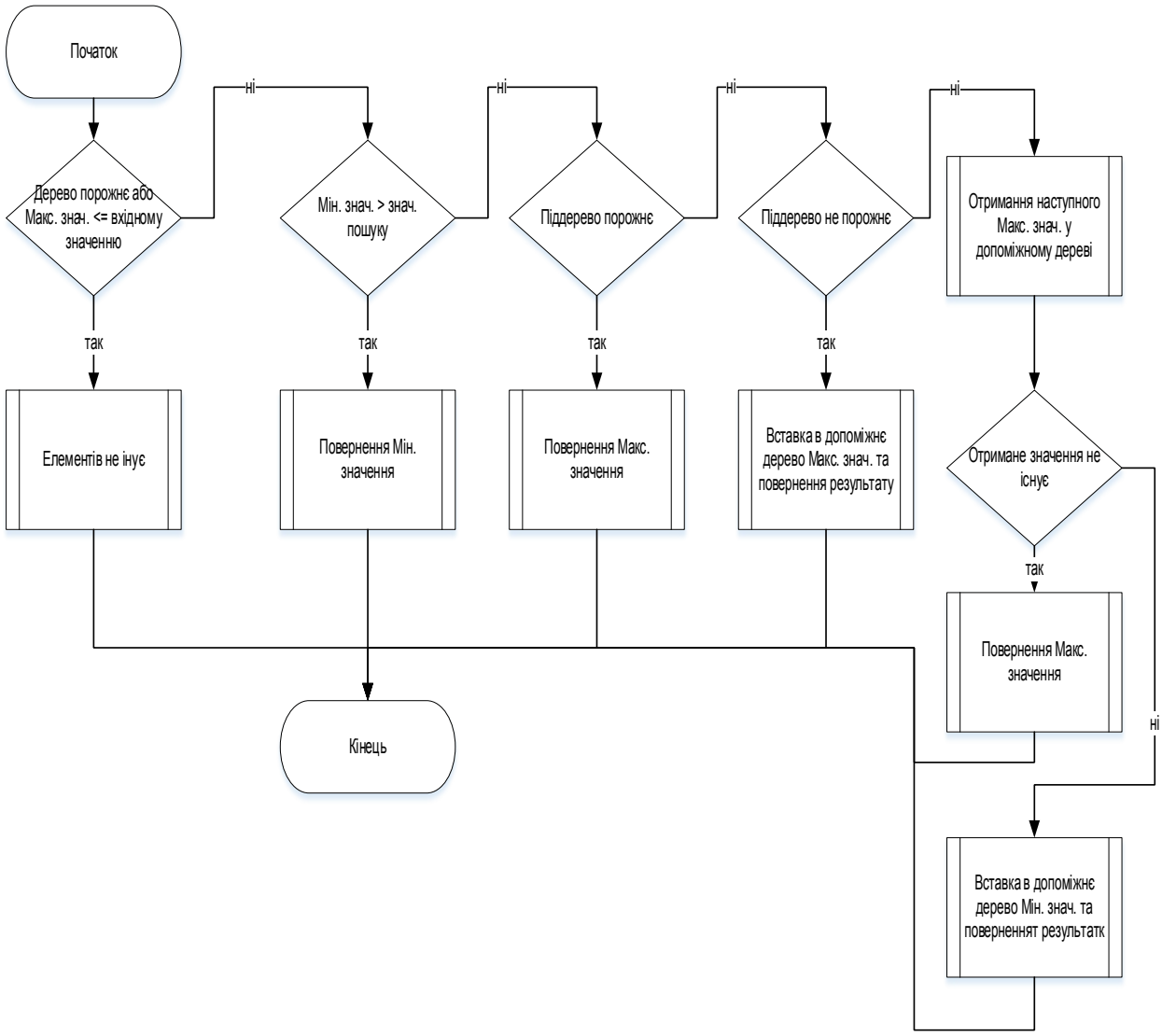

Рисунок 3.4 –Схема одержання наступного запису

3.3 Алгоритм утиліти, що виконує загальний план експерименту

Для проведення експерименту потрібно зробити вибір БД із відкритим вихідним кодом, у якій і буде зроблена заміна структури індексу.

Вибір буде здійснюється серед 3-х найбільш розвинених open-source БД: Sqlite, Perst, Firebird. Наведені БД мають схожі показники продуктивності й

способи обробки даних, а також не мають обмежень при використанні безкоштовних версій.

Sqlite – легка реляційна база даних, що вбудовується,. Вихідний код бібліотеки переданий у суспільне надбання. Sqlite не використовує парадигму клієнт-сервер, тобто движок Sqlite не є окремо працюючим процесом, з яким взаємодіє програма, а надає бібліотеку, з якої програма компонується й движок стає складовою частиною програми. На рис. 3.5 алгоритм утиліти, яка виконує загальний план експерименту.

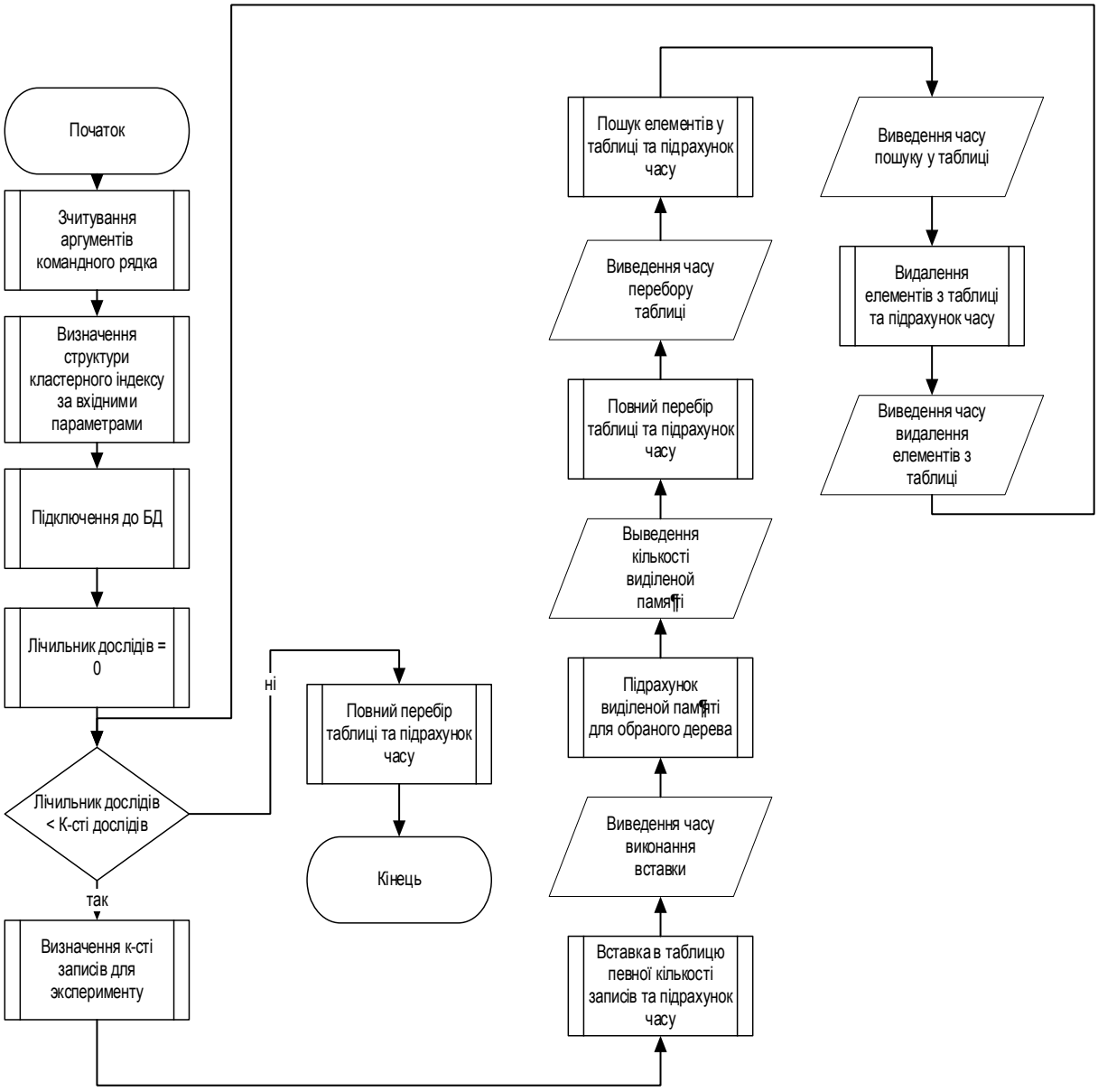

Рисунок 3.5 –Схема утиліти, що виконує загальний план експерименту

Таким чином, у якості протоколу обміну використовуються виклики функцій (API) бібліотеки Sqlite. Такий підхід зменшує накладні витрати, час відгуку й спрощує програму. Sqlite зберігає всю базу даних (включаючи визначення, таблиці, індекси й дані) у єдиному стандартному файлі на тому комп'ютері, на якім виконується програма. Простота реалізації досягається за рахунок того, що перед початком виконання транзакції записи весь файл, що зберігає базу даних, блокується; Acid-функції досягаються в тому числі за рахунок створення файлу журналу.

Кілька процесів або потоків можуть одночасно без яких-небудь проблем читати дані з однієї бази. Запис у базу можна здійснити тільки в тому випадку, якщо ніяких інших запитів у цей момент не обслуговується; а якщо ні, то спроба запису кінчається невдачею, і в програму вертається код помилки. Іншим варіантом розвитку подій є автоматичне повторення спроб записи протягом заданого інтервалу часу.

У комплекті поставки йде також функціональна клієнтська частина у вигляді файлу, що виконується, sqlite3, за допомогою якого демонструється реалізація функцій основної бібліотеки. Клієнтська частина працює з командного рядка, дозволяє звертатися до файлу БД на основі типових функцій ОС.

Завдяки архітектурі движка можливо використовувати Sqlite як на системах, що вбудовуються, так і на виділених машинах з гігабайтними масивами даних.

Perst – швидка й компактна БД, яка може використовуватися як для настільних PC, так і для мобільних обладнань.

Основні можливості:

− зберігає дані безпосередньо в об'єктах Java і .NET, крім часу на перетворення необхідне реляційним і об'єктно-орієнтованим БД;

− компактна (складання для .NET важить усього ледве більше 500 кБ);

− підтримка транзакцій;

− легко використовувані інструменти розробки;

− є можливість використовувати Sql-подібну мову запитів;

− експорт в XML;

− реплікація;

− повнотекстовий індекс;

− швидка система відновлення БД після збоїв (без використання logйфайлів).

СКБД Firebird (Firebirdsql) – компактна, кросплатформена, вільна система керування базами даних, що працює на ОС Linux, Microsoft Windows і різноманітних Unix-платформах.

У якості переваг Firebird можна відзначити багато-версійну архітектуру, що забезпечує паралельну обробку оперативних і аналітичних запитів (це можливо тому, що читаючі користувачі не блокують пишучих), компактність (дистрибутив 5Mb), високу ефективність і потужну язикову підтримку для збережених процедур і тригерів.

Firebird використовується в різних промислових системах (складські й господарські, фінансовий і державний сектори). Це комерційно незалежний проект C і C++ програмістів, технічних радників і розроблювачів мультиплатформенных систем керування базами даних, заснований на вихідному коді, у вигляді вільної версії Interbase 6.0.

Серед недоліків слід відзначити відсутність кэша результатів запитів, повнотекстових індексів.

Можна зробити висновок що СКБД Perst є кращим кандидатом для проведення дослідження. Головною особливістю Perst, для проведення експерименту, є наявність вихідних модулів, написаних з використанням C#.

Об'єктно-орієнтована БД `Perst`. Це дозволяє безпосередньо зберігати й завантажувати об'єкти цілком і надає зручний і потужний механізм для зберігання більших обсягів даних. Perst реалізує ряд тверджень:

− об'єкти постійного використання повинні бути доступні так само, як і тимчасові об'єкти;

− ядро бази даних повинне могти ефективно управляти набагато більшим обсягом даних, чим може міститися в основній пам'яті;

− відсутні спеціалізовані препроцесори, компілятори, віртуальні машини.

База даних Perst складається із двох основних просторів імен: Perst і Impl. В основному використовуються інтерфейси й класи із простору імен Perst, у той час як Impl забезпечує реалізацію цих класів і інтерфейсів. Ця структура дозволяє легко змінювати реалізації того або іншого класу, не зачіпаючи додатків, що використовують його.

На рис. 3.6 представлено діаграму класів, що реалізують кластерні індекси в БД Perst.

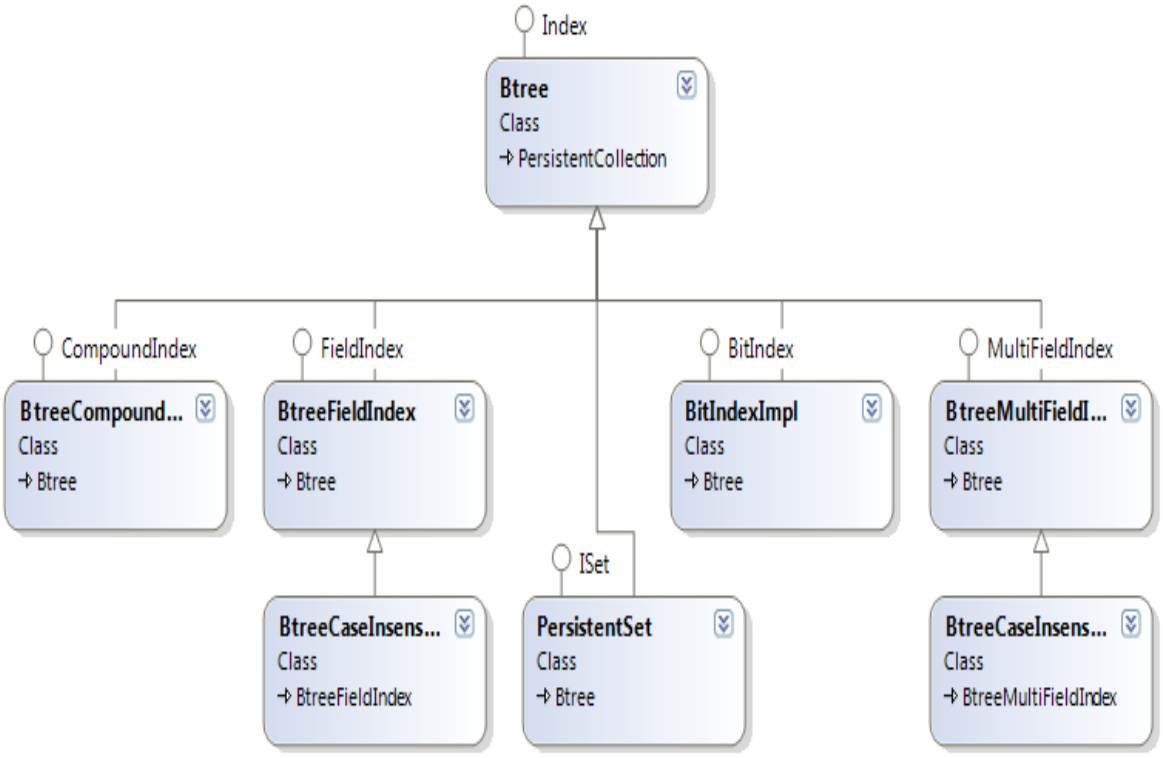

Рисунок 3.6 – Діаграма класів, що реалізують кластерні індекси в БД Perst

З погляду додатка, `Perst` складається із класу Persistent, який є базовим класом для постійних об'єктів і надає методи для завантаження/блокування збережених об'єктів; різних класів колекцій (індексів), що забезпечують швидкий доступ до об'єктів; класу `Storage`, який є й відповідальним за керування транзакціями, і використовується в якості фабрики по індексу класів.

У БД Perst кластерні індекси представлені у вигляді B-дерев. Що б зробити повноцінну заміну структури ВвДерева на дерево ван Едме Боаса потрібно

перевизначити всі методи певні в класі Btree, який і представляє структуру ВвДереві в Perst. Btree клас успадковує Persistentcollection, Index інтерфейси, через які проводиться маніпулювання даними.

На рис. 3.7 наведено основні методи, описані в класі `Btree`.

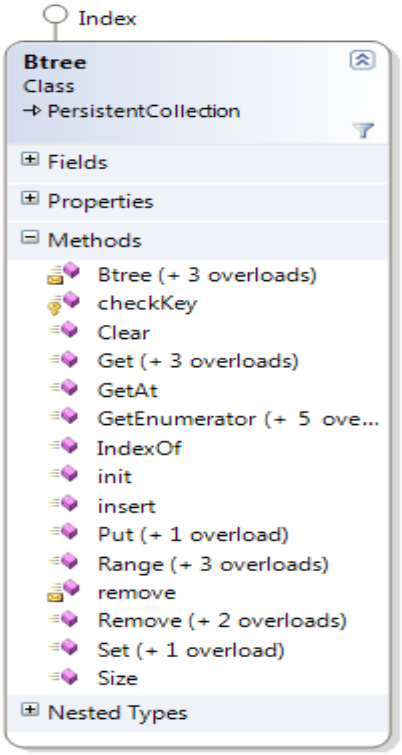

Рисунок 3.7 – Методи в класі Btree

Для проведення дослідження не потрібно заміняти всі методи, описані в класі Btree. Функціонування індексу можливо при перевизначенні наступних методів:

- Init() – ініціалізація дерева;
- Count() – повертає кількість елементів у дереві;
- Getenumerator() – ітерації по елементах дерева;
- Get() – повертає об'єкт по ключу;
- Find() – робить пошук по всім дереву;
- Insert() – вставка об'єкта в дерево;
- Remove() – видалення об'єкта з дерева.

4 ОПИС РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Проектування утиліти програмного експерименту

Для виконання плану експерименту буде розроблена додаткова утиліта – ПО, що дозволяє підключитися до БД і робити операції вставки, пошуку, видалення й вибірки. Утиліту написано мовою C#, так как вихідні модули Perst також написано з використанням мови C#, бо C# – це об'єктно-орієнтована мова програмування. Розроблена у компанії Microsoft як мова розробки додатків для платформи Microsoft .NET Framework, вонавідноситься до родини мов з C-подібним синтаксисом, з них її синтаксис найбільш близький до C++ і Java.

Мова має статичну типізацію, підтримує поліморфізм, перевантаження операторів (у тому числі операторів явного й неявного приведення типу), делегати, атрибути, події, властивості, узагальнені типи й методи, ітератори, анонімні функції з підтримкою замикань, LINQ, виключення, коментарі у форматі XML.

Середовищем розробки обрано Microsoft Visual Studio, що є зручним інструментом для розробки програм, що використовують платформу .NET. Даний продукт дозволяє розробляти як консольні додатки, так і додатка із графічним інтерфейсом, у тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-додатки. Visual Studio містить у собі редактор вихідного коду з підтримкою технології Intellisense і можливістю найпростішого рефакторінгу коду. Вбудований отладчик може працювати як отладчик рівня вихідного коду, так і як отладчик машинного рівня. Інші інструменти, що вбудовуються, містять у собі редактор форм для спрощення створення графічного інтерфейсу додатка, веб-редактор, дизайнер класів і дизайнер схеми бази даних. Використовуючи ці компоненти, можна буде без усяких завад зробити написання утиліти для роботи із БД Perst.

4.2 Опис інтерфейсу користувача утиліти для проведення експерименту

Утиліта являє собою консольний додаток, що виконує запити до БД час, що підраховує, виконання запитів і кількість використаної пам'яті.

На рис. 4.1 представлений приклад результатів утиліти, які будуть отримані при завершенні роботи.

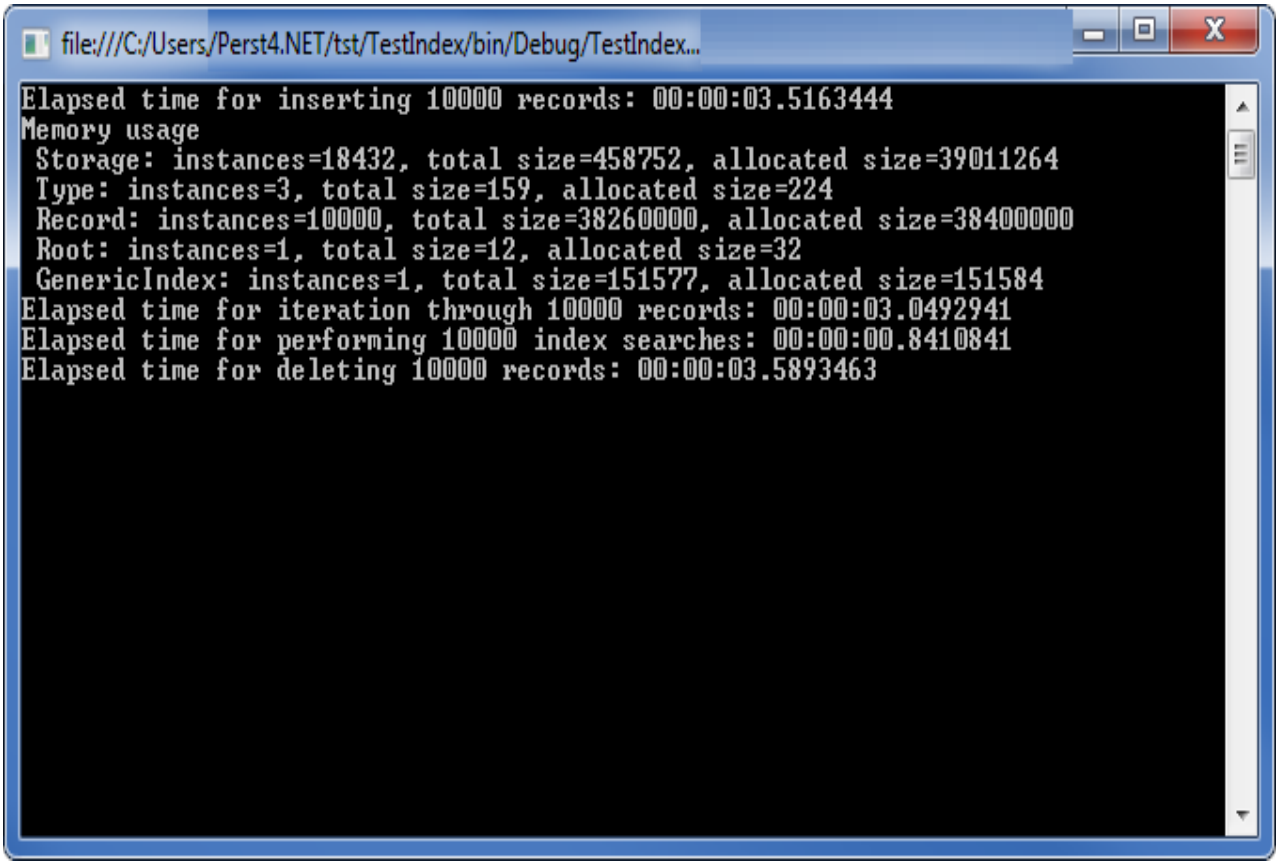

Рисунок 4.1 – Результати роботи утиліти

Консольний додаток буде перебувати на одній робочій машині разом з базою даних. Тому додаток не є розподіленим, звертається прямо до БД. У цьому випадку затримками з'єднання й передачі даних до БД можна зневажити.

Архітектуру додатку представлено на рис. 4.2

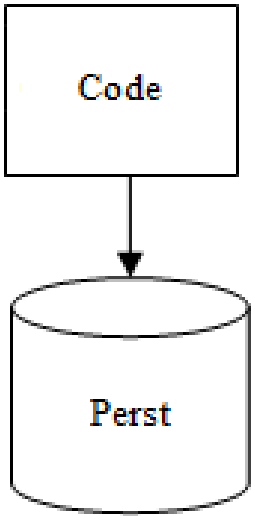

Рисунок 4.2 – Архітектура додатку для запитів до БД

4.3 Проект додатку

Загальна схема роботи утиліти:

- підключення до БД Perst;
- добуток операції вставки в таблицю з індексом;
- підрахунок пам'яті виділеної для дерева;
- повний перебір записів таблиці;
- пошук випадкових елементів;
- видалення всіх записів таблиці.

При виконанні кожного кроку також буде зроблений вимір часу.

У результаті проведення експериментів по оцінці продуктивності СКБД були отримані відгуки систем для кожного опиту, усього опитів було поставлено 10 для кожного виду СКБД. Відгуки представлені в секундах.

На рис. 4.3 зображено діаграму класів додатку.

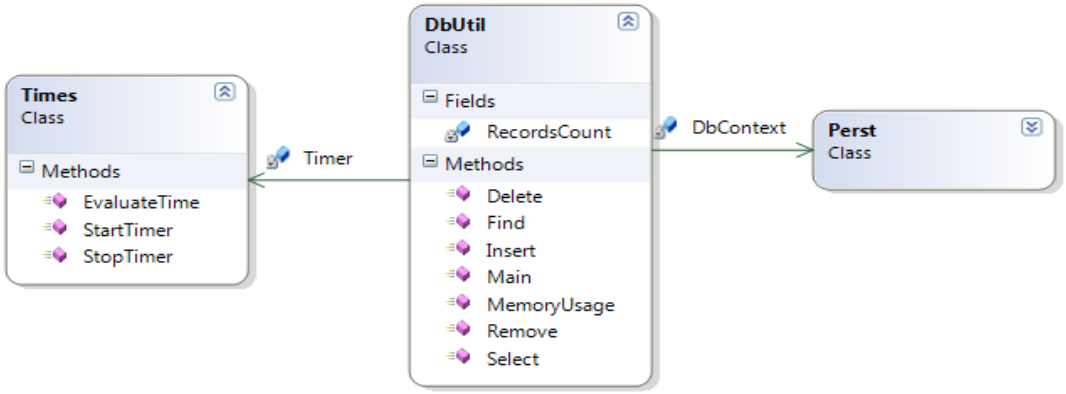

Рисунок 4.3 – Діаграма класів додатку

4.4 Опис проведення програмного експерименту

У таблицях 4.1 – 4.2 наведено результати роботи БД при використанні кластерних індексів на основі B дерев.

Таблиця 4.1 – Результати роботи БД для 10000 записів

№ п/п	Вставка (мс.)	Вибірка (мс.)	Пошук (мс.)	Видалення (мс.)
1	743,043	106,006	117,011	148,009
2	789,046	125,007	120,009	131,008
3	715,042	144,008	100,006	114,007
4	763,044	116,007	131,008	146,009
5	790,046	85,0049	87,0051	95,0055
6	782,046	93,0054	102,006	125,007
7	772,045	129,008	106,006	115,007
8	820,048	87,005	90,0053	107,006
9	746,043	104,006	136,008	149,009
10	723,042	90,0052	89,0052	127,007

Використання пам'яті склало – 39,011,264 байтів.

Таблиця 4.2 – Результати роботи БД для 100000 записів

№ п/п	Вставка (мс.)	Вибірка (мс.)	Пошук (мс.)	Видалення (мс.)
1	5078,3653	2172,074	1698,099	5960,108
2	4982,2783	2038,101	1725,1	6158,405
3	5046,3053	2095,116	1698,099	5972,323
4	4965,289	2143,125	1719,1	6003,402
5	4919,2746	2111,123	1732,101	6036,421
6	5135,2989	2179,126	1773,103	6141,369
7	5084,3017	2147,125	1723,1	5983,319
8	5037,2932	2297,134	1753,102	6122,385
9	4915,2861	2161,126	1718,1	6102,367
10	5089,3195	2192,128	1759,102	5969,39

Використання пам'яті – 384,000,000 байт

4.5 Результати программного експерименту дерев ван Едме Боаса

У таблицях 4.3 та 4.4 наведено результати роботи БД при використанні кластерних індексів на основі дерев ван Едме Боаса.

Таблиця 4.3 – Результати роботи БД для 10000 записів

№ п/п	Вставка (мс.)	Вибірка (мс.)	Пошук (мс.)	Видалення (мс.)
1	753,669	95,1	152,39	132,6

Продовження табл. 4.3

2	716,786	96,65	135,34	143,14
3	799,767	82,68	131,4	156,39
4	758,377	113,22	152,65	176,73
5	757,793	128,37	141,5	159,95
6	770,699	84,71	150,29	150,38
7	710,418	96,2	151,22	141,25
8	795,23	100,75	143,98	126,3
9	776,984	104,96	140,51	165,17
10	738,143	122,78	140,48	161,89

Таблиця 4.4 – Результати роботи БД для 100000 записів

№ п/п	Вставка (мс.)	Вибірка (мс.)	Пошук (мс.)	Видалення (мс.)
1	4528,962	1510,104	1344,502	4687,744
2	4727,163	1447,699	1422,24	4825,241
3	4512,252	1522,488	1399,628	4828,459
4	4676,995	1493,487	1426,158	4816,267
5	4646,75	1460,623	1389,2	4771,798
6	4566,622	1596,882	1384,298	4791,67
7	4658,824	1534,557	1399,396	4877,75
8	4539,844	1441,415	1372,65	4793,237
9	4668,471	1575,67	1410,355	4989,801
10	4556,479	1637,714	1386,589	4800,656

Використання пам'яті – 510,126,860 байтів

Таблиця 4.5 – Незміщена оцінка для кластерних індексів на основі B-дерев

	Вставка (мс.)	Вибірка (мс.)	Пошук (мс.)	Видаленн я (мс.)
10000	763,044	116,007	107,80696	146,009
100000	5025,30119	2153,6178	1729,9006	6044,9489
1000000	159318,767	1184493,514	90178,18603	100305,2522

Таблиця 4.6 – Незміщена оцінка для кластерних індексів на основі дерев ван Едме Боаса

	Вставка (мс.)	Вибірка (мс.)	Пошук (мс.)	Видалення (мс.)
10000	757,7866	102,542	143,976	151,38
100000	4608,2362	1522,0639	1393,5016	4818,2623
1000000	129730,412	1011700,394	79999,3018	88060,3827

Результати програмного експерименту з озрахунку незміщеної оцінки дисперсії наведено в табл. 4.7 та 4.8

Таблиця 4.7 – Незміщена оцінка дисперсії для кластерних індексів на основі B-дерев

	Вставка (мс.)	Вибірка (мс.)	Пошук (мс.)	Видалення (мс.)
10000	967,3275037	362,9318	276,8063275	302,6504
100000	5187,707094	4185,226284	561,0200416	5524,107169
1000000	434553,4812	2377323,8	64427,81598	314869,0434

Таблиця 4.8 – Незміщена оцінка дисперсії для кластерних індексів на основі дерев ван Едме Боаса

	Вставка (мс.)	Вибірка (мс.)	Пошук (мс.)	Видалення (мс.)
10000	802,7640158	204,253916	50,147784	219,5001
100000	4598,2362	1523,064	1264,802	4503,262
1000000	584470,912	25553026,88	24253,68368	2289,997097

Таким чином, спроектований прототип ПЗ, необхідного для проведення експерименту – утиліти, що дозволяє робити вставку, видалення, пошук, вибірку із БД Perst, а також зробити виміри часу. Були складені загальна діаграма архітектури додатка, а також діаграма класів. Обране інструментальне середовище для розробки допоміжного ПЗ.

Описані алгоритми для заміни структури кластерного індексу БД. Проведений аналіз методів алгоритмізації й обрані блок-схеми як засіб для опису алгоритму, тому що він має винну гнучкість і наочністю.

За допомогою псевдокоду представлені наступні алгоритми:

- додавання нового запису в дерево індексу;
- видалення запису з дерева індексу;
- пошук по дереву індексу;
- знаходження наступного елемента;
- одержання максимуму й мінімуму.

5 ОПИС МОЖЛИВОСТІ ВИКОРИСТАННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

Порівняльний аналіз кластерних індексів заснованих на різних структурах представлений в графічному поданні результатів

На рис. 5.1 наведено екранну форму, що представляє графік, який порівнює середній час відгуків по кожному дереву для 10000 записів.

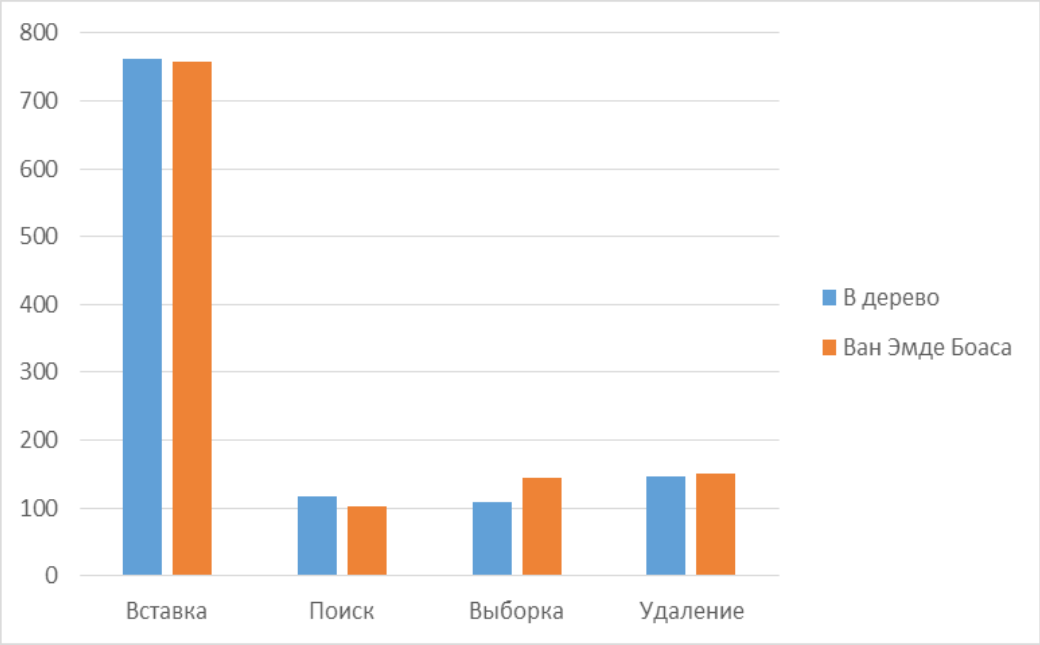

Рисунок 5.1 – Час відгуків по кожному дереву для 10000 записів

На рис. 5.2 представлений графік, який порівнює середній час відгуків по кожнім дереву для 100000 записів.

Графік, що порівнює середній час відгуків по кожнім дереву для 1000000 записів представлений на рис. 5.3.

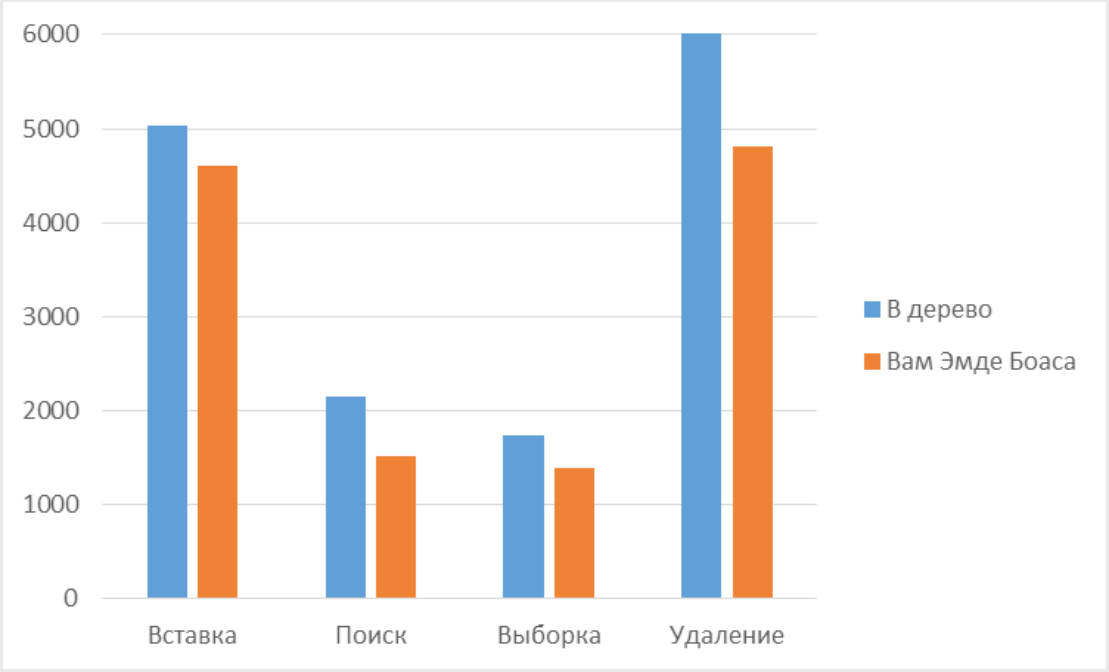

Рисунок 5.2 – Час відгуків по кожнім дереву для 100000 записів

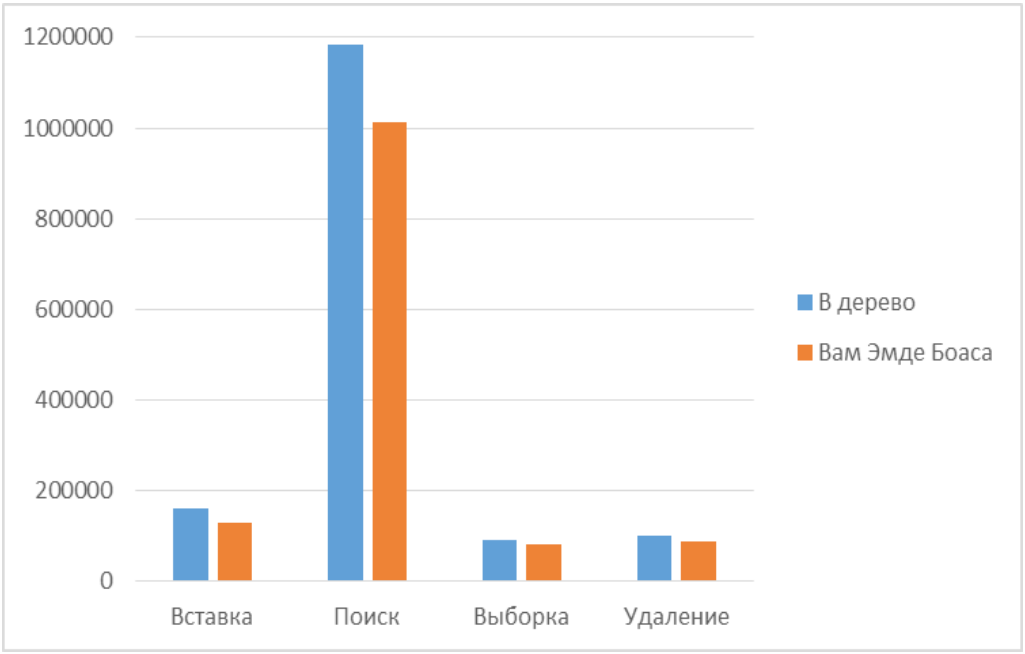

Рисунок 5. 3 – Час відгуків по кожнім дереву для 1000000 записів

На основі проведеного програмного експерименту можуть бути зроблені рекомендації з використання різних структур в основі кластерних індексів у СКБД. Після проведення експерименту можна зробити висновок, що заміна структури кластерного індексу на дерево ван Едме Боаса в СКБД `Perst`

дозволить підвищити продуктивність СКБД, але призведе до різкого збільшення використання пам'яті для зберігання нової структури. У всіх експериментах, що проведено, кластерні індекси на основі дерева ван Едме Боаса показали більшу продуктивність, крім першого досвіду, де кількість даних, що тестувалися, було найменшим. Це дає підстави зробити висновки, що при використанні таблиць із розміром записів близько 10000 слід використовувати кластерні індекси на основі B-дерев, що дасть можливість і заощадити пам'ять і домогтися високої продуктивності СКБД.

Перераховані вище алгоритми дозволили перейти до етапу проведення експерименту.

Також був безпосередньо проведений експеримент, отримані розрахунки для СКБД Perst при різних операціях. Зроблений розрахунок і аналіз математичної моделі. Були побудовані діаграми, що порівнюють відгуки для різних структур, використаних у кластерному індексі СКБД `Perst`. Значення відгуків показали виграш кластерних індексів з використанням дерев ван Едме Боаса в продуктивності при великій кількості даних.

По закінченню експерименту й обробки результатів були зроблені висновки по доцільності використання різних структур.

ВИСНОВКИ

Виконано аналіз предметної області, розглянуті основні принципи, що стосуються систем баз даних. Визначене поняття СКБД, виділені основні функції, зроблена класифікація індексів у СКБД.

Розглянуто деякі реалізації структури кластерних індексів та шляхи рішення поставленої в даній роботі проблеми – а саме збільшення показників продуктивності кластерних індексів у СКБД. Найбільш актуальним був обраний спосіб використання іншої структури в основі кластерного індексу – дерева ван Едме Боаса.

Проведена ідентифікація й вибір рівнів факторів, обґрунтована загальна схема організації експериментальних досліджень, доведена необхідність проведення серій послідовних експериментів.

Описані алгоритми для заміни структури кластерного індексу БД. Проведений аналіз методів алгоритмізації й обрані блок-схеми як засіб для опису алгоритму, тому що вони мають винну гнучкість і наочністю. За допомогою псевдокоду представлені алгоритми: додавання новому запису в дерево індексу; видалення запису з дерева індексу; пошук по дереву індексу; знаходження наступного елемента; одержання максимуму й мінімуму; більш приватні алгоритми роботи програмного забезпечення, для проведення експерименту.

Спроектований прототип ПЗ, необхідного для проведення експерименту – Treeutil. Складені загальна діаграма архітектури додатка, а також більш детальна діаграма класів. Обране інструментальне середовище для розробки даного допоміжного ПЗ.

Також був безпосередньо проведений експеримент, отримані відгуки для СКБД Perst при різних операціях. Був зроблений розрахунок і аналіз математичної моделі. Були побудовані діаграми, що порівнюють відгуки для різних структур, використаних у кластерном індексі СКБД Perst. Значення відгуків показали

виграш кластерних індексів з використанням дерев ван Едме Боаса в продуктивності при великій кількості даних.

По закінченню експерименту й обробки результатів були зроблені висновки по доцільності використання різних структур. Складені графіки, що порівнюють індекси на підставі різних дерев за експериментальними значенням відгуків, що явно показують виграш дерев ван Едме Боаса в продуктивності при великій кількості записів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1 Алгоритм кластеризации *k*-means // Електроний ресурс // URL: http://robocraft.ru/blog/computervision/1061.html (дата звернення 20.03.20 р.)

2 Айвазян С. А., Бухштабер В. М., Енюков И. С., Мешалкин Л. Д. Прикладная статистика: Классификация и снижение размерности. М.: Финансы и статистика. – 2009.

3 Афонин А.А., Крейнес М.Г. Кластеризация текстовых коллекций: помощь при содержательном поиске и аналитический инструмент // Сборник научных статей «Интернет-порталы: содержание и технологии». Выпуск 4 / ФГУ «Информика». – М.: Просвещение. – 2007. – С. 510-537.

4 Барсегян А. А. Методы и модели анализа данных: OLAP и Data mining. / А. А. Барсегян, М. С. Куприяенов, В. В. Степаненко, И. И. Холод. – СПб. : БХВ-. – 2014.

5 Басакер Р., Саатн Т. Конечные графы и сети. Перевод с английского. – М: Наука. – 2003.

6 Библиотека работы с DOM HTML-документов для C# // Электронный ресурс // URL: http://htmlagilitypack.codeplex.com/ (дата звернення 20.03.20 р.)

7 Воронцов К. В., Колосков А. О. Профили компактности и выделение опорных объектов в метрических алгоритмах классификации // Искусственный интеллект. – 2016. № 2. – С. 30-33.

8 Воронцов К. В. Лекции по алгоритмам кластеризации и многомерного шкалирования // Електронний ресурс URL: http://www.ccas.ru/voron/download/Clustering.pdf (дата звернення 28.02.20 р.)

9 Гайдышев И. Анализ и обработка данных: специальный справочник. – СПб.: Питер – 2011. – 752 с.

10 Гулин В.В. Исследование и разработка методов и программных средств классификации текстовых документов // Електронний ресурс URL:

http://www.mpei.ru/LANG/RUS/Publish/InfoAcadCncl/2013/GulinVV.pdf (дата звернення 25.03.20 р.)

11 Дунаев Е. В. Автоматическая рубрикация web-страниц в интернет-каталоге с иерархической структурой / Е. В. Дунаев, А. А. Шелестов // Интернет-математика 2005. Автоматическая обработка веб-данных. – М. 2005. – С. 382-398 .

12 Bien J., Tibshirani R. Hierarchical Clustering With Prototypes via Minimax Linkage // Journal of the American Statistical Association. 2011 // Електронний ресурс // URL: http://faculty.bscb.cornell.edu/~bien/papers/jasa2011minimax.pdf (дата звернення 22.03.20 р.)

13 Chakarbarti S. Mining the web: discovering knowledge from hypertext data. San Francisco: Morgan Kaufmann Publishers. – 2019.

14 Easily parse HTML Documents in C# // Електронний ресурс // URL: http:// olussier.net/2010/03/30/easily-parse-html-documents-in-csharp/ (дата звернення 22.02.20 р.)

15 Kogan J. Introduction to Clustering Large and High-Dimensional data. – N. Y. : Cambridge University Press. – 2006.

16 Robertson S. Understanding Inverse Document Frequency: on theoretical arguments for IDF // Journal of Documentation, 2014, №5. – P. 503-520

17 Филлипс Д. Методи аналізу мереж / Д. Филлипс, А. Гарсиа-Диас. – М.: Мир, 1984. – 496 с.

18 Глушенкова І. С. Прийняття рішень у системах керування земельними ресурсами / І. С. Глушенкова // Застосування інформаційних технологій у підготовці та діяльності сил охорони правопорядку : зб. тез допов. міжнар. наук.-практичн. конф. (березень 2010). – Х. : Академія внутрішніх військ МВС України, 2010. – С. 71–72.

19 Лєсна Н.С., Фоменко О.А. Дослідження методів і моделей асинхронного оновлення даних для підвищення продуктивності web-систем /Наука онлайн: міжнародний електронний науковий журнал. - 2019. - №1..- С.6-10

20 Глушенкова І. С. Формальна модель прийняття рішень про стан складних об'єктів / І. С. Глушенкова // Інформаційно-Керуючі системи і комплекси : зб. тез допов. міжнар. наук.–техніч. конф. (квітень 2018). – Миколаїв : ІАЕ НУК, 2018. – С. 51

21 Aref M. A multi-agent system for natural language understanding. // In Proc. of International Conference on Integration of Knowledge Intensive Multi-Agent Systems (KIMAS '03) . – USA. – 2003. – P. 36-40.

22 Web search engine .Wikipedia .The Free Encylopedia. URL: http://en.wikipedia.org/wiki/Web_search_engine. (дата звернення: 27.04.2020).

23 Sheth B., Maes P. Evolving agents for personalized information filtering. // In Proc. of IEEE Conference on Artificial Intelligence for Applications (CAIA-93) . – 1993. – P. 345–352

24 Lieberman H. Letizia: An agent that assists web browsing. // In Proc. of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95) . – Canada. 2015. – P. – 924–929.

25. Письман Д. М., Дегтерев А. С. GERT-мережний аналіз часу виконання задачі на неспеціалізованому гетерогенному кластері // Фундаментальні дослідження. 2015. № 4. С. 79-80.

26. Царьов, М. Ю., Царьов Р. Ю., Шевчук С. Ф. Модифікація Герт-сети для аналізу часових характеристик мережних моделей // Вісник Сибгау. 2009. № 1 (22). Ч. 2. С. 74-78.

27. Лєсна Н. С., Чачанідзе С. С. Дослідження методів аналізу продуктивності СКБД.// Topical Issues Of The Development Of Modern Science - Abstracts of IX International Scientific and Practical Conference. Sofia, Bulgaria, 6-8 May 2020, 504 - 508 pp