Індексна організація даних та організація даних типу «збалансоване дерево».

***Індекс* – це певна структура даних, яка в якості „параметра" отримує задану „властивість" записів (зазвичай це значення одного чи кількох полів) та „швидко" знаходить записи, що мають цю властивість.**

Вдалий індекс дозволяє відшукати потрібний запис, звертаючись лише до невеликої частини всіх записів (в ідеалі – безпосередньо до групи шуканих записів). Поля, на основі яких конструюється індекс, називають „ключем пошуку" або просто „ключем".

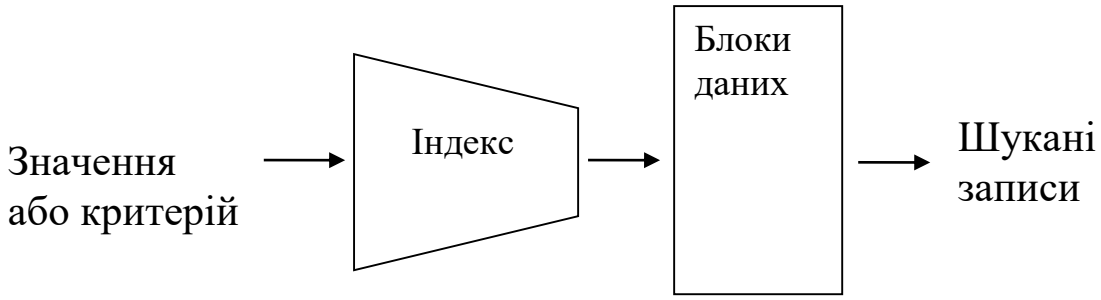


Рисунок 1 - Індекс дозволяє знайти записи, що задовольняють заданому критерію пошуку

Якщо існує таблиця в БД, можна створити на її основі індекс – послідовність блоків, яка містить лише ключі записів даних і посилання на ці записи, тобто адреси записів на диску або в оперативній пам'яті.

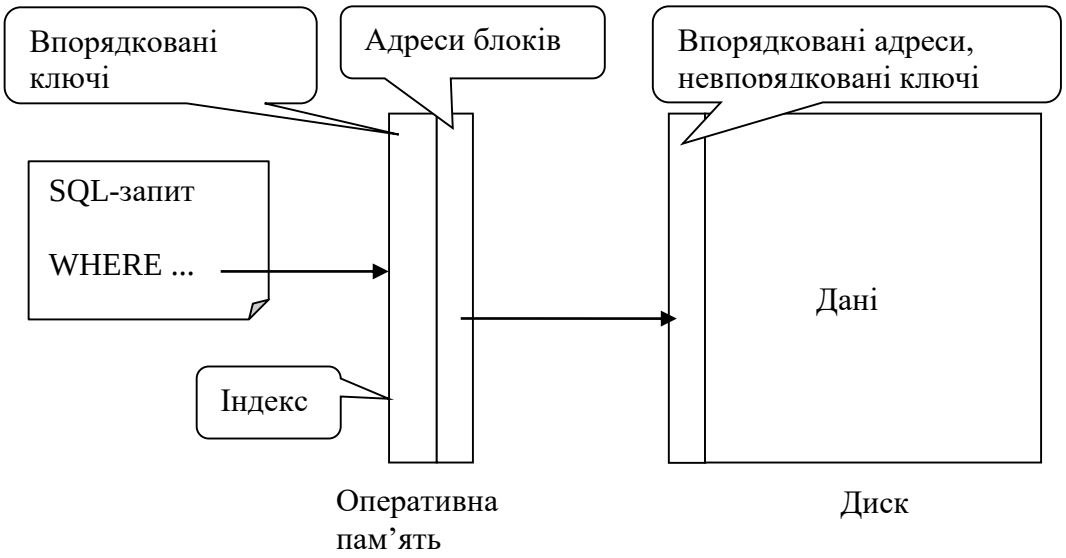


Рисунок 2 - Пошук кортежів з певним значенням ключа за допомогою індексу

Розмір індексу зазвичай набагато менше розміру таблиці, тому часто його можна повністю завантажити в оперативну пам'ять.

Архітектура xBase

Бази на dbf-файлах набули великої популярності в СРСР з середини 80-х років (з появи ПК) до кінця 90-х років (розповсюдження Microsoft Windows & Office). Також популярними були відповідні системи програмування (Dbase, Clipper, Foxbase, згодом FoxPro; спільна назва - xBase). Така база є каталогом, який містить набір файлів з розширенням .dbf. Цей файл містить таблицю: заголовок з описом структури запису фіксованої довжини, і далі самі

записи. Індекси (можливо, кілька для dbf-файлу) також існують як файли. Зміст індексу: відсортований по значенню ключа список значень ключових полів і відповідних номерів записів.

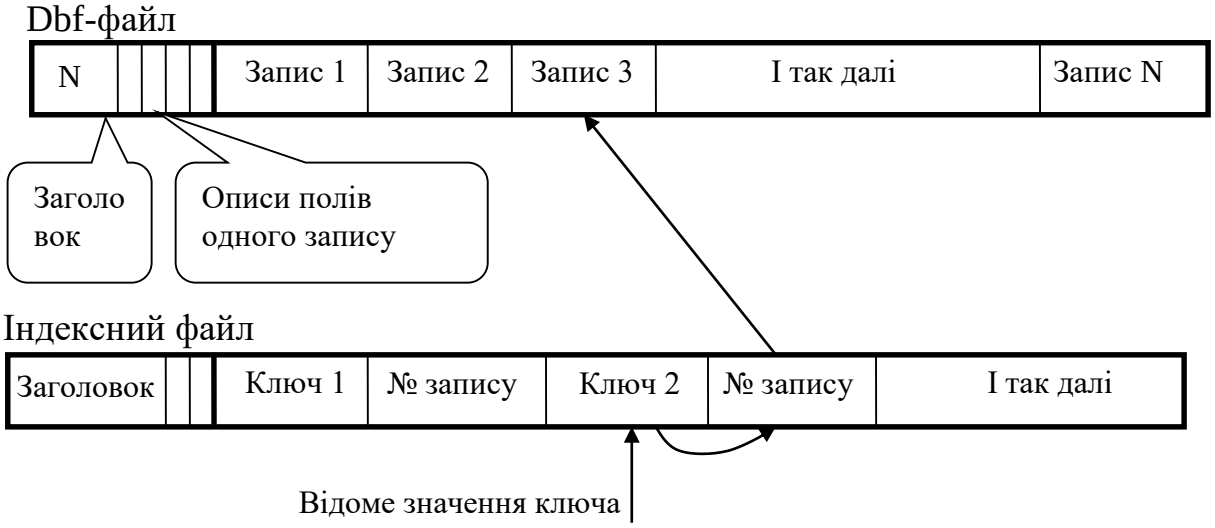


Рисунок 3 - Структура dbf- та його індексного файлу

Послідовність індексного доступу по ключу до запису в dbf-файлі:

- Знаючи значення ключа, методом половинного ділення (наприклад) знаходимо його в індексному файлі.
- Визначаємо № запису в dbf.
- Спираючись на фіксовану довжину запису в dbf, обчислюємо адресу потрібного запису.
- Читаємо запис у пам’ять.

Попри чималу продуктивність, такі бази є вразливими до дискових збоїв та несанкціонованих змін, тому число записів у dbf-файлах реально обмежувалось мільйонами (КНПЗ). Досить часто доводилось перебудовувати індекси. Замість запитів доводилось програмувати алгоритми відбору даних. Для корпоративного рівня вони виявились недостатньо захищені, а як настільні СУБД були витіснені MsAcess.

В сучасних СУБД таблиці і індекси містяться в файлі бази даних.

Щільний індекс

Щільним (dense) називають індекс, що містить ключі для кожного запису таблиці.

Розріджений (sparse) індекс зазвичай містить по одному ключу для кожного блоку таблиці.

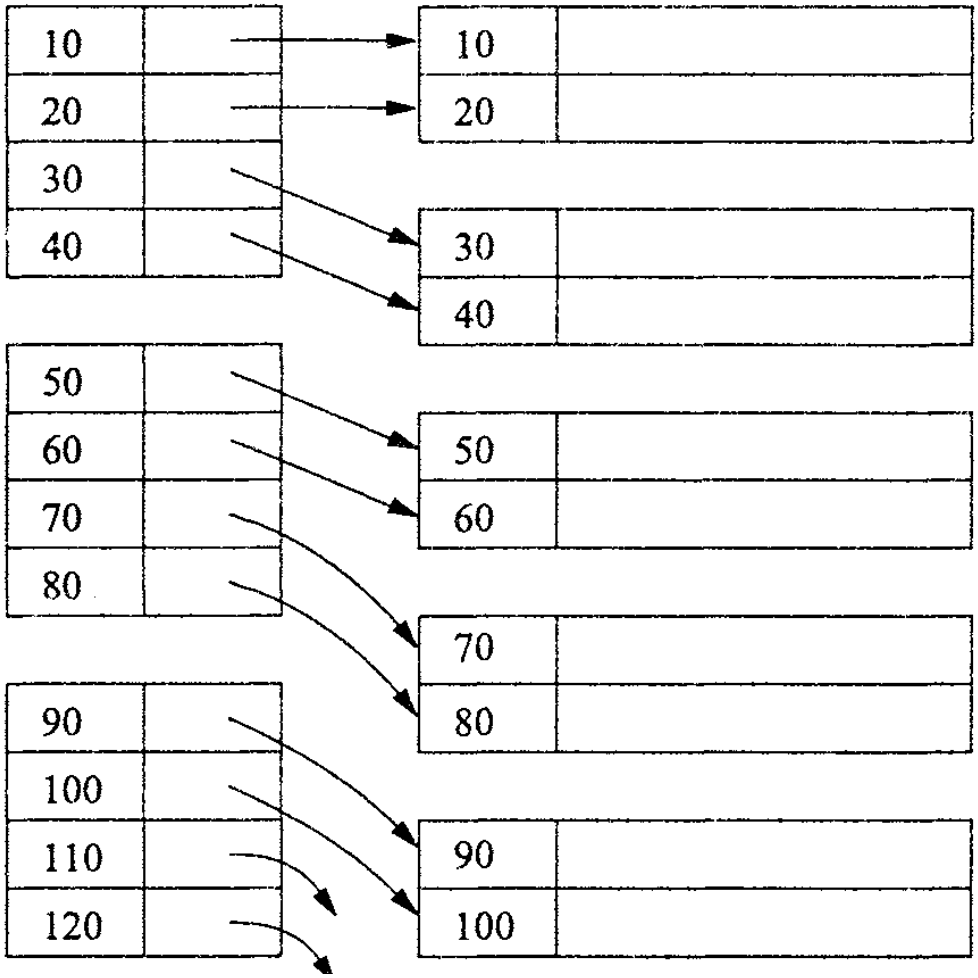


Рисунок 4 - Щільний індекс (ліворуч) для таблиці (праворуч). Для спрощення малюнку вважаємо таблицю відсортованою, хоча взагалі це не так.

Якщо щільний індекс повністю завантажується у пам’ять, можна знайти запис у таблиці з заданим значенням ключа K за допомогою лише одної операції дискового вводу-виводу. У цьому разі система продивляється блоки індексу в пошуку значення K ключа, а потім, коли запис індексу з ключем K знайдено, звертається по адресі запису з цим ключем.

Технологія маніпулювання даними.

Алгоритм пошуку.

Пошук в файлі індексу (ФІ) здійснюється за алгоритмом бінарного пошуку, отже для пошуку необхідно $Log_2(Nфзфі)$, де Nфзфі – кількість фізичних записів у ФІ.

Якщо пошук в ФІ вдалий, надається доступ до відповідного фізичного запису (ФЗ) файлу даних (ФД) за посиланням із знайденого запису ФІ.

Алгоритм додавання запису.

В результаті пошуку знаходимо ФЗ ФІ, в якому мав би бути відповідний запис. Якщо в цьому ФЗ є вільна ділянка, то в ній створюється новий запис. Інакше всі ФЗ ФІ від того, що був знайдений до того, де знайдеться вільне місце, переписуються, що забезпечує збереження відсортованості.

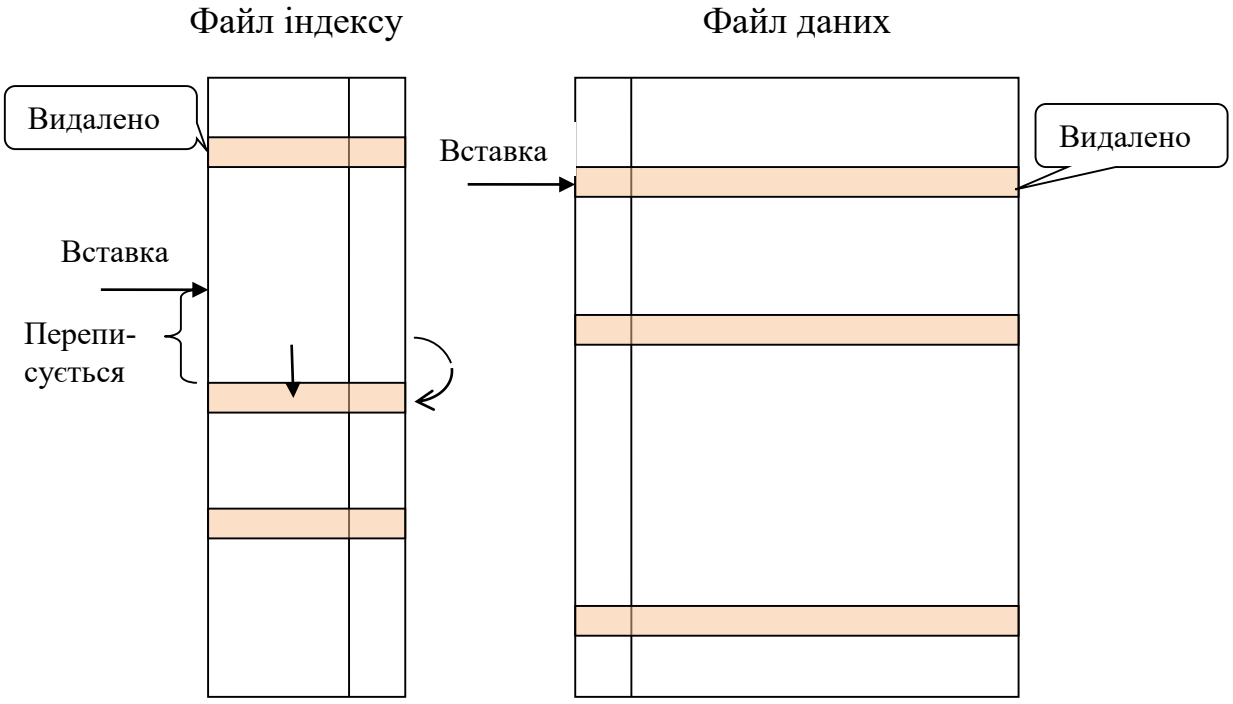


Рисунок 5 – Додавання запису в разі використання щільного індексу

Далі створюємо новий логічний запис (ЛЗ) в ФД. Оскільки ФД не відсортований, то новий ЛЗ в ФД можна створити в будь якому ФЗ ФД, де є вільна ділянка, або в новому ФЗ, який можна отримати через систему обліку вільного простору файлу (СОВПФ). Посилання ФІ дорівнює логічному номеру нового ЛЗ ФД.

Алгоритм видалення запису.

В результаті пошуку знаходимо ФЗ ФІ, в який містить відповідне значення ключа. Відповідна ділянка помічається, як вільна, в ФІ, і аналогічно в ФД відповідна посиланню ділянка помічається, як вільна. Якщо наповненість ФЗ ФІ чи ФД досягла дуже низького рівня, то фізичний запис повертається в систему обліку вільного простору файлу (СОВПФ), а записи заповнених ділянок розподіляються між сусідніми ФЗ. Якщо для ФІ цей процес не викликає ускладнень, то для ФД викликає, тому реорганізацію ФД простіше здійснити шляхом його переписування на нове місце з ущільненням та побудовою нового ФІ.

Алгоритм модифікації запису.

В результаті пошуку знаходимо ФЗ в ФІ, який містить відповідне значення ключа. Відповідна ділянка помічається. За посиланням в ФД робимо доступ та змінюємо відповідну ділянку ФЗ ФД. Тут враховується, що стандартна модифікація не дозволяє зміну значення ключових полів.

Оцінка ефективності.

Пошук вимагає логарифмічної кількості доступів. Найбільш критичною є операція додавання, особливо якщо новий запис припадає в верхню частину файлу, то знадобиться в середньому $N_{ФЗ}$ доступів для переписування ФІ.

Затрати на реорганізацію ФІ можливо знизити шляхом підтримки оптимального рівня наповненості записів ФІ. Це не суттєво погіршить швидкість пошуку внаслідок логарифмічної залежності часу пошуку від кількості записів, тоді як для додавання ця залежність лінійна.

Сфера застосування.

Головним обмеженням є погіршення швидкості маніпулювання при зростанні кількості записів файлу понад 10^5. Другим обмеженням є проблема погіршення якості через

необхідність підтримки великої кількості ФІ, а саме потрібно k! ФІ для підтримки будь-яких пошуків та порядків по ключових полях, де k – кількість ключових полів.

Розріджений індекс

Якщо щільний індекс є завеликим, інколи доцільно використання розрідженого індексу, який менше щільного індексу за рахунок імовірного збільшення проміжку часу, необхідного для пошуку запису у блоці по заданому значенню ключа.

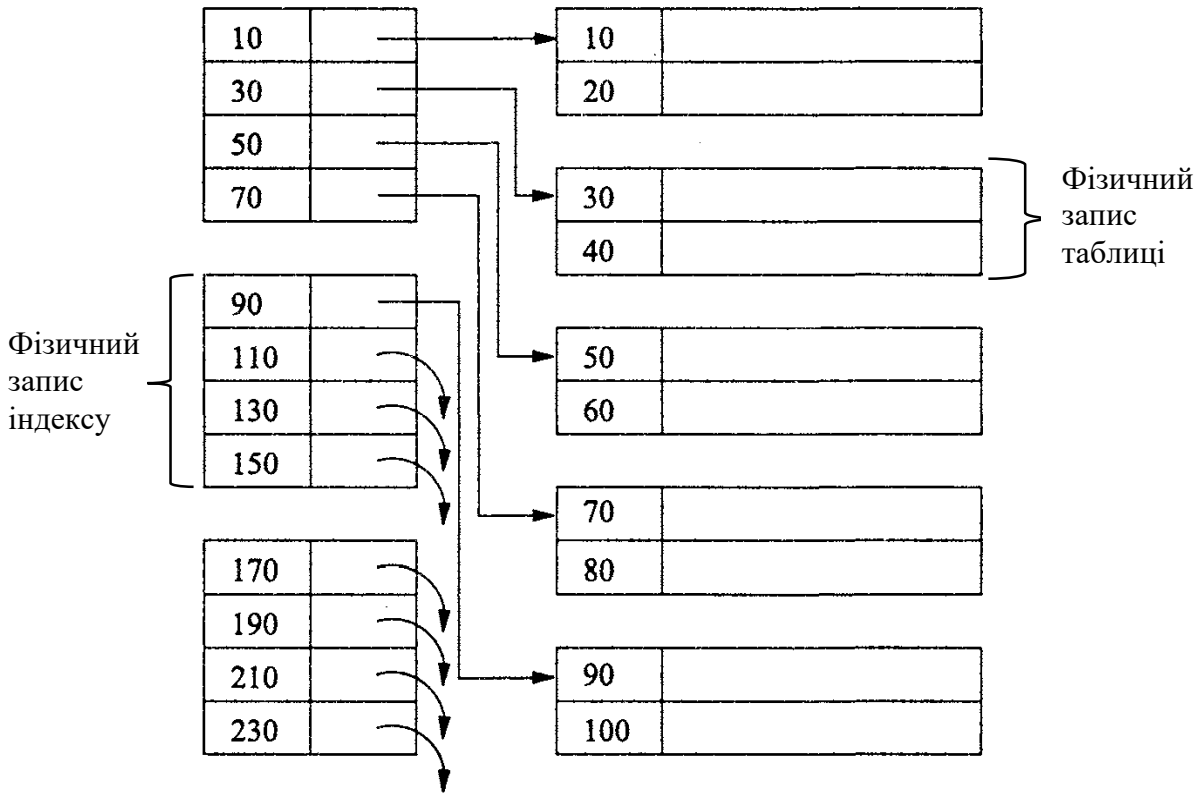


Рисунок 6 - Розріджений індекс, що містить мінімальне значення ключа у блоці. Для такого індексу таблиця має бути згрупованою.

Розріджений індекс містить лише по одному посиланню на кожний блок таблиці даних, а ключами індексу є ключові значення перших записів кожного блоку даних – тобто мінімальне (або максимальне) значення ключа логічних записів, які зберігаються у блоці таблиці даних. Тобто розріджений індекс каже, на якій сторінці файлу БД шукати логічний запис з певним ключем.

Індекс має організацію послідовного відсортованого файлу, в якому кожному фізичному запису файлу (таблиці) даних відповідає один логічний запис. Отже, розмір ФІ значно зменшується.

Файл даних має пряму організацією відносно ерзац-ключа, яким є номер фізичного запису, але накладається вимога, щоб ФД був згрупованим. Це означає, що логічні записи ФД згруповані по діапазонах значень ключів у фізичних записах ФД. Не допускається, щоб для значення ключа X з одного ФЗ в іншому ФЗ знайшлись би записи з ключами X_1 та X_2 такі, що $X_1 < X < X_2$ або $X_2 < X < X_1$. Один блок – один діапазон.

Розріджений індекс набагато менше щільного і тому імовірніше поміститься в оперативну пам'ять. З іншої сторони, щільний індекс дозволяє системі обробляти запити виду „Чи існує запис з ключовим значенням K?", взагалі не звертаючись до диску.

Технологія маніпулювання даними.

Алгоритм пошуку

Щоби за допомогою розрідженого індексу знайти запис даних з ключем K, треба:

1. Знайти запис індексу з найбільшим ключовим значенням, яке менше або дорівнює K. Оскільки індекс відсортовано по ключу, ключове значення K легко знайти, застосувавши модифікований алгоритм бінарного пошуку.
2. Звернувшись до блоку даних за адресою, яка відповідає ключу K в індексі, система повинна знайти в ньому запис з ключем K.

Алгоритм додавання запису.

В результаті пошуку знаходимо ФЗ ФІ, в якому мав би бути відповідний логічний запис. Якщо в цьому ФЗ є вільна ділянка, то в ній створюється новий логічний запис.

Інакше всі ФЗ ФІ від того, що був знайдений до того, де знайдеться вільне місце, переписуються, що забезпечує збереження відсортованості. Після створення ЛЗ ФІ переходимо до створення нового ЛЗ в ФД.

Оскільки ФД обов'язково має бути згрупованим, то новий ЛЗ в ФД можна створити не в будь якому ФЗ ФД, де є вільна ділянка, а лише в тому, на який вказує ФІ. Якщо у відповідному ФЗ ФД немає вільної ділянки, то створюється новий ФЗ ФД, в якому одна ділянка віддається новому запису. Для оптимізації операцій додавання наповненість сусідніх записів вирівнюється, тобто деякі логічні записи ФД з сусідніх ФЗ ФД переміщуються в новий ФЗ ФД з відповідним корегуванням ФІ.

Алгоритм вилучення запису.

В результаті пошуку знаходимо ФЗ ФІ, який містить відповідне значення ключа. Відповідна ділянка помічається як вільна і аналогічно у ФД ділянка, відповідна логічному запису, що видаляється, помічається, як вільна. Якщо наповненість ФЗ ФІ чи ФД досягла критично низького рівня, то запис повертається в СОВПФ, а записи заповнених ділянок розподіляються між логічно сусідніми ФЗ.

Алгоритм модифікації запису.

В результаті пошуку знаходимо посилання на ФЗ в ФІ, який містить відповідне значення ключа. За посиланням у ФД виконуємо доступ та змінюємо відповідну ділянку ФЗ ФД. Тут враховується, що стандартна модифікація не дозволяє зміну значення ключових полів.

Приклад

У файлі індексу позначимо " . " номери блоків, у логічних записів яких ключ більше значення ліворуч і не більше значення праворуч. (1,3) – логічний запис з ключем 1 і неключовим значенням 3.

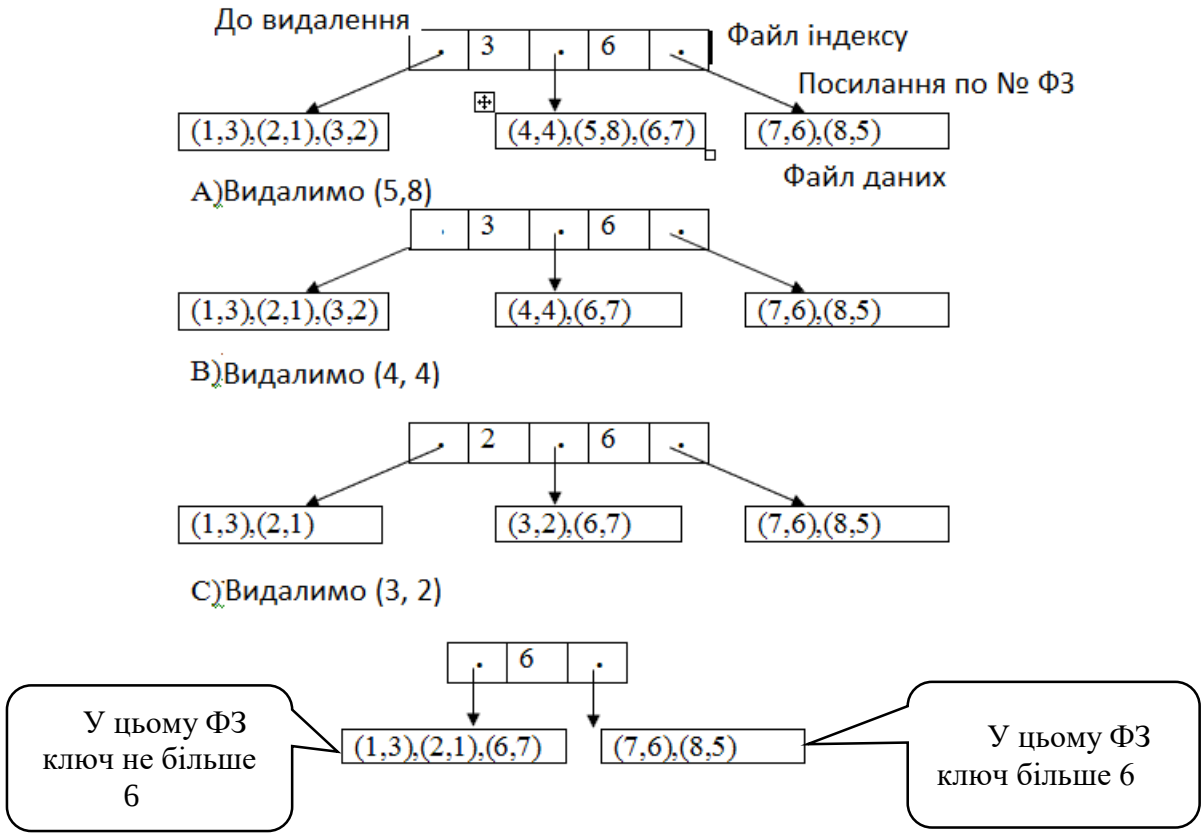


Рисунок 7 – Видалення запису в разі використання розрідженого індексу

У цій структурі треба постійно підтримувати збалансованість, тобто приблизно однакову кількість невидалених логічних записів у кожному фізичному записі таблиці.

Оцінка ефективності

Пошук вимагає логарифмічної кількості доступів, яка додатково зменшується за рахунок зменшення кількості ФЗ ФІ. Найбільш критичною є операція додавання, особливо якщо новий запис припадає в верхню частину файлу, тоді знадобиться в середньому $N_{ФЗ}$ доступів для переписування ФІ.

Затрати на реорганізацію ФІ можливо знизити, шляхом підтримки оптимального рівня наповненості записів ФІ аналогічно щільному індексу.

Сфера застосування.

Організація має дуже широке застосування в областях, де її недоліки не критичні.

Головним обмеженням є погіршення швидкості маніпулювання при зростанні кількості записів файлу понад 10^7.

Другим обмеженням є необхідність підтримки великої кількості ФІ, а саме потрібно k! файлів індексу для підтримки будь-яких пошуків та порядків, де k – кількість ключових полів.

Зрозуміло, що тільки один із файлів індексу може бути розрідженим, а саме той, який відповідає згрупованості ФЗ ФД, а інші мають бути щільними.

Багаторівневий індекс

Навіть якщо для пошуку запису в індексі використовується бінарний пошук, системі залишається чимало операцій дискового вводу-виводу. Для пришвидшення проглядання індексу можна створити додатковий індекс другого рівня – „індекс для індексу”.

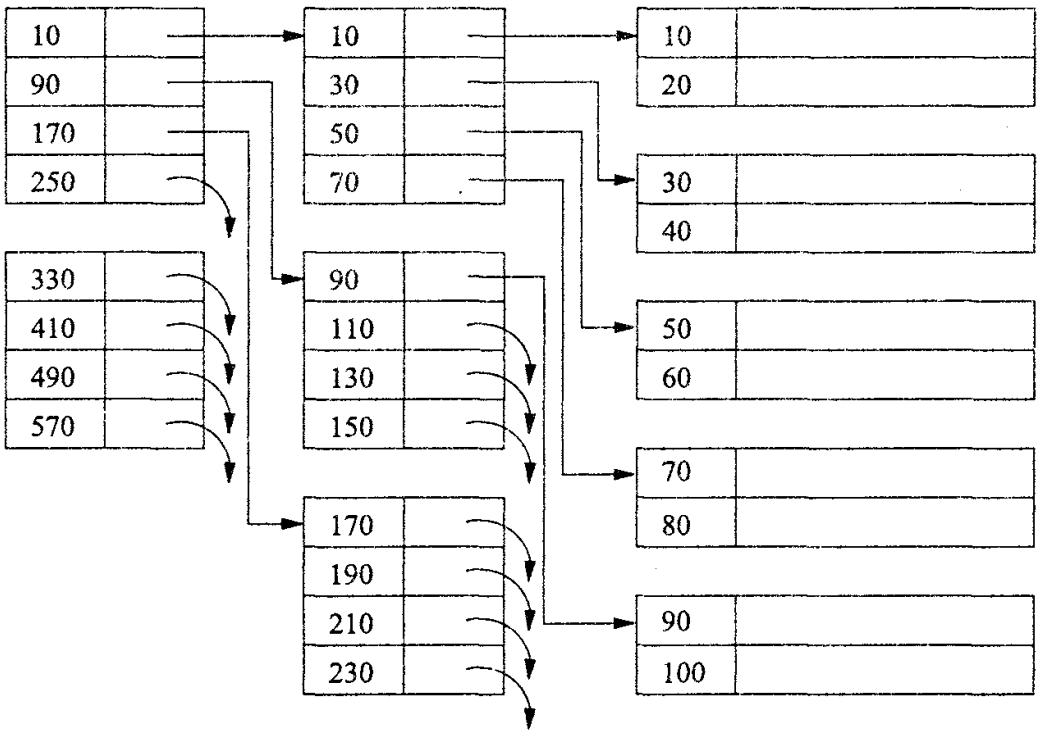


Рисунок 8 - Приклад двохрівневого розрідженого індексу

У цьому прикладі індекс першого рівня є розрідженим, хоча він міг би бути і щільним. Але індекси вищих рівнів завжди мають бути все більш розрідженими.

B-дерева

Хоча багаторівневі індекси вважаються непоганим інструментом пришвидшення обробки запитів, існує більш загальний, гнучкий і ефективний різновид структур, які знаходять найширше застосування в комерційних СУБД. Це сімейство структур, що називається *B-дерева* (B-tree). Їхні основні особливості:

- Автоматична підтримка необхідної кількості рівнів індексації, що відповідає розміру таблиці.
- Ефективне управління розміром вільних областей всередині блоків (рівень заповнення блоків коливається від 50% до 100%).

Структура B-дерева

Блоки індексу типу B-дерево організовані в вигляді деревовидного графу (tree-like graph). B-дерево є збалансованим (balanced) у тому сенсі, що довжини всіх шляхів від кореневої (root) вершини до будь-якої з кінцевих вершин (листків – leaves) однакові. Типове B-дерево містить 3 рівні:

- Кореневу вершину,
- Проміжні вершини,
- Листки,

хоча воно може містити довільну кількість проміжних рівнів.

Кожному B-деревовидному індексу відповідає параметр n, що визначає властивості компоновки блоків B-дерева. Кожний блок має достатньо місця для розміщення n значень ключа пошуку і n+1 посилань. Таким чином, n залежить від розміру дискового блоку і довжини ключа пошуку.

Приклад 1. Якщо дисковий блок має 4096 байт, довжина ключа пошуку – 4 байти, а посилання займає 8 байт, то найбільше ціле, що задовольняє умові
$4n + 8(n+1) \leq 4096$,
дорівнює 340. Це кількість ключів у блоці.

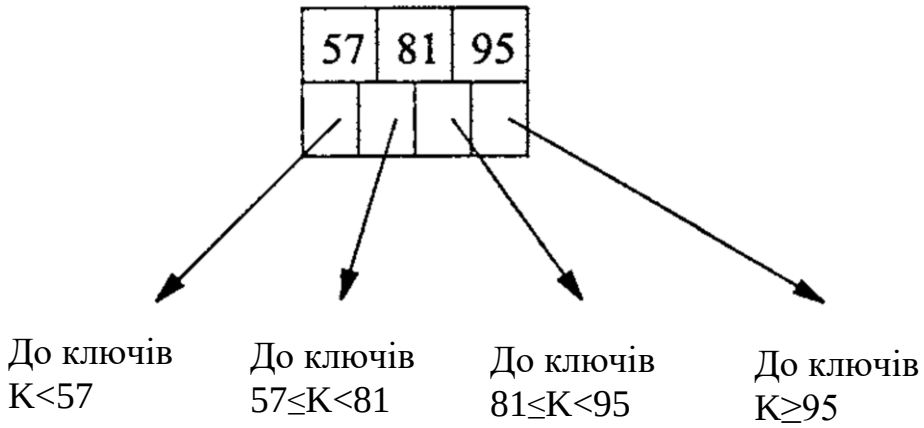


Рисунок 9 - Приклад проміжної вершини B-дерева для n=3

Правила, які регламентують зміст блоків B-дерева:

- Ключові значення, вказані в вершинах-листках, є копіями ключів записів таблиці. Ключі розподілені по вершинах-листках зліва направо в порядку збільшення.
- Коренева вершина містить як мінімум 2 посилання, які адресують блоки нижчого рівня.
- Останнє посилання вершини-листка вказує на наступну вершину-листок праворуч, тобто на блок, що містить наступну порцію ключів. Якщо поточна вершина остання, це посилання є Null.
- Можуть використовуватись не всі посилання у блоці. Якщо використовуються j посилань, їм відповідають j-1 ключів (скажімо, $K_1, K_2, ..., K_{j-1}$). Перше посилання адресує ту частину дерева, яка дозволяє відшукати записи з ключовими значеннями, меншими K_1. Друге посилання посилається на частину дерева зі значеннями ключа не менше K_1 і строго меншими K_2, і т.д. Нарешті, останнє, j-е посилання виводить на гілку дерева, яка має ключові значення $\geq K_{j-1}$.

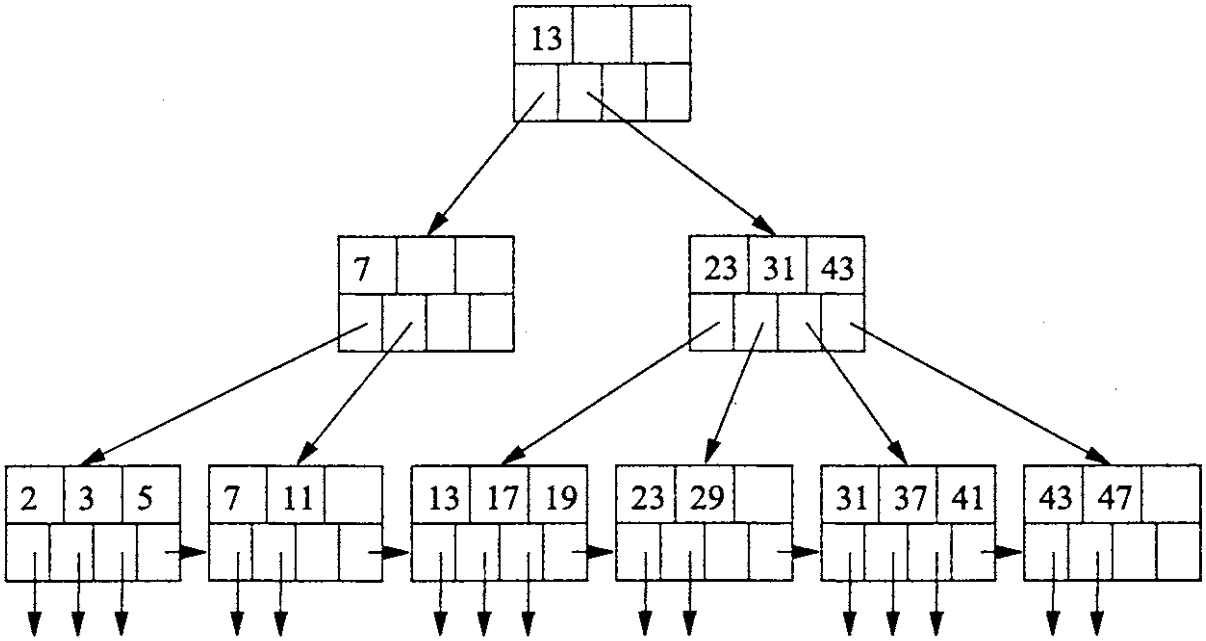


Рисунок 10 - Приклад B-дерева

На малюнку 20 ми маємо повне трьохрівневе B-дерево. Ключі відповідної таблиці – прості числа в інтервалі від 2 до 47. Єдиний ключ кореневої вершини ділить множину ключів на 2 частини: одна доступна через перше посилання, а друга – через друге. Ключі зі значеннями 2-11 (а не 2-12, оскільки ключі – прості числа) – в лівому піддереві відносно кореня, а ключі зі значеннями 13 і вище – у правому піддереві.

Перша ліва дочірня вершина кореня, що містить ключ 7, також має 2 посилання, з них ліве посилається на ключі менше 7, а праве – на ключі більші або рівні 7. Останнє дозволяє звернутися до ключів 7 і 11, а ключ 13 досягається з сусідньої проміжної вершини.

Нарешті, права дочірня вершина кореня містить усі 4 посилання. Перше адресує ключі, менші 23 (а саме – 13, 17 і 19), а останнє – ключі $K \geq 43$.

Пошук у B-деревах

Як приклад розглянемо варіант використання B-дерева:

1. Вершини-листки не містять значень ключів, що повторюються,
2. B-деревовидний індекс є щільним.

Нехай треба знайти запис зі значенням ключа K. Процес пошуку є рекурсивним – він починається з кореня і завершується по досягненню вершини-листка. Алгоритм має вигляд:

1. (Базис) Якщо досягнута певна вершина-листок та і-те значення ключа дорівнює K, і-те посилання адресує потрібний запис і пошук завершено.
2. (Індукція) Якщо досягнута проміжна вершина, що містить ключі $K_1, K_2, ..., K_n$, треба знайти вершину, ключ якої найменший з тих, що більші K. Якщо ключів більше K у вузлі нема, перейти на вершину по посиланню n+1.
3. Продовжити пошук рекурсивно від знайденої вершини, для цього перейти на пункт 1.

Приклад 2. Нехай у B-дереві з попереднього рисунку треба знайти запис зі значенням ключа 17.

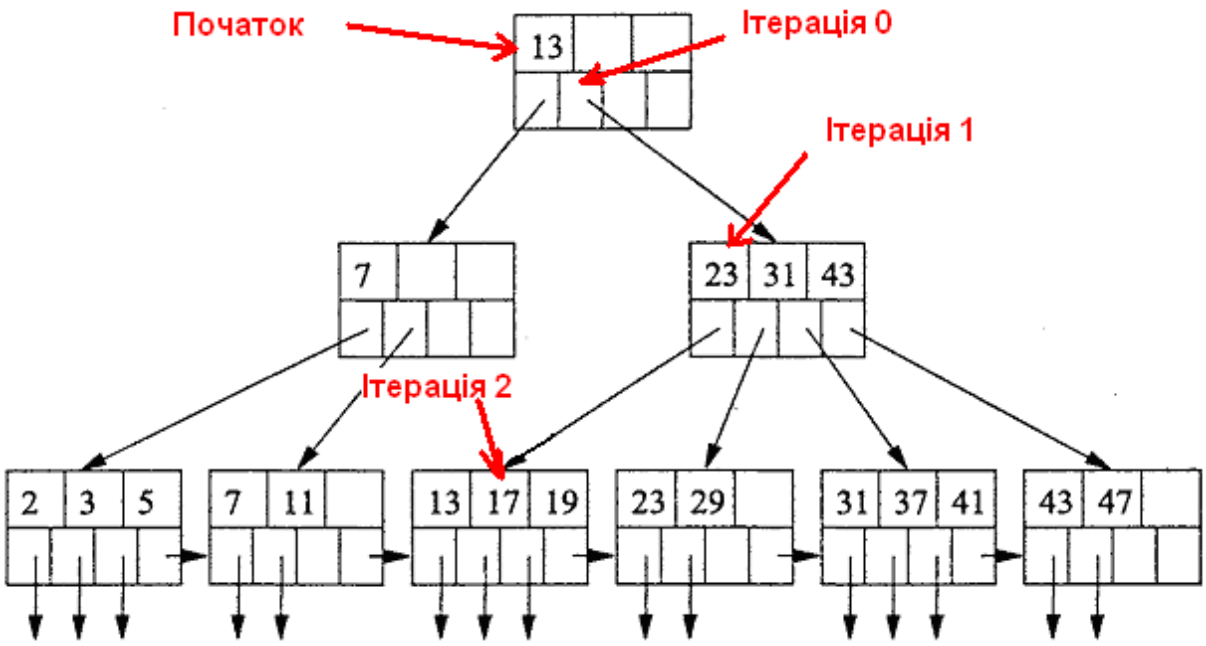


Рисунок 11 - Пошук у B-дереві ключа зі значенням 17

- Починаємо з кореня.
- Ітерація 0. Оскільки корінь не містить ключів більших 17, переходимо по останньому посиланню.
- Ітерація 1. У правому проміжному вузлі найменший ключ більший 17 – це 23. Переходимо по посиланню на третій зліва листок.
- Ітерація 2. Серед значень ключів вузла знаходимо 17. Відповідне посилання є шуканим рішенням.

Вставка елементів у B-дерево

Одна з переваг моделі B-дерева – спрощення операцій вставки нових ключових елементів у відповідь на вставку записів у таблицю. Процес теж є рекурсивним.

- Знайти у відповідному блоці-листку вільне місце, підходяще для розміщення нової пари „ключ-посилання”, та вставити пару, якщо таке місце існує.
- Якщо вільного місця нема, розщепити листок на два листка та розподілити між ними множину ключів і посилань приблизно порівну.
- Розщеплення вершини на одному рівні тягне необхідність вставки нової пари „ключ-посилання” у відповідну вершину наступного вищого рівня. Дії такі самі: якщо місця достатньо, вставити пару; інакше розщепити вершину на дві і продовжити процес вверх по дереву.
- Особлива ситуація: при спробі вставки нової пари в корінь виявляється, що вільного місця нема. Тоді коренева вершина розщеплюється на дві проміжні та створюється нова коренева вершина. (Незалежно від кількості n ключів і посилань у вузлі, кореневій вершині дозволено містити лише 1 ключ і 2 посилання).

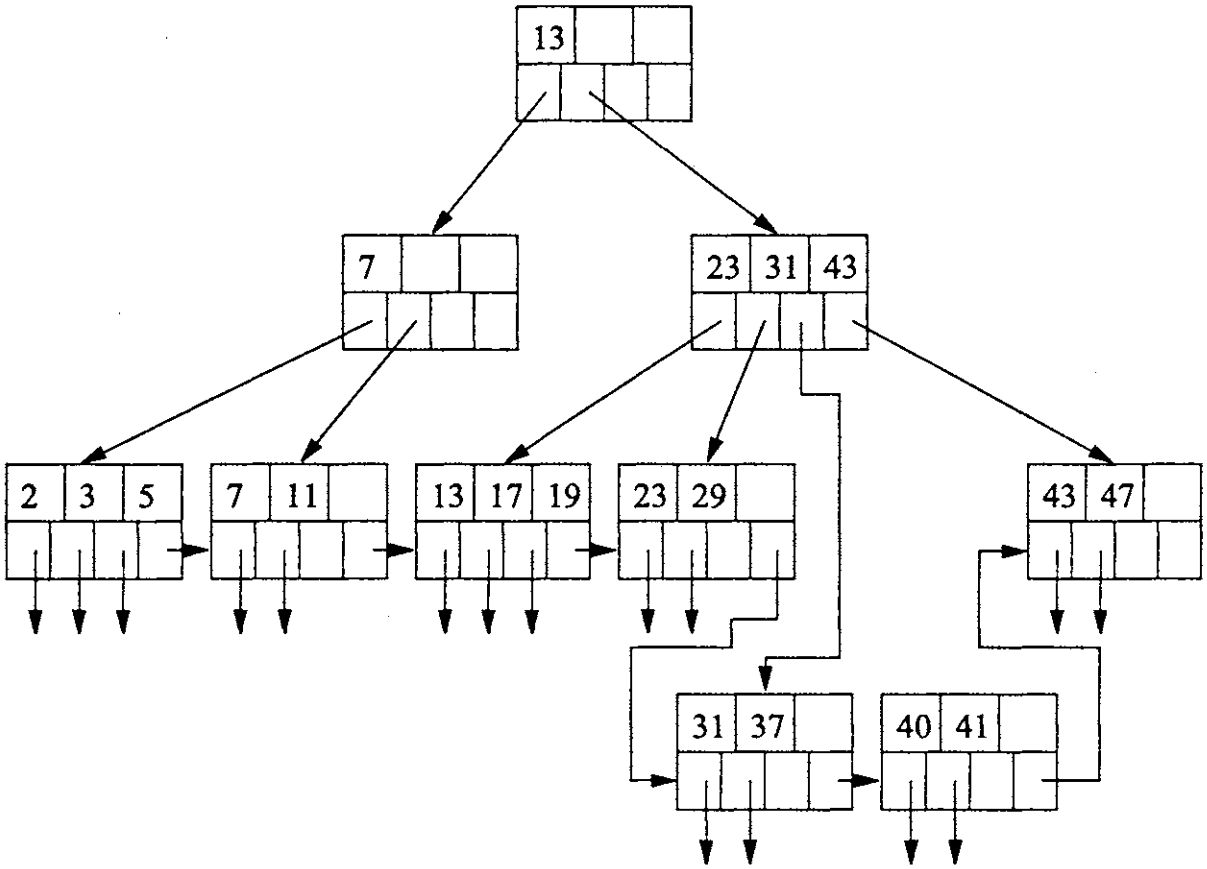


Рисунок 12 - Результат виконання першої фази вставки ключа 40 у B-дерево на рис.7.10: виконане розщеплення вершини (31, 37, 41).

Наступні дії у процесі вставки:

- На нову вершину (40, 41) нема посилання. Треба додати його у вершину (23, 31, 43).
- Оскільки в цій вершині нема місця для нових посилань, треба її розщепити на (23, 31) і (40,43).
- Треба додати посилання на додану вершину (40,43) у корінь.
- Оскільки корінь містить лише 1 ключ, доведеться його розщепити і додати один рівень.

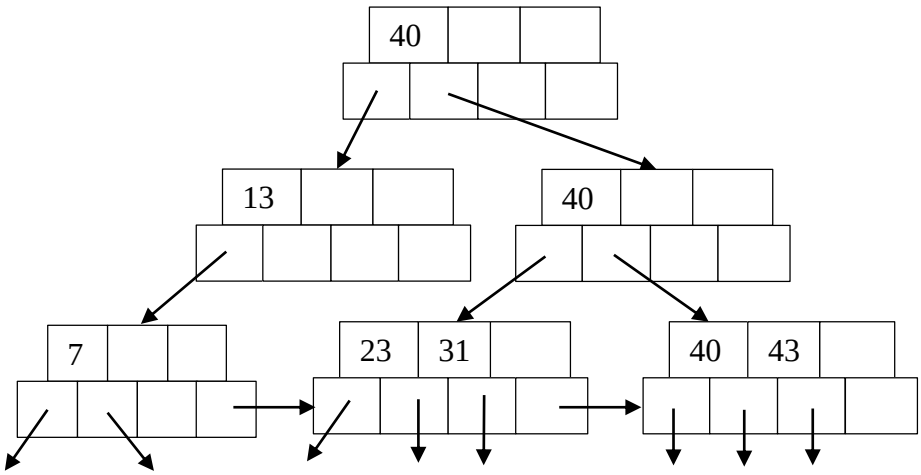


Рисунок 13 – Верхня частина дерева після розщеплення кореня

Видалення елементів з B-дерева

У разі видалення записів з таблиці треба виконати наступні дії:

- Знайти в таблиці запис, який підлягає видаленню.
- Видалити його.
- Видалити ключ цього запису з B-дерева. Відбуваються зворотні дії відносно процесу вставки ключа в B-дерево, серед яких:
  - Якщо кількість пар „ключ-посилання” у вузлі стає меншим припустимого (половини n), виконується спроба перемістити додаткові пари „ключ-посилання” з суміжних вершин, справа або зліва, зберігаючи порядок ключів. Імовірно, в цьому разі треба змінити значення ключів у батьківській вершині.
  - Якщо під час такої спроби виявляється, що суміжна вершина вже має лише мінімально припустиму кількість пар „ключ-посилання”, можливе злиття двох цих суміжних вершин. Це тягне виправлення значень ключів батьківської вершини і видалення в ній одної пари „ключ-посилання”. Якщо ступінь заповнення батьківської вершини залишається задовільною, процес завершується; інакше процес рекурсивним чином повторюється для батьківських вершин, аж до можливого видалення одного рівня дерева.

Оцінка ефективності B-деревовидних індексів

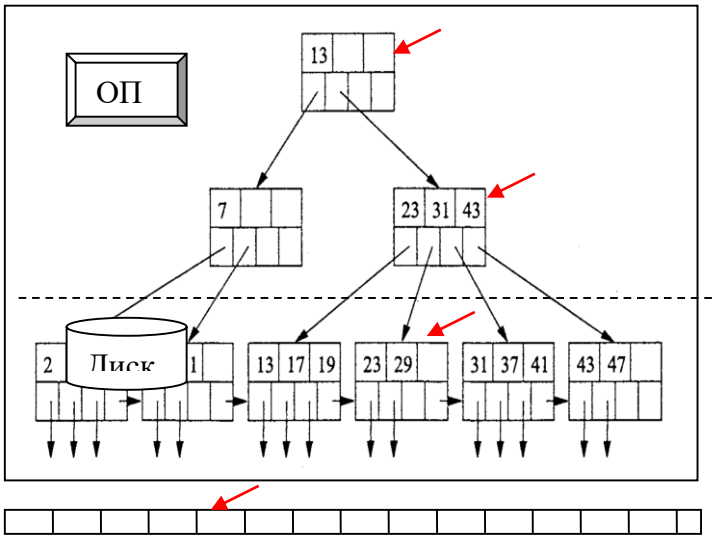
У прикладі 1 визначено, що один блок „звичайного” розміру 4096 байт містить максимально 340 пар „ключ-посилання”. Таким чином, мінімальне їх число – 170, а середнє – (340+170)/2=255. Таким чином,

- Коренева вершина адресує 255 проміжних вершин,
- Проміжні вершини в сукупності адресують $255^2 = 65025$ вершин-листків,
- Листки в сукупності адресують $255^3 = 16{,}6$ млн. записів.

Тобто, якщо кількість записів у таблиці не перевищує 16,6 млн., для індексації такого файлу достатньо 3-рівневого B-дерева.

Реорганізація B-дерева відбувається рідко, і ця дія майже завжди зачіпає лише 2 листка і батьківську вершину, тому витратами на реорганізацію B-дерева можна знехтувати.

Скільки звертань до диску треба, наприклад, для пошуку запису по ключу? Для цього треба пройти все дерево і прочитати сам запис. Максимально звернень до диску: (кількість рівнів дерева) + 1 = 4. Але на практиці це число менше, зважаючи на наступне.



1. Прочитати корінь	Завжди в ОП
2. Прочитати проміжну вершину	Може бути в ОП
3. Прочитати листок	На диску
4. Прочитати запис	На диску

Рисунок 14 - Пошук по B-деревовидному індексу і читання запису з таблиці

- Кореневий блок постійно розміщується в оперативній пам’яті.
- При певних обставинах блоки проміжного рівня також можуть бути розміщені в пам’яті.

Таким чином, необхідне 1-2 звернення до диску для читання запису і 2-3 звернення для оновлення.

Кластерні та некластерні індекси

В Ms SQL Server структурою як кластерного, так і некластерного індексу є B-дерево.

На рівні листків B-дерева кластерний індекс містить записи таблиці, а некластерний – посилання на записи.

Кластерний індекс

Іншими словами, таблиця з кластерним індексом відсортована по ньому. З цього випливає:

- У таблиці може бути лише один кластерний індекс.
- Спочатку для таблиці слід створити кластерний індекс, а потім усі інші – некластерні.
- Індекс первинного ключа може бути як кластерним (рекомендовано), так і некластерним (якщо до його створення вже існував кластерний індекс).
- Для стовпця з обмеженням UNIQUE автоматично створюється некластерний індекс.

Некластерний індекс

У таблиці може і не бути жодного кластерного індексу. В цьому разі таблиця є невпорядкованою і називається *купою*. Таблиця як купа є досить розповсюдженою практикою (див. звертання по індексу до dbf-файлу; також рис. 2). Купа не має проблем з додаванням записів: вони завжди додаються в її кінець.

Некластерний індекс, створений для купи, містить посилання на записи таблиці – row ID, або RID. Структура такого посилання:

- Номер файлу (оскільки база може бути розміщена в кількох файлах .mdf на кількох пристроях),
- Номер сторінки (блоку) в файлі,
- Номер комірки на сторінці.

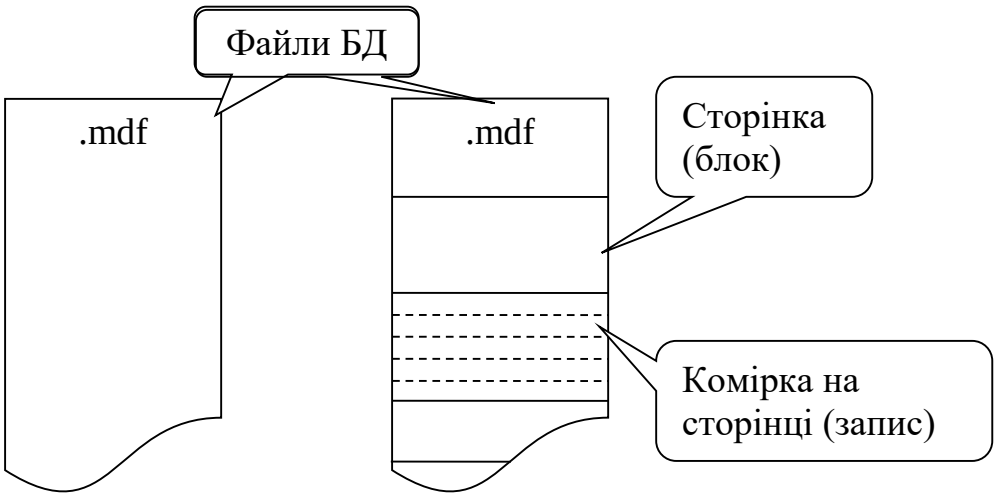


Рисунок 15 - Розміщення таблиці в файлах, сторінках, комірках.

При наявності кластерного індексу сторінки некластерного індексу містять ключі кластерного індексу, а не RID.

Фізична структура бази даних SQL Server

https://msdn.microsoft.com/en-us/library/ms190969.aspx

Фундаментальною одиницею зберігання даних в SQL Server є *сторінка*. Дисковий простір, що виділяється файлу даних (.mdf або .ndf) у базі даних, логічно розділений на сторінки, які нумеруються послідовно від 0 до n. Операції вводу-виводу диска виконуються на рівні сторінки. Тобто, SQL Server читає або записує цілі сторінки даних.

Log-файли .ldf не містять сторінок, вони містять послідовність записів журналу.

Екстенти

Екстент являє собою набір з восьми фізично суміжних сторінок і використовується для ефективного управління сторінками. Всі сторінки зберігаються в екстентах.

SQL Server не виділяє для таблиць з невеликим об'ємом даних цілі екстенти. В SQL Server є два типи екстентів:

- однорідні екстенти, що належать одному об'єкту; тільки об’єкт-власник може використовувати всі вісім сторінок екстенту;
- мішані екстенти, у яких може бути до восьми об'єктів-власників.

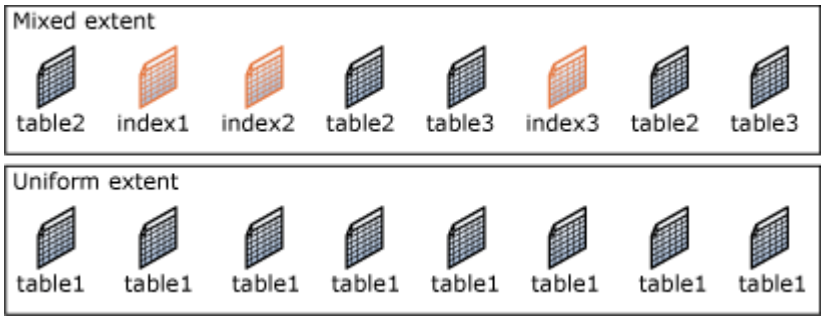


Рисунок 16 - Мішані та однорідні екстенти

У новій таблиці або індексі, як правило, виділяються мішані сторінки. Коли таблиця або індекс зростає і займає більш ніж вісім сторінок, він потім переходить на використання однорідних екстентів. Якщо ви створюєте індекс у існуючій таблиці, який займе вісім сторінок, всі надані індексу екстенти будуть однорідні.

Сторінки

У SQL Server розмір сторінки становить 8 Кб. Кожна сторінка починається з 96-байтного заголовка, який використовується для зберігання системної інформації про сторінку. Ця інформація включає номер сторінки, тип сторінки, кількість вільного місця на сторінці та ідентифікатор блоку розподілу об'єкта, який належить цій сторінці (як-от таблиця або індекс).

Таблиця 1- Типи сторінок, які використовуються у файлах даних бази даних SQL Server

Тип сторінки	Зміст
Data	Рядки даних (кортежі) зі всіма даними, за виключенням типів даних text, ntext, image, nvarchar(max), varchar(max), varbinary(max), і xml
Index	Рядки індексів

Text/Image	-Типи даних великих розмірів: text, ntext, image, nvarchar(max), varchar(max), varbinary(max), і xml -Типи даних змінної довжини в разі коли рядок перевищує 8кб: varchar, nvarchar, varbinary, and sql_variant
Global Allocation Map (GAM), Shared Global Allocation Map (SGAM)	Інформація про те, чи виділено екстенти. Кожна сторінка GAM містить інформацію щодо більш ніж 64000 екстентів, або приблизно 4Гб даних. У GAM кожному екстенту відповідає один біт. Якщо він дорівнює 1, екстент вільний, якщо 0 – зайнятий. В SGAM реєструються екстенти, які в даний момент використовуються як мішані і в яких є як мінімум одна вільна сторінка. Кожна сторінка SGAM містить інформацію щодо більш ніж 64000 екстентів, або приблизно 4Гб даних. У SGAM кожному екстенту відповідає один біт. Якщо він дорівнює 1, екстент мішаний і у нього є вільні сторінки. Якщо біт =0, екстент не використовується як мішаний або, якщо він мішаний, усі його сторінки зайняті.
Page Free Space	Інформація про розподіл сторінок та вільне місце на сторінках
Index Allocation Map	Інформація про екстенти, що використовуються таблицею або індексом для об’єкту-власника.
Bulk Changed Map	Інформація про екстенти, модифіковані масовими (bulk) операціями після останньої команди BACKUP LOG для об’єкту-власника.
Differential Changed Map	Інформація про екстенти, модифіковані після останньої команди BACKUP DATABASE для об’єкту-власника.

Сторінки даних та індексів

Рядки даних поміщаються на сторінку послідовно, починаючи відразу після заголовка. Таблиця зміщень, яка містить зміщення початків кожного рядка відносно початку сторінки, починається з кінця сторінки, і містить один запис для кожного рядка на сторінці. Записи в таблиці зміщень знаходяться в зворотній послідовності відносно послідовності рядків на сторінці.

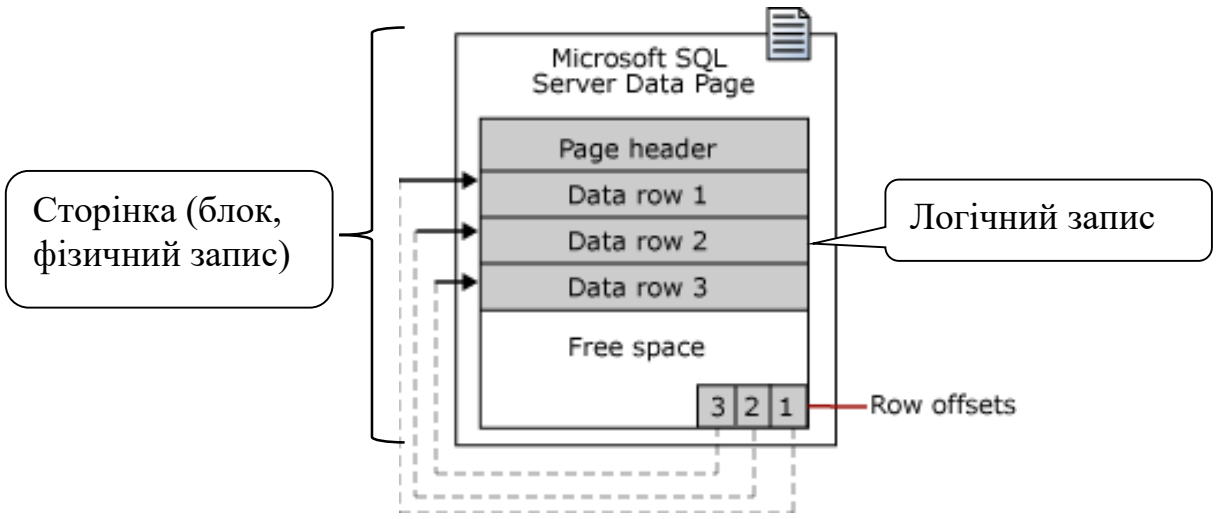


Рисунок 17 - Структура сторінки даних

Організація сторінок даних

Для організації сторінок даних в таблицях SQL Server застосовується один з двох методів: кластерні таблиці або купи.

- Кластерні таблиці. Це таблиці з кластерним індексом. Рядки даних зберігаються в порядку, який визначається ключем кластерного індексу. Індекс реалізовано в вигляді збалансованого дерева (B-дерева). Сторінки на кожному рівні індексу, в тому числі сторінки на рівні листків дерева, пов'язані в двозв’язний список, але перехід між рівнями здійснюється за допомогою ключа.

Рисунок 18 - Організація сторінок даних для таблиць з кластерним індексом

- Купи. Це таблиці без кластерного індексу. Рядки даних зберігаються без певного порядку, і послідовність сторінок даних також не впорядкована. Сторінки даних не організовані в зв’язний список. Натомість у сторінках індексу все впорядковано.