4.3. ІНДЕКСНО-ПОСЛІДОВНІ ФАЙЛИ: МЕТОДИ ОБРОБКИ. БАГАТОРІВНЕВА ІНДЕКСАЦІЯ

Індексно-послідовні файли, це такі файли, коли для основного файлу створюється ще додатковий файл, який містить значення поля (сукупності полів), що індексуються, і вказівника на адресу, де знаходиться запис в основному файлі. Приклади індексації як по одному полю, так і по декільком полям наводилися на лекціях раніше, тому вони тут не розглядаються. Нагадаємо тільки, що при індексації індексний файл завжди упорядковують і пошук по ньому здійснюється описаними вище методами. Оскільки індексний файл набагато менший за основний, то і пошук здійснюється набагато швидше. При цьому треба ще враховувати, що зчитування із зовнішньої пам'яті здійснюється блоками (сторінками), то може так трапитися, що весь індексний файл одразу розміститься в оперативній пам'яті.

Додатково визначимо, що таке щільні і нещільні індекси. Раніше розглядалися тільки щільні індекси, тобто така організація, коли вказівник (індекс) створюється для кожного запису. Як вже говорилося, зчитування із зовнішньої пам'яті здійснюється блоками, тому необов'язково мати вказівник для кожного запису основного файлу, а, наприклад, тільки значення одного поля (мінімальне або максимальне) серед записів, що розміщені на частині файлу, що розміщується на сторінці (блоку). Наприклад, індексний файл має таку структуру:

006	1024	058	2048	124	3072	350	4096	426	5120	567	6144

В цьому файлі наведена така інформація: на блоці с адресою 1024 находяться записи, що починаються з такого, значення індексованого поля дорівнює «006» і більше, а з адреси 2048 – значення «058» і більше і т. д. Основний файл може мати, наприклад, такий вміст:

006	Data1	028	Data2	034	Data3	048	Data4	058	Data5	067	…

Багаторівнева індексація

У тих випадках, коли індексний файл стає великим за обсягом, встають ті ж самі проблеми, що і при використанні послідовних файлів.

В таких випадках індексний файл індексують як послідовний файл. Такий файл по відношенню до основного завжди буде мати нещільні індекси.

На рис. 4.1 наведено схему дворівневої індексації для основного файлу, у якого тільки два (2) записи розміщуються на блоці розміром 4 Мб. Цифрами наведено значення ключів.

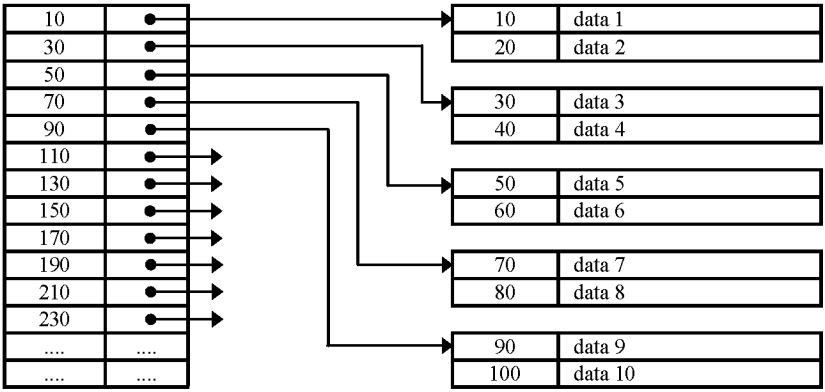

Рис. 4.1. Схема дворівневої індексації

В практиці файлових систем не зустрічалося більш, ніж три рівні індексації. На рис. 4.2 наведено схему трирівневої індексації для того ж файлу.

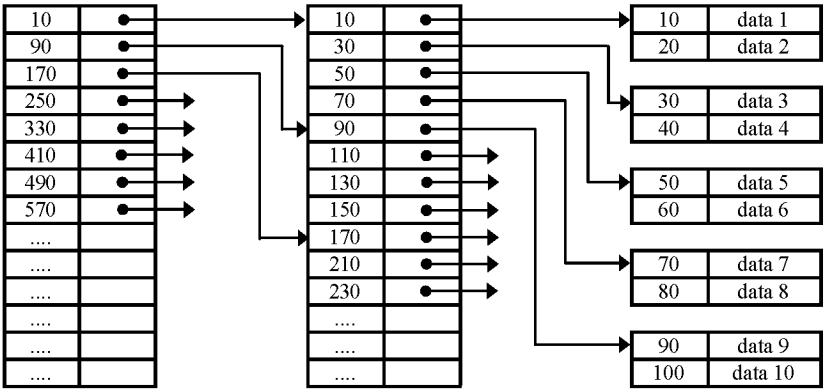

Рис. 4.2. Схема трирівневої індексації

При багаторівневій індексації використовується ієрархічність побудови зовнішньої пам'яті ЕОМ: блоки циліндрів, циліндри, доріжки. Так, у

при організації індексних файлів типу ISAM (Indexed Storage Access Method), що були запропоновані фірмою IBM для мейнфреймів, використовуються така ієрархія індексів:

- головний індекс або індекс областей циліндрів;
- індекс циліндрів;
- індекс доріжок.

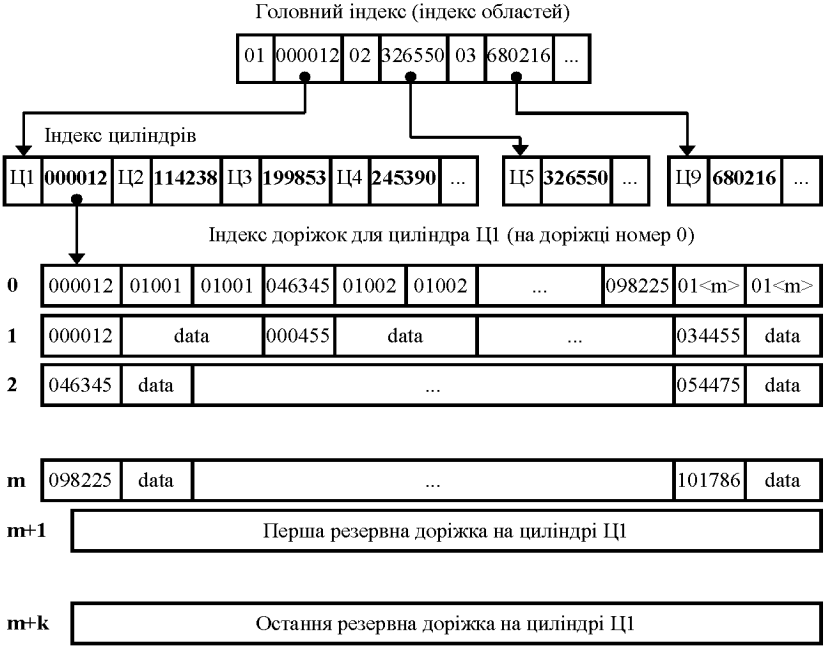

Рис. 4.3. Схема трирівневої індексації в методі *ISAM*

Тут на головному індексі задається мінімальне значення ключа відповідної області циліндрів (на рис. 4.3 зображено область із трьох циліндрів). На індексі першого циліндра задається мінімальне значення ключа на цьому циліндрі з посиланням на його нульову доріжку. Остання використовується як індекс доріжок. Перше значення на нульовій доріжці дорівнює мінімальному значенню ключа на першій доріжці, після якого ідуть самі дані. Треба звернути увагу на те, що на нульовій доріжці після кожного вказівника на доріжку в методі *ISAM* відводиться два поля: в першому задано номер доріжки, на якій знаходиться задане мінімальне значення ключа, а в другому полі – номер доріжки (можливо, з визначенням

іншого циліндра), де знаходиться запис при вставці нового запису, коли він за своїм порядком має стати на цьому місці. Це так звана область переповнення. Тому на рис. 4.3 прийнята така структура коду доріжки: XXNNN, де XX – номер циліндра, а NNN – номер доріжки. При початку роботи з файлом значення обох полів, що показані на рисунку праворуч значення індексу, співпадають, але з часом, коли вставляються записи з новими значеннями ключів або видаляються існуючі записи, спочатку заповнюються резервні доріжки, потім – резервний циліндр. Перехід на резервний циліндр приводить до переміщення головок зчитування, що значно уповільнює час пошуку інформації. В таких випадках адміністратор бази даних має переіндексувати відповідні файли бази даних.

Bitmap-індекси

Основна галузь застосування *Bitmap*-індексів – індексація дуже великих таблиць. Незважаючи на те, що надійна та ефективна реалізація *bitmap*-індексів потребувала зусиль, за своєю концепцією вони дуже прості. В такому індексі кожному значенню ключа, що відрізняється, відповідає бітова карта (*bitmap*), в якій для кожного рядка таблиці виділено 1 біт. Якщо цей біт встановлений в 1, рядок містить дане ключове значення; якщо біт рівний 0, його у рядку немає.

Давайте уявимо можливі набори елементів *bitmap*-індексів за кольором волосся та очей, які можуть бути присутніми в таблиці, що містить відомості про 25 осіб.

Колір волосся	Бітова карта
Black	0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0
Dark Brown	1 1 1 0 0 1 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 1
Light Brown	0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0
Gray	0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0
Blonde	0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0

Оскільки ми визначили в цій невеликій популяції (25 осіб) всього п'ять кольорів волосся, то в нашому індексі присутні п'ять бітових карт. Неможна сказати, що це єдине можливе значення – бітова карта для конкретного значення ключа створюється після того, як дане значення знайдено у таблиці. Тобто, в результаті операції INSERT чи UPDATE може виникнути необхідність побудувати нову бітову карту. Аналогічним чином в результаті операції DELETE чи UPDATE над таблицею бітова карта може стати зайвою.

Колір очей	Бітова карта
Black	1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0
Brown	0 1 0 0 1 0 0 0 1 0 0 1 1 0 1 0 0 1 0 0 1 0 0 0 1
Blue	0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
Green	0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1 1 0 0

Нижче наведено приклад створення bitmap-індексів у середовищі такої популярної СКБД як Oracle, синтаксис речень яких, сподіваємося, буде зрозумілим:

```
CREATE BITMAP INDEX persons_hair_color ON persons
   (hair_color);
CREATE BITMAP INDEX persons_eye_color ON persons
   (eyer_color);
```

Тепер, щоб знайти в нашій таблиці всіх синьооких блондинів, ми передамо в Oracle такий запит:

```
SELECT first_name, last_name, gender
FROM   persons
WHERE  hair_color='BLONDE'
    AND eye_color='BLUE';
```

Для пошуку потрібних нам рядків СКБД повинна просто звернутися до двох відповідних бітових карт та виконати над ними операцію AND. Наприклад:

AND	0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
	0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

В результаті ми швидко знаходимо, що заданим характеристикам відповідає тільки восьмий рядок таблиці. В залежності від того, для яких запитів буде використовуватись ця таблиця, нам, можливо, знадобиться ввести й інші bitmap-індекси за такими ознаками, як стать, вікова група, улюблений музичний жанр і т. ін.

Вже багато років в таких відомих системах керування базами даних (СКБД), як *Oracle, MS SQL Server* та їм подібних при запитах до кількох таблиць виконується «злиття» індексів, які мають структуру *B**-дерева. Це означає, що пошук за рівністю значень у двох стовпцях приводить до пошуку відповідного ключа в кожному індексі та злиттю отриманих списків ROWID. Однак при збільшенні кількості та обсягів з'єднуваних таблиць, кількості полів, які приймають участь у з'єднанні (сполученні) продуктивність даного механізму стала у численних випадках незадовільною. *Bitmap*-індекси надають більш швидкодіючий механізм, який, до того ж, є більш легким для розуміння. Крім того, обробка *bitmap*-індексів може бути розпаралелена, якщо вони секціонуються не по власному ключу.

Bitmap-індекси рекомендується використовувати, головним чином, в тих випадках, коли є набори атрибутів, по яких можна індексувати таблицю (як, наприклад, в наведеній вище таблиці службовців). В ідеальному варіанті ці атрибути не повинні мати велику кількість окремих значень. Припустимо, що очікується багато запитів, в яких буде задано рівність по декількох з таких атрибутах, але при цьому всі можливі комбінації ключових стовпців мають однакову ймовірність появи. Ми не можемо вирішити цю проблему за допомогою традиційних індексів з складеними ключами, оскільки їх знадобилось би занадто багато. Вибравши варіант з *bitmap*-індексом по кожному атрибуту (стовпцю), ми примусимо оптимізатор запитів динамічно визначити, які індекси необхідно проглянути в паралельному режимі та об'єднати шляхом порозрядних логічних операцій.

Інший реальний недолік індексів, які мають структуру *B**-дерева, – їх розмір (мінімальна потреба у просторі на індексований рядок складає 13 байтів). Рядки зі значенням поля, рівним `NULL`, в індекс не входять, тому не можна сказати, що мінімум завжди дорівнює 13 байтам на рядок. Для типового рядка `VARCHAR2(2)` в таблиці великого розміру кількість кілобайтів, необхідна для звичайного індексу по цьому стовпцю, можна приблизно визначити шляхом простого множення очікуваного числа рядків на 14 та ділення результату на 1000 (чи 1024). Теоретично, мінімальна потреба у просторі в bitmap-індексі складає 1 біт на рядок для кожного значення ключа, але *bitmap*-індекси зберігаються у стислому вигляді, тому займають менше місця, ніж виходить при грубому підрахунку. Отримати з бітової карти `ROWID` (номер блока в таблиці та номер рядка у блоці) – задача не з тривіальних, тому механізм, що використовується для цього, потребує додаткових витрат. Тим не менш, в проведених тестах *bitmap*-індекси з обмеженою кількістю значень ключа виявлялись набагато меншими за обсягом, ніж еквівалентні їм індекси на основі *B**-дерева (в деяких випадках вони складали лише 10 % обсягу звичайних індексів). Навіть при побудові *bitmap*-індекса за унікальним ключем (що, взагалі-то, нерозумно) було виявлено, що отриманий в результаті індекс всього на 50 % більше звичайного (при цьому, щоправда, вибірка рядків виконувалась трохи повільніше).

Таким чином, ми представили *bitmap*-індекс у вигляді таблиці з одним рядком для кожного значення ключа. В багатьох відношеннях саме так він і виглядає, і саме так до bitmap-індексу застосовується блокування на рівні рядка. Виключне блокування `(exclusive lock)` розповсюджується на всю бітову карту для даного ключа. Таким чином, висока частота операцій з елементами індексу, в яких використовується

одне й те ж значення ключа, приведе до появи конфліктів блокувань. Оскільки *bitmap*-індекси більше підходять для ключів з низькою вибірковістю (і, відповідно, з більш високою кількістю елементів бітової карти для кожного ключа), конфліктів блокувань практично не уникнути, якщо *bitmap*-індекси використовуються з таблицями певного вигляду. Це таблиці, в яких виконуються операції вставки та видалення, а також таблиці, індексовані значення стовпців в рядках яких відновлюються в багатокористувацькому режимі.

Bitmap-індекси – цінна річ, але краще обмежувати їх застосування стовпцями, які мають відносно мало різних значень і знаходяться в різних таблицях, які обробляються в єдиному потоці (щоб уникнути потенціальних конфліктів блокування). Bitmap-індекси не мають користі при виконанні операцій `BETWEEN`, але являють собою потужний інструмент швидкого доступу до перетинів значень ключа по декількох стовпцях. Такі індекси – практично єдина альтернатива повному скануванню таблиці у випадках, коли сполучення стовпців, яке вказується в запитах, часто змінюється. Однак при використанні `Parallel Query Option` повне сканування таблиці може бути рішенням більш прийнятним. За нашими прогнозами, *bitmap*-індекси будуть швидко застосовуватися у прикладних програмах, пов'язаних із сховищами даних для стовпців розмірності в таблицях фактів, зокрема тому, що ці таблиці рідко оновлюються.