Machine Translated by Google

CÁC CƠ SỞ DỮ LIỆU SQL CHỦ YẾU THUỘC TẤT CẢ

SQL MÀN BIỂU DIỄN GIẢI THÍCH

BẢN TIẾNG ANH

MỌI NHÀ PHÁT TRIỂN CẦN BIẾT VỀ HIỆU SUẤT SQL

TIẾP THỊ CHIẾN THẮNG

Machine Translated by Google

Machine Translated by Google

Thỏa thuận cấp phép

Sách điện tử này chỉ được cấp phép cho mục đích cá nhân của bạn. Sách điện tử này không được bán lại hoặc tặng cho người khác. Nếu bạn muốn chia sẻ cuốn sách này với người khác, vui lòng mua thêm một bản cho mỗi người. Nếu bạn đang đọc cuốn sách này và không mua nó, hoặc nó không được mua chỉ để sử dụng cho bạn, vui lòng quay lại http://SQL-Performance-Explained.com/ và mua bản sao của riêng bạn. Cảm ơn bạn đã tôn trọng công sức của tác giả.

Bản sao này được cấp phép cho: GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Nhà xuất bản:
Markus Winand

Maderspergerstasse 1-3 / 9/11
1160 Wien
ÚC
<office@winand.at>

Bản quyền © 2012 Markus Winand

Đã đăng ký Bản quyền. Không một phần nào của ấn phẩm này có thể được sao chép, lưu trữ hoặc truyền tải dưới bất kỳ hình thức nào hoặc bằng bất kỳ phương tiện nào – điện tử, cơ khí, sao chụp, ghi âm, hoặc bằng cách khác – mà không có sự đồng ý trước của nhà xuất bản.

Nhiều tên được các nhà sản xuất và người bán sử dụng để phân biệt sản phẩm của họ đã được đăng ký nhãn hiệu. Bất cứ nơi nào các ký hiệu như vậy xuất hiện trong cuốn sách này và chúng tôi đã biết về khiếu nại về nhãn hiệu, các tên đó đã được in hoa toàn bộ hoặc viết hoa đầu dòng.

Mặc dù mọi biện pháp phòng ngừa đã được thực hiện trong quá trình soạn thảo cuốn sách này, nhà xuất bản và tác giả không chịu trách nhiệm về những sai sót và thiếu sót, hoặc về những thiệt hại do việc sử dụng thông tin có trong tài liệu này.

Cuốn sách chỉ phản ánh quan điểm của tác giả. Những người đàn ông cung cấp cơ sở dữ liệu đã hẹn giờ đã không hỗ trợ công việc về mặt tài chính cũng như không xác minh nội dung.

DGS - Say xỉn. Graphikservice GmbH - Wien - Áo

Thiết kế bìa:
tomasio.design - Mag. Thomas Weninger - Wien - Áo

Ảnh bìa:
Brian Arnold - Turriff - Vương quốc Anh

Sao chép trình chỉnh sửa:
Nathan Ingvalson - Graz - Áo

2014-08-26

Machine Translated by Google

Giải thích về hiệu suất SQL

Mọi thứ nhà phát triển cần biết về hiệu suất SQL

Markus Winand

Thủ đô Viên, nước Áo

Machine Translated by Google

Nội dung

Lời nói đầu vi

1. Giải phẫu một chỉ mục 1
Các nút lá chỉ mục 2
Cây tìm kiếm (B-Tree) .... 4 Chỉ
số Chậm, Phần I 6

2. Mệnh đề Where .... 9 Nhà
điều hành bình đẳng 9
Chính Các phím ... . 10 Chỉ mục kết
hợp .... 12 Chỉ số Chậm, Phần
II .... 18 Chức
năng 24
Tìm kiếm không phân biệt chữ hoa chữ thường sử dụng UPPER hoặc LOWER ....
24 Chức năng do Người dùng Định nghĩa ....
29 Lập chỉ mục quá mức ....
31 Truy vấn được tham số hóa ....
32 Tìm kiếm phạm vi 39
Lớn hơn, ít hơn và GIỮA .... chỉ.mục.Bộ.lọc..... 39 Lập
LIKE .. .. 45 Hợp nhất
chỉ mục 49
Chỉ mục bộ phận ....
51 NULL trong Cơ sở dữ liệu Oracle ....
53 Chỉ mục NULL ...54.KHÔNG.ĐẦY.ĐỦ.Hạn.. .....
chế .... 56 Mô phỏng các chỉ
số từng phần .... 60 Điều kiện bị xáo
trộn .... 62 Loại
ngày ....
62 Chuỗi số ... . 68 Kết hợp các
cột .... 70 Logic thông
minh 72
Toán 77

Machine Translated by Google

3. Hiệu suất và khả năng mở rộng .... 79 Tác động đến hiệu suất của khối lượng dữ liệu .... 80 Tác động đến hiệu suất của tải hệ thống .... 85 Thời gian đáp ứng và thông lượng 87

4. Hoạt động tham gia .... 91 Vòng lặp lồng nhau .... 92 Tham gia băm .... 101 Sắp xếp Hợp nhất 109

5. Phân cụm dữ liệu .... 111 Dự đoán bộ lọc chỉ mục được sử dụng có chủ ý ..... 112 Chỉ quét chỉ mục .... 116 Bảng được tổ chức theo chỉ mục 122

6. Sắp xếp và phân nhóm .... 129 Thứ tự lập chỉ mục của .... 130 Lập chỉ mục ASC, DESC và NULLS FIRST / CUỐI CÙNG ..... 134 Nhóm lập chỉ mục theo 139

7. Kết quả từng phần .... 143 Truy vấn hàng Top-N .... 143 Phân trang thông qua kết quả ... . 147 Sử dụng các chức năng của cửa sổ để phân trang 156

8. Sửa đổi dữ liệu .... 159 Phụ lục ... . 159 Xóa .... 162 Cập nhật 163

A. Kế hoạch thực hiện .... 165 Cơ sở dữ liệu Oracle .... 166 PostgreSQL .... 172 Máy chủ SQL .... 180 MySQL 188

Mục lục 193

Machine Translated by Google

Nhà phát triển cần lập chỉ mục

Các vấn đề về hiệu suất SQL cũng cũ như chính SQL – một số thậm chí có thể nói rằng SQL vốn đã chậm. Mặc dù điều này có thể đúng trong những ngày đầu của SQL, nhưng nó chắc chắn không còn đúng nữa. Tuy nhiên, các vấn đề về hiệu suất SQL vẫn còn phổ biến. Làm thế nào điều này xảy ra?

Ngôn ngữ SQL có lẽ là ngôn ngữ lập trình thế hệ thứ tư (4GL) thành công nhất. Lợi ích chính của nó là khả năng tách biệt "cái gì" và "như thế nào". Câu lệnh SQL là một mô tả đơn giản những gì cần thiết mà không có hướng dẫn về cách thực hiện nó. Hãy xem xét ví dụ sau:

```
CHỌN date_of_birth
  TỪ nhân viên
 WHERE last_name = 'THẮNG'
```

Truy vấn SQL đọc giống như một câu tiếng Anh giải thích dữ liệu được yêu cầu. Viết câu lệnh SQL thường không yêu cầu bất kỳ kiến thức nào về hoạt động bên trong của cơ sở dữ liệu hoặc hệ thống lưu trữ (chẳng hạn như đĩa, tệp, v.v.). Không cần phải cho cơ sở dữ liệu biết tệp nào sẽ mở hoặc cách tìm các hàng được yêu cầu. Nhiều nhà phát triển có nhiều năm kinh nghiệm về SQL nhưng họ biết rất ít về quá trình xử lý xảy ra trong cơ sở dữ liệu.

Việc tách biệt các mối quan tâm – cái gì cần thiết so với cách lấy nó – hoạt động rất tốt trong SQL, nhưng nó vẫn chưa hoàn hảo. Tính trừu tượng đạt đến giới hạn của nó khi nói đến hiệu suất: tác giả của câu lệnh SQL theo định nghĩa không quan tâm đến cách cơ sở dữ liệu thực thi câu lệnh.
Do đó, tác giả không chịu trách nhiệm về việc thực hiện chậm. Tuy nhiên, kinh nghiệm chứng minh điều ngược lại; tức là, tác giả phải biết một chút về cơ sở dữ liệu để ngăn ngừa các vấn đề về hiệu suất.

Nó chỉ ra rằng điều duy nhất mà các nhà phát triển cần học là cách lập chỉ mục. Trên thực tế, lập chỉ mục cơ sở dữ liệu là một nhiệm vụ phát triển. Đó là bởi vì thông tin quan trọng nhất để lập chỉ mục thích hợp không phải là cấu hình hệ thống lưu trữ hoặc thiết lập phần cứng. Thông tin quan trọng nhất để lập chỉ mục là cách ứng dụng truy vấn dữ liệu. Kiến thức này – về

Machine Translated by Google

đường dẫn truy cập – quản trị viên cơ sở dữ liệu (DBA) hoặc chuyên gia tư vấn bên ngoài không thể truy cập được. Cần khá nhiều thời gian để thu thập thông tin này thông qua thiết kế ngược của ứng dụng: mặt khác, phát triển vẫn có thông tin đó.

Cuốn sách này bao gồm mọi thứ mà nhà phát triển cần biết về chỉ mục – và không có gì khác. Nói chính xác hơn, cuốn sách chỉ đề cập đến loại chỉ mục quan trọng nhất: chỉ mục B-tree.

Chỉ mục B-tree hoạt động gần như giống nhau trong nhiều cơ sở dữ liệu. Cuốn sách chỉ sử dụng thuật ngữ của cơ sở dữ liệu Oracle®, nhưng các nguyên tắc cũng áp dụng cho các cơ sở dữ liệu khác. Ghi chú bên cung cấp thông tin liên quan cho MySQL, PostgreSQL và SQL Server®.

Cấu trúc của cuốn sách được thiết kế riêng cho các nhà phát triển; hầu hết các chương tương ứng với một phần cụ thể của câu lệnh SQL.

CHƯƠNG 1 - Giải phẫu một chỉ mục

Chương đầu tiên là chương duy nhất không trình bày cụ thể về SQL; nó là về cấu trúc cơ bản của một chỉ mục. Sự hiểu biết về cấu trúc chỉ mục là điều cần thiết để theo dõi các chương sau – đừng bỏ qua phần này!

Mặc dù chương này khá ngắn - chỉ khoảng tám trang - sau khi xem qua chương này, bạn sẽ hiểu hiện tượng lập chỉ mục chậm.

CHƯƠNG 2 - Mệnh đề Where

Đây là nơi chúng tôi rút ra tất cả các điểm dừng. Chương này giải thích tất cả các khía cạnh của mệnh đề where , từ tra cứu cột đơn rất đơn giản đến các mệnh đề phức tạp cho các phạm vi và các trường hợp đặc biệt như LIKE.

Chương này tạo nên phần chính của cuốn sách. Một khi bạn học cách sử dụng các kỹ thuật này, bạn sẽ viết SQL nhanh hơn nhiều.

CHƯƠNG 3 - Hiệu suất và khả năng mở rộng Chương này

là một bài phân tích nhỏ về các phép đo hiệu suất và khả năng mở rộng cơ sở dữ liệu. Xem tại sao thêm phần cứng không phải là giải pháp tốt nhất để làm chậm các truy vấn.

CHƯƠNG 4 - Hoạt động tham gia

Quay lại SQL: ở đây bạn sẽ tìm thấy giải thích về cách sử dụng các chỉ mục để thực hiện phép nối bảng nhanh.

Machine Translated by Google

CHƯƠNG 5 - Dữ liệu phân cụm

Bạn đã bao giờ tự hỏi liệu có sự khác biệt nào giữa việc chọn một cột hay tất cả các cột không? Đây là câu trả lời – cùng với một thủ thuật để có được hiệu suất tốt hơn nữa.

CHƯƠNG 6 - Sắp xếp và phân nhóm Thứ tự chẵn

theo và nhóm theo có thể sử dụng các chỉ mục.

CHƯƠNG 7 - Kết quả từng phần Chương

này giải thích cách hưởng lợi từ việc thực thi “pipelined” nếu bạn không cần tập hợp kết quả đầy đủ.

CHƯƠNG 8 - Chèn, Xóa và Cập nhật

Các chỉ mục ảnh hưởng đến hiệu suất ghi như thế nào? Chỉ mục không miễn phí – hãy sử dụng chúng một cách khôn ngoan!

PHỤ LỤC A - Kế hoạch Thực hiện

Hỏi cơ sở dữ liệu cách nó thực thi một câu lệnh.

Machine Translated by Google

Chương 1

Giải phẫu của một chỉ mục

"Một chỉ mục giúp truy vấn nhanh hơn" là lời giải thích cơ bản nhất về một chỉ mục mà tôi từng thấy. Mặc dù nó mô tả rất tốt khía cạnh quan trọng nhất của một chỉ mục, nhưng điều đó - thật không may - không đủ cho cuốn sách này. Chương này mô tả cấu trúc chỉ mục theo cách ít hời hợt hơn nhưng không đi sâu vào chi tiết. Nó cung cấp cái nhìn sâu sắc vừa đủ để người ta hiểu các khía cạnh hiệu suất SQL được thảo luận trong suốt cuốn sách.

Chỉ mục là một cấu trúc riêng biệt trong cơ sở dữ liệu được xây dựng bằng cách sử dụng câu lệnh tạo chỉ mục . Nó yêu cầu không gian đĩa riêng và chứa một bản sao của dữ liệu bảng đã được lập chỉ mục. Điều đó có nghĩa là một chỉ mục là dự phòng thuần túy. Tạo chỉ mục không thay đổi dữ liệu bảng; nó chỉ tạo ra một cấu trúc dữ liệu mới tham chiếu đến bảng. Chỉ mục cơ sở dữ liệu, xét cho cùng, rất giống chỉ mục ở cuối sách: nó chiếm không gian riêng, rất dư thừa và đề cập đến thông tin thực tế được lưu trữ ở một nơi khác.

> Các chỉ mục theo cụm SQL Server
>
> và MySQL (sử dụng InnoDB) có một cái nhìn rộng hơn về "chỉ mục" nghĩa là gì. Chúng đề cập đến các bảng chỉ bao gồm cấu trúc chỉ mục dưới dạng các chỉ mục được nhóm lại. Các bảng này được gọi là Bảng tổ chức theo chỉ mục (IOT) trong cơ sở dữ liệu Oracle.
>
> Chương 5, "Dữ liệu phân cụm", mô tả chúng chi tiết hơn và giải thích những ưu điểm và nhược điểm của chúng.

Tìm kiếm trong chỉ mục cơ sở dữ liệu giống như tìm kiếm trong danh bạ điện thoại in. Khái niệm chính là tất cả các mục được sắp xếp theo một thứ tự được xác định rõ ràng. Việc tìm kiếm dữ liệu trong một tập dữ liệu có thứ tự rất nhanh chóng và dễ dàng vì thứ tự sắp xếp xác định vị trí của từng mục nhập.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Tuy nhiên, một chỉ mục cơ sở dữ liệu phức tạp hơn một thư mục in vì nó trải qua sự thay đổi liên tục. Không thể cập nhật thư mục in cho mọi thay đổi vì lý do đơn giản là không có khoảng trống giữa các mục hiện có để thêm các mục mới. Thư mục đã in sẽ bỏ qua vấn đề này bằng cách chỉ xử lý các bản cập nhật được tích lũy trong lần in tiếp theo.
Cơ sở dữ liệu SQL không thể đợi lâu như vậy. Nó phải xử lý các câu lệnh chèn, xóa và cập nhật ngay lập tức, giữ thứ tự chỉ mục mà không cần di chuyển một lượng lớn dữ liệu.

Cơ sở dữ liệu kết hợp hai cấu trúc dữ liệu để đáp ứng thách thức: danh sách được liên kết kép và cây tìm kiếm. Hai cấu trúc này giải thích hầu hết các đặc điểm hoạt động của cơ sở dữ liệu.

Các nút lá chỉ mục

Mục đích chính của chỉ mục là cung cấp một biểu diễn có thứ tự của dữ liệu được lập chỉ mục. Tuy nhiên, không thể lưu trữ dữ liệu một cách tuần tự vì một câu lệnh chèn sẽ cần phải di chuyển các mục nhập sau để nhường chỗ cho mục nhập mới. Di chuyển một lượng lớn dữ liệu rất tốn thời gian nên câu lệnh chèn sẽ rất chậm. Giải pháp cho vấn đề là thiết lập một trật tự logic độc lập với trật tự vật lý trong bộ nhớ.

Thứ tự hợp lý được thiết lập thông qua một danh sách được liên kết kép. Mỗi nút đều có liên kết đến hai mục nhập lân cận, rất giống một chuỗi. Các nút mới được chèn vào giữa hai nút hiện có bằng cách cập nhật các liên kết của chúng để tham chiếu đến nút mới. Vị trí vật lý của nút mới không quan trọng vì danh sách được liên kết kép duy trì thứ tự logic.

Cấu trúc dữ liệu được gọi là danh sách liên kết kép vì mỗi nút tham chiếu đến nút trước và nút sau. Nó cho phép cơ sở dữ liệu đọc chỉ mục chuyển tiếp hoặc ngược lại nếu cần. Do đó, có thể chèn các mục nhập mới mà không cần di chuyển một lượng lớn dữ liệu — nó chỉ cần thay đổi một số con trỏ.

Danh sách liên kết đôi cũng được sử dụng cho các bộ sưu tập (vùng chứa) trong nhiều ngôn ngữ lập trình.

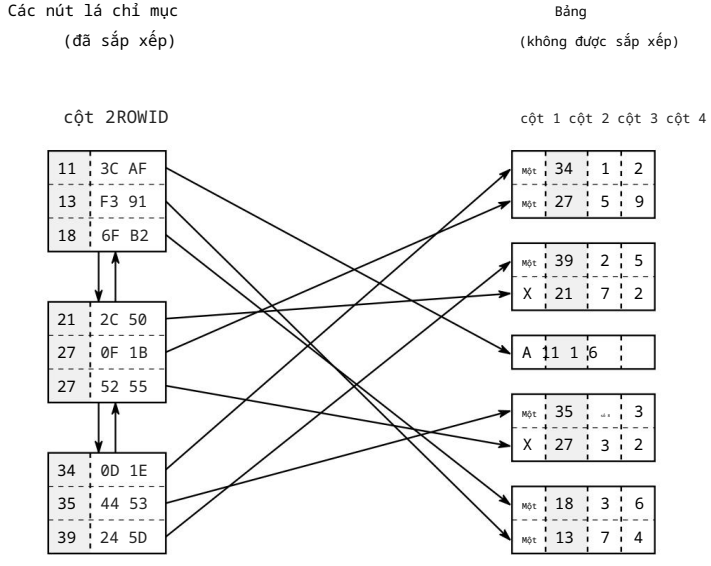
Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Ngôn ngữ lập trình	Tên
Java .NET Framework	java.util.LinkedList
	System.Collections.Generic.LinkedList
C ++	std :: list

Cơ sở dữ liệu sử dụng danh sách được liên kết kép để kết nối cái gọi là nút lá chỉ mục. Mỗi nút lá được lưu trữ trong một khối hoặc trang cơ sở dữ liệu; nghĩa là đơn vị lưu trữ nhỏ nhất của cơ sở dữ liệu. Tất cả các khối chỉ mục đều có cùng kích thước – thường là vài kilobyte. Cơ sở dữ liệu sử dụng không gian trong mỗi khối trong phạm vi có thể và lưu trữ càng nhiều mục nhập chỉ mục trong mỗi khối càng tốt. Điều đó có nghĩa là thứ tự chỉ mục được duy trì ở hai cấp độ khác nhau: các mục nhập chỉ mục trong mỗi nút lá và các nút lá giữa nhau bằng cách sử dụng danh sách được liên kết kép.

Hình 1.1. Các nút lá chỉ mục và dữ liệu bảng tương ứng



Hình 1.1 minh họa các nút lá chỉ mục và kết nối của chúng với dữ liệu bảng. Mỗi mục nhập chỉ mục bao gồm các cột được lập chỉ mục (khóa, cột 2) và tham chiếu đến hàng bảng tương ứng (thông qua ROWID hoặc RID). Không giống như chỉ mục, dữ liệu bảng được lưu trữ trong cấu trúc heap và hoàn toàn không được sắp xếp. Không có mối quan hệ giữa các hàng được lưu trữ trong cùng một khối bảng cũng như không có bất kỳ kết nối nào giữa các khối.

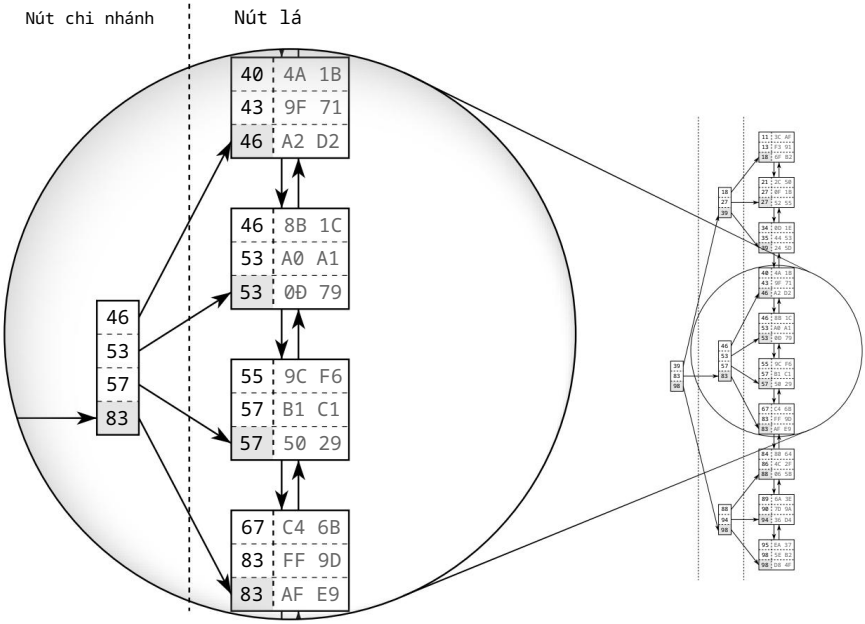
Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Cây Tìm kiếm (B-Tree)

Các nút lá chỉ mục được lưu trữ theo một thứ tự tùy ý — vị trí trên đĩa không tương ứng với vị trí logic theo thứ tự chỉ mục. Nó giống như một danh bạ điện thoại với các trang bị xáo trộn. Nếu bạn tìm kiếm "Smith" nhưng trước tiên mở thư mục tại "Robinson", thì không có nghĩa là Smith theo sau Robinson. Một cơ sở dữ liệu cần có cấu trúc thứ hai để tìm mục nhập trong số các trang được xáo trộn một cách nhanh chóng: cây tìm kiếm cân bằng— nói ngắn gọn: cây B.

Hình 1.2. Cấu trúc cây B



Hình 1.2 cho thấy một chỉ mục ví dụ với 30 mục nhập. Danh sách liên kết kép thiết lập thứ tự logic giữa các nút lá. Các nút gốc và nút nhánh hỗ trợ tìm kiếm nhanh giữa các nút lá.

Hình vẽ làm nổi bật một nút nhánh và các nút lá mà nó đề cập đến. Mỗi mục nhập nút nhánh tương ứng với giá trị lớn nhất trong nút lá tương ứng. Tức là, 46 trong nút lá đầu tiên để mục nhập nút nhánh đầu tiên cũng là 46. Điều này cũng đúng với các nút lá khác để cuối cùng

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

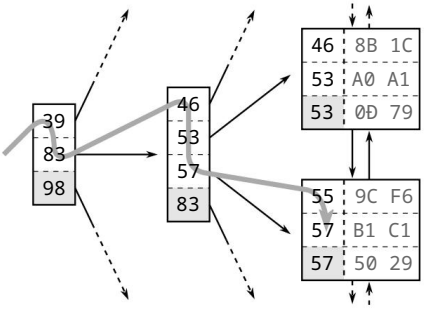
nút nhánh có các giá trị 46, 53, 57 và 83. Theo sơ đồ này, một lớp nhánh được xây dựng cho đến khi tất cả các nút lá được bao phủ bởi một nút nhánh.

Lớp tiếp theo được xây dựng tương tự, nhưng ở trên cùng của mức nút nhánh đầu tiên. Quy trình lặp lại cho đến khi tất cả các khóa vừa với một nút duy nhất, nút gốc. Cấu trúc là một cây tìm kiếm cân bằng vì độ sâu của cây bằng nhau ở mọi vị trí; khoảng cách giữa nút gốc và nút lá là như nhau ở mọi nơi.

> Ghi chú
>
> Cây B là cây cân bằng — không phải cây nhị phân.

Sau khi được tạo, cơ sở dữ liệu sẽ tự động duy trì chỉ mục. Nó áp dụng mọi lần chèn, xóa và cập nhật vào chỉ mục và giữ cho cây ở trạng thái cân bằng, do đó gây ra chi phí bảo trì cho các hoạt động ghi. Chương 8, "Sửa đổi dữ liệu", giải thích điều này chi tiết hơn.

Hình 1.3. B-Tree Traversal



Hình 1.3 cho thấy một phân đoạn chỉ mục để minh họa việc tìm kiếm khóa "57". Quá trình duyệt cây bắt đầu từ nút gốc ở phía bên tay trái. Mỗi mục nhập được xử lý theo thứ tự tăng dần cho đến khi giá trị lớn hơn hoặc bằng (> =) cụm từ tìm kiếm (57). Trong hình, đây là mục nhập 83. Cơ sở dữ liệu theo tham chiếu đến nút nhánh tương ứng và lặp lại quy trình cho đến khi đường truyền cây đến nút lá.

> Quan trọng
>
> Cây B cho phép cơ sở dữ liệu tìm thấy một nút lá một cách nhanh chóng.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Duyệt cây là một hoạt động rất hiệu quả – hiệu quả đến mức tôi coi nó như là sức mạnh đầu tiên của việc lập chỉ mục. Nó hoạt động gần như ngay lập tức – ngay cả trên một tập dữ liệu lớn. Điều đó chủ yếu là do sự cân bằng cây, cho phép truy cập tất cả các phần tử có cùng số bước và thứ hai là do sự tăng trưởng theo lôgarit của độ sâu cây. Điều đó có nghĩa là chiều sâu cây phát triển rất chậm so với số lượng nút lá. Các chỉ mục thế giới thực với hàng triệu bản ghi có độ sâu cây là bốn hoặc năm. Độ sâu sáu cây hiếm khi được nhìn thấy. Hộp "Khả năng mở rộng lôgarit" mô tả điều này chi tiết hơn.

Chỉ số chậm, Phần I

Mặc dù hiệu quả của việc duyệt cây, vẫn có những trường hợp tra cứu chỉ mục không hoạt động nhanh như mong đợi. Sự mâu thuẫn này đã thúc đẩy huyền thoại về "chỉ số thoái hóa" trong một thời gian dài. Huyền thoại tuyên bố xây dựng lại chỉ mục là giải pháp kỳ diệu. Lý do thực sự các câu lệnh tầm thường có thể chậm - ngay cả khi sử dụng một chỉ mục - có thể được giải thích dựa trên cơ sở của các phần trước.

Thành phần đầu tiên để tra cứu chỉ số chậm là chuỗi nút lá. Hãy xem xét lại tìm kiếm "57" trong Hình 1.3. Rõ ràng là có hai mục nhập phù hợp trong chỉ mục. Nói chính xác hơn, có ít nhất hai mục nhập giống nhau: nút lá tiếp theo có thể có thêm mục nhập cho "57". Cơ sở dữ liệu phải đọc nút lá tiếp theo để xem có thêm mục nào phù hợp hay không. Điều đó có nghĩa là một tra cứu chỉ mục không chỉ cần thực hiện duyệt cây mà còn cần tuân theo chuỗi nút lá.

Thành phần thứ hai để tra cứu chỉ mục chậm là truy cập bảng.
Ngay cả một nút lá đơn lẻ cũng có thể chứa nhiều lần truy cập – thường là hàng trăm. Dữ liệu bảng tương ứng thường nằm rải rác trên nhiều khối bảng (xem Hình 1.1, "Các nút chỉ mục và dữ liệu bảng tương ứng"). Điều đó có nghĩa là có một quyền truy cập bảng bổ sung cho mỗi lần truy cập.

Việc tra cứu chỉ mục yêu cầu ba bước: (1) duyệt cây; (2) theo chuỗi nút lá; (3) tìm nạp dữ liệu bảng. Duyệt cây là bước duy nhất có giới hạn trên cho số lượng khối được truy cập – độ sâu chỉ mục. Hai bước còn lại có thể cần truy cập nhiều khối – chúng gây ra việc tra cứu chỉ mục chậm.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Khả năng mở rộng lôgarit

Trong toán học, logarit của một số với một cơ số nhất định là lũy thừa hoặc số mũ mà cơ số đó phải được nâng lên để tạo ra số [Wikipedia1].

Trong cây tìm kiếm, cơ sở tương ứng với số mục nhập trên mỗi nút nhánh và số mũ đối với độ sâu của cây. Chỉ mục ví dụ trong Hình 1.2 chứa tối đa bốn mục nhập trên mỗi nút và có độ sâu cây là ba. Điều đó có nghĩa là chỉ số có thể chứa tới 64 (4^3) mục. Nếu nó phát triển thêm một cấp, nó đã có thể chứa 256 mục nhập (4^4). Mỗi lần (4 cấp được thêm vào, số mục nhập chỉ mục tối đa tăng gấp bốn lần. Lôgarit đảo ngược chức năng này. Do đó, độ sâu của cây là log4 (số-mục-chỉ-mục).

Sự tăng trưởng logarit cho phép chỉ mục ví dụ tìm kiếm một triệu bản ghi với mười cấp độ cây, nhưng một chỉ mục trong thế giới thực thậm chí còn hiệu quả hơn. Yếu tố chính ảnh hưởng đến độ sâu của cây, và do đó hiệu suất tra cứu, là số lượng mục nhập trong mỗi nút cây. Con số này tương ứng với – nói một cách toán học – cơ sở của loga rithm. Cơ sở càng cao, cây càng nông, tốc độ truyền tải càng nhanh.

Mục nhập chỉ mục	độ sâu cây
3	64
4	256
5	1,024
6	4.096
7	16.384
[illegible]	65.536
9	262.144
10	1.048.576

Cơ sở dữ liệu khai thác khái niệm này ở mức tối đa và đặt càng nhiều mục nhập vào mỗi nút càng tốt - thường là hàng trăm. Điều đó có nghĩa là mỗi cấp chỉ mục mới hỗ trợ nhiều mục nhập hơn hàng trăm lần.

[1] http://en.wikipedia.org/wiki/Logarithm

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Nguồn gốc của huyền thoại "chỉ số chậm" là sự tin tưởng sai lầm rằng việc tra cứu chỉ mục chỉ đi ngang qua cây, do đó có ý kiến cho rằng chỉ số chậm phải do cây "bị hỏng" hoặc "không cân bằng" gây ra. Sự thật là bạn thực sự có thể hỏi hầu hết các cơ sở dữ liệu cách họ sử dụng một chỉ mục. Cơ sở dữ liệu Oracle khá dài dòng về mặt này và có ba hoạt động riêng biệt mô tả một tra cứu chỉ mục cơ bản:

QUÉT DUY NHẤT INDEX QUÉT

DUY NHẤT INDEX chỉ thực hiện duyệt cây. Cơ sở dữ liệu Oracle sử dụng thao tác này nếu một ràng buộc duy nhất đảm bảo rằng các tiêu chí tìm kiếm sẽ không khớp với nhiều hơn một mục nhập.

QUÉT NGUỒN CHỈ SỐ

QUÉT NGUỒN CHỈ SỐ thực hiện duyệt cây và đi theo chuỗi nút lá để tìm tất cả các mục nhập phù hợp. Đây là hoạt động dự phòng nếu nhiều mục nhập có thể phù hợp với tiêu chí tìm kiếm.

TRUY CẬP BẢNG THEO CHỈ SỐ ROWID

Thao tác TABLE ACCESS BY INDEX ROWID truy xuất hàng từ bảng. Thao tác này được thực hiện (thường xuyên) cho mọi bản ghi phù hợp từ thao tác quét chỉ mục trước đó.

Điểm quan trọng là QUÉT NGUỒN CHỈ SỐ có thể đọc được một phần lớn chỉ mục. Nếu có thêm một quyền truy cập bảng cho mỗi hàng, truy vấn có thể trở nên chậm ngay cả khi sử dụng chỉ mục.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

chương 2

Mệnh đề Where

Chương trước đã mô tả cấu trúc của các chỉ mục và giải thích nguyên nhân của hoạt động chỉ mục kém. Trong bước tiếp theo, chúng ta học cách phát hiện và tránh những vấn đề này trong câu lệnh SQL. Chúng ta bắt đầu bằng cách xem xét mệnh đề where .

Mệnh đề where xác định điều kiện tìm kiếm của một câu lệnh SQL và do đó nó nằm trong miền chức năng cốt lõi của một chỉ mục: tìm kiếm dữ liệu một cách nhanh chóng.
Mặc dù mệnh đề where có tác động rất lớn đến hiệu suất, nhưng nó thường được diễn giải một cách cẩu thả khiến cơ sở dữ liệu phải quét một phần lớn chỉ mục.
Kết quả: mệnh đề where được viết kém là thành phần đầu tiên của một truy vấn chậm.

Chương này giải thích cách các toán tử khác nhau ảnh hưởng đến việc sử dụng chỉ mục và cách đảm bảo rằng một chỉ mục có thể sử dụng được cho càng nhiều truy vấn càng tốt. Phần cuối cùng hiển thị các mô hình chống phổ biến và trình bày các giải pháp thay thế mang lại hiệu suất tốt hơn.

Nhà điều hành bình đẳng

Toán tử bình đẳng vừa là toán tử SQL nhỏ nhất vừa được sử dụng thường xuyên nhất. Các lỗi lập chỉ mục ảnh hưởng đến hiệu suất vẫn rất phổ biến và trong đó các mệnh đề kết hợp nhiều điều kiện đặc biệt dễ bị ảnh hưởng.

Phần này trình bày cách xác minh việc sử dụng chỉ mục và giải thích cách các chỉ mục được nối có thể tối ưu hóa các điều kiện kết hợp. Để hỗ trợ sự hiểu biết, chúng tôi sẽ phân tích một truy vấn chậm để xem tác động thực tế của các nguyên nhân được giải thích trong Chương 1.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Các phím chính

Chúng ta bắt đầu với mệnh đề where đơn giản nhất nhưng phổ biến nhất : khóa chính tra cứu. Đối với các ví dụ trong suốt chương này, chúng tôi sử dụng NHÂN VIÊN bảng được định nghĩa như sau:

```
TẠO nhân viên BẢNG (
    nhân viên_id NUMBER                 CÓ GIÁ TRỊ,
    họ               VARCHAR2 (1000) KHÔNG ĐỦ,
    họ               VARCHAR2 (1000) KHÔNG ĐỦ,
    date_of_birth DATE                  CÓ GIÁ TRỊ,
    phone_number VARCHAR2 (1000) KHÔNG ĐỦ,
    CONSTRAINT staff_pk KEY CHÍNH (worker_id)
);
```

Cơ sở dữ liệu tự động tạo chỉ mục cho khóa chính. Cái đó có nghĩa là có một chỉ mục trên cột EMPLOYEE_ID , mặc dù có không có câu lệnh tạo chỉ mục .

Truy vấn sau sử dụng khóa chính để truy xuất tên của nhân viên:

```
CHỌN first_name, last_name
   TỪ nhân viên
 WHERE worker_id = 123
```

Mệnh đề where không thể khớp với nhiều hàng vì khóa chính ràng buộc đảm bảo tính duy nhất của các giá trị EMPLOYEE_ID . Cơ sở dữ liệu không không cần theo dõi các nút lá chỉ mục — chỉ cần lướt qua chỉ mục là đủ cây. Chúng tôi có thể sử dụng cái gọi là kế hoạch thực thi để xác minh:

```
-----------------------------------------------------------------------------
| Id | Hoạt động                          | Tên              | Hàng | Chi phí |
-----------------------------------------------------------------------------
| 0 | TUYỂN CHỌN BÁO CÁO | | | 1                                          |    |       2 |
      | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | TUYỂN NHÂN VIÊN |                    |    |       2 |
| * 2 | CHỈ SỐ QUÉT DUY NHẤT                | NHÂN VIÊN_PK |                        |    |         |
-----------------------------------------------------------------------------

Thông tin vị ngữ (được xác định bằng id hoạt động):
---------------------------------------------------

    2 - truy cập ("EMPLOYEE_ID" = 123)
```


Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Kế hoạch thực thi Oracle hiển thị một INDEX UNIQUE SCAN — hoạt động chỉ duyệt qua cây chỉ mục. Nó hoàn toàn sử dụng khả năng mở rộng logarit của chỉ mục để tìm mục nhập rất nhanh chóng — hầu như không phụ thuộc vào kích thước bảng.

Mẹo

Kế hoạch thực thi (đôi khi giải thích kế hoạch hoặc kế hoạch truy vấn) cho thấy các bước cơ sở dữ liệu thực hiện để thực hiện một câu lệnh SQL. Phụ lục A trên trang 165 giải thích cách truy xuất và đọc các kế hoạch thực thi với các cơ sở dữ liệu khác.

Sau khi truy cập chỉ mục, cơ sở dữ liệu phải thực hiện thêm một bước nữa để tìm nạp dữ liệu được truy vấn (FIRST_NAME, LAST_NAME) từ bộ lưu trữ bảng: thao tác TABLE ACCESS BY INDEX ROWID . Thao tác này có thể trở thành một nút thắt cổ chai về hiệu suất — như được giải thích trong "Chỉ mục chậm, Phần I" —nhưng không có rủi ro nào như vậy liên quan đến QUÉT DUY NHẤT INDEX. Thao tác này không thể cung cấp nhiều hơn một mục nhập, vì vậy nó không thể kích hoạt nhiều hơn một quyền truy cập bảng. Điều đó có nghĩa là các thành phần của một truy vấn chậm không có trong INDEX UNIQUE SCAN.

Khóa chính không có chỉ mục duy nhất Một khóa chính không

nhất thiết cần có chỉ mục duy nhất — bạn cũng có thể sử dụng chỉ mục không phải là duy nhất. Trong trường hợp đó, cơ sở dữ liệu Oracle không sử dụng INDEX UNIQUE SCAN mà thay vào đó là hoạt động INDEX RANGE SCAN . Tuy nhiên, ràng buộc vẫn duy trì tính duy nhất của các khóa để tra cứu chỉ mục cung cấp nhiều nhất một mục nhập.

Một trong những lý do sử dụng các chỉ mục không phải là duy nhất cho khóa chính là các ràng buộc có thể trì hoãn. Trái ngược với các ràng buộc thông thường, được xác thực trong quá trình thực thi câu lệnh, cơ sở dữ liệu hoãn việc xác nhận các ràng buộc có thể hoãn lại cho đến khi giao dịch được cam kết. Các ràng buộc hoãn lại được yêu cầu để chèn dữ liệu vào các bảng có phụ thuộc vòng tròn.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Các chỉ mục liên kết

Mặc dù cơ sở dữ liệu tạo chỉ mục cho khóa chính tự động, vẫn còn chỗ cho các sàng lọc thủ công nếu khóa bao gồm của nhiều cột. Trong trường hợp đó, cơ sở dữ liệu tạo một chỉ mục trên tất cả các cột khóa chính – cái gọi là chỉ mục nối (còn được gọi là chỉ mục nhiều cột, tổng hợp hoặc kết hợp). Lưu ý rằng thứ tự cột của một chỉ mục nối có tác động lớn đến khả năng sử dụng của nó vì vậy nó phải được chọn cẩn thận.

Để chứng minh, hãy giả sử có một vụ sáp nhập công ty. Các nhân viên của công ty khác được thêm vào bảng NHÂN VIÊN của chúng tôi nên nó trở nên lớn gấp mười lần. Chỉ có một vấn đề: EMPLOYEE_ID là không phải là duy nhất trên cả hai công ty. Chúng tôi cần mở rộng khóa chính bằng cách một số nhận dạng bổ sung – ví dụ: một ID phụ. Do đó, khóa chính mới có hai các cột: EMPLOYEE_ID như trước đây và SUBSIDIARY_ID để thiết lập lại tính độc đáo.

Do đó, chỉ mục cho khóa chính mới được xác định theo cách sau:

```
TẠO CHỈ SỐ DUY NHẤT NHÂN VIÊN_pk
      ON nhân viên (worker_id, công ty con);
```

Một truy vấn cho một nhân viên cụ thể phải đưa toàn bộ khóa chính vào tài khoản – nghĩa là, cột SUBSIDIARY_ID cũng phải được sử dụng:

```
CHỌN first_name, last_name
   TỪ nhân viên
 WHERE worker_id = 123
    AND con_mục = 30;

-------------------------------------------------------------------------------
| Id | Hoạt động                                  | Tên                  | Hàng | Chi phí |
-------------------------------------------------------------------------------
| 0 | TUYỂN CHỌN BÁO CÁO | | | 1                                                  ¹                2 |
      | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | TUYỂN NHÂN VIÊN |                      ¹                2 |
| * 2 | CHỈ SỐ QUÉT DUY NHẤT                       | NHÂN VIÊN_PK |                ¹                ¹
-------------------------------------------------------------------------------


Thông tin vị ngữ (được xác định bằng id hoạt động):
---------------------------------------------------

     2 - quyền truy cập ("EMPLOYEE_ID" = 123 VÀ "SUBSIDIARY_ID" = 30)
```


Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Bất cứ khi nào truy vấn sử dụng khóa chính hoàn chỉnh, cơ sở dữ liệu có thể sử dụng CHỈ SỐ QUÉT DUY NHẤT — bất kể chỉ mục có bao nhiêu cột. Nhưng điều gì sẽ xảy ra khi chỉ sử dụng một trong các cột quan trọng, chẳng hạn khi tìm kiếm tất cả nhân viên của một công ty con?

```
CHỌN first_name, last_name
   TỪ nhân viên
 WHERE công ty con = 20;

-----------------------------------------------------------------------------------------------
| Id | Hoạt động                        | Tên              | Hàng | Chi phí |
-----------------------------------------------------------------------------------------------
| 0 | TUYỂN CHỌN BÁO CÁO | | 106 | 478 | | * 1
        | BẢNG TRUY CẬP ĐẦY ĐỦ | TUYỂN NHÂN VIÊN | 106 | 478 |
-----------------------------------------------------------------------------------------------


Thông tin vị ngữ (được xác định bằng id hoạt động):
-----------------------------------------------------------------------------------------------

    1 - bộ lọc ("SUBSIDIARY_ID" = 20)
```

Kế hoạch thực thi tiết lộ rằng cơ sở dữ liệu không sử dụng chỉ mục. Thay vào đó, nó thực hiện QUÉT BẢNG ĐẦY ĐỦ. Kết quả là cơ sở dữ liệu đọc toàn bộ bảng và đánh giá mọi hàng so với mệnh đề where . Thời gian thực hiện tăng lên theo kích thước bảng: nếu bảng tăng lên gấp mười lần, QUÉT BẢNG ĐẦY ĐỦ sẽ mất thời gian dài gấp mười lần. Sự nguy hiểm của hoạt động này là nó thường đủ nhanh trong một môi trường phát triển nhỏ, nhưng nó lại gây ra các vấn đề nghiêm trọng về hiệu suất trong sản xuất.

Quét toàn bảng

Hoạt động TABLE ACCESS FULL, còn được gọi là quét toàn bộ bảng, có thể là hoạt động hiệu quả nhất trong một số trường hợp, đặc biệt là khi truy xuất một phần lớn của bảng.

Điều này một phần là do chi phí cho chính việc tra cứu chỉ mục, điều này không xảy ra đối với hoạt động ĐẦY ĐỦ TRUY CẬP BẢNG . Điều này chủ yếu là do tra cứu chỉ mục đọc hết khối này đến khối khác vì cơ sở dữ liệu không biết khối nào sẽ đọc tiếp theo cho đến khi khối hiện tại được xử lý. QUÉT BẢNG ĐẦY ĐỦ vẫn phải nhận được toàn bộ bảng để cơ sở dữ liệu có thể đọc các khối lớn hơn cùng một lúc (đọc nhiều khối). Mặc dù cơ sở dữ liệu đọc nhiều dữ liệu hơn, nhưng nó có thể cần thực hiện ít thao tác đọc hơn.

Machine Translated by Google

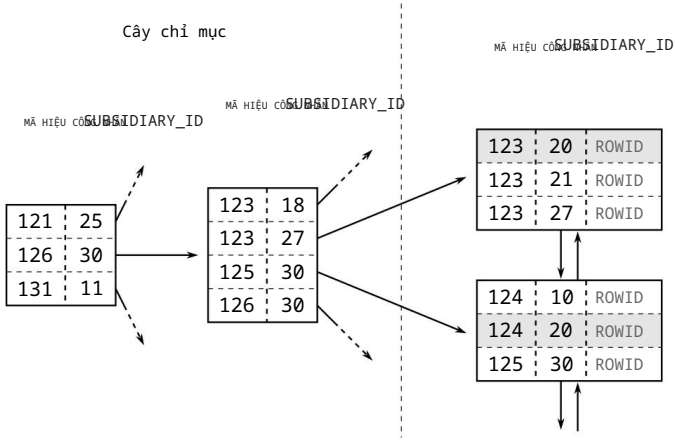
Cơ sở dữ liệu không sử dụng chỉ mục vì nó không thể sử dụng các cột đơn từ một chỉ mục được nối một cách tùy ý. Xem xét kỹ hơn cấu trúc chỉ mục sẽ làm rõ điều này.

Chỉ mục nối chỉ là một chỉ mục cây B giống như bất kỳ chỉ mục nào khác giữ dữ liệu được lập chỉ mục trong một danh sách được sắp xếp. Cơ sở dữ liệu xem xét từng cột theo vị trí của nó trong định nghĩa chỉ mục để sắp xếp các mục nhập chỉ mục. Cột đầu tiên là tiêu chí sắp xếp chính và cột thứ hai chỉ xác định thứ tự nếu hai mục nhập có cùng giá trị trong cột đầu tiên, v.v.

> **Quan trọng**
>
> Chỉ mục được nối là một chỉ mục trên nhiều cột.

Thứ tự của một chỉ mục hai cột do đó giống như thứ tự của một danh bạ điện thoại: trước tiên nó được sắp xếp theo họ, sau đó là tên. Điều đó có nghĩa là chỉ mục hai cột không hỗ trợ tìm kiếm trên cột thứ hai; điều đó giống như tìm kiếm danh bạ điện thoại trước tiên
Tên.

Hình 2.1. Chỉ mục liên kết



Đoạn trích chỉ mục trong Hình 2.1 cho thấy rằng các mục nhập cho công ty con 20 không được lưu trữ bên cạnh nhau. Rõ ràng là không có mục nhập nào có SUBSIDIARY_ID = 20 trong cây, mặc dù chúng tồn tại trong các nút lá.
Do đó cây vô dụng cho truy vấn này.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Mẹo

Hình dung một chỉ mục giúp hiểu được những truy vấn mà chỉ mục hỗ trợ. Bạn có thể truy vấn cơ sở dữ liệu để truy xuất các mục theo thứ tự chỉ mục (cú pháp SQL: 2008, xem trang 144 để biết các giải pháp độc quyền sử dụng LIMIT, TOP hoặc ROWNUM):

```
CHỌN <DANH SÁCH CỘT CHỈ SỐ>
    TỪ <BẢNG>
  ĐẶT HÀNG THEO <DANH SÁCH CỘT CHỈ SỐ>
  CHỈ TÌM 100 ROWS ĐẦU TIÊN;
```

Nếu bạn đặt định nghĩa chỉ mục và tên bảng vào truy vấn, bạn sẽ nhận được một mẫu từ chỉ mục. Tự hỏi bản thân xem các hàng được yêu cầu có được gom lại ở một vị trí trung tâm hay không. Nếu không, cây chỉ mục không thể giúp tìm ra nơi đó.

Tất nhiên, chúng tôi có thể thêm một chỉ mục khác trên SUBSIDIARY_ID để cải thiện tốc độ truy vấn. Tuy nhiên, có một giải pháp tốt hơn – ít nhất là nếu chúng ta cho rằng chỉ tìm kiếm trên EMPLOYEE_ID là không có ý nghĩa.

Chúng ta có thể tận dụng lợi thế của thực tế là cột chỉ mục đầu tiên luôn có thể sử dụng được để tìm kiếm. Một lần nữa, nó giống như một danh bạ điện thoại: bạn không cần biết tên để tìm kiếm theo họ. Mẹo là đảo ngược thứ tự cột chỉ mục để SUBSIDIARY_ID ở vị trí đầu tiên:

```
TẠO CHỈ SỐ DUY NHẤT NHÂN VIÊN_PK
      TRÊN NHÂN VIÊN (SUBSIDIARY_ID, EMPLOYEE_ID);
```

Cả hai cột cùng nhau vẫn là duy nhất nên các truy vấn có khóa chính đầy đủ vẫn có thể sử dụng QUÉT DUY NHẤT INDEX nhưng trình tự các mục nhập chỉ mục hoàn toàn khác nhau. SUBSIDIARY_ID đã trở thành tiêu chí sắp xếp chính.
Điều đó có nghĩa là tất cả các mục nhập cho một công ty con đều nằm trong chỉ mục liên tục để cơ sở dữ liệu có thể sử dụng cây B để tìm vị trí của chúng.

Machine Translated by Google

Quan trọng

Cân nhắc quan trọng nhất khi xác định một chỉ mục là cách chọn thứ tự cột để nó có thể được sử dụng thường xuyên càng tốt.

Kế hoạch thực thi xác nhận rằng cơ sở dữ liệu sử dụng chỉ mục "đảo ngược". Chỉ riêng SUBSIDIARY_ID không còn là duy nhất nữa nên cơ sở dữ liệu phải theo dõi các nút lá để tìm tất cả các mục phù hợp: do đó sử dụng thao tác QUÉT NGUỒN CHỈ SỐ .

```
| Id | Hoạt động                                  | Tên                 | Hàng | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO | 106 | | 1              |                                        75 |
      | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | TUYỂN NHÂN VIÊN | 106 |                                        75 |
| * 2 | QUÉT NGUỒN CHỈ SỐ                          | NHÂN VIÊN_PK | 106 |                                         2 |


Thông tin vị ngữ (được xác định bằng id hoạt động):

    2 - truy cập ("SUBSIDIARY_ID" = 20)
```

Nói chung, một cơ sở dữ liệu có thể sử dụng một chỉ mục được nối khi tìm kiếm với các cột hàng đầu (ngoài cùng bên trái). Chỉ mục có ba cột có thể được sử dụng khi tìm kiếm cột đầu tiên, khi tìm kiếm với hai cột đầu tiên các cột với nhau và khi tìm kiếm bằng cách sử dụng tất cả các cột.

Mặc dù giải pháp hai chỉ mục mang lại hiệu suất chọn lọc rất tốt cũng như, giải pháp chỉ mục đơn được ưu tiên hơn. Nó không chỉ tiết kiệm dung lượng không gian, mà còn là chi phí bảo trì cho chỉ mục thứ hai. Càng ít chỉ mục của một bảng, thì hiệu suất chèn, xóa và cập nhật càng tốt.

Để xác định một chỉ số tối ưu, bạn phải hiểu nhiều hơn là chỉ cách chỉ mục hoạt động – bạn cũng phải biết cách ứng dụng truy vấn dữ liệu. Điều này có nghĩa là bạn phải biết các kết hợp cột xuất hiện trong mệnh đề where .

Do đó, việc xác định một chỉ số tối ưu là rất khó đối với các nhà tư vấn bên ngoài vì họ không có cái nhìn tổng quan về các đường dẫn truy cập của ứng dụng. Các nhà tư vấn thường chỉ có thể xem xét một truy vấn. Họ không khai thác lợi ích bổ sung mà chỉ mục có thể mang lại cho các truy vấn khác. Cơ sở dữ liệu quản trị viên ở vị trí tương tự vì họ có thể biết cơ sở dữ liệu lược đồ nhưng không có cái nhìn sâu sắc về các đường dẫn truy cập.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Nơi duy nhất mà kiến thức cơ sở dữ liệu kỹ thuật đáp ứng kiến thức chức năng của lĩnh vực kinh doanh là bộ phận phát triển. Các nhà phát triển có cảm giác về dữ liệu và biết đường dẫn truy cập. Họ có thể lập chỉ mục đúng cách để có được lợi ích tốt nhất cho ứng dụng tổng thể mà không cần nỗ lực nhiều.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Chỉ số chậm, Phần II

Phần trước đã giải thích cách đạt được các lợi ích bổ sung từ một chỉ mục hiện có bằng cách thay đổi thứ tự cột của nó, nhưng ví dụ này chỉ xem xét hai câu lệnh SQL. Tuy nhiên, việc thay đổi chỉ mục có thể ảnh hưởng đến tất cả các truy vấn trên bảng đã lập chỉ mục. Phần này giải thích cách cơ sở dữ liệu chọn một chỉ mục và giải thích các tác dụng phụ có thể xảy ra khi thay đổi các chỉ mục hiện có.

Chỉ mục EMPLOYEE_PK được thông qua cải thiện hiệu suất của tất cả các truy vấn chỉ tìm kiếm theo công ty con. Tuy nhiên, nó có thể sử dụng được cho tất cả các truy vấn tìm kiếm bằng SUBSIDIARY_ID — bất kể có thêm bất kỳ tiêu chí tìm kiếm nào hay không. Điều đó có nghĩa là chỉ mục trở nên có thể sử dụng được cho các truy vấn đã từng sử dụng một chỉ mục khác với một phần khác của mệnh đề where . Trong trường hợp đó, nếu có sẵn nhiều đường dẫn truy cập, công việc của trình tối ưu hóa là chọn đường dẫn tốt nhất.

Trình tối ưu hoá Truy vấn Trình tối

ưu hoá truy vấn, hoặc công cụ lập kế hoạch truy vấn, là thành phần cơ sở dữ liệu chuyển đổi một câu lệnh SQL thành một kế hoạch thực thi. Quá trình này còn được gọi là biên dịch hoặc phân tích cú pháp. Có hai loại trình tối ưu hóa riêng biệt.

Trình tối ưu hóa dựa trên chi phí (CBO) tạo ra nhiều biến thể kế hoạch thực hiện và tính toán giá trị chi phí cho từng kế hoạch. Việc tính toán chi phí dựa trên các hoạt động đang sử dụng và số lượng hàng ước tính. Cuối cùng, giá trị chi phí đóng vai trò là tiêu chuẩn để chọn kế hoạch thực hiện "tốt nhất".

Trình tối ưu hóa dựa trên quy tắc (RBO) tạo kế hoạch thực thi bằng cách sử dụng bộ quy tắc được mã hóa cứng. Các trình tối ưu hóa dựa trên quy tắc kém linh hoạt hơn và hiếm khi được sử dụng ngày nay.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Thay đổi chỉ số cũng có thể có những tác dụng phụ khó chịu. Trong của chúng tôi ví dụ, đó là ứng dụng danh bạ điện thoại nội bộ đã trở thành rất chậm kể từ khi sáp nhập. Phân tích đầu tiên đã xác định truy vấn sau là nguyên nhân gây ra sự chậm lại:

```
CHỌN first_name, last_name, child_id, phone_number
   TỪ nhân viên
 WHERE last_name = 'THẮNG'
    AND con_mục = 30;
```

Kế hoạch thực hiện là:

Ví dụ 2.1. Kế hoạch thực thi với Chỉ mục khóa chính được sửa đổi

```
-----------------------------------------------------------------------------------------
| Id | Hoạt động                                  | Tên                  | Hàng | Chi phí |
-----------------------------------------------------------------------------------------
| 0 | TUYỂN CHỌN BÁO CÁO | | 30 |
| * 1 | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | TUYỂN NHÂN VIÊN | 30 |          ''        1 1
| * 2 | QUÉT NGUỒN CHỈ SỐ                          | NHÂN VIÊN_PK | 40 | 2 |
-----------------------------------------------------------------------------------------


Thông tin vị ngữ (được xác định bằng id hoạt động):
---------------------------------------------------

   1 - bộ lọc ("LAST_NAME" = 'WINAND')
   2 - truy cập ("SUBSIDIARY_ID" = 30)
```

Kế hoạch thực hiện sử dụng một chỉ mục và có giá trị chi phí tổng thể là 30. Càng xa càng tốt. Tuy nhiên, điều đáng ngờ là nó sử dụng chỉ mục mà chúng tôi chỉ đã thay đổi – đó là lý do đủ để nghi ngờ rằng thay đổi chỉ mục của chúng tôi đã gây ra vấn đề về hiệu suất, đặc biệt là khi mang định nghĩa chỉ mục cũ lưu ý – nó bắt đầu với cột EMPLOYEE_ID không phải là một phần của mệnh đề where ở tất cả. Truy vấn không thể sử dụng chỉ mục đó trước đây.

Để phân tích sâu hơn, sẽ rất tốt nếu bạn so sánh kế hoạch thực hiện trước đó và sau khi thay đổi. Để có được kế hoạch thực thi ban đầu, chúng tôi có thể triển khai lại định nghĩa chỉ mục cũ, tuy nhiên hầu hết các cơ sở dữ liệu đều cung cấp phương pháp đơn giản hơn để ngăn chặn việc sử dụng chỉ mục cho một truy vấn cụ thể. Sau ví dụ sử dụng gợi ý trình tối ưu hóa Oracle cho mục đích đó.

```
SELECT / * + NO_INDEX (NHÂN VIÊN NHÂN VIÊN) * / first_name,
           last_name, công ty con, phone_number
   TỪ nhân viên
 WHERE last_name = 'THẮNG'
    AND con_mục = 30;
```

Machine Translated by Google

Kế hoạch thực thi có lẽ đã được sử dụng trước khi thay đổi chỉ mục hoàn toàn không sử dụng chỉ mục:

```
-----------------------------------------------------------------------------
| Id | Hoạt động                      | Tên             | Hàng | Chi phí |
-----------------------------------------------------------------------------
| 0 | TUYỂN CHỌN BÁO CÁO | | | * 1                                  | 477 | |
        | BẢNG TRUY CẬP ĐẦY ĐỦ | TUYỂN NHÂN VIÊN |              1 1  477 |
-----------------------------------------------------------------------------


Thông tin vị ngữ (được xác định bằng id hoạt động):
---------------------------------------------------
     1 - bộ lọc ("LAST_NAME" = 'WINAND' VÀ "SUBSIDIARY_ID" = 30)
```

Mặc dù TABLE ACCESS FULL phải đọc và xử lý toàn bộ bảng, nó có vẻ nhanh hơn so với việc sử dụng chỉ mục trong trường hợp này. Điều đó đặc biệt bất thường vì truy vấn chỉ khớp với một hàng. Sử dụng chỉ mục để tìm một hàng sẽ nhanh hơn nhiều so với quét toàn bộ bảng, nhưng trong trường hợp này thì không. Chỉ số có vẻ chậm.

Trong những trường hợp như vậy, cách tốt nhất là đi qua từng bước của kế hoạch thực hiện rắc rối. Bước đầu tiên là QUÉT NGUỒN CHỈ SỐ trên chỉ mục EMPLOYEES_PK . Chỉ mục đó không bao gồm cột LAST_NAME — QUÉT NGUỒN CHỈ SỐ chỉ có thể coi là bộ lọc SUBSIDIARY_ID ; cơ sở dữ liệu Oracle hiển thị điều này trong vùng "Thông tin dự đoán" — mục "2" của kế hoạch thực thi. Ở đó bạn có thể thấy các điều kiện được áp dụng cho mỗi hoạt động.

Mẹo

Phụ lục A, "Kế hoạch thực hiện", giải thích cách tìm "Thông tin dự đoán" cho các cơ sở dữ liệu khác.

QUÉT NGUỒN CHỈ SỐ với ID hoạt động 2 (Ví dụ 2.1 trên trang 19) chỉ áp dụng bộ lọc SUBSIDIARY_ID = 30 . Điều đó có nghĩa là nó duyệt cây chỉ mục để tìm mục nhập đầu tiên cho SUBSIDIARY_ID 30. Tiếp theo, nó đi theo chuỗi nút lá để tìm tất cả các mục nhập khác cho công ty con đó. Kết quả của QUÉT NGUỒN CHỈ SỐ là danh sách các ROWID đáp ứng điều kiện SUBSIDIARY_ID : tùy thuộc vào quy mô công ty con, có thể chỉ có một vài hoặc có thể có nhiều hàng trăm.

Bước tiếp theo là thao tác TABLE ACCESS BY INDEX ROWID . Nó sử dụng ROWID từ bước trước để tìm nạp các hàng — tất cả các cột — từ bảng. Khi cột LAST_NAME khả dụng, cơ sở dữ liệu có thể đánh giá phần còn lại của mệnh đề where . Điều đó có nghĩa là cơ sở dữ liệu phải tìm nạp tất cả các hàng cho SUBSIDIARY_ID = 30 trước khi có thể áp dụng bộ lọc LAST_NAME .

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Thời gian phản hồi của câu lệnh không phụ thuộc vào quy mô tập hợp kết quả mà phụ thuộc vào số lượng nhân viên trong công ty con cụ thể. Nếu công ty con chỉ có một vài thành viên, CHỈ SỐ QUÉT sẽ cung cấp hiệu suất tốt hơn. Tuy nhiên, ĐẦY ĐỦ TRUY CẬP BẢNG có thể nhanh hơn đối với một công ty con lớn vì nó có thể đọc các phần lớn từ bảng trong một lần (xem "Quét Toàn Bảng" trên trang 13).

Truy vấn chậm vì tra cứu chỉ mục trả về nhiều ROWID — một ROWID cho mỗi nhân viên của công ty ban đầu — và cơ sở dữ liệu phải tìm nạp từng ROWID. Đó là sự kết hợp hoàn hảo của hai thành phần làm cho một chỉ mục trở nên chậm chạp: cơ sở dữ liệu đọc một phạm vi chỉ mục rộng và phải tìm nạp nhiều hàng riêng lẻ.

Việc chọn kế hoạch thực thi tốt nhất cũng phụ thuộc vào phân phối dữ liệu của bảng, do đó trình tối ưu hóa sử dụng thống kê về nội dung của cơ sở dữ liệu. Trong ví dụ của chúng tôi, một biểu đồ chứa sự phân bổ của nhân viên qua các công ty con được sử dụng. Điều này cho phép trình tối ưu hóa ước tính số hàng được trả về từ tra cứu chỉ mục — kết quả được sử dụng để tính toán chi phí.

Số liệu thống kê

Trình tối ưu hóa dựa trên chi phí sử dụng thống kê về bảng, cột và chỉ mục. Hầu hết các số liệu thống kê được thu thập ở cấp độ cột: số lượng giá trị khác biệt, giá trị nhỏ nhất và lớn nhất (phạm vi dữ liệu), số lần xuất hiện NULL và biểu đồ cột (phân phối dữ liệu). Giá trị thống kê quan trọng nhất cho một bảng là kích thước của nó (theo hàng và khối).

Các thống kê chỉ mục quan trọng nhất là độ sâu của cây, số lượng nút lá, số lượng khóa riêng biệt và hệ số phân cụm (xem Chương 5, "Dữ liệu phân cụm").

Trình tối ưu hóa sử dụng các giá trị này để ước tính tính chọn lọc của các vị từ mệnh đề where .

Machine Translated by Google

Nếu không có sẵn số liệu thống kê — ví dụ: vì chúng đã bị xóa— trình tối ưu hóa sử dụng các giá trị mặc định. Thống kê mặc định của Oracle cơ sở dữ liệu đề xuất một chỉ số nhỏ với độ chọn lọc trung bình. Chúng dẫn đến ước tính rằng INDEX RANGE SCAN sẽ trả về 40 hàng. Kế hoạch thực hiện hiển thị ước tính này trong cột Hàng (một lần nữa, xem Ví dụ 2.1 trên trang 19). Rõ ràng đây là một đánh giá thấp, vì có 1000 nhân viên làm việc cho công ty con này.

Nếu chúng tôi cung cấp số liệu thống kê chính xác, trình tối ưu hóa hoạt động tốt hơn. Các kế hoạch thực thi sau đây cho thấy ước tính mới: 1000 hàng cho QUÉT NGUỒN CHỈ SỐ. Do đó, nó đã tính toán một giá trị chi phí cao hơn cho truy cập bảng tiếp theo.

```
| Id | Hoạt động                                  | Tên                  | Hàng | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO | | 680 |
| * 1 | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | TUYỂN NHÂN VIÊN | 680 |
| * 2 | QUÉT NGUỒN CHỈ SỐ                          | NHÂN VIÊN_PK | 1000 | 4 |


Thông tin vị ngữ (được xác định bằng id hoạt động):

   1 - bộ lọc ("LAST_NAME" = 'WINAND')
   2 - truy cập ("SUBSIDIARY_ID" = 30)
```

Giá trị chi phí của 680 thậm chí còn cao hơn giá trị chi phí để thực hiện lập kế hoạch sử dụng QUÉT BẢNG ĐẦY ĐỦ (477, xem trang 20). Trình tối ưu hóa sẽ do đó tự động thích QUÉT BẢNG ĐẦY ĐỦ.

Ví dụ về chỉ mục chậm này không nên che giấu thực tế rằng lập chỉ mục thích hợp là giải pháp tốt nhất. Tất nhiên tìm kiếm theo họ được hỗ trợ tốt nhất bởi một chỉ mục trên LAST_NAME:

```
TẠO INDEX emp_name ON nhân viên (last_name);
```


Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Sử dụng chỉ mục mới, trình tối ưu hóa sẽ tính toán giá trị chi phí là 3:

Ví dụ 2.2. Kế hoạch thực thi với chỉ mục chuyên dụng

```
| Id | Hoạt động                                    | Tên              | Hàng | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO | | * 1                     |                          1        3 |
        | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | TUYỂN NHÂN VIÊN |                  1        3 |
| * 2 | QUÉT NGUỒN CHỈ SỐ                            | EMP_NAME |               1        1 |


Thông tin vị ngữ (được xác định bằng id hoạt động):

    1 - bộ lọc ("SUBSIDIARY_ID" = 30)
    2 - truy cập ("LAST_NAME" = 'WINAND')
```

Quyền truy cập chỉ mục phân phối – theo ước tính của trình tối ưu hóa – một hàng chỉ. Do đó, cơ sở dữ liệu chỉ phải tìm nạp hàng đó từ bảng: this chắc chắn nhanh hơn QUÉT BẢNG ĐẦY ĐỦ. Một chỉ mục được xác định đúng vẫn là tốt hơn so với bản quét toàn bộ ban đầu.

Hai kế hoạch thực hiện từ Ví dụ 2.1 (trang 19) và Ví dụ 2.2 gần như giống hệt nhau. Cơ sở dữ liệu thực hiện các hoạt động giống nhau và trình tối ưu hóa đã tính toán các giá trị chi phí tương tự, tuy nhiên, kế hoạch thứ hai hoạt động tốt hơn nhiều. Hiệu quả của QUÉT NGUỒN CHỈ SỐ có thể khác nhau trên một phạm vi rộng – đặc biệt khi theo sau là quyền truy cập bảng. Sử dụng một chỉ mục không tự động có nghĩa là một câu lệnh được thực thi theo cách tốt nhất khả thi.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Chức năng

Chỉ mục trên LAST_NAME đã cải thiện đáng kể hiệu suất, nhưng nó yêu cầu bạn tìm kiếm bằng cách sử dụng cùng một chữ hoa (chữ hoa / chữ thường) được lưu trữ trong cơ sở dữ liệu. Phần này giải thích cách loại bỏ hạn chế này mà không làm giảm hiệu suất.

> Lưu ý
>
> MySQL 5.6 không hỗ trợ lập chỉ mục dựa trên hàm như được mô tả bên dưới. Để thay thế, các cột ảo đã được lên kế hoạch cho MySQL 6.0 nhưng chỉ được giới thiệu trong MariaDB 5.2.

Tìm kiếm không phân biệt chữ hoa chữ thường sử dụng UPPER hoặc LOWER

Bỏ qua trường hợp trong mệnh đề where rất đơn giản. Ví dụ: bạn có thể chuyển đổi cả hai mặt của phép so sánh thành tất cả các ký hiệu viết hoa:

```
CHỌN first_name, last_name, phone_number
  TỪ nhân viên
 WHERE UPPER (last_name) = UPPER ('winand');
```

Bất kể cách viết hoa được sử dụng cho cụm từ tìm kiếm hoặc cột LAST_NAME , hàm UPPER làm cho chúng khớp như mong muốn.

> Ghi chú
>
> Một cách khác để đối sánh không phân biệt chữ hoa chữ thường là sử dụng một "đối chiếu" khác. Các đối chiếu mặc định được SQL Server và MySQL sử dụng không phân biệt chữ hoa và chữ thường – chúng không phân biệt chữ hoa chữ thường theo mặc định.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Logic của truy vấn này là hoàn toàn hợp lý nhưng kế hoạch thực thi thì không:

```
| Id | Hoạt động                      | Tên             | Hàng | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO | | | * 1                          10 | 477 | 10 |
        | BẢNG TRUY CẬP ĐẦY ĐỦ | TUYỂN NHÂN VIÊN |            477 |


Thông tin vị ngữ (được xác định bằng id hoạt động):

     1 - bộ lọc (UPPER ("LAST_NAME") = 'WINAND')
```

Đó là một sự trở lại của người bạn cũ của chúng tôi khi quét toàn bộ bảng. Mặc dù có một chỉ mục trên LAST_NAME, nhưng nó không thể sử dụng được – vì tìm kiếm không phải trên LAST_NAME mà trên UPPER (LAST_NAME). Từ quan điểm của cơ sở dữ liệu, đó là một cái gì đó hoàn toàn khác.

Đây là một cái bẫy mà tất cả chúng ta có thể rơi vào. Chúng tôi nhận ra mối quan hệ giữa LAST_NAME và UPPER (LAST_NAME) ngay lập tức và mong muốn cơ sở dữ liệu cũng "nhìn thấy" điều đó. Trong thực tế, chế độ xem của trình tối ưu hóa giống như thế này:

```
CHỌN first_name, last_name, phone_number
   TỪ nhân viên WHERE
  BLACKBOX (...) = 'THẮNG';
```

Chức năng UPPER chỉ là một hộp đen. Các tham số của hàm không liên quan vì không có mối quan hệ chung giữa các tham số của hàm và kết quả.

Mẹo

Thay thế tên chức năng bằng BLACKBOX để hiểu quan điểm của opti mizer.

Đánh giá thời gian biên dịch

Trình tối ưu hóa có thể đánh giá biểu thức ở phía bên phải trong "thời gian biên dịch" vì nó có tất cả các tham số đầu vào. Do đó, kế hoạch thực thi Oracle (phần "Thông tin dự đoán") chỉ hiển thị ký hiệu chữ hoa của cụm từ tìm kiếm. Hành vi này rất giống với một trình biên dịch đánh giá các biểu thức không đổi tại thời điểm biên dịch.

Machine Translated by Google

Để hỗ trợ truy vấn đó, chúng tôi cần một chỉ mục bao gồm cụm từ tìm kiếm thực tế. Điều đó có nghĩa là chúng tôi không cần chỉ mục trên LAST_NAME mà trên UPPER (LAST_NAME):

```
TẠO INDEX emp_up_name
      BẬT nhân viên (LÊN (tên_ cuối));
```

Một chỉ mục có định nghĩa chứa các hàm hoặc biểu thức được gọi là chỉ số dựa trên chức năng (FBI). Thay vì sao chép dữ liệu cột trực tiếp vào chỉ mục, chỉ mục dựa trên chức năng áp dụng chức năng đầu tiên và đặt kết quả vào chỉ mục. Do đó, chỉ mục lưu trữ các tên được viết hoa toàn bộ ký hiệu.

Cơ sở dữ liệu có thể sử dụng chỉ mục dựa trên hàm nếu biểu thức chính xác của định nghĩa chỉ mục xuất hiện trong một câu lệnh SQL – như trong ví dụ trên. Kế hoạch thực hiện xác nhận điều này:

```
| Id | Hoạt động                                | Tên                | Hàng | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO | 100 | |                |                                      41 |
  1   | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | TUYỂN NHÂN VIÊN | 100 |                          41 |
| * 2 | QUÉT NGUỒN CHỈ SỐ                        | EMP_UP_NAME | 40 |                        1 |


Thông tin vị ngữ (được xác định bằng id hoạt động):

   2 - truy cập (UPPER ("LAST_NAME") = 'WINAND')
```

Đây là một QUÉT NGUỒN CHỈ SỐ thông thường như được mô tả trong Chương 1. Cơ sở dữ liệu đi ngang cây B và theo chuỗi nút lá. Không có chuyên dụng hoạt động hoặc từ khóa cho các chỉ mục dựa trên chức năng.

Cảnh báo

Đôi khi các công cụ ORM sử dụng UPPER và LOWER mà không cần trình phát triển hiểu biết. Ví dụ: Hibernate đưa một LOWER ngầm định cho các tìm kiếm không phân biệt chữ hoa chữ thường.

Kế hoạch thực hiện vẫn chưa giống như ở phần trước không có LÊN; ước tính số hàng quá cao. Nó đặc biệt kỳ lạ mà trình tối ưu hóa mong đợi tìm nạp nhiều hàng hơn từ bảng so với INDEX RANGE SCAN phân phối ngay từ đầu. Làm cách nào nó có thể lấy 100 hàng từ bảng nếu quét chỉ mục trước đó chỉ trả về 40 hàng? Câu trả lời là rằng nó không thể. Các ước tính mâu thuẫn như thế này thường chỉ ra các vấn đề với các số liệu thống kê. Trong trường hợp cụ thể này là do cơ sở dữ liệu Oracle

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

không cập nhật thống kê bảng khi tạo chỉ mục mới (xem thêm
"Thống kê Oracle cho các Chỉ mục Dựa trên Chức năng" trên trang 28).

Sau khi cập nhật thống kê, trình tối ưu hóa sẽ tính toán chính xác hơn ước tính:

```
| Id | Hoạt động                                  | Tên                | Hàng | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO | |                      |                                  1 |       3 |
   1   | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | TUYỂN NHÂN VIÊN |                  1 |       3 |
| * 2 | QUÉT NGUỒN CHỈ SỐ                          | EMP_UP_NAME |                1 |       1 |


Thông tin vị ngữ (được xác định bằng id hoạt động):

   2 - truy cập (UPPER ("LAST_NAME") = 'WINAND')
```

> **Ghi chú**
>
> Cái gọi là "thống kê mở rộng" trên các biểu thức và nhóm cột đã được giới thiệu với bản phát hành 11g của Oracle .

Mặc dù các thống kê cập nhật không cải thiện hiệu suất thực thi trong trường hợp này – dù sao thì chỉ mục cũng được sử dụng đúng cách – luôn luôn là một ý kiến hay kiểm tra ước tính của trình tối ưu hóa. Số lượng hàng được xử lý cho mỗi hoạt động (ước tính số lượng) là một con số đặc biệt quan trọng là cũng được hiển thị trong SQL Server và kế hoạch thực thi PostgreSQL.

> **Mẹo**
>
> Phụ lục A, "Kế hoạch thực hiện", mô tả ước tính số lượng hàng trong Kế hoạch thực thi SQL Server và PostgreSQL.

SQL Server không hỗ trợ các chỉ mục dựa trên chức năng như được mô tả nhưng nó có cung cấp các cột được tính toán có thể được sử dụng thay thế. Để tận dụng điều này, trước tiên bạn phải thêm một cột được tính toán vào bảng có thể được lập chỉ mục sau đó:

```
ALTER TABLE nhân viên THÊM last_name_up AS UPPER (last_name);
TẠO INDEX emp_up_name ON nhân viên (last_name_up);
```

SQL Server có thể sử dụng chỉ mục này bất cứ khi nào biểu thức được lập chỉ mục xuất hiện trong tuyên bố. Bạn không cần phải viết lại truy vấn của mình để sử dụng cột tính toán.

Machine Translated by Google

Thống kê Oracle cho các chỉ mục dựa trên chức năng

Cơ sở dữ liệu Oracle duy trì thông tin về số lượng giá trị cột riêng biệt như một phần của thống kê bảng. Những số liệu này được sử dụng lại nếu một cột là một phần của nhiều chỉ mục.

Thống kê cho một chỉ mục dựa trên chức năng (FBI) cũng được giữ ở cấp bảng dưới dạng cột ảo. Mặc dù cơ sở dữ liệu Oracle thu thập thống kê chỉ mục cho các chỉ mục mới tự động (kể từ khi phát hành 10g), nó không cập nhật thống kê bảng. Vì lý do này, tài liệu Oracle khuyên bạn nên cập nhật thống kê bảng sau khi tạo chỉ mục dựa trên hàm:

> Sau khi tạo chỉ mục dựa trên chức năng, hãy thu thập thống kê trên cả chỉ mục và bảng cơ sở của nó bằng cách sử dụng gói DBMS_STATS . Số liệu thống kê như vậy sẽ cho phép Cơ sở dữ liệu Oracle quyết định chính xác thời điểm sử dụng chỉ mục.
>
> —Tham chiếu ngôn ngữ SQL của cơ sở dữ liệu thần kỳ

Khuyến nghị cá nhân của tôi thậm chí còn đi xa hơn: sau mỗi lần thay đổi chỉ mục, hãy cập nhật số liệu thống kê cho bảng cơ sở và tất cả các chỉ mục của nó. Tuy nhiên, điều đó cũng có thể dẫn đến các tác dụng phụ không mong muốn. Phối hợp hoạt động này với quản trị viên cơ sở dữ liệu (DBA) và sao lưu các số liệu thống kê ban đầu.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Các chức năng do người dùng xác định

Lập chỉ mục dựa trên chức năng là một cách tiếp cận rất chung chung. Bên cạnh các hàm như UPPER , bạn cũng có thể lập chỉ mục các biểu thức như A + B và thậm chí sử dụng các hàm do người dùng xác định trong định nghĩa chỉ mục.

Có một ngoại lệ quan trọng. Ví dụ, không thể tham chiếu đến thời gian hiện tại trong một định nghĩa chỉ mục, không trực tiếp hay gián tiếp, như trong ví dụ sau.

```
TẠO FUNCTION get_age (date_of_birth DATE)
QUAY LẠI SỐ
BẰNG
BẮT ĐẦU
   TRỞ VỀ
      TRUNC (MONTHS_BETWEEN (SYSDATE, date_of_birth) / 12);
CHẤM
ĐỨT; /
```

Hàm GET_AGE sử dụng ngày hiện tại (SYSDATE) để tính tuổi dựa trên ngày sinh được cung cấp. Bạn có thể sử dụng hàm này trong tất cả các phần của truy vấn SQL, chẳng hạn như trong select và mệnh đề where :

```
CHỌN first_name, last_name, get_age (date_of_birth)
  TỪ nhân viên
 WHERE get_age (date_of_birth) = 42;
```

Truy vấn liệt kê tất cả nhân viên 42 tuổi. Sử dụng chỉ mục dựa trên hàm là một ý tưởng hiển nhiên để tối ưu hóa truy vấn này, nhưng bạn không thể sử dụng hàm GET_AGE trong định nghĩa chỉ mục vì nó không xác định. Điều đó có nghĩa là kết quả của lời gọi hàm không được xác định đầy đủ bởi các tham số của nó. Chỉ các hàm luôn trả về cùng một kết quả cho các tham số giống nhau - các hàm có tính xác định - mới có thể được lập chỉ mục.

Lý do đằng sau hạn chế này rất đơn giản. Khi chèn một hàng mới, cơ sở dữ liệu sẽ gọi hàm và lưu trữ kết quả trong chỉ mục và ở đó, nó vẫn ở đó, không thay đổi. Không có quy trình định kỳ cập nhật chỉ mục. Cơ sở dữ liệu chỉ cập nhật độ tuổi được lập chỉ mục khi ngày sinh được thay đổi bởi một câu lệnh cập nhật . Sau sinh nhật tiếp theo, tuổi được lưu trong chỉ mục sẽ bị sai.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Ngoài tính xác định, PostgreSQL và cơ sở dữ liệu Oracle yêu cầu các hàm phải được khai báo để xác định khi được sử dụng trong một chỉ mục, vì vậy bạn phải sử dụng từ khóa DETERMINISTIC (Oracle) hoặc IMMUTABLE (PostgreSQL).

Thận trọng

PostgreSQL và cơ sở dữ liệu Oracle tin tưởng các khai báo DETERMINISTIC hoặc IMMUTABLE — điều đó có nghĩa là chúng tin cậy nhà phát triển.

Bạn có thể khai báo hàm GET_AGE là hàm xác định và sử dụng nó trong định nghĩa chỉ mục. Bất kể khai báo nào, nó sẽ không hoạt động như dự định vì tuổi được lưu trữ trong chỉ mục sẽ không tăng khi năm tháng trôi qua; nhân viên sẽ không già đi — ít nhất là không có trong chỉ mục.

Các ví dụ khác cho các hàm không thể được "lập chỉ mục" là bộ tạo số ngẫu nhiên và các hàm phụ thuộc vào các biến môi trường.

Hãy nghĩ về nó

Làm cách nào bạn vẫn có thể sử dụng chỉ mục để tối ưu hóa truy vấn cho tất cả nhân viên 42 tuổi?

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Lập chỉ mục quá mức

Nếu khái niệm lập chỉ mục dựa trên hàm là mới đối với bạn, bạn có thể bị cám dỗ chỉ lập chỉ mục mọi thứ, nhưng thực tế đây là điều cuối cùng bạn nên làm. Lý do là mọi chỉ mục đều gây ra bảo trì liên tục.
Các chỉ mục dựa trên hàm đặc biệt rắc rối vì chúng rất dễ tạo ra các chỉ mục dư thừa.

Tìm kiếm không phân biệt chữ hoa chữ thường ở trên có thể được triển khai với
Chức năng LOWER cũng như:

```
CHỌN first_name, last_name, phone_number
  TỪ nhân viên
 WHERE LOWER (last_name) = LOWER ('winand');
```

Một chỉ mục duy nhất không thể hỗ trợ cả hai phương pháp bỏ qua trường hợp này. Tất nhiên, chúng tôi có thể tạo chỉ mục thứ hai trên LOWER (last_name) cho truy vấn này, nhưng điều đó có nghĩa là cơ sở dữ liệu phải duy trì hai chỉ mục cho mỗi câu lệnh chèn, cập nhật và xóa (xem thêm Chương 8, "Sửa đổi dữ liệu") . Để tạo ra một chỉ mục là đủ, bạn nên sử dụng một cách nhất quán cùng một chức năng trong toàn bộ ứng dụng của mình.

Mẹo

Hợp nhất đường dẫn truy cập để một chỉ mục có thể được sử dụng bởi một số truy vấn.

Mẹo

Luôn hướng tới việc lập chỉ mục dữ liệu gốc vì đó thường là thông tin hữu ích nhất mà bạn có thể đưa vào chỉ mục.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Truy vấn được tham số hóa

Phần này bao gồm một chủ đề bị bỏ qua trong hầu hết các sách giáo khoa SQL: truy vấn được tham số hóa và tham số ràng buộc.

Tham số ràng buộc – còn được gọi là tham số động hoặc biến ràng buộc – là một cách thay thế để truyền dữ liệu vào cơ sở dữ liệu. Thay vì đặt các giá trị trực tiếp vào câu lệnh SQL, bạn chỉ cần sử dụng trình giữ chỗ như ?, : name hoặc @name và cung cấp các giá trị thực bằng lệnh gọi API riêng biệt.

Không có gì xấu khi viết các giá trị trực tiếp vào các câu lệnh đặc biệt; Tuy nhiên, có hai lý do chính đáng để sử dụng các tham số ràng buộc trong các chương trình:

Bảo vệ

Biến ràng buộc là cách tốt nhất để ngăn chặn SQL injection[1] .

Cơ sở dữ liệu

Hiệu suất với bộ đệm kế hoạch thực thi như SQL Server và cơ sở dữ liệu Oracle có thể sử dụng lại một kế hoạch thực thi khi thực thi cùng một câu lệnh nhiều lần. Nó giúp tiết kiệm nỗ lực trong việc xây dựng lại kế hoạch thực thi nhưng chỉ hoạt động nếu câu lệnh SQL hoàn toàn giống nhau. Nếu bạn đặt các giá trị khác nhau vào câu lệnh SQL, cơ sở dữ liệu sẽ xử lý nó giống như một câu lệnh khác và tạo lại kế hoạch thực thi.

Khi sử dụng các tham số ràng buộc, bạn không viết các giá trị thực mà thay vào đó chèn các trình giữ chỗ vào câu lệnh SQL. Bằng cách đó, các câu lệnh không thay đổi khi thực thi chúng với các giá trị khác nhau.

[1] http://en.wikipedia.org/wiki/SQL_injection

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Đương nhiên có những ngoại lệ, ví dụ: nếu khối lượng dữ liệu bị ảnh hưởng phụ thuộc vào các giá trị thực tế:

```
99 hàng được chọn.

CHỌN first_name, last_name
   TỪ nhân viên
 WHERE công ty con = 20;

| Id | Hoạt động                                  | Tên                | Hàng | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO | 1                        |                    | 99 | 70 |
      | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | TUYỂN NHÂN VIÊN | 99 | 70 |
| * 2 | QUÉT NGUỒN CHỈ SỐ                          | NHÂN VIÊN_PK | 99 | 2 |


Thông tin vị ngữ (được xác định bằng id hoạt động):


    2 - truy cập ("SUBSIDIARY_ID" = 20)
```

Tra cứu chỉ mục mang lại hiệu suất tốt nhất cho các công ty con nhỏ, nhưng TABLE ACCESS FULL có thể làm tốt hơn chỉ số cho các công ty con lớn:

```
Đã chọn 1000 hàng.

CHỌN first_name, last_name
   TỪ nhân viên
 WHERE công ty con = 30;

| Id | Hoạt động                      | Tên             | Hàng | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO | | 1000 | 478 |
| * 1  | BẢNG TRUY CẬP ĐẦY ĐỦ | TUYỂN NHÂN VIÊN | 1000 | 478 |


Thông tin vị ngữ (được xác định bằng id hoạt động):


    1 - bộ lọc ("SUBSIDIARY_ID" = 30)
```

Trong trường hợp này, biểu đồ trên SUBSIDIARY_ID đáp ứng mục đích của nó. Các trình tối ưu hóa sử dụng nó để xác định tần suất của ID phụ được đề cập trong truy vấn SQL. Do đó, nó nhận được hai ước tính số hàng khác nhau cho cả hai truy vấn.

Machine Translated by Google

Do đó, việc tính toán chi phí tiếp theo sẽ dẫn đến hai giá trị chi phí khác nhau. Khi trình tối ưu hóa cuối cùng chọn một kế hoạch thực thi, nó sẽ chọn kế hoạch có giá trị chi phí thấp nhất. Đối với công ty con nhỏ hơn, nó là công ty sử dụng chỉ số.

Chi phí của hoạt động TABLE ACCESS BY INDEX ROWID rất nhạy cảm với ước tính số lượng hàng. Việc chọn số hàng nhiều gấp mười lần sẽ nâng cao giá trị chi phí theo hệ số đó. Chi phí tổng thể khi sử dụng chỉ mục thậm chí còn cao hơn so với việc quét toàn bộ bảng. Do đó, trình tối ưu hóa sẽ chọn kế hoạch thực thi khác cho công ty con lớn hơn.

Khi sử dụng các tham số liên kết, trình tối ưu hóa không có sẵn các giá trị cụ thể để xác định tần suất của chúng. Sau đó, nó chỉ giả sử một phân phối bằng nhau và luôn nhận được các ước tính số lượng hàng và giá trị chi phí giống nhau.
Cuối cùng, nó sẽ luôn chọn cùng một kế hoạch thực hiện.

Mẹo

Biểu đồ cột hữu ích nhất nếu các giá trị không được phân phối đồng đều.

Đối với các cột có phân bố đồng đều, thường là đủ để chia số lượng các giá trị riêng biệt cho số hàng trong bảng.
Phương thức này cũng hoạt động khi sử dụng các tham số ràng buộc.

Nếu chúng ta so sánh trình tối ưu hóa với trình biên dịch, các biến ràng buộc giống như các biến chương trình, nhưng nếu bạn viết các giá trị trực tiếp vào câu lệnh thì chúng giống như hằng số hơn. Cơ sở dữ liệu có thể sử dụng các giá trị từ câu lệnh SQL trong quá trình tối ưu hóa giống như một trình biên dịch có thể đánh giá các biểu thức hằng trong quá trình biên dịch. Các tham số ràng buộc, nói một cách đơn giản, không hiển thị với trình tối ưu hóa cũng như các giá trị thời gian chạy của các biến không được trình biên dịch biết đến.

Từ góc độ này, có một chút nghịch lý là các tham số liên kết có thể cải thiện hiệu suất nếu không sử dụng các tham số liên kết cho phép trình tối ưu hóa luôn chọn kế hoạch thực thi tốt nhất. Nhưng câu hỏi là ở mức giá nào?
Việc tạo và đánh giá tất cả các biến thể của kế hoạch thực thi là một nỗ lực rất lớn nhưng sẽ không được đền đáp nếu cuối cùng bạn vẫn nhận được cùng một kết quả.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Mẹo

Không sử dụng các tham số ràng buộc giống như việc biên dịch lại một chương trình mỗi lần.

Quyết định xây dựng một kế hoạch thực hiện chuyên biệt hoặc chung chung đưa ra một tình huống khó xử đối với cơ sở dữ liệu. Nỗ lực được thực hiện để đánh giá tất cả các biến thể kế hoạch có thể có cho mỗi lần thực hiện để luôn có được kế hoạch thực hiện tốt nhất hoặc chi phí tối ưu hóa được lưu và kế hoạch thực hiện được lưu trong bộ nhớ cache được sử dụng bất cứ khi nào có thể – chấp nhận rủi ro khi sử dụng kế hoạch thực hiện dưới mức tối ưu. Vấn đề khó khăn là cơ sở dữ liệu không biết liệu chu kỳ tối ưu hóa đầy đủ có đưa ra một kế hoạch thực thi khác mà không thực sự thực hiện tối ưu hóa đầy đủ hay không. Các nhà cung cấp cơ sở dữ liệu cố gắng giải quyết tình huống khó xử này bằng các phương pháp heuristic - nhưng thành công rất hạn chế.

Với tư cách là nhà phát triển, bạn có thể chủ ý sử dụng các tham số ràng buộc để giúp giải quyết tình huống khó xử này. Đó là, bạn nên luôn sử dụng các tham số ràng buộc ngoại trừ các giá trị sẽ ảnh hưởng đến kế hoạch thực thi.

Các mã trạng thái được phân phối không đồng đều như “việc cần làm” và “đã xong” là một ví dụ điển hình. Số lượng các mục "đã hoàn thành" thường vượt quá các bản ghi "việc cần làm" theo một thứ tự về độ lớn. Sử dụng một chỉ mục chỉ có ý nghĩa khi tìm kiếm các mục nhập "việc cần làm" trong trường hợp đó. Phân vùng là một ví dụ khác – nghĩa là, nếu bạn chia các bảng và chỉ mục trên nhiều vùng lưu trữ. Các giá trị thực tế sau đó có thể ảnh hưởng đến phân vùng nào phải được quét. Hiệu suất của các truy vấn LIKE có thể bị ảnh hưởng bởi các tham số ràng buộc cũng như chúng ta sẽ thấy trong phần tiếp theo.

Mẹo

Trong thực tế, chỉ có một số trường hợp giá trị thực ảnh hưởng đến kế hoạch thực thi. Do đó, bạn nên sử dụng các tham số liên kết nếu nghi ngờ – chỉ để ngăn chặn việc đưa vào SQL.

Các đoạn mã sau đây cho thấy cách sử dụng các tham số liên kết trong các ngôn ngữ lập trình khác nhau.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

C#

Không có tham số ràng buộc:

```
int child_id;
SqlCommand cmd = new SqlCommand (
                     "select first_name, last_name"
                   + "từ nhân viên"             "
                   +   nơi con_id = , kết nối);   + widget_id
```

Sử dụng tham số liên kết:

```
int child_id;
SqlCommand cmd =
        SqlCommand mới (
                         "select first_name, last_name"
                       + "từ nhân viên"
                       +   trong đó công ty con_id = @subsidiary_id
                       , sự liên quan);
cmd.Parameters.AddWithValue ("@ công ty con_id", công ty con_id);
```

Xem thêm: Tài liệu về lớp SqlParameterCollection .

Java

Không có tham số ràng buộc:

```
int child_id;
Câu lệnh lệnh = connection.createStatement (
                       "select first_name, last_name"
                     + "từ nhân viên"             "
                     +   nơi mà công ty con =       + widget_id
                     );
```

Sử dụng tham số liên kết:

```
int child_id;
Lệnh PreparedStatement = connection.prepareStatement (
                       "select first_name, last_name"
                     + "từ nhân viên"
                     +   nơi công ty con =? "
                     );
command.setInt (1, child_id);
```

Xem thêm: Tài liệu về lớp PreparedStatement .

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Perl

Không có tham số ràng buộc:

```
$ công ty con của tôi;
my $ sth = $ dbh-> chuẩn bị (
                      "select first_name, last_name"
                    . "từ nhân viên"
                    .   trong đó công ty con_id = $ công ty con_id "
                    );
$ sth-> thi hành ();
```

Sử dụng tham số liên kết:

```
$ công ty con của tôi;
my $ sth = $ dbh-> chuẩn bị (
                      "select first_name, last_name"
                    . "từ nhân viên"
                    .   nơi công ty con =? "
                    );
$ sth-> thi hành ($ child_id);
```

Xem: Lập trình Perl DBI.

PHP

Sử dụng MySQL, không có tham số ràng buộc:

```
$ mysqli-> query ("select first_name,
                . last_name" "từ nhân viên"
                .   nơi mà công ty con =     . $ công_trục);
```

Sử dụng tham số liên kết:

```
if ($ stmt = $ mysqli-> standard ("select first_name,
                              . last_name" "từ nhân viên"
                              .   nơi công ty con = ? "))
{
   $ stmt-> bind_param ("i", $ child_id);
   $ stmt-> thi hành ();
} khác {
  / * xử lý lỗi SQL * /
}
```

Xem thêm: tài liệu lớp mysqli_stmt :: bind_param và "Đã chuẩn bị các tuyên bố và các thủ tục được lưu trữ "trong tài liệu PDO.

Machine Translated by Google

Ruby

Không có tham số ràng buộc:

```
dbh.execute ("chọn first_name, last_name"
           + "từ nhân viên"
           +   nơi công ty con = {công ty con_id} ");
```

Sử dụng tham số liên kết:

```
dbh.prepare ("chọn first_name, last_name"
           + "từ nhân viên"
           +   nơi công ty con =? ");
dbh.execute (con_mua_trinh);
```

Xem thêm: “Trích dẫn, Trình giữ chỗ và Liên kết Tham số” trong Ruby Hướng dẫn sử dụng DBI.

Dấu chấm hỏi (?) Là ký tự giữ chỗ duy nhất mà SQL tiêu chuẩn xác định. Dấu hỏi là tham số vị trí. Điều đó có nghĩa là các dấu hỏi được đánh số từ trái sang phải. Để ràng buộc một giá trị với một dấu hỏi cụ thể, bạn phải chỉ định số của nó. Việc đó có thể, tuy nhiên, rất không thực tế vì việc đánh số thay đổi khi thêm hoặc xóa trình giữ chỗ. Nhiều cơ sở dữ liệu cung cấp phần mở rộng độc quyền cho các tham số được đặt tên để giải quyết vấn đề này — ví dụ: sử dụng ký hiệu “at” (@name) hoặc dấu hai chấm (: tên).

Ghi chú

Tham số ràng buộc không thể thay đổi cấu trúc của một câu lệnh SQL.

Điều đó có nghĩa là bạn không thể sử dụng các tham số liên kết cho bảng hoặc cột những cái tên. Các tham số liên kết sau không hoạt động:

```
String sql = standard ("CHỌN * TỪ? Ở ĐÂU?");

sql.execute ('nhân viên', 'worker_id = 1');
```

Nếu bạn cần thay đổi cấu trúc của câu lệnh SQL trong thời gian chạy, sử dụng SQL động.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Chia sẻ con trỏ và Tham số tự động

Trình tối ưu hóa và truy vấn SQL càng trở nên phức tạp, thì bộ nhớ đệm kế hoạch thực thi càng trở nên quan trọng hơn. Cơ sở dữ liệu SQL Server và Oracle có các tính năng tự động thay thế các giá trị chữ trong chuỗi SQL bằng các tham số ràng buộc. Các tính năng này được gọi là CURSOR_SHARING (Oracle) hoặc tham số hóa bắt buộc (SQL Server).

Cả hai tính năng đều là giải pháp thay thế cho các ứng dụng hoàn toàn không sử dụng các tham số ràng buộc. Việc kích hoạt các tính năng này ngăn các nhà phát triển cố ý sử dụng các giá trị theo nghĩa đen.

Tìm kiếm Dãy

Các toán tử bất đẳng thức như <, > và between có thể sử dụng các chỉ mục giống như toán tử bằng được giải thích ở trên. Ngay cả bộ lọc LIKE cũng có thể – trong một số trường hợp nhất định – sử dụng một chỉ mục giống như các điều kiện phạm vi.

Việc sử dụng các thao tác này sẽ giới hạn sự lựa chọn thứ tự cột trong các chỉ mục nhiều cột. Hạn chế này thậm chí có thể loại trừ tất cả các tùy chọn lập chỉ mục tối ưu – có những truy vấn mà bạn chỉ đơn giản là không thể xác định thứ tự cột "đúng".

Lớn hơn, ít hơn và GIỮA

Rủi ro hiệu suất lớn nhất của QUÉT NGUỒN CHỈ SỐ là việc duyệt nút lá. Do đó, quy tắc vàng của lập chỉ mục là giữ cho phạm vi chỉ mục được quét càng nhỏ càng tốt. Bạn có thể kiểm tra điều đó bằng cách tự hỏi mình nơi bắt đầu quét chỉ mục và nơi kết thúc.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Câu hỏi rất dễ trả lời nếu câu lệnh SQL đề cập đến điều kiện bắt đầu và dừng một cách rõ ràng:

```
CHỌN first_name, last_name, date_of_birth
  TỪ nhân viên
 WHERE date_of_birth> = TO_DATE (?, 'YYYY-MM-DD')
   VÀ date_of_birth <= TO_DATE (?, 'YYYY-MM-DD')
```

Chỉ mục vào DATE_OF_BIRTH chỉ được quét trong phạm vi được chỉ định. Quá trình quét bắt đầu vào ngày đầu tiên và kết thúc vào ngày thứ hai. Chúng tôi không thể thu hẹp phạm vi chỉ mục đã quét thêm nữa.

Điều kiện bắt đầu và dừng ít rõ ràng hơn nếu cột thứ hai có liên quan:

```
CHỌN first_name, last_name, date_of_birth
  TỪ nhân viên
 WHERE date_of_birth> = TO_DATE (?, 'YYYY-MM-DD')
   VÀ date_of_birth <= TO_DATE (?, 'YYYY-MM-DD')
   VÀ child_id =?
```

Tất nhiên một chỉ số lý tưởng phải bao gồm cả hai cột, nhưng câu hỏi đặt ra là theo thứ tự nào?

Các số liệu sau đây cho thấy ảnh hưởng của thứ tự cột đối với phạm vi chỉ mục được quét. Đối với hình minh họa này, chúng tôi tìm kiếm tất cả nhân viên của công ty con 27, những người sinh từ ngày 1 tháng 1 đến ngày 9 tháng 1 năm 1971.

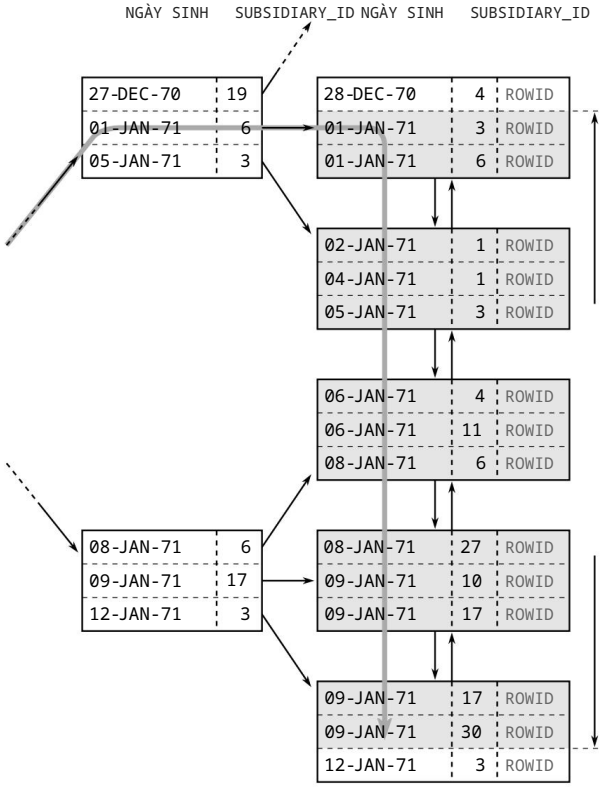
Hình 2.2 hiển thị chi tiết của chỉ mục trên DATE_OF_BIRTH và SUBSIDIARY_ID — theo thứ tự đó. Cơ sở dữ liệu sẽ bắt đầu từ đâu để tuân theo chuỗi nút lá, hay nói một cách khác: việc duyệt cây sẽ kết thúc ở đâu?

Chỉ số được sắp xếp theo ngày tháng năm sinh trước. Chỉ khi hai nhân viên sinh cùng ngày thì SUBSIDIARY_ID mới được sử dụng để sắp xếp các bản ghi này. Tuy nhiên, truy vấn bao gồm một phạm vi ngày. Do đó, thứ tự SUBSIDIARY_ID vô dụng trong quá trình duyệt cây. Điều đó trở nên rõ ràng nếu bạn nhận ra rằng không có mục nhập nào cho công ty con 27 trong các nút nhánh — mặc dù có một mục trong các nút lá. Do đó, bộ lọc vào DATE_OF_BIRTH là điều kiện duy nhất giới hạn phạm vi chỉ mục được quét. Nó bắt đầu ở mục nhập đầu tiên phù hợp với phạm vi ngày và kết thúc ở mục cuối cùng — tất cả năm nút lá được thể hiện trong Hình 2.2.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Hình 2.2. Quét phạm vi trong DATE_OF_BIRTH, Chỉ mục SUBSIDIARY_ID

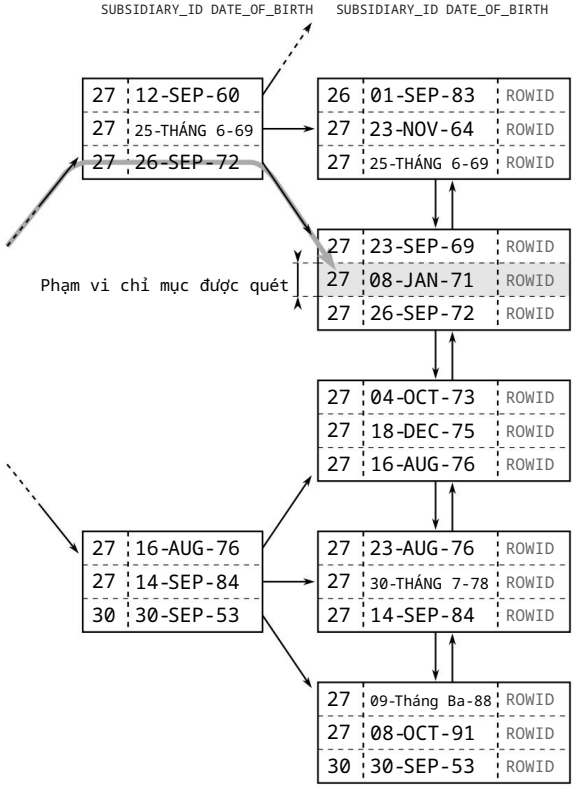


Hình ảnh trông hoàn toàn khác khi đảo ngược thứ tự cột.
Hình 2.3 minh họa quá trình quét nếu chỉ mục bắt đầu bằng cột SUBSIDIARY_ID .

Sự khác biệt là toán tử bằng giới hạn cột chỉ mục đầu tiên ở một giá trị duy nhất. Trong phạm vi cho giá trị này (SUBSIDIARY_ID 27), chỉ mục được sắp xếp theo cột thứ hai - ngày sinh - vì vậy không cần phải truy cập vào nút lá đầu tiên vì nút nhánh đã chỉ ra rằng không có nhân viên nào cho công ty con 27 sinh sau ngày 25 tháng 6 năm 1969 tại
nút lá đầu tiên.

Machine Translated by Google

Hình 2.3. Quét phạm vi trong chỉ mục SUBSIDIARY_ID , DATE_OF_BIRTH



Đường đi ngang của cây trực tiếp dẫn đến nút lá thứ hai. Trong trường hợp này, tất cả các điều kiện mệnh đề where giới hạn phạm vi chỉ mục được quét để quá trình quét kết thúc tại cùng một nút lá.

Mẹo

Quy tắc ngón tay cái: lập chỉ mục cho bình đẳng trước - sau đó cho các phạm vi.

Sự khác biệt về hiệu suất thực tế phụ thuộc vào dữ liệu và tiêu chí tìm kiếm. Sự khác biệt có thể không đáng kể nếu bộ lọc vào DATE_OF_BIRTH tự nó rất chọn lọc. Phạm vi ngày càng lớn, sự khác biệt về hiệu suất sẽ càng lớn.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Với ví dụ này, chúng ta cũng có thể làm sai lệch huyền thoại được chọn lọc nhất cột phải ở vị trí chỉ mục ngoài cùng bên trái. Nếu chúng ta nhìn vào các số liệu và chỉ xem xét tính chọn lọc của cột đầu tiên, chúng tôi thấy rằng cả hai điều kiện phù hợp với 13 bản ghi. Đây là trường hợp bất kể chúng tôi lọc chỉ bởi DATE_OF_BIRTH hoặc chỉ SUBSIDIARY_ID . Tính chọn lọc không có ích lợi gì ở đây, nhưng thứ tự một cột vẫn tốt hơn thứ tự cột kia.

Để tối ưu hóa hiệu suất, điều rất quan trọng là phải biết chỉ mục được quét phạm vi. Với hầu hết các cơ sở dữ liệu, bạn thậm chí có thể thấy điều này trong kế hoạch thực thi— bạn chỉ cần biết những gì cần tìm. Kế hoạch thực hiện sau đây từ cơ sở dữ liệu Oracle chỉ ra rõ ràng rằng chỉ mục EMP_TEST bắt đầu với cột DATE_OF_BIRTH .

```
| Id | Hoạt động                                          | Tên             | Hàng | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO | |                                                          1          4 |
| * 1 | BỘ LỌC | |                                                                          |
| 2 | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | TUYỂN NHÂN VIÊN | | * 3 |          | | |        4 |
              QUÉT NGUỒN CHỈ SỐ                          | EMP_TEST |          1 2 |        2 |


Thông tin vị ngữ (được xác định bằng id hoạt động):

1 - bộ lọc (: END_DT> =: START_DT)
3 - truy cập (DATE_OF_BIRTH> =: START_DT
          VÀ DATE_OF_BIRTH <=: END_DT)
      bộ lọc (SUBSIDIARY_ID =: SUBS_ID)
```

Thông tin vị từ cho QUÉT NGUỒN CHỈ SỐ cung cấp gợi ý quan trọng. Nó xác định các điều kiện của mệnh đề where là quyền truy cập hoặc là bộ lọc các vị ngữ. Đây là cách cơ sở dữ liệu cho chúng ta biết cách nó sử dụng từng điều kiện.

> **Ghi chú**
>
> Kế hoạch thực hiện đã được đơn giản hóa để rõ ràng. Phần phụ lục trên trang 170 giải thích các chi tiết của phần "Thông tin dự đoán" trong một kế hoạch thực thi Oracle.

Các điều kiện trên cột DATE_OF_BIRTH là những điều kiện duy nhất được liệt kê dưới dạng vị ngữ truy cập; chúng giới hạn phạm vi chỉ mục được quét. DATE_OF_BIRTH là _ do đó là cột đầu tiên trong chỉ mục EMP_TEST . Cột SUBSIDIARY_ID _ chỉ được sử dụng như một bộ lọc.

Machine Translated by Google

Quan trọng

Các vị từ truy cập là điều kiện bắt đầu và dừng cho một chỉ mục tra cứu. Chúng xác định phạm vi chỉ mục được quét.

Các vị từ bộ lọc chỉ mục chỉ được áp dụng trong quá trình duyệt nút lá. Chúng không thu hẹp phạm vi chỉ mục được quét.

Phụ lục A giải thích cách nhận ra các vị từ truy cập trong các cơ sở dữ liệu.

Cơ sở dữ liệu có thể sử dụng tất cả các điều kiện làm vị từ truy cập nếu chúng ta chuyển chỉ mục định nghĩa xung quanh:

```
-----------------------------------------------------------------------------------------
| Id | Hoạt động                                          | Tên             | Hàng | Chi phí |
-----------------------------------------------------------------------------------------
| 0 | TUYỂN CHỌN BÁO CÁO | | | * 1                                                  1 |         3 |
        | BỘ LỌC | |                                                                |           |
| 2 | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | TUYỂN NHÂN VIÊN | | * 3 |                      |         3 |
                QUÉT NGUỒN CHỈ SỐ                          | EMP_TEST2 |            1 1 |         2 |
-----------------------------------------------------------------------------------------


Thông tin vị ngữ (được xác định bằng id hoạt động):
---------------------------------------------------

1 - bộ lọc (: END_DT> =: START_DT)
3 - truy cập (SUBSIDIARY_ID =: SUBS_ID
          VÀ DATE_OF_BIRTH> =: START_DT
          VÀ DATE_OF_BIRTH <=: END_T)
```

Cuối cùng, có toán tử giữa . Nó cho phép bạn chỉ định phần trên và giới hạn thấp hơn trong một điều kiện:

```
DATE_OF_BIRTH GIỮA '01 -JAN-71 '
                          VÀ '10 -JAN-71 '
```

Lưu ý rằng giữa luôn bao gồm các giá trị được chỉ định, giống như sử dụng nhỏ hơn hoặc bằng (<=) và lớn hơn hoặc bằng (> =) các toán tử:

```
      DATE_OF_BIRTH> = '01 -JAN-71 '
VÀ DATE_OF_BIRTH <= '10 -JAN-71 '
```


Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Lập chỉ mục Bộ lọc LIKE

Toán tử LIKE trong SQL rất thường gây ra hành vi hiệu suất không mong muốn bởi vì một số cụm từ tìm kiếm ngăn cản việc sử dụng chỉ mục hiệu quả. Đó nghĩa là có những cụm từ tìm kiếm có thể được lập chỉ mục rất tốt, nhưng những cụm từ khác thì không. Nó là vị trí của các ký tự đại diện tạo nên sự khác biệt.

Ví dụ sau sử dụng ký tự đại diện % ở giữa cụm từ tìm kiếm:

```
CHỌN first_name, last_name, date_of_birth
   TỪ nhân viên
 NƠI LÊN (last_name) THÍCH 'THẮNG% D'

+----+-----------------------------------+-----------------+------+-------+
| Id | Hoạt động                         | Tên             | Hàng | Chi phí |
+----+-----------------------------------+-----------------+------+-------+
| 0 | TUYỂN CHỌN BÁO CÁO | | 1                    |                 |      |       4 |
      | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | TUYỂN NHÂN VIÊN |                 |      |       4 |
| * 2 | QUÉT NGUỒN CHỈ SỐ                 | EMP_UP_NAME |                 |      |       2 |
+----+-----------------------------------+-----------------+------+-------+
```

Bộ lọc LIKE chỉ có thể sử dụng các ký tự trước ký tự đại diện đầu tiên trong cây đi ngang. Các ký tự còn lại chỉ là các vị từ bộ lọc không thu hẹp phạm vi chỉ mục được quét. Do đó, một biểu thức LIKE duy nhất có thể chứa hai loại vị từ: (1) phần trước ký tự đại diện đầu tiên như một vị ngữ truy cập; (2) các ký tự khác làm vị ngữ bộ lọc.

> **Thận trọng**
>
> Đối với cơ sở dữ liệu PostgreSQL, bạn có thể cần chỉ định một toán tử lớp (ví dụ: varchar_pattern_ops) để sử dụng biểu thức LIKE làm quyền truy cập các vị ngữ. Tham khảo "Các lớp nhà điều hành và họ nhà điều hành" trong Tài liệu PostgreSQL để biết thêm chi tiết.

Tiền tố trước ký tự đại diện đầu tiên càng chọn lọc thì càng nhỏ phạm vi chỉ mục được quét sẽ trở thành. Điều đó, đến lượt nó, làm cho việc tra cứu chỉ mục nhanh hơn. Hình 2.4 minh họa mối quan hệ này bằng cách sử dụng ba LIKE khác nhau biểu thức. Cả ba đều chọn cùng một hàng, nhưng phạm vi chỉ mục được quét— và do đó hiệu suất — rất khác nhau.

Machine Translated by Google

Hình 2.4. Các tìm kiếm LIKE khác nhau

THÍCH 'WI% ND'	THÍCH 'CHIẾN THẮNG% D'	THÍCH 'WINA%'
WIAW	WIAW	WIAW
WIBLQQNPUA	WIBLQQNPUA	WIBLQQNPUA
WIBYHSNZ	WIBYHSNZ	WIBYHSNZ
WIFMDWUQMB	WIFMDWUQMB	WIFMDWUQMB
WIGLZX	WIGLZX	WIGLZX
WIH	WIH	WIH
WIHTFVZNLC	WIHTFVZNLC	WIHTFVZNLC
WIJYAXPP	WIJYAXPP	WIJYAXPP
CHIẾN THẮNG	CHIẾN THẮNG	CHIẾN THẮNG
WINBKYDSKW	WINBKYDSKW	WINBKYDSKW
WIPOJ	WIPOJ	WIPOJ
WISRGPK	WISRGPK	WISRGPK
WITJIVQJ	WITJIVQJ	WITJIVQJ
WIW	WIW	WIW
WIWGPJMQGG	WIWGPJMQGG	WIWGPJMQGG
WIWKHLBJ	WIWKHLBJ	WIWKHLBJ
WIYETHN	WIYETHN	WIYETHN
WIYJ	WIYJ	WIYJ

Biểu thức đầu tiên có hai ký tự trước ký tự đại diện. Chúng giới hạn phạm vi chỉ mục được quét là 18 hàng. Chỉ một trong số chúng phù hợp với toàn bộ biểu thức LIKE — 17 biểu thức còn lại được tìm nạp nhưng bị loại bỏ. Biểu thức thứ hai có tiền tố dài hơn giúp thu hẹp phạm vi chỉ mục được quét xuống còn hai hàng. Với biểu thức này, cơ sở dữ liệu chỉ đọc thêm một hàng không liên quan đến kết quả. Biểu thức cuối cùng hoàn toàn không có một vị từ bộ lọc: cơ sở dữ liệu chỉ đọc mục nhập khớp với toàn bộ biểu thức LIKE .

Quan trọng

Chỉ phần trước ký tự đại diện đầu tiên đóng vai trò là vị từ truy cập.

Các ký tự còn lại không thu hẹp phạm vi chỉ mục đã quét— các mục nhập không khớp chỉ bị loại ra khỏi kết quả.

Trường hợp ngược lại cũng có thể xảy ra: biểu thức LIKE bắt đầu bằng ký tự đại diện. Biểu thức LIKE như vậy không thể đóng vai trò là một vị từ truy cập. Cơ sở dữ liệu phải quét toàn bộ bảng nếu không có điều kiện nào khác cung cấp các vị từ truy cập.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Mẹo

Tránh biểu thức LIKE có ký tự đại diện đứng đầu (ví dụ: '% TERM').

Vị trí của các ký tự đại diện ảnh hưởng đến việc sử dụng chỉ mục – ít nhất là trên lý thuyết. Trong thực tế, trình tối ưu hóa tạo ra một kế hoạch thực thi chung khi cụm từ tìm kiếm được cung cấp thông qua các tham số liên kết. Trong trường hợp đó, trình tối ưu hóa phải đoán xem liệu phần lớn các lần thực thi sẽ có ký tự đại diện đứng đầu hay không.

Hầu hết các cơ sở dữ liệu chỉ giả sử rằng không có ký tự đại diện đứng đầu khi tối ưu hóa điều kiện LIKE với tham số ràng buộc, nhưng giả định này là sai nếu biểu thức LIKE được sử dụng cho tìm kiếm toàn văn. Thật không may, không có cách nào trực tiếp để gắn thẻ điều kiện LIKE dưới dạng tìm kiếm toàn văn.
Hộp "Gắn nhãn Biểu thức LIKE toàn văn bản " cho thấy một nỗ lực không hoạt động. Chỉ định cụm từ tìm kiếm mà không có tham số ràng buộc là giải pháp rõ ràng nhất, nhưng điều đó làm tăng chi phí tối ưu hóa và mở ra lỗ hổng SQL injection. Một giải pháp hiệu quả nhưng vẫn an toàn và di động là cố ý làm xáo trộn điều kiện LIKE . "Kết hợp các cột" trên trang 70 giải thích điều này một cách chi tiết.

Gắn nhãn Biểu thức LIKE toàn văn bản

Khi sử dụng toán tử LIKE cho tìm kiếm toàn văn, chúng tôi có thể tách các ký tự đại diện khỏi cụm từ tìm kiếm:

```
WHERE text_column LIKE '%' || ? || '%'
```

Các ký tự đại diện được viết trực tiếp vào câu lệnh SQL, nhưng chúng tôi sử dụng một tham số ràng buộc cho cụm từ tìm kiếm. Biểu thức LIKE cuối cùng được xây dựng bởi chính cơ sở dữ liệu bằng cách sử dụng toán tử nối chuỗi || (Oracle, PostgreSQL). Mặc dù sử dụng tham số ràng buộc, biểu thức LIKE cuối cùng sẽ luôn bắt đầu bằng ký tự đại diện. Thật không may, cơ sở dữ liệu không nhận ra điều đó.

Đối với cơ sở dữ liệu PostgreSQL, vấn đề lại khác vì PostgreSQL giả định có một ký tự đại diện đứng đầu khi sử dụng các tham số liên kết cho một biểu thức LIKE . PostgreSQL chỉ không sử dụng một chỉ mục trong trường hợp đó. Cách duy nhất để có được quyền truy cập chỉ mục cho biểu thức LIKE là làm cho

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

cụm từ tìm kiếm hiển thị cho trình tối ưu hóa. Nếu bạn không sử dụng tham số liên kết nhưng đặt cụm từ tìm kiếm trực tiếp vào câu lệnh SQL, bạn phải thực hiện các biện pháp phòng ngừa khác chống lại các cuộc tấn công SQL injection!

Ngay cả khi cơ sở dữ liệu tối ưu hóa kế hoạch thực thi cho một ký tự đại diện hàng đầu, nó vẫn có thể cung cấp hiệu suất không đủ. Bạn có thể sử dụng một phần khác của mệnh đề where để truy cập dữ liệu một cách hiệu quả trong trường hợp đó — xem thêm "Các dự đoán bộ lọc chỉ mục được sử dụng có chủ ý" trên trang 112. Nếu không có đường dẫn truy cập nào khác, bạn có thể sử dụng một trong các toàn văn độc quyền sau các giải pháp chỉ mục.

MySQL

MySQL cung cấp các từ khóa đối sánh và đối sánh để tìm kiếm toàn văn. Bắt đầu với MySQL 5.6, bạn cũng có thể tạo các chỉ mục toàn văn cho các bảng InnoDB — trước đây, điều này chỉ có thể thực hiện được với các bảng MyISAM. Xem "Các chức năng tìm kiếm toàn văn bản" trong tài liệu MySQL.

Cơ sở dữ liệu Oracle

Cơ sở dữ liệu Oracle cung cấp từ khóa chứa . Xem "Hướng dẫn của nhà phát triển ứng dụng văn bản Oracle".

PostgreSQL

PostgreSQL cung cấp toán tử @@ để triển khai các tìm kiếm toàn văn. Xem "Tìm kiếm toàn văn bản" trong tài liệu PostgreSQL.

Một tùy chọn khác là sử dụng WildSpeed2 tiện ích mở rộng để tối ưu hóa các biểu thức LIKE trực tiếp. Tiện ích mở rộng lưu trữ văn bản trong tất cả các cách xoay có thể để mỗi ký tự ở đầu một lần. Điều đó có nghĩa là văn bản được lập chỉ mục không chỉ được lưu trữ một lần mà thay vào đó là nhiều lần khi có các ký tự trong chuỗi — do đó, nó cần rất nhiều không gian.

Máy chủ SQL

SQL Server cung cấp từ khóa chứa . Xem "Tìm kiếm toàn văn bản" trong tài liệu SQL Server.

Hãy nghĩ về nó

Làm cách nào bạn có thể lập chỉ mục một tìm kiếm LIKE chỉ có một ký tự đại diện ở đầu cụm từ tìm kiếm ('% TERM')?

[2] http://www.sai.msu.su/~megera/wiki/wildspeed

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Hợp nhất chỉ mục

Đó là một trong những câu hỏi phổ biến nhất về lập chỉ mục: tạo một chỉ mục cho mỗi cột hay một chỉ mục duy nhất cho tất cả các cột của mệnh đề where là tốt hơn? Câu trả lời rất đơn giản trong hầu hết các trường hợp: một chỉ mục có nhiều cột sẽ tốt hơn.

Tuy nhiên, có những truy vấn trong đó một chỉ mục không thể thực hiện công việc hoàn hảo, cho dù bạn xác định chỉ mục như thế nào; ví dụ: các truy vấn có hai hoặc nhiều điều kiện phạm vi độc lập như trong ví dụ sau:

```
CHỌN first_name, last_name, date_of_birth
  TỪ nhân viên
 LÊN ĐẦU (last_name) <?
   VÀ date_of_birth <?
```

Không thể xác định chỉ mục B-tree sẽ hỗ trợ truy vấn này mà không có các vị từ bộ lọc. Để giải thích, bạn chỉ cần nhớ rằng một chỉ mục là một danh sách được liên kết.

Nếu bạn xác định chỉ mục là UPPER (LAST_NAME), DATE_OF_BIRTH (theo thứ tự đó), danh sách bắt đầu bằng A và kết thúc bằng Z. Ngày sinh chỉ được xem xét khi có hai nhân viên trùng tên. Nếu bạn xác định chỉ số theo cách khác, nó sẽ bắt đầu với những nhân viên lớn tuổi nhất và kết thúc với những người trẻ nhất. Trong trường hợp đó, các tên chỉ có tác động nhỏ đến thứ tự sắp xếp.

Bất kể bạn xoay và xoay định nghĩa chỉ mục như thế nào, các mục nhập luôn được sắp xếp dọc theo một chuỗi. Ở một đầu, bạn có các mục nhỏ và ở đầu kia là các mục lớn. Do đó, một chỉ mục chỉ có thể hỗ trợ một điều kiện phạm vi như một vị từ truy cập. Hỗ trợ hai điều kiện phạm vi độc lập yêu cầu trục thứ hai, ví dụ như bàn cờ. Sau đó, truy vấn trên sẽ khớp với tất cả các mục nhập từ một góc của bàn cờ, nhưng một chỉ mục không giống như một bàn cờ - nó giống như một chuỗi. Không có một góc nào.

Tất nhiên, bạn có thể chấp nhận vị từ bộ lọc và sử dụng chỉ mục nhiều cột. Dù sao đó cũng là giải pháp tốt nhất trong nhiều trường hợp. Định nghĩa chỉ mục sau đó nên đề cập đến cột chọn lọc hơn trước để nó có thể được sử dụng với một vị từ truy cập. Đó có thể là nguồn gốc của huyền thoại "đầu tiên được chọn lọc nhất" nhưng quy tắc này chỉ đúng nếu bạn không thể tránh một vị từ bộ lọc.

Machine Translated by Google

Tùy chọn khác là sử dụng hai chỉ mục riêng biệt, một chỉ mục cho mỗi cột. Sau đó, cơ sở dữ liệu phải quét cả hai chỉ mục trước và sau đó kết hợp các kết quả.

Chỉ riêng việc tra cứu chỉ mục trùng lặp đã đòi hỏi nhiều nỗ lực hơn vì cơ sở dữ liệu phải duyệt qua hai cây chỉ mục. Ngoài ra, cơ sở dữ liệu cần nhiều bộ nhớ và thời gian CPU để kết hợp các kết quả trung gian.

Ghi chú

Một lần quét chỉ mục nhanh hơn hai lần.

Cơ sở dữ liệu sử dụng hai phương pháp để kết hợp các chỉ mục. Đầu tiên là tham gia chỉ mục. Chương 4, "Hoạt động kết hợp" giải thích chi tiết các thuật toán liên quan. Cách tiếp cận thứ hai sử dụng chức năng từ thế giới kho dữ liệu.

Kho dữ liệu là mẹ của tất cả các truy vấn đặc biệt. Nó chỉ cần một vài cú nhấp chuột để kết hợp các điều kiện tùy ý vào truy vấn mà bạn lựa chọn.
Không thể dự đoán các kết hợp cột có thể xuất hiện trong mệnh đề where và điều đó làm cho việc lập chỉ mục, như đã giải thích cho đến nay, gần như là không thể.

Kho dữ liệu sử dụng một loại chỉ mục mục đích đặc biệt để giải quyết vấn đề đó: cái gọi là chỉ mục bitmap. Ưu điểm của chỉ mục bitmap là chúng có thể được kết hợp khá dễ dàng. Điều đó có nghĩa là bạn sẽ có được hiệu suất tốt khi lập chỉ mục từng cột riêng lẻ. Ngược lại, nếu bạn biết trước truy vấn, để bạn có thể tạo chỉ mục cây B đa cột được điều chỉnh phù hợp, thì vẫn nhanh hơn việc kết hợp nhiều chỉ mục bitmap.

Cho đến nay, điểm yếu lớn nhất của chỉ mục bitmap là khả năng mở rộng chèn, cập nhật và xóa vô lý . Các thao tác ghi đồng thời hầu như không thể thực hiện được. Điều đó không có vấn đề gì trong kho dữ liệu vì các quá trình tải lần lượt được lên lịch. Trong các ứng dụng trực tuyến, chỉ mục bitmap hầu như vô dụng.

Quan trọng

Các chỉ mục bitmap hầu như không sử dụng được cho giao dịch chuyên nghiệp trực tuyến (OLTP).

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Nhiều sản phẩm cơ sở dữ liệu cung cấp giải pháp kết hợp giữa chỉ mục B-tree và bitmap. Trong trường hợp không có đường dẫn truy cập tốt hơn, họ chuyển đổi kết quả của một số lần quét cây B thành cấu trúc bitmap trong bộ nhớ. Chúng có thể được kết hợp một cách hiệu quả. Các cấu trúc bitmap không được lưu trữ liên tục mà bị loại bỏ sau khi thực hiện câu lệnh, do đó bỏ qua vấn đề về khả năng mở rộng ghi kém. Nhược điểm là nó cần nhiều bộ nhớ và thời gian của CPU. Xét cho cùng, phương pháp này là một hành động tuyệt vọng của người tối ưu hóa.

Chỉ mục một phần

Cho đến nay chúng ta chỉ thảo luận về những cột nào để thêm vào một chỉ mục. Với chỉ mục một phần (PostgreSQL) hoặc được lọc (SQL Server), bạn cũng có thể chỉ định các hàng được lập chỉ mục.

> Thận trọng
>
> Cơ sở dữ liệu Oracle có một cách tiếp cận duy nhất để lập chỉ mục một phần. Phần tiếp theo giải thích nó trong khi xây dựng dựa trên phần này.

Chỉ mục một phần rất hữu ích khi được sử dụng phổ biến trong đó các điều kiện sử dụng giá trị không đổi – như mã trạng thái trong ví dụ sau:

```
CHỌN thông báo TỪ
   các tin nhắn WHERE
 đã xử lý = 'N'
    VÀ người nhận =?
```

Các truy vấn như thế này rất phổ biến trong hệ thống xếp hàng. Truy vấn tìm nạp tất cả các thư chưa được xử lý cho một người nhận cụ thể. Các tin nhắn đã được xử lý hiếm khi cần đến. Nếu cần, chúng thường được truy cập bằng một tiêu chí cụ thể hơn như khóa chính.

Chúng tôi có thể tối ưu hóa truy vấn này với chỉ mục hai cột. Chỉ xem xét truy vấn này, thứ tự cột không quan trọng vì không có điều kiện phạm vi.

```
TẠO INDEX messages_todo
               BẬT tin nhắn (người nhận, đã xử lý)
```

Chỉ mục đáp ứng mục đích của nó, nhưng nó bao gồm nhiều hàng không bao giờ được tìm kiếm, cụ thể là tất cả các thông báo đã được xử lý. Tuy nhiên, do khả năng mở rộng logarit, chỉ mục làm cho truy vấn rất nhanh mặc dù nó lãng phí rất nhiều dung lượng ổ đĩa.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Với lập chỉ mục một phần, bạn có thể giới hạn chỉ mục chỉ bao gồm các thư chưa được xử lý. Cú pháp cho điều này đơn giản một cách đáng ngạc nhiên: một mệnh đề where .

```
TẠO INDEX messages_todo
            BẬT tin nhắn (người nhận)
        WHERE đã xử lý = 'N'
```

Chỉ mục chỉ chứa các hàng thỏa mãn mệnh đề where . Trong trường hợp cụ thể này, chúng tôi thậm chí có thể xóa cột ĐÃ ĐƯỢC XỬ LÝ vì dù sao thì cột này luôn là 'N' . Điều đó có nghĩa là chỉ mục giảm kích thước của nó theo hai chiều: theo chiều dọc, vì nó chứa ít hàng hơn; theo chiều ngang, do cột bị loại bỏ.

Chỉ số này do đó rất nhỏ. Đối với một hàng đợi, nó thậm chí có thể có nghĩa là kích thước chỉ mục vẫn không thay đổi mặc dù bảng phát triển không giới hạn. Chỉ mục không chứa tất cả các thư, chỉ là những thư chưa được xử lý.

Mệnh đề where của một chỉ mục một phần có thể trở nên phức tạp tùy ý. Hạn chế cơ bản duy nhất là về các hàm: bạn chỉ có thể sử dụng các hàm xác định như trường hợp ở mọi nơi trong định nghĩa chỉ mục. Tuy nhiên, SQL Server có nhiều quy tắc hạn chế hơn và không cho phép các hàm cũng như toán tử OR trong các vị từ chỉ mục.

Cơ sở dữ liệu có thể sử dụng chỉ mục một phần bất cứ khi nào mệnh đề where xuất hiện trong một truy vấn.

Hãy nghĩ về nó

Đặc biệt nào có chỉ số nhỏ nhất có thể cho truy vấn sau:

```
CHỌN tin nhắn
   TỪ tin nhắn
 WHERE đã xử lý = 'N';
```


Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

NULL trong Cơ sở dữ liệu Oracle

SQL's NULL thường gây ra nhầm lẫn. Mặc dù ý tưởng cơ bản của NULL – để đại diện cho dữ liệu bị thiếu – khá đơn giản, có một số đặc thù. Bạn chẳng hạn phải sử dụng IS NULL thay vì = NULL . Hơn nữa Oracle cơ sở dữ liệu có thêm NULL kỳ lạ, một mặt vì nó không phải lúc nào cũng xử lý NULL theo yêu cầu của tiêu chuẩn và mặt khác bởi vì nó có cách xử lý NULL rất "đặc biệt" trong các chỉ mục.

Tiêu chuẩn SQL không xác định NULL như một giá trị mà là một trình giữ chỗ cho một giá trị bị thiếu hoặc không xác định. Do đó, không có giá trị nào có thể là NULL. Thay vào đó, cơ sở dữ liệu Oracle xử lý một chuỗi trống là NULL:

```
     LỰA CHỌN        '0 LÀ KHÔNG ???' NHƯ "NULL là gì?" TỪ kép
       Ở ĐÂU            0 LÀ KHÔNG ĐỦ
ĐOÀN KẾT TẤT CẢ
     LỰA CHỌN       '0 không rỗng' Từ kép
       Ở ĐÂU           0 KHÔNG ĐẦY ĐỦ
ĐOÀN KẾT TẤT CẢ
                '' '' '
     LỰA CHỌN          CÓ ĐẦY ĐỦ KHÔNG ??? ' TỪ kép
                    ''
       Ở ĐÂU           LÀ KHÔNG
ĐOÀN KẾT TẤT CẢ
                '' '' '
     LỰA CHỌN          không rỗng 'FROM kép
                    ''
       Ở ĐÂU           KHÔNG ĐẦY ĐỦ;

NULL là gì?
--------------

0 không rỗng
''
     LÀ KHÔNG ???
```

Để thêm vào sự nhầm lẫn, thậm chí có một trường hợp khi cơ sở dữ liệu Oracle coi NULL là chuỗi trống:

```
CHỌN hình nộm            ''
        , hình nộm
        || , hình nộm || VÔ GIÁ TRỊ
   TỪ kép;

DDD
---
XXX
```

Nối cột DUMMY (luôn chứa 'X') với NULL nên trả về NULL.

Machine Translated by Google

Khái niệm NULL được sử dụng trong nhiều ngôn ngữ lập trình. Bất kể bạn nhìn ở đâu, một chuỗi rỗng không bao giờ là NULL. ngoại trừ trong cơ sở dữ liệu Oracle. Trên thực tế, không thể lưu trữ một chuỗi trống trong trường VARCHAR2 . Nếu bạn cố gắng, cơ sở dữ liệu Oracle chỉ lưu trữ NULL.

Đặc thù này không chỉ kỳ lạ; nó cũng nguy hiểm. Ngoài ra, sự kỳ lạ NULL của cơ sở dữ liệu Oracle không dừng lại ở đây - nó tiếp tục với việc lập chỉ mục.

Lập chỉ mục NULL

Cơ sở dữ liệu Oracle không bao gồm các hàng trong một chỉ mục nếu tất cả các cột được lập chỉ mục là NULL. Điều đó có nghĩa là mọi chỉ mục đều là chỉ mục một phần - giống như có mệnh đề where :

```
TẠO idx INDEX
            TRÊN tbl (A, B, C, ...)
        NƠI A KHÔNG ĐỦ HOẶC B KHÔNG
            ĐỦ HOẶC C KHÔNG ĐỦ

                . . .
```

Xem xét chỉ mục EMP_DOB . Nó chỉ có một cột: DATE_OF_BIRTH. Hàng không có giá trị DATE_OF_BIRTH sẽ không được thêm vào chỉ mục này.

```
CHÈN VÀO nhân viên (công ty con, tên_người_đầu_tên, số_điện
                          , thoại)          , họ

                  GIÁ TRỊ (?,?,?,?,?);
```

Câu lệnh chèn không đặt DATE_OF_BIRTH nên nó được đặt mặc định là NULL— do đó, bản ghi không được thêm vào chỉ mục EMP_DOB . Do đó, chỉ mục không thể hỗ trợ truy vấn cho các bản ghi mà DATE_OF_BIRTH LÀ KHÔNG ĐỦ:

```
CHỌN first_name, last_name
  TỪ nhân viên
 WHERE date_of_birth LÀ KHÔNG CÓ;
```


Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google


```
| Id | Hoạt động                      | Tên             | Hàng | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO | | | * 1                                | 477 |
        | BẢNG TRUY CẬP ĐẦY ĐỦ | TUYỂN NHÂN VIÊN |              1 1  | 477 |


Thông tin vị ngữ (được xác định bằng id hoạt động):

     1 - bộ lọc ("DATE_OF_BIRTH" LÀ KHÔNG CÓ)
```

Tuy nhiên, bản ghi được chèn vào một chỉ mục được nối nếu ít nhất một cột chỉ mục không phải là NULL:

```
TẠO INDEX demo_null
               ON nhân viên (công_tân, ngày_thành_sinh);
```

Hàng đã tạo ở trên được thêm vào chỉ mục vì SUBSIDIARY_ID là không NULL. Do đó, chỉ mục này có thể hỗ trợ một truy vấn cho tất cả nhân viên của một công ty con không có giá trị DATE_OF_BIRTH :

```
CHỌN first_name, last_name
   TỪ nhân viên
 WHERE công ty con =?
    AND date_of_birth LÀ KHÔNG CÓ;

| Id | Hoạt động                                    | Tên             | Hàng | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO | |                        |                        1                2 |
    1   | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | TUYỂN NHÂN VIÊN |                1                2 |
| * 2 | QUÉT NGUỒN CHỈ SỐ                            | DEMO_NULL |            1                1   |


Thông tin vị ngữ (được xác định bằng id hoạt động):

     2 - quyền truy cập ("SUBSIDIARY_ID" = TO_NUMBER (?)
               VÀ "DATE_OF_BIRTH" LÀ KHÔNG)
```

Xin lưu ý rằng chỉ mục bao gồm toàn bộ mệnh đề where ; tất cả các bộ lọc được sử dụng làm vị từ truy cập trong QUÉT NGUỒN CHỈ SỐ.

Chúng tôi có thể mở rộng khái niệm này cho truy vấn ban đầu để tìm tất cả các bản ghi trong đó DATE_OF_BIRTH LÀ ĐẦY ĐỦ. Vì vậy, cột DATE_OF_BIRTH phải là cột cột ngoài cùng bên trái trong chỉ mục để nó có thể được sử dụng làm vị từ truy cập. Mặc dù chúng tôi không cần cột chỉ mục thứ hai cho chính truy vấn, chúng tôi thêm một cột khác không bao giờ có thể là NULL để đảm bảo chỉ mục có tất cả các hàng.

Machine Translated by Google

Chúng tôi có thể sử dụng bất kỳ cột nào có ràng buộc NOT NULL , như SUBSIDIARY_ID, cho mục đích đó.

Ngoài ra, chúng ta có thể sử dụng một biểu thức hằng không bao giờ có thể là NULL. Điều đó đảm bảo chỉ mục có tất cả các hàng – ngay cả khi DATE_OF_BIRTH là NULL.

```
DROP INDEX emp_dob;
TẠO INDEX emp_dob ON nhân viên (date_of_birth, '1');
```

Về mặt kỹ thuật, chỉ số này là một chỉ số dựa trên chức năng. Ví dụ này cũng chứng minh sai lầm rằng cơ sở dữ liệu Oracle không thể lập chỉ mục NULL.

Mẹo

Thêm một cột không thể là NULL vào chỉ mục NULL giống như bất kỳ giá trị nào.

Ràng buộc KHÔNG ĐẦY ĐỦ

Để lập chỉ mục một điều kiện IS NULL trong cơ sở dữ liệu Oracle, chỉ mục phải có một cột không bao giờ có thể là NULL.

Điều đó nói rằng không đủ nếu không có mục nhập NULL . Cơ sở dữ liệu phải chắc chắn rằng không bao giờ có mục nhập NULL , nếu không cơ sở dữ liệu phải giả định rằng bảng có các hàng không có trong chỉ mục.

Chỉ mục sau chỉ hỗ trợ truy vấn nếu cột LAST_NAME có ràng buộc NOT NULL :

```
DROP INDEX emp_dob;
TẠO INDEX emp_dob_name
            ON nhân viên (ngày_tháng_ sinh, họ_vụ);

LỰA CHỌN *
   TỪ nhân viên
 WHERE date_of_birth LÀ KHÔNG CÓ;
```


Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google


```
---------------------------------------------------------------------------
| Id | Hoạt động                                  | Tên                  | Hàng | Chi phí |
---------------------------------------------------------------------------
| 0 | TUYỂN CHỌN BÁO CÁO | | | 1                                                  |  1  |        3 |
      | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | TUYỂN NHÂN VIÊN |                        |  1  |        3 |
| * 2 | QUÉT NGUỒN CHỈ SỐ                          | EMP_DOB_NAME |                 |  1  |        2 |
---------------------------------------------------------------------------


Thông tin vị ngữ (được xác định bằng id hoạt động):
---------------------------------------------------

    2 - quyền truy cập ("DATE_OF_BIRTH" LÀ KHÔNG ĐỦ)
```

Loại bỏ ràng buộc NOT NULL làm cho chỉ mục không thể sử dụng được cho việc này truy vấn:

```
Nhân viên ALTER TABLE MODIFY last_name NULL;

LỰA CHỌN *
   TỪ nhân viên
 WHERE date_of_birth LÀ KHÔNG CÓ;

---------------------------------------------------------------------
| Id | Hoạt động                        | Tên              | Hàng | Chi phí |
---------------------------------------------------------------------
| 0 | TUYỂN CHỌN BÁO CÁO | | | * 1                                  | 477 |
        | BẢNG TRUY CẬP ĐẦY ĐỦ | TUYỂN NHÂN VIÊN |               1 1  | 477 |
---------------------------------------------------------------------
```

Mẹo

Thiếu ràng buộc NOT NULL có thể ngăn việc sử dụng chỉ mục trong Oracle cơ sở dữ liệu — đặc biệt cho các truy vấn đếm (*) .

Bên cạnh các ràng buộc NOT NULL , cơ sở dữ liệu cũng biết rằng hằng số các biểu thức như trong phần trước không thể trở thành NULL.

Tuy nhiên, một chỉ mục trên một chức năng do người dùng xác định không áp đặt NOT NULL ràng buộc đối với biểu thức chỉ mục:

```
TẠO HOẶC THAY THẾ hộp đen CHỨC NĂNG (id TRONG SỐ) QUAY LẠI SỐ
XÁC ĐỊNH
ĐÃ BẮT ĐẦU
     RETURN id;
CHẤM DỨT;


DROP INDEX emp_dob_name;
TẠO INDEX emp_dob_bb
      ON nhân viên (date_of_birth, blackbox (worker_id));
```

Machine Translated by Google


```
LỰA CHỌN *
   TỪ nhân viên
 WHERE date_of_birth LÀ KHÔNG CÓ;

| Id | Hoạt động                      | Tên             | Hàng | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO | | | * 1                                | 477 |
        | BẢNG TRUY CẬP ĐẦY ĐỦ | TUYỂN NHÂN VIÊN |              1 1  | 477 |
```

Tên hàm BLACKBOX nhấn mạnh thực tế là trình tối ưu hóa có không có ý tưởng về những gì chức năng làm. Chúng ta có thể thấy rằng hàm vượt qua giá trị đầu vào trực tiếp, nhưng đối với cơ sở dữ liệu, nó chỉ là một hàm trả về một số. Thuộc tính NOT NULL của tham số bị mất. Mặc dù chỉ mục phải có tất cả các hàng, cơ sở dữ liệu không biết điều đó nên nó không thể sử dụng chỉ mục cho truy vấn.

Nếu bạn biết rằng hàm không bao giờ trả về NULL, như trong ví dụ này, bạn có thể thay đổi truy vấn để phản ánh rằng:

```
LỰA CHỌN *
   TỪ nhân viên
 WHERE date_of_birth LÀ KHÔNG CÓ
    VÀ hộp đen (worker_id) KHÔNG ĐẦY ĐỦ;

| Id | Hoạt động                                | Tên               | Hàng | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO | | 1                     |                          1          3 |
      | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | TUYỂN NHÂN VIÊN |                  1          3 |
| * 2 | QUÉT NGUỒN CHỈ SỐ                        | EMP_DOB_BB |              1          2 |
```

Điều kiện bổ sung trong mệnh đề where luôn đúng và do đó không thay đổi kết quả. Tuy nhiên, cơ sở dữ liệu Oracle nhận ra rằng bạn chỉ truy vấn các hàng phải có trong chỉ mục cho mỗi định nghĩa.

Thật không may, không có cách nào để gắn thẻ một hàm không bao giờ trả về NULL nhưng bạn có thể di chuyển lệnh gọi hàm đến một cột ảo (kể từ 11g) và đặt một ràng buộc KHÔNG ĐẦY ĐỦ trên cột này.

```
ALTER TABLE nhân viên THÊM bb_expression
         ĐƯỢC PHÁT SINH LUÔN NHƯ (hộp đen (worker_id)) KHÔNG ĐỦ;

DROP INDEX emp_dob_bb;
TẠO INDEX emp_dob_bb
      ON nhân viên (ngày_tháng_sinh, biểu_thức);
```


Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google


```
LỰA CHỌN *
   TỪ nhân viên
 WHERE date_of_birth LÀ KHÔNG CÓ;

| Id | Hoạt động                                  | Tên                | Hàng | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO | | 1                    |                                  1 |        3 |
      | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | TUYỂN NHÂN VIÊN |                  1 |        3 |
| * 2 | QUÉT NGUỒN CHỈ SỐ                          | EMP_DOB_BB |                  1 |        2 |
```

Cơ sở dữ liệu Oracle biết rằng một số hàm nội bộ chỉ trả về NULL nếu NULL được cung cấp làm đầu vào.

```
DROP INDEX emp_dob_bb;

TẠO INDEX emp_dob_upname
      ON nhân viên (ngày_tháng_ sinh, trên (họ_vụ));

LỰA CHỌN *
   TỪ nhân viên
 WHERE date_of_birth LÀ KHÔNG CÓ;

| Id | Hoạt động                                  | Tên                     | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO | | |                                                        3 |
   1  | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | TUYỂN NHÂN VIÊN |                        3 |
| * 2 | QUÉT NGUỒN CHỈ SỐ                          | EMP_DOB_UPNAME |               2 |
```

Hàm UPPER bảo toàn thuộc tính NOT NULL của LAST_NAME cột. Tuy nhiên, việc loại bỏ ràng buộc sẽ làm cho chỉ mục không thể sử dụng được:

```
Nhân viên ALTER TABLE MODIFY last_name NULL;

LỰA CHỌN *
   TỪ nhân viên
 WHERE date_of_birth LÀ KHÔNG CÓ;

| Id | Hoạt động                        | Tên              | Hàng | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO | | | * 1                                  | 477 |
        | BẢNG TRUY CẬP ĐẦY ĐỦ | TUYỂN NHÂN VIÊN |              1 1  | 477 |
```

Machine Translated by Google

Mô phỏng các chỉ mục một phần

Cách kỳ lạ mà cơ sở dữ liệu Oracle xử lý NULL trong các chỉ mục có thể được sử dụng để mô phỏng các chỉ mục từng phần. Đối với điều đó, chúng ta chỉ cần sử dụng NULL cho các hàng không nên được lập chỉ mục.

Để chứng minh, chúng tôi mô phỏng chỉ mục từng phần sau:

```
TẠO INDEX messages_todo
              BẬT tin nhắn (người nhận)
          WHERE đã xử lý = 'N'
```

Đầu tiên, chúng ta cần một hàm chỉ trả về giá trị RECEIVER nếu giá trị PROCESSED là 'N'.

```
TẠO HOẶC THAY THẾ
FUNCTION pi_processed (CHAR đã xử lý, người nhận NUMBER)
QUAY LẠI SỐ
XÁC ĐỊNH
NHƯ BẮT ĐẦU
     NẾU được xử lý VÀO ('N') THEN
         RETURN bộ thu;
     KHÁC
         QUAY LẠI NULL;
     HẾT NẾU;
CHẤM
DỪT; /
```

Hàm phải có tính xác định để nó có thể được sử dụng trong định nghĩa chỉ mục.

Bây giờ chúng ta có thể tạo một chỉ mục chỉ chứa các hàng có PROCESSED = 'N'.

```
TẠO INDEX messages_todo
              ON thông báo (pi_processed (đã xử lý, người nhận));
```


Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Để sử dụng chỉ mục, bạn phải sử dụng biểu thức được lập chỉ mục trong truy vấn:

```
CHỌN tin nhắn
   TỪ tin nhắn
 WHERE pi_processed (đã xử lý, người nhận) =?

--------------------------------------------------------------------------------

| Id | Hoạt động                                    | Tên                    | Chi phí |
--------------------------------------------------------------------------------

| 0 | TUYỂN CHỌN BÁO CÁO | | 5330 |
| 1   | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | THÔNG ĐIỆP | 5330 |
| * 2 | QUÉT NGUỒN CHỈ SỐ                            | MESSAGES_TODO | 5303 |
--------------------------------------------------------------------------------


Thông tin vị ngữ (được xác định bằng id hoạt động):
---------------------------------------------------

     2 - truy cập ("PI_PROCESSED" ("PROCESSED", "RECEIVER") =: X)
```

Chỉ mục từng phần, Phần II

Kể từ khi phát hành 11g, có một cách tiếp cận thứ hai - cũng đáng sợ không kém - để mô phỏng các chỉ mục từng phần trong cơ sở dữ liệu Oracle bằng cách sử dụng phân vùng chỉ mục bị hỏng cố ý và SKIP_UNUSABLE_INDEX tham số.

Machine Translated by Google

Điều kiện bị xáo trộn

Các phần sau đây trình bày một số phương pháp phổ biến cho các điều kiện xáo trộn. Điều kiện xáo trộn là nơi các mệnh đề được diễn giải theo cách ngăn cản việc sử dụng chỉ mục thích hợp. Phần này là tập hợp các mô hình chống mà mọi nhà phát triển nên biết và tránh.

Loại ngày

Hầu hết các xáo trộn liên quan đến các loại DATE . Cơ sở dữ liệu Oracle đặc biệt dễ bị tấn công về mặt này vì nó chỉ có một kiểu DATE luôn bao gồm cả thành phần thời gian.

Việc sử dụng hàm TRUNC để loại bỏ thành phần thời gian đã trở thành thông lệ phổ biến . Trên thực tế, nó không loại bỏ thời gian mà thay vào đó đặt nó thành nửa đêm vì cơ sở dữ liệu Oracle không có kiểu DATE thuần túy . Để bỏ qua thành phần thời gian cho một tìm kiếm, bạn có thể sử dụng hàm TRUNC ở cả hai phía của so sánh — ví dụ: để tìm kiếm doanh số bán hàng của ngày hôm qua:

```
LỰA CHỌN ...
   TỪ bán hàng
 WHERE TRUNC (sale_date) = TRUNC (sysdate - INTERVAL '1' DAY)
```

Đây là một câu lệnh hoàn toàn hợp lệ và chính xác nhưng nó không thể sử dụng một chỉ mục đúng cách vào SALE_DATE. Nó được giải thích trong “Tìm kiếm không phân biệt chữ hoa chữ thường bằng cách sử dụng UPPER hoặc LOWER” trên trang 24; TRUNC (sale_date) là một cái gì đó hoàn toàn khác với SALE_DATE — các hàm là hộp đen đối với cơ sở dữ liệu.

Có một giải pháp khá đơn giản cho vấn đề này: một chỉ mục dựa trên hàm.

```
TẠO INDEX index_name
           ON table_name (TRUNC (sale_date))
```

Nhưng sau đó bạn phải luôn sử dụng TRUNC (date_column) trong mệnh đề where .
Nếu bạn sử dụng nó không nhất quán — đôi khi có, đôi khi không có TRUNC — thì bạn cần hai chỉ mục!

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Sự cố cũng xảy ra với cơ sở dữ liệu có kiểu ngày thuần túy nếu bạn tìm kiếm trong một khoảng thời gian dài hơn như được hiển thị trong truy vấn MySQL sau:

```
LỰA CHỌN ...
   TỪ bán hàng
 WHERE DATE_FORMAT (sale_date, "% Y-% M")
      = DATE_FORMAT (bây giờ () , "% Y-% M ')
```

Truy vấn sử dụng định dạng ngày chỉ chứa năm và tháng: một lần nữa, đây là một truy vấn hoàn toàn chính xác có cùng vấn đề như trước đây.
Tuy nhiên giải pháp từ trên không áp dụng ở đây vì MySQL không có chỉ mục dựa trên hàm.

Giải pháp thay thế là sử dụng điều kiện phạm vi rõ ràng. Đây là một giải pháp chung hoạt động cho tất cả các cơ sở dữ liệu:

```
LỰA CHỌN ...
   TỪ bán hàng
 WHERE sale_date BETWEEN quý_begin (?)
                         AND 1/4_end (?)
```

Nếu bạn đã làm bài tập về nhà của mình, bạn có thể nhận ra mô hình từ bài tập về tất cả nhân viên 42 tuổi.

Chỉ mục thẳng vào SALE_DATE là đủ để tối ưu hóa truy vấn này. Các hàm QUARTER_BEGIN và QUARTER_END tính toán các ngày ranh giới.
Việc tính toán có thể trở nên phức tạp một chút vì toán tử giữa luôn bao gồm các giá trị biên. Do đó, hàm QUARTER_END phải trả về dấu thời gian ngay trước ngày đầu tiên của quý tiếp theo nếu SALE_DATE có thành phần thời gian. Logic này có thể được ẩn trong hàm.

Các ví dụ trên các trang sau đây cho thấy cách triển khai các hàm QUARTER_BEGIN và QUARTER_END cho các cơ sở dữ liệu khác nhau.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

MySQL

```
TẠO FUNCTION phần tư_nguồn (dt DATETIME)
QUAY LẠI GIỜ XÁC ĐỊNH NGÀY
QUAY LẠI CHUYỂN ĐỔI
          (
            CONCAT
            (CHUYỂN ĐỔI (NĂM (dt), CHAR (4))
              '-'
            ,
            , CHUYỂN ĐỔI (QUARTER (dt) * 3-2, CHAR (2))
            ,  '-01'
            )
          , ngày giờ
          );

TẠO FUNCTION phần tư (cuối ngày) (dt DATETIME)
QUAY LẠI GIỜ XÁC ĐỊNH NGÀY
QUAY LẠI DATE_ADD
          (DATE_ADD (1/4_begin (dt), INTERVAL 3 THÁNG)
          , INTERVAL -1 MICROSECOND);
```

Cơ sở dữ liệu Oracle

```
TẠO FUNCTION số tiền ký quỹ (dt TRONG NGÀY)
NGÀY TRỞ LẠI
BẰNG
BẮT ĐẦU
    QUAY TRUNC (dt, 'Q');
CHẤM DỨT;
/

TẠO FUNCTION phần tư_nguồn (dt TRONG NGÀY)
NGÀY TRỞ LẠI
BẰNG
BẮT ĐẦU
    - loại Oracle DATE có độ phân giải giây
    - trừ một giây cho ngày đầu tiên của quý tiếp theo

    QUAY LẠI TRUNC (ADD_MONTHS (dt, +3), 'Q') - (1 /
           (24 * 60 * 60));
CHẤM DỨT;
/
```


Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

PostgreSQL

```
TẠO FUNCTION 1/4_begin (dấu thời gian dt với múi giờ)
RETURNS dấu thời gian với múi giờ AS $$
BẮT ĐẦU
      RETURN date_trunc ('quý', dt);
CHẤM DỨT;
$$ LANGUAGE plpgsql;

TẠO FUNCTION cuối cùng (dấu thời gian dt với múi giờ)
RETURNS dấu thời gian với múi giờ AS $$
BẮT ĐẦU
    RETURN date_trunc ('quý', dt) + khoảng thời gian
              '3 tháng' - khoảng thời gian '1 micro
              giây';
CHẤM DỨT;
$$ LANGUAGE plpgsql;
```

Máy chủ SQL

```
TẠO FUNCTION 1/4_begin (@dt DATETIME)
QUAY LẠI NGÀY
BẮT ĐẦU
   QUAY LẠI DATEADD (qq, DATEDIFF (qq, 0, @dt), 0)
CHẤM DỨT
ĐI

TẠO FUNCTION Quarter_end (@dt DATETIME)
QUAY LẠI NGÀY
BẮT ĐẦU
   QUAY LẠI DATEADD
             (mili giây
             ,  -3
             , DATEADD (mm, 3, dbo.quarter_begin (@dt)));

CHẤM DỨT
ĐI
```

Bạn có thể sử dụng các hàm bổ trợ tương tự cho các khoảng thời gian khác – hầu hết chúng sẽ ít phức tạp hơn các ví dụ trên, đặc biệt khi sử dụng các điều kiện lớn hơn hoặc bằng (> =) và nhỏ hơn (<) thay vì toán tử giữa . Tất nhiên, bạn có thể tính toán các ngày ranh giới trong đơn đăng ký của mình nếu bạn muốn.

Machine Translated by Google

Mẹo

Viết truy vấn cho các khoảng thời gian liên tục dưới dạng điều kiện phạm vi rõ ràng. Làm điều này ngay cả trong một ngày – ví dụ: đối với cơ sở dữ liệu Oracle:

```
      sale_date> = TRUNC (sysdate)
AND sale_date <TRUNC (sysdate + INTERVAL '1' DAY)
```

Một sự xáo trộn phổ biến khác là so sánh ngày tháng dưới dạng chuỗi như được hiển thị trong ví dụ PostgreSQL sau:

```
LỰA CHỌN ...
   TỪ bán hàng
 WHERE TO_CHAR (sale_Date, 'YYYY-MM-DD') = '1970-01-01'
```

Một lần nữa, vấn đề là chuyển đổi DATE_COLUMN. Các điều kiện như vậy thường được tạo ra với niềm tin rằng bạn không thể chuyển các kiểu khác với số và chuỗi vào cơ sở dữ liệu. Tuy nhiên, tham số ràng buộc hỗ trợ tất cả các kiểu dữ liệu. Điều đó có nghĩa là bạn có thể ví dụ sử dụng đối tượng java.util.Date làm tham số ràng buộc. Đây là một lợi ích khác của các tham số ràng buộc.

Nếu bạn không thể làm điều đó, bạn chỉ cần chuyển đổi cụm từ tìm kiếm thay vì cột bảng:

```
LỰA CHỌN ...
   TỪ bán hàng
 WHERE sale_date = TO_DATE ('1970-01-01', 'YYYY-MM-DD')
```

Truy vấn này có thể sử dụng một chỉ mục thẳng vào SALE_DATE. Hơn nữa, nó chuyển đổi chuỗi đầu vào chỉ một lần. Câu lệnh trước đó phải chuyển đổi tất cả các ngày được lưu trữ trong bảng trước khi nó có thể so sánh chúng với cụm từ tìm kiếm.

Bất kỳ thay đổi nào bạn thực hiện – sử dụng tham số liên kết hoặc chuyển đổi mặt khác của so sánh – bạn có thể dễ dàng phát sinh lỗi nếu SALE_DATE có thành phần thời gian. Bạn phải sử dụng điều kiện phạm vi rõ ràng trong đó trường hợp:

```
LỰA CHỌN ...
   TỪ bán hàng
 WHERE sale_date> = TO_DATE ('1970-01-01', 'YYYY-MM-DD')
   AND sale_date <TO_DATE ('1970-01-01', 'YYYY-MM-DD')
                      + INTERVAL '1' NGÀY
```

Luôn cân nhắc sử dụng điều kiện phạm vi rõ ràng khi so sánh ngày.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

LIKE trên các loại ngày

Cách xử lý sau đặc biệt phức tạp:

```
sale_date LIKE SYSDATE
```

Thoạt nhìn, nó không giống như một sự xáo trộn vì nó không sử dụng bất kỳ chức năng nào.

Tuy nhiên , toán tử LIKE thực thi so sánh chuỗi.
Tùy thuộc vào cơ sở dữ liệu, điều đó có thể gây ra lỗi hoặc gây ra chuyển đổi kiểu ngầm định ở cả hai phía. Phần "Thông tin dự đoán" của kế hoạch thực thi hiển thị những gì cơ sở dữ liệu Oracle thực hiện:

```
bộ lọc (INTERNAL_FUNCTION (SALE_DATE)
    THÍCH TO_CHAR (SYSDATE @!))
```

Hàm INTERNAL_FUNCTION chuyển đổi kiểu của cột SALE_DATE . Như một tác dụng phụ, nó cũng ngăn việc sử dụng chỉ mục thẳng trên DATE_COLUMN giống như bất kỳ hàm nào khác.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Chuỗi số

Chuỗi số là các số được lưu trữ trong các cột văn bản. Mặc dù đây là một thực hành rất tệ, nhưng nó không tự động làm cho một chỉ mục trở nên vô dụng nếu bạn luôn coi nó là chuỗi:

```
LỰA CHỌN ...
  TỪ    ...
 WHERE numeric_string = '42'
```

Tất nhiên câu lệnh này có thể sử dụng chỉ mục trên NUMERIC_STRING. Tuy nhiên, nếu bạn so sánh nó bằng một số, thì cơ sở dữ liệu không còn có thể sử dụng điều kiện này như một vị từ truy cập nữa.

```
LỰA CHỌN ...
  TỪ    ...
 WHERE numeric_string = 42
```

Lưu ý các dấu ngoặc kép còn thiếu. Mặc dù một số cơ sở dữ liệu mang lại lỗi (ví dụ: PostgreSQL) nhiều cơ sở dữ liệu chỉ thêm một chuyển đổi kiểu ngầm.

```
LỰA CHỌN ...
  TỪ    ...
 WHERE TO_NUMBER (numeric_string) = 42
```

Đó là vấn đề tương tự như trước đây. Không thể sử dụng chỉ mục trên NUMERIC_STRING do lệnh gọi hàm. Giải pháp cũng giống như trước: không chuyển đổi cột bảng, thay vào đó chuyển đổi cụm từ tìm kiếm.

```
LỰA CHỌN ...
  TỪ    ...
 WHERE numeric_string = TO_CHAR (42)
```

Bạn có thể thắc mắc tại sao cơ sở dữ liệu không tự động làm theo cách này? Đó là bởi vì việc chuyển đổi một chuỗi thành một số luôn cho kết quả rõ ràng. Điều này không đúng theo chiều ngược lại. Một số, được định dạng dưới dạng văn bản, có thể chứa khoảng trắng, dấu chấm câu và số không ở đầu. Một giá trị duy nhất có thể được viết theo nhiều cách:

```
42
042
0042 00042
 ...
```


Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Cơ sở dữ liệu không thể biết định dạng số được sử dụng trong cột NUMERIC_STRING vì vậy nó thực hiện theo cách khác: cơ sở dữ liệu chuyển đổi các chuỗi thành số – đây là một chuyển đổi rõ ràng.

Hàm TO_CHAR chỉ trả về một biểu diễn chuỗi của số.
Do đó, nó sẽ chỉ khớp với chuỗi đầu tiên được liệt kê ở trên. Nếu chúng tôi sử dụng TO_NUMBER, nó khớp với tất cả chúng. Điều đó có nghĩa là không chỉ có sự khác biệt về hiệu suất giữa hai biến thể mà còn có sự khác biệt về ngữ nghĩa!

Việc sử dụng các chuỗi số nói chung là rắc rối: quan trọng nhất là nó gây ra các vấn đề về hiệu suất do chuyển đổi ngầm và cũng dẫn đến nguy cơ gặp phải lỗi chuyển đổi do các số không hợp lệ. Ngay cả truy vấn nhỏ nhất không sử dụng bất kỳ hàm nào trong mệnh đề where cũng có thể gây ra sự hủy bỏ với lỗi chuyển đổi nếu chỉ có một số không hợp lệ được lưu trữ trong bảng.

Mẹo

Sử dụng các kiểu số để lưu trữ số.

Lưu ý rằng vấn đề không tồn tại theo chiều ngược lại:

```
LỰA CHỌN  ...
   TỪ     ...
  WHERE numeric_number = '42'
```

Cơ sở dữ liệu sẽ luôn biến đổi chuỗi thành một số. Nó không áp dụng một chức năng trên cột có khả năng được lập chỉ mục: một chỉ mục thông thường do đó sẽ hoạt động. Tuy nhiên, có thể thực hiện chuyển đổi thủ công theo cách sai:

```
LỰA CHỌN  ...
   TỪ     ...
  WHERE TO_CHAR (numeric_number) = '42'
```

Machine Translated by Google

Kết hợp các cột

Phần này nói về một sự xáo trộn phổ biến ảnh hưởng đến các chỉ mục được nối.

Ví dụ đầu tiên là về các loại ngày và giờ nhưng ngược lại. Truy vấn MySQL sau đây kết hợp dữ liệu và cột thời gian để áp dụng bộ lọc phạm vi trên cả hai.

```
LỰA CHỌN  ...
   TỪ     ...
 WHERE ADDTIME (cột ngày tháng, cột thời gian)
      > DATE_ADD (bây giờ (), INTERVAL -1 NGÀY)
```

Nó chọn tất cả các bản ghi từ 24 giờ qua. Truy vấn không thể sử dụng chỉ mục nối vào (DATE_COLUMN, TIME_COLUMN) đúng cách vì tìm kiếm không được thực hiện trên các cột được lập chỉ mục mà trên dữ liệu dẫn xuất.

Bạn có thể tránh sự cố này bằng cách sử dụng kiểu dữ liệu có cả thành phần ngày và giờ (ví dụ: MySQL DATETIME). Sau đó, bạn có thể sử dụng cột này mà không cần gọi hàm:

```
LỰA CHỌN  ...
   TỪ     ...
 WHERE datetime_column
      > DATE_ADD (bây giờ (), INTERVAL -1 NGÀY)
```

Thật không may, thường không thể thay đổi bảng khi gặp vấn đề này.

Tùy chọn tiếp theo là chỉ mục dựa trên chức năng nếu cơ sở dữ liệu hỗ trợ nó - mặc dù điều này có tất cả các nhược điểm đã được thảo luận trước đây. Khi sử dụng MySQL, các chỉ mục dựa trên hàm không phải là một tùy chọn.

Vẫn có thể viết truy vấn để cơ sở dữ liệu có thể sử dụng chỉ mục được nối vào DATE_COLUMN, TIME_COLUMN với một vị từ truy cập— ít nhất một phần. Đối với điều đó, chúng tôi thêm một điều kiện bổ sung vào DATE_COLUMN.

```
 WHERE ADDTIME (cột ngày tháng, cột thời gian)
      > DATE_ADD (bây giờ (), INTERVAL -1 NGÀY)
   VÀ cột ngày tháng
     > = DATE (DATE_ADD (bây giờ (), INTERVAL -1 NGÀY))
```


Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Điều kiện mới là hoàn toàn thừa nhưng nó là một bộ lọc thẳng trên DATE_COLUMN có thể được sử dụng làm vị từ truy cập. Mặc dù kỹ thuật này không hoàn hảo, nhưng nó thường là một phép gần đúng đủ tốt.

Mẹo

Sử dụng điều kiện dư thừa trên cột quan trọng nhất khi điều kiện phạm vi kết hợp nhiều cột.

Đối với PostgreSQL, bạn nên sử dụng cú pháp giá trị hàng được mô tả trên trang 151.

Bạn cũng có thể sử dụng kỹ thuật này khi lưu trữ ngày và giờ trong các cột văn bản, nhưng bạn phải sử dụng các định dạng ngày và giờ tạo ra thứ tự thời gian khi được sắp xếp theo từ vựng – ví dụ: như được đề xuất bởi ISO 8601 (YYYY-MM-DD HH: MM: SS). Ví dụ sau sử dụng hàm TO_CHAR của cơ sở dữ liệu Oracle cho mục đích đó:

```
LỰA CHỌN  ...
   TỪ     ...
 WHERE date_string || time_string
      > TO_CHAR (ngày tháng - 1, 'YYYY-MM-DD HH24: MI: SS')
    AND date_string
      > = TO_CHAR (sysdate - 1, 'YYYY-MM-DD')
```

Chúng ta sẽ phải đối mặt với vấn đề áp dụng điều kiện phạm vi trên nhiều cột một lần nữa trong phần có tên “Phân trang thông qua kết quả”. Chúng tôi cũng sẽ sử dụng cùng một phương pháp ước lượng để giảm thiểu nó.

Đôi khi chúng ta gặp trường hợp ngược lại và có thể cố ý làm xáo trộn một điều kiện để nó không thể được sử dụng làm vị từ truy cập nữa.
Chúng ta đã xem xét vấn đề đó khi thảo luận về ảnh hưởng của các tham số ràng buộc đối với các điều kiện LIKE . Hãy xem xét ví dụ sau:

```
CHỌN last_name, first_name, worker_id TỪ các nhân
   viên WHERE child_id =?

    VÀ last_name LIKE?
```

Giả sử có một chỉ mục trên SUBSIDIARY_ID và một chỉ mục khác trên LAST_NAME, thì chỉ mục nào tốt hơn cho truy vấn này?

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Nếu không biết vị trí của ký tự đại diện trong cụm từ tìm kiếm, không thể đưa ra câu trả lời đủ điều kiện. Trình tối ưu hóa không có lựa chọn nào khác ngoài "đoán". Nếu bạn biết rằng luôn có một ký tự đại diện đứng đầu, bạn có thể cố ý làm xáo trộn điều kiện LIKE để trình tối ưu hóa không còn có thể xem xét chỉ mục trên LAST_NAME nữa.

```
CHỌN last_name, first_name, worker_id
  TỪ nhân viên
 WHERE công ty con_id =?,,
    AND last_name || GIỐNG ?
```

Chỉ cần nối một chuỗi trống vào cột LAST_NAME là đủ . Tuy nhiên, đây là một lựa chọn cuối cùng. Chỉ thực hiện khi thực sự cần thiết.

Logic thông minh

Một trong những tính năng chính của cơ sở dữ liệu SQL là hỗ trợ của chúng cho các truy vấn đặc biệt: các truy vấn mới có thể được thực thi bất cứ lúc nào. Điều này chỉ có thể thực hiện được vì trình tối ưu hóa truy vấn (công cụ lập kế hoạch truy vấn) hoạt động trong thời gian chạy; nó phân tích từng câu lệnh khi nhận được và tạo ra một kế hoạch thực thi hợp lý ngay lập tức. Chi phí được giới thiệu bởi tối ưu hóa thời gian chạy có thể được giảm thiểu với các tham số ràng buộc.

Ý chính của bản tóm tắt đó là cơ sở dữ liệu được tối ưu hóa cho SQL động— vì vậy hãy sử dụng nó nếu bạn cần.

Tuy nhiên, có một thực tiễn được sử dụng rộng rãi là tránh SQL động thay vì SQL tĩnh — thường là do lầm tưởng “SQL động là chậm”. Thực hành này có hại nhiều hơn là có lợi nếu cơ sở dữ liệu sử dụng bộ đệm kế hoạch thực thi được chia sẻ như DB2, cơ sở dữ liệu Oracle hoặc SQL Server.

Để minh họa, hãy tưởng tượng một ứng dụng truy vấn bảng NHÂN VIÊN . Ứng dụng cho phép tìm kiếm id công ty con, id nhân viên và họ (không phân biệt chữ hoa chữ thường) trong bất kỳ sự kết hợp nào. Vẫn có thể viết một truy vấn duy nhất bao gồm tất cả các trường hợp bằng cách sử dụng logic "thông minh".

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google


```
CHỌN first_name, last_name, công ty con, worker_id
   TỪ nhân viên
 WHERE (công ty con =: sub_id HOẶC: sub_id LÀ KHÔNG ĐỦ)
    VÀ (worker_id =: emp_id HOẶC: emp_id LÀ KHÔNG ĐỦ)
    VÀ (LÊN (last_name) =: tên HOẶC: tên LÀ KHÔNG ĐỦ)
```

Truy vấn sử dụng các biến liên kết được đặt tên để dễ đọc hơn. Tất cả các bộ lọc có thể biểu thức được mã hóa tĩnh trong câu lệnh. Bất cứ khi nào bộ lọc không cần thiết, bạn chỉ cần sử dụng NULL thay vì một cụm từ tìm kiếm: nó vô hiệu hóa điều kiện thông qua logic HOẶC .

Đó là một câu lệnh SQL hoàn toàn hợp lý. Việc sử dụng NULL thậm chí còn phù hợp với định nghĩa của nó theo logic ba giá trị của SQL. Tuy nhiên nó là một trong những mô hình chống hiệu suất tồi tệ nhất của tất cả.

Cơ sở dữ liệu không thể tối ưu hóa kế hoạch thực thi cho một bộ lọc cụ thể bởi vì bất kỳ trong số chúng có thể bị hủy bỏ trong thời gian chạy. Cơ sở dữ liệu cần để chuẩn bị cho trường hợp xấu nhất – nếu tất cả các bộ lọc bị tắt:

```
-----------------------------------------------------------------------------
| Id | Hoạt động                        | Tên              | Hàng | Chi phí |
-----------------------------------------------------------------------------
| 0 | TUYỂN CHỌN BÁO CÁO | | | * 1                                2 | 478 |
        | BẢNG TRUY CẬP ĐẦY ĐỦ | TUYỂN NHÂN VIÊN |                2 | 478 |
-----------------------------------------------------------------------------


Thông tin vị ngữ (được xác định bằng id hoạt động):
---------------------------------------------------
1 - bộ lọc ((: TÊN ĐẦY ĐỦ HOẶC LÊN ("LAST_NAME") =: TÊN)
           VÀ (: EMP_ID LÀ NULL HOẶC "EMPLOYEE_ID" =: EMP_ID)
           VÀ (: SUB_ID LÀ KHÔNG ĐỦ HOẶC "SUBSIDIARY_ID" =: SUB_ID))
```

Do đó, cơ sở dữ liệu sử dụng quét toàn bộ bảng ngay cả khi có chỉ mục cho mỗi cột.

Không phải là cơ sở dữ liệu không thể giải quyết logic "thông minh". Nó tạo ra kế hoạch thực thi chung do sử dụng các tham số liên kết để nó có thể được lưu vào bộ nhớ đệm và được sử dụng lại với các giá trị khác sau này. Nếu chúng ta không sử dụng các tham số ràng buộc nhưng viết các giá trị thực tế trong câu lệnh SQL, trình tối ưu hóa sẽ chọn chỉ mục thích hợp cho bộ lọc đang hoạt động:

```
CHỌN first_name, last_name, công ty con, worker_id
   TỪ nhân viên
 WHERE (công ty con_id                = KHÔNG          HOẶC NULL LÀ KHÔNG)
    AND (worker_id                    = KHÔNG          HOẶC NULL LÀ KHÔNG)
    AND ( UPPER (last_name) = 'WINAND' HOẶC 'WINAND' LÀ KHÔNG )
```

Machine Translated by Google


```
| Id | Hoạt động                                  | Tên                | Hàng | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO | | 1                          |                                  |                    2 |
      | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | TUYỂN NHÂN VIÊN |                                  |                    2 |
| * 2 | QUÉT NGUỒN CHỈ SỐ                          | EMP_UP_NAME |                                  |                    1 |


Thông tin vị ngữ (được xác định bằng id hoạt động):

   2 - truy cập (UPPER ("LAST_NAME") = 'WINAND')
```

Tuy nhiên, điều này không phải là giải pháp. Nó chỉ chứng minh rằng cơ sở dữ liệu có thể giải quyết những điều kiện này.

Cảnh báo

Sử dụng các giá trị theo nghĩa đen khiến ứng dụng của bạn dễ bị tấn công bởi SQL tấn công tiêm và có thể gây ra các vấn đề về hiệu suất do tăng chi phí tối ưu hóa.

Giải pháp rõ ràng cho các truy vấn động là SQL động. Dựa theo nguyên tắc KISS[3] , chỉ cần cho cơ sở dữ liệu biết những gì bạn cần ngay bây giờ – và không có gì khác.

```
CHỌN first_name, last_name, công ty con, worker_id
   TỪ nhân viên
  WHERE UPPER (last_name) =: name
```

Lưu ý rằng truy vấn sử dụng một tham số ràng buộc.

Mẹo

Sử dụng SQL động nếu bạn cần mệnh đề where động.

Vẫn sử dụng các tham số liên kết khi tạo SQL động – nếu không huyền thoại "SQL động chậm" trở thành sự thật.

Vấn đề được mô tả trong phần này là phổ biến. Tất cả các cơ sở dữ liệu sử dụng bộ đệm ẩn kế hoạch thực thi được chia sẻ có một tính năng để đối phó với nó – thường giới thiệu các vấn đề và lỗi mới.

[3] http://en.wikipedia.org/wiki/KISS_principle

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

MySQL

MySQL không bị vấn đề cụ thể này vì nó không có bộ đệm kế hoạch thực thi nào cả. Một yêu cầu tính năng từ năm 2009 thảo luận về tác động của bộ nhớ đệm kế hoạch thực thi. Có vẻ như trình tối ưu hóa của MySQL đủ đơn giản để bộ nhớ đệm kế hoạch thực thi không mang lại hiệu quả.

Cơ sở dữ liệu Oracle

Cơ sở dữ liệu Oracle sử dụng bộ đệm kế hoạch thực thi được chia sẻ ("Vùng SQL") và hoàn toàn giải quyết được vấn đề được mô tả trong phần này.

Oracle đã giới thiệu cái gọi là liên kết nhìn trộm với bản phát hành 9i. Tính năng xem liên kết cho phép trình tối ưu hóa sử dụng các giá trị ràng buộc thực tế của lần thực thi đầu tiên khi chuẩn bị một kế hoạch thực thi. Vấn đề với cách tiếp cận này là hành vi không xác định của nó: các giá trị từ lần thực thi đầu tiên ảnh hưởng đến tất cả các lần thực thi. Kế hoạch thực thi có thể thay đổi bất cứ khi nào cơ sở dữ liệu được khởi động lại hoặc ít dự đoán hơn, kế hoạch đã lưu trong bộ nhớ cache hết hạn và trình tối ưu hóa tạo lại nó bằng cách sử dụng các giá trị khác nhau vào lần tiếp theo câu lệnh được thực thi.

Bản phát hành 11g đã giới thiệu tính năng chia sẻ con trỏ thích ứng để cải thiện tình hình hơn nữa. Tính năng này cho phép cơ sở dữ liệu lưu vào bộ đệm nhiều kế hoạch thực thi cho cùng một câu lệnh SQL. Hơn nữa, trình tối ưu hóa xem trước các tham số liên kết và lưu trữ độ chọn lọc ước tính của chúng cùng với kế hoạch thực thi. Khi bộ nhớ cache được truy cập sau đó, tính chọn lọc của các giá trị ràng buộc hiện tại phải nằm trong phạm vi chọn lọc của kế hoạch thực thi được lưu trong bộ nhớ cache để được sử dụng lại. Nếu không, trình tối ưu hóa sẽ tạo một kế hoạch thực thi mới và so sánh nó với các kế hoạch thực thi đã được lưu trong bộ nhớ cache cho truy vấn này. Nếu đã có một kế hoạch thực thi như vậy, cơ sở dữ liệu sẽ thay thế nó bằng một kế hoạch thực thi mới cũng bao gồm các ước tính chọn lọc của các giá trị ràng buộc hiện tại. Nếu không, nó sẽ lưu vào bộ nhớ cache một biến thể kế hoạch thực thi mới cho truy vấn này - tất nhiên là cùng với các ước tính chọn lọc.

PostgreSQL Bộ

đệm kế hoạch truy vấn PostgreSQL chỉ hoạt động đối với các câu lệnh mở — miễn là bạn vẫn mở PreparedStatement . Sự cố được mô tả ở trên chỉ xảy ra khi sử dụng lại một xử lý câu lệnh. Lưu ý rằng trình điều khiển JDBC của PostgresSQL chỉ bật bộ đệm sau lần thực thi thứ năm.

Machine Translated by Google

SQL Server

SQL Server sử dụng cái gọi là đánh giá tham số. Đánh giá tham số cho phép trình tối ưu hóa sử dụng các giá trị ràng buộc thực tế của lần thực thi đầu tiên trong quá trình phân tích cú pháp. Vấn đề với cách tiếp cận này là hành vi không xác định của nó: các giá trị từ lần thực thi đầu tiên ảnh hưởng đến tất cả các lần thực thi. Kế hoạch thực thi có thể thay đổi bất cứ khi nào cơ sở dữ liệu được khởi động lại hoặc ít dự đoán hơn, kế hoạch đã lưu trong bộ nhớ cache hết hạn và trình tối ưu hóa tạo lại nó bằng cách sử dụng các giá trị khác nhau vào lần tiếp theo câu lệnh được thực thi.

SQL Server 2005 đã thêm các gợi ý truy vấn mới để có nhiều quyền kiểm soát hơn đối với việc dò tìm và biên dịch lại tham số. Gợi ý truy vấn RECOMPILE bỏ qua bộ đệm kế hoạch cho một câu lệnh đã chọn. OPTIMIZE FOR cho phép đặc tả các giá trị tham số thực tế chỉ được sử dụng để tối ưu hóa. Cuối cùng, bạn có thể cung cấp toàn bộ kế hoạch thực hiện với gợi ý KẾ HOẠCH SỬ DỤNG .

Việc triển khai ban đầu của gợi ý TÙY CHỌN (RECOMPILE) có một lỗi nên nó không xem xét tất cả các biến ràng buộc. Việc triển khai mới được giới thiệu với SQL Server 2008 có một lỗi khác, làm cho tình hình trở nên rất khó hiểu. Erland Sommarskog4 đã thu thập tất cả thông tin liên quan về tất cả các bản phát hành SQL Server.

Mặc dù các phương pháp heuristic có thể cải thiện vấn đề “logic thông minh” ở một mức độ nhất định, chúng thực sự được xây dựng để giải quyết các vấn đề của tham số ràng buộc liên quan đến biểu đồ cột và biểu thức LIKE .

Phương pháp đáng tin cậy nhất để đạt được kế hoạch thực thi tốt nhất là tránh các bộ lọc không cần thiết trong câu lệnh SQL.

[4] http://www.sommarskog.se/dyn-search-2008.html

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

môn Toán

Có một lớp nữa của obfuscations thông minh và ngăn chặn việc sử dụng chỉ mục thích hợp. Thay vì sử dụng các biểu thức logic, nó đang sử dụng một phép tính.

Hãy xem xét tuyên bố sau đây. Nó có thể sử dụng chỉ mục trên NUMERIC_NUMBER không?

```
CHỌN số_số_số
   TỪ tên_bảng
  WHERE numeric_number - 1000>?
```

Tương tự, câu lệnh sau có thể sử dụng chỉ mục trên A và B — bạn chọn thứ tự?

```
CHỌN a, b
   TỪ tên_bảng
  TRONG ĐÓ 3 * a + 5 = b
```

Hãy đặt những câu hỏi này vào một góc nhìn khác; nếu bạn đang phát triển một cơ sở dữ liệu SQL, bạn có thêm một trình giải phương trình không? Hầu hết các nhà cung cấp cơ sở dữ liệu chỉ nói "Không!" và do đó, cả hai ví dụ đều không sử dụng chỉ mục.

Bạn thậm chí có thể sử dụng toán học để làm xáo trộn một điều kiện có chủ ý — như chúng tôi đã làm trước đây cho tìm kiếm LIKE toàn văn . Thêm số 0 vào là đủ, ví dụ:

```
CHỌN số_số_số
   TỪ tên_bảng
  WHERE numeric_number + 0 =?
```

Tuy nhiên, chúng ta có thể lập chỉ mục các biểu thức này bằng chỉ mục dựa trên hàm nếu chúng ta sử dụng các phép tính một cách thông minh và biến đổi mệnh đề where giống như một phương trình:

```
CHỌN a, b
   TỪ tên_bảng
  TRONG ĐÓ 3 * a - b = -5
```

Chúng tôi chỉ di chuyển các tham chiếu bảng sang bên này và các hằng số sang bên kia. Sau đó, chúng ta có thể tạo một chỉ mục dựa trên hàm cho phía bên trái của phương trình:

```
TẠO INDEX toán học TRÊN tên_bảng (3 * a - b)
```

Machine Translated by Google

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Chương 3

Hiệu suất và khả năng mở rộng

Chương này nói về hiệu suất và khả năng mở rộng của cơ sở dữ liệu.

Trong ngữ cảnh này, tôi đang sử dụng định nghĩa sau cho khả năng mở rộng:

> Khả năng mở rộng là khả năng của một hệ thống, mạng hoặc quy trình, xử lý khối lượng công việc ngày càng tăng theo cách có khả năng hoặc khả năng mở rộng của nó để thích ứng với sự tăng trưởng đó.
>
> —Wikipedia[1]

Bạn thấy rằng thực tế có hai định nghĩa. Đầu tiên là về tác động của tải ngày càng tăng đối với hệ thống và thứ hai là về việc phát triển một hệ thống để xử lý nhiều tải hơn.

Định nghĩa thứ hai phổ biến hơn nhiều so với định nghĩa đầu tiên.
Bất cứ khi nào ai đó nói về khả năng mở rộng, thì hầu như luôn luôn là về việc sử dụng nhiều phần cứng hơn. Mở rộng quy mô và mở rộng quy mô là các từ khóa tương ứng gần đây đã được bổ sung bởi các từ thông dụng mới như quy mô web.

Nói rộng ra, khả năng mở rộng là về tác động hiệu suất của những thay đổi môi trường. Phần cứng chỉ là một thông số môi trường có thể thay đổi. Chương này cũng bao gồm các tham số khác như khối lượng dữ liệu và tải hệ thống.

[1] http://en.wikipedia.org/wiki/Scalability

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Tác động đến hiệu suất của khối lượng dữ liệu

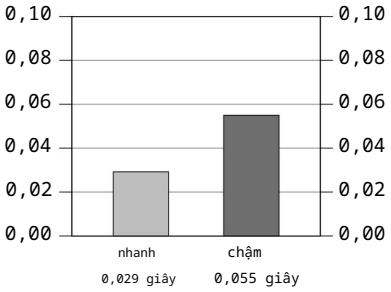
Số lượng dữ liệu được lưu trữ trong cơ sở dữ liệu có ảnh hưởng lớn đến hiệu suất của nó. Nó thường được chấp nhận rằng một truy vấn trở nên chậm hơn với dữ liệu bổ sung trong cơ sở dữ liệu. Nhưng tác động hiệu suất lớn đến mức nào nếu khối lượng dữ liệu tăng gấp đôi? Và làm thế nào chúng ta có thể cải thiện tỷ lệ này? Đây là những câu hỏi quan trọng khi thảo luận về khả năng mở rộng cơ sở dữ liệu.

Như một ví dụ, chúng tôi phân tích thời gian phản hồi của truy vấn sau đây khi sử dụng hai chỉ mục khác nhau. Các định nghĩa chỉ mục sẽ vẫn chưa được biết đến trong thời gian này - chúng sẽ được tiết lộ trong quá trình thảo luận.

```
CHỌN số lượng (*)
   Phần FROM scale_data
 WHERE =?
    VÀ id2 =?
```

Cột SECTION có mục đích đặc biệt trong truy vấn này: nó kiểm soát khối lượng dữ liệu. Số SECTION càng lớn thì truy vấn chọn càng nhiều hàng. Hình 3.1 cho thấy thời gian phản hồi cho một PHẦN nhỏ.

Hình 3.1. So sánh hiệu suất



Có một sự khác biệt đáng kể về hiệu suất giữa hai biến thể lập chỉ mục. Cả hai thời gian phản hồi vẫn thấp hơn một phần mười giây vì vậy ngay cả truy vấn chậm hơn cũng có thể đủ nhanh trong hầu hết các trường hợp. Tuy nhiên, biểu đồ hiệu suất chỉ hiển thị một điểm kiểm tra. Thảo luận về khả năng mở rộng có nghĩa là xem xét tác động của hiệu suất khi thay đổi các thông số môi trường – chẳng hạn như khối lượng dữ liệu.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

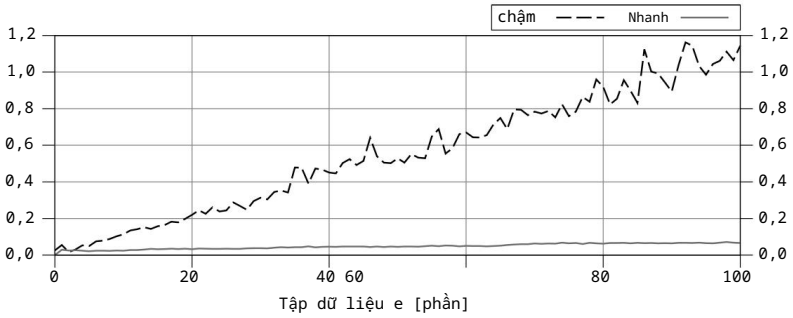
Quan trọng

Khả năng mở rộng cho thấy sự phụ thuộc của hiệu suất vào các yếu tố như khối lượng dữ liệu.

Giá trị hiệu suất chỉ là một điểm dữ liệu duy nhất trên biểu đồ khả năng mở rộng.

Hình 3.2 cho thấy thời gian phản hồi qua số SECTION – điều đó có nghĩa là đối với khối lượng dữ liệu ngày càng tăng.

Hình 3.2. Khả năng mở rộng theo khối lượng dữ liệu



Biểu đồ cho thấy thời gian phản hồi ngày càng tăng cho cả hai chỉ mục. Ở phía bên phải của biểu đồ, khi khối lượng dữ liệu cao gấp một trăm lần, truy vấn nhanh hơn cần thời gian lâu hơn gấp đôi so với thời gian ban đầu trong khi thời gian phản hồi của truy vấn chậm hơn tăng lên 20 đến hơn một giây.

Thời gian phản hồi của một truy vấn SQL phụ thuộc vào nhiều yếu tố. Khối lượng dữ liệu là một trong số đó. Nếu một truy vấn đủ nhanh trong các điều kiện thử nghiệm nhất định, điều đó không có nghĩa là nó sẽ đủ nhanh trong quá trình sản xuất. Điều đó đặc biệt xảy ra trong các môi trường phát triển chỉ có một phần nhỏ dữ liệu của hệ thống sản xuất.

Tuy nhiên, không có gì ngạc nhiên khi các truy vấn chậm hơn khi khối lượng dữ liệu tăng lên. Nhưng khoảng cách chênh lệch giữa hai chỉ số có phần bất ngờ. Lý do cho tốc độ tăng trưởng khác nhau là gì?

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Sẽ dễ dàng tìm ra lý do bằng cách so sánh cả hai kế hoạch thực hiện.

```
..................................................
| Id | Hoạt động                      | Tên              | Hàng | Chi phí |
..................................................
| 0 | TUYỂN CHỌN BÁO CÁO | | 972 |
|   1   | SẮP XẾP TỔNG HỢP | | |                  ''         ''
| * 2 | QUÉT NGUỒN CHỈ SỐ | QUY MÔ | 3000 | 972 |
..................................................


..................................................
| Hoạt động Id                       | Tên              | Hàng | Chi phí |
..................................................
| 0 | TUYỂN CHỌN BÁO CÁO | | 13 |
|   1   | SẮP XẾP TỔNG HỢP | | |                  ''         ''
| * 2 | QUÉT NGUỒN CHỈ SỐ | SCALE_FAST | 3000 | 13 |
..................................................
```

Các kế hoạch thực hiện gần như giống hệt nhau – chúng chỉ sử dụng một chỉ mục khác. Mặc dù các giá trị chi phí phản ánh sự khác biệt về tốc độ, nhưng lý do không phải hiển thị trong kế hoạch thực hiện.

Có vẻ như chúng ta đang đối mặt với "trải nghiệm lập chỉ mục chậm"; truy vấn chậm mặc dù nó sử dụng một chỉ mục. Tuy nhiên, chúng tôi không tin vào huyền thoại về "chỉ mục bị hỏng" nữa. Thay vào đó, chúng tôi nhớ hai thành phần làm chậm tra cứu chỉ mục: (1) truy cập bảng và (2) quét phạm vi chỉ số rộng.

Cả kế hoạch thực thi đều không hiển thị hoạt động TRUY CẬP BẢNG THEO CHỈ SỐ ROWID vì vậy một kế hoạch thực thi phải quét một phạm vi chỉ mục rộng hơn so với kế hoạch còn lại. Vậy ở đâu kế hoạch thực thi có hiển thị phạm vi chỉ mục được quét không? Trong vị ngữ thông tin tất nhiên!

Mẹo

Chú ý đến thông tin vị ngữ.

Thông tin vị ngữ hoàn toàn không phải là một chi tiết không cần thiết mà bạn có thể bỏ qua như đã được thực hiện ở trên. Một kế hoạch thực thi không có thông tin vị từ không đầy đủ. Điều đó có nghĩa là bạn không thể thấy lý do cho hiệu suất sự khác biệt trong các kế hoạch hiển thị ở trên. Nếu chúng ta nhìn vào quá trình thực hiện hoàn chỉnh kế hoạch, chúng ta có thể thấy sự khác biệt.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google


```
-----------------------------------------------------
| Id | Hoạt động                    | Tên              | Hàng | Chi phí |
-----------------------------------------------------
| 0 | TUYỂN CHỌN BÁO CÁO | | 972 |
|   1  | SẮP XẾP TỔNG HỢP | | |                       ''        1 1
| * 2 | QUÉT NGUỒN CHỈ SỐ | QUY MÔ | 3000 | 972 |
-----------------------------------------------------


Thông tin vị ngữ (được xác định bằng id hoạt động):
    2 - truy cập ("SECTION" = TO_NUMBER (: A))
          bộ lọc ("ID2" = TO_NUMBER (: B))

-----------------------------------------------------
| Hoạt động Id                 | Tên              | Hàng | Chi phí |
-----------------------------------------------------
| 0 | TUYỂN CHỌN BÁO CÁO | | |                                  13 |
    1  | SẮP XẾP TỔNG HỢP | |                       ''        1 1          |
| * 2 | QUÉT NGUỒN CHỈ SỐ | SCALE_FAST | 3000 |                  13 |
-----------------------------------------------------


Thông tin vị ngữ (được xác định bằng id hoạt động):
    2 - quyền truy cập ("SECTION" = TO_NUMBER (: A) VÀ "ID2" = TO_NUMBER (: B))
```

> **Ghi chú**
>
> Kế hoạch thực hiện đã được đơn giản hóa để rõ ràng. Phần phụ lục trên trang 170 giải thích các chi tiết của phần "Thông tin dự đoán" trong một kế hoạch thực thi Oracle.

Sự khác biệt là rõ ràng bây giờ: chỉ có điều kiện trên SECTION là quyền truy cập vị ngữ khi sử dụng chỉ mục SCALE_SLOW . Cơ sở dữ liệu đọc tất cả các hàng khỏi phần và loại bỏ những phần không phù hợp với vị từ bộ lọc trên ID2. Thời gian phản hồi tăng theo số hàng trong phần. Với chỉ mục SCALE_FAST , cơ sở dữ liệu sử dụng tất cả các điều kiện làm vị từ truy cập. Thời gian phản hồi tăng lên theo số lượng hàng đã chọn.

> **Quan trọng**
>
> Các vị từ bộ lọc giống như các thiết bị vật liệu chưa nổ. Họ có thể nổ bất cứ lúc nào.

Phần còn thiếu cuối cùng trong câu đố của chúng tôi là định nghĩa chỉ mục. Chúng ta có thể xây dựng lại các định nghĩa chỉ mục từ các kế hoạch thực thi?

Machine Translated by Google

Định nghĩa của chỉ mục SACLE_SLOW phải bắt đầu bằng cột SECTION— nếu không, nó không thể được sử dụng làm vị từ truy cập. Điều kiện trên ID2 không phải là một vị từ truy cập — vì vậy nó không thể tuân theo SECTION trong định nghĩa chỉ mục.
Điều đó có nghĩa là chỉ mục SCALE_SLOW phải có tối thiểu ba cột trong đó SECTION là cột đầu tiên và ID2 không phải cột thứ hai. Đó chính xác là cách nó diễn ra trong định nghĩa chỉ mục được sử dụng cho bài kiểm tra này:

```
TẠO INDEX scale_slow ON scale_data (section, id1, id2);
```

Cơ sở dữ liệu không thể sử dụng ID2 làm vị từ truy cập do cột ID1 ở vị trí thứ hai.

Định nghĩa của chỉ mục SCALE_FAST phải có cột SECTION và ID2 ở hai vị trí đầu tiên vì cả hai đều được sử dụng cho các vị từ truy cập. Tuy nhiên, chúng tôi không thể nói bất cứ điều gì về đơn đặt hàng của họ. Chỉ mục được sử dụng cho bài kiểm tra bắt đầu bằng cột SECTION và có thêm cột ID1 ở vị trí thứ ba:

```
TẠO INDEX scale_fast ON scale_data (section, id2, id1);
```

Cột ID1 vừa được thêm vào nên chỉ mục này có cùng kích thước với SCALE_SLOW — nếu không, bạn có thể nhận được hiển thị mà kích thước gây ra sự khác biệt.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

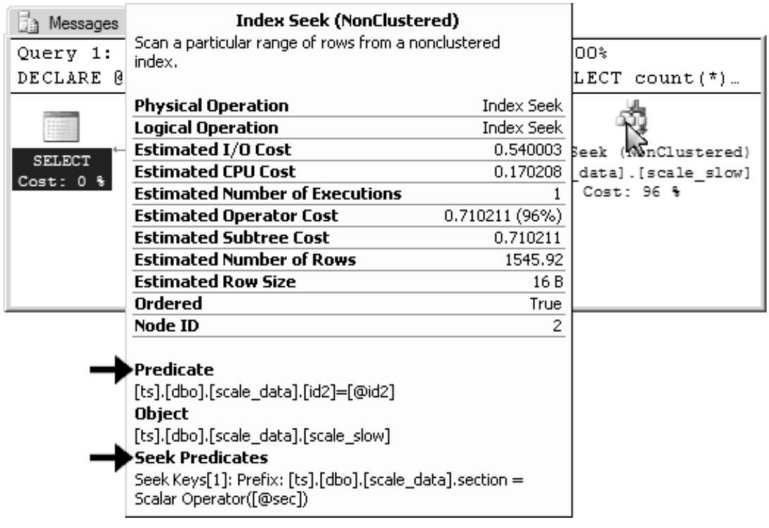
Tác động hiệu suất của tải hệ thống

Việc xem xét cách xác định chỉ mục nhiều cột thường dừng lại ngay sau khi chỉ mục được sử dụng cho truy vấn đang được điều chỉnh. Tuy nhiên, trình tối ưu hóa không sử dụng chỉ mục vì nó là chỉ mục "phù hợp" cho truy vấn, thay vì nó hiệu quả hơn so với quét toàn bộ bảng. Điều đó không có nghĩa nó là chỉ mục tối ưu cho truy vấn.

Ví dụ trước đã chỉ ra những khó khăn trong việc nhận ra thứ tự cột không chính xác trong một kế hoạch thực thi. Thông thường thông tin vị từ được ẩn rất nhiều, vì vậy bạn phải tìm kiếm nó một cách cụ thể để xác minh việc sử dụng chỉ mục tối ưu.

Ví dụ: SQL Server Management Studio chỉ hiển thị thông tin vị từ như một mẹo công cụ khi di chuyển con trỏ chuột qua thao tác lập chỉ mục ("hover"). Kế hoạch thực hiện sau sử dụng chỉ mục SCALE_SLOW ; do đó nó hiển thị điều kiện trên ID2 dưới dạng vị từ bộ lọc (chỉ là "Vị từ", không có Seek).

Hình 3.3. Xác định thông tin như một mẹo công cụ



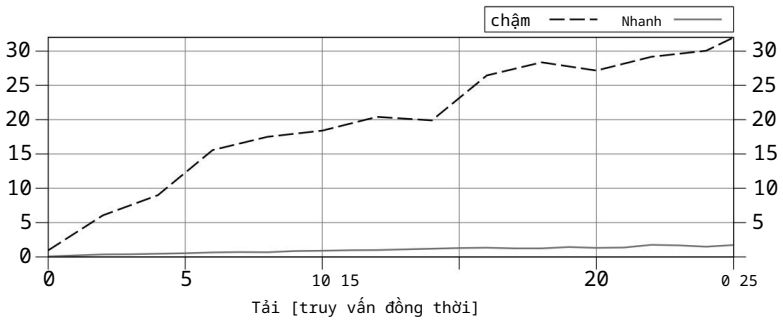
Việc lấy thông tin vị từ từ kế hoạch thực thi MySQL hoặc PostgreSQL thậm chí còn khó khăn hơn. Phụ lục A trên trang 165 có các chi tiết.

Machine Translated by Google

Cho dù thông tin vị ngữ xuất hiện trong kế hoạch thực thi không quan trọng đến mức nào, nó cũng có tác động lớn đến hiệu suất - đặc biệt là khi hệ thống phát triển. Hãy nhớ rằng không chỉ khối lượng dữ liệu tăng lên mà còn cả tốc độ truy cập. Đây là một tham số khác của hàm khả năng mở rộng.

Hình 3.4 vẽ biểu đồ thời gian phản hồi như một hàm của tốc độ truy cập – khối lượng dữ liệu không thay đổi. Nó hiển thị thời gian thực hiện của cùng một truy vấn như trước đây và luôn sử dụng phần có khối lượng dữ liệu lớn nhất.
Điều đó có nghĩa là điểm cuối cùng từ Hình 3.2 trên trang 81 tương ứng với điểm đầu tiên trong biểu đồ này.

Hình 3.4. Khả năng mở rộng theo tải hệ thống



Đường đứt nét biểu thị thời gian phản hồi khi sử dụng chỉ số SCALE_SLOW .
Nó tăng lên đến 32 giây nếu có 25 truy vấn chạy cùng một lúc. So với thời gian phản hồi khi không tải nền – như trường hợp xảy ra trong môi trường phát triển của bạn – thì thời gian này lâu hơn gấp 30 lần. Ngay cả khi bạn có bản sao đầy đủ của cơ sở dữ liệu sản xuất trong môi trường phát triển của mình, tải nền vẫn có thể khiến truy vấn chạy chậm hơn nhiều trong quá trình sản xuất.

Đường liền nét hiển thị thời gian phản hồi bằng cách sử dụng chỉ mục SCALE_FAST – nó không có bất kỳ vị từ bộ lọc nào. Thời gian phản hồi vẫn dưới hai giây ngay cả khi có 25 truy vấn chạy đồng thời.

Lưu ý

Kiểm tra kế hoạch thực hiện cẩn thận mang lại sự tự tin hơn so với các điểm chuẩn bề ngoài .

Một bài kiểm tra căng thẳng đầy đủ vẫn đáng giá – nhưng chi phí rất cao.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Thời gian phản hồi đáng ngờ thường được xem nhẹ trong quá trình phát triển. Điều này phần lớn là do chúng tôi mong đợi "phần cứng sản xuất mạnh mẽ hơn" sẽ mang lại hiệu suất tốt hơn. Thường thì ngược lại, vì cơ sở hạ tầng sản xuất phức tạp hơn và tích tụ các độ trễ không xảy ra trong môi trường phát triển. Ngay cả khi thử nghiệm trên cơ sở hạ tầng sản xuất tương đương, tải nền vẫn có thể gây ra thời gian phản hồi khác nhau. Trong phần tiếp theo, chúng ta sẽ thấy rằng nói chung là không hợp lý khi mong đợi phản hồi nhanh hơn từ "phần cứng lớn hơn".

Thời gian đáp ứng và thông lượng

Phần cứng lớn hơn không phải lúc nào cũng nhanh hơn - nhưng nó thường có thể xử lý nhiều tải hơn. Phần cứng lớn hơn giống như một đường cao tốc rộng hơn một chiếc ô tô chạy nhanh hơn: bạn không thể lái nhanh hơn — tốt, bạn không được phép — chỉ vì có nhiều làn đường hơn. Đó là lý do mà nhiều phần cứng hơn không tự động cải thiện các truy vấn SQL chậm.

Chúng ta không còn ở những năm 1990 nữa. Sức mạnh tính toán của CPU lõi đơn đang tăng lên nhanh chóng vào thời điểm đó. Hầu hết các vấn đề về thời gian phản hồi đã biến mất trên phần cứng mới hơn — chỉ vì CPU được cải tiến. Nó giống như các mẫu ô tô mới liên tục chạy nhanh gấp đôi các mẫu cũ — mỗi năm!
Tuy nhiên, sức mạnh CPU lõi đơn đã chạm ngưỡng trong vài năm đầu của thế kỷ 21. Gần như không có cải tiến nào trên trục này nữa.
Để tiếp tục xây dựng các CPU mạnh hơn bao giờ hết, các nhà cung cấp đã phải chuyển sang chiến lược đa lõi. Mặc dù nó cho phép nhiều tác vụ chạy đồng thời, nhưng nó không cải thiện hiệu suất nếu chỉ có một tác vụ.
Hiệu suất không chỉ có một thứ nguyên.

Chia tỷ lệ theo chiều ngang (thêm nhiều máy chủ hơn) cũng có những hạn chế tương tự. Mặc dù nhiều máy chủ hơn có thể xử lý nhiều yêu cầu hơn, nhưng chúng không giúp cải thiện thời gian phản hồi cho một truy vấn cụ thể. Để tìm kiếm nhanh hơn, bạn cần một cây tìm kiếm hiệu quả — ngay cả trong các hệ thống không quan hệ như CouchDB và MongoDB.

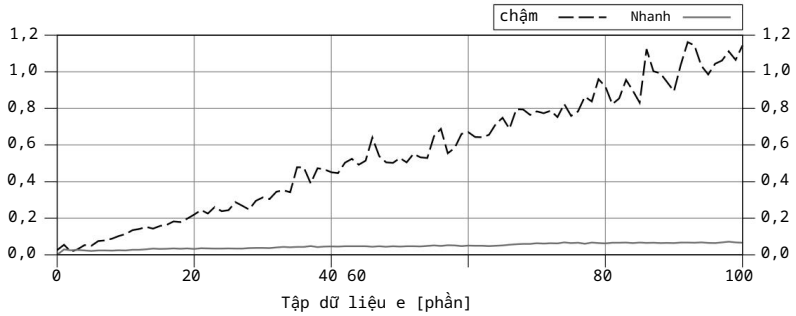
Quan trọng

Lập chỉ mục thích hợp là cách tốt nhất để giảm thời gian phản hồi truy vấn — trong cơ sở dữ liệu SQL quan hệ cũng như trong các hệ thống không quan hệ.

Machine Translated by Google

Việc lập chỉ mục thích hợp nhằm mục đích khai thác triệt để khả năng mở rộng logarit của chỉ mục cây B. Thật không may, việc lập chỉ mục thường được thực hiện một cách rất cẩu thả. Biểu đồ trong "Tác động đến hiệu suất của khối lượng dữ liệu" cho thấy rõ tác động của việc lập chỉ mục cẩu thả.

Hình 3.5. Thời gian phản hồi theo khối lượng dữ liệu



Sự khác biệt về thời gian phản hồi giữa một chỉ số cẩu thả và một chỉ số thích hợp là đáng kinh ngạc. Khó có thể bù đắp cho hiệu ứng này bằng cách bổ sung thêm phần cứng. Ngay cả khi bạn quản lý để cắt giảm thời gian phản hồi bằng phần cứng, vẫn còn là vấn đề liệu nó có phải là giải pháp tốt nhất cho vấn đề này hay không.

Nhiều hệ thống được gọi là NoSQL vẫn yêu cầu giải quyết tất cả các vấn đề về hiệu suất với khả năng mở rộng theo chiều ngang. Tuy nhiên, khả năng mở rộng này chủ yếu bị giới hạn ở các hoạt động ghi và được thực hiện với cái gọi là mô hình nhất quán cuối cùng. Cơ sở dữ liệu SQL sử dụng mô hình nhất quán chặt chẽ làm chậm hoạt động ghi, nhưng điều đó không nhất thiết ngụ ý thông lượng kém. Tìm hiểu thêm về điều này trong hộp có tên "Tính nhất quán cuối cùng và Định lý CAP".

Nhiều phần cứng hơn thường sẽ không cải thiện thời gian phản hồi. Trên thực tế, nó thậm chí có thể làm cho hệ thống chậm hơn vì độ phức tạp bổ sung có thể tích lũy thêm độ trễ. Độ trễ mạng sẽ không thành vấn đề nếu ứng dụng và cơ sở dữ liệu chạy trên cùng một máy tính, nhưng thiết lập này khá phổ biến trong môi trường sản xuất nơi cơ sở dữ liệu và ứng dụng thường được cài đặt trong phần cứng chuyên dụng. Các chính sách bảo mật thậm chí có thể yêu cầu tường lửa giữa máy chủ ứng dụng và cơ sở dữ liệu — thường làm tăng gấp đôi độ trễ mạng. Cơ sở hạ tầng càng phức tạp, độ trễ tích tụ càng nhiều và phản hồi càng chậm. Hiệu ứng này thường dẫn đến quan sát phản trực giác rằng phần cứng sản xuất đắt tiền chậm hơn môi trường máy tính để bàn rẻ tiền được sử dụng để phát triển.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Tính nhất quán cuối cùng và Định lý CAP

Duy trì tính nhất quán chặt chẽ trong một hệ thống phân tán đòi hỏi sự phối hợp đồng bộ của tất cả các hoạt động ghi giữa các nút.
Nguyên tắc này có hai tác dụng phụ khó chịu: (1) nó thêm độ trễ và tăng thời gian phản hồi; (2) nó làm giảm tính khả dụng tổng thể vì phải có nhiều thành viên cùng một lúc để hoàn thành thao tác ghi.

Cơ sở dữ liệu SQL phân tán thường bị nhầm lẫn với các cụm máy tính sử dụng hệ thống lưu trữ dùng chung hoặc bản sao master-slave. Trên thực tế, cơ sở dữ liệu phân tán giống như một cửa hàng web được tích hợp với hệ thống ERP – thường là hai sản phẩm khác nhau của các nhà cung cấp khác nhau.
Sự nhất quán giữa cả hai hệ thống vẫn là một mục tiêu mong muốn thường đạt được bằng cách sử dụng giao thức cam kết hai pha (2PC). Giao thức này thiết lập các giao dịch toàn cầu mang lại hành vi “tất cả hoặc không” nổi tiếng trên nhiều cơ sở dữ liệu. Việc hoàn thành giao dịch toàn cầu chỉ có thể thực hiện được nếu tất cả các thành viên đóng góp đều có sẵn. Do đó, nó làm giảm tính khả dụng tổng thể.

Hệ thống phân tán càng có nhiều nút, thì tính nhất quán chặt chẽ càng trở nên rắc rối. Việc duy trì tính nhất quán chặt chẽ gần như là không thể nếu hệ thống có nhiều hơn một vài nút. Mặt khác, việc loại bỏ tính nhất quán nghiêm ngặt sẽ giải quyết được vấn đề về tính khả dụng và loại bỏ thời gian phản hồi tăng lên. Ý tưởng cơ bản là thiết lập lại tính nhất quán toàn cục sau khi hoàn thành thao tác ghi trên một tập con các nút. Cách tiếp cận này chỉ để lại một vấn đề chưa được giải quyết: không thể ngăn chặn xung đột nếu hai nút chấp nhận những thay đổi trái ngược nhau. Cuối cùng đạt được sự nhất quán bằng cách xử lý các xung đột, chứ không phải bằng cách ngăn chặn chúng. Trong bối cảnh đó, tính nhất quán có nghĩa là tất cả các nút đều có cùng một dữ liệu – nó không nhất thiết phải là dữ liệu chính xác hoặc tốt nhất.

Định lý CAP của Brewer mô tả sự phụ thuộc chung giữa tính nhất quán, tính khả dụng và dung sai phân vùng.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Một độ trễ rất quan trọng khác là thời gian tìm kiếm đĩa. Việc quay các ổ đĩa cứng (HDD) cần một thời gian khá dài để đặt các bộ phận cơ học sao cho có thể đọc được dữ liệu được yêu cầu – thường là vài mili giây. Độ trễ này xảy ra bốn lần khi duyệt qua cây bốn cấp B – tổng cộng: vài chục mili giây. Mặc dù đó là một nửa vĩnh cửu đối với máy tính, nó vẫn còn thấp hơn nhiều so với ngưỡng nhận thức. khi chỉ được thực hiện một lần. Tuy nhiên, rất dễ dàng kích hoạt hàng trăm hoặc thậm chí hàng nghìn lần tìm kiếm đĩa với một câu lệnh SQL duy nhất, đặc biệt là khi kết hợp nhiều bảng với một thao tác nối. Mặc dù bộ nhớ đệm giảm đáng kể vấn đề và các công nghệ mới như SSD làm giảm thời gian tìm kiếm theo một mức độ lớn, các phép nối vẫn thường bị nghi ngờ là chậm. Do đó, chương tiếp theo sẽ giải thích cách sử dụng các chỉ mục để nối bảng hiệu quả.

Đĩa trạng thái rắn (SSD) và Đĩa trạng thái rắn trong bộ đệm

(SSD) là một công nghệ lưu trữ khối lượng lớn không sử dụng bộ phận chuyển động. Thời gian tìm kiếm điển hình của SSD nhanh hơn theo một bậc lớn hơn so với thời gian tìm kiếm của ổ cứng. SSD đã có sẵn để lưu trữ doanh nghiệp vào khoảng năm 2010 nhưng do giá thành cao và tuổi thọ hạn chế nên không được sử dụng phổ biến cho cơ sở dữ liệu.

Tuy nhiên, cơ sở dữ liệu làm bộ nhớ cache dữ liệu được truy cập thường xuyên trong bộ nhớ chính. Điều này đặc biệt hữu ích cho dữ liệu cần thiết cho mọi truy cập chỉ mục – ví dụ: các nút gốc của chỉ mục. Cơ sở dữ liệu có thể lưu vào bộ đệm đầy đủ các chỉ mục được sử dụng thường xuyên để tra cứu chỉ mục không kích hoạt tìm kiếm đĩa đơn.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Chương 4

Hoạt động Tham gia

Một truy vấn SQL đi vào một thanh và thấy hai bảng.
Anh ấy đến gần họ và hỏi 'Tôi có thể tham gia cùng bạn không?'

—Nguồn: Không xác định

Phép toán kết hợp chuyển đổi dữ liệu từ một mô hình chuẩn hóa thành một dạng không chuẩn hóa phù hợp với một mục đích xử lý cụ thể. Tham gia đặc biệt nhạy cảm với độ trễ tìm kiếm đĩa vì nó kết hợp các phân đoạn dữ liệu phân tán. Lập chỉ mục thích hợp một lần nữa là giải pháp tốt nhất để giảm thời gian phản hồi. Tuy nhiên, chỉ mục chính xác phụ thuộc vào thuật toán nối chung nào được sử dụng cho truy vấn.

Tuy nhiên, có một điều phổ biến đối với tất cả các thuật toán nối: chúng chỉ xử lý hai bảng cùng một lúc. Truy vấn SQL với nhiều bảng hơn yêu cầu nhiều bước: đầu tiên xây dựng tập hợp kết quả trung gian bằng cách nối hai bảng, sau đó nối kết quả với bảng tiếp theo, v.v.

Mặc dù thứ tự tham gia không ảnh hưởng đến kết quả cuối cùng, nhưng nó vẫn ảnh hưởng đến hiệu suất. Do đó, trình tối ưu hóa sẽ đánh giá tất cả các hoán vị thứ tự tham gia có thể có và chọn lựa chọn tốt nhất. Điều đó có nghĩa là chỉ cần tối ưu hóa một câu lệnh phức tạp có thể trở thành một vấn đề về hiệu suất. Càng nhiều bảng tham gia, càng có nhiều biến thể kế hoạch thực thi để đánh giá — nói một cách toán học: n! (tăng trưởng giai thừa), mặc dù đây không phải là vấn đề khi sử dụng các tham số ràng buộc.

Quan trọng Câu

lệnh càng phức tạp thì việc sử dụng các tham số ràng buộc càng trở nên quan trọng.

Không sử dụng các tham số ràng buộc giống như việc biên dịch lại một chương trình mỗi lần.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Pipelining kết quả trung gian

Mặc dù các kết quả trung gian giải thích thuật toán rất tốt, nhưng điều đó không có nghĩa là cơ sở dữ liệu phải hiện thực hóa nó. Điều đó có nghĩa là lưu trữ kết quả trung gian của phép nối đầu tiên trước khi bắt đầu phép nối tiếp theo. Thay vào đó, cơ sở dữ liệu sử dụng pipelining để giảm mức sử dụng bộ nhớ. Điều đó có nghĩa là mỗi hàng từ kết quả trung gian ngay lập tức được chuyển đến thao tác nối tiếp theo — tránh phải lưu tập hợp kết quả trung gian.

Vòng lồng nhau

Phép nối vòng lặp lồng nhau là thuật toán nối cơ bản nhất. Nó hoạt động giống như sử dụng hai truy vấn lồng nhau: truy vấn bên ngoài hoặc truy vấn lái xe để tìm nạp kết quả từ một bảng và truy vấn thứ hai cho mỗi hàng từ truy vấn lái xe để tìm nạp dữ liệu tương ứng từ bảng khác.

Bạn thực sự có thể sử dụng "lựa chọn lồng nhau" để tự triển khai thuật toán vòng lặp lồng nhau. Tuy nhiên, đó là một cách tiếp cận rắc rối vì độ trễ mạng xảy ra cùng với độ trễ đĩa — làm cho thời gian phản hồi tổng thể thậm chí còn tồi tệ hơn. "Các lựa chọn lồng nhau" vẫn rất phổ biến vì bạn có thể dễ dàng thực hiện chúng mà không cần biết. Các công cụ ánh xạ quan hệ đối tượng (ORM) đặc biệt "hữu ích" về mặt này. đến mức cái gọi là vấn đề lựa chọn N + 1 đã gây được tiếng vang lớn trong lĩnh vực này.

Các ví dụ sau đây cho thấy các phép nối "lựa chọn lồng nhau ngẫu nhiên" này được tạo bằng các công cụ ORM khác nhau. Các ví dụ tìm kiếm nhân viên có họ bắt đầu bằng 'WIN' và lấy tất cả SALES cho những nhân viên này.

Các ORM không tạo các phép nối SQL — thay vào đó chúng truy vấn bảng SALES với các lựa chọn lồng nhau. Hiệu ứng này được gọi là "vấn đề chọn N + 1" hoặc ngắn hơn là "vấn đề N + 1" vì nó thực hiện tổng cộng N + 1 lựa chọn nếu truy vấn điều khiển trả về N hàng.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Java

Ví dụ JPA sử dụng giao diện CriteriaBuilder.

```
CriteriaBuilder queryBuilder = em.getCriteriaBuilder (); CriteriaQuery
<Nhân viên>
    query = queryBuilder.createQuery (Nhân viên.class);
Root <Nhân viên> r = query.from (Nhân viên. lớp); query.where
( queryBuilder.like ( queryBuilder.upper (r.get (Nhân
   viên_.lastName)), "THẮNG%"



));

Danh sách <Nhân viên> emp = em.createQuery (truy vấn) .getResultList ();

cho (Nhân viên e: emp) {// xử lý
   Nhân viên cho (Bán hàng s:
   e.getSales ()) {// xử lý bán hàng cho
      Nhân viên}}
```

Hibernate JPA 3.6.0 tạo N + 1 truy vấn chọn :

```
chọn nhân viên0_.subsidiary_id làm công ty con1_0_
          - NHIỀU COLUMNS từ
   nhân viên nhân viên0_ nơi cấp trên
 (nhân viên0_.last_name) như thế nào?

   chọn sales0_.subsidiary_id làm công ty con4_0_1_
             - NHIỀU CỘT HƠN từ
      doanh số bán hàng0_ nơi
    doanh số bán hàng0_.subsidiary_id =? và
       sales0_.employee_id =?

   chọn sales0_.subsidiary_id làm công ty con4_0_1_
             - NHIỀU CỘT HƠN từ
      doanh số bán hàng0_ nơi
    doanh số bán hàng0_.subsidiary_id =? và
       sales0_.employee_id =?
```

Machine Translated by Google

Perl

Mẫu sau minh họa khung DBIx :: Class của Perl:

```
my @employees = $
    schema-> resultset ('Nhân viên') ->
             tìm kiếm ({'UPPER (last_name)' => {-like => 'WIN%'}});

foreach my $ worker (@employees) {process
      Nhân viên báo trước việc bán $ của
    tôi ($ worker-> sales) {process Bán cho Nhân
          viên
    }
}
```

DBIx :: Lớp 0.08192 tạo N + 1 truy vấn chọn :

```
CHỌN me.employee_id, me.subsidiary_id,
       me.last_name, me.first_name, me.date_of_birth TỪ nhân
   viên của tôi Ở ĐÂU (LÊN (last_name) THÍCH?)



    CHỌN me.sale_id, me.employee_id, me.subsidiary_id, me.sale_date,
          me.eur_value TỪ bán hàng cho tôi

     WHERE ((me.employee_id =?
        VÀ me.subsidiary_id =? ))

    CHỌN me.sale_id, me.employee_id, me.subsidiary_id, me.sale_date,
          me.eur_value TỪ bán hàng cho tôi Ở ĐÂU ((me.employee_id
       =?

        VÀ me.subsidiary_id =? ))
```

PHP

Mẫu Doctrine sử dụng giao diện trình tạo truy vấn:

```
$ qb = $ em-> createQueryBuilder (); $
qb-> select ('e') -> from ('Nhân viên',
    'e') -> where ("upper (e.last_name)
    like: last_name") -> setParameter ('last_name', 'WIN%' );

$ r = $ qb-> getQuery () -> getResult ();

foreach ($ r as $ row) {//
    process foreach Nhân
    viên ($ row-> getSales () as $ sale) {// xử
        lý Sale for Employee
    }
}
```


Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Doctrine 2.0.5 tạo ra N + 1 truy vấn được chọn :

```
CHỌN e0_.employee_id AS worker_id0 - NHIỀU CỘT KHÁC
   TỪ nhân viên e0_ LÊN ĐÂU
  (e0_.last_name) NHƯ THẾ NÀO?

     CHỌN t0.sale_id NHƯ SALE_ID1 - NHIỀU CỘT KHÁC TỪ doanh số bán hàng t0

      WHERE t0.subsidiary_id =?
         VÀ t0.employee_id =?

     CHỌN t0.sale_id NHƯ SALE_ID1 - NHIỀU CỘT KHÁC TỪ doanh số bán hàng t0

      WHERE t0.subsidiary_id =?
         VÀ t0.employee_id =?
```

Bật ghi nhật ký SQL Bật ghi nhật ký

SQL trong quá trình phát triển và xem lại các câu lệnh SQL đã tạo.

DBIx :: Lớp

xuất DBIC_TRACE = 1 trong trình bao của bạn.

Doctrine Chỉ

ở cấp mã nguồn – đừng quên tắt tính năng này cho quá trình sản xuất. Cân nhắc xây dựng trình ghi nhật ký có thể định cấu hình của riêng bạn.

```
$ logger = new \ Doctrine \ DBAL \ Logging \ EchoSqlLogger; $ config->
setSQLLogger ($ logger);
```

Hibernate (native)

<property name = "show_sql"> true </property> trong App.config hoặc hibernate.cfg.xml

JPA

Trong Persence.xml nhưng tùy thuộc vào nhà cung cấp JPA:

```
<property name = "eclipselink.logging.level" value = "FINE" /> <property name =
"hibernate.show_sql" value = "TRUE" /> <property name = "openjpa.Log" value =
"SQL = TRACE" />
```

Hầu hết các ORM đều cung cấp một cách lập trình để kích hoạt tính năng ghi nhật ký SQL. Điều đó liên quan đến rủi ro vô tình triển khai cài đặt trong sản xuất.

Machine Translated by Google

Mặc dù cách tiếp cận "lựa chọn lồng nhau" là một phương pháp chống lại mẫu, nó vẫn giải thích các vòng lặp lồng nhau tham gia khá tốt. Cơ sở dữ liệu thực thi phép nối chính xác như các công cụ ORM ở trên. Do đó, lập chỉ mục cho một phép nối các vòng lồng nhau giống như lập chỉ mục cho các câu lệnh chọn được hiển thị ở trên . Đó là chỉ mục dựa trên hàm trên bảng NHÂN VIÊN và chỉ mục được nối cho các vị từ tham gia trên bảng BÁN HÀNG :

```
TẠO INDEX emp_up_name ON nhân viên (UPPER (last_name));
TẠO INDEX sales_emp TRÊN doanh số bán hàng (công ty con, nhân_viên);
```

Một phép nối SQL vẫn hiệu quả hơn so với cách tiếp cận lựa chọn lồng nhau – ngay cả khi nó thực hiện các tra cứu chỉ mục giống nhau – vì nó tránh được nhiều giao tiếp mạng. Nó thậm chí còn nhanh hơn nếu tổng lượng dữ liệu được chuyển lớn hơn vì sự trùng lặp của các thuộc tính nhân viên cho mỗi lần bán hàng. Đó là vì hai khía cạnh của hiệu suất: thời gian phản hồi và thông lượng; trong mạng máy tính, chúng tôi gọi chúng là độ trễ và băng thông. Băng thông chỉ có tác động nhỏ đến thời gian phản hồi nhưng độ trễ có tác động rất lớn. Điều đó có nghĩa là số lượng chuyến đi vòng quanh cơ sở dữ liệu quan trọng hơn đối với thời gian phản hồi hơn là số lượng dữ liệu được truyền.

Mẹo

Thực thi kết hợp trong cơ sở dữ liệu.

Hầu hết các công cụ ORM cung cấp một số cách để tạo các phép nối SQL. Cái gọi là chế độ tìm nạp háo hức có lẽ là chế độ quan trọng nhất. Nó thường được định cấu hình ở cấp thuộc tính trong ánh xạ thực thể – ví dụ: cho thuộc tính nhân viên trong lớp Bán hàng . Sau đó, công cụ ORM sẽ luôn tham gia vào bảng NHÂN VIÊN khi truy cập vào bảng BÁN HÀNG . Định cấu hình tìm nạp háo hức trong ánh xạ thực thể chỉ có ý nghĩa nếu bạn luôn cần thông tin chi tiết về nhân viên cùng với dữ liệu bán hàng.

Tìm nạp háo hức sẽ phản tác dụng nếu bạn không cần các bản ghi con mỗi khi bạn truy cập đối tượng mẹ. Đối với ứng dụng danh bạ điện thoại, không có ý nghĩa gì khi tải hồ sơ BÁN HÀNG khi hiển thị chi tiết nhân viên. Bạn có thể cần dữ liệu bán hàng liên quan trong các trường hợp khác – nhưng không phải lúc nào cũng vậy. Một cấu hình tĩnh không phải là giải pháp.

Để có hiệu suất tối ưu, bạn cần có toàn quyền kiểm soát đối với các phép nối. Các ví dụ sau đây cho thấy cách có được tính linh hoạt cao nhất bằng cách kiểm soát hành vi tham gia trong thời gian chạy.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Java

Giao diện JPA CriteriaBuilder cung cấp phương thức Root <>. Fetch () để kiểm soát các phép nối. Nó cho phép bạn chỉ định thời điểm và cách nối các đối tượng được giới thiệu vào truy vấn chính. Trong ví dụ này, chúng tôi sử dụng một phép nối bên trái để truy xuất tất cả nhân viên ngay cả khi một số người trong số họ không có doanh số bán hàng.

Cảnh báo JPA

và Hibernate trả lại nhân viên cho mỗi lần bán hàng.

Điều đó có nghĩa là một nhân viên có 30 lần bán hàng sẽ xuất hiện 30 lần. Mặc dù rất đáng lo ngại, đó là hành vi được chỉ định (tính bền bỉ của EJB 3.0, đoạn 4.4.5.3 “Tìm nạp tham gia”). Bạn có thể khử trùng lặp mối quan hệ mẹ theo cách thủ công hoặc sử dụng hàm diff () như được minh họa trong ví dụ.

```
CriteriaBuilder qb = em.getCriteriaBuilder (); CriteriaQuery
<Nhân viên> q = qb.createQuery (Nhân viên. lớp); Root <Nhân viên> r = q.from
(Nhân viên. lớp); q.where (queryBuilder.like (

     queryBuilder.upper (r.get (Nhân viên_.lastName)), "THẮNG%")

);

r.fetch ("bán hàng", JoinType.LEFT); //
cần thiết để tránh trùng lặp các bản ghi Nhân viên query.distinct
(true);

Danh sách <Nhân viên> emp = em.createQuery (truy vấn) .getResultList ();
```

Hibernate 3.6.0 tạo câu lệnh SQL sau:

```
chọn nhân viên khác
         biệt0_.subsidiary_id làm công ty con1_0_0_
       , nhân viên0_.employee_id với tư cách là nhân viên2_0_0_
         - THÊM COLUMNS
       , sales1_.sale_id as sale1_0__ from
   the staff staff0_ left ngoai tham gia
   sales1_ on staff0_.subsidiary_id =
             sales1_.subsidiary_id va staff0_.employee_id = sales1_.employee_id
            trong đó cấp trên (nhân viên0_.last_name) như thế nào?
```

Truy vấn có kết hợp bên trái được mong đợi nhưng cũng có một từ khóa riêng biệt không cần thiết. Thật không may, JPA không cung cấp các lệnh gọi API riêng biệt để lọc các mục nhập mẹ trùng lặp mà không khử trùng lặp các bản ghi con như

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Tốt. Từ khóa khác biệt trong truy vấn SQL đáng báo động vì hầu hết cơ sở dữ liệu sẽ thực sự lọc các bản ghi trùng lặp. Chỉ một số cơ sở dữ liệu nhận ra rằng các khóa chính đảm bảo tính duy nhất trong trường hợp đó dù sao.

API Hibernate bản địa giải quyết vấn đề ở phía máy khách bằng cách sử dụng bộ biến áp kết quả:

```
Tiêu chí c = session.createCriteria (Nhân viên. lớp);
c.add (Restrictions.ilike ("lastName", 'Win%'));

c.setFetchMode ("bán hàng", FetchMode.JOIN);
c.setResultTransformer (Criteria.DISTINCT_ROOT_ENTITY);

Danh sách <Nhân viên> result = c.list ();
```

Nó tạo ra truy vấn sau:

```
chọn this_.subsidiary_id làm công ty con1_0_1_
       , this_.employee_id với tư cách là nhân viên2_0_1_
         - MORE this_ cột về nhân viên
       , sales2_.sale_id as sale1_3_
         - MORE sales2_ cột về doanh số bán hàng
   từ nhân viên this_ left
   bên ngoài tham gia sales2_ on
            this_.subsidiary_id = sales2_.subsidiary_id và
           this_.employee_id = sales2_.employee_id ở đâu thấp hơn
  (this_.last_name) như thế nào?
```

Phương pháp này tạo ra SQL thẳng không có mệnh đề ngoài ý muốn. Ghi chú rằng Hibernate sử dụng low () cho các truy vấn không phân biệt chữ hoa chữ thường – một điều quan trọng chi tiết để lập chỉ mục dựa trên chức năng.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Perl

Ví dụ sau sử dụng khung DBIx :: Class của Perl:

```
my @employees = $
    schema-> resultset ('Nhân viên')
             -> tìm kiếm ({'UPPER (last_name)' => {-like => 'WIN%'}
                        , {prefetch => ['sales']}
                        });
```

DBIx :: Class 0.08192 tạo câu lệnh SQL sau:

```
CHỌN me.employee_id, me.subsidiary_id, me.last_name
         - NHIỀU CỘT
   TỪ nhân viên của tôi
   TRÁI THAM GIA doanh số bán hàng
            BẬT (sales.employee_id = me.employee_id VÀ
           sales.subsidiary_id = me.subsidiary_id)
 Ở ĐÂU (LÊN (last_name) THÍCH?)
 ĐẶT HÀNG THEO sales.employee_id, sales.subsidiary_id
```

Lưu ý thứ tự theo điều khoản – nó không được ứng dụng yêu cầu. Các cơ sở dữ liệu phải sắp xếp tập hợp kết quả cho phù hợp và điều đó có thể mất một trong khi.

PHP

Ví dụ sau sử dụng khung Doctrine của PHP:

```
$ qb = $ em-> createQueryBuilder ();
$ qb-> select ('e, s')
    -> from ('Nhân viên', 'e')
    -> leftJoin ('e.sales', 's')
    -> where ("upper (e.last_name) như: last_name")
    -> setParameter ('last_name', 'WIN%');
$ r = $ qb-> getQuery () -> getResult ();
```

Doctrine 2.0.5 tạo câu lệnh SQL sau:

```
CHỌN e0_.employee_id AS worker_id0
         - NHIỀU CỘT
   TỪ nhân viên e0_ TRÁI
   THAM GIA bán hàng s1_
            TRÊN e0_.subsidiary_id = s1_.subsidiary_id VÀ
           e0_.employee_id = s1_.employee_id NGƯỜI LÊN LỚP
 (e0_.last_name) THÍCH Ở ĐÂU?
```

Machine Translated by Google

Kế hoạch thực thi hiển thị hoạt động NESTED LOOPS OUTER :

```
| Id | Hoạt động                                  | Tên                | Hàng | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO | 822 | | 1                                                          38 |
      | NESTED LOOPS OUTER | 822 |                                                              38 |
| 2 | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | TUYỂN NHÂN VIÊN | 1 |                                 4 |
| * 3 | QUÉT NGUỒN CHỈ SỐ                          | EMP_UP_NAME | | 1                                    |
| 4 | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | BÁN HÀNG | 821 |                                      34 |
| * 5 | QUÉT NGUỒN CHỈ SỐ                          | SALES_EMP | 31 |                                        |


Thông tin vị ngữ (được xác định bằng id hoạt động):

   3 - truy cập (UPPER ("LAST_NAME") LIKE 'WIN%')
         bộ lọc (UPPER ("LAST_NAME") LIKE 'WIN%')
   5 - quyền truy cập ("E0 _". "SUBSIDIARY_ID" = "S1 _". "SUBSIDIARY_ID" (+)
           VÀ "E0 _". "EMPLOYEE_ID" = "S1 _". "EMPLOYEE_ID" (+))
```

Cơ sở dữ liệu truy xuất kết quả từ bảng EMPLOYEES qua EMP_UP_NAME đầu tiên và tìm nạp các bản ghi tương ứng từ bảng SALES cho mỗi nhân viên sau đó.

Mẹo

Tìm hiểu ORM của bạn và kiểm soát các liên kết.

Kết nối các vòng lồng nhau mang lại hiệu suất tốt nếu truy vấn lái xe trả về một tập hợp kết quả nhỏ. Nếu không, trình tối ưu hóa có thể chọn một thuật toán liên kết khác nhau — như liên kết băm được mô tả trong phần tiếp theo, nhưng điều này chỉ có thể thực hiện được nếu ứng dụng sử dụng phép nối để thông báo cho cơ sở dữ liệu dữ liệu mà nó thực sự cần.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Tham gia băm

Thuật toán tham gia băm nhằm mục đích tìm ra điểm yếu của tham gia vòng lặp lồng nhau: nhiều B-tree duyệt khi thực hiện truy vấn bên trong. Thay vào đó, nó tải các bản ghi ứng viên từ một phía của phép nối vào một bảng băm có thể được thăm dò rất nhanh cho mỗi hàng từ phía bên kia của phép nối. Việc điều chỉnh tham gia băm yêu cầu một cách tiếp cận lập chỉ mục hoàn toàn khác với tham gia vòng lặp lồng nhau. Ngoài ra, cũng có thể cải thiện hiệu suất tham gia băm bằng cách chọn ít cột hơn — một thách thức đối với hầu hết các công cụ ORM.

Chiến lược lập chỉ mục cho phép nối băm rất khác vì không cần lập chỉ mục các cột nối. Chỉ các chỉ mục cho các vị từ độc lập cải thiện hiệu suất tham gia băm.

Mẹo

Lập chỉ mục các vị từ độc lập ở đâu để cải thiện hiệu suất tham gia băm.

Hãy xem xét ví dụ sau. Nó chọn tất cả doanh số bán hàng trong sáu tháng qua với thông tin chi tiết về nhân viên tương ứng:

```
LỰA CHỌN *
   TỪ bán hàng s
   THAM GIA nhân viên e BẬT (s.subsidiary_id = e.subsidiary_id VÀ s.employee_id =
                            e.employee_id)
 WHERE s.sale_date> trunc (sysdate) - INTERVAL '6' THÁNG
```

Bộ lọc SALE_DATE là mệnh đề where độc lập duy nhất — điều đó có nghĩa là nó chỉ đề cập đến một bảng và không thuộc về các vị từ nối.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google


```
| Id | Hoạt động                    | Tên             | Hàng | Số byte | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO | | 49244 | | * 1                                  59 triệu | 12049 |
        | HASH THAM GIA | | 49244 |                                          59 triệu | 12049 |
| 2 | BẢNG TRUY CẬP ĐẦY ĐỦ | TUYỂN NHÂN VIÊN | 10000 |                                   9 triệu | 478 |
| * 3 | BẢNG TRUY CẬP ĐẦY ĐỦ | BÁN HÀNG | 49244 |                                  10 triệu | 10521 |


Thông tin vị ngữ (được xác định bằng id hoạt động):

     1 - quyền truy cập ("S". "SUBSIDIARY_ID" = "E". "SUBSIDIARY_ID"
               VÀ "S". "EMPLOYEE_ID" = "E". "EMPLOYEE_ID")
     3 - bộ lọc ("S". "SALE_DATE"> TRUNC (SYSDATE @!)
                                        -INTERVAL '+ 00-06' NĂM (2) ĐẾN THÁNG)
```

Bước thực hiện đầu tiên là quét toàn bộ bảng để tải tất cả nhân viên vào bảng băm (id kế hoạch 2). Bảng băm sử dụng các vị từ nối làm khóa. bên trong bước tiếp theo, cơ sở dữ liệu thực hiện quét toàn bộ bảng khác trên bảng SALES và loại bỏ tất cả doanh số bán hàng không đáp ứng điều kiện vào SALE_DATE (id gói 3). Đối với các bản ghi SALES còn lại , cơ sở dữ liệu truy cập bảng băm để tải chi tiết nhân viên tương ứng.

Mục đích duy nhất của bảng băm là hoạt động như một bộ nhớ tạm thời cấu trúc để tránh truy cập bảng NHÂN VIÊN nhiều lần. Bảng băm được tải ban đầu trong một lần chụp để không cần chỉ mục để tìm nạp các bản ghi đơn lẻ một cách hiệu quả. Thông tin vị từ xác nhận rằng không một bộ lọc nào được áp dụng trên bảng NHÂN VIÊN (id kế hoạch 2). Truy vấn không có bất kỳ vị từ độc lập nào trên bảng này.

Quan trọng

Lập chỉ mục các vị từ tham gia liên kết không cải thiện hiệu suất tham gia băm.

Điều đó không có nghĩa là không thể lập chỉ mục một tham gia băm. Độc lập các vị từ có thể được lập chỉ mục. Đây là những điều kiện được áp dụng trong một trong hai thao tác truy cập bảng. Trong ví dụ trên, nó là bộ lọc vào SALE_DATE.

```
TẠO INDEX sales_date ON sales (sale_date);
```

Kế hoạch thực hiện sau sử dụng chỉ mục này. Tuy nhiên, nó sử dụng một quét bảng để tìm bảng NHÂN VIÊN vì truy vấn không có vị ngữ where on EMPLOYEES.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google


```
---------------------------------------------------------------------------
| Id | Hoạt động                                    | Tên              | Số byte | Chi phí |
---------------------------------------------------------------------------
| 0 | TUYỂN CHỌN BÁO CÁO |                                                | 59 triệu | 3252 |
| * 1   | HASH THAM GIA |                                                | 59 triệu | 3252 |
| 2 | TRUY CẬP BẢNG ĐẦY ĐỦ                          | TUYỂN NHÂN VIÊN |       9 triệu | 478 |
| 3 | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | BÁN HÀNG                | 10 triệu | Năm 1724 |
| * 4 |         QUÉT NGUỒN CHỈ SỐ                    | SALES_DATE | | |
---------------------------------------------------------------------------


Thông tin vị ngữ (được xác định bằng id hoạt động):
---------------------------------------------------

     1 - quyền truy cập ("S". "SUBSIDIARY_ID" = "E". "SUBSIDIARY_ID"
               VÀ "S". "EMPLOYEE_ID" = "E". "EMPLOYEE_ID")
     4 - truy cập ("S". "SALE_DATE"> TRUNC (SYSDATE @!)
                                        -INTERVAL '+ 00-06' NĂM (2) ĐẾN THÁNG)
```

Lập chỉ mục một phép nối băm – trái với phép nối vòng lặp lồng nhau – đối xứng. Cái đó có nghĩa là thứ tự tham gia không ảnh hưởng đến việc lập chỉ mục. Chỉ mục SALES_DATE có thể được sử dụng để tải bảng băm nếu thứ tự nối bị đảo ngược.

Ghi chú

Lập chỉ mục một phép tham gia băm độc lập với thứ tự tham gia.

Một cách tiếp cận khá khác để tối ưu hóa hiệu suất tham gia băm là giảm thiểu kích thước bảng băm. Phương pháp này hoạt động vì một hàm băm tối ưu chỉ có thể tham gia nếu toàn bộ bảng băm phù hợp với bộ nhớ. Trình tối ưu hóa do đó sẽ tự động sử dụng phía nhỏ hơn của phép nối cho hàm băm bàn. Kế hoạch thực thi Oracle cho thấy yêu cầu bộ nhớ ước tính trong cột "Byte". Trong kế hoạch thực hiện ở trên, bảng NHÂN VIÊN cần chín megabyte và do đó nhỏ hơn.

Cũng có thể giảm kích thước bảng băm bằng cách thay đổi truy vấn SQL, chẳng hạn bằng cách thêm các điều kiện bổ sung để cơ sở dữ liệu tải ít hơn hồ sơ ứng viên vào bảng băm. Tiếp tục ví dụ trên nó có nghĩa là thêm bộ lọc trên thuộc tính DEPARTMENT để chỉ nhân viên bán hàng được xem xét. Điều này cải thiện hiệu suất tham gia băm ngay cả khi không có chỉ mục trên thuộc tính DEPARTMENT vì cơ sở dữ liệu không cần nhân viên cửa hàng không thể có doanh số bán hàng trong bảng băm. Khi làm vì vậy bạn phải đảm bảo rằng không có hồ sơ BÁN HÀNG cho nhân viên không làm việc trong bộ phận tương ứng. Sử dụng các ràng buộc để bảo vệ các giả định.

Machine Translated by Google

Khi giảm thiểu kích thước bảng băm, yếu tố liên quan không phải là số của các hàng trừ dấu chân bộ nhớ. Trên thực tế, nó cũng có thể làm giảm kích thước bảng băm bằng cách chọn ít cột hơn — chỉ những thuộc tính bạn thực sự cần:

```
CHỌN s.sale_date, s.eur_value
        , e.last_name, e.first_name
   TỪ bán hàng s
   THAM GIA nhân viên e BẬT (s.subsidiary_id = e.subsidiary_id
                             VÀ s.employee_id = e.employee_id)
 WHERE s.sale_date> trunc (sysdate) - INTERVAL '6' THÁNG
```

Phương pháp đó hiếm khi tạo ra lỗi vì thả sai cột có thể sẽ nhanh chóng dẫn đến một thông báo lỗi. Tuy nhiên, nó là có thể để giảm đáng kể kích thước bảng băm, trong trường hợp cụ thể này là từ 9 megabyte xuống 234 kilobyte — giảm 97%.

```
| Id | Hoạt động                                  | Tên             | Số byte | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO |                                                | 2067K | 2202 |
| * 1  | HASH THAM GIA |                                                | 2067K | 2202 |
| 2 | TRUY CẬP BẢNG ĐẦY ĐỦ                          | TUYỂN NHÂN VIÊN | 234K | 478 |
| 3 | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | BÁN HÀNG                | 913K | Năm 1724 |
| * 4 |        QUÉT NGUỒN CHỈ SỐ                    | SALES_DATE | | 133 |
```

> **Mẹo**
>
> Chọn ít cột hơn để cải thiện hiệu suất tham gia băm.

Mặc dù thoạt nhìn có vẻ đơn giản để xóa một vài cột khỏi một câu lệnh SQL, đó là một thách thức thực sự khi sử dụng một đối tượng-quan hệ công cụ ánh xạ (ORM). Hỗ trợ cho cái gọi là các đối tượng một phần là rất thưa thớt. Các các ví dụ sau đây cho thấy một số khả năng.

Java

JPA định nghĩa FetchType.LAZY trong chú thích @Basic . Nó có thể áp dụng trên cấp độ thuộc tính:

```
@Column (name = "rác")
@Basic (tìm nạp = FetchType.LAZY)
chuỗi riêng tư rác;
```


Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Các nhà cung cấp JPA có thể bỏ qua nó:

> Chiến lược LAZY là một gợi ý cho thời gian chạy của trình cung cấp bền vững rằng dữ liệu nên được tìm nạp một cách lười biếng khi nó được truy cập lần đầu tiên. Việc triển khai được phép tìm nạp nhanh dữ liệu mà gợi ý chiến lược LAZY đã được chỉ định.
>
> —EJB 3.0 JPA, đoạn 9.1.18

Hibernate 3.6 thực hiện tìm nạp thuộc tính lười biếng thông qua thiết bị đo bytecode thời gian biên dịch. Công cụ đo thêm mã bổ sung vào các lớp đã biên dịch không tìm nạp các thuộc tính LAZY cho đến khi được truy cập.
Cách tiếp cận hoàn toàn minh bạch đối với ứng dụng nhưng nó mở ra cánh cửa cho một chiều hướng mới của N + 1 vấn đề: một lựa chọn cho mỗi bản ghi và thuộc tính. Điều này đặc biệt nguy hiểm vì JPA không cung cấp kiểm soát thời gian chạy để tìm nạp một cách háo hức nếu cần.

Ngôn ngữ truy vấn bản địa của Hibernate HQL giải quyết vấn đề với TÌM KIẾM tất cả mệnh đề THUỘC TÍNH :

```
chọn s từ Bộ phận bán hàng TÌM KIẾM TẤT CẢ CÁC THUỘC TÍNH
  tham gia bên trong tìm nạp s.employee e TÌM TẤT CẢ CÁC THUỘC TÍNH
  trong đó s.saleDate>: dt
```

Mệnh đề FETCH ALL PROPERTIES buộc Hibernate háo hức tìm nạp thực thể — ngay cả khi sử dụng mã công cụ và chú thích LAZY .

Một tùy chọn khác để chỉ tải các cột đã chọn là sử dụng các đối tượng vận chuyển dữ liệu (DTO) thay vì các thực thể. Phương thức này hoạt động theo cùng một cách trong HQL và JPQL, đó là bạn khởi tạo một đối tượng trong truy vấn:

```
chọn SalesHeadDTO mới (s.saleDate,,      s.eurValue
                       e.firstName, e.lastName)
  from Sales s
  tham gia s.employee
 e where s.saleDate>: dt
```

Truy vấn chỉ chọn dữ liệu được yêu cầu và trả về một đối tượng SalesHeadDTO — một đối tượng Java đơn giản (POJO), không phải một thực thể.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Perl

Khung DBIx :: Lớp không hoạt động như trình quản lý thực thể để kế thừa không gây ra vấn đề răng cưa[1] . Sách dạy nấu ăn hỗ trợ cách tiếp cận này. Định nghĩa lược đồ sau xác định lớp Bán hàng trên hai cấp độ:

```
gói UseTheIndexLuke :: Schema :: Result :: SalesHead; sử dụng cơ
sở qw / DBIx :: Class :: Core /;

__PACKAGE __-> table ('sales');
__PACKAGE __-> add_columns (qw / sale_id worker_id child_id
                                  sale_date eur_value /);
__PACKAGE __-> set_primary_key (qw / sale_id /);
__PACKAGE __-> Thuộc_to ('nhân viên', 'Nhân viên',
             {'Foreign.employee_id' => 'self.employee_id',
              'Foreign.subsidiary_id' => 'self.subsidiary_id'});

gói UseTheIndexLuke :: Schema :: Result :: Sales; sử dụng
base qw / UseTheIndexLuke :: Schema :: Result :: SalesHead /;

__PACKAGE __-> table ('sales');
__PACKAGE __-> add_columns (qw / junk /);
```

Lớp Bán hàng có nguồn gốc từ lớp SalesHead và thêm thuộc tính bị thiếu. Bạn có thể sử dụng cả hai lớp khi cần. Xin lưu ý rằng thiết lập bảng cũng được yêu cầu trong lớp dẫn xuất.

Bạn có thể tìm nạp tất cả thông tin chi tiết về nhân viên thông qua tìm nạp trước hoặc chỉ các cột đã chọn như được hiển thị bên dưới:

```
my @sales = $
    schema-> resultset ('SalesHead') -> tìm
             kiếm ($ cond, { , '+ cột' =>
                       ['worker.first_name', 'nhân viên'
                        'worker.last_name']


                       });
```

Không thể chỉ tải các cột đã chọn từ bảng gốc— trong trường hợp này là SalesHead .

[1] http://en.wikipedia.org/wiki/Aliasing_%28computing%29

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

DBIx :: Lớp 0.08192 tạo ra SQL sau. Nó tìm nạp tất cả các cột từ bảng BÁN HÀNG và các thuộc tính đã chọn từ NHÂN VIÊN:

```
CHỌN me.sale_id,
        me.employee_id,
        me.subsidiary_id,
        me.sale_date,
        me.eur_value,
        worker.first_name,
        worker.last_name TỪ
  bán hàng cho tôi THAM GIA
  nhân viên BẬT
         (worker.employee_id = me.employee_id VÀ nhân
        viên.subsidiary_id = me). widget_id)
 WHERE (sale_date>?)
```

Phiên bản PHP 2 của khung Doctrine hỗ trợ lựa chọn thuộc tính trong thời gian chạy. Tài liệu nói rằng các đối tượng được tải một phần có thể hoạt động kỳ lạ và yêu cầu từ khóa part phải thừa nhận rủi ro. Hơn nữa, bạn phải chọn các cột khóa chính một cách rõ ràng:

```
$ qb = $ em-> createQueryBuilder ();
$ qb-> select (' part s. {sale_id, sale_date, eur_value},' . ',
            part e . {worker_id, child_id,. ' first_name -> from
                        ('Sales', 's') -> join ('s.employee ','
   e ') -> where ("s.sale_date>: dt") -> setParameter (' dt ', $
   dt, Type :: DATETIME);
```

SQL được tạo chứa các cột được yêu cầu và một lần nữa SUBSIDIARY_ID và EMPLOYEE_ID từ bảng SALES .

Machine Translated by Google


```
CHỌN s0_.sale_id AS sale_id0,
        s0_.sale_date AS sale_date1,
        s0_.eur_value AS eur_value2,
        e1_.employee_id AS worker_id3,
        e1_.subsidiary_id AS công ty con_id4,
        e1_.first_name AS first_name5,
        e1_.last_name AS last_name6,
        s0_.subsidiary_id AS công ty con_id7,
        s0_.employee_id AS worker_id8
  TỪ doanh số s0_
 Nhân viên INNER JOIN e1_
          TRÊN s0_.subsidiary_id = e1_.subsidiary_id
         VÀ s0_.employee_id = e1_.employee_id
 WHERE s0_.sale_date>?
```

Các đối tượng được trả về tương thích với các đối tượng được tải đầy đủ, nhưng các cột bị thiếu vẫn chưa được khởi tạo. Truy cập chúng không kích hoạt một ngoại lệ.

Cảnh báo

MySQL hoàn toàn không hỗ trợ kết hợp băm (yêu cầu tính năng # 59025)

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Sắp xếp Hợp nhất

Phép nối sắp xếp-hợp nhất kết hợp hai danh sách được sắp xếp giống như một dây kéo. Cả hai vế của phép nối phải được sắp xếp theo các vị từ phép nối.

Một phép nối hợp nhất sắp xếp cần có các chỉ mục giống như phép nối băm, đó là một chỉ mục cho các điều kiện độc lập để đọc tất cả các bản ghi ứng viên trong một lần.
Lập chỉ mục các vị từ nối là vô ích. Mọi thứ chỉ giống như một tham gia băm cho đến nay.
Tuy nhiên, có một khía cạnh duy nhất đối với phép nối sắp xếp-hợp nhất: tính đối xứng tuyệt đối. Thứ tự tham gia không tạo ra bất kỳ sự khác biệt nào — thậm chí không có hiệu quả.
Thuộc tính này rất hữu ích cho các phép nối bên ngoài. Đối với các thuật toán khác, hướng của các phép nối bên ngoài (trái hoặc phải) ngụ ý thứ tự nối — nhưng không phải cho phép nối sắp xếp-hợp nhất. Phép nối sắp xếp-hợp nhất thậm chí có thể thực hiện phép nối ngoài trái và phải cùng một lúc — một phép nối ngoài đầy đủ.

Mặc dù phép nối sắp xếp-hợp nhất hoạt động rất tốt khi các đầu vào được sắp xếp, nhưng nó hầu như không được sử dụng vì việc sắp xếp cả hai bên rất tốn kém. Mặt khác, tham gia băm chỉ cần xử lý trước một mặt.

Sức mạnh của phép kết hợp sắp xếp nổi lên nếu các đầu vào đã được sắp xếp.
Điều này có thể thực hiện được bằng cách khai thác thứ tự chỉ mục để tránh hoàn toàn các thao tác sắp xếp. Chương 6, "Sắp xếp và phân nhóm", giải thích chi tiết khái niệm này.
Tuy nhiên, thuật toán tham gia băm là ưu việt hơn trong nhiều trường hợp.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Chương 5

Phân cụm dữ liệu

Sức mạnh thứ hai của lập chỉ mục

Cụm thuật ngữ được sử dụng trong nhiều lĩnh vực khác nhau. Ví dụ, một cụm sao là một nhóm các ngôi sao. Mặt khác, cụm máy tính là một nhóm các máy tính làm việc chặt chẽ với nhau – để giải quyết một vấn đề phức tạp (cụm máy tính hiệu suất cao) hoặc để tăng tính khả dụng (cụm chuyển đổi dự phòng). Nói chung, các cụm là những thứ liên quan xuất hiện cùng nhau.

Trong lĩnh vực máy tính, có một loại cụm nữa - một loại thường bị hiểu nhầm: cụm dữ liệu. Phân cụm dữ liệu có nghĩa là lưu trữ dữ liệu được truy cập liên tiếp gần nhau để việc truy cập nó yêu cầu ít hoạt động IO hơn. Các cụm dữ liệu rất quan trọng về mặt điều chỉnh cơ sở dữ liệu. Mặt khác, cụm máy tính cũng rất phổ biến trong ngữ cảnh cơ sở dữ liệu - do đó làm cho cụm từ rất mơ hồ. Câu "Hãy sử dụng một cụm để cải thiện hiệu suất cơ sở dữ liệu" chỉ là một ví dụ; nó có thể đề cập đến một cụm máy tính nhưng cũng có thể có nghĩa là một cụm dữ liệu. Trong chương này, cụm thường đề cập đến các cụm dữ liệu.

Cụm dữ liệu đơn giản nhất trong cơ sở dữ liệu SQL là hàng. Cơ sở dữ liệu lưu trữ tất cả các cột của một hàng trong cùng một khối cơ sở dữ liệu nếu có thể. Các ngoại lệ áp dụng nếu một hàng không vừa với một khối đơn lẻ – ví dụ: khi các loại LOB có liên quan.

Cửa hàng cột

Cơ sở dữ liệu hướng theo cột, hoặc lưu trữ cột, tổ chức các bảng theo cách được xếp theo cột. Mô hình này có lợi khi truy cập nhiều hàng nhưng chỉ một vài cột – một mô hình rất phổ biến trong kho dữ liệu (OLAP).

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Các chỉ mục cho phép một nhóm dữ liệu. Cơ sở cho điều này đã được giải thích trong Chương 1, "Giải phẫu một chỉ mục": các nút lá chỉ mục lưu trữ các cột được lập chỉ mục theo kiểu có thứ tự để các giá trị tương tự được lưu trữ bên cạnh nhau. Điều đó có nghĩa là các chỉ mục xây dựng các cụm hàng có giá trị tương tự nhau.

Khả năng phân cụm dữ liệu này quan trọng đến mức tôi coi nó như sức mạnh thứ hai của việc lập chỉ mục.

Các phần sau giải thích cách sử dụng chỉ mục để phân cụm dữ liệu và cải thiện hiệu suất truy vấn.

Dự đoán bộ lọc chỉ mục được sử dụng có chủ ý

Các vị từ bộ lọc chỉ mục rất thường xuyên cho biết việc sử dụng chỉ mục không đúng do thứ tự cột không chính xác trong chỉ mục được nối. Tuy nhiên, các vị từ bộ lọc chỉ mục cũng có thể được sử dụng vì một lý do chính đáng — không phải để cải thiện hiệu suất quét phạm vi mà để nhóm dữ liệu được truy cập liên tục lại với nhau.

Trong đó các vị từ mệnh đề không thể đóng vai trò là vị từ truy cập là những ứng cử viên tốt cho kỹ thuật này:

```
CHỌN first_name, last_name, child_id, phone_number
  TỪ nhân viên
 WHERE công ty con_id =?
   VÀ LÊN (last_name) THÍCH '% INA%';
```

Hãy nhớ rằng biểu thức LIKE với các ký tự đại diện đứng đầu không thể sử dụng cây chỉ mục. Điều đó có nghĩa là việc lập chỉ mục LAST_NAME không thu hẹp phạm vi chỉ mục được quét — bất kể bạn lập chỉ mục LAST_NAME hay UPPER (last_name). Do đó, điều kiện này không phải là ứng cử viên tốt để lập chỉ mục.

Tuy nhiên, điều kiện trên SUBSIDIARY_ID rất phù hợp để lập chỉ mục. Chúng tôi thậm chí không cần thêm chỉ mục mới vì SUBSIDIARY_ID đã là cột hàng đầu trong chỉ mục cho khóa chính.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google


```
-----------------------------------------------------------------------------
| Id | Hoạt động                                    | Tên               | Hàng | Chi phí |
-----------------------------------------------------------------------------
| 0 | TUYỂN CHỌN BÁO CÁO | 17 | 230 |               |
| * 1 | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | TUYỂN NHÂN VIÊN | 17 | 230 |
| * 2 | QUÉT NGUỒN CHỈ SỐ 2 |                        | NHÂN VIÊN_PK | 333 |
-----------------------------------------------------------------------------


Thông tin vị ngữ (được xác định bằng id hoạt động):
---------------------------------------------------

    1 - bộ lọc (UPPER ("LAST_NAME") LIKE '% INA%') 2 - truy cập
    ("SUBSIDIARY_ID" = TO_NUMBER (: A))
```

Trong kế hoạch thực thi ở trên, giá trị chi phí tăng hàng trăm lần từ QUÉT NGUỒN CHỈ SỐ đến thao tác TRUY CẬP BẢNG BẰNG CHỈ SỐ ROWID tiếp theo . Nói cách khác: quyền truy cập bảng gây ra nhiều công việc nhất. Nó thực sự là một khuôn mẫu phổ biến và tự nó không phải là một vấn đề. Tuy nhiên, nó là yếu tố đóng góp đáng kể nhất vào thời gian thực thi tổng thể của truy vấn này.

Quyền truy cập bảng không nhất thiết phải là một nút cổ chai nếu các hàng đã truy cập được lưu trữ trong một khối bảng duy nhất vì cơ sở dữ liệu có thể tìm nạp tất cả các hàng bằng một thao tác đọc duy nhất. Ngược lại, nếu các hàng giống nhau nằm trên nhiều khối khác nhau, thì việc truy cập bảng có thể trở thành một vấn đề nghiêm trọng về hiệu suất vì cơ sở dữ liệu phải tìm nạp nhiều khối để truy xuất tất cả các hàng. Điều đó có nghĩa là hiệu suất phụ thuộc vào phân phối vật lý của các hàng được truy cập — nói cách khác: nó phụ thuộc vào việc phân nhóm các hàng.

Ghi chú

Mối tương quan giữa thứ tự chỉ mục và thứ tự bảng là một tiêu chuẩn hiệu suất — cái gọi là yếu tố phân nhóm chỉ mục.

Trên thực tế, có thể cải thiện hiệu suất truy vấn bằng cách sắp xếp lại các hàng trong bảng để chúng tương ứng với thứ tự chỉ mục. Tuy nhiên, phương pháp này hiếm khi được áp dụng vì bạn chỉ có thể lưu trữ các hàng của bảng trong một trình tự.
Điều đó có nghĩa là bạn có thể tối ưu hóa bảng chỉ cho một chỉ mục. Ngay cả khi bạn có thể chọn một chỉ mục duy nhất mà bạn muốn tối ưu hóa bảng, đó vẫn là một nhiệm vụ khó khăn vì hầu hết các cơ sở dữ liệu chỉ cung cấp các công cụ thô sơ cho nhiệm vụ này. Cái gọi là sắp xếp theo hàng , xét cho cùng, là một cách tiếp cận khá phi thực tế.

Machine Translated by Google

Yếu tố phân nhóm chỉ mục

Yếu tố phân nhóm chỉ số là một thước đo gián tiếp của xác suất rằng hai mục nhập chỉ mục kế tiếp tham chiếu đến cùng một khối bảng. Các trình tối ưu hóa có tính đến xác suất này khi tính toán giá trị chi phí của hoạt động TABLE ACCESS BY INDEX ROWID .

Đây chính xác là nơi xuất hiện sức mạnh thứ hai của lập chỉ mục – phân nhóm dữ liệu – trong. Bạn có thể thêm nhiều cột vào một chỉ mục để chúng tự động được lưu trữ theo một thứ tự được xác định rõ. Điều đó làm cho một chỉ mục trở nên mạnh mẽ nhưng đơn giản công cụ phân cụm dữ liệu.

Để áp dụng khái niệm này cho truy vấn trên, chúng ta phải mở rộng chỉ mục để bao gồm tất cả các cột từ mệnh đề where – ngay cả khi chúng không thu hẹp phạm vi đã quét phạm vi chỉ mục:

```
TẠO INDEX empsubupnam TRÊN nhân viên
          (tên_mục_thước, UPPER (tên_mục cuối));
```

Cột SUBSIDIARY_ID là cột chỉ mục đầu tiên nên nó có thể được sử dụng làm một vị từ truy cập. Biểu thức UPPER (last_name) bao gồm bộ lọc LIKE làm vị ngữ bộ lọc chỉ mục. Lập chỉ mục biểu diễn chữ hoa tiết kiệm được một vài CPU chu kỳ trong quá trình thực thi, nhưng một chỉ mục thẳng trên LAST_NAME sẽ hoạt động cũng. Bạn sẽ tìm thấy thêm về điều này trong phần tiếp theo.

```
-------------------------------------------------------------------------
| Id | Hoạt động                                | Tên               | Hàng | Chi phí |
-------------------------------------------------------------------------
| 0 | TUYỂN CHỌN BÁO CÁO | | 1                      |                        17 | 20 |
      | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | TUYỂN NHÂN VIÊN |                  17 | 20 |
| * 2 | QUÉT NGUỒN CHỈ SỐ                          | EMPSUBUPNAM | 17 | 3 |
-------------------------------------------------------------------------


Thông tin vị ngữ (được xác định bằng id hoạt động):
---------------------------------------------------

    2 - truy cập ("SUBSIDIARY_ID" = TO_NUMBER (: A))
          bộ lọc (UPPER ("LAST_NAME") LIKE '% INA%')
```

Kế hoạch thực hiện mới cho thấy các hoạt động giống như trước đây. Chi phí Tuy nhiên, giá trị đã giảm đáng kể. Trong thông tin vị ngữ, chúng tôi có thể thấy rằng bộ lọc THÍCH đã được áp dụng trong QUÉT NGUỒN CHỈ SỐ. Các dòng không đáp ứng bộ lọc LIKE sẽ bị loại bỏ ngay lập tức. Cái bàn quyền truy cập không có bất kỳ vị từ bộ lọc nào nữa. Điều đó có nghĩa là nó không tải các hàng không đáp ứng mệnh đề where .

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Sự khác biệt giữa hai kế hoạch thực hiện có thể nhìn thấy rõ ràng trong cột "Hàng". Theo ước tính của trình tối ưu hóa, truy vấn cuối cùng khớp với 17 bản ghi. Tuy nhiên, việc quét chỉ mục trong kế hoạch thực thi đầu tiên cung cấp 333 hàng. Sau đó, cơ sở dữ liệu phải tải 333 hàng này từ bảng để áp dụng bộ lọc LIKE làm giảm kết quả xuống còn 17 hàng. Trong kế hoạch thực thi thứ hai, quyền truy cập chỉ mục không cung cấp các hàng đó ngay từ đầu vì vậy cơ sở dữ liệu chỉ cần thực hiện thao tác TABLE ACCESS BY INDEX ROWID chỉ 17 lần.

Bạn cũng nên lưu ý rằng giá trị chi phí của hoạt động QUÉT CHỈ SỐ đã tăng từ hai lên ba vì cột bổ sung làm cho chỉ mục lớn hơn. Theo quan điểm của hiệu suất đạt được, đó là một thỏa hiệp có thể chấp nhận được.

Cảnh báo

Không giới thiệu một chỉ mục mới cho mục đích duy nhất là các vị từ bộ lọc. Thay vào đó, hãy mở rộng chỉ mục hiện có và giữ cho nỗ lực bảo trì ở mức thấp. Với một số cơ sở dữ liệu, bạn thậm chí có thể thêm các cột vào chỉ mục cho khóa chính không phải là một phần của khóa chính.

Ví dụ nhỏ này dường như xác nhận sự khôn ngoan thông thường để lập chỉ mục mọi cột từ mệnh đề where . Tuy nhiên, “sự khôn ngoan” này bỏ qua mức độ liên quan của thứ tự cột xác định những điều kiện nào có thể được sử dụng làm vị từ truy cập và do đó có tác động rất lớn đến hiệu suất. Do đó, quyết định về thứ tự cột không bao giờ được phó mặc cho sự ngẫu nhiên.

Kích thước chỉ mục cũng tăng theo số lượng cột — đặc biệt là khi thêm các cột văn bản. Tất nhiên hiệu suất không trở nên tốt hơn đối với một chỉ số lớn hơn mặc dù khả năng mở rộng logarit hạn chế tác động đáng kể. Bạn không nên thêm tất cả các cột được đề cập trong mệnh đề where vào một chỉ mục mà thay vào đó chỉ sử dụng các vị từ bộ lọc chỉ mục có chủ đích để giảm khối lượng dữ liệu trong một bước thực thi trước đó.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Chỉ quét chỉ mục

Quét chỉ theo chỉ mục là một trong những phương pháp điều chỉnh mạnh mẽ nhất. Nó không chỉ tránh truy cập vào bảng để đánh giá mệnh đề where , mà còn tránh hoàn toàn truy cập vào bảng nếu cơ sở dữ liệu có thể tìm thấy các cột trong chính chỉ mục.

Để bao gồm toàn bộ truy vấn, một chỉ mục phải chứa tất cả các cột từ SQL câu lệnh – đặc biệt là các cột từ mệnh đề select như được hiển thị trong ví dụ sau:

```
TẠO INDEX sales_sub_eur
      Đang bán
        (giá_trị_cấp, eur_value );

CHỌN SUM (eur_value)
   TỪ bán hàng
  WHERE công ty con = ?;
```

Tất nhiên, lập chỉ mục mệnh đề where được ưu tiên hơn mệnh đề kia điều khoản. Do đó, cột SUBSIDIARY_ID ở vị trí đầu tiên nên nó đủ điều kiện như một vị từ truy cập.

Kế hoạch thực thi hiển thị quá trình quét chỉ mục mà không có quyền truy cập bảng tiếp theo (TRUY CẬP BẢNG THEO CHỈ SỐ ROWID).

```
--------------------------------------------------------------------------------

| Id | Hoạt động                          | Tên                     | Hàng | Chi phí |

--------------------------------------------------------------------------------

| | |0104TUYỂN CHỌN BÁO CÁO | 1
…         | SẮP XẾP TỔNG HỢP |                                                  1 1
| * 2 | QUÉT NGUỒN CHỈ SỐ | SALES_SUB_EUR | 40388 | 104 |
--------------------------------------------------------------------------------


Thông tin vị ngữ (được xác định bằng id hoạt động):
---------------------------------------------------

     2 - truy cập ("SUBSIDIARY_ID" = TO_NUMBER (: A))
```

Chỉ mục bao gồm toàn bộ truy vấn nên nó còn được gọi là chỉ mục bao trùm.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Ghi chú

Nếu một chỉ mục ngăn truy cập bảng thì nó cũng được gọi là chỉ mục bao trùm.

Tuy nhiên, thuật ngữ này gây hiểu lầm vì nó nghe giống như một thuộc tính chỉ mục. Việc quét chỉ chỉ mục cụm từ gợi ý một cách chính xác rằng đó là một hoạt động kế hoạch thực thi.

Chỉ mục có một bản sao của cột EUR_VALUE để cơ sở dữ liệu có thể sử dụng giá trị được lưu trữ trong chỉ mục. Truy cập bảng là không bắt buộc vì chỉ mục có tất cả thông tin để đáp ứng truy vấn.

Quét chỉ theo chỉ mục có thể cải thiện hiệu suất rất nhiều. Chỉ cần nhìn vào ước tính số hàng trong kế hoạch thực thi: trình tối ưu hóa dự kiến tổng hợp hơn 40.000 hàng. Điều đó có nghĩa là quá trình quét chỉ theo chỉ mục ngăn chặn 40.000 lần tìm nạp bảng — nếu mỗi hàng nằm trong một khối bảng khác nhau. Nếu chỉ mục có hệ số phân cụm tốt — nghĩa là, nếu các hàng tương ứng được phân cụm tốt trong một vài khối bảng — thì lợi thế có thể thấp hơn đáng kể.

Bên cạnh yếu tố phân nhóm, số lượng hàng được chọn giới hạn mức tăng hiệu suất tiềm năng của quá trình quét chỉ chỉ mục. Ví dụ: nếu bạn chọn một hàng, bạn chỉ có thể lưu một quyền truy cập bảng duy nhất. Xem xét rằng quá trình duyệt cây cũng cần tìm nạp một vài khối, quyền truy cập bảng đã lưu có thể trở nên không đáng kể.

Quan trọng

Lợi thế về hiệu suất của quét chỉ theo chỉ mục phụ thuộc vào số lượng hàng được truy cập và hệ số phân cụm chỉ mục.

Chỉ quét chỉ mục là một chiến lược lập chỉ mục tích cực. Không thiết kế chỉ mục để quét chỉ chỉ mục khi nghi ngờ chỉ vì nó sử dụng bộ nhớ một cách không cần thiết và tăng nỗ lực bảo trì cần thiết cho các câu lệnh cập nhật . Xem Chương 8, “Sửa đổi dữ liệu”. Trong thực tế, trước tiên bạn nên lập chỉ mục mà không cần xem xét mệnh đề select và chỉ mở rộng chỉ mục nếu cần.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Quét chỉ mục cũng có thể gây ra những bất ngờ khó chịu, chẳng hạn như nếu chúng tôi giới hạn truy vấn đối với doanh số bán hàng gần đây:

```
CHỌN SUM (eur_value)
   TỪ bán hàng
 WHERE công ty con =?
    VÀ ngày bán> ?;
```

Nếu không nhìn vào kế hoạch thực thi, người ta có thể mong đợi truy vấn chạy nhanh hơn vì nó chọn ít hàng hơn. Tuy nhiên , mệnh đề where đề cập đến một cột không có trong chỉ mục để cơ sở dữ liệu phải truy cập vào bảng để tải cột này.

```
--------------------------------------------------------------------------------
| Id | Hoạt động                                    | Tên              | Hàng | Chi phí |
--------------------------------------------------------------------------------
| 0 | TUYỂN CHỌN BÁO CÁO | | 371 |
| 1   | SẮP XẾP TỔNG HỢP | | |                                                  1 1
| * 2 | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | BÁN HÀNG | Năm 2019 | 371 |
| * 3 |       QUÉT NGUỒN CHỈ SỐ                      | SALES_DATE | 10541 | 30 |
--------------------------------------------------------------------------------

Thông tin vị ngữ (được xác định bằng id hoạt động):
---------------------------------------------------

     2 - bộ lọc ("SUBSIDIARY_ID" = TO_NUMBER (: A))
     3 - truy cập ("SALE_DATE">: B)
```

Quyền truy cập bảng làm tăng thời gian phản hồi mặc dù truy vấn chọn ít hàng hơn. Yếu tố có liên quan không phải là số hàng mà truy vấn phân phối nhưng cơ sở dữ liệu phải kiểm tra bao nhiêu hàng để tìm thấy chúng.

Cảnh báo

Việc mở rộng mệnh đề where có thể gây ra hiệu suất "phi logic" hành vi. Kiểm tra kế hoạch thực thi trước khi mở rộng các truy vấn.

Nếu một chỉ mục không còn có thể được sử dụng cho quá trình quét chỉ chỉ mục, trình tối ưu hóa sẽ chọn kế hoạch thực hiện tốt nhất tiếp theo. Điều đó có nghĩa là trình tối ưu hóa có thể chọn một kế hoạch thực hiện hoàn toàn khác hoặc, như trên, một kế hoạch thực hiện tương tự với một chỉ mục khác. Trong trường hợp này, nó sử dụng một chỉ mục vào SALE_DATE, là một còn sót lại từ chương trước.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Từ quan điểm của trình tối ưu hóa, chỉ mục này có hai lợi thế so với SALES_SUB_EUR. Trình tối ưu hóa tin rằng bộ lọc vào SALE_DATE có tính chọn lọc cao hơn bộ lọc trên SUBSIDIARY_ID . Bạn có thể thấy điều đó trong cột "Hàng" tương ứng của hai kế hoạch thực hiện cuối cùng (khoảng 10.000 so với 40.000).
Tuy nhiên, những ước tính này hoàn toàn là tùy ý vì truy vấn sử dụng các tham số ràng buộc. Ví dụ: điều kiện SALE_DATE có thể chọn toàn bộ bảng khi cung cấp ngày bán hàng đầu tiên.

Ưu điểm thứ hai của chỉ mục SALES_DATE là có hệ số phân cụm tốt hơn. Đây là một lý do hợp lệ vì bảng SALES chỉ phát triển theo thứ tự thời gian. Các hàng mới luôn được nối vào cuối bảng miễn là không có hàng nào bị xóa. Do đó, thứ tự bảng tương ứng với thứ tự chỉ mục vì cả hai đều được sắp xếp gần đúng theo thứ tự thời gian – chỉ mục có hệ số phân cụm tốt.

Khi sử dụng chỉ mục có hệ số phân cụm tốt, các hàng của bảng đã chọn được lưu trữ gần nhau để cơ sở dữ liệu chỉ cần đọc một vài khối bảng để nhận được tất cả các hàng. Sử dụng chỉ mục này, truy vấn có thể đủ nhanh mà không cần quét chỉ chỉ mục. Trong trường hợp này, chúng ta nên xóa các cột không cần thiết khỏi chỉ mục khác một lần nữa.

Ghi chú

Một số chỉ mục có hệ số phân cụm tốt tự động để lợi thế về hiệu suất của quá trình quét chỉ chỉ mục là tối thiểu.

Trong ví dụ cụ thể này, có một sự trùng hợp đáng mừng. Bộ lọc mới vào SALE_DATE không chỉ ngăn quá trình quét chỉ lập chỉ mục mà còn mở ra một đường dẫn truy cập mới cùng lúc. Do đó, trình tối ưu hóa có thể hạn chế tác động đến hiệu suất của thay đổi này. Tuy nhiên, cũng có thể ngăn chỉ quét chỉ mục bằng cách thêm cột vào các mệnh đề khác. Tuy nhiên, việc thêm một cột vào mệnh đề select không bao giờ có thể mở ra một đường dẫn truy cập mới, điều này có thể hạn chế tác động của việc mất quét chỉ mục.

Mẹo

Duy trì các bản quét chỉ lập chỉ mục của bạn.

Thêm nhận xét nhắc nhở bạn về quá trình quét chỉ lập chỉ mục và tham chiếu đến trang đó để bất kỳ ai cũng có thể đọc về nó.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Các chỉ mục dựa trên chức năng cũng có thể gây ra những bất ngờ khó chịu liên quan đến việc quét chỉ theo chỉ mục. Không thể sử dụng chỉ mục trên UPPER (last_name) để quét chỉ chỉ mục khi chọn cột LAST_NAME . Trong phần trước, chúng ta nên lập chỉ mục cho chính cột LAST_NAME để hỗ trợ bộ lọc LIKE và cho phép nó được sử dụng để quét chỉ lập chỉ mục khi chọn cột LAST_NAME .

Mẹo

Luôn hướng tới việc lập chỉ mục dữ liệu gốc vì đó thường là thông tin hữu ích nhất mà bạn có thể đưa vào chỉ mục.

Tránh lập chỉ mục dựa trên hàm cho các biểu thức không thể được sử dụng làm vị từ truy cập.

Các truy vấn tổng hợp như được hiển thị ở trên tạo nên những ứng cử viên tốt cho việc quét chỉ theo chỉ mục. Họ truy vấn nhiều hàng nhưng chỉ một vài cột, tạo ra một chỉ mục mỏng đủ để hỗ trợ quét chỉ chỉ mục. Bạn càng truy vấn nhiều cột, bạn càng phải thêm nhiều cột vào lập chỉ mục để hỗ trợ quá trình quét chỉ lập chỉ mục. Là một nhà phát triển, do đó, bạn chỉ nên chọn những cột bạn thực sự cần.

Mẹo

Tránh chọn * và chỉ tìm nạp các cột bạn cần.

Bất kể thực tế là lập chỉ mục nhiều hàng cần nhiều dung lượng, bạn cũng có thể đạt đến giới hạn của cơ sở dữ liệu của mình. Hầu hết các cơ sở dữ liệu áp đặt các giới hạn khá cứng nhắc về số lượng cột trên mỗi chỉ mục và tổng kích thước của một mục nhập chỉ mục. Điều đó có nghĩa là bạn không thể lập chỉ mục một số cột tùy ý cũng như các cột dài tùy ý. Tổng quan sau đây liệt kê những hạn chế quan trọng nhất. Tuy nhiên, có những chỉ mục bao gồm toàn bộ bảng như chúng ta thấy trong phần tiếp theo.

Hãy nghĩ về nó

Các truy vấn không chọn bất kỳ cột nào trong bảng thường được thực hiện bằng cách quét chỉ chỉ mục.

Bạn có thể nghĩ về một ví dụ có ý nghĩa không?

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

MySQL

MySQL 5.6 với InnoDB giới hạn mỗi cột đơn lẻ là 767 byte và tất cả các cột cùng nhau là 3072 byte. Chỉ mục MyISAM được giới hạn trong 16 cột và độ dài khóa tối đa là 1000 byte.

MySQL có một tính năng duy nhất được gọi là "lập chỉ mục tiền tố" (đôi khi còn được gọi là "lập chỉ mục một phần"). Điều này có nghĩa là chỉ lập chỉ mục một số ký tự đầu tiên của một cột — vì vậy nó không liên quan gì đến các chỉ mục từng phần được mô tả trong Chương 2. Nếu bạn lập chỉ mục một cột vượt quá độ dài cột cho phép (767 byte cho InnoDB), MySQL sẽ tự động cắt bớt cột cho phù hợp. Đây là lý do tại sao câu lệnh tạo chỉ mục thành công với cảnh báo "Khóa được chỉ định quá dài; độ dài khóa tối đa là 767 byte "nếu bạn vượt quá giới hạn. Điều đó có nghĩa là chỉ mục không chứa bản sao đầy đủ của cột nữa và do đó chỉ được sử dụng hạn chế cho việc quét chỉ chỉ mục (tương tự như chỉ mục dựa trên hàm).

Bạn có thể sử dụng lập chỉ mục tiền tố của MySQL một cách rõ ràng để ngăn vượt quá giới hạn tổng độ dài khóa nếu bạn nhận được thông báo lỗi "Khóa được chỉ định quá dài; độ dài khóa tối đa là [1000/3072] byte. " Ví dụ sau chỉ lập chỉ mục mười ký tự đầu tiên của cột LAST_NAME .

```
TẠO CHỈ SỐ        .. ON nhân viên (last_name (10));
```

Cơ sở dữ liệu Oracle

Độ dài khóa chỉ mục tối đa phụ thuộc vào kích thước khối và các tham số lưu trữ chỉ mục (75% kích thước khối cơ sở dữ liệu trừ đi một số chi phí). Chỉ mục cây B được giới hạn trong 32 cột.

Khi sử dụng Oracle 11g với tất cả các giá trị mặc định tại chỗ (8k khối), độ dài khóa chỉ mục tối đa là 6398 byte. Vượt quá giới hạn này gây ra thông báo lỗi "ORA-01450: đã vượt quá độ dài khóa tối đa (6398)."

PostgreSQL Cơ

sở dữ liệu PostgreSQL hỗ trợ quét chỉ mục kể từ bản phát hành 9.2.

Độ dài khóa của chỉ mục cây B được giới hạn ở 2713 byte (mã cứng, xấp xỉ BLCKSZ / 3). Thông báo lỗi tương ứng "kích thước hàng chỉ mục ... vượt quá btree tối đa, 2713" chỉ xuất hiện khi thực hiện chèn hoặc cập nhật vượt quá giới hạn. Chỉ mục cây B có thể chứa tối đa 32 cột.

Machine Translated by Google

SQL Server

SQL Server giới hạn độ dài khóa là 900 byte và 16 cột khóa.
Tuy nhiên, SQL Server có một tính năng cho phép bạn thêm các cột dài tùy ý vào một chỉ mục với mục đích duy nhất là hỗ trợ quét chỉ chỉ mục. Vì vậy, SQL Server phân biệt giữa cột khóa và cột không khóa.

Các cột chính là các cột chỉ mục như chúng đã được thảo luận cho đến nay. Các cột không khóa là các cột bổ sung chỉ được lưu trữ trong các nút lá chỉ mục. Các cột nonkey có thể dài tùy ý nhưng không thể được sử dụng làm vị từ truy cập (tìm vị từ).

Các cột nonkey được xác định bằng từ khóa include của lệnh tạo chỉ mục :

```
 TẠO INDEX empsubupnam
      BẬT nhân viên
         (công ty con_id, last_name)
BAO GỒM (số_điện_đầu, tên_người);
```

Bảng được tổ chức theo chỉ mục

Quá trình quét chỉ chỉ mục thực thi một câu lệnh SQL chỉ sử dụng dữ liệu dư thừa được lưu trữ trong chỉ mục. Dữ liệu gốc trong bảng heap là không cần thiết.
Nếu chúng ta đưa khái niệm đó lên cấp độ tiếp theo và đặt tất cả các cột vào chỉ mục, bạn có thể tự hỏi tại sao chúng ta cần bảng heap.

Một số cơ sở dữ liệu thực sự có thể sử dụng một chỉ mục làm nơi lưu trữ bảng chính. Cơ sở dữ liệu Oracle gọi khái niệm này là các bảng được tổ chức theo chỉ mục (IOT), các cơ sở dữ liệu khác sử dụng thuật ngữ chỉ mục phân cụm. Trong phần này, cả hai thuật ngữ đều được sử dụng để nhấn mạnh vào bảng hoặc các đặc điểm của chỉ mục khi cần thiết.

Do đó, một bảng được tổ chức theo chỉ mục là một chỉ mục B-tree không có bảng heap. Điều này dẫn đến hai lợi ích: (1) nó tiết kiệm không gian cho cấu trúc heap; (2) mọi truy cập trên một chỉ mục được phân nhóm sẽ tự động là một quá trình quét chỉ theo chỉ mục. Cả hai lợi ích nghe có vẻ hứa hẹn nhưng khó có thể đạt được trong thực tế.

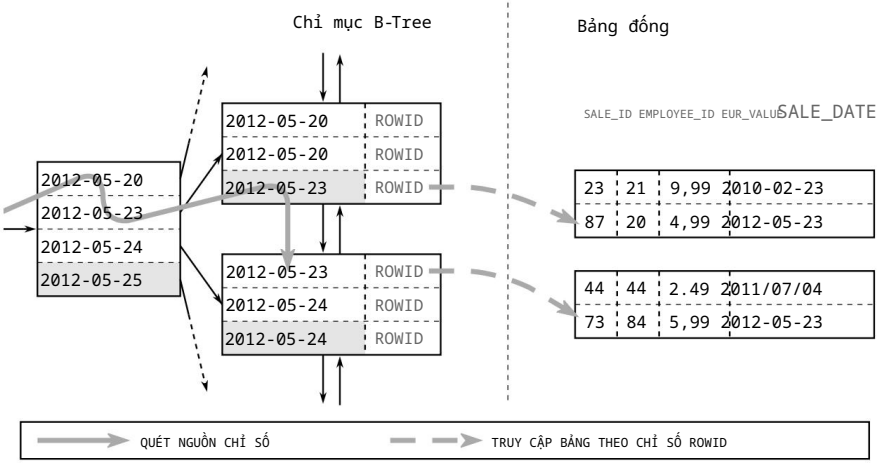
Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Hạn chế của một bảng được tổ chức theo chỉ mục trở nên rõ ràng khi tạo một chỉ mục khác trên cùng một bảng. Tương tự với một chỉ mục thông thường, cái gọi là chỉ mục phụ đề cập đến dữ liệu bảng gốc – được lưu trữ trong chỉ mục được phân cụm. Ở đó, dữ liệu không được lưu trữ tĩnh như trong bảng heap mà có thể di chuyển bất cứ lúc nào để duy trì thứ tự chỉ mục. Do đó, không thể lưu trữ vị trí thực của các hàng trong bảng được tổ chức theo chỉ mục trong chỉ mục phụ. Cơ sở dữ liệu phải sử dụng khóa logic để thay thế.

Các số liệu sau đây cho thấy một bảng tra cứu chỉ mục để tìm tất cả doanh số bán hàng vào ngày 23 tháng 5 năm 2012. Để so sánh, trước tiên chúng ta sẽ xem Hình 5.1 cho thấy quy trình khi sử dụng bảng heap. Việc thực hiện bao gồm hai bước: (1) QUÉT NGUỒN CHỈ SỐ; (2) TRUY CẬP BẢNG THEO CHỈ SỐ ROWID.

Hình 5.1. Truy cập dựa trên chỉ mục trên một bảng Heap



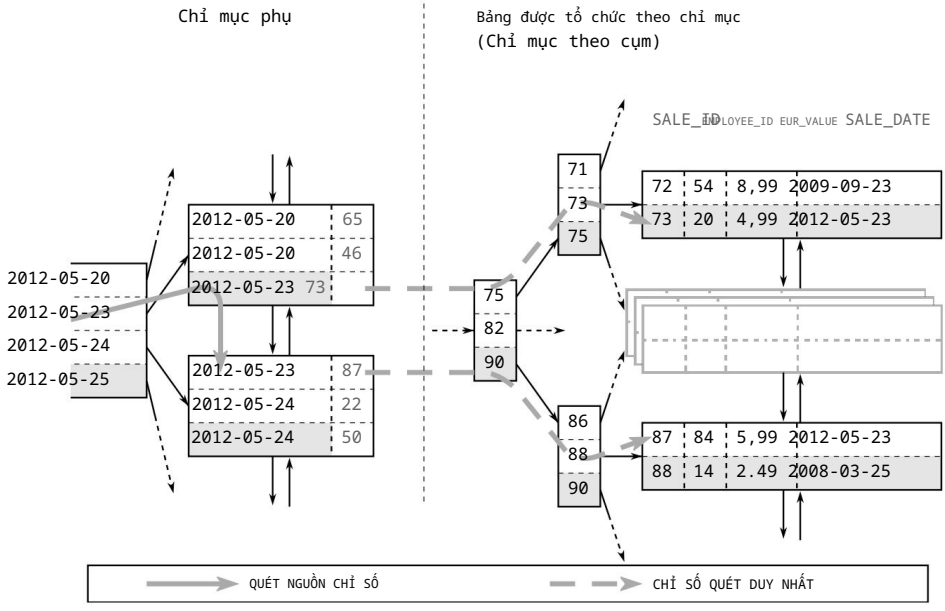
Mặc dù quyền truy cập bảng có thể trở thành tắc nghẽn, nhưng nó vẫn bị giới hạn ở một thao tác đọc trên mỗi hàng vì chỉ mục có ROWID làm con trỏ trực tiếp đến hàng trong bảng. Cơ sở dữ liệu có thể tải ngay hàng từ bảng heap vì chỉ mục có vị trí chính xác của nó. Tuy nhiên, hình ảnh sẽ thay đổi khi sử dụng một chỉ mục phụ trên một bảng được tổ chức theo chỉ mục. Chỉ mục phụ không lưu trữ một con trỏ vật lý (ROWID) mà chỉ lưu trữ các giá trị khóa của chỉ mục được phân nhóm – cái gọi là khóa phân cụm. Thường thì đó là khóa chính của bảng được tổ chức theo chỉ mục.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Việc truy cập chỉ mục phụ không cung cấp ROWID mà là một khóa logic để tìm kiếm chỉ mục được phân nhóm. Tuy nhiên, một truy cập đơn lẻ là không đủ để tìm kiếm chỉ mục được phân cụm – nó yêu cầu truyền tải toàn cây. Điều đó có nghĩa là việc truy cập bảng thông qua chỉ mục phụ sẽ tìm kiếm hai chỉ mục: chỉ mục phụ một lần (INDEX RANGE SCAN), sau đó là chỉ mục nhóm cho mỗi hàng được tìm thấy trong chỉ mục phụ (INDEX UNIQUE SCAN).

Hình 5.2. Chỉ mục phụ trên IOT



Hình 5.2 cho thấy rõ ràng rằng cây B của chỉ mục được phân cụm đứng giữa chỉ mục phụ và dữ liệu bảng.

Việc truy cập bảng được tổ chức theo chỉ mục thông qua chỉ mục phụ là rất kém hiệu quả và nó có thể bị ngăn chặn giống như cách ngăn truy cập bảng trên bảng heap: bằng cách sử dụng quét chỉ chỉ mục – trong trường hợp này được mô tả tốt hơn là "thứ cấp-chỉ quét chỉ mục ". Lợi thế về hiệu suất của quét chỉ theo chỉ mục thậm chí còn lớn hơn bởi vì nó không chỉ ngăn chặn một truy cập duy nhất mà là toàn bộ CHỈ SỐ QUÉT DUY NHẤT.

Quan trọng Việc

truy cập một bảng được sắp xếp theo chỉ mục thông qua một chỉ mục phụ là rất kém hiệu quả.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Sử dụng ví dụ này, chúng ta cũng có thể thấy rằng cơ sở dữ liệu khai thác tất cả dư thừa mà họ có. Hãy nhớ rằng một chỉ mục phụ lưu trữ khóa phân cụm cho mỗi mục nhập chỉ mục. Do đó, chúng tôi có thể truy vấn khóa phân cụm từ một chỉ mục phụ mà không cần truy cập vào bảng được tổ chức chỉ mục:

```
SELECT sale_id
   FROM sales_iot
 Nơi bán_date = ?;

-----------------------------------------------
| Id | Hoạt động                    | Tên                    | Chi phí |
-----------------------------------------------
| 0 | TUYỂN CHỌN BÁO CÁO | | | * 1                                  4 |
        | QUÉT NGUỒN CHỈ SỐ | SALES_IOT_DATE |                     4 |
-----------------------------------------------


Thông tin vị ngữ (được xác định bằng id hoạt động):
---------------------------------------------

     1 - truy cập ("SALE_DATE" =: DT)
```

Bảng SALES_IOT là một bảng được tổ chức theo chỉ mục sử dụng SALE_ID làm khóa cụm. Mặc dù chỉ số SALE_IOT_DATE là vào SALE_DATE chỉ, nó vẫn có một bản sao của khóa phân cụm SALE_ID để nó có thể đáp ứng truy vấn chỉ sử dụng chỉ mục phụ.

Khi chọn các cột khác, cơ sở dữ liệu phải chạy một CHỈ SỐ QUÉT DUY NHẤT trên chỉ mục được nhóm cho mỗi hàng:

```
CHỌN eur_value
   FROM sales_iot
 Nơi bán_date = ?;

---------------------------------------------

| Id | Hoạt động                       | Tên                     | Chi phí |
---------------------------------------------

|     0 | TUYỂN CHỌN BÁO CÁO | | | * 1                                 13 |
          | CHỈ SỐ QUÉT ĐỘC ĐÁO | SALES_IOT_PK |                      13 |
| * 2 | QUÉT NGUỒN CHỈ SỐ | SALES_IOT_DATE |                           4 |
---------------------------------------------


Thông tin vị ngữ (được xác định bằng id hoạt động):
---------------------------------------------

     1 - truy cập ("SALE_DATE" =: DT)
     2 - truy cập ("SALE_DATE" =: DT)
```

Machine Translated by Google

Rốt cuộc, các bảng được tổ chức theo chỉ mục và các chỉ mục được phân cụm không hữu ích như nó có vẻ như ngay từ cái nhìn đầu tiên. Các cải tiến về hiệu suất trên chỉ mục được nhóm lại dễ dàng bị mất khi sử dụng chỉ mục phụ. Khóa phân cụm thường dài hơn ROWID để các chỉ mục phụ lớn hơn so với chúng trên bảng heap, thường loại bỏ khoản tiết kiệm từ việc bỏ qua bảng heap. Sức mạnh của các bảng được tổ chức theo chỉ mục và các chỉ mục theo nhóm hầu hết được giới hạn ở các bảng không cần chỉ mục thứ hai. Bảng đống có lợi ích là cung cấp một bản sao chính tĩnh có thể được tham chiếu dễ dàng.

Các bảng quan

trọng chỉ có một chỉ mục được triển khai tốt nhất dưới dạng chỉ mục được nhóm hoặc bảng được tổ chức theo chỉ mục.

Các bảng có nhiều chỉ mục hơn thường có thể được hưởng lợi từ bảng đống. Bạn vẫn có thể sử dụng quét chỉ mục để tránh truy cập bảng. Điều này cung cấp cho bạn hiệu suất được chọn của một chỉ mục được phân nhóm mà không làm chậm các chỉ mục khác.

Hỗ trợ cơ sở dữ liệu cho các bảng được tổ chức theo chỉ mục và chỉ mục theo nhóm là rất không nhất quán. Tổng quan trên trang tiếp theo giải thích các chi tiết cụ thể quan trọng nhất.

Tại sao Chỉ mục phụ không có ROWID

Một con trỏ trực tiếp đến hàng của bảng cũng sẽ được mong muốn cho một chỉ mục phụ. Nhưng điều đó chỉ có thể xảy ra, nếu hàng của bảng vẫn ở các vị trí lưu trữ cố định. Điều đó, thật không may, không thể thực hiện được nếu hàng là một phần của cấu trúc chỉ mục, được giữ theo thứ tự. Giữ trật tự chỉ mục cần thỉnh thoảng di chuyển các hàng. Điều này cũng đúng đối với các thao tác không ảnh hưởng đến chính hàng. Ví dụ, một câu lệnh chèn có thể tách một nút lá để lấy không gian cho mục nhập mới. Điều đó có nghĩa là một số mục nhập được chuyển đến một khối dữ liệu mới ở một nơi khác.

Mặt khác, một bảng đống không giữ các hàng theo bất kỳ thứ tự nào.
Cơ sở dữ liệu lưu các mục nhập mới ở bất kỳ nơi nào nó tìm thấy đủ dung lượng. Sau khi được viết, dữ liệu không di chuyển trong bảng đống.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

MySQL

Công cụ MyISAM chỉ sử dụng bảng đống trong khi công cụ InnoDB luôn sử dụng các chỉ mục được phân cụm. Điều đó có nghĩa là bạn không trực tiếp có sự lựa chọn.

Cơ sở dữ liệu Oracle

Cơ sở dữ liệu Oracle sử dụng bảng heap theo mặc định. Các bảng được tổ chức theo chỉ mục có thể được tạo bằng mệnh đề ORGANIZATION INDEX :

```
TẠO BẢNG (id SỐ
    KHÔNG ĐẦY ĐỦ,
    KHÓA CHÍNH, [...]

) CHỈ SỐ TỔ CHỨC;
```

Cơ sở dữ liệu Oracle luôn sử dụng khóa chính làm khóa phân cụm.

PostgreSQL

PostgreSQL chỉ sử dụng bảng heap.

Tuy nhiên, bạn có thể sử dụng mệnh đề CLUSTER để căn chỉnh nội dung của bảng heap với một chỉ mục.

Máy chủ SQL

Theo mặc định, SQL Server sử dụng chỉ mục được phân cụm (bảng được tổ chức theo chỉ mục) bằng cách sử dụng khóa chính làm khóa phân cụm. Tuy nhiên, bạn có thể sử dụng các cột tùy ý cho khóa phân cụm — ngay cả các cột không phải là duy nhất.

Để tạo bảng heap, bạn phải sử dụng mệnh đề KHÔNG CHỈNH SỬA trong định nghĩa khóa chính:

```
TẠO BẢNG (id SỐ
    KHÔNG ĐẦY ĐỦ,
    [...]

    CONSTRAINT pk KHÓA CHÍNH KHÔNG ĐƯỢC ĐIỀU CHỈNH (id)
);
```

Việc giảm một chỉ mục được phân nhóm sẽ biến bảng thành một bảng heap.

Hành vi mặc định của SQL Server thường gây ra các vấn đề về hiệu suất khi sử dụng các chỉ mục phụ.

Machine Translated by Google

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Chương 6

Sắp xếp và phân nhóm

Sắp xếp là một hoạt động rất tốn tài nguyên. Nó cần một lượng thời gian tương đối của CPU, nhưng vấn đề chính là cơ sở dữ liệu phải tạm thời đệm kết quả. Rốt cuộc, một hoạt động sắp xếp phải đọc đầu vào hoàn chỉnh trước khi nó có thể tạo ra đầu ra đầu tiên. Các hoạt động sắp xếp không thể được thực hiện theo cách thức tổng hợp – điều này có thể trở thành một vấn đề đối với các tập dữ liệu lớn.

Một chỉ mục cung cấp một biểu diễn theo thứ tự của dữ liệu được lập chỉ mục: nguyên tắc này đã được mô tả trong Chương 1. Chúng ta cũng có thể nói rằng một chỉ mục lưu trữ dữ liệu theo kiểu được sắp xếp trước. Trên thực tế, chỉ mục được sắp xếp giống như khi sử dụng định nghĩa chỉ mục theo thứ tự từng mệnh đề. Do đó, không có gì ngạc nhiên khi chúng ta có thể sử dụng các chỉ mục để tránh thao tác sắp xếp để đáp ứng một mệnh đề theo thứ tự .

Trớ trêu thay, QUÉT NGUỒN CHỈ SỐ cũng trở nên không hiệu quả đối với các tập dữ liệu lớn – đặc biệt là khi theo sau là truy cập bảng. Điều này có thể vô hiệu hóa khoản tiết kiệm tránh được hoạt động sắp xếp. QUÉT BẢNG ĐẦY ĐỦ với thao tác sắp xếp rõ ràng có thể còn nhanh hơn trong trường hợp này. Một lần nữa, công việc của trình tối ưu hóa là đánh giá các kế hoạch thực thi khác nhau và chọn kế hoạch tốt nhất.

Tuy nhiên, một đơn đặt hàng được lập chỉ mục bằng cách thực thi không chỉ tiết kiệm nỗ lực sắp xếp; nó cũng có thể trả về kết quả đầu tiên mà không cần xử lý tất cả dữ liệu đầu vào. Do đó, lệnh của được thực hiện theo một phương thức pipelined . Chương 7, “Kết quả từng phần”, giải thích cách khai thác việc thực thi pipelined để thực hiện các truy vấn phân trang hiệu quả. Điều này làm cho thứ tự pipelined trở nên quan trọng đến mức tôi coi nó như là sức mạnh thứ ba của lập chỉ mục.

Chương này giải thích cách sử dụng một chỉ mục cho một lệnh pipelined bằng cách thực thi. Để đạt được điều này, chúng ta phải đặc biệt chú ý đến các tương tác với mệnh đề where cũng như các bổ ngữ ASC và DESC . Chương này kết thúc bằng cách áp dụng các kỹ thuật này để nhóm theo các mệnh đề.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Lập chỉ mục Thứ tự Theo

Các truy vấn SQL với mệnh đề thứ tự không cần phải sắp xếp kết quả một cách rõ ràng nếu chỉ mục liên quan đã phân phối các hàng theo thứ tự bắt buộc. Cái đó có nghĩa là cùng một chỉ mục được sử dụng cho mệnh đề where cũng phải bao gồm thứ tự theo mệnh đề.

Ví dụ: hãy xem xét truy vấn sau đây chọn doanh số bán hàng của ngày hôm qua đặt hàng theo dữ liệu bán hàng và ID sản phẩm:

```
CHỌN ngày_hàng_mua_hàng ,_sản_phẩm, số lượng
   TỪ bán hàng
 WHERE sale_date = TRUNC (sysdate) - INTERVAL '1' DAY
 ĐẶT HÀNG THEO ngày bán, ngày_sản phẩm;
```

Đã có một chỉ mục vào SALE_DATE có thể được sử dụng cho mệnh đề where . Tuy nhiên, cơ sở dữ liệu phải thực hiện một thao tác sắp xếp rõ ràng để đáp ứng thứ tự theo mệnh đề:

```
-----------------------------------------------------------------------------------------
| Id | Hoạt động                                          | Tên               | Hàng | Chi phí |
-----------------------------------------------------------------------------------------
| 0 | TUYỂN CHỌN BÁO CÁO | 1                                                  | 320 | |       18 |
      | SẮP XẾP ĐƠN HÀNG THEO                              ''                  320 | |         18 |
| 2 | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | BÁN HÀNG | * 3 |                  320 |           17 |
              QUÉT NGUỒN CHỈ SỐ                           | SALES_DATE | 320 |                 3 |
-----------------------------------------------------------------------------------------
```

QUÉT NGUỒN CHỈ SỐ vẫn cung cấp kết quả theo thứ tự chỉ mục. Để lấy lợi thế của thực tế này, chúng tôi chỉ cần mở rộng định nghĩa chỉ mục để nó tương ứng với thứ tự theo mệnh đề:

```
   DROP INDEX bán_ngày;
TẠO INDEX sales_dt_pr ON sales (sale_date, product_id);
```

```
-----------------------------------------------------------------------------------
| Id | Hoạt động                                        | Tên               | Hàng | Chi phí |
-----------------------------------------------------------------------------------
| 0 | TUYỂN CHỌN BÁO CÁO | 1                           |                   | 320 | 300 |
      | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | BÁN HÀNG                                | 320 | 300 |
| * 2 | QUÉT NGUỒN CHỈ SỐ                               | SALES_DT_PR | 320 | 4 |
-----------------------------------------------------------------------------------
```


Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Thao tác sắp xếp SORT ORDER BY đã biến mất khỏi kế hoạch thực thi mặc dù truy vấn vẫn có mệnh đề order by . Cơ sở dữ liệu khai thác thứ tự chỉ mục và bỏ qua thao tác sắp xếp rõ ràng.

Quan trọng

Nếu thứ tự chỉ mục tương ứng với mệnh đề order by , thì cơ sở dữ liệu có thể bỏ qua thao tác sắp xếp rõ ràng.

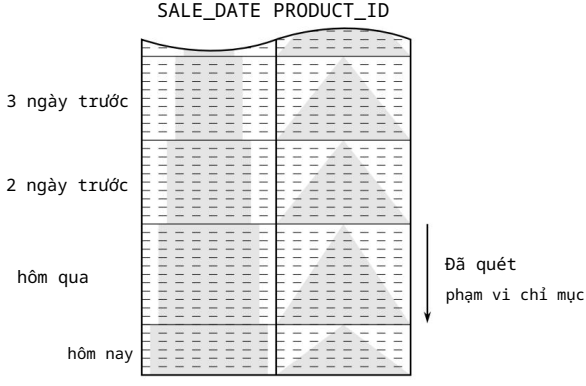
Ngay cả khi kế hoạch thực thi mới có ít thao tác hơn, giá trị chi phí đã tăng lên đáng kể vì hệ số phân nhóm của chỉ mục mới kém hơn (xem “Hệ số phân cụm được tối ưu hóa tự động” trên trang 133).
Tại thời điểm này, chỉ cần lưu ý rằng giá trị chi phí không phải lúc nào cũng là một chỉ báo tốt về nỗ lực thực hiện.

Đối với sự tối ưu hóa này, đủ để phạm vi chỉ mục được quét được sắp xếp theo thứ tự theo mệnh đề. Do đó, tối ưu hóa cũng hoạt động cho ví dụ cụ thể này khi chỉ sắp xếp theo PRODUCT_ID :

```
CHỌN ngày_hàng_mua_hàng ,_sản_phẩm, số lượng
   TỪ bán hàng
 WHERE sale_date = TRUNC (sysdate) - INTERVAL '1' DAY
 ĐẶT HÀNG BẰNG product_id;
```

Trong Hình 6.1, chúng ta có thể thấy rằng PRODUCT_ID là tiêu chí sắp xếp phù hợp duy nhất trong phạm vi chỉ mục được quét. Do đó thứ tự chỉ mục tương ứng với mệnh đề thứ tự trong phạm vi chỉ mục này để cơ sở dữ liệu có thể bỏ qua thao tác sắp xếp.

Hình 6.1. Sắp xếp Thứ tự trong Phạm vi Chỉ mục Có liên quan



Machine Translated by Google

Việc tối ưu hóa này có thể gây ra hành vi không mong muốn khi mở rộng phạm vi chỉ mục được quét:

```
CHỌN ngày_hàng_mua_hàng ,_sản_phẩm, số lượng
   TỪ bán hàng
 WHERE sale_date> = TRUNC (sysdate) - INTERVAL '1' DAY
 ĐẶT HÀNG BẰNG product_id;
```

Truy vấn này không truy xuất doanh số bán hàng của ngày hôm qua mà là tất cả doanh số bán hàng từ ngày hôm qua. Điều đó có nghĩa là nó bao gồm vài ngày và quét một phạm vi chỉ mục không được phân loại riêng bởi PRODUCT_ID. Nếu chúng ta xem lại Hình 6.1 và mở rộng phạm vi chỉ mục đã quét xuống dưới cùng, chúng ta có thể thấy rằng ở đó lại là các giá trị PRODUCT_ID nhỏ hơn . Do đó, cơ sở dữ liệu phải sử dụng một hoạt động sắp xếp rõ ràng để đáp ứng thứ tự theo mệnh đề.

```
| Id | Hoạt động                                  | Tên                | Hàng | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO | 1                                              | 320 | 301 |
      | SẮP XẾP ĐƠN HÀNG THEO                      |                    | 320 | 301 |
| 2 | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | BÁN HÀNG                      | 320 | 300 |
| * 3 | QUÉT NGUỒN CHỈ SỐ                          | SALES_DT_PR | 320 | 4 |
```

Nếu cơ sở dữ liệu sử dụng thao tác sắp xếp mặc dù bạn đã mong đợi thực thi, nó có thể có hai lý do: (1) kế hoạch thực hiện với hoạt động sắp xếp có giá trị chi phí tốt hơn; (2) thứ tự chỉ mục trong bản quét phạm vi chỉ mục không tương ứng với mệnh đề thứ tự .

Một cách đơn giản để phân biệt hai trường hợp là sử dụng định nghĩa chỉ mục đầy đủ trong mệnh đề theo thứ tự — điều đó có nghĩa là điều chỉnh truy vấn thành chỉ mục theo thứ tự để loại bỏ nguyên nhân thứ hai. Nếu cơ sở dữ liệu vẫn sử dụng một sắp xếp rõ ràng hoạt động, trình tối ưu hóa thích kế hoạch này do giá trị chi phí của nó; nếu không thì cơ sở dữ liệu không thể sử dụng chỉ mục cho mệnh đề thứ tự ban đầu .

Mẹo

Sử dụng định nghĩa chỉ mục đầy đủ theo thứ tự từng mệnh đề để tìm lý do cho một hoạt động sắp xếp rõ ràng.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Trong cả hai trường hợp, bạn có thể tự hỏi liệu và làm thế nào bạn có thể đạt được một đơn đặt hàng liên kết bằng cách thực hiện. Đối với điều này, bạn có thể thực hiện truy vấn với định nghĩa chỉ mục đầy đủ theo thứ tự từng mệnh đề và kiểm tra kết quả. Bạn sẽ thường nhận ra rằng bạn có nhận thức sai về chỉ mục và thứ tự chỉ mục thực sự không như yêu cầu của mệnh đề thứ tự ban đầu nên cơ sở dữ liệu không thể sử dụng chỉ mục để tránh thao tác sắp xếp.

Nếu trình tối ưu hóa thích một hoạt động sắp xếp rõ ràng cho giá trị chi phí của nó, điều đó thường là do trình tối ưu hóa có kế hoạch thực thi tốt nhất để thực hiện đầy đủ truy vấn. Nói cách khác, trình tối ưu hóa chọn kế hoạch thực thi nhanh nhất để có được bản ghi cuối cùng. Nếu cơ sở dữ liệu phát hiện ra rằng ứng dụng chỉ tìm nạp một vài hàng đầu tiên, thì đến lượt nó, nó có thể thích một thứ tự được lập chỉ mục hơn . Chương 7, "Kết quả từng phần", giải thích các phương pháp tối ưu hóa tương ứng.

Hệ số phân cụm được tối ưu hóa tự động

Cơ sở dữ liệu Oracle giữ yếu tố phân cụm ở mức tối thiểu bằng cách xem xét ROWID cho thứ tự chỉ mục. Bất cứ khi nào hai mục nhập chỉ mục có cùng giá trị khóa, ROWID sẽ quyết định theo thứ tự cuối cùng của chúng.
Do đó, chỉ mục cũng được sắp xếp theo thứ tự bảng và do đó có hệ số phân cụm nhỏ nhất có thể vì ROWID đại diện cho địa chỉ vật lý của hàng trong bảng.

Bằng cách thêm một cột khác vào chỉ mục, bạn chèn một tiêu chí sắp xếp mới trước ROWID . Cơ sở dữ liệu có ít quyền tự do hơn trong việc sắp xếp các mục nhập chỉ mục theo thứ tự bảng nên yếu tố phân cụm chỉ mục chỉ có thể trở nên tồi tệ hơn.

Bất kể, vẫn có thể thứ tự chỉ mục tương ứng với thứ tự bảng. Doanh số bán hàng của một ngày có thể vẫn được nhóm lại với nhau trong bảng cũng như trong chỉ mục — mặc dù trình tự của chúng không còn giống nhau nữa. Cơ sở dữ liệu phải đọc các khối bảng nhiều lần khi sử dụng chỉ mục SALE_DT_PR — nhưng đây chỉ là các khối bảng giống như trước đây. Do bộ nhớ đệm của dữ liệu được truy cập thường xuyên, tác động đến hiệu suất có thể thấp hơn đáng kể so với các giá trị chi phí được chỉ ra.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Lập chỉ mục ASC, DESC và NULLS FIRST / LAST

Cơ sở dữ liệu có thể đọc các chỉ mục theo cả hai hướng. Điều đó có nghĩa là một pipelined cũng có thể theo thứ tự nếu phạm vi chỉ mục được quét hoàn toàn ngược lại thứ tự theo quy định của thứ tự theo mệnh đề. Mặc dù các công cụ sửa đổi ASC và DESC theo thứ tự theo mệnh đề có thể ngăn chặn việc thực thi pipelined, hầu hết các cơ sở dữ liệu đưa ra một cách đơn giản để thay đổi thứ tự chỉ mục để một chỉ mục có thể sử dụng được cho một đơn đặt hàng pipelined bởi.

Ví dụ sau sử dụng một chỉ mục theo thứ tự ngược lại. Nó mang lại doanh số kể từ ngày hôm qua được sắp xếp theo ngày giảm dần và PRODUCT_ID giảm dần.

```
CHỌN ngày_giá_mua_mua_sản_phẩm, số lượng
   TỪ bán hàng
 WHERE sale_date> = TRUNC (sysdate) - INTERVAL '1' DAY
 ĐẶT HÀNG THEO ngày bán_sản phẩm DESC, sản_phẩm DESC;
```

Kế hoạch thực thi cho thấy rằng cơ sở dữ liệu đọc chỉ mục theo chiều giảm dần hướng đi.

```
-------------------------------------------------------------------------
| Id | Hoạt động                                  | Tên                | Hàng | Chi phí |
-------------------------------------------------------------------------
| 0 | TUYỂN CHỌN BÁO CÁO | 320 | 300 |              |
|  1   | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | BÁN HÀNG | 320 | 300 |
| * 2 | MÔ TẢ QUÉT CHỈ SỐ RANGE | SALES_DT_PR | 320 | 4 |
-------------------------------------------------------------------------
```

Trong trường hợp này, cơ sở dữ liệu sử dụng cây chỉ mục để tìm kết quả phù hợp cuối cùng lối vào. Từ đó trở đi, nó đi theo chuỗi nút lá "lên trên" như hình minh họa trong Hình 6.2. Rốt cuộc, đây là lý do tại sao cơ sở dữ liệu sử dụng danh sách được liên kết kép để xây dựng chuỗi nút lá.

Tất nhiên, điều quan trọng là phạm vi chỉ mục được quét hoàn toàn ngược lại thứ tự khi cần thiết cho thứ tự theo mệnh đề.

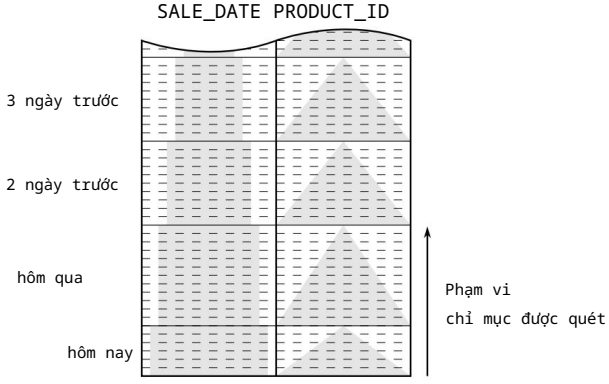
Quan trọng

Cơ sở dữ liệu có thể đọc các chỉ mục theo cả hai hướng.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Hình 6.2. Quét chỉ mục ngược

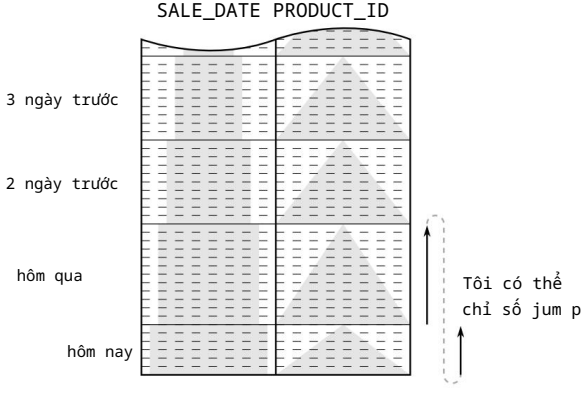


Ví dụ sau không đáp ứng điều kiện tiên quyết này vì nó kết hợp các công cụ sửa đổi ASC và DESC theo thứ tự của mệnh đề:

```
CHỌN ngày_giá_mua_mua_sản_phẩm, số lượng
   TỪ bán hàng
 WHERE sale_date> = TRUNC (sysdate) - INTERVAL '1' DAY
 ĐẶT HÀNG THEO ASC ngày bán, product_id DESC;
```

Trước tiên, truy vấn phải cung cấp doanh số bán hàng của ngày hôm qua được đặt hàng theo PRODUCT_ID giảm dần và sau đó là doanh số bán hàng của ngày hôm nay, một lần nữa bằng cách giảm dần PRODUCT_ID. Hình 6.3 minh họa quá trình này. Để có được doanh số theo thứ tự cần thiết, cơ sở dữ liệu sẽ phải "nhảy" trong quá trình quét chỉ mục.

Hình 6.3. Thứ tự không thể Pipelined bởi



Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Tuy nhiên, chỉ số này không có mối liên hệ nào từ đợt bán hàng ngày hôm qua với giá trị nhỏ nhất PRODUCT_ID để giảm giá hôm nay với nhiều nhất. Do đó, cơ sở dữ liệu có thể không sử dụng chỉ mục này để tránh thao tác sắp xếp rõ ràng.

Đối với những trường hợp như thế này, hầu hết các cơ sở dữ liệu cung cấp một phương pháp đơn giản để điều chỉnh chỉ mục thứ tự theo thứ tự của mệnh đề. Cụ thể, điều này có nghĩa là bạn có thể sử dụng ASC và các công cụ sửa đổi DESC trong khai báo chỉ mục:

```
   DROP INDEX sales_dt_pr;

TẠO INDEX sales_dt_pr
      BẬT bán hàng ( ASC ngày bán, product_id DESC);
```

Cảnh báo

Cơ sở dữ liệu MySQL bỏ qua các công cụ sửa đổi ASC và DESC trong chỉ mục Định nghĩa.

Bây giờ thứ tự chỉ mục tương ứng với thứ tự theo mệnh đề để cơ sở dữ liệu có thể bỏ qua thao tác sắp xếp:

```
--------------------------------------------------------------------------------
| Id | Hoạt động                                  | Tên                | Hàng | Chi phí |
--------------------------------------------------------------------------------
| 0 | TUYỂN CHỌN BÁO CÁO | 1                      |                    | 320 | 301 |
      | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | BÁN HÀNG                      | 320 | 301 |
| * 2 | QUÉT NGUỒN CHỈ SỐ                          | SALES_DT_PR | 320 | 4 |
--------------------------------------------------------------------------------
```

Hình 6.4 cho thấy thứ tự chỉ mục mới. Sự thay đổi trong hướng sắp xếp cho cột thứ hai theo cách hoán đổi hướng của các mũi tên từ hình trước. Điều đó làm cho mũi tên đầu tiên kết thúc ở chỗ mũi tên thứ hai bắt đầu để chỉ mục đó có các hàng theo thứ tự mong muốn.

Quan trọng

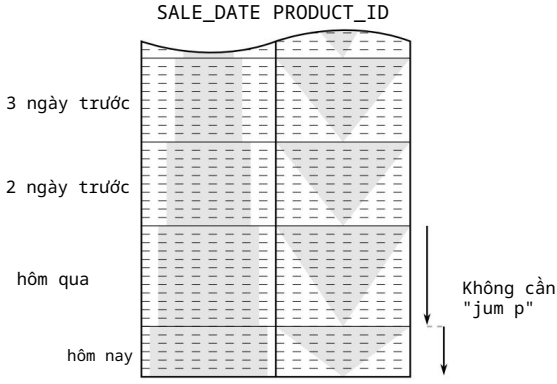
Khi sử dụng công cụ sửa đổi ASC và DESC hỗn hợp theo thứ tự từng mệnh đề, bạn phải xác định chỉ mục tương tự như vậy để sử dụng nó cho một đặt bởi.

Điều này không ảnh hưởng đến khả năng sử dụng của chỉ mục đối với mệnh đề where .

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Hình 6.4. Chỉ số đơn đặt hàng hỗn hợp



Chỉ mục ASC / DESC chỉ cần thiết để sắp xếp các cột riêng lẻ theo hướng ngược lại. Không cần thiết phải đảo ngược thứ tự của tất cả các cột vì cơ sở dữ liệu vẫn có thể đọc chỉ mục theo thứ tự giảm dần nếu cần— chỉ mục phụ trên các bảng được tổ chức theo chỉ mục là ngoại lệ duy nhất.

Các chỉ mục phụ thêm ngầm khóa phân cụm vào chỉ mục mà không cung cấp bất kỳ khả năng nào để chỉ định thứ tự sắp xếp. Nếu bạn cần sắp xếp khóa phân cụm theo thứ tự giảm dần, bạn không có lựa chọn nào khác ngoài việc sắp xếp tất cả các cột khác theo thứ tự giảm dần. Cơ sở dữ liệu sau đó có thể đọc chỉ mục theo hướng ngược lại để có được thứ tự mong muốn.

Bên cạnh ASC và DESC, tiêu chuẩn SQL xác định hai công cụ sửa đổi hầu như không được biết đến cho mệnh đề thứ tự : NULLS FIRST và NULLS LAST. Kiểm soát rõ ràng đối với sắp xếp NULL đã được "giới thiệu gần đây" như một phần mở rộng tùy chọn với SQL: 2003. Kết quả là, hỗ trợ cơ sở dữ liệu rất thưa thớt. Điều này đặc biệt đáng lo ngại vì tiêu chuẩn không xác định chính xác thứ tự sắp xếp của NULL. Nó chỉ nói rằng tất cả các NULL phải xuất hiện cùng nhau sau khi sắp xếp, nhưng nó không chỉ định liệu chúng sẽ xuất hiện trước hay sau các mục nhập khác.

Nói một cách chính xác, bạn sẽ thực sự cần chỉ định sắp xếp NULL cho tất cả các cột có thể rỗng theo thứ tự theo mệnh đề để có được một hành vi được xác định rõ ràng.

Tuy nhiên, thực tế là phần mở rộng tùy chọn không được SQL Server 2012 cũng như MySQL 5.6 triển khai. Cơ sở dữ liệu Oracle, ngược lại, đã hỗ trợ sắp xếp NULLS ngay cả trước khi nó được đưa vào tiêu chuẩn, nhưng nó không chấp nhận nó trong các định nghĩa chỉ mục kể từ bản phát hành 11g. Do đó, cơ sở dữ liệu Oracle không thể thực hiện một thứ tự liên tục khi sắp xếp với

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

NULLS ĐẦU TIÊN. Chỉ cơ sở dữ liệu PostgreSQL (kể từ bản phát hành 8.3) hỗ trợ công cụ sửa đổi NULLS theo cả thứ tự theo mệnh đề và định nghĩa chỉ mục.

Tổng quan sau đây tóm tắt các tính năng được cung cấp bởi các cơ sở dữ liệu khác nhau.

Hình 6.5. Cơ sở dữ liệu / Ma trận tính năng

	SQL Server	Oracle	MySQL	PostgreSQL
Đọc ngược chỉ mục	✓	✓	✓	✓
Đặt hàng theo ASC / DESC	✓	✓	✓	✓
Chỉ mục ASC / DESC	✗	✓	✓	✓
Đặt hàng bởi NULLS FIRST / LAST	✗	✓	✓	✗
Đơn hàng NULLS mặc định	Ngày thứ nhất	Cuối	Cuối	Ngày thứ nhất
Chỉ mục NULLS ĐẦU / CUỐI	✗	✗	✓	✗

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Nhóm lập chỉ mục theo

Cơ sở dữ liệu SQL sử dụng hai nhóm hoàn toàn khác nhau theo các thuật toán. Cái đầu tiên, thuật toán băm, tổng hợp các bản ghi đầu vào trong một bảng băm tạm thời. Khi tất cả các bản ghi đầu vào được xử lý, bảng băm được trả về dưới dạng kết quả. Thuật toán thứ hai, thuật toán sắp xếp / nhóm, đầu tiên sắp xếp đầu vào dữ liệu theo khóa nhóm để các hàng của mỗi nhóm tiếp nối nhau liên tiếp ngay lập tức. Sau đó, cơ sở dữ liệu chỉ cần tổng hợp họ. Nói chung, cả hai thuật toán đều cần thực hiện một trung gian trạng thái, vì vậy chúng không được thực thi theo cách thức định hướng. Tuy nhiên, loại / thuật toán nhóm có thể sử dụng một chỉ mục để tránh thao tác sắp xếp, do đó cho phép một nhóm pipelined bởi.

> Ghi chú
>
> MySQL 5.6 không sử dụng thuật toán băm. Tuy nhiên, tối ưu hóa cho thuật toán sắp xếp / nhóm hoạt động như mô tả bên dưới.

Hãy xem xét truy vấn sau đây. Nó mang lại doanh thu của ngày hôm qua được nhóm lại bởi ID SẢN PHẨM:

```
SELECT product_id, sum (eur_value)
   TỪ bán hàng
 WHERE sale_date = TRUNC (sysdate) - INTERVAL '1' DAY
 GROUP BY product_id;
```

Biết chỉ mục vào SALE_DATE và PRODUCT_ID từ phần trước, thuật toán sắp xếp / nhóm thích hợp hơn vì QUÉT NGUỒN CHỈ SỐ tự động phân phối các hàng theo thứ tự yêu cầu. Điều đó có nghĩa là cơ sở dữ liệu tránh hiện tượng hóa vì nó không cần sắp xếp rõ ràng hoạt động — nhóm của được thực thi theo cách thức tổng hợp.

```
| Id | Hoạt động                                  | Tên                 | Hàng | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO | 1                                                          | 17 | 192 |
      | SẮP XẾP NHÓM THEO NOSORT                                                | 17 | 192 |
| 2 | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | BÁN HÀNG                                | 321 | 192 |
| * 3 | QUÉT NGUỒN CHỈ SỐ                          | SALES_DT_PR | 321 | 3 |
```

Machine Translated by Google

Kế hoạch thực thi của cơ sở dữ liệu Oracle đánh dấu một NHÓM SẮP XẾP BẰNG hoạt động với phụ lục NOSORT . Kế hoạch thực hiện của khác cơ sở dữ liệu hoàn toàn không đề cập đến bất kỳ thao tác sắp xếp nào.

Nhóm pipelined bởi có cùng điều kiện tiên quyết như nhóm pipelined theo thứ tự, ngoại trừ không có công cụ sửa đổi ASC và DESC . Đó nghĩa là xác định một chỉ mục với các công cụ sửa đổi ASC / DESC sẽ không ảnh hưởng đến pipelined nhóm theo cách thực hiện. Điều này cũng đúng với NULLS FIRST / LAST. Tuy nhiên có những cơ sở dữ liệu không thể sử dụng chỉ mục ASC / DESC đúng cách cho một nhóm pipelined bởi.

Cảnh báo

Đối với PostgreSQL, bạn phải thêm mệnh đề theo thứ tự để tạo chỉ mục với NULLS LAST sắp xếp có thể sử dụng được cho một nhóm pipelined theo. Cơ sở dữ liệu Oracle không thể đọc ngược một chỉ mục để thực hiện một nhóm pipelined bởi nhóm đó được theo sau bởi một lệnh bởi.

Nếu chúng tôi mở rộng truy vấn để xem xét tất cả các lần bán hàng kể từ ngày hôm qua, như chúng tôi đã làm trong ví dụ cho thứ tự pipelined bởi, nó ngăn chặn nhóm pipelined bởi for cùng một lý do như trước đây: QUÉT NGUỒN CHỈ SỐ không phân phối các hàng sắp xếp theo khóa nhóm (so sánh Hình 6.1 trang 131).

```
SELECT product_id, sum (eur_value)
   TỪ bán hàng
 WHERE sale_date> = TRUNC (sysdate) - INTERVAL '1' DAY
 GROUP BY product_id;

--------------------------------------------------------------------------------
| Id | Hoạt động                                  | Tên                | Hàng | Chi phí |
--------------------------------------------------------------------------------
| 0 | TUYỂN CHỌN BÁO CÁO |                                                | 24 | 356 |
   1   | NHÓM HASH BỞI                                                    | 24 | 356 |
| 2 | TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | BÁN HÀNG                          | 596 | 355 |
| * 3 | QUÉT NGUỒN CHỈ SỐ                          | SALES_DT_PR | 596 | 4 |
--------------------------------------------------------------------------------
```

Thay vào đó, cơ sở dữ liệu Oracle sử dụng thuật toán băm. Lợi thế của thuật toán băm là nó chỉ cần đệm kết quả tổng hợp, trong khi thuật toán sắp xếp / nhóm hiện thực hóa tập hợp đầu vào hoàn chỉnh. Trong nói cách khác: thuật toán băm cần ít bộ nhớ hơn.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Cũng như đối với lệnh pipelined by, thực hiện nhanh không phải là khía cạnh quan trọng nhất của nhóm pipelined theo thực thi. Điều quan trọng hơn là cơ sở dữ liệu thực thi nó theo cách thức tổng hợp và đưa ra kết quả đầu tiên trước khi đọc toàn bộ dữ liệu đầu vào. Đây là điều kiện tiên quyết cho các phương pháp tối ưu hóa nâng cao được giải thích trong chương tiếp theo.

Hãy nghĩ về nó

Bạn có thể nghĩ về bất kỳ hoạt động cơ sở dữ liệu nào khác - ngoài sắp xếp và nhóm - có thể sử dụng một chỉ mục để tránh sắp xếp không?

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Chương 7

Kết quả một phần

Đôi khi bạn không cần kết quả đầy đủ của một truy vấn SQL mà chỉ cần một vài hàng đầu tiên – ví dụ: chỉ hiển thị mười thông báo gần đây nhất. Trong trường hợp này, người ta cũng thường cho phép người dùng duyệt qua các thư cũ hơn – sử dụng điều hướng phân trang truyền thống hoặc biến thể "cuộn vô hạn" hiện đại hơn. Tuy nhiên, các truy vấn SQL liên quan được sử dụng cho chức năng này có thể gây ra các vấn đề nghiêm trọng về hiệu suất nếu tất cả các thư phải được sắp xếp để tìm những thư mới nhất. Do đó, một lệnh pipelined là một phương tiện tối ưu hóa rất mạnh mẽ cho các truy vấn như vậy.

Chương này trình bày cách sử dụng một thứ tự pipelined để truy xuất các kết quả từng phần một cách hiệu quả. Mặc dù cú pháp của các truy vấn này khác nhau giữa các cơ sở dữ liệu, nhưng chúng vẫn thực hiện các truy vấn theo cách rất giống nhau.
Một lần nữa, điều này minh họa rằng tất cả họ đều để quần trên một chân.

Truy vấn hàng đầu-N

Các truy vấn Top-N là các truy vấn giới hạn kết quả ở một số hàng cụ thể.
Đây thường là các truy vấn cho các mục nhập gần đây nhất hoặc "tốt nhất" của một tập hợp kết quả. Để thực thi hiệu quả, việc xếp hạng phải được thực hiện với một thứ tự được định sẵn bởi.

Cách đơn giản nhất để chỉ tìm nạp các hàng đầu tiên của truy vấn là tìm nạp các hàng bắt buộc và sau đó đóng câu lệnh. Thật không may, trình tối ưu hóa không thể thấy trước điều đó khi chuẩn bị kế hoạch thực thi. Để chọn kế hoạch thực thi tốt nhất, trình tối ưu hóa phải biết liệu cuối cùng ứng dụng có tìm nạp tất cả các hàng hay không. Trong trường hợp đó, việc quét toàn bộ bảng với thao tác sắp xếp rõ ràng có thể hoạt động tốt nhất, mặc dù một thứ tự được tổng hợp bởi có thể tốt hơn khi chỉ tìm nạp mười hàng – ngay cả khi cơ sở dữ liệu phải tìm nạp từng hàng riêng lẻ. Điều đó có nghĩa là trình tối ưu hóa phải biết liệu bạn có định hủy bỏ câu lệnh hay không trước khi tìm nạp tất cả các hàng để nó có thể chọn kế hoạch thực thi tốt nhất.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Mẹo

Thông báo cho cơ sở dữ liệu bất cứ khi nào bạn không cần tất cả các hàng.

Tiêu chuẩn SQL đã loại trừ yêu cầu này trong một thời gian dài. Phần mở rộng tương ứng (tìm nạp trước) vừa được giới thiệu với SQL: 2008 và hiện chỉ có sẵn trong IBM DB2, PostgreSQL và SQL Server 2012.
Một mặt, điều này là do tính năng này là một phần mở rộng không phải cốt lõi và mặt khác là do mỗi cơ sở dữ liệu đã cung cấp giải pháp độc quyền của riêng mình trong nhiều năm.

Các ví dụ sau đây cho thấy việc sử dụng các tiện ích mở rộng nổi tiếng này bằng cách truy vấn mười lần bán hàng gần đây nhất. Cơ sở luôn giống nhau: tìm nạp tất cả doanh số bán hàng, bắt đầu bằng doanh số bán hàng gần đây nhất. Cú pháp top-N tương ứng chỉ hủy bỏ việc thực thi sau khi tìm nạp mười hàng.

MySQL

MySQL và PostgreSQL sử dụng mệnh đề giới hạn để hạn chế số hàng được tìm nạp.

```
LỰA CHỌN *
   TỪ bán hàng
 ĐẶT HÀNG THEO sale_date DESC
 GIỚI HẠN 10;
```

Cơ sở dữ liệu

Oracle Cơ sở dữ liệu Oracle cung cấp cột giả ROWNUM đánh số các hàng trong tập kết quả một cách tự động. Để sử dụng cột này trong một bộ lọc, chúng tôi phải kết thúc truy vấn:

```
LỰA CHỌN *
   TỪ (
          LỰA CHỌN *
             TỪ bán hàng
           ĐẶT HÀNG THEO sale_date DESC)

 WHERE rownum <= 10;
```


Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

PostgreSQL

PostgreSQL hỗ trợ phần mở rộng tìm nạp đầu tiên kể từ phiên bản 8.4. Các mệnh đề giới hạn đã sử dụng trước đó vẫn hoạt động như được hiển thị trong MySQL thí dụ.

```
LỰA CHỌN *
   TỪ bán hàng
 ĐẶT HÀNG THEO sale_date DESC
 CHỈ TÌM 10 CON ĐƯỜNG ĐẦU TIÊN;
```

Máy chủ SQL

SQL Server cung cấp mệnh đề trên cùng để hạn chế số hàng được tìm nạp.

```
CHỌN HÀNG ĐẦU 10 *
   TỪ bán hàng
 ĐẶT HÀNG BẰNG ngày bán hàng DESC;
```

Bắt đầu với bản phát hành 2012, SQL Server hỗ trợ tìm nạp trước cả phần mở rộng.

Tất cả các truy vấn SQL được hiển thị ở trên đều đặc biệt vì cơ sở dữ liệu nhận ra chúng là các truy vấn hàng đầu.

Quan trọng

Cơ sở dữ liệu chỉ có thể tối ưu hóa một truy vấn cho một phần kết quả nếu nó biết điều này ngay từ đầu.

Nếu trình tối ưu hóa nhận thức được thực tế là chúng ta chỉ cần mười hàng, nó sẽ thích sử dụng một thứ tự pipelined bằng cách :

```
| Hoạt động                                  | Tên                | Hàng | Chi phí |

| TUYỂN CHỌN BÁO CÁO | 10 |                                                          9 |
| ĐẾM DỪNG LẠI | |                                                                |
| XEM | 10 | |                                                          9 |
       TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | BÁN HÀNG      | 1004K | |      9 |
         MÔ TẢ QUÉT CHỈ SỐ ĐẦY ĐỦ | SALES_DT_PR | 10 |                          3 |
```

Kế hoạch thực thi Oracle chỉ ra sự kết thúc theo kế hoạch với COUNT STOPKEY hoạt động. Điều đó có nghĩa là cơ sở dữ liệu đã nhận ra top-N cú pháp.

Machine Translated by Google

> **Mẹo**
>
> Phụ lục A, "Kế hoạch thực hiện", tóm tắt các hoạt động cho MySQL, Oracle, PostgreSQL và SQL Server.

Sử dụng đúng cú pháp chỉ là một nửa câu chuyện vì hiệu quả việc chấm dứt việc thực thi yêu cầu các hoạt động cơ bản phải được thực hiện theo cách pipelined. Điều đó có nghĩa là thứ tự theo mệnh đề phải được bao phủ bởi một chỉ mục — chỉ mục SALE_DT_PR vào SALE_DATE và PRODUCT_ID trong ví dụ này. Bằng cách sử dụng chỉ mục này, cơ sở dữ liệu có thể tránh được sự sắp xếp rõ ràng hoạt động và do đó có thể ngay lập tức gửi các hàng tới ứng dụng dưới dạng đã đọc từ chỉ mục. Quá trình thực thi bị hủy bỏ sau khi tìm nạp mười hàng, do đó cơ sở dữ liệu không đọc nhiều hàng hơn số hàng đã chọn.

> **Quan trọng**
>
> Một truy vấn top-N tổng hợp không cần đọc và sắp xếp toàn bộ kết quả bộ.

Nếu không có chỉ mục phù hợp vào SALE_DATE cho một đơn đặt hàng được kết nối bởi, cơ sở dữ liệu phải đọc và sắp xếp toàn bộ bảng. Hàng đầu tiên chỉ được giao sau khi đọc hàng cuối cùng từ bảng.

```
-----------------------------------------------------
| Hoạt động                  | Tên | Hàng | Chi phí |
-----------------------------------------------------
| TUYỂN CHỌN BÁO CÁO | | 10 | 59558 |
| ĐẾM DỪNG LẠI | | | |
| XEM |                                | 1004K | 59558 |
|     SẮP XẾP ĐẶT HÀNG THEO STOPKEY |          | 1004K | 59558 |
|      BẢNG TRUY CẬP ĐẦY ĐỦ | BÁN HÀNG | 1004K | 9246 |
-----------------------------------------------------
```

Kế hoạch thực hiện này không có thứ tự cốt lõi và gần như chậm như hủy bỏ việc thực thi từ phía máy khách. Vẫn sử dụng cú pháp top-N tốt hơn vì cơ sở dữ liệu không cần hiện thực hóa kết quả đầy đủ nhưng chỉ mười hàng gần đây nhất. Điều này đòi hỏi ít bộ nhớ hơn đáng kể. Các Kế hoạch thực thi Oracle chỉ ra sự tối ưu hóa này với công cụ sửa đổi STOPKEY trên thao tác SORT ORDER BY .

Những lợi thế của truy vấn top-N được tổng hợp không chỉ bao gồm tăng hiệu suất nhưng cũng cải thiện khả năng mở rộng. Không sử dụng pipelined thực thi, thời gian phản hồi của truy vấn top-N này tăng theo bảng kích thước. Tuy nhiên, thời gian phản hồi bằng cách sử dụng thực thi pipelined chỉ tăng lên

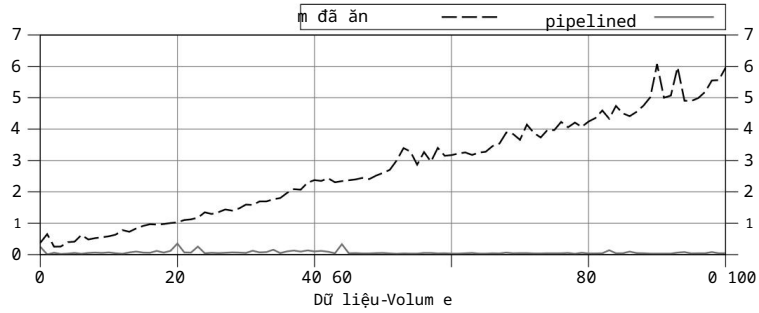
Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

với số lượng hàng đã chọn. Nói cách khác, thời gian phản hồi của một truy vấn top-N được xâu chuỗi luôn bằng nhau; điều này gần như không phụ thuộc vào kích thước bảng. Chỉ khi độ sâu cây B phát triển thì truy vấn mới trở nên chậm hơn một chút.

Hình 7.1 cho thấy khả năng mở rộng cho cả hai biến thể trên một khối lượng dữ liệu ngày càng tăng. Tăng thời gian phản hồi tuyến tính đối với một thực thi không có thứ tự liên kết bởi có thể nhìn thấy rõ ràng. Thời gian phản hồi cho việc thực thi pipelined vẫn không đổi.

Hình 7.1. Khả năng mở rộng của các truy vấn Top-N



Mặc dù thời gian phản hồi của truy vấn top-N không phụ thuộc vào kích thước bảng, nhưng nó vẫn tăng theo số lượng hàng đã chọn. Do đó, thời gian phản hồi sẽ tăng gấp đôi khi chọn nhiều hàng gấp đôi.
Điều này đặc biệt quan trọng đối với các truy vấn "phân trang" tải thêm kết quả vì những truy vấn này thường bắt đầu lại ở mục nhập đầu tiên; họ sẽ đọc các hàng đã được hiển thị trên trang trước và loại bỏ chúng trước khi cuối cùng đến kết quả cho trang thứ hai. Tuy nhiên, có một giải pháp cho vấn đề này cũng như chúng ta sẽ xem trong phần tiếp theo.

Phân trang thông qua kết quả

Sau khi triển khai một truy vấn top-N xoay quanh để truy xuất trang đầu tiên một cách hiệu quả, bạn cũng sẽ cần một truy vấn khác để tìm nạp các trang tiếp theo.
Thách thức kết quả là nó phải bỏ qua các hàng từ các trang trước. Có hai phương pháp khác nhau để đáp ứng thách thức này: trước hết là phương pháp bù đắp, đánh số các hàng ngay từ đầu và sử dụng bộ lọc trên số hàng này để loại bỏ các hàng trước trang được yêu cầu. Phương thức thứ hai, mà tôi gọi là phương thức seek, tìm kiếm mục nhập cuối cùng của trang trước và chỉ tìm nạp các hàng sau.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Các ví dụ sau đây cho thấy phương pháp bù đắp được sử dụng rộng rãi hơn. Ưu điểm chính của nó là nó rất dễ xử lý — đặc biệt là với các cơ sở dữ liệu có từ khóa dành riêng cho nó (offset). Từ khóa này thậm chí còn được đưa vào tiêu chuẩn SQL như một phần của phần mở rộng tìm nạp đầu tiên .

MySQL

MySQL và PostgreSQL đưa ra điều khoản bù trừ để loại bỏ số hàng được chỉ định từ đầu truy vấn top-N. Điều khoản giới hạn được áp dụng sau đó.

```
LỰA CHỌN *
   TỪ bán hàng
  ĐẶT HÀNG THEO sale_date DESC
  GIỚI HẠN 10 OFFSET 10;
```

Cơ sở dữ liệu

Oracle Cơ sở dữ liệu Oracle cung cấp cột giả ROWNUM đánh số các hàng trong tập kết quả một cách tự động. Tuy nhiên, không thể áp dụng bộ lọc lớn hơn hoặc bằng (> =) trên cột giả này. Để thực hiện công việc này, trước tiên bạn cần phải "cụ thể hóa" số hàng bằng cách đổi tên cột bằng bí danh.

```
CHỌN * TỪ
   (CHỌN tmp. *, Rownum rn
                 TỪ (CHỌN *
                              TỪ bán hàng
                            ĐẶT HÀNG THEO sale_date DESC) tmp

               WHERE rownum <= 20
          )
  TRONG ĐÓ rn> 10;
```

Lưu ý việc sử dụng bí danh RN cho giới hạn dưới và chính cột giả ROWNUM cho giới hạn trên.

PostgreSQL

Phần mở rộng tìm nạp đầu tiên cũng xác định một mệnh đề hàng ... bù đắp .

PostgreSQL, tuy nhiên, chỉ chấp nhận bù đắp mà không có từ khóa hàng .
Cú pháp giới hạn / bù đắp được sử dụng trước đó vẫn hoạt động như được hiển thị trong ví dụ MySQL.

```
LỰA CHỌN *
   TỪ bán hàng
  ĐẶT HÀNG THEO sale_date DESC
OFFSET 10
  CHỈ TÌM 10 CON ĐƯỜNG TIẾP THEO;
```


Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

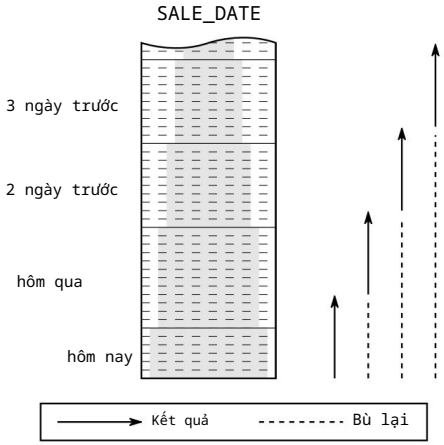
SQL Server

SQL Server không có phần mở rộng "bù đắp" cho mệnh đề hàng đầu độc quyền của nó nhưng đã giới thiệu phần mở rộng tìm nạp đầu tiên với SQL Server 2012 ("Denali"). Điều khoản bù đắp là bắt buộc mặc dù tiêu chuẩn xác định nó là một phụ lục không bắt buộc.

```
LỰA CHỌN *
   TỪ bán hàng
 ĐẶT HÀNG THEO sale_date DESC
OFFSET 10 ROWS
 CHỈ TÌM 10 CON ĐƯỜNG TIẾP THEO;
```

Bên cạnh sự đơn giản, một ưu điểm khác của phương pháp này là bạn chỉ cần offset hàng để tìm nạp một trang tùy ý. Tuy nhiên, cơ sở dữ liệu phải đếm tất cả các hàng từ đầu cho đến khi nó đến trang được yêu cầu. Hình 7.2 cho thấy phạm vi chỉ mục được quét sẽ lớn hơn khi tìm nạp nhiều trang hơn.

Hình 7.2. Truy cập bằng phương pháp bù đắp



Điều này có hai nhược điểm: (1) các trang bị trôi khi chèn doanh số mới vì việc đánh số luôn được thực hiện từ đầu; (2) thời gian phản hồi tăng lên khi duyệt xa hơn.

Phương thức seek tránh được cả hai vấn đề vì nó sử dụng các giá trị của trang trước làm dấu phân cách. Điều đó có nghĩa là nó tìm kiếm các giá trị phải đứng sau mục nhập cuối cùng từ trang trước. Điều này có thể được diễn đạt bằng mệnh đề where đơn giản . Nói cách khác: phương thức seek đơn giản là không chọn các giá trị đã được hiển thị.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Ví dụ tiếp theo cho thấy phương pháp tìm kiếm. Để chứng minh, chúng tôi sẽ bắt đầu với giả định rằng chỉ có một lần bán hàng mỗi ngày. Điều này làm cho SALE_DATE trở thành một khóa duy nhất. Để chọn doanh số bán hàng phải đến sau một ngày cụ thể, bạn phải sử dụng điều kiện nhỏ hơn (<) vì thứ tự sắp xếp giảm dần. Đối với một thứ tự tăng dần, bạn sẽ phải sử dụng điều kiện lớn hơn (>) . Mệnh đề tìm nạp đầu tiên chỉ được sử dụng để giới hạn kết quả ở mười hàng.

```
CHỌN * TỪ
   bán hàng WHERE
  sale_date <?
  ĐẶT HÀNG THEO_date_date MÔ TẢ
  CHỈ TÌM 10 CHI PHÍ ĐẦU TIÊN;
```

Thay vì một số hàng, bạn sử dụng giá trị cuối cùng của trang trước đó để chỉ định giới hạn dưới. Điều này có một lợi ích to lớn về mặt hiệu suất vì cơ sở dữ liệu có thể sử dụng SALE_DATE <? điều kiện để truy cập chỉ mục.

Điều đó có nghĩa là cơ sở dữ liệu thực sự có thể bỏ qua các hàng từ các trang trước. Trên hết, bạn cũng sẽ nhận được kết quả ổn định nếu các hàng mới được chèn vào.

Tuy nhiên, phương pháp này không hoạt động nếu có nhiều hơn một lần bán hàng mỗi ngày — như trong Hình 7.2 — bởi vì việc sử dụng ngày cuối cùng từ trang đầu tiên ("hôm qua") sẽ bỏ qua tất cả các kết quả từ ngày hôm qua — không chỉ những kết quả đã được hiển thị trên trang đầu tiên. Vấn đề là thứ tự theo mệnh đề không thiết lập một chuỗi hàng xác định. Tuy nhiên, đó là điều kiện tiên quyết để sử dụng điều kiện phạm vi đơn giản cho ngắt trang.

Không có mệnh đề thứ tự xác định , cơ sở dữ liệu theo định nghĩa không cung cấp một chuỗi hàng xác định. Lý do duy nhất bạn thường nhận được một chuỗi hàng nhất quán là cơ sở dữ liệu thường thực hiện truy vấn theo cùng một cách. Tuy nhiên, trên thực tế, cơ sở dữ liệu có thể xáo trộn các hàng có cùng SALE_DATE và vẫn đáp ứng thứ tự theo mệnh đề. Trong các bản phát hành gần đây, nó thực sự có thể xảy ra rằng bạn nhận được kết quả theo một thứ tự khác nhau mỗi khi bạn chạy truy vấn, không phải vì cơ sở dữ liệu cố ý xáo trộn kết quả mà vì cơ sở dữ liệu có thể sử dụng thực thi truy vấn song song. Điều đó có nghĩa là cùng một kế hoạch thực thi có thể dẫn đến một chuỗi hàng khác nhau vì các luồng đang thực thi kết thúc theo một thứ tự không xác định.

Quan trọng

Phân trang yêu cầu một thứ tự sắp xếp xác định.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Ngay cả khi các đặc tả chức năng chỉ yêu cầu sắp xếp "theo ngày, muộn nhất trước", chúng tôi với tư cách là nhà phát triển phải đảm bảo thứ tự theo mệnh đề mang lại một chuỗi hàng xác định. Với mục đích này, chúng ta có thể cần mở rộng thứ tự theo mệnh đề với các cột tùy ý chỉ để đảm bảo rằng chúng ta nhận được một chuỗi hàng xác định. Nếu chỉ mục được sử dụng cho thứ tự pipelined bởi có các cột bổ sung, thì đó là một khởi đầu tốt để thêm chúng vào mệnh đề order by để chúng ta có thể tiếp tục sử dụng chỉ mục này cho thứ tự pipelined bởi. Nếu điều này vẫn không mang lại thứ tự sắp xếp xác định, chỉ cần thêm bất kỳ (các) cột duy nhất nào và mở rộng chỉ mục cho phù hợp.

Trong ví dụ sau, chúng tôi mở rộng thứ tự theo mệnh đề và chỉ mục với khóa chính SALE_ID để có được một chuỗi hàng xác định. Hơn nữa, chúng ta phải áp dụng logic “đến sau” cho cả hai cột cùng nhau để có được kết quả mong muốn:

```
TẠO INDEX sl_dtid ON sales (sale_date, sale_id);

CHỌN * TỪ
   bán hàng
 WHERE (sale_date, sale_id) <(?,?)
 ĐẶT HÀNG BẰNG ngày sale_date DESC, sale_id DESC CHỈ
 TÌM 10 CHI PHÍ ĐẦU TIÊN;
```

Mệnh đề where sử dụng cú pháp "giá trị hàng" ít được biết đến (xem hộp có tên "Giá trị hàng trong SQL"). Nó kết hợp nhiều giá trị thành một đơn vị logic có thể áp dụng cho các toán tử so sánh thông thường. Như với các giá trị vô hướng, điều kiện nhỏ hơn tương ứng với "đến sau" khi sắp xếp theo thứ tự giảm dần. Điều đó có nghĩa là truy vấn chỉ xem xét doanh số bán hàng sau cặp SALE_DATE, SALE_ID đã cho .

Mặc dù cú pháp giá trị hàng là một phần của tiêu chuẩn SQL, chỉ một số cơ sở dữ liệu hỗ trợ nó. SQL Server 2012 (“Denali”) hoàn toàn không hỗ trợ giá trị hàng. Cơ sở dữ liệu Oracle về nguyên tắc hỗ trợ các giá trị hàng, nhưng không thể áp dụng các toán tử phạm vi trên chúng (ORA-01796). MySQL đánh giá các biểu thức giá trị hàng một cách chính xác nhưng không thể sử dụng chúng làm vị từ truy cập trong quá trình truy cập chỉ mục. Tuy nhiên, PostgreSQL hỗ trợ cú pháp giá trị hàng và sử dụng chúng để truy cập chỉ mục nếu có sẵn chỉ mục tương ứng.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Tuy nhiên, có thể sử dụng một biến thể gần đúng của tìm kiếm phương pháp với cơ sở dữ liệu không hỗ trợ đúng giá trị hàng — thậm chí mặc dù giá trị gần đúng không thanh lịch và hiệu quả như các giá trị hàng trong PostgreSQL. Đối với giá trị gần đúng này, chúng ta phải sử dụng so sánh "thông thường" để thể hiện logic được yêu cầu như được hiển thị trong ví dụ Oracle này:

```
LỰA CHỌN *
   TỪ (CHỌN *
                Từ bán hàng
              WHERE sale_date <=?
                 VÀ KHÔNG (sale_date =? AND sale_id> =?)
              ĐẶT HÀNG THEO sale_date DESC, sale_id DESC
          )
 WHERE rownum <= 10;
```

Mệnh đề where bao gồm hai phần. Phần đầu tiên xem xét Chỉ SALE_DATE và sử dụng điều kiện nhỏ hơn hoặc bằng (<=) — nó sẽ chọn nhiều hàng hơn nếu cần. Phần này của mệnh đề where đủ đơn giản để tất cả các cơ sở dữ liệu có thể sử dụng nó để truy cập chỉ mục. Phần thứ hai của nơi mệnh đề loại bỏ các hàng thừa đã được hiển thị trên trang. Hộp có tên “Lập chỉ mục Logic Tương đương” giải thích lý do tại sao mệnh đề được diễn đạt theo cách này.

Kế hoạch thực thi cho thấy rằng cơ sở dữ liệu sử dụng phần đầu tiên của nơi mệnh đề làm vị ngữ truy cập.

```
-------------------------------------------------------------------------------
| Id | Hoạt động                                    | Tên          | Hàng | Chi phí |
-------------------------------------------------------------------------------
| 0 | TUYỂN CHỌN ĐỀ SỐ 10 |                                                    4 |
| * 1 | ĐẾM DỪNG LẠI |                                                          |
| 2 | XEM 10 | | 3 |                                                        4 |
             TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | BÁN HÀNG | 50218 | | * 4 | 2 |   4 |
               MÔ TẢ QUÉT CHỈ SỐ RANGE | SL_DTIT |                            3 |
-------------------------------------------------------------------------------


Thông tin vị ngữ (được xác định bằng id hoạt động):
---------------------------------------------------

    1 - bộ lọc (ROWNUM <= 10)
    4 - truy cập ("SALE_DATE" <=: SALE_DATE)
          bộ lọc ("SALE_DATE" <>: SALE_DATE
                HOẶC "SALE_ID" <TO_NUMBER (: SALE_ID))
```

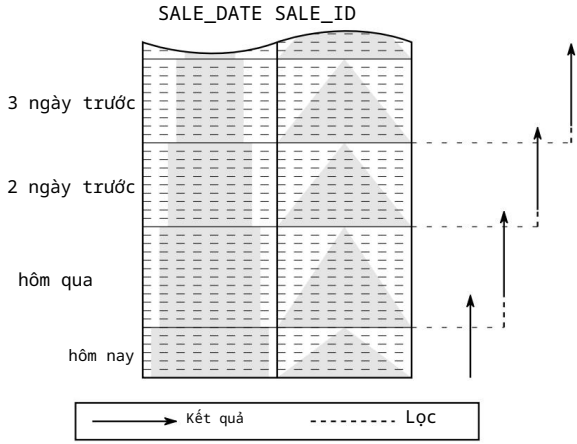
Các vị từ truy cập vào SALE_DATE cho phép cơ sở dữ liệu bỏ qua ngày được hiển thị đầy đủ trên các trang trước. Phần thứ hai của nơi mệnh đề chỉ là một vị từ bộ lọc. Điều đó có nghĩa là cơ sở dữ liệu kiểm tra

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

một vài mục nhập từ trang trước một lần nữa, nhưng bỏ chúng ngay lập tức. Hình 7.3 cho thấy đường dẫn truy cập tương ứng.

Hình 7.3. Truy cập bằng phương pháp tìm kiếm



Giá trị hàng trong SQL Bên

cạnh các giá trị vô hướng thông thường, tiêu chuẩn SQL cũng định nghĩa các hàm tạo giá trị hàng được gọi là. Chúng "Chỉ định một bộ giá trị có thứ tự sẽ được xây dựng thành một hàng hoặc một phần hàng" [SQL: 92, §7.1: <phương thức tạo giá trị hàng>]. Về mặt cú pháp, giá trị hàng là danh sách trong ngoặc. Cú pháp này được biết đến nhiều nhất để sử dụng trong câu lệnh chèn .

Tuy nhiên, sử dụng các hàm tạo giá trị hàng trong mệnh đề where ít được biết đến hơn nhưng vẫn hoàn toàn hợp lệ. Tiêu chuẩn SQL thực sự định nghĩa tất cả các toán tử so sánh cho các hàm tạo giá trị hàng. Ví dụ, định nghĩa cho các phép toán nhỏ hơn là như sau:

> "Rx <Ry" đúng nếu và chỉ khi RXi = RYi với mọi i <n và RXn <RYn với một số n.
>
> —SQL: 92, §8.2.7.2

Trong đó i và n phản ánh các chỉ số vị trí trong danh sách. Điều đó có nghĩa là giá trị hàng RX nhỏ hơn RY nếu bất kỳ giá trị nào RXn nhỏ hơn RYn tương ứng và tất cả các cặp giá trị trước đó đều bằng nhau (RXi = RYi; for i <n).

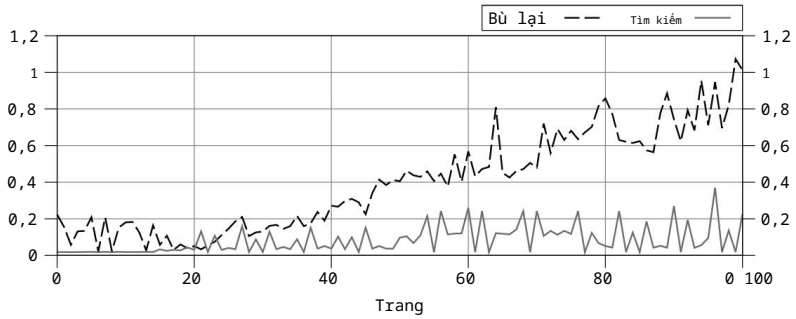
Định nghĩa này làm cho biểu thức RX <RY đồng nghĩa với "RX sắp xếp trước RY", đây chính xác là logic mà chúng ta cần cho phương pháp tìm kiếm.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Hình 7.4 so sánh các đặc tính hiệu suất của phương pháp bù và phương pháp tìm kiếm. Độ chính xác của phép đo không đủ để thấy sự khác biệt ở phía bên trái của biểu đồ, tuy nhiên sự khác biệt có thể nhìn thấy rõ ràng từ khoảng trang 20 trở đi.

Hình 7.4. Khả năng mở rộng khi tìm nạp trang tiếp theo



Tất nhiên, phương pháp tìm kiếm cũng có nhược điểm, khó khăn trong việc xử lý nó là điều quan trọng nhất. Bạn không chỉ phải cụm từ mệnh đề where một cách cẩn thận — bạn cũng không thể tìm nạp các trang tùy ý. Hơn nữa bạn cần phải đảo ngược tất cả các thao tác so sánh và sắp xếp để thay đổi hướng duyệt. Chính xác là hai chức năng này — bỏ qua trang và duyệt ngược — không cần thiết khi sử dụng cơ chế cuộn vô hạn cho giao diện người dùng.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Lập chỉ mục Logic tương đương Một điều

kiện logic luôn có thể được biểu diễn theo những cách khác nhau. Ví dụ: bạn cũng có thể triển khai logic bỏ qua được hiển thị ở trên như sau:

```
Ở ĐÂU (
           (sale_date <?)
         HOẶC
           (sale_date =? VÀ sale_id <?)
        )
```

Biến thể này chỉ sử dụng bao gồm các điều kiện và có lẽ dễ hiểu hơn – ít nhất là đối với con người. Cơ sở dữ liệu có một quan điểm khác. Họ không nhận ra rằng mệnh đề where chọn tất cả các hàng bắt đầu bằng cặp SALE_DATE / SALE_ID tương ứng – miễn là SALE_DATE giống nhau cho cả hai nhánh. Thay vào đó, cơ sở dữ liệu sử dụng toàn bộ mệnh đề where làm vị ngữ bộ lọc. Ít nhất chúng tôi có thể mong đợi trình tối ưu hóa "tính đến điều kiện SALE_DATE <=? ngoài "của hai hoặc-chi nhánh, nhưng không có cơ sở dữ liệu nào cung cấp dịch vụ này.

Tuy nhiên, chúng tôi có thể thêm điều kiện thừa này theo cách thủ công – mặc dù nó không làm tăng khả năng đọc:

```
WHERE sale_date <=?
  VÀ (
           (sale_date <?)

         HOẶC (sale_date =? VÀ sale_id <?)
        )
```

May mắn thay, tất cả các cơ sở dữ liệu đều có thể sử dụng phần này của mệnh đề where làm vị từ truy cập. Tuy nhiên, mệnh đề đó thậm chí còn khó hiểu hơn như logic xấp xỉ được trình bày ở trên. Hơn nữa, logic ban đầu tránh rủi ro rằng phần "không cần thiết" (thừa) vô tình bị loại bỏ khỏi mệnh đề where sau này.

Machine Translated by Google

Sử dụng các chức năng của cửa sổ để phân trang

Các hàm cửa sổ cung cấp một cách khác để thực hiện phân trang trong SQL. Đây là một phương pháp linh hoạt và trên hết là tuân thủ các tiêu chuẩn. Tuy nhiên, chỉ SQL Server và cơ sở dữ liệu Oracle mới có thể sử dụng chúng cho một truy vấn N hàng đầu. PostgreSQL không hủy bỏ quá trình quét chỉ mục sau khi tìm nạp đủ và do đó thực thi các truy vấn này rất kém hiệu quả. MySQL không không hỗ trợ các chức năng cửa sổ ở tất cả.

Ví dụ sau sử dụng hàm cửa sổ ROW_NUMBER cho một truy vấn phân trang:

```
LỰA CHỌN *
   FROM (CHỌN doanh số. *
                    , ROW_NUMBER () HẾT (ORDER BY sale_date DESC
                                                                  ,  sale_id DESC) rn
               Từ bán hàng
          ) tmp
 WHERE rn từ 11 đến 20
 ĐẶT HÀNG THEO ngày bán_giá DESC, ngày_mua_được DESC;
```

Hàm ROW_NUMBER liệt kê các hàng theo thứ tự sắp xếp được định nghĩa trong mệnh đề over . Mệnh đề where bên ngoài sử dụng cách liệt kê này để giới hạn kết quả ở trang thứ hai (từ hàng 11 đến hàng 20).

Cơ sở dữ liệu Oracle nhận ra điều kiện hủy bỏ và sử dụng chỉ mục trên SALE_DATE và SALE_ID để tạo ra hành vi top-N liên tục:

```
-----------------------------------------------------------------------------
| Id | Hoạt động                                        | Tên          | Hàng | Chi phí |
-----------------------------------------------------------------------------
| 0 | CHỌN BÁO CÁO                                                      | 1004K | 36877 |
| * 1 | LƯỢT XEM                                                          | 1004K | 36877 |
| * 2 | WINDOW NOSORT STOPKEY                                             | 1004K | 36877 |
| 3 |        TRUY CẬP BẢNG THEO CHỈ SỐ ROWID | BÁN HÀNG | 1004K | 36877 |
| 4 |         MÔ TẢ QUÉT CHỈ SỐ ĐẦY ĐỦ | SL_DTID | 1004K | 2955 |
-----------------------------------------------------------------------------


Thông tin vị ngữ (được xác định bằng id hoạt động):
---------------------------------------------------

1 - bộ lọc ("RN"> = 11 VÀ "RN" <= 20)
2 - bộ lọc (ROW_NUMBER () HẾT (
                ĐẶT HÀNG THEO "SALE_DATE" DESC, "SALE_ID" DESC) <= 20)
```


Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Thao tác WINDOW NOSORT STOPKEY chỉ ra rằng không có thao tác sắp xếp (NOSORT) và cơ sở dữ liệu hủy bỏ việc thực thi khi đạt đến ngưỡng trên (STOPKEY). Xem xét rằng các hoạt động bị hủy bỏ được thực hiện theo cách thức tổng hợp, điều đó có nghĩa là truy vấn này hiệu quả như phương pháp bù đắp được giải thích trong phần trước.

Tuy nhiên, điểm mạnh của các hàm cửa sổ không phải là phân trang mà là tính toán phân tích. Nếu bạn chưa bao giờ sử dụng các chức năng cửa sổ trước đây, bạn chắc chắn nên dành một vài giờ để nghiên cứu tài liệu tương ứng.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Chương 8

Sửa đổi dữ liệu

Cho đến nay chúng ta mới chỉ thảo luận về hiệu suất truy vấn, nhưng SQL không chỉ là về các truy vấn. Nó cũng hỗ trợ thao tác dữ liệu. Các lệnh tương ứng— chèn, xóa và cập nhật — tạo thành cái gọi là "ngôn ngữ thao tác dữ liệu" (DML) —một phần của tiêu chuẩn SQL. Hiệu suất của các lệnh này phần lớn bị ảnh hưởng tiêu cực bởi các chỉ mục.

Một chỉ mục là dự phòng thuần túy. Nó chỉ chứa dữ liệu cũng được lưu trữ trong bảng. Trong các hoạt động ghi, cơ sở dữ liệu phải giữ cho những phần dư thừa đó nhất quán. Cụ thể, nó có nghĩa là chèn, xóa và cập nhật không chỉ ảnh hưởng đến bảng mà còn cả các chỉ mục lưu giữ bản sao của dữ liệu bị ảnh hưởng.

Chèn

Số lượng chỉ mục trên một bảng là yếu tố chi phối nhiều nhất đối với hiệu suất chèn . Bảng càng có nhiều chỉ mục, việc thực thi càng trở nên chậm hơn. Câu lệnh insert là thao tác duy nhất không thể trực tiếp hưởng lợi từ việc lập chỉ mục vì nó không có mệnh đề where .

Thêm một hàng mới vào bảng bao gồm một số bước. Đầu tiên, cơ sở dữ liệu phải tìm một nơi để lưu hàng. Đối với một bảng heap thông thường — không có thứ tự hàng cụ thể — cơ sở dữ liệu có thể lấy bất kỳ khối bảng nào có đủ dung lượng trống. Đây là một quá trình rất đơn giản và nhanh chóng, chủ yếu được thực thi trong bộ nhớ chính. Tất cả những gì cơ sở dữ liệu phải làm sau đó là thêm mục nhập mới vào khối dữ liệu tương ứng.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

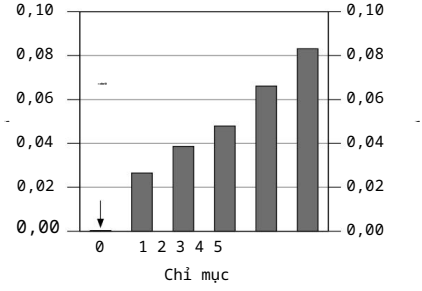
Nếu có các chỉ mục trên bảng, cơ sở dữ liệu phải đảm bảo mục nhập mới cũng được tìm thấy thông qua các chỉ mục này. Vì lý do này, nó phải thêm mục nhập mới vào mỗi và mọi chỉ mục trên bảng đó. Do đó, số lượng chỉ mục là một cấp số nhân cho chi phí của một câu lệnh chèn .

Hơn nữa, việc thêm một mục nhập vào một chỉ mục sẽ đắt hơn nhiều so với việc chèn một mục vào cấu trúc heap vì cơ sở dữ liệu phải giữ thứ tự chỉ mục và sự cân bằng của cây. Điều đó có nghĩa là mục nhập mới không thể được ghi vào bất kỳ khối nào - nó thuộc về một nút lá cụ thể. Mặc dù cơ sở dữ liệu sử dụng chính cây chỉ mục để tìm nút lá chính xác, nhưng nó vẫn phải đọc một vài khối chỉ mục cho quá trình duyệt qua cây.

Khi nút lá chính xác đã được xác định, cơ sở dữ liệu xác nhận rằng có đủ không gian trống còn lại trong nút này. Nếu không, cơ sở dữ liệu sẽ chia nút lá và phân phối các mục nhập giữa nút cũ và nút mới. Quá trình này cũng ảnh hưởng đến tham chiếu trong nút nhánh tương ứng vì nó cũng phải được sao chép. Không cần phải nói, nút nhánh cũng có thể hết dung lượng nên nó cũng có thể phải được chia nhỏ. Trong trường hợp xấu nhất, cơ sở dữ liệu phải chia tất cả các nút cho đến nút gốc. Đây là trường hợp duy nhất mà cây có thêm một lớp và phát triển theo chiều sâu.

Rốt cuộc, duy trì chỉ mục là phần tốn kém nhất của hoạt động chèn . Điều đó cũng có thể thấy trong Hình 8.1, “Chèn hiệu suất theo số chỉ mục”: thời gian thực hiện hầu như không hiển thị nếu bảng không có bất kỳ chỉ mục nào. Tuy nhiên, thêm một chỉ mục duy nhất là đủ để tăng thời gian thực thi lên một trăm. Mỗi chỉ mục bổ sung sẽ làm chậm quá trình thực thi hơn nữa.

Hình 8.1. Chèn hiệu suất theo số chỉ mục



Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Ghi chú

Chỉ số đầu tiên tạo ra sự khác biệt lớn nhất.

Để tối ưu hóa hiệu suất chèn , điều rất quan trọng là giữ cho số lượng chỉ mục nhỏ.

Mẹo

Sử dụng các chỉ mục một cách có chủ ý và tiết kiệm, và tránh các chỉ mục dư thừa bất cứ khi nào có thể. Điều này cũng có lợi cho việc xóa và cập nhật các câu lệnh.

Chỉ xem xét các câu lệnh chèn , tốt nhất là tránh hoàn toàn các chỉ mục – điều này mang lại hiệu suất chèn tốt nhất cho đến nay . Tuy nhiên, các bảng không có chỉ mục là không thực tế trong các ứng dụng thế giới thực. Bạn thường muốn lấy lại dữ liệu đã lưu trữ vì vậy bạn cần chỉ mục để cải thiện tốc độ truy vấn. Ngay cả các bảng nhật ký chỉ ghi thường có một khóa chính và một chỉ mục tương ứng.

Tuy nhiên, hiệu suất không có chỉ mục tốt đến mức có thể tạm thời loại bỏ tất cả các chỉ mục trong khi tải một lượng lớn dữ liệu - miễn là các chỉ mục không cần thiết bởi bất kỳ câu lệnh SQL nào khác trong thời gian chờ đợi. Điều này có thể giải phóng một tốc độ đáng kể có thể nhìn thấy trong biểu đồ và trên thực tế, đây là một thực tế phổ biến trong các kho dữ liệu.

Hãy nghĩ về nó

Hình 8.1 sẽ thay đổi như thế nào khi sử dụng bảng được tổ chức theo chỉ mục hoặc chỉ mục theo nhóm?

Có cách gián tiếp nào mà một câu lệnh chèn có thể có lợi từ việc lập chỉ mục không? Đó là, một chỉ mục bổ sung có thể làm cho một câu lệnh chèn nhanh hơn không?

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

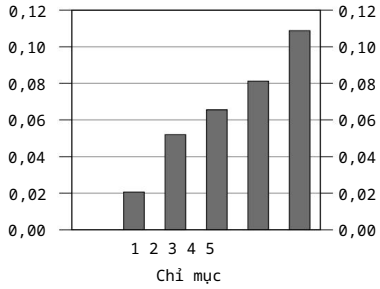
Machine Translated by Google

Xóa bỏ

Không giống như câu lệnh insert , câu lệnh delete có mệnh đề where có thể sử dụng tất cả các phương pháp được mô tả trong Chương 2, "Mệnh đề Where", để hưởng lợi trực tiếp từ các chỉ mục. Trên thực tế, câu lệnh xóa hoạt động giống như một lựa chọn được theo sau bởi một bước bổ sung để xóa các hàng đã xác định.

Việc xóa hàng thực sự là một quá trình tương tự như việc chèn một hàng mới – đặc biệt là việc xóa các tham chiếu khỏi các chỉ mục và các hoạt động để giữ cân bằng các cây chỉ mục. Do đó, biểu đồ hiệu suất được thể hiện trong Hình 8.2 rất giống với biểu đồ được hiển thị cho phần chèn.

Hình 8.2. Xóa hiệu suất theo số chỉ mục



Về lý thuyết, chúng tôi mong đợi hiệu suất xóa tốt nhất cho một bảng không có bất kỳ chỉ mục nào – cũng như đối với chèn. Tuy nhiên, nếu không có chỉ mục, cơ sở dữ liệu phải đọc toàn bộ bảng để tìm các hàng sẽ bị xóa. Điều đó có nghĩa là xóa hàng sẽ nhanh nhưng việc tìm kiếm sẽ rất chậm. Do đó trường hợp này không được thể hiện trong Hình 8.2.

Tuy nhiên, việc thực thi một câu lệnh xóa mà không có chỉ mục có thể có ý nghĩa cũng như việc thực thi một câu lệnh select mà không có chỉ mục có thể có ý nghĩa nếu nó trả về một phần lớn của bảng.

Mẹo

Ngay cả các câu lệnh xóa và cập nhật cũng có một kế hoạch thực thi.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Câu lệnh xóa không có mệnh đề where là một ví dụ rõ ràng trong đó cơ sở dữ liệu không thể sử dụng chỉ mục, mặc dù đây là trường hợp đặc biệt có lệnh SQL riêng: bảng truncate. Lệnh này có tác dụng tương tự như lệnh xóa mà không có ngoại trừ việc nó xóa tất cả các hàng trong một lần chụp. Nó rất nhanh nhưng có hai tác dụng phụ quan trọng: (1) nó thực hiện một cam kết ngầm (ngoại lệ: PostgreSQL); (2) nó không thực hiện bất kỳ trình kích hoạt nào.

> Tác dụng phụ của điều khiển đồng
>
> thời đa vũ trụ MVCC (MVCC) là một cơ chế cơ sở dữ liệu cho phép truy cập dữ liệu đồng thời không chặn và chế độ xem giao dịch nhất quán. Tuy nhiên, việc triển khai khác nhau giữa các cơ sở dữ liệu và thậm chí có thể có những ảnh hưởng đáng kể đến hiệu suất.
>
> Ví dụ: cơ sở dữ liệu PostgreSQL chỉ giữ thông tin phiên bản (= thông tin hiển thị) ở cấp bảng: xóa một hàng chỉ đặt cờ "đã xóa" trong khối bảng. Do đó, hiệu suất xóa của PostgreSQL không phụ thuộc vào số lượng chỉ mục trên bảng. Việc xóa vật lý hàng trong bảng và duy trì chỉ mục liên quan chỉ được thực hiện trong quá trình VACCUM.

Cập nhật

Một câu lệnh cập nhật phải định vị lại các mục chỉ mục đã thay đổi để duy trì thứ tự chỉ mục. Đối với điều đó, cơ sở dữ liệu phải xóa mục nhập cũ và thêm mục nhập mới tại vị trí mới. Thời gian phản hồi về cơ bản giống như đối với các câu lệnh xóa và chèn tương ứng với nhau.

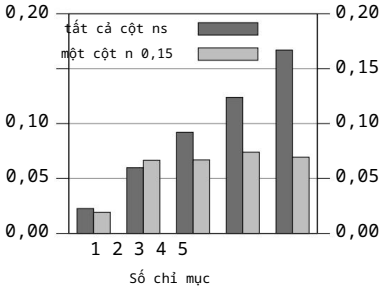
Hiệu suất cập nhật , giống như chèn và xóa, cũng phụ thuộc vào số lượng chỉ mục trên bảng. Sự khác biệt duy nhất là các câu lệnh cập nhật không nhất thiết ảnh hưởng đến tất cả các cột vì chúng thường chỉ sửa đổi một số cột được chọn. Do đó, một câu lệnh cập nhật không nhất thiết ảnh hưởng đến tất cả các chỉ mục trên bảng mà chỉ ảnh hưởng đến những chỉ mục chứa các cột được cập nhật.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Hình 8.3 cho thấy thời gian phản hồi cho hai câu lệnh cập nhật : câu lệnh đặt tất cả các cột và ảnh hưởng đến tất cả các chỉ mục và câu lệnh thứ hai cập nhật một cột duy nhất để nó chỉ ảnh hưởng đến một chỉ mục.

Hình 8.3. Cập nhật hiệu suất theo chỉ mục và số lượng cột



Cập nhật trên tất cả các cột hiển thị cùng một kiểu mà chúng ta đã quan sát trong các phần trước: thời gian phản hồi tăng lên với mỗi chỉ mục bổ sung. Thời gian phản hồi của câu lệnh cập nhật chỉ ảnh hưởng đến một chỉ mục không tăng quá nhiều vì nó giữ nguyên hầu hết các chỉ mục.

Để tối ưu hóa hiệu suất cập nhật , bạn phải chú ý chỉ cập nhật những cột đã được thay đổi. Điều này là hiển nhiên nếu bạn viết câu lệnh cập nhật theo cách thủ công. Tuy nhiên, các công cụ ORM có thể tạo các câu lệnh cập nhật thiết lập tất cả các cột. Ví dụ: Hibernate thực hiện điều này khi tắt chế độ cập nhật động. Kể từ phiên bản 4.0, chế độ này được bật theo mặc định.

Khi sử dụng các công cụ ORM, bạn nên thỉnh thoảng bật tính năng ghi truy vấn trong môi trường phát triển để xác minh các câu lệnh SQL đã tạo. Mẹo có tên “Bật tính năng ghi nhật ký SQL” trên trang 95 có tổng quan ngắn gọn về cách bật tính năng ghi nhật ký SQL trong một số công cụ ORM được sử dụng rộng rãi.

Hãy nghĩ về nó

Bạn có thể nghĩ đến trường hợp các câu lệnh chèn hoặc xóa không ảnh hưởng đến tất cả các chỉ mục của bảng không?

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Phụ lục A

Kế hoạch thực hiện

Trước khi cơ sở dữ liệu có thể thực thi một câu lệnh SQL, trình tối ưu hóa phải tạo một kế hoạch thực thi cho nó. Cơ sở dữ liệu sau đó thực hiện kế hoạch này theo cách từng bước. Về mặt này, trình tối ưu hóa rất giống với trình biên dịch vì nó dịch mã nguồn (câu lệnh SQL) thành một chương trình thực thi (kế hoạch thực thi).

Kế hoạch thực thi là nơi đầu tiên cần xem xét khi tìm kiếm nguyên nhân của các báo cáo chậm. Các phần sau giải thích cách truy xuất và đọc một kế hoạch thực thi để tối ưu hóa hiệu suất trong các cơ sở dữ liệu khác nhau.

Nội dung

Cơ sở dữ liệu Oracle ....
166 Lập kế hoạch thực thi ....
166 Hoạt động ....
167 Phân biệt Access và Filter-Predicates 170
PostgreSQL ....
172 Bắt một kế hoạch thực hiện ... . 172 Hoạt
động ....
174 Phân biệt Access và Filter-Predicates .. .. 177 Máy chủ
SQL ....
180 Lập kế hoạch thực hiện ....
180 Hoạt động ....
182 Phân biệt Access và Filter-Predicates 185
MySQL ... .
188 Lập kế hoạch thực hiện ....
188 Hoạt
động ....

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Cơ sở dữ liệu Oracle

Hầu hết các môi trường phát triển (IDE) có thể rất dễ dàng hiển thị một kế hoạch thực thi nhưng sử dụng các cách rất khác nhau để định dạng chúng trên màn hình. Phương pháp được mô tả trong phần này cung cấp kế hoạch thực thi như được hiển thị trong suốt cuốn sách và chỉ yêu cầu cơ sở dữ liệu Oracle trong phiên bản 9iR2 hoặc mới hơn.

Lên kế hoạch thực thi

Xem một kế hoạch thực thi trong cơ sở dữ liệu Oracle bao gồm hai bước:

1. giải thích kế hoạch cho - lưu kế hoạch thực thi trong PLAN_TABLE.

2. Định dạng và hiển thị kế hoạch thực hiện.

Tạo và lưu một kế hoạch thực thi

Để tạo một kế hoạch thực thi, bạn chỉ cần thêm tiền tố câu lệnh SQL tương ứng với kế hoạch giải thích cho:

```
GIẢI THÍCH KẾ HOẠCH CHO select * từ kép;
```

Bạn có thể thực hiện kế hoạch giải thích cho lệnh trong bất kỳ môi trường phát triển nào hoặc SQL * Plus. Tuy nhiên, nó sẽ không hiển thị kế hoạch mà lưu nó vào một bảng có tên PLAN_TABLE. Bắt đầu với bản phát hành 10g, bảng này tự động có sẵn dưới dạng bảng tạm thời toàn cầu. Với các bản phát hành trước, bạn phải tạo nó trong mỗi lược đồ nếu cần. Yêu cầu quản trị viên cơ sở dữ liệu của bạn tạo nó cho bạn hoặc cung cấp câu lệnh tạo bảng từ cài đặt cơ sở dữ liệu Oracle:

```
$ ORACLE_HOME / rdbms / admin / utlxplan.sql
```

Bạn có thể thực hiện câu lệnh này trong bất kỳ lược đồ nào bạn muốn để tạo PLAN_TABLE trong lược đồ này.

Cảnh báo Kế

hoạch giải thích cho lệnh không nhất thiết phải tạo ra cùng một kế hoạch thực thi như khi thực hiện câu lệnh.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Hiển thị các Kế hoạch Thực thi

Gói DBMS_XPLAN được giới thiệu với bản phát hành 9iR2 và có thể định dạng và hiển thị các kế hoạch thực thi từ PLAN_TABLE. Ví dụ sau cho biết cách hiển thị kế hoạch thực thi cuối cùng đã được giải thích trong phiên cơ sở dữ liệu hiện tại:

```
chọn * từ bảng (dbms_xplan.display);
```

Một lần nữa, nếu câu lệnh đó không hoạt động hiệu quả, bạn nên hỏi DBA của bạn để được hỗ trợ.

Truy vấn sẽ hiển thị kế hoạch thực hiện như trong sách:

```
------------------------------------------------------------------------------
| Id | Hoạt động                    | Tên | Hàng | Số byte | Chi phí (% CPU) |.
------------------------------------------------------------------------------
| 0 | TUYỂN CHỌN BÁO CÁO | | |                        1 |         2 |         2 (0) |.
    1  | BẢNG TRUY CẬP ĐẦY ĐỦ | KÉP |                  1 |         2 |         2 (0) |.
------------------------------------------------------------------------------
```

Một số cột được hiển thị trong kế hoạch thực thi này đã bị xóa trong cuốn sách cho phù hợp hơn trên trang.

Hoạt động

Chỉ mục và Quyền truy cập Bảng

CHỈ SỐ QUÉT DUY NHẤT

INDEX UNIQUE SCAN chỉ thực hiện duyệt cây B. Kho dữ liệu sử dụng thao tác này nếu một ràng buộc duy nhất đảm bảo rằng tìm kiếm tiêu chí sẽ không khớp với nhiều hơn một mục nhập. Xem thêm Chương 1, "Giải phẫu của một chỉ mục".

QUÉT NGUỒN CHỈ SỐ

QUÉT NGUỒN CHỈ SỐ thực hiện chuyển ngang cây B và đi theo lá chuỗi nút để tìm tất cả các mục nhập phù hợp. Xem thêm Chương 1, “Giải phẫu của một Chỉ số ”.

Cái được gọi là các vị từ bộ lọc chỉ mục thường gây ra các vấn đề về hiệu suất cho QUÉT NGUỒN CHỈ SỐ. Phần tiếp theo giải thích cách xác định họ.

Machine Translated by Google

CHỈ SỐ QUÉT ĐẦY ĐỦ

Đọc toàn bộ chỉ mục — tất cả các hàng — theo thứ tự chỉ mục. Tùy thuộc vào các số liệu thống kê khác nhau của hệ thống, cơ sở dữ liệu có thể thực hiện thao tác này nếu nó cần tất cả các hàng theo thứ tự chỉ mục — ví dụ: vì thứ tự theo mệnh đề tương ứng. Thay vào đó, trình tối ưu hóa cũng có thể sử dụng INDEX FAST FULL SCAN và thực hiện một thao tác sắp xếp bổ sung. Xem Chương 6, "Sắp xếp và phân nhóm".

CHỈ SỐ QUÉT ĐẦY ĐỦ NHANH CHÓNG

Đọc toàn bộ chỉ mục — tất cả các hàng — khi được lưu trữ trên đĩa. Thao tác này thường được thực hiện thay vì quét toàn bộ bảng nếu tất cả các cột bắt buộc đều có sẵn trong chỉ mục. Tương tự như TABLE ACCESS FULL, INDEX FAST FULL SCAN có thể được hưởng lợi từ các hoạt động đọc nhiều khối.
Xem Chương 5, "Dữ liệu phân cụm".

TRUY CẬP BẢNG THEO CHỈ SỐ ROWID

Lấy một hàng từ bảng bằng ROWID được truy xuất từ lần tra cứu chỉ mục trước đó. Xem thêm Chương 1, "Giải phẫu một chỉ mục".

TABLE ACCESS FULL Đây

còn được gọi là quét toàn bộ bảng. Đọc toàn bộ bảng — tất cả các hàng và cột — như được lưu trữ trên đĩa. Mặc dù các thao tác đọc đa khối cải thiện đáng kể tốc độ quét toàn bảng, nhưng đây vẫn là một trong những thao tác tốn kém nhất. Bên cạnh tỷ lệ IO cao, quá trình quét toàn bộ bảng phải kiểm tra tất cả các hàng của bảng vì vậy nó cũng có thể tiêu tốn một lượng thời gian CPU đáng kể. Xem thêm "Quét toàn bảng" trên trang 13.

Tham gia

Nói chung, quy trình hoạt động tham gia chỉ xử lý hai bảng cùng một lúc. Trong trường hợp một truy vấn có nhiều liên kết hơn, chúng được thực hiện tuần tự: hai bảng đầu tiên, sau đó là kết quả trung gian với bảng tiếp theo. Trong ngữ cảnh của phép nối, thuật ngữ "bảng" do đó cũng có thể có nghĩa là "kết quả trung gian".

NESTED LOOPS JOIN Kết

hợp hai bảng bằng cách tìm nạp kết quả từ một bảng và truy vấn bảng khác cho mỗi hàng từ bảng đầu tiên. Xem thêm "Các vòng lặp lồng nhau" trên trang 92.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

THAM GIA HASH

Tham gia băm tải các bản ghi ứng viên từ một phía của phép nối vào một bảng băm, sau đó được thăm dò cho từng hàng từ phía bên kia của phép nối. Xem thêm “Tham gia băm” trên trang 101.

MERGE THAM GIA

Kết hợp hợp nhất kết hợp hai danh sách được sắp xếp giống như một dây kéo. Cả hai mặt của phép nối phải được sắp xếp trước. Xem thêm “Sắp xếp Hợp nhất” trên trang 109.

Sắp xếp và phân nhóm

SẮP XẾP LỆNH THEO

Sắp xếp kết quả theo mệnh đề thứ tự . Thao tác này cần một lượng lớn bộ nhớ để hiện thực hóa kết quả trung gian (không phải là pipelined). Xem thêm “Thứ tự lập chỉ mục theo” trên trang 130.

SẮP XẾP ĐƠN HÀNG THEO STOPKEY

Sắp xếp một tập hợp con của kết quả theo thứ tự của mệnh đề. Được sử dụng cho các truy vấn top-N nếu không thể thực thi pipelined. Xem thêm “Truy vấn các hàng Top-N” trên trang 143.

SẮP XẾP NHÓM THEO

Sắp xếp tập hợp kết quả trên nhóm theo cột và tổng hợp kết quả đã sắp xếp trong bước thứ hai. Thao tác này cần một lượng lớn bộ nhớ để hiện thực hóa tập kết quả trung gian (không phải là pipelined). Xem thêm “Nhóm lập chỉ mục theo” trên trang 139.

SẮP XẾP NHÓM THEO NOSORT

Tổng hợp một tập hợp được sắp xếp trước theo nhóm theo mệnh đề. Thao tác này không đệm cho kết quả trung gian: nó được thực hiện theo cách pipelined. Xem thêm “Nhóm lập chỉ mục theo” trên trang 139.

HASH GROUP BY Nhóm

kết quả bằng cách sử dụng một bảng băm. Thao tác này cần một lượng lớn bộ nhớ để hiện thực hóa tập kết quả trung gian (không phải là pipelined). Đầu ra không được sắp xếp theo bất kỳ cách nào có ý nghĩa. Xem thêm “Nhóm lập chỉ mục theo” trên trang 139.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Truy vấn Top-N

Hiệu quả của các truy vấn top-N phụ thuộc vào chế độ thực thi của các hoạt động bên dưới. Chúng rất kém hiệu quả khi hủy bỏ các hoạt động không liên kết như SORT ORDER BY.

COUNT STOPKEY Hủy
: bỏ các hoạt động cơ bản khi số lượng hàng mong muốn được tìm nạp. Xem thêm phần có tên “Truy vấn hàng đầu-N”.

WINDOW NOSORT STOPKEY Sử dụng
: một hàm cửa sổ (mệnh đề over) để hủy bỏ việc thực thi khi số lượng hàng mong muốn được tìm nạp. Xem thêm “Sử dụng các chức năng của cửa sổ để phân trang” trên trang 156.

Phân biệt Quyền truy cập và Dự đoán bộ lọc

Cơ sở dữ liệu Oracle sử dụng ba phương pháp khác nhau để áp dụng mệnh đề trong đó (vị từ) :

Vị từ Access (“truy cập”)
: Các vị từ truy cập thể hiện điều kiện bắt đầu và dừng của việc truyền tải nút lá.

Vị từ bộ lọc chỉ mục (“bộ lọc” cho các hoạt động chỉ mục)
: Các vị từ bộ lọc chỉ mục chỉ được áp dụng trong quá trình duyệt nút lá. Chúng không đóng góp vào điều kiện bắt đầu và dừng và không thu hẹp phạm vi quét.

Vị từ bộ lọc mức bảng (“bộ lọc” cho các thao tác trên bảng)
: Các dự đoán trên các cột không phải là một phần của chỉ mục được đánh giá ở cấp bảng. Để điều đó xảy ra, cơ sở dữ liệu phải tải hàng từ bảng trước.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Các kế hoạch thực thi đã được tạo bằng tiện ích DBMS_XPLAN (xem “Bắt một Kế hoạch Thực thi ”trên trang 166), hiển thị việc sử dụng chỉ mục trong“ Dự đoán Phần thông tin ”bên dưới kế hoạch thực thi dạng bảng:

```
| Id | Hoạt động                    | Tên              | Hàng | Chi phí |

| 0 | TUYỂN CHỌN BÁO CÁO | | 14 giờ 45 |
|   1  | SẮP XẾP TỔNG HỢP | | |
| * 2 | QUÉT NGUỒN CHỈ SỐ | QUY MÔ | 4485 | 14 giờ 45 |


Thông tin vị ngữ (được xác định bằng id hoạt động):
     2 - quyền truy cập ("SECTION" =: A VÀ "ID2" =: B)
           bộ lọc ("ID2" =: B)
```

Việc đánh số thông tin vị từ đề cập đến cột "Id" của kế hoạch thực hiện. Ở đó, cơ sở dữ liệu cũng hiển thị một dấu sao để đánh dấu các phép toán có thông tin vị ngữ.

Ví dụ này, được lấy từ chương “Hiệu suất và khả năng mở rộng”, cho thấy QUÉT NGUỒN CHỈ SỐ có quyền truy cập và lọc các vị từ. Người tiên tri cơ sở dữ liệu có đặc điểm là cũng hiển thị một số vị từ bộ lọc dưới dạng quyền truy cập các vị từ — ví dụ: ID2 =: B trong kế hoạch thực thi ở trên.

Quan trọng

Nếu một điều kiện hiển thị dưới dạng vị từ bộ lọc, thì đó là một vị từ bộ lọc — nó không quan trọng nếu nó cũng được hiển thị dưới dạng vị từ truy cập.

Điều này có nghĩa là QUÉT NGUỒN CHỈ SỐ quét toàn bộ phạm vi để tìm điều kiện "SECTION" =: A và áp dụng bộ lọc "ID2" =: B trên mỗi hàng.

Các vị từ bộ lọc ở cấp bảng được hiển thị cho quyền truy cập bảng tương ứng chẳng hạn như TRUY CẬP BẢNG THEO CHỈ SỐ ROWID hoặc TRUY CẬP BẢNG ĐẦY ĐỦ.

Machine Translated by Google

PostgreSQL

Các phương pháp được mô tả trong phần này áp dụng cho PostgreSQL 8.0 trở lên.

Lên kế hoạch thực thi

Một kế hoạch thực thi PostgreSQL được tìm nạp bằng cách đặt lệnh giải thích trước câu lệnh SQL. Tuy nhiên, có một hạn chế quan trọng: các câu lệnh SQL có tham số ràng buộc (ví dụ: $ 1, $ 2, v.v.) không thể được giải thích theo cách này — chúng cần được chuẩn bị trước:

```
CHUẨN BỊ stmt (int) NHƯ CHỌN $ 1;
```

Lưu ý rằng PostgreSQL sử dụng "$ n" cho các tham số liên kết. Lớp trừu tượng cơ sở dữ liệu của bạn có thể ẩn điều này để bạn có thể sử dụng dấu hỏi như được định nghĩa bởi tiêu chuẩn SQL.

Việc thực hiện câu lệnh đã chuẩn bị có thể được giải thích:

```
GIẢI THÍCH THỰC HIỆN stmt (1);
```

Cho đến PostgreSQL 9.1, kế hoạch thực thi đã được tạo bằng lệnh gọi chuẩn bị và do đó không thể xem xét các giá trị thực tế được cung cấp khi thực thi. Vì PostgreSQL 9.2, việc tạo kế hoạch thực thi bị hoãn lại cho đến khi thực thi và do đó có thể xem xét các giá trị thực tế cho các tham số ràng buộc.

Ghi chú

Các câu lệnh không có tham số ràng buộc có thể được giải thích trực tiếp:

```
GIẢI THÍCH CHỌN 1;
```

Trong trường hợp này, trình tối ưu hóa luôn xem xét các giá trị thực tế trong quá trình lập kế hoạch truy vấn. Nếu bạn sử dụng PostgreSQL 9.1 hoặc phiên bản cũ hơn và liên kết các tham số trong chương trình của mình, bạn cũng nên sử dụng giải thích với các tham số ràng buộc để truy xuất cùng một kế hoạch thực thi.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Đầu ra kế hoạch giải thích như sau:

```
                        KẾ HOẠCH QUERY
-------------------------------------------

 Kết quả (chi phí = 0,00..0,01 hàng = 1 chiều rộng = 0)
```

Đầu ra có thông tin tương tự như các kế hoạch thực thi Oracle được hiển thị trong suốt cuốn sách: tên hoạt động ("Kết quả"), chi phí liên quan, ước tính số hàng và chiều rộng hàng dự kiến.

Lưu ý rằng PostgreSQL hiển thị hai giá trị chi phí. Đầu tiên là chi phí cho việc khởi động, thứ hai là tổng chi phí cho việc thực thi nếu tất cả các hàng được truy xuất. Kế hoạch thực thi của cơ sở dữ liệu Oracle chỉ hiển thị giá trị thứ hai.

Lệnh giải thích PostgreSQL có hai tùy chọn. Tùy chọn VERBOSE cung cấp thông tin bổ sung như tên bảng đủ điều kiện – VERBOSE thường không có giá trị lắm.

Tùy chọn giải thích thứ hai là PHÂN TÍCH. Mặc dù nó được sử dụng rộng rãi, tôi khuyên bạn không nên có thói quen sử dụng nó tự động vì nó thực sự thực thi câu lệnh. Điều đó hầu như vô hại đối với các câu lệnh được chọn nhưng nó sửa đổi dữ liệu của bạn khi sử dụng nó để chèn, cập nhật hoặc xóa. Để tránh rủi ro vô tình sửa đổi dữ liệu của bạn, bạn có thể gửi kèm nó trong một giao dịch và thực hiện khôi phục dữ liệu sau đó.

Tùy chọn ANALYZE thực hiện câu lệnh và ghi lại thời gian thực và số lượng hàng. Điều đó có giá trị trong việc tìm ra nguyên nhân của các ước tính số lượng không chính xác (ước tính số lượng hàng):

```
BẮT ĐẦU;
GIẢI THÍCH THỰC HIỆN PHÂN TÍCH stmt (1);

                            KẾ HOẠCH QUERY
--------------------------------------------------

 Kết quả (chi phí = 0,00..0,01 hàng = 1 chiều rộng = 0)
            (thời gian thực = 0,002..0,002 hàng = 1 vòng lặp = 1)
 Tổng thời gian chạy: 0,020 ms

ROLLBACK;
```

Lưu ý rằng kế hoạch được định dạng để phù hợp hơn trên trang. PostgreSQL in các giá trị "thực tế" trên cùng một dòng với các giá trị ước tính.

Machine Translated by Google

> **Cảnh báo**
>
> giải thích phân tích thực thi câu lệnh được giải thích, ngay cả khi câu lệnh là một phần chèn, cập nhật hoặc xóa.

Số hàng là giá trị duy nhất được hiển thị trong cả hai phần — trong số liệu ước tính và số liệu thực tế. Điều đó cho phép bạn nhanh chóng tìm ra các ước tính số lượng sai sót.

Cuối cùng nhưng không kém phần quan trọng, các câu lệnh đã chuẩn bị phải được đóng lại:

```
THƯ VIỆN stmt;
```

Hoạt động

Chỉ mục và Quyền truy cập Bảng

Quét Seq

Thao tác quét Seq quét toàn bộ quan hệ (bảng) như được lưu trữ trên đĩa (như TABLE ACCESS FULL).

Quét chỉ mục

Quét chỉ mục thực hiện duyệt cây B, đi qua các nút lá để tìm tất cả các mục nhập phù hợp và tìm nạp dữ liệu bảng tương ứng. Nó giống như một thao tác QUÉT NGANG CHỈ SỐ , theo sau là hoạt động TRUY CẬP BẢNG THEO CHỈ SỐ ROWID . Xem thêm Chương 1, “Giải phẫu một chỉ mục”.

Cái gọi là các vị từ bộ lọc chỉ mục thường gây ra các vấn đề về hiệu suất cho Quét chỉ mục. Phần tiếp theo giải thích cách xác định chúng.

Chỉ quét chỉ mục (kể từ PostgreSQL 9.2)

Chỉ mục chỉ quét thực hiện duyệt cây B và đi qua các nút lá để tìm tất cả các mục nhập phù hợp. Không cần truy cập bảng vì chỉ mục có tất cả các cột để đáp ứng truy vấn (ngoại lệ: thông tin về khả năng hiển thị MVCC). Xem thêm “Quét chỉ lập chỉ mục” trên trang 116.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Bitmap Index Scan / Bitmap Heap Scan / Recheck Bài đăng của Cond
Tom Lane về danh sách gửi thư hiệu suất PostgreSQL rất rõ ràng và ngắn gọn.

> Quét chỉ mục thuần túy tìm nạp mỗi lần một bộ con trỏ từ chỉ mục và ngay lập tức truy cập bộ mã đó trong bảng. Quét bitmap tìm nạp tất cả các tuple-trỏ từ chỉ mục trong một lần, sắp xếp chúng bằng cách sử dụng cấu trúc dữ liệu "bitmap" trong bộ nhớ, sau đó truy cập các bộ dữ liệu trong bảng theo thứ tự vị trí tuple thực tế.
>
> — Ngõ 1

Tham gia hoạt động

Nói chung, quy trình hoạt động tham gia chỉ xử lý hai bảng cùng một lúc. Trong trường hợp một truy vấn có nhiều liên kết hơn, chúng được thực hiện tuần tự: hai bảng đầu tiên, sau đó là kết quả trung gian với bảng tiếp theo. Trong ngữ cảnh của phép nối, thuật ngữ "bảng" do đó cũng có thể có nghĩa là "kết quả trung gian".

Vòng lặp lồng
nhau Nối hai bảng bằng cách tìm nạp kết quả từ một bảng và truy vấn bảng khác cho mỗi hàng từ bảng đầu tiên. Xem thêm "Các vòng lặp lồng nhau" trên trang 92.

Hash Join / Hash
Tham gia băm tải các bản ghi ứng viên từ một phía của phép nối vào một bảng băm (được đánh dấu bằng Hash trong kế hoạch), sau đó được thăm dò cho từng bản ghi từ phía bên kia của phép nối. Xem thêm "Tham gia băm" trên trang 101.

Hợp nhất Tham
gia Liên kết hợp nhất (sắp xếp) kết hợp hai danh sách được sắp xếp giống như một dây kéo. Cả hai mặt của phép nối phải được sắp xếp trước. Xem thêm "Sắp xếp Hợp nhất" trên trang 109.

[1] http://archives.postgresql.org/pgsql-performance/2005-12/msg00623.php

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Sắp xếp và phân nhóm

Sắp xếp / Khóa

sắp xếp Sắp xếp tập hợp trên các cột được đề cập trong Khóa sắp xếp. Thao tác Sắp xếp cần một lượng lớn bộ nhớ để hiện thực hóa kết quả trung gian (không phải là pipelined). Xem thêm “Thứ tự lập chỉ mục theo” trên trang 130.

GroupAggregate Tổng

hợp một tập hợp được sắp xếp trước theo nhóm theo mệnh đề. Thao tác này không đệm số lượng lớn dữ liệu (pipelined). Xem thêm “Nhóm lập chỉ mục theo” trên trang 139.

HashAggregate Sử

dụng bảng băm tạm thời để nhóm các bản ghi. Hoạt động HashAggregate không yêu cầu tập dữ liệu được sắp xếp trước, thay vào đó nó sử dụng một lượng lớn bộ nhớ để hiện thực hóa kết quả trung gian (không phải là pipelined). Đầu ra không được sắp xếp theo bất kỳ cách nào có ý nghĩa. Xem thêm “Nhóm lập chỉ mục theo” trên trang 139.

Truy vấn Top-N

Giới hạn

Hủy bỏ các hoạt động cơ bản khi số lượng hàng mong muốn đã được tìm nạp. Xem thêm “Truy vấn các hàng Top-N” trên trang 143.

Hiệu quả của truy vấn top-N phụ thuộc vào chế độ thực thi của các hoạt động bên dưới. Nó rất kém hiệu quả khi hủy bỏ các hoạt động không theo chuỗi chẳng hạn như Sắp xếp.

WindowAgg

Cho biết việc sử dụng các chức năng của cửa sổ. Xem thêm “Sử dụng các chức năng của cửa sổ để phân trang” trên trang 156.

Thận trọng

PostgreSQL không thể thực thi các truy vấn top-N theo chuỗi khi sử dụng các hàm cửa sổ.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Phân biệt Quyền truy cập và Dự đoán bộ lọc

Cơ sở dữ liệu PostgreSQL sử dụng ba phương pháp khác nhau để áp dụng mệnh đề (vị từ) trong đó:

Access Predicate ("Chỉ mục Điều kiện")
: Các vị từ truy cập thể hiện điều kiện bắt đầu và dừng của việc truyền tải nút lá.

Dự đoán bộ lọc chỉ mục ("Điều kiện lập chỉ mục")
: Các vị từ bộ lọc chỉ mục chỉ được áp dụng trong quá trình duyệt nút lá. Chúng không đóng góp vào điều kiện bắt đầu và dừng và không thu hẹp phạm vi quét.

Vị từ bộ lọc cấp bảng ("Bộ lọc")
: Các dự đoán trên các cột không phải là một phần của chỉ mục được đánh giá ở cấp bảng. Để điều đó xảy ra, cơ sở dữ liệu phải tải hàng từ bảng heap trước.

Các kế hoạch thực thi PostgreSQL không hiển thị quyền truy cập chỉ mục và lọc các vị từ riêng biệt — cả hai đều hiển thị dưới dạng "Điều kiện chỉ mục". Điều đó có nghĩa là kế hoạch thực thi phải được so sánh với định nghĩa chỉ mục để phân biệt các vị từ truy cập với các vị từ bộ lọc chỉ mục.

> **Ghi chú**
>
> Kế hoạch giải thích PostgreSQL không cung cấp đủ thông tin để tìm các vị từ bộ lọc chỉ mục.

Các vị từ được hiển thị dưới dạng "Bộ lọc" luôn là các vị từ bộ lọc cấp bảng— ngay cả khi được hiển thị cho hoạt động Quét chỉ mục .

Hãy xem xét ví dụ sau, ví dụ này ban đầu xuất hiện trong Chương "Hiệu suất và khả năng mở rộng":

```
TẠO TABLE scale_data (
    phần NUMERIC NOT NULL, id1
               SỐ KHÔNG ĐẦY ĐỦ, id2
               SỐ KHÔNG ĐẦY ĐỦ
);
TẠO INDEX scale_data_key ON scale_data (section, id1);
```

Machine Translated by Google

Các bộ lọc chọn sau trên cột ID2 , không có trong chỉ mục:

```
CHUẨN BỊ stmt (int) AS SELECT count (*)
                                FROM scale_data
                              Phần WHERE = 1
                                 VÀ id2 = $ 1;
GIẢI THÍCH THỰC HIỆN stmt (1);

                              KẾ HOẠCH QUERY
..................................................

Tổng hợp (chi phí = 529346,31..529346,32 hàng = 1 chiều rộng = 0)
   Đầu ra: count (*)
   -> Quét chỉ mục bằng cách sử dụng scale_data_key trên scale_data
        (chi phí = 0,00..529338,83 hàng = 2989 chiều rộng = 0)
        Điều kiện chỉ mục: (scale_data.section = 1 :: numeric)
        Bộ lọc: (scale_data.id2 = ($ 1) :: số)
```

Vị từ ID2 hiển thị dưới dạng "Bộ lọc" bên dưới hoạt động Quét chỉ mục . Điều này là do PostgreSQL thực hiện quyền truy cập bảng như một phần của hoạt động Quét chỉ mục . Nói cách khác, hoạt động TABLE ACCESS BY INDEX ROWID của cơ sở dữ liệu Oracle được ẩn trong hoạt động Quét chỉ mục của PostgreSQL . Do đó, có thể Quét chỉ mục lọc trên các cột không được bao gồm trong chỉ mục.

Quan trọng

Các vị từ Bộ lọc PostgreSQL là các vị từ bộ lọc cấp bảng— ngay cả khi được hiển thị cho Quét chỉ mục.

Khi chúng tôi thêm chỉ mục từ chương “Hiệu suất và khả năng mở rộng”, chúng tôi có thể thấy rằng tất cả các cột hiển thị dưới dạng “Điều kiện chỉ mục” — bất kể chúng là vị từ truy cập hay bộ lọc.

```
TẠO INDEX scale_slow
              ON scale_data (section, id1, id2);
```

Kế hoạch thực thi với chỉ mục mới không hiển thị bất kỳ điều kiện lọc nào:

```
                              KẾ HOẠCH QUERY
..................................................

Tổng hợp (chi phí = 14215,98..14215,99 hàng = 1 chiều rộng = 0)
   Đầu ra: count (*)
   -> Quét chỉ mục bằng cách sử dụng scale_slow trên scale_data
        (chi phí = 0,00..14208,51 hàng = 2989 chiều rộng = 0)
        Điều kiện chỉ số: (section = 1 :: numeric AND id2 = ($ 1) :: numeric)
```


Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Xin lưu ý rằng điều kiện trên ID2 không thể thu hẹp đường truyền nút lá vì chỉ mục có cột ID1 trước ID2. Điều đó có nghĩa là, Quét chỉ mục sẽ quét toàn bộ phạm vi cho điều kiện SECTION = 1 :: số và áp dụng bộ lọc ID2 = ($ 1) :: số trên mỗi hàng đáp ứng mệnh đề
trên PHẦN.

Machine Translated by Google

Máy chủ SQL

Phương pháp được mô tả trong phần này áp dụng cho SQL Server Management Studio 2005 trở lên.

Lên kế hoạch thực thi

Với SQL Server, có một số cách để tìm nạp một kế hoạch thực thi. Hai phương pháp quan trọng nhất là:

Đồ họa

Biểu diễn đồ họa của các kế hoạch thực thi SQL Server có thể dễ dàng truy cập trong Management Studio nhưng khó chia sẻ vì thông tin vị từ chỉ hiển thị khi con chuột được di chuyển qua hoạt động cụ thể ("di chuột").

Bảng

Kế hoạch thực thi dạng bảng khó đọc nhưng dễ sao chép vì nó hiển thị tất cả thông tin liên quan cùng một lúc.

Đồ họa

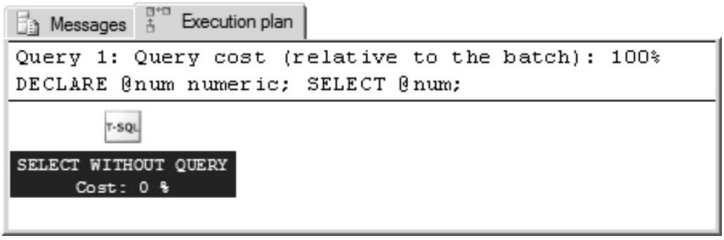
Kế hoạch giải thích bằng đồ họa được tạo bằng một trong hai nút được đánh dấu bên dưới.

Nút bên trái giải thích trực tiếp câu lệnh được đánh dấu. Quyền sẽ nắm bắt kế hoạch vào lần tiếp theo một câu lệnh SQL được thực thi.

Trong cả hai trường hợp, biểu diễn đồ họa của kế hoạch thực hiện sẽ xuất hiện trong tab "Kế hoạch thực hiện" của ngăn "Kết quả".

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google



Biểu diễn đồ họa rất dễ đọc với một chút thực hành.
Tuy nhiên, nó chỉ hiển thị thông tin cơ bản nhất: các hoạt động và bảng hoặc chỉ mục mà chúng hoạt động.

Management Studio hiển thị thêm thông tin khi di chuyển chuột qua một thao tác (mouseover / hover). Điều này làm cho việc chia sẻ một kế hoạch thực hiện với tất cả các chi tiết của nó trở nên khó khăn.

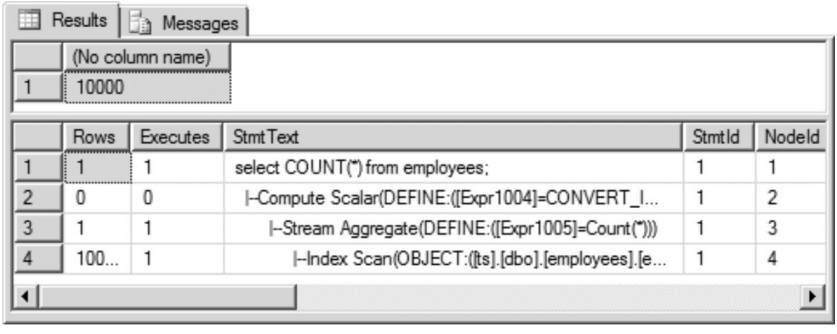
Bảng

Biểu diễn dạng bảng của kế hoạch thực thi SQL Server được tìm nạp bằng cách lập hồ sơ thực thi một câu lệnh. Lệnh sau cho phép nó:

```
ĐẶT HỒ SƠ THỐNG KÊ BẬT
```

Sau khi được kích hoạt, mỗi câu lệnh được thực thi sẽ tạo ra một tập hợp kết quả bổ sung. ví dụ: select các câu lệnh tạo ra hai tập kết quả – trước tiên là kết quả của câu lệnh sau đó là kế hoạch thực thi.

Kế hoạch thực thi dạng bảng khó có thể sử dụng được trong SQL Server Management Studio vì StmtText quá rộng để vừa trên màn hình.



Machine Translated by Google

Ưu điểm của cách biểu diễn này là nó có thể được sao chép mà không làm mất thông tin liên quan. Điều này rất hữu ích nếu bạn muốn đăng kế hoạch thực thi SQL Server trên một diễn đàn hoặc nền tảng tương tự. Trong trường hợp này, thường là đủ để sao chép cột StmtText và định dạng lại nó một chút:

```
chọn COUNT (*) từ nhân viên; | --Compute
   vô hướng (DEFINE: ([Expr1004] = CONVERT_IMPLICIT (...))
          | --Stream Aggregate (DEFINE: ([Expr1005] = Count (*)))
                 | --Index Scan (ĐỐI TƯỢNG: ([nhân viên]. [Nhân viên_pk]))
```

Cuối cùng, bạn có thể tắt cấu hình một lần nữa:

```
TẮT HỒ SƠ THỐNG KÊ
```

Hoạt động

Chỉ mục và Quyền truy cập Bảng

SQL Server có một thuật ngữ đơn giản: Các thao tác "Quét" đọc toàn bộ chỉ mục hoặc bảng trong khi các thao tác "Tìm kiếm" sử dụng cây B hoặc một địa chỉ vật lý (RID, như Oracle ROWID) để truy cập một phần cụ thể của chỉ mục hoặc bảng.

Tìm kiếm chỉ mục

Index Seek thực hiện duyệt cây B và đi qua các nút lá để tìm tất cả các mục nhập phù hợp. Xem thêm "Giải phẫu một chỉ mục" trên trang 1.

Quét chỉ mục

Đọc toàn bộ chỉ mục — tất cả các hàng — theo thứ tự chỉ mục. Tùy thuộc vào các số liệu thống kê khác nhau của hệ thống, cơ sở dữ liệu có thể thực hiện thao tác này nếu nó cần tất cả các hàng theo thứ tự chỉ mục — ví dụ: vì thứ tự theo mệnh đề tương ứng.

Tra cứu khóa (Nhóm)

Lấy một hàng từ một chỉ mục được phân nhóm. Điều này tương tự như Oracle INDEX UNIQUE SCAN cho một Index-Organized-Table (IOT). Xem thêm "Dữ liệu phân nhóm" trên trang 111.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Tra cứu RID (Heap)

Lấy một hàng từ một bảng — như Oracle TABLE ACCESS BY INDEX ROWID. Xem thêm “Giải phẫu một chỉ mục” trên trang 1.

Quét bảng

Đây còn được gọi là quét toàn bộ bảng. Đọc toàn bộ bảng — tất cả các hàng và cột — như được lưu trữ trên đĩa. Mặc dù các thao tác đọc đa khối có thể cải thiện đáng kể tốc độ của Quét bảng , nhưng đây vẫn là một trong những thao tác tốn kém nhất. Bên cạnh tỷ lệ IO cao, quét bảng cũng phải kiểm tra tất cả các hàng của bảng nên nó cũng có thể tiêu tốn một lượng thời gian CPU đáng kể. Xem thêm “Quét toàn bảng” trên trang 13.

Tham gia hoạt động

Nói chung, quy trình hoạt động tham gia chỉ xử lý hai bảng cùng một lúc. Trong trường hợp một truy vấn có nhiều liên kết hơn, chúng được thực hiện tuần tự: hai bảng đầu tiên, sau đó là kết quả trung gian với bảng tiếp theo. Trong ngữ cảnh của phép nối, thuật ngữ “bảng” do đó cũng có thể có nghĩa là “kết quả trung gian”.

Vòng lồng nhau

Nối hai bảng bằng cách tìm nạp kết quả từ một bảng và truy vấn bảng khác cho mỗi hàng từ bảng đầu tiên. SQL Server cũng sử dụng thao tác vòng lặp lồng nhau để truy xuất dữ liệu bảng sau khi truy cập chỉ mục. Xem thêm “Các vòng lặp lồng nhau” trên trang 92.

Kết hợp băm

Phép so khớp băm tải các bản ghi ứng viên từ một phía của phép nối vào một bảng băm, sau đó được thăm dò cho từng hàng từ phía bên kia của phép nối. Xem thêm “Tham gia băm” trên trang 101.

Kết hợp Tham

gia Kết hợp kết hợp hai danh sách được sắp xếp giống như một dây kéo. Cả hai mặt của phép nối phải được sắp xếp trước. Xem thêm “Sắp xếp Hợp nhất” trên trang 109.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Sắp xếp và phân nhóm

Sắp

xếp Sắp xếp kết quả theo thứ tự theo mệnh đề. Thao tác này cần một lượng lớn bộ nhớ để hiện thực hóa kết quả trung gian (không phải là pipelined). Xem thêm "Thứ tự lập chỉ mục theo" trên trang 130.

Sắp xếp (Sắp xếp N hàng đầu)

Sắp xếp một tập hợp con của kết quả theo thứ tự của mệnh đề. Được sử dụng cho các truy vấn top-N nếu không thể thực thi pipelined. Xem thêm "Truy vấn các hàng Top-N" trên trang 143.

Tổng hợp luồng

Tổng hợp một tập hợp được sắp xếp trước theo nhóm theo mệnh đề. Thao tác này không đệm cho kết quả trung gian — nó được thực hiện theo cách pipelined. Xem thêm "Nhóm lập chỉ mục theo" trên trang 139.

Kết hợp băm (Tổng hợp)

Nhóm kết quả bằng bảng băm. Thao tác này cần một lượng lớn bộ nhớ để hiện thực hóa kết quả trung gian (không phải là pipelined). Đầu ra không được sắp xếp theo bất kỳ cách nào có ý nghĩa. Xem thêm "Nhóm lập chỉ mục theo" trên trang 139.

Truy vấn Top-N

Top

Hủy bỏ các thao tác cơ bản khi số lượng hàng mong muốn đã được tìm nạp. Xem thêm "Truy vấn các hàng Top-N" trên trang 143.

Hiệu quả của truy vấn top-N phụ thuộc vào chế độ thực thi của các hoạt động bên dưới. Nó rất kém hiệu quả khi hủy bỏ các hoạt động không theo chuỗi chẳng hạn như Sắp xếp.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Phân biệt Quyền truy cập và Dự đoán bộ lọc

Cơ sở dữ liệu SQL Server sử dụng ba phương pháp khác nhau để áp dụng mệnh đề where (vị từ):

Truy cập vị từ (“Tìm kiếm vị ngữ”)
: Các vị từ truy cập thể hiện điều kiện bắt đầu và dừng của việc truyền tải nút lá.

Dự đoán bộ lọc chỉ mục (“Dự đoán” hoặc “vị trí” cho các hoạt động chỉ mục)
: Các vị từ bộ lọc chỉ mục chỉ được áp dụng trong quá trình duyệt nút lá. Chúng không đóng góp vào điều kiện bắt đầu và dừng và không thu hẹp phạm vi quét.

Vị từ bộ lọc mức bảng (“ở đâu” cho các thao tác trên bảng)
: Các dự đoán trên các cột không phải là một phần của chỉ mục được đánh giá ở cấp bảng. Để điều đó xảy ra, cơ sở dữ liệu phải tải hàng từ bảng heap trước.

Phần sau giải thích cách xác định các vị từ bộ lọc trong kế hoạch thực thi SQL Server. Nó dựa trên mẫu được sử dụng để chứng minh tác động của các vị từ bộ lọc chỉ mục trong Chương 3.

```
TẠO TABLE scale_data (
    phần NUMERIC NOT NULL, id1
                SỐ KHÔNG ĐẦY ĐỦ, id2
                SỐ KHÔNG ĐẦY ĐỦ
);

TẠO INDEX scale_slow ON scale_data (section, id1, id2);
```

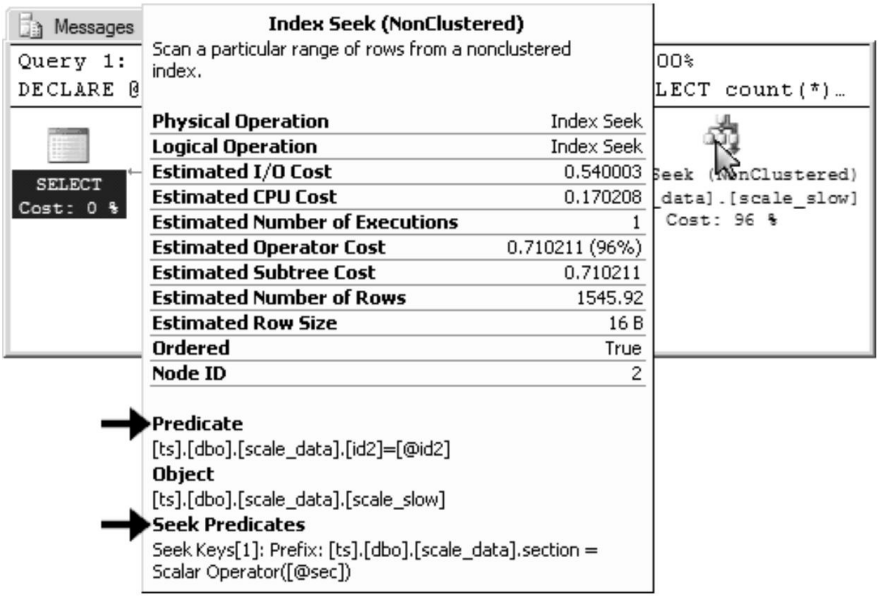
Câu lệnh mẫu chọn theo SECTION và ID2:

```
CHỌN số lượng (*)
   FROM scale_data
  Phần WHERE = @sec
     VÀ id2 = @ id2
```

Machine Translated by Google

Trong kế hoạch thực thi đồ họa

Kế hoạch thực thi đồ họa ẩn thông tin vị từ trong chú giải công cụ chỉ được hiển thị khi di chuyển chuột qua thao tác Tìm kiếm chỉ mục .



Các dự đoán tìm kiếm của SQL Server tương ứng với các vị từ truy cập của Oracle— chúng thu hẹp khả năng truyền tải nút lá. Vị ngữ bộ lọc chỉ được gắn nhãn Dự đoán trong kế hoạch thực thi đồ họa của SQL Server.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Trong kế hoạch thực thi dạng bảng

Các kế hoạch thực thi dạng bảng có thông tin vị từ trong cùng một cột mà các hoạt động xuất hiện. Do đó, rất dễ dàng sao chép và dán tất cả các thông tin liên quan chỉ trong một lần.

```
DECLARE @sec số;
DECLARE @ id2 số;

ĐẶT HỒ SƠ THỐNG KÊ BẬT


CHỌN số lượng (*)
   FROM scale_data
 Phần WHERE = @sec
    VÀ id2 = @ id2

TẮT HỒ SƠ THỐNG KÊ
```

Kế hoạch thực hiện được hiển thị dưới dạng tập hợp kết quả thứ hai trong ngăn kết quả. Sau đây là cột StmtText – với một chút định dạng lại để đọc tốt hơn:

```
| --Compute vô hướng (DEFINE: ([Expr1004] = CONVERT_IMPLICIT (...))
       | --Stream Aggregate (DEFINE: ([Expr1008] = Count (*)))
              | --Index Seek (OBJECT: ([scale_data]. [Scale_slow]), SEEK:
                  ([scale_data]. [Section] = [@ giây])
                            ĐÃ ĐẶT HÀNG
                  WHERE: ([scale_data]. [Id2] = [@ id2]))
```

Nhãn SEEK giới thiệu các vị từ truy cập, nhãn WHERE đánh dấu các vị từ bộ lọc.

Machine Translated by Google

MySQL

Phương pháp được mô tả trong phần này áp dụng cho tất cả các phiên bản của MySQL.

Lên kế hoạch thực thi

Đặt giải thích trước câu lệnh SQL để truy xuất kế hoạch thực thi.

```
GIẢI THÍCH CHỌN 1;
```

Kế hoạch được hiển thị dưới dạng bảng (đã loại bỏ một số cột ít quan trọng hơn):

```
~ + ------- + ------ + --------------- + ------ + ~ + ------ + - ----------- ~
~ | bảng | loại | khóa_có_thể | phím | ~ | hàng | Thêm
~ + ------- + ------ + --------------- + ------ + ~ + ------ + - ----------- ~
~ | NULL | NULL | VÔ GIÁ TRỊ              | NULL | ~ | NULL | Không có bảng ...
~ + ------- + ------ + --------------- + ------ + ~ + ------ + - ----------- ~
```

Thông tin quan trọng nhất nằm trong cột TYPE . Mặc dù tài liệu MySQL gọi nó là “kiểu tham gia”, tôi thích mô tả nó là “kiểu truy cập” vì nó thực sự chỉ định cách dữ liệu được truy cập. Ý nghĩa của giá trị kiểu được mô tả trong phần tiếp theo.

Hoạt động

Chỉ mục và Quyền truy cập Bảng

Kế hoạch giải thích của MySQL có xu hướng mang lại cảm giác an toàn sai lầm vì nó nói rất nhiều về các chỉ mục đang được sử dụng. Mặc dù đúng về mặt kỹ thuật, nhưng điều đó không có nghĩa là nó đang sử dụng chỉ mục một cách hiệu quả. Thông tin quan trọng nhất nằm trong cột TYPE của đầu ra giải thích của MySQL — nhưng ngay cả ở đó, từ khóa INDEX không cho biết lập chỉ mục thích hợp.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

eq_ref

Chỉ thực hiện truyền ngang cây B. Cơ sở dữ liệu sử dụng thao tác này nếu khóa chính hoặc ràng buộc duy nhất đảm bảo rằng tiêu chí tìm kiếm sẽ không khớp với nhiều hơn một mục nhập. Xem thêm “Giải phẫu một chỉ mục” trên trang 1.

giới hạn trọng tải

Thực hiện duyệt cây B và đi qua các nút lá để tìm tất cả các mục nhập phù hợp (tương tự như QUÉT NGUỒN CHỈ SỐ). Xem thêm “Giải phẫu một chỉ mục” trên trang 1.

mục lục

Đọc toàn bộ chỉ mục — tất cả các hàng — theo thứ tự chỉ mục (tương tự như INDEX FULL SCAN).

TẤT CẢ CÁC

Đọc toàn bộ bảng — tất cả các hàng và cột — như được lưu trữ trên đĩa.

Bên cạnh tỷ lệ IO cao, quá trình quét bảng cũng phải kiểm tra tất cả các hàng từ bảng để nó cũng có thể đặt một tải đáng kể lên CPU. Xem thêm “Quét toàn bảng” trên trang 13.

Sử dụng Chỉ mục (trong cột "Thêm")

Khi cột “Thêm” hiển thị “Sử dụng chỉ mục”, điều đó có nghĩa là bảng không được truy cập vì chỉ mục có tất cả dữ liệu bắt buộc. Hãy nghĩ đến việc “CHỈ sử dụng chỉ mục”. Xem thêm “Dữ liệu phân nhóm” trên trang 111.

CHÍNH (trong cột “khóa” hoặc “khóa có thể”)

PRIMARY là tên của chỉ mục được tạo tự động cho khóa chính.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Sắp xếp và phân nhóm

using filesort (trong cột "Extra") "using filesort" trong cột Extra chỉ ra một thao tác sắp xếp rõ ràng — bất kể việc sắp xếp diễn ra ở đâu (bộ nhớ chính hay trên đĩa). “Sử dụng filesort” cần một lượng lớn bộ nhớ để hiện thực hóa kết quả trung gian (không phải là pipelined). Xem thêm “Thứ tự lập chỉ mục theo” trên trang 130.

Truy vấn Top-N

ngầm định: không có "using filesort" trong cột "Extra"

Kế hoạch thực thi MySQL không hiển thị truy vấn top-N một cách rõ ràng. Nếu bạn đang sử dụng cú pháp giới hạn và không thấy "using filesort" trong cột bổ sung, thì nó được thực thi theo cách pipelined. Xem thêm “Truy vấn N hàng trên cùng” trên trang 143.

Phân biệt Quyền truy cập và Dự đoán bộ lọc

Cơ sở dữ liệu MySQL sử dụng ba cách khác nhau để đánh giá mệnh đề (vị từ) ở đâu :

Truy cập vị từ (thông qua cột “key_len”)
Các vị từ truy cập thể hiện điều kiện bắt đầu và dừng của việc truyền tải nút lá.

Vị từ bộ lọc chỉ mục (“Sử dụng điều kiện chỉ mục”, kể từ MySQL 5.6)
Các vị từ bộ lọc chỉ mục chỉ được áp dụng trong quá trình duyệt nút lá. Chúng không đóng góp vào điều kiện bắt đầu và dừng và không thu hẹp phạm vi quét.

Vị từ bộ lọc cấp bảng (“Sử dụng ở đâu” trong cột “Thêm”)
Các dự đoán trên các cột không phải là một phần của chỉ mục được đánh giá ở cấp bảng. Để điều đó xảy ra, cơ sở dữ liệu phải tải hàng từ bảng trước.

Các kế hoạch thực thi MySQL không hiển thị loại vị từ nào được sử dụng cho mỗi điều kiện — chúng chỉ liệt kê các loại vị từ được sử dụng.

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Trong ví dụ sau, toàn bộ mệnh đề where được sử dụng làm vị từ truy cập:

```
TẠO BẢNG demo (
    id1 NUMERIC
  , id2 NUMERIC
  , id3 NUMERIC
  , val NUMERIC);

CHÈN VÀO GIÁ TRỊ demo (1,1,1,1);
CHÈN VÀO GIÁ TRỊ demo (2,2,2,2);

TẠO INDEX demo_idx
                BẬT bản trình diễn
                     (id1, id2, id3);

GIẢI THÍCH
 LỰA CHỌN *
    TỪ bản trình diễn
   WHERE id1 = 1
      VÀ id2 = 1;

+ ------ + ---------- + --------- + ------ + ------- +
| loại | Chìa khóa         | key_len | hàng | Thêm |
+ ------ + ---------- + --------- + ------ + ------- +
| giới thiệu | demo_idx | 12        |     1  |          |
+ ------ + ---------- + --------- + ------ + ------- +
```

Không có "Sử dụng ở đâu" hoặc "Sử dụng điều kiện chỉ mục" được hiển thị trong "Thêm" cột. Tuy nhiên, chỉ mục được sử dụng (type = ref, key = demo_idx) để bạn có thể giả sử rằng toàn bộ mệnh đề where đủ điều kiện làm vị từ truy cập.

Bạn có thể sử dụng giá trị key_len để xác minh điều này. Nó cho thấy rằng truy vấn sử dụng 12 byte đầu tiên của định nghĩa chỉ mục. Để ánh xạ điều này với tên cột, bạn "Chỉ" cần biết mỗi cột cần bao nhiêu dung lượng lưu trữ (xem "Dữ liệu Nhập Yêu cầu Lưu trữ ”trong tài liệu MySQL). Không có ràng buộc NOT NULL , MySQL cần thêm một byte cho mỗi cột. Sau tất cả, mỗi cột NUMERIC cần 6 byte trong ví dụ. Do đó, chìa khóa độ dài 12 xác nhận rằng hai cột chỉ mục đầu tiên được sử dụng làm quyền truy cập các vị ngữ.

Machine Translated by Google

Khi lọc với cột ID3 (thay vì ID2) MySQL 5.6 trở lên sử dụng một vị từ bộ lọc chỉ mục (“Sử dụng điều kiện chỉ mục”):

```
GIẢI THÍCH
 LỰA CHỌN *
    TỪ bản trình diễn
  WHERE id1 = 1
     VÀ id3 = 1;

+ ------ + ---------- + --------- + ------ + -------------- --------- +
| loại | Chìa khóa         | key_len | hàng | Thêm                                    |
+ ------ + ---------- + --------- + ------ + -------------- --------- +
| giới thiệu | demo_idx | 6             |      1  | Sử dụng điều kiện chỉ mục |
+ ------ + ---------- + --------- + ------ + -------------- --------- +
```

Trong trường hợp này, độ dài khóa là sáu có nghĩa là chỉ một cột được sử dụng làm quyền truy cập Thuộc tính.

Các phiên bản trước của MySQL đã sử dụng một vị từ bộ lọc cấp bảng cho điều này truy vấn – được xác định bằng cách “Sử dụng ở đâu” trong cột “Thêm”:

```
+ ------ + ---------- + --------- + ------ + ------------- +
| loại | Chìa khóa         | key_len | hàng | Thêm                      |
+ ------ + ---------- + --------- + ------ + ------------- +
| giới thiệu | demo_idx | 6             |      1  | Sử dụng ở đâu |
+ ------ + ---------- + --------- + ------ + ------------- +
```


Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

Mục lục

Các ký hiệu 2PC,
89 ?: Var, @var
(xem tham số ràng buộc)

Một

Vị ngữ truy cập, 44 vị
từ truy cập nhận dạng
trong các kế hoạch thực thi
Oracle, 170
PostgreSQL, 177
SQL Server, 185
chia sẻ con trỏ thích ứng (Oracle), 75 tham
số tự động (SQL Server), 39

B

Cây B (cây tìm kiếm cân bằng), 4 giữa,
44
tham số ràng buộc, 32
chống chỉ định
biểu đồ, 34
Bộ lọc LIKE , 47
phân vùng, 35 cho
bộ nhớ đệm kế hoạch thực thi, 32 kiểu
an toàn, 66 liên kết nhìn trộm
(Oracle), 75 chỉ mục bitmap, 50

Quét chỉ mục bitmap (PostgreSQL), 175
Định lý CAP của Brewer, 89

C

Định lý CAP, 89 ước
lượng thẻ số, 27
CBO (xem trình tối ưu hóa, dựa trên chi
phí) chỉ mục nhóm, 122 chuyển đổi thành
bảng heap SQL Server, 127

yếu tố phân cụm, 21, 114 được
tối ưu hóa tự động, 133 khóa phân
cụm, 123 đối chiếu, 24 ràng buộc cam kết
có thể hoãn lại, 11 ngầm định cho bảng
cắt ngắn, 163 hai pha, 89 biên dịch, 18
cột được tính (SQL Server), 27 ràng
buộc có thể hoãn lại, 11

NOT NULL, giá
trị chi phí 56 , 18

đếm(*)
thường là quét chỉ theo chỉ mục, 120
Oracle yêu cầu ràng buộc NOT NULL , 57
COUNT STOPKEY, 145 chia
sẻ con trỏ (Oracle), 39

Đối tượng vận chuyển dữ liệu D (DTO), 105
NGÀY
làm việc hiệu quả với, 62
DBMS_XPLAN, 167
DEALLOCATE, 174
Ràng buộc DEFERRABLE , 11
DETERMINISTIC (Oracle), 30 riêng
biệt, 97 riêng biệt () trong JPA
và Hibernate, 97

DML, danh
sách liên kết kép 159 ,
2 cập nhật động (Hibernate), 164

E

háo hức tìm nạp, 96
tính nhất quán cuối cùng, 89
kế hoạch thực thi, 10, 165 bộ
nhớ cache, 32, 75 tạo

MySQL, 188
Oracle, 166
PostgreSQL, 172
SQL Server, 180
hoạt động
MySQL, 188
Oracle, 167
PostgreSQL, 174
SQL Server, 182
giải thích
MySQL, 188
Oracle, 166
PostgreSQL, 172

F

FBI (xem chỉ mục, dựa trên chức năng)
TÌM KIẾM TẤT CẢ CÁC THUỘC TÍNH (HQL),
105 lần tìm nạp đầu tiên, 144 hiệu ứng
vị ngữ bộ lọc (biểu đồ), 81 nhận dạng
trong kế hoạch thực thi

Oracle, 170
PostgreSQL, 177
Máy chủ SQL, 185

Machine Translated by Google

quét toàn bộ bảng, 13
Tất cả (MySQL), 189
Quét Seq (PostgreSQL), 174
TRUY CẬP BẢNG ĐẦY ĐỦ (Oracle), 168
Quét bảng (SQL Server), 183 chức năng, 24 xác định, 29 trong chỉ mục một phần, 52 cửa sổ, 156

G

nhóm theo, 139
với PostrgesSQL và cơ sở dữ liệu Oracle và một chỉ mục ASC / DESC không được liên kết, 140

H

tham gia băm, 101
HASH GROUP BY, 169
HASH JOIN (Oracle), 169
Tham gia HASH (PostgreSQL), 175
Kết hợp băm, 183
Kết hợp băm (Tổng hợp), 184 bảng
heap, 3, 122 tạo trong SQL Server, 127
Tìm nạp háo
hức ngủ đông, 96
ILIKE sử dụng LOWER, 98
cập nhật tất cả các cột, 164
gợi ý, 19

I

IMMUTABLE (PostgreSQL), 30 chỉ mục bao phủ, 117 văn bản đầy đủ, 48
dựa trên hàm, 24 phân biệt chữ hoa chữ thường, 24 phép tính
toán học lập chỉ mục, 77 phép nối, 50 giới hạn
MySQL, Oracle, PostgreSQL, 121
SQL Server, 122
hợp nhất, 49 nhiều cột, 12 thứ tự sai (hiệu ứng), 81 một phần, 51 tiền tố
(MySQL), 121 thứ cấp, 123 chỉ mục
trong kế hoạch thực thi MySQL, 189
chỉ quét chỉ mục, 116 bảng được tổ chức theo chỉ mục, 122 hỗ trợ cơ sở dữ liệu, 127
Chỉ số Cond (PostgreSQL), 177
INDEX FAST FULL SCAN, 168
INDEX FULL SCAN, 168
Chỉ quét chỉ mục (PostgreSQL), 174
QUÉT NGUỒN CHỈ SỐ, 167
Quét chỉ mục (PostgreSQL), 174
Index Seek, 182
CHỈ SỐ QUÉT DUY NHẤT, 167 khi truy cập IOT, 124
INTERNAL_FUNCTION, 67
IOT (bảng được sắp xếp theo chỉ mục), 122

J

tham gia, 91 bên ngoài đầy đủ, 109

K

Tra cứu khóa (Cụm), 182
Tìm nạp lười
biếng cho các thuộc tính vô hướng (cột), 104
nút lá, chia 2 , 160
LIKE, 45
lựa chọn thay thế, 48 làm vị từ bộ lọc chỉ mục, 112
trên cột DATE, 67 trên cột DATE , 67 giới hạn (MySQL, PostgreSQL), 144 khả năng mở rộng logarit, 7
LOWER, 24

M

Hợp nhất Tham gia, 109
PostgreSQL, 175
Máy chủ SQL, 183
MERGE JOIN (Oracle), 169 lần đọc nhiều khối để quét toàn bộ bảng, 13 cho INDEX FAST FULL SCAN, 168
MVCC, 163 ảnh
hưởng đến quá trình quét chỉ chỉ mục PostgreSQL, 174 huyền thoại SQL động chậm, 72, 74 cột chọn lọc đầu tiên không chọn lọc nhất, 43 nguồn gốc, 49
Oracle không thể lập chỉ mục NULL, 56

N

N + 1 vấn đề, 92
Vòng lặp lồng nhau, 92
PostgreSQL, 175
Máy chủ SQL, 183
NESTED LOOPS (Oracle), 168

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

NOSORT
SẮP XẾP NHÓM THEO, 140
WINDOW, 157
VÔ GIÁ TRỊ
lập chỉ mục trong Oracle, 54

O

offset (MySQL, PostgreSQL), 148 trình tối ưu hóa, 18 dựa trên chi phí, 18 gợi ý, 19
dựa trên quy tắc, 18 thống kê, 21

TỐI ƯU CHO (SQL Server), 76
TÙY CHỌN (Máy chủ SQL), 76
HOẶC
để tắt bộ lọc, 72 sắp xếp theo, 130
ASC, DESC, 134
NULLS ĐẦU / CUỐI, ma trận
hỗ trợ 137 , 138
HẾT (), 156

Phân trang P ,
147 phương pháp bù đắp,
148 phương thức tìm
kiếm, 149 xấp xỉ, 152
đánh giá tham số (SQL Server), 76 phân tích cú pháp, 18 chỉ mục một phần, 51 đối tượng một phần (ORM), 104 phân vùng và tham số liên kết, 35 pipelining, 92 PLAN_TABLE, 166 vị từ thông tin, 20 vị từ truy cập so với bộ lọc, 44 trong kế hoạch thực thi MySQL, 190 Oracle, 170 SQL Server, 185 chuẩn bị (PostgreSQL), 172 khóa chính với chỉ mục duy nhất, 11

Công cụ lập kế hoạch truy vấn Q (xem trình tối ưu hóa)

R

RBO (xem trình tối ưu hóa, dựa trên quy tắc)
RECOMPILE (gợi ý SQL Server), biến áp đặt 76 kết quả, 98
RID, 3
Tra cứu RID (Heap), 183 nút gốc, 5 lần tách, 160
trình tự hàng, 113 giá trị hàng, 153
ROWID, 3
ROWNUM (cột giả Oracle), 144, 148
ROW_NUMBER, 156

Khả năng mở rộng
S , 81 ngang, 87
logarit, 7
Khả năng mở rộng, 79
Tìm kiếm dự đoán (SQL Server), 185 chọn *, tránh để bật quét chỉ chỉ mục, 120 cải
thiện hiệu suất tham gia băm, 104

Quét Seq, 174
Sắp xếp (SQL Server), 184
SẮP XẾP NHÓM THEO, 169
NOSORT, 140
SẮP XẾP ĐẶT HÀNG THEO, 130
STOPKEY, 145
Vùng SQL, 75
SQL injection, 32
SSD (Solid State Disk), 90 thống kê, 21 cho các chỉ mục dựa trên
chức năng Oracle, 28
HỒ SƠ THỐNG KÊ, 181
DỪNG LẠI
COUNT, 145
SẮP XẾP ĐẶT HÀNG THEO, 146
WINDOW, 157
Tổng hợp dòng, 184

T

đầu (SQL Server), 145
Truy vấn Top-N, 143
TO_CHAR (DATE), 66
TRUNC (DATE), 62
bảng cắt ngắn, 163 trình
kích hoạt không được thực thi, 163

U

LÊN, 24

V

Vaccum (PostgreSQL), 163 cột ảo cho các ràng buộc KHÔNG ĐỦ đối với FBI, 58

W

where, 9
có điều kiện, 72
trong kế hoạch thực thi SQL Server, 187
hàm cửa sổ, 156

Machine Translated by Google

Giải thích về hiệu suất SQL! - Giờ thì sao?

Có thể bạn vẫn còn một số câu hỏi hoặc một vấn đề rất cụ thể mà "SQL Performance Explained" không trả lời thỏa đáng? Huấn luyện tức thì là giải pháp cho bạn.

Huấn luyện tức thì

Huấn luyện tức thì là cách nhanh chóng, dễ dàng và không phức tạp cho các nhà phát triển để giải quyết các vấn đề khó khăn về hiệu suất cơ sở dữ liệu SQL thông qua chia sẻ máy tính để bàn.

Huấn luyện tức thì không sử dụng các ví dụ được tạo sẵn; các trường hợp thực tế được trình bày vì đây là cách tối ưu để giúp bạn giải quyết các vấn đề hiện tại với cơ sở dữ liệu SQL chậm. Huấn luyện tức thì độc lập với nhà cung cấp và hiệu quả. và được cung cấp ở một mức giá rất phải chăng!

Hãy thử một lần!

Bây giờ là lúc để tìm hiểu thêm về Huấn luyện tức thì! Đến thăm trang web của chúng tôi để biết thêm thông tin:

http://winand.at

Ex Libris GHEORGHE GABRIEL SICHIM <gabriel.sichim@gmail.com>

Machine Translated by Google

SQL Performance giải thích SQL Performance giúp các nhà phát triển cải thiện hiệu suất cơ sở dữ liệu cho mance. Trọng tâm là SQL – nó bao gồm tất cả các cơ sở dữ liệu SQL chính mà không bị lạc vào chi tiết của bất kỳ sản phẩm cụ thể nào.

Bắt đầu với những điều cơ bản về lập chỉ mục và mệnh đề where , Hiệu suất SQL được giải thích hướng dẫn các nhà phát triển thông qua tất cả các phần của một câu lệnh SQL và giải thích các cạm bẫy của các công cụ ánh xạ quan hệ đối tượng (orm) như Hibernate.

Các chủ đề được đề cập bao gồm:

- Sử dụng chỉ mục nhiều cột
- áp dụng đúng các hàm SQL
- sử dụng đủ các truy vấn LIKE
- tối ưu hóa hoạt động tham gia
- phân nhóm dữ liệu để cải thiện hiệu suất
- Thực hiện liên kết theo thứ tự và nhóm theo
- Đạt được hiệu suất tốt nhất cho các truy vấn phân trang
- Hiểu khả năng mở rộng của cơ sở dữ liệu

Cấu trúc có hệ thống của nó khiến SQL Performance được giải thích vừa là sách giáo khoa vừa là sách hướng dẫn tham khảo nên có trên giá sách của mọi nhà phát triển.

bìa

✔ cơ sở dữ liệu Oracle ✔ Máy chủ SQL ✔ mySQL ✔ PostgreSQL

về markus Winand

markus Winand đã phát triển các ứng dụng SQL từ năm 1998. Các mối quan tâm chính của ông bao gồm hiệu suất, khả năng mở rộng, độ tin cậy và nói chung là tất cả các khía cạnh kỹ thuật khác của chất lượng phần mềm. markus hiện đang làm việc với tư cách là một huấn luyện viên và huấn luyện viên độc lập tại Vienna, Áo. http://winand.at/

ISbN 978-3-9503078-2-5

eUr 29,95
GbP 26,99

783950 307825 >