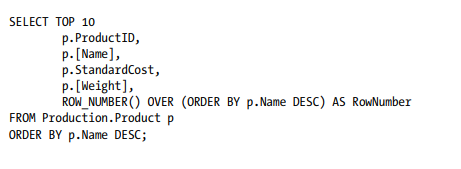
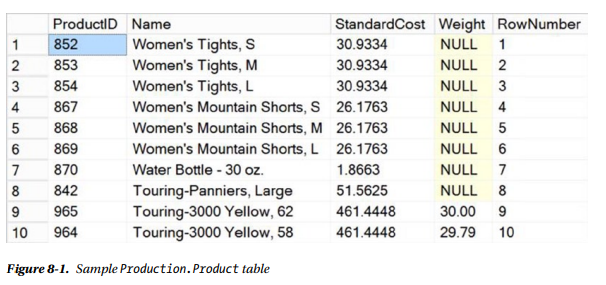
Đúng index ở đúng cột hoặc nhiều cột là cơ sở để bắt đầu điều chỉnh truy vấn. Một index bị thiếu hoặc một index được đặt sai ở một cột hay nhiều cột, có thể là cơ sở cho tất cả các vấn đề về hiệu suất bắt đầu với việc truy cập dữ liệu cơ bản, tiếp tục thông qua các phép Join và kết thúc bằng các mệnh đề lọc. Vì những lý do này, điều cực kỳ quan trọng đối với mọi người, không chỉ là một DBA, để hiểu các kỹ thuật lập chỉ mục khác nhau có thể được sử dụng để tối ưu hóa thiết kế cơ sở dữ liệu

**What Is an Index?**

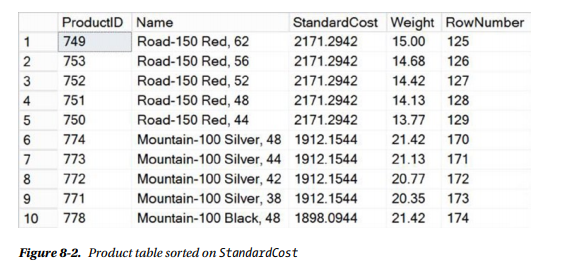
Một trong những cách tốt nhất để giảm I/O đĩa là sử dụng một index. Một index cho phép SQL Server tìm dữ liệu trong một bảng mà không cần quét toàn bộ bảng. Một index trong cơ sở dữ liệu tương tự như một chỉ mục trong một cuốn sách. Ví dụ, nói rằng bạn muốn tra cứu *table scan* trong cuốn sách này. Trong phiên bản giấy, không có index ở mặt sau của cuốn sách, bạn sẽ phải xem lại toàn bộ cuốn sách để tìm văn bản bạn cần. Với index, bạn biết chính xác nơi thông tin bạn muốn được lưu trữ.

Trong khi điều chỉnh cơ sở dữ liệu về hiệu suất, bạn tạo các index trên các cột khác nhau được sử dụng trong truy vấn để giúp SQL Server tìm dữ liệu nhanh chóng. Ví dụ: truy vấn sau đối với bảng Production. Product dẫn đến dữ liệu được hiển thị trong Hình 8-1 (10 đầu tiên trong số hơn 500 hàng)

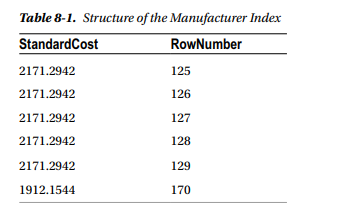
Truy vấn trước đó đã quét toàn bộ bảng vì không có mệnh đề WHERE. Nếu bạn cần thêm bộ lọc thông qua mệnh đề WHERE để truy xuất tất cả các sản phẩm có StandardCost lớn hơn 150, không có index, bảng sẽ vẫn phải được quét, kiểm tra giá trị của StandardCost ở mỗi hàng để xác định hàng nào chứa giá trị lớn hơn 150. Một index trên cột StandardCost có thể tăng tốc quá trình này bằng cách cung cấp một cơ chế cho phép tìm kiếm có cấu trúc dựa trên dữ liệu thay vì kiểm tra theo từng hàng. Bạn có thể thực hiện hai cách tiếp cận khác nhau và cơ bản để tạo index này:

- Giống như một từ điển (Like a dictionary): Một từ điển là một danh sách riêng biệt của các từ theo thứ tự bảng chữ cái. Một index có thể được lưu trữ theo cách tương tự. Dữ liệu được sắp xếp, mặc dù nó vẫn sẽ có các trùng lặp. Mười hàng đầu tiên, được sắp xếp theo StandardCost DESC thay vì theo Name, sẽ trông giống như dữ liệu được hiển thị trong Hình 8-2. Lưu ý cột RowNumber hiển thị vị trí ban đầu của hàng khi order theo Name.



Vì vậy, bây giờ nếu bạn muốn tìm tất cả dữ liệu trong các hàng có StandardCost lớn hơn 150, index sẽ cho phép bạn tìm thấy chúng ngay lập tức bằng cách chuyển xuống giá trị đầu tiên lớn hơn 150. Một index áp dụng thứ tự cho dữ liệu được lưu trữ dựa trên thứ tự key index được gọi là clustered index. Do cách SQL Server lưu trữ dữ liệu, đây là một trong những index quan trọng nhất trong thiết kế cơ sở dữ liệu của bạn. Tôi giải thích điều này chi tiết sau trong chương này.

- Giống như kiến trúc Index của một cuốn sách (Like a book’s Index architecture): Một ordered list có thể được tạo mà không làm thay đổi bố cục của bảng. Tương tự như cách tạo index của một cuốn sách. Giống như keyword index của một cuốn sách liệt kê các keyword trong một phần riêng biệt với số trang để tham khảo nội dung chính của cuốn sách, danh sách các giá trị StandardCost được tạo thành một cấu trúc riêng biệt và tham chiếu đến hàng tương ứng trong bảng Product thông qua một con trỏ. Ví dụ, tôi sẽ sử dụng RowNumber làm con trỏ. Bảng 8-1 cho thấy cấu trúc của manufacturer index.



SQL Server có thể quét manufacturer index để tìm các hàng trong đó StandardCost lớn hơn 150. Vì các giá trị StandardCost được sắp xếp theo sorted order, SQL Server có thể dừng quét ngay khi gặp hàng có giá trị từ 150 trở xuống. Loại chỉ mục này được gọi là nonclustered index và tôi sẽ giải thích chi tiết về nó sau trong chương này.

Trong cả hai trường hợp, SQL Server sẽ có thể tìm thấy tất cả các sản phẩm trong đó StandardCost lớn hơn 150 nhanh hơn so với không có index trong hầu hết các trường hợp.

Bạn có thể tạo các index trên một cột đơn (như được mô tả trước đây) hoặc kết hợp các cột trong một bảng. SQL Server tự động tạo các chỉ mục cho các loại ràng buộc nhất định (ví dụ: các ràng buộc PRIMARY KEY và UNIQUE).

**The Benefit of Indexes**

SQL Server phải có khả năng tìm dữ liệu, ngay cả khi không có index nào xuất hiện trên bảng. Khi không có clustered index nào xuất hiện để thiết lập một thứ tự lưu trữ cho dữ liệu, công cụ lưu trữ sẽ dơn giản đọc qua toàn bộ bảng để tìm những gì nó cần.

Một bảng không có clustered index được gọi là bảng heap. Một heap là một chồng dữ liệu không có thứ tự với một hàng định danh (row identifier) làm con trỏ đến vị trí lưu trữ. Dữ liệu này không được sắp xếp hoặc tìm kiếm ngoại trừ bằng cách đi qua dữ liệu, từng hàng, trong một quy trình được gọi là quét (scan).

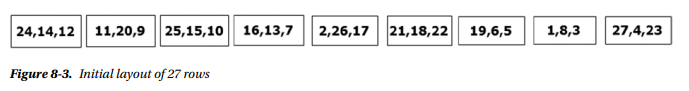
Khi một clustered index được đặt trên một bảng, các giá trị chính của index sẽ thiết lập một thứ tự cho dữ liệu. Hơn nữa, với một clustered index, dữ liệu được lưu trữ với index để bây giờ dữ liệu được sắp xếp. Khi có một clustered index, con trỏ trên nonclustered index bao gồm các giá trị xác định clustered index key. Đây là một phần quan trọng làm cho các clustered index rất quan trọng.

Dữ liệu trong SQL Server được lưu trữ trên một page có kích thước 8KB. Một page là lượng thông tin tối thiểu di chuyển khỏi đĩa và vào bộ nhớ, do đó, số lượng bạn có thể lưu trữ trên một trang trở nên quan trọng. Vì một trang có số lượng không gian hạn chế, nó có thể lưu trữ số lượng row lớn hơn nếu các row chứa số lượng cột ít hơn hoặc các cột có kích thước nhỏ hơn. Nonclustered index thường không có (và không nên) chứa tất cả các cột của bảng; nó thường chỉ chứa một số lượng hạn chế của các cột. Do đó, một trang sẽ có thể lưu trữ nhiều hàng của một nonclustered index hơn các hàng của chính table đó mà chứa tất cả các cột. Do đó, SQL Server sẽ có thể đọc nhiều giá trị hơn cho một cột từ một trang đại diện cho một nonclustered index trên cột so với từ một trang đại diện cho bảng có chứa cột.

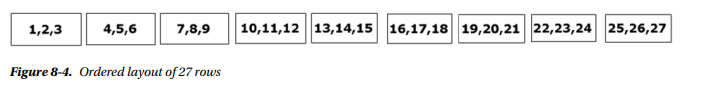
Một lợi ích khác của một nonclustered index là vì nó nằm trong một cấu trúc riêng biệt từ bảng dữ liệu, nên nó có thể được đặt trong một nhóm tệp khác, với một đường dẫn I / O khác, như được giải thích trong Chương 3. Điều này có nghĩa là SQL Server có thể truy cập chỉ mục và bảng đồng thời, làm cho tìm kiếm thậm chí nhanh hơn.

Các index lưu trữ thông tin của chúng trong một cây cân bằng, được gọi là cây B, cấu trúc, do đó số lần đọc cần thiết để tìm một hàng cụ thể được giảm thiểu. Ví dụ sau đây cho thấy lợi ích của cấu trúc cây B.

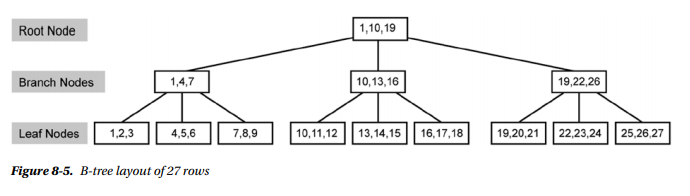
Hãy xem xét một bảng có một cột với 27 hàng theo thứ tự ngẫu nhiên và chỉ có 3 hàng trên mỗi leaf page. Giả sử bố trí của các hàng trong các trang như trong Hình 8-3.



Để tìm kiếm hàng (hoặc nhiều hàng) cho giá trị cột là 5, SQL Server phải quét tất cả các hàng và trang, vì ngay cả hàng cuối cùng trong trang cuối có thể có giá trị 5. Bởi vì số lần đọc phụ thuộc vào số lượng trang được truy cập, chín thao tác đọc (lấy các trang từ đĩa và chuyển chúng vào bộ nhớ) phải được thực hiện mà không có index trên cột. Nội dung này có thể được sắp xếp bằng cách tạo một index trên cột, với bố cục tổng hợp của các hàng và trang được hiển thị trong Hình 8-4.



Lập chỉ mục cột sắp xếp nội dung theo cách sắp xếp. Điều này cho phép SQL Server xác định giá trị có thể cho vị trí hàng trong cột đối với giá trị của vị trí hàng khác trong cột. Ví dụ, trong Hình 8-4, khi SQL Server tìm thấy hàng đầu tiên có giá trị cột 6, có thể chắc chắn rằng không còn hàng nào có giá trị cột 5. Do đó, chỉ có hai thao tác đọc được yêu cầu để tìm nạp các hàng với giá trị 5 khi nội dung được lập chỉ mục. Tuy nhiên, điều gì xảy ra nếu bạn muốn tìm kiếm giá trị cột 25? Điều này sẽ yêu cầu chín hoạt động đọc! Vấn đề này được giải quyết bằng cách thực hiện các chỉ mục bằng cách sử dụng cấu trúc cây B (như trong Hình 8-5).



Cây B bao gồm một nút bắt đầu (hoặc trang) được gọi là nút gốc với các nút nhánh (hoặc trang) mọc ra từ nó (hoặc được liên kết với nó). Tất cả các key được lưu trữ trong lá. Chứa trong mỗi nút bên trong (phía trên các nút lá) là các con trỏ tới các nút nhánh của nó và các giá trị đại diện cho giá trị nhỏ nhất được tìm thấy trong nút nhánh. Các khóa được giữ theo thứ tự sắp xếp trong mỗi nút. Cây B sử dụng cấu trúc cây cân bằng để truy xuất bản ghi hiệu quả. Cây B được cân bằng khi các nút lá đều ở cùng cấp độ từ nút gốc. Ví dụ, tạo một chỉ mục trên nội dung trước sẽ tạo ra cấu trúc cây B cân bằng được hiển thị trong Hình 8-5. Ở cấp độ dưới cùng, tất cả các nút lá được kết nối với nhau thông qua một danh sách được liên kết đôi, có nghĩa là mỗi trang trỏ đến trang theo sau nó và trang theo sau nó quay lại trang trước. Điều này ngăn chặn việc phải sao lưu chuỗi khi các trang được vượt qua các định nghĩa của các trang trung gian (This prevents having to go back up the chain when pages are traversed beyond the definitions of the intermediate pages.).

Thuật toán B-tree giảm thiểu số lượng trang cần truy cập để xác định vị trí khóa mong muốn, từ đó tăng tốc quá trình truy cập dữ liệu. Ví dụ, trong Hình 8-5, việc tìm kiếm giá trị khóa 5 bắt đầu ở nút gốc trên cùng. Vì giá trị khóa nằm trong khoảng từ 1 đến 10, quá trình tìm kiếm theo nhánh bên trái đến nút tiếp theo. Vì giá trị khóa 5 nằm giữa các giá trị 4 và 7, quá trình tìm kiếm đi theo nhánh giữa đến nút tiếp theo với giá trị khóa bắt đầu là 4. Quá trình tìm kiếm lấy giá trị khóa 5 từ trang lá này. Nếu giá trị khóa 5 không tồn tại trong trang này, quá trình tìm kiếm sẽ dừng lại kể từ khi nó khởi động trang lá. Tương tự, giá trị khóa 25 cũng có thể được tìm kiếm bằng cách sử dụng cùng số lần đọc.

**Index Overhead**

Lợi ích hiệu suất của các index đi kèm với chi phí. Các bảng có chỉ mục yêu cầu nhiều dung lượng lưu trữ và bộ nhớ hơn cho các index page ngoài các trang dữ liệu của bảng. Các truy vấn thao tác dữ liệu (các câu lệnh INSERT, UPDATE và DELETE hoặc phần CUD) có thể mất nhiều thời gian hơn và cần nhiều thời gian xử lý hơn để duy trì các chỉ mục của các bảng thay đổi liên tục. Điều này là do, không giống như một câu lệnh CHỌN, các truy vấn thao tác dữ liệu sẽ sửa đổi nội dung dữ liệu của một bảng. Nếu một câu lệnh INSERT thêm một hàng vào bảng, thì nó cũng phải thêm một hàng trong cấu trúc index. Nếu chỉ mục là một clustered index, thì chi phí vẫn cao hơn vì hàng phải được thêm vào các trang dữ liệu theo đúng thứ tự, điều này có thể yêu cầu các hàng dữ liệu khác được định vị lại bên dưới vị trí nhập của hàng mới. Các truy vấn thao tác dữ liệu UPDATE và DELETE thay đổi các trang chỉ mục một cách tương tự.

Khi thiết kế các index, bạn sẽ vận hành theo hai quan điểm khác nhau: hệ thống hiện có, đã được sản xuất, nơi bạn cần đo lường tác động tổng thể của một index và cách tiếp cận chiến thuật mà tất cả những gì bạn lo lắng là lợi ích trước mắt của một index, thường là khi ban đầu thiết kế một hệ thống. Khi bạn phải đối phó với hệ thống hiện có, bạn nên đảm bảo rằng lợi ích hiệu suất của một index lớn hơn chi phí bổ sung trong xử lý tài nguyên. Bạn có thể làm điều này bằng cách sử dụng các Extended Events (được giải thích trong Chương 3) để thực hiện tối ưu hóa khối lượng công việc tổng thể (được giải thích trong Chương 25). Khi bạn tập trung hoàn toàn vào các lợi ích trước mắt của một chỉ mục, SQL Server

cung cấp một loạt các khung nhìn quản lý động cung cấp thông tin chi tiết về hiệu suất của các chỉ mục, sys.dm\_db\_index\_operational\_stats hoặc sys.dm\_db\_index\_usage\_stats. Chế độ xem sys.dm\_db\_index\_operational\_stats hiển thị hoạt động cấp thấp, chẳng hạn như khóa và I / O, trên một index đang được sử dụng. Chế độ xem sys.dm\_db\_index\_usage\_stats trả về số liệu thống kê của các hoạt động index khác nhau đã xảy ra với một chỉ mục theo thời gian. Cả hai thứ này sẽ được sử dụng rộng rãi hơn trong Chương 20 khi tôi thảo luận về việc blocking

**Index Design Recommendations**

Các khuyến nghị chính cho thiết kế chỉ mục như sau:

* Kiểm tra mệnh đề WHERE và câu lệnh JOIN
  + Sử dụng các narrow indexs.
  + Kiểm tra tính duy nhất của cột.
  + Kiểm tra kiểu dữ liệu cột.
  + Xem xét thứ tự cột.
  + Xem xét loại chỉ mục (clustered so với nonclustered)

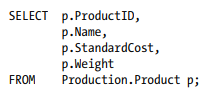
Hãy lần lượt xem xét từng đề nghị này.

- Examine the WHERE Clause and JOIN Criteria Columns

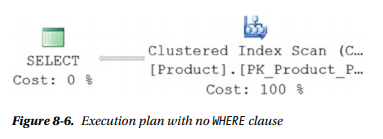
Khi một truy vấn được gửi đến SQL Server, trình tối ưu hóa truy vấn sẽ cố gắng tìm cơ chế truy cập dữ liệu tốt nhất cho mọi bảng được đề cập trong truy vấn. Đây là cách nó thực hiện điều này:

* Trình tối ưu hóa xác định các cột có trong mệnh đề WHERE và tiêu chí JOIN.
* Trình tối ưu hóa sau đó kiểm tra các chỉ mục trên các cột đó.
* Trình tối ưu hóa đánh giá tính hữu dụng của từng chỉ số bằng cách xác định độ chọn lọc của
* mệnh đề (nghĩa là có bao nhiêu hàng sẽ được trả về) từ số liệu thống kê được duy trì trên chỉ mục.
* Các ràng buộc như khóa chính và khóa ngoại cũng được đánh giá và sử dụng bởi trình tối ưu hóa để xác định tính chọn lọc của các đối tượng được sử dụng trong truy vấn.
* Cuối cùng, trình tối ưu hóa ước tính phương pháp lấy các hàng đủ điều kiện ít tốn kém nhất, dựa trên thông tin thu thập trong các bước trước

Để hiểu tầm quan trọng của cột mệnh đề WHERE trong truy vấn, hãy xem xét một ví dụ. Hãy trở lại Hãy quay lại list code ban đầu giúp bạn hiểu index là gì; truy vấn bao gồm một câu lệnh SELECT mà không có bất kỳ mệnh đề WHERE nào, như sau:



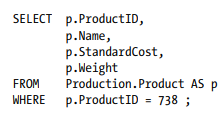
Trình tối ưu hóa truy vấn thực hiện quét clustered index, tương đương với quét bảng so với heap trên bảng có clustered index, để đọc các hàng như trong Hình 8-6 (switch on the Include Actual Execution Plan option by using Ctrl+M inside a query window, as well as the Set Statistics I0 option by right-clicking and selecting Query Options and then selecting the appropriate check box in the Advanced tab).



Số lần đọc logic được báo cáo bởi SET STATISTICS I0 cho câu lệnh SELECT như sau:

Table 'Product'. Scan count 1, logical reads 15

Để hiểu tác dụng của cột mệnh đề WHERE đối với các quyết định của trình tối ưu hóa truy vấn, hãy thêm mệnh đề WHERE để truy xuất một hàng đơn



Với mệnh đề WHERE được đặt ra, trình tối ưu hóa truy vấn sẽ kiểm tra cột mệnh đề WHERE ProductID, xác định tính khả dụng của chỉ mục PK\_Product\_ProductId trên cột Productld, đánh giá tính chọn lọc cao (nghĩa là chỉ một hàng sẽ được trả về) cho mệnh đề WHERE từ thống kê trên chỉ mục PK\_Product\_ProductId và quyết định sử dụng chỉ mục đó để truy xuất dữ liệu, như trong Hình 8-7

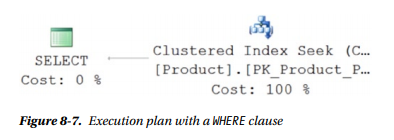


Table 'Product'. Scan count 0, logical reads 2

Hành vi của trình tối ưu hóa truy vấn cho thấy cột mệnh đề WHERE giúp trình tối ưu hóa chọn một hoạt động lập index tối ưu cho truy vấn. Điều này cũng có thể áp dụng cho một cột được sử dụng trong tiêu chí JOIN giữa hai bảng.

Trình tối ưu hóa tìm kiếm các index trên cột mệnh đề WHERE hoặc cột tiêu chí JOIN và, nếu có, xem xét sử dụng index để truy xuất các hàng từ bảng. Trình tối ưu hóa truy vấn xem xét các index trên các cột mệnh đề WHERE và các cột tiêu chí JOIN trong khi thực hiện một truy vấn. Do đó, việc có các chỉ mục trên các cột được sử dụng thường xuyên trong mệnh đề WHERE, mệnh đề HAVING và tiêu chí JOIN của truy vấn SQL giúp trình tối ưu hóa tránh quét base table. Khi lượng dữ liệu bên trong một bảng quá nhỏ đến mức vừa với một trang (8KB), quét bảng có thể hoạt động tốt hơn so với tìm kiếm index. Nếu bạn có một index tốt tại chỗ nhưng bạn vẫn đang quét, hãy xem xét vấn đề này.

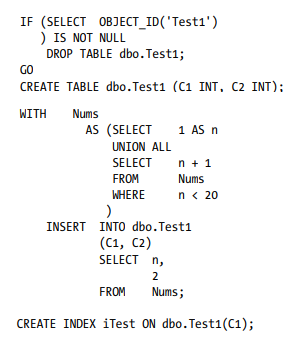
**Use Narrow Indexes**

Để có hiệu suất tốt nhất, bạn nên sử dụng loại dữ liệu hẹp như thực tế khi tạo chỉ mục. Thu hẹp trong bối cảnh này có nghĩa là một loại dữ liệu nhỏ như bạn thực tế có thể. Bạn cũng nên tránh các cột loại dữ liệu rất rộng trong một chỉ mục. Các cột có kiểu dữ liệu chuỗi (CHAR, VARCHAR, NCHAR và NVARCHAR) đôi khi có thể khá rộng, cũng như các mã định danh nhị phân và duy nhất trên toàn cầu (GUID). Trừ khi chúng thực sự cần thiết, giảm thiểu việc sử dụng các cột kiểu dữ liệu rộng với kích thước lớn trong một chỉ mục. Bạn có thể tạo các chỉ mục trên một tổ hợp các cột trong một bảng. Để có hiệu suất tốt nhất, hãy sử dụng càng ít cột trong một chỉ mục nếu cần. Nhưng, sử dụng các cột bạn cần sử dụng để xác định khóa hữu ích cho index.

Một chỉ mục hẹp có thể chứa nhiều hàng hơn trong trang chỉ mục 8KB so với chỉ mục rộng. Điều này có tác dụng sau:

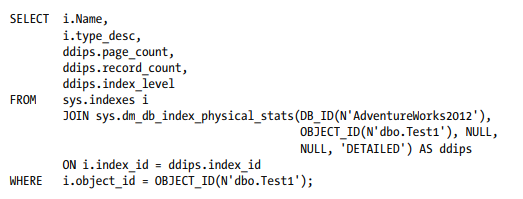
* Giảm I / O (bằng cách đọc ít hơn 8KB trang)
* Làm cho bộ đệm cơ sở dữ liệu hiệu quả hơn vì SQL Server có thể lưu trữ ít index page hơn, do đó giảm số lần đọc logic cần thiết cho các index page trong bộ nhớ
* Giảm dung lượng lưu trữ cho cơ sở dữ liệu

Để hiểu làm thế nào một narrow index có thể giảm số lần đọc logic, hãy tạo một bảng kiểm tra với 20 hàng và một chỉ mục.



Do cột được lập chỉ mục hẹp (kiểu dữ liệu INT là 4 byte), tất cả các hàng chỉ mục có thể được cung cấp trong một index page 8KB. Như được hiển thị trong Hình 8-8, bạn có thể xác nhận điều này trong các khung nhìn quản lý động được liên kết với các chỉ mục

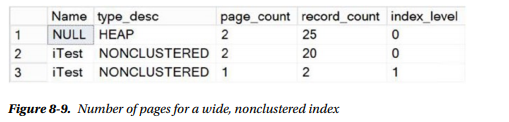




Bảng hệ thống sys.indexes được lưu trữ trong mỗi cơ sở dữ liệu và chứa thông tin cơ bản trên mọi chỉ mục trong cơ sở dữ liệu. Hàm quản lý động, sys.dm\_db\_index\_physical\_stats, chứa thông tin chi tiết hơn về số liệu thống kê trên chỉ mục (bạn sẽ tìm hiểu thêm về DMF này trong Chương 13). Để hiểu nhược điểm của khóa chỉ mục rộng, hãy sửa đổi kiểu dữ liệu của cột được lập chỉ mục c1 từ INT thành CHAR (500) (hẹp\_ notify.sql khi tải xuống).



Độ rộng của cột có kiểu dữ liệu INT là 4 byte và chiều rộng của cột có kiểu dữ liệu CHAR (500) là 500 byte. Do chiều rộng lớn của cột được lập chỉ mục (indexed column), cần có hai index page để chứa tất cả 20 index row. Bạn có thể xác nhận điều này trong hàm quản lý động sys.dm\_db\_index\_physical\_stats bằng cách chạy lại truy vấn với nó (xem Hình 8-9).



Kích thước khóa chỉ mục lớn làm tăng số lượng index page, do đó tăng lượng bộ nhớ và hoạt động đĩa cần thiết cho chỉ mục. Chúng tôi luôn khuyến nghị rằng kích thước khóa chỉ mục phải hẹp như bạn có thể làm cho nó. Drop test table trước khi tiếp tục.

DROP TABLE dbo.Test1;

**Examine Column Uniqueness**

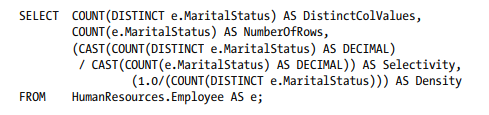
Tạo một chỉ mục trên các cột có phạm vi giá trị duy nhất rất thấp (như MaritalStatus) sẽ không có lợi cho hiệu suất vì trình tối ưu hóa truy vấn sẽ không thể sử dụng chỉ mục để thu hẹp hiệu quả các hàng được trả về. Hãy xem xét một cột MaritalStatus chỉ có hai giá trị duy nhất: M và S. Khi bạn thực hiện truy vấn với cột MaritalStatus trong mệnh đề WHERE, bạn kết thúc với một số lượng lớn các hàng từ bảng (giả sử phân phối của M và S tương đối như nhau), dẫn đến một bảng tốn kém hoặc quét clustered index. Luôn luôn có các cột trong mệnh đề WHERE với nhiều hàng duy nhất (hoặc độ chọn lọc cao) để hạn chế số lượng hàng được truy cập.

Bạn nên tạo một chỉ mục trên các cột đó để giúp trình tối ưu hóa truy cập vào tập kết quả nhỏ

Hơn nữa, trong khi tạo một chỉ mục trên nhiều cột, còn được gọi là chỉ mục tổng hợp, thứ tự cột có vấn đề. Trong nhiều trường hợp, trước tiên, sử dụng cột chọn lọc nhất sẽ giúp lọc các index row hiệu quả hơn

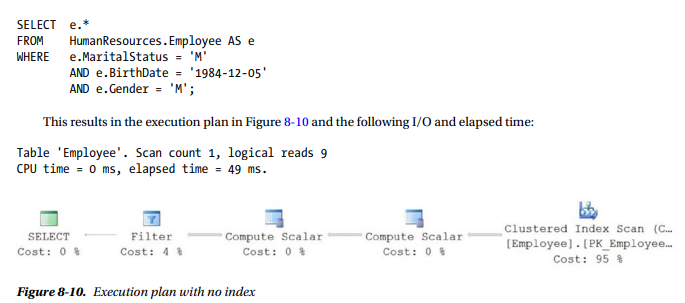
Từ điều này, bạn có thể thấy rằng điều quan trọng là phải biết tính chọn lọc của một cột trước khi tạo một chỉ mục trên đó.

Bạn có thể tìm thấy điều này bằng cách thực hiện một truy vấn như thế này; chỉ cần thay thế tên bảng và cột



Cột có số lượng giá trị duy nhất (hoặc độ chọn lọc) cao nhất có thể là ứng cử viên tốt nhất để lập chỉ mục khi được đề cập trong mệnh đề WHERE hoặc tiêu chí tham gia. Bạn cũng có thể có dữ liệu đặc biệt nơi bạn có hàng trăm hàng dữ liệu chung chỉ có một vài dữ liệu duy nhất. Một số ít cũng sẽ được hưởng lợi từ một chỉ số. Bạn có thể làm cho điều này thậm chí có lợi hơn bằng cách sử dụng các chỉ mục được lọc (sẽ thảo luận chi tiết hơn sau).

Để hiểu mức độ chọn lọc của index key ảnh hưởng đến việc sử dụng chỉ mục, hãy xem cột MaritalStatus trong bảng HumanResource.Employee. Nếu bạn chạy truy vấn trước đó, bạn sẽ thấy rằng nó chỉ chứa hai giá trị riêng biệt trong 290 hàng, đó là độ chọn lọc là 0,0069 và mật độ 0,5. Một truy vấn chỉ tìm MaritalStatus của M sẽ giống như thế này:



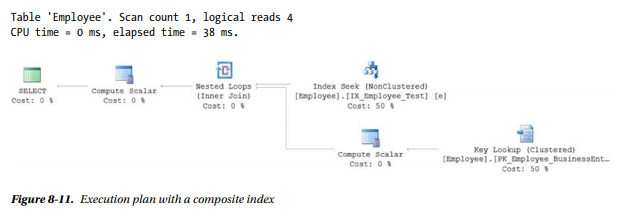
Dữ liệu được trả về bằng cách quét clustered index (nơi lưu trữ dữ liệu) để tìm các giá trị phù hợp trong đó MaritalStatus = 'M'. (Các toán tử khác sẽ được trình bày trong Chương 14 và 15.) Nếu bạn đặt một chỉ mục trên cột, như vậy và chạy lại truy vấn, execution plan vẫn giữ nguyên.



Dữ liệu chỉ là không đủ chọn lọc để index được sử dụng, chứ chưa nói là hữu ích. Nếu thay vào đó, bạn sử dụng một chỉ mục tổng hợp trông như thế này:



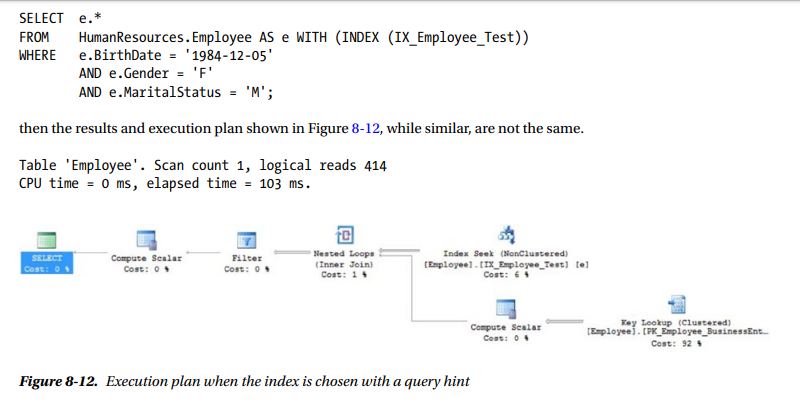
và sau đó chạy lại truy vấn để xem kế hoạch thực hiện trong Hình 8-11 và kết quả thực hiện, bạn nhận được điều này:



Bây giờ bạn đang làm tốt hơn bạn làm với quét clustered index. Một hoạt động Index Seek sạch đẹp mất ít hơn một nửa thời gian để thu thập dữ liệu. Phần còn lại được sử dụng trong hoạt động Key Lookup. Key Lookup được sử dụng để gọi là tra cứu dấu trang.

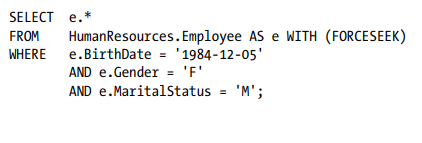
Mặc dù không có cột nào trong câu hỏi có thể có tính tự chọn đủ để tạo ra một chỉ số hợp lý, ngoại trừ có thể là cột ngày sinh, cùng với đó chúng cung cấp đủ độ chọn lọc để trình tối ưu hóa tận dụng chỉ số được cung cấp.

Có thể cố gắng buộc truy vấn sử dụng chỉ mục kiểm tra đầu tiên bạn đã tạo. Nếu bạn bỏ chỉ mục ghép, hãy tạo lại bản gốc, sau đó sửa đổi truy vấn như sau bằng cách sử dụng một gợi ý truy vấn để buộc sử dụng kiến trúc Index gốc

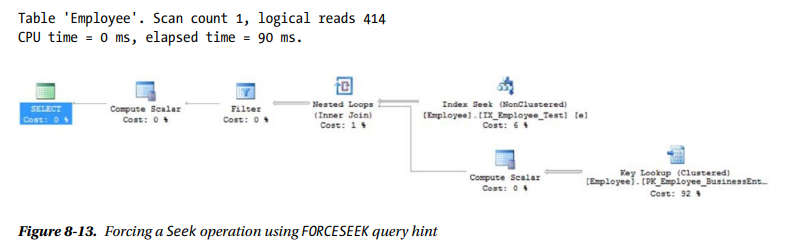


Bạn thấy cùng một chỉ mục tìm kiếm, nhưng số lần đọc đã tăng hơn gấp đôi và chi phí ước tính trong kế hoạch thực hiện đã thay đổi. Mặc dù việc buộc trình tối ưu hóa phải chọn một chỉ mục là có thể, nhưng rõ ràng đây không phải là một cách tiếp cận tối ưu.

Một cách khác để buộc một hành vi khác kể từ SQL Server 2012 là gợi ý truy vấn FORCESEEK. FORCESEEK làm cho nó để trình tối ưu hóa sẽ chỉ chọn các hoạt động Index Seek. Nếu truy vấn được viết lại như thế này:



thay đổi kết quả I / O, thời gian thực hiện và kế hoạch thực hiện một lần nữa (Hình 8-13), bạn kết thúc với những kết quả này



Việc giới hạn các tùy chọn của trình tối ưu hóa và buộc các hành vi có thể giúp ích trong một số trường hợp, nhưng thường xuyên, như thể hiện với kết quả ở đây, việc tăng thời gian thực hiện và số lần đọc là không hữu ích.

Trước khi tiếp tục, hãy nhớ bỏ test index khỏi bảng.



**Examine the Column Data Type**

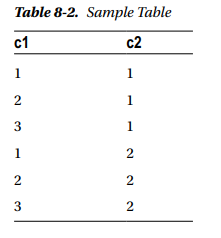
Kiểu dữ liệu của một chỉ mục quan trọng. Ví dụ: tìm kiếm chỉ mục trên các khóa số nguyên là nhanh do kích thước nhỏ và thao tác số học dễ dàng của kiểu dữ liệu INTEGER (hoặc INT). Bạn cũng có thể sử dụng các biến thể khác của các loại dữ liệu số nguyên (BIGINT, SMALLINT và TINYINT) cho các cột chỉ mục, trong khi các loại dữ liệu chuỗi (CHAR, VARCHAR, NCHAR và NVARCHAR) yêu cầu thao tác khớp chuỗi, thường tốn kém hơn so với khớp số nguyên hoạt động.

Giả sử bạn muốn tạo một chỉ mục trên một cột và bạn có hai cột ứng cử viên. Một cột có kiểu dữ liệu INTEGER và cột kia có kiểu dữ liệu CHAR (4). Mặc dù kích thước của cả hai loại dữ liệu là 4 byte trong SQL Server 2014, bạn vẫn nên thích index loại dữ liệu INTEGER. Nhìn vào các phép toán số học làm ví dụ. Giá trị 1 trong kiểu dữ liệu CHAR (4) thực sự được lưu trữ dưới dạng 1 theo sau là ba khoảng trắng, kết hợp của bốn byte sau: 0x35, 0x20, 0x20 và 0x20. CPU không hiểu cách thực hiện các phép toán số học trên dữ liệu này và do đó nó chuyển đổi thành kiểu dữ liệu số nguyên trước các phép toán số học, trong khi giá trị 1 trong kiểu dữ liệu số nguyên được lưu là 0x00000001. CPU có thể dễ dàng thực hiện các hoạt động số học trên dữ liệu này.

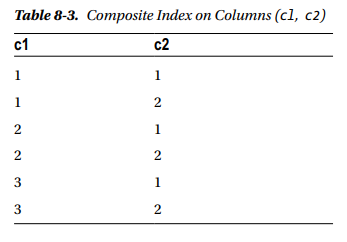
Tất nhiên, hầu hết thời gian, bạn đã giành được sự lựa chọn đơn giản giữa các loại dữ liệu có kích thước giống hệt nhau, cho phép bạn chọn loại tối ưu hơn. Giữ thông tin này trong tâm trí khi thiết kế và xây dựng các chỉ mục của bạn.

**Consider Column Order**

Index key được sắp xếp trên cột đầu tiên của index và sau đó được đặt vào cột tiếp theo trong mỗi giá trị của cột trước đó. Cột đầu tiên trong một chỉ mục ghép thường được gọi là cạnh hàng đầu của chỉ mục. Ví dụ, hãy xem xét Bảng 8-2.



Nếu một chỉ mục tổng hợp được tạo trên các cột (c1, c2), thì chỉ mục sẽ được sắp xếp như trong Bảng 8-3.

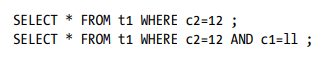


Như được hiển thị trong Bảng 8-3, dữ liệu được sắp xếp trên cột đầu tiên (c1) trong chỉ mục tổng hợp. Trong mỗi giá trị của cột đầu tiên, dữ liệu được sắp xếp thêm trên cột thứ hai (c2).

Do đó, thứ tự cột trong một chỉ số tổng hợp là một yếu tố quan trọng trong hiệu quả của chỉ số. Bạn có thể thấy điều này bằng cách xem xét những điều sau đây

* Column uniqueness
* Column width
* Column data type

Ví dụ: giả sử hầu hết các truy vấn của bạn trên bảng t1 tương tự như sau:

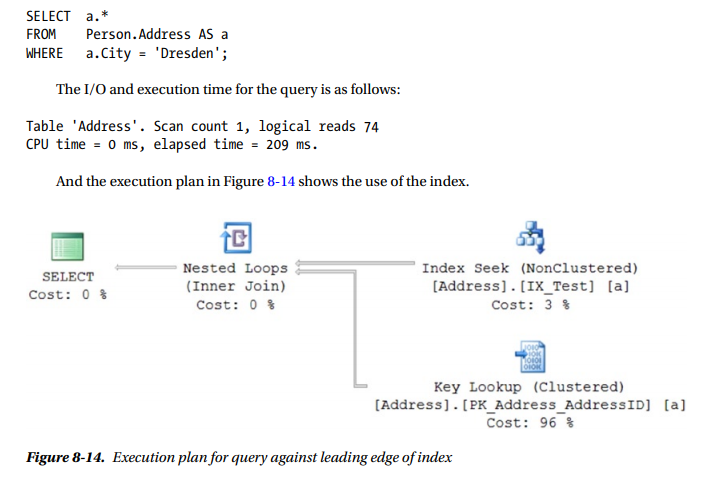


Một chỉ mục trên (c2, c1) sẽ có lợi cho cả 2 truy vấn. Nhưng một chỉ mục trên (c1, c2) sẽ không hữu ích cho cả hai truy vấn vì nó sẽ sắp xếp dữ liệu ban đầu trên cột c1, trong khi câu lệnh WHERE đầu tiên cần dữ liệu được sắp xếp trên cột c2.

Để hiểu tầm quan trọng của thứ tự cột trong một chỉ mục, hãy xem xét ví dụ sau. Trong bảng Person.Address, có một cột cho City và một cột khác cho PostalCode. Tạo một chỉ mục trên bảng như thế này:

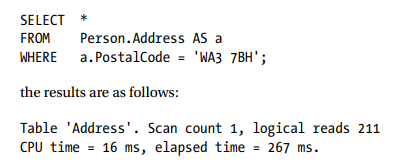


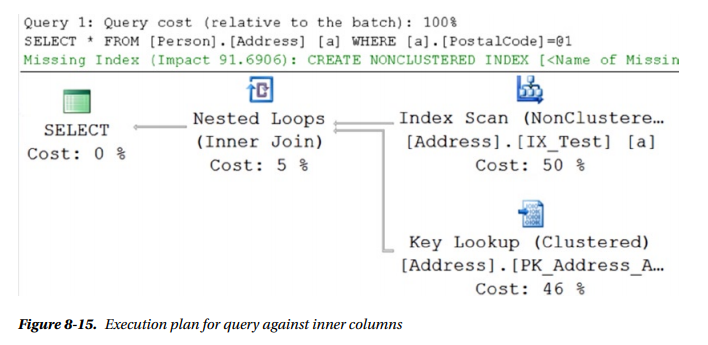
Một câu lệnh SELECT đơn giản chạy với bảng sẽ sử dụng chỉ mục mới này sẽ trông giống như thế này:



Vì vậy, truy vấn này đang tận dụng lợi thế dẫn đầu của chỉ mục để thực hiện thao tác Seek để lấy dữ liệu.

Nếu, thay vì truy vấn bằng cách sử dụng cạnh đầu, bạn sử dụng một cột khác trong chỉ mục như truy vấn sau



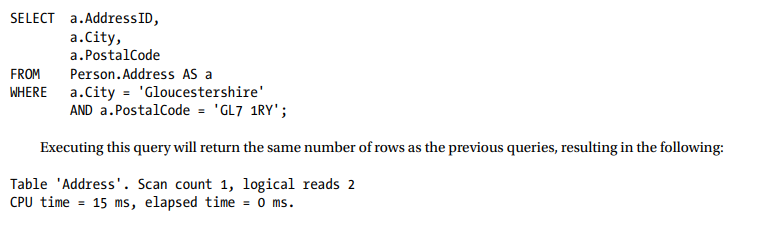


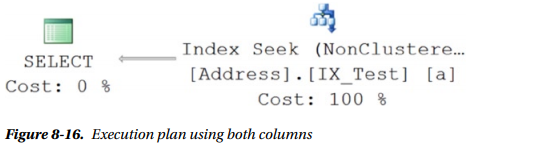
Cả hai truy vấn trả về 31 hàng từ cùng một bảng, nhưng số lần đọc tănsg từ 74 lên 180. Bạn bắt đầu thấy sự khác biệt giữa hoạt động Index Seek trong Hình 8-14 và hoạt động Scan chỉ mục trong Hình 8-15.

Cũng lưu ý rằng vì phải thực hiện quét, trình tối ưu hóa chỉ ra rằng có thể có một chỉ mục khả thi để giúp cải thiện hiệu suất của truy vấn.

Thiếu thông tin chỉ mục hữu ích như một con trỏ dẫn đến tiềm năng cho một chỉ mục mới hoặc tốt hơn trên bảng, nhưng đừng cho rằng nó luôn luôn đúng. Bạn có thể nhấp chuột phải vào vị trí thiếu thông tin chỉ mục và Missing Index Details từ context menu. Điều đó sẽ mở ra một cửa sổ truy vấn mới với các chi tiết của chỉ mục được trình bày, sẵn sàng để tạo. Nếu bạn quyết định kiểm tra chỉ mục đó, hãy đảm bảo bạn đổi tên nó từ tên mặc định.

Cuối cùng, để xem thứ tự của chỉ mục thực sự tỏa sáng, hãy thay đổi truy vấn này:





Những thay đổi căn bản trong I / O và execution plan thể hiện việc sử dụng thực sự của một chỉ số ghép, chỉ số bao trùm.

Điều này được đề cập chi tiết trong phần Covering Indexes của họ trong chương 9.



Xem xét Type of Index

Trong SQL Server, từ tất cả các loại chỉ mục khác nhau có sẵn cho bạn, hầu hết thời gian bạn sẽ làm việc với hai loại chỉ mục chính: clustered và nonclustered. Cả hai loại đều có cấu trúc cây B. Sự khác biệt chính giữa hai loại là các trang lá trong một clustered index là các trang dữ liệu của bảng và do đó theo cùng thứ tự với dữ liệu mà chúng trỏ tới. Điều này có nghĩa là clustered index là bảng. Khi bạn tiến hành, bạn sẽ thấy rằng sự khác biệt ở cấp độ lá giữa hai loại chỉ mục trở nên quan trọng khi xác định loại chỉ mục sẽ sử dụng

**Clustered Indexes**

Các trang lá của một clustered index và các trang dữ liệu của bảng mà chỉ mục được bật là một và giống nhau. Do đó, các hàng của bảng được sắp xếp vật lý trênclustered index column và do chỉ có thể có một thứ tự vật lý của dữ liệu bảng, nên một bảng chỉ có thể có một clustered index

Khi bạn tạo một ràng buộc khóa chính, SQL Server sẽ tự động tạo nó như là một clustered index duy nhất trên primary key, nếu không tồn tại và nếu nó không được chỉ định rõ ràng thì chỉ mục đó phải là một nonclustered index duy nhất. Đây không phải là một yêu cầu; Nó chỉ là hành vi mặc định. Bạn có thể thay đổi định nghĩa của khóa chính trước khi tạo nó trên bảng.

**Heap Tables**

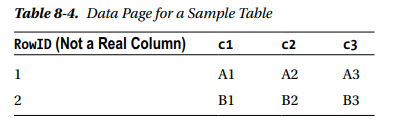
Như đã đề cập trước đó trong chương, một bảng không có clustered index được gọi là bảng heap. Các hàng dữ liệu của bảng heap không được lưu trữ theo bất kỳ thứ tự cụ thể nào hoặc được liên kết với các trang liền kề trong bảng. Cấu trúc không có tổ chức này của bảng heap thường làm tăng chi phí truy cập của bảng heap lớn khi so sánh với việc truy cập một bảng nonheap lớn (một bảng có clustered index).

**Relationship with Nonclustered Indexes**

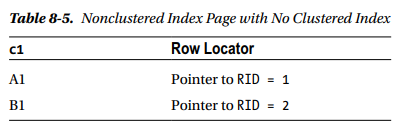
Có một mối quan hệ thú vị giữa clustered index và nonclustered indexes trong SQL Server. Một index row của một nonclustered index có chứa một con trỏ tới hàng dữ liệu tương ứng của bảng. Con trỏ này được gọi là một bộ định vị hàng. Giá trị của bộ định vị hàng phụ thuộc vào việc các trang dữ liệu được lưu trữ trong một heap hay trên một clustered index.

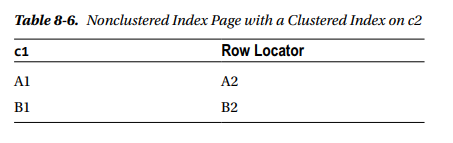
Đối với một nonclustered index, bộ định vị hàng là một con trỏ tới mã định danh hàng (RID) cho hàng dữ liệu trong một heap. Đối với một bảng có clustered index, bộ định vị hàng là clustered index key value

Ví dụ: giả sử bạn có một bảng heap không có clustered index, như được hiển thị trong Bảng 8-4

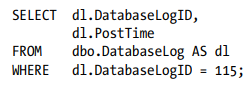


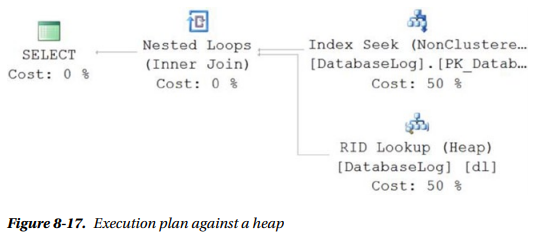
Một nonclustered index trên cột c1 trong một heap sẽ khiến bộ định vị hàng cho các index row chứa một con trỏ tới hàng dữ liệu tương ứng trong bảng cơ sở dữ liệu, như trong Bảng 8-5.



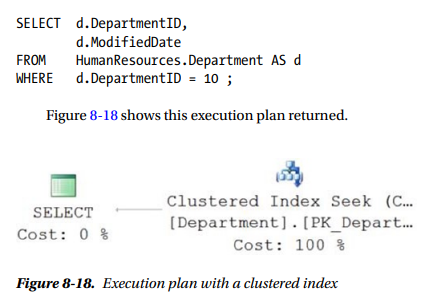


Để xác minh sự phụ thuộc này giữa một clustered và một nonclustered index, hãy để xem xét một ví dụ. Trong cơ sở dữ liệu AdventureWorks2012, bảng dbo.DatabaseLog không chứa clustered index, chỉ là nonclustered primary key. Nếu một truy vấn được chạy với nó như sau, thì việc thực thi sẽ giống như Hình 8-17.





Như bạn có thể thấy, chỉ mục đã được sử dụng trong thao tác Tìm kiếm. Nhưng vì dữ liệu được lưu trữ riêng biệt với nonclustered index, nên một thao tác bổ sung, hoạt động Tra cứu RID, được yêu cầu để truy xuất dữ liệu, sau đó được nối lại với thông tin từ hoạt động Index Seek thông qua hoạt động Vòng lặp lồng nhau. Điều này là một ví dụ kinh điển về cái được gọi là tra cứu, trong trường hợp này là tra cứu RID, được giải thích chi tiết hơn trong phần Defining the Lookup. Một truy vấn tương tự chạy với một bảng có clustered index tại chỗ sẽ trông như thế này



Mặc dù khóa chính được sử dụng theo cách tương tự như truy vấn trước đó, nhưng lần này, nó chống lại một clustered index.

Như bạn đã biết, điều này có nghĩa là dữ liệu được lưu trữ cùng với chỉ mục, do đó, cột bổ sung không có yêu cầu thao tác tra cứu để lấy dữ liệu. Mọi thứ được trả về bởi hoạt động clustered Index Seek đơn giản

Để điều hướng từ một nonclustered index row đến một data row, mối quan hệ này giữa hai loại chỉ mục đòi hỏi một sự bổ sung để điều hướng cấu trúc cây B của clustered index. Nếu không có clustered index, bộ định vị hàng của nonclustered index sẽ có thể điều hướng trực tiếp từ nonclustered index row đến data rowtrong bảng cơ sở. Sự hiện diện của clustered index làm cho việc điều hướng từ nonclustered index row đến data row đi qua cấu trúc cây B của clustered index, vì giá trị của trình định vị hàng mới trỏ clustered index key.

Mặt khác, hãy xem xét việc chèn một hàng trung gian theo thứ tự clustered index keyhoặc mở rộng nội dung của một hàng trung gian. Ví dụ: hãy tưởng tượng clustered index table chứa bốn hàng trên một trang, với các clustered index column là 1, 2, 4 và 5. Thêm một hàng mới trong bảng với clustered index là 3 sẽ yêu cầu khoảng trống trong trang giữa các giá trị 2 và 4. Nếu không đủ không gian ở vị trí đó, việc phân chia trang sẽ xảy ra trên trang dữ liệu (hoặc clustered index leaf page). Mặc dù việc phân chia trang dữ liệu sẽ gây ra việc di chuyển các hàng dữ liệu, các giá trị của nonclustered index row locator không cần phải được cập nhật.

Các bộ định vị hàng này tiếp tục trỏ đến cùng các giá trị khóa logic của clustered index key, mặc dù các hàng dữ liệu đã di chuyển vật lý đến một vị trí khác. Trong trường hợp phân chia trang dữ liệu, các bộ định vị hàng của các nonclustered index không cần phải được cập nhật. Đây là một điểm quan trọng vì các bảng thường có một số lượng lớn các nonclustered index.

Mọi thứ không hoạt động theo cùng một cách cho bảng heap. Mặc dù việc chia trang thành một heap không phải là một sự xuất hiện phổ biến và khi các heap bị chia tách, chúng không sắp xếp lại các vị trí theo cùng một cách với các clustered index, bạn có thể có các hàng di chuyển trong một heap, thường là do các cập nhật khiến cho heap không phù hợp với trang hiện tại của nó. Bất cứ điều gì làm cho vị trí của các hàng được di chuyển trong một heap đều dẫn đến một bản ghi chuyển tiếp được đặt vào vị trí ban đầu chỉ đến vị trí mới đó, đòi hỏi phải có thêm hoạt động I / O.

**Clustered Index Recommendations**

Mối quan hệ giữa một clustered index và một nonclustered index áp đặt một số cân nhắc clustered index, được giải thích trong các phần sau

**Create the Clustered Index First**

Vì tất cả nonclustered indexes giữ clustered index keys trong các index rows của chúng, nên thứ tự tạo nonclustered và clustered indexlà rất quan trọng. Ví dụ: nếu các nonclustered index được xây dựng trước khi clustered index được taho, thì nonclustered index row locator sẽ chứa một con trỏ tới RID tương ứng của bảng. Tạo clustered index sau sẽ sửa đổi tất cả các nonclustered index để chứa các clustered index key làm giá trị định vị hàng mới. Điều này xây dựng lại một cách hiệu quả tất cả các nonclustered index

Để có hiệu suất tốt nhất, tôi khuyên bạn nên tạo clustered index trước khi bạn tạo bất kỳ nonclustered index nào.

Điều này cho phép các nonclustered index có bộ định vị hàng của chúng được đặt thành clustered index key tại thời điểm tạo.

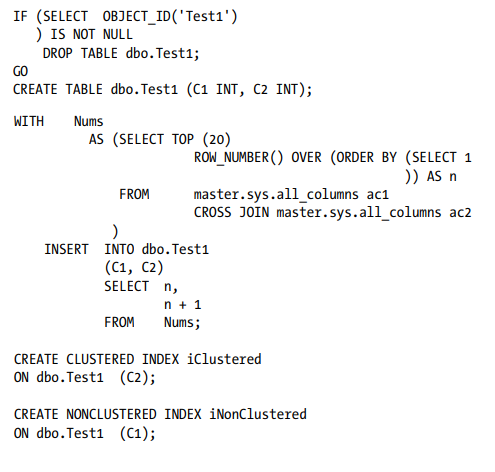
Điều này không có bất kỳ ảnh hưởng nào đến hiệu suất cuối cùng, nhưng việc xây dựng lại các index có thể là một công việc khá lớn.

Là một phần của việc tạo clustered index trước tiên, tôi cũng đề nghị bạn thiết kế các bảng trong cơ sở dữ liệu của bạn xung quanh clustered index. Nó phải là chỉ mục đầu tiên được tạo bởi vì bạn nên lưu trữ dữ liệu của mình dưới dạng một clustered index theo mặc định.

**Keep Clustered Indexes Narrow**

Vì tất cả nonclustered indexes giữ clustered keys làm công cụ định vị hàng của chúng, để hiệu suất tốt nhất giữ kích thước byte tổng thể của clustered index càng nhỏ càng tốt. Nếu bạn tạo một clustered index rộng, giả sử CHAR (500), ngoài việc có ít hàng trên mỗi trang trong cụm, điều này sẽ thêm 500 byte cho mỗi nonclustered index. Do đó, hãy giữ số lượng cột trong clustered keys ở mức tối thiểu và xem xét cẩn thận kích thước byte của mỗi cột được đưa vào clustered keys. Một cột của kiểu dữ liệu số nguyên thường tạo ra một ứng cử viên tốt cho một clustered keys, trong khi đó một cột kiểu dữ liệu chuỗi sẽ là một lựa chọn ít tối ưu hơn.

Để hiểu tác động của một clustered index rộng trên một nonclustered index, hãy xem xét ví dụ này. Tạo một bảng thử nghiệm nhỏ với một clustered index và một nonclustered index:

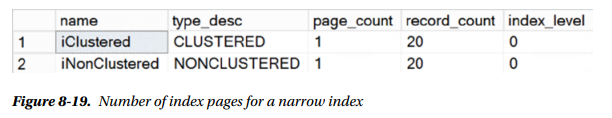


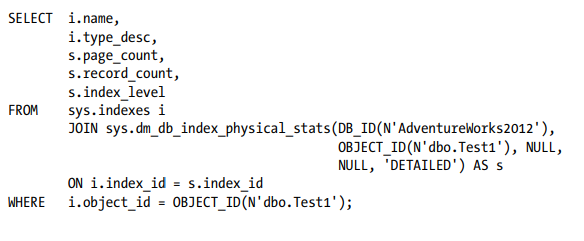
Vì bảng có một clustered index, nên bộ định vị hàng của nonclustered index có chứa giá trị clustered index key value. vì thế:

* Width of the nonclustered index row = width of the nonclustered index column + width of the clustered index column = size of INT data type + size of INT data type

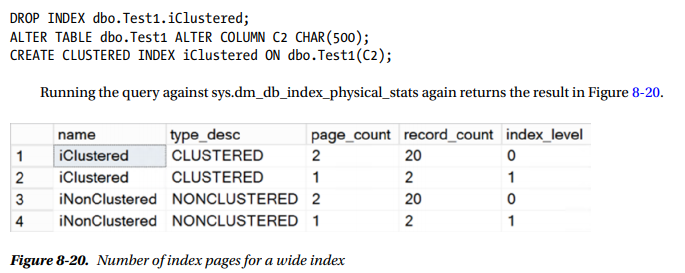
= 4 bytes + 4 bytes = 8 bytes

Với kích thước nhỏ này của một nonclustered index row, tất cả các hàng có thể được lưu trữ trong một trang chỉ mục. Bạn có thể xác nhận điều này bằng cách truy vấn các số liệu thống kê chỉ số, như trong Hình 8-19.





Để hiểu tác động của một clustered index rộng trên một nonclustered index, hãy sửa đổi kiểu dữ liệu của cột được lập chỉ mục cụm c2 từ INT thành CHAR (500).



Bạn có thể thấy rằng một clustered index rộng làm tăng chiều rộng của kích thước nonclustered index row. Do chiều rộng lớn của nonclustered index row, một trang chỉ mục 8KB không thể chứa tất cả các hàng chỉ mục. Thay vào đó, hai trang chỉ mục sẽ được yêu cầu lưu trữ tất cả 20 hàng chỉ mục. Trong trường hợp bảng lớn, việc mở rộng kích thước của các nonclustered index do kích thước clustered index key lớn có thể làm tăng đáng kể số lượng trang của các nonclustered index.

Do đó, kích thước của clustered index key size lớn không chỉ ảnh hưởng đến chiều rộng của chính nó mà còn mở rộng tất cả các nonclustered indexes trên bảng. Điều này làm tăng số lượng trang chỉ mục cho tất cả các chỉ mục trên bảng, tăng số lần đọc logic và I / O đĩa cần thiết cho các chỉ mục

**Rebuild the Clustered Index in a Single Step**

Do sự phụ thuộc của các nonclustered indexes vào clustered index, nên việc xây dựng lại clustered index thành các câu lệnh DROP INDEX và CREATE INDEX riêng biệt khiến tất cả các nonclustered index bị xây dựng lại hai lần. Để tránh điều này, hãy sử dụng mệnh đề DROP\_EXISTING của câu lệnh CREATE INDEX để xây dựng lại clustered index trong một bước nguyên tử duy nhất.

Tương tự, bạn cũng có thể sử dụng mệnh đề DROP\_EXISTING với nonclustered index.

Điều đáng chú ý là trong SQL Server 2005 và mới hơn, khi bạn thực hiện xây dựng lại một clustered index, bạn sẽ không thấy các nonclustered index cũng được xây dựng lại.

**Where Possible, Make the Clustered Index Unique**

Vì clustered index được sử dụng để lưu trữ dữ liệu, bạn phải có thể tìm thấy từng hàng. Mặc dù clustered index không nhất thiết phải là duy nhất về định nghĩa và lưu trữ của nó, nếu các giá trị khóa không phải là duy nhất, SQL Server sẽ không thể tìm thấy các hàng trừ khi có cách làm cho cụm xác định vị trí duy nhất của từng hàng dữ liệu riêng biệt. Vì vậy, SQL Server sẽ thêm một giá trị vào clustered index nonunique để làm cho nó trở nên duy nhất. Giá trị này được gọi là duy nhất. Nó thêm vào kích thước của clustered index của bạn cũng như tất cả các nonclustered index, như đã lưu ý trước đó.

Nó cũng có nghĩa là một chút xử lý được thêm vào để có được giá trị duy nhất khi mỗi hàng được chèn vào. Vì tất cả những lý do này, sẽ hợp lý khi làm cho clustered index duy nhất ở nơi bạn có thể. Đây là một lý do lớn tại sao hành vi mặc định cho các khóa chính là làm cho chúng trở thành một clustered index

Bạn không phải làm cho clustered index duy nhất. Nhưng bạn cần phải tính đến trình duy nhất trong tài khoản khi bạn xác định bộ nhớ và chỉ mục của mình.

**When to Use a Clustered Index**

**Accessing the Data Directly**

Với tất cả dữ liệu được lưu trữ trên các trang lá của một clustered index, bất cứ khi nào bạn truy cập vào cụm, dữ liệu sẽ có sẵn ngay lập tức. Một cách sử dụng cho một clustered index là để hỗ trợ đường dẫn truy cập được sử dụng phổ biến nhất cho dữ liệu. Bất kỳ quyền truy cập nào của clustered index không yêu cầu bất kỳ lần đọc bổ sung nào để truy xuất dữ liệu, điều đó có nghĩa là tìm kiếm hoặc quét đối với clustered index không yêu cầu bất kỳ lần đọc bổ sung nào để truy xuất dữ liệu đó. Đây là một lý do có khả năng khác mà Microsoft đã biến khóa chính thành một clustered index theo mặc định. Vì khóa chính thường là phương tiện có khả năng nhất để truy cập dữ liệu trong bảng, nên nó phục vụ tốt như một clustered index.

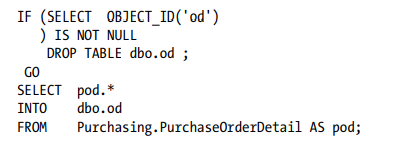
Chỉ cần nhớ rằng khóa chính là clustered index là một hành vi mặc định nhưng không nhất thiết là đường dẫn truy cập phổ biến nhất đến dữ liệu. Điều này có thể thông qua các ràng buộc khóa ngoài, các khóa thay thế trong bảng hoặc các cột khác. Lập kế hoạch và thiết kế các cụm với lưu trữ và truy cập trong tâm trí, và bạn sẽ ổn.

Clustered index hoạt động tốt như đường dẫn chính đến dữ liệu chỉ khi bạn truy cập một phần dữ liệu đáng kể trong một bảng. Mặt khác, nếu bạn truy cập vào các tập hợp con nhỏ của dữ liệu, bạn có thể tốt hơn với nonclustered covering index. Ngoài ra, bạn phải tính đến số lượng và loại cột xác định đường dẫn truy cập vào dữ liệu. Vì key của một clustered index trở thành con trỏ cho các nonclustered index, các clustered index key quá rộng có thể ảnh hưởng nghiêm trọng đến hiệu suất và lưu trữ cho các nonclustered index.

**Retrieving Presorted Data**

Clustered index đặc biệt hiệu quả khi việc truy xuất dữ liệu cần được sắp xếp (một nonclustered index cũng rất hữu ích cho việc này). Nếu bạn tạo một clustered index trên cột hoặc các cột mà bạn có thể cần sắp xếp theo, thì các hàng sẽ được lưu trữ theo thứ tự vật lý đó, loại bỏ chi phí sắp xếp dữ liệu sau khi được truy xuất.

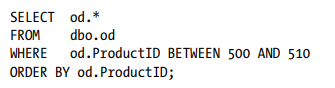
Hãy xem điều này trong hành động. Tạo một bảng thử nghiệm như sau:



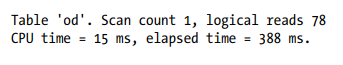
Bảng od mới được tạo chỉ với dữ liệu. Nó không có bất kỳ chỉ mục nào. Bạn có thể xác minh các chỉ mục trên bảng bằng cách thực hiện như sau, không trả về gì cả



Để hiểu việc sử dụng một clustered index, hãy tìm nạp một phạm vi lớn các hàng được sắp xếp trên một cột nhất định.



Bạn có thể có được chi phí thực hiện truy vấn này (không có bất kỳ chỉ mục nào) từ đầu ra IO STATISTICS.



Để cải thiện hiệu suất của truy vấn này, bạn nên tạo một chỉ mục trên cột mệnh đề WHERE. Truy vấn này yêu cầu cả phạm vi hàng và đầu ra được sắp xếp. Yêu cầu thiết lập kết quả của truy vấn này đáp ứng các khuyến nghị cho một clustered index. Do đó, tạo một clustered index như sau và xem xét lại chi phí của truy vấn:

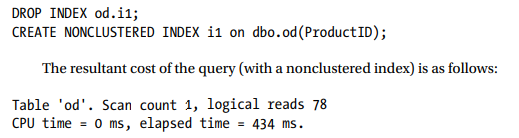


Khi bạn chạy lại truy vấn, chi phí kết quả của truy vấn (với chỉ mục được nhóm) như sau:



Tạo clustered index làm giảm số lần đọc logic và do đó sẽ góp phần cải thiện hiệu suất truy vấn.

Mặt khác, nếu bạn tạo một nonclustered index (thay vì clustered index) trên cột ứng cử viên, thì hiệu suất truy vấn có thể bị ảnh hưởng bất lợi. Hãy để xác minh hiệu quả của một nonclustered index trong trường hợp này



Nonclustered index thậm chí không được sử dụng trực tiếp trong kế hoạch thực hiện kết quả. Thay vào đó, bạn có được quét bảng, nhưng chi phí ước tính để sắp xếp dữ liệu trong kế hoạch mới này khác với quét bảng ban đầu do tính chọn lọc bổ sung mà chỉ mục cung cấp trình tối ưu hóa để ước tính chi phí, mặc dù chỉ mục không được sử dụng . DROP bảng kiểm tra khi bạn làm xong.



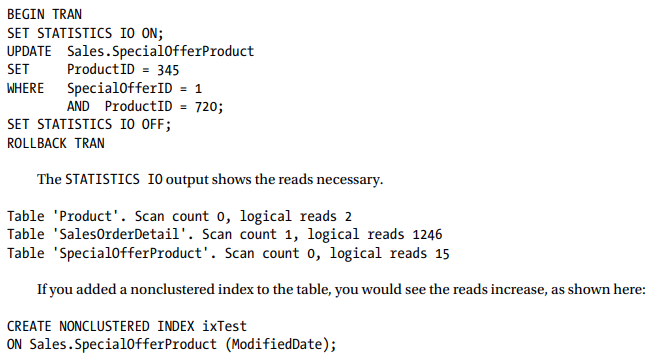
**Poor Design Practices for a Clustered Index**

Trong một số trường hợp nhất định, tốt hơn hết là bạn không sử dụng chỉ mục cụm. Tôi thảo luận về những điều này trong các phần sau.

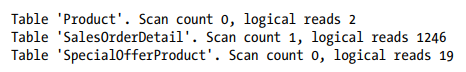
**Frequently Updatable Columns**

Nếu các cột clustered index được cập nhật thường xuyên, điều này sẽ khiến bộ định vị hàng của tất cả các nonclustered index được cập nhật tương ứng, làm tăng đáng kể chi phí của các truy vấn hành động có liên quan. Điều này cũng ảnh hưởng đến sự tương tranh của cơ sở dữ liệu bằng cách chặn tất cả các truy vấn khác đề cập đến cùng một phần của bảng và các nonclustered index trong khoảng thời gian đó. Do đó, tránh tạo clustered index trên các cột có khả năng cập nhật cao.

Để hiểu làm thế nào chi phí của một câu lệnh UPDATE chỉ sửa đổi một clustered key column được tăng lên bởi sự hiện diện của các nonclustered index trên bảng, hãy xem xét ví dụ sau. Bảng Sales.SpecialOfferProduct có một composite clustered index trên khóa chính, cũng là khóa ngoại từ hai bảng khác nhau; đây là một sự tham gia nhiều-nhiều-cổ điển. Trong ví dụ này, tôi cập nhật một trong hai cột bằng cách sử dụng câu lệnh sau (lưu ý việc sử dụng transaction để giữ nguyên dữ liệu thử nghiệm):



Khi bạn chạy lại cùng một truy vấn, đầu ra của STATISTICS IO sẽ thay đổi bảng SpecialOfferProduct



Số lần đọc gây ra bởi việc cập nhật clustered index được tăng lên cùng với việc thêm nonclustered index. Hãy chắc chắn để giảm chỉ số.



**Wide Keys**

Vì tất cả các nonclustered index giữ các clustered key làm công cụ định vị hàng của chúng, vì lý do hiệu suất, bạn nên tránh tạo một nonclustered index trên một cột rất rộng (hoặc cột) hoặc trên quá nhiều cột. Như đã giải thích trong phần trước, một chỉ số được nhóm phải càng hẹp càng tốt.

**Nonclustered Indexes**

Một nonclustered index không ảnh hưởng đến thứ tự của dữ liệu trong các trang của bảng vì các trang lá của một nonclustered index và các trang dữ liệu của bảng là riêng biệt. Một con trỏ (bộ định vị hàng) được yêu cầu để điều hướng từ một index row trong nonclustered index đến data row, cho dù được lưu trữ trên một cụm hoặc trong một heap. Như bạn đã học trong phần trước đó của Cluster Clustered Indexes, cấu trúc của trình định vị hàng phụ thuộc vào việc các trang dữ liệu được lưu trữ trong một heap hay một clustered index. Đối với một heap, bộ định vị hàng là một con trỏ tới RID cho hàng dữ liệu; đối với một bảng có clustered index, bộ định vị hàng là clustered index key

**Nonclustered Index Maintenance**

Giá trị định vị hàng của các nonclustered index tiếp tục có cùng giá trị clustered index, ngay cả khi các clustered index row được di dời về mặt vật lý.

Trong một bảng là một heap, trong đó không có clustered index, để tối ưu hóa chi phí bảo trì này, SQL Server thêm một con trỏ vào trang dữ liệu cũ để trỏ đến trang dữ liệu mới sau khi tách trang, thay vì cập nhật trình định vị hàng của tất cả các nonclustered index có liên quan. Mặc dù điều này làm giảm chi phí bảo trì của các nonclustered index, nhưng nó làm tăng chi phí điều hướng từ nonclustered index row đến hàng dữ liệu trong heap, do một liên kết bổ sung được thêm vào giữa trang dữ liệu cũ và trang dữ liệu mới. Do đó, việc có một clustered index làm công cụ định vị hàng làm giảm chi phí này liên quan đến nonclustered index.

**Defining the Lookup Operation**

Khi một truy vấn yêu cầu các cột không phải là một phần của nonclustered index được chọn bởi trình tối ưu hóa, việc tra cứu là bắt buộc.

Đây có thể là một tra cứu quan trọng khi đi ngược lại với một clustered index hoặc một tra cứu RID khi được thực hiện đối với một heap. Trong quá khứ, thuật ngữ chung cho các tra cứu này xuất phát từ tên định nghĩa cũ, tra cứu dấu trang. Thuật ngữ đó đang được sử dụng ngày càng ít hơn vì mọi người chưa thấy cụm từ đó trong các execution plan kể từ SQL Server 2000. Bây giờ bạn chỉ cần xem nó như một tra cứu và sau đó xác định loại, khóa hoặc RID. Tra cứu tìm nạp hàng dữ liệu tương ứng từ bảng bằng cách theo giá trị của trình định vị hàng từ hàng chỉ mục, yêu cầu đọc logic trên trang dữ liệu bên cạnh việc đọc logic trên trang chỉ mục và thao tác nối để đưa dữ liệu vào cùng một đầu ra chung . Tuy nhiên, nếu tất cả các cột theo yêu cầu của truy vấn đều có sẵn trong chính chỉ mục, thì không cần truy cập vào trang dữ liệu. Điều này được gọi là một chỉ số bao gồm.

Các tra cứu này là lý do mà các tập kết quả lớn được phục vụ tốt hơn với một clustered index. Một clustered index không yêu cầu tra cứu, vì các trang lá và trang dữ liệu cho một clustered index là như nhau.

**Nonclustered Index Recommendations**

Vì một bảng chỉ có thể có một clustered index, bạn có thể sử dụng tính linh hoạt của nhiều nonclustered index để giúp cải thiện hiệu suất.

**When to Use a Nonclustered Index**

Một nonclustered index là hữu ích nhất khi tất cả những gì bạn muốn làm là lấy một số lượng nhỏ hàng và cột từ một bảng lớn. Khi số lượng cột được truy xuất tăng lên, khả năng có covering index sẽ giảm. Sau đó, nếu bạn cũng lấy một số lượng lớn hàng, chi phí cho bất kỳ tra cứu nào cũng tăng theo tỷ lệ thuận. Để lấy một số lượng nhỏ hàng từ một bảng, cột được lập chỉ mục phải có tính chọn lọc cao

Hơn nữa, sẽ có các yêu cầu lập chỉ mục không phù hợp với clustered index, như được giải thích trong phần Clustered Indexes

* Frequently updatable columns
* Wide keys

Trong những trường hợp này, bạn có thể sử dụng một nonclustered index vì không giống như một clustered index, nó không ảnh hưởng đến các chỉ mục khác trong bảng. Một nonclustered index trên một cột có thể cập nhật thường xuyên không tốn kém như có một clustered index trên cột đó. Hoạt động UPDATE trên một chỉ mục không bao gồm được giới hạn trong bảng cơ sở và chỉ mục không bao gồm. Nó không ảnh hưởng đến bất kỳ chỉ mục không bao gồm nào khác trên bảng. Tương tự, một clustered index trên một cột rộng (hoặc tập hợp các cột) không làm tăng kích thước của bất kỳ chỉ mục nào khác, không giống như chỉ mục được nhóm. Tuy nhiên, vẫn thận trọng, ngay cả khi tạo nonclustered index trên cột có khả năng cập nhật cao hoặc cột rộng (hoặc bộ cột), vì điều này có thể làm tăng chi phí của các truy vấn hành động, như đã giải thích trước đó trong chương.

**When Not to Use a Nonclustered Index**

Nonclustered indexes gồm không phù hợp cho các truy vấn truy xuất một số lượng lớn các hàng. Các truy vấn như vậy được phục vụ tốt hơn với một clustered index, điều này không yêu cầu một tra cứu riêng để lấy một hàng dữ liệu. Do việc tra cứu yêu cầu các lần đọc logic bổ sung để đến trang dữ liệu bên cạnh việc đọc logic trên trang chỉ mục không bao gồm, nên chi phí của một truy vấn sử dụng nonclustered index tăng đáng kể cho một số lượng lớn các hàng, chẳng hạn như khi tham gia vòng lặp yêu cầu hết lần này đến lần khác Trình tối ưu hóa truy vấn SQL Server có hiệu lực chi phí này và do đó có thể loại bỏ nonclustered index khi truy xuất một tập kết quả lớn.

Nếu yêu cầu của bạn là truy xuất một tập kết quả lớn từ một bảng, thì việc có một nonclustered index trên tiêu chí bộ lọc (hoặc tiêu chí tham gia) có thể sẽ không hữu ích trừ khi bạn sử dụng một loại nonclustered index đặc biệt được gọi là chỉ mục che phủ. Tôi mô tả loại chỉ số này một cách chi tiết trong Chương 9.

**Clustered vs. Nonclustered Indexes**

Các cân nhắc chính trong việc lựa chọn giữa một chỉ số được nhóm và không bao gồm như sau:

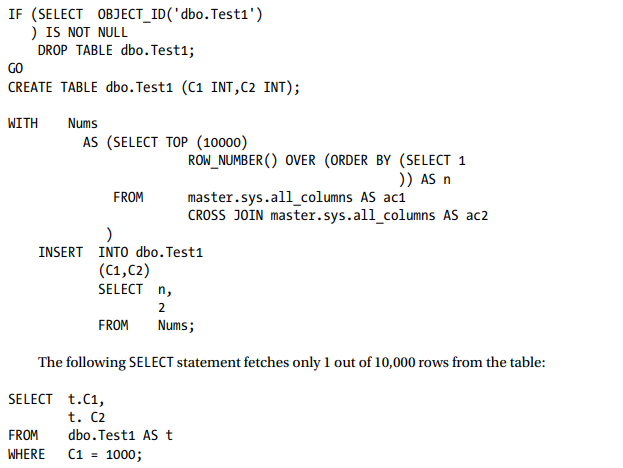
* Number of rows to be retrieved
* Data-ordering requirement
* Index key width
* Column update frequency
* Lookup cost
* Any disk hot spots

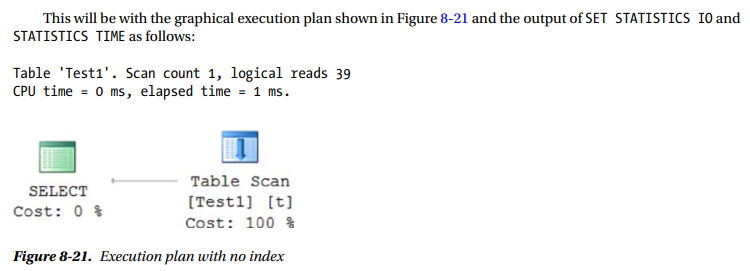
**Benefits of a Clustered Index over a Nonclustered Index**

Khi quyết định chọn một loại chỉ mục trên bảng không có chỉ mục, clustered index thường là lựa chọn ưu tiên.

Do index page và the data pages giống nhau, nên clustered index không phải nhảy từ index row sang base row như được yêu cầu trong trường hợp nonclustered index.

Để hiểu làm thế nào một clustered index có thể vượt trội hơn một nonclustered index trong các trường hợp này, ngay cả khi truy xuất một số lượng nhỏ hàng, hãy tạo một bảng thử nghiệm có độ chọn lọc cao cho một cột.

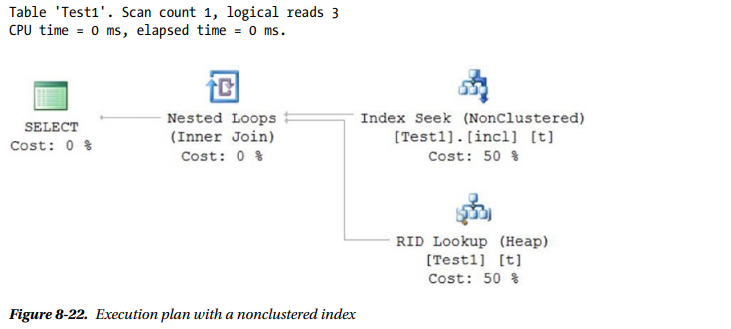




Xem xét kích thước nhỏ của tập kết quả được lấy bởi câu lệnh SELECT trước đó, một nonclustered column trên c1 có thể là một lựa chọn tốt.



Bạn có thể chạy lại lệnh SELECT tương tự. Do việc truy xuất một số lượng nhỏ các hàng thông qua một chỉ mục không bao gồm sẽ kinh tế hơn so với quét bảng, trình tối ưu hóa đã sử dụng chỉ mục không bao gồm trên cột c1, như trong Hình 8-22. Số lần đọc logic được báo cáo bởi STATISTICS IO như sau:

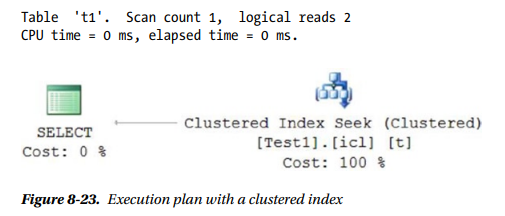


Mặc dù việc truy xuất một tập kết quả nhỏ bằng cách sử dụng một cột có độ chọn lọc cao là một con trỏ tốt để tạo ra một nonclustered index trên cột, một clustered index trên cùng một cột có thể có lợi như nhau hoặc thậm chí tốt hơn.

Để đánh giá làm thế nào clustered index có thể có lợi hơn so với nonclustered index, hãy tạo clustered index trên cùng một cột.



Chạy lại lệnh SELECT tương tự. Từ kế hoạch thực hiện kết quả (xem Hình 8-22) của câu lệnh SELECT trước đó, bạn có thể thấy rằng trình tối ưu hóa đã sử dụng chỉ mục được nhóm (thay vì chỉ mục không bao gồm) ngay cả đối với một tập kết quả nhỏ. Số lần đọc logic cho câu lệnh SELECT giảm từ ba xuống còn hai (Hình 8-23).



**Benefits of a Nonclustered Index over a Clustered Index**

* When the index key size is large.
* Để tránh chi phí cao liên quan đến một clustered index kể từ khi xây dựng lại clustered index,xây dựng lại tất cả các nonclustered index của bảng.
* Để giải quyết việc chặn bằng cách có trình đọc cơ sở dữ liệu hoạt động trên các trang của chỉ mục không được bao gồm, trong khi trình ghi cơ sở dữ liệu sửa đổi các cột khác (không được bao gồm trong nonclustered index) trong trang dữ liệu, trong trường hợp này, người viết làm việc trên trang dữ liệu sẽ không chặn một trình đọc có thể nhận được tất cả các giá trị cột cần thiết từ chỉ mục không bao gồm mà không cần nhấn vào bảng cơ sở. Tôi sẽ giải thích điều này một cách chi tiết trong Chương 13.
* Khi tất cả các cột (từ một bảng) được tham chiếu bởi một truy vấn có thể được cung cấp một cách an toàn trong chính nonclustered index, như được giải thích trong phần này.

**Blocking and Blocked Processes**

Bạn lý tưởng muốn ứng dụng cơ sở dữ liệu của mình mở rộng tuyến tính với số lượng người dùng cơ sở dữ liệu và khối lượng dữ liệu. Tuy nhiên, người ta thường thấy rằng hiệu suất giảm khi số lượng người dùng tăng lên và khi khối lượng dữ liệu tăng lên. Một nguyên nhân cho sự xuống cấp, đặc biệt là liên quan đến quy mô ngày càng tăng, là blocking. Trong thực tế, database blocking thường là một trong những kẻ thù lớn nhất về khả năng mở rộng cho các ứng dụng cơ sở dữ liệu.

Trong chương này, tôi đề cập đến các chủ đề sau:

• Nguyên tắc cơ bản của việc bloking trong SQL Server

• Các thuộc tính ACID của transactional database

• Database lock granularity, escalation, modes, and compatibility

• Mức cô lập ANSI

• Ảnh hưởng của các chỉ số đến khóa

• Thông tin cần thiết để phân tích blocking

• SQL script để thu thập thông tin chặn

• Nghị quyết và khuyến nghị để tránh chặn

• Kỹ thuật tự động hóa quá trình phát hiện và thu thập thông tin blocking

**Blocking Fundamentals**

Trong một thế giới lý tưởng, mọi truy vấn SQL sẽ có thể thực hiện đồng thời mà không bị chặn bởi các truy vấn khác.

Tuy nhiên, trong thế giới thực, các truy vấn chặn nhau, tương tự như cách một chiếc ô tô đi qua tín hiệu giao thông màu xanh lá cây tại một giao lộ chặn các xe khác đang chờ để băng qua ngã tư. Trong SQL Server, quản lý lưu lượng này có dạng lock manager, điều khiển truy cập đồng thời vào tài nguyên cơ sở dữ liệu để duy trì tính nhất quán của dữ liệu. Quyền truy cập đồng thời vào tài nguyên cơ sở dữ liệu được kiểm soát trên nhiều kết nối cơ sở dữ liệu

Tôi muốn đảm bảo mọi thứ rõ ràng trước khi tiếp tục. Ba thuật ngữ được sử dụng trong các cơ sở dữ liệu nghe có vẻ giống nhau và có liên quan đến nhau nhưng có ý nghĩa khác nhau. Đây thường là nhầm lẫn, và mọi người thường sử dụng các thuật ngữ không chính xác. Các điều khoản này là locking, blocking, and deadlocking. Locking là một phần không thể thiếu trong quá trình SQL Server quản lý nhiều phiên. Khi một phiên cần truy cập vào một phần dữ liệu, một loại khóa nào đó được đặt trên đó. Điều này khác với việc chặn, đó là khi một phiên hoặc luồng, cần truy cập vào một phần dữ liệu và phải chờ khóa của phiên khác clear. Cuối cùng, deadlocking là khi hai phiên, hoặc thread, hình thành những gì đôi khi được gọi là một cái ôm chết người. Họ đang chờ đợi nhau để khóa clear. Deadlock cũng có thể được gọi là một tình huống chặn vĩnh viễn, nhưng nó không thể giải quyết bằng cách chờ đợi bất kỳ khoảng thời gian nào.

Deadlocking sẽ được đề cập chi tiết hơn trong Chương 21. Vì vậy, lock có thể dẫn đến các blocks và cả lock và block đóng một phần trong các deadlock, nhưng đây là ba khái niệm rất khác biệt. Hãy hiểu sự khác biệt giữa các điều khoản này và sử dụng chúng một cách chính xác. Nó sẽ giúp bạn hiểu về hệ thống, khả năng khắc phục sự cố và khả năng giao tiếp với các quản trị viên và nhà phát triển cơ sở dữ liệu khác.

Trong SQL Server, kết nối cơ sở dữ liệu được xác định bởi session ID. Các kết nối có thể từ một hoặc nhiều ứng dụng và một hoặc nhiều người dùng trên các ứng dụng đó; theo như SQL Server, mọi kết nối được coi là một phiên riêng biệt. Bloking giữa hai session truy cập cùng một phần dữ liệu cùng một lúc là một hiện tượng tự nhiên trong SQL Server. Bất cứ khi nào hai session cố gắng truy cập tài nguyên cơ sở dữ liệu chung theo các cách xung đột, trình quản lý khóa đảm bảo rằng phiên thứ hai sẽ đợi cho đến khi phiên đầu tiên hoàn thành công việc kết hợp với việc quản lý các transaction trong hệ thống. Ví dụ: một session có thể sửa đổi table record trong khi phiên khác cố xóa bản ghi. Vì hai yêu cầu truy cập dữ liệu này không tương thích, phiên thứ hai sẽ bị chặn cho đến khi phiên đầu tiên hoàn thành nhiệm vụ của nó.

Mặt khác, nếu hai phiên cố gắng đọc đồng thời một bảng, cả hai yêu cầu đều được phép thực hiện mà không bị chặn, vì các yêu cầu truy cập dữ liệu này tương thích với nhau.

Thông thường, hiệu ứng bloking trên một phiên khá nhỏ và không ảnh hưởng đến hiệu suất của nó. Tuy nhiên, đôi khi, do truy vấn kém và / hoặc thiết kế transaction (hoặc có thể không may mắn), việc blocking có thể ảnh hưởng đáng kể đến hiệu suất truy vấn. Trong một ứng dụng cơ sở dữ liệu, mọi nỗ lực nên được thực hiện để giảm thiểu việc blocking và do đó tăng số lượng người dùng đồng thời có thể sử dụng cơ sở dữ liệu.

Với việc giới thiệu các bảng trong bộ nhớ trong SQL Server 2014, việc khóa, ít nhất là đối với các bảng này, sẽ có các kích thước hoàn toàn mới. Tôi sẽ bao gồm hành vi của họ một cách riêng biệt trong Chương 23.

**Understanding Blocking**

Trong SQL Server, một truy vấn cơ sở dữ liệu có thể tự thực hiện như một đơn vị công việc logic hoặc nó có thể tham gia vào một đơn vị công việc logic lớn hơn. Một đơn vị công việc logic lớn hơn có thể được xác định bằng cách sử dụng câu lệnh BEGIN TRANSACTION cùng với các câu lệnh COMIT và / hoặc ROLLBACK. Mỗi đơn vị công việc logic phải tuân theo một bộ bốn thuộc tính được gọi là Thuộc tính ACID:

• Atomicity

• Consistency

• Isolation

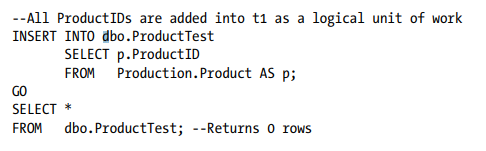
• Durability

Tôi bao gồm các thuộc tính này trong các phần tiếp theo bởi vì hiểu cách thức transaction hoạt động là cơ bản để hiểu cách blocking.

**Atomicity**

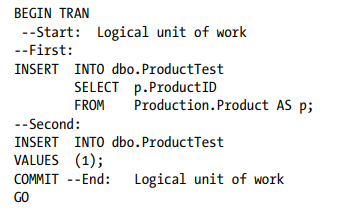
Một đơn vị logic của công việc phải là nguyên tử. Đó là, hoặc tất cả các hành động của đơn vị công việc logic được hoàn thành hoặc không có hiệu lực được giữ lại. Để hiểu tính nguyên tử của một đơn vị công việc hợp lý, hãy xem xét ví dụ sau:





SQL Server coi câu lệnh INSERT trước đó là một đơn vị công việc logic. Ràng buộc CHECK trên cột ProductID của bảng dbo. ProducttTest chỉ cho phép giá trị là 1. Mặc dù cột ProductID trong bảng Production.Product bắt đầu bằng giá trị 1, nhưng nó cũng chứa các giá trị khác. Vì lý do này, câu lệnh INSERT sẽ không thêm bất kỳ bản ghi nào vào bảng dbo. ProducttTest và một lỗi được đưa ra do ràng buộc CHECK. Tính nguyên tử này được SQL Server tự động đảm bảo.

Càng xa càng tốt. Nhưng trong trường hợp đơn vị công việc logic lớn hơn, bạn nên biết về một hành vi thú vị của SQL Server. Hãy tưởng tượng rằng tác vụ chèn trước đó bao gồm nhiều câu lệnh INSERT. Chúng có thể được kết hợp để tạo thành một đơn vị công việc logic lớn hơn, như sau:



Với bảng dbo. ProducttTest đã được tạo trong tập lệnh trước, cặp câu lệnh BEGIN TRAN và Commit xác định một đơn vị công việc logic, cho thấy rằng tất cả các câu lệnh trong transaction phải có tính nguyên tử. Tuy nhiên, hành vi mặc định của SQL Server không đảm bảo rằng sự thất bại của một trong các câu lệnh trong phạm vi transaction do người dùng xác định sẽ hoàn tác tác động của (các) câu lệnh trước. Trong transaction trước, câu lệnh INSERT đầu tiên sẽ thất bại như được giải thích trước đó, trong khi câu lệnh INSERT thứ hai hoàn toàn ổn. Hành vi mặc định của SQL Server cho phép câu lệnh INSERT thứ hai thực thi, mặc dù câu lệnh INSERT đầu tiên không thành công. Một câu lệnh SELECT, như được hiển thị trong đoạn mã sau, sẽ trả về hàng được chèn bởi câu lệnh INSERT thứ hai:



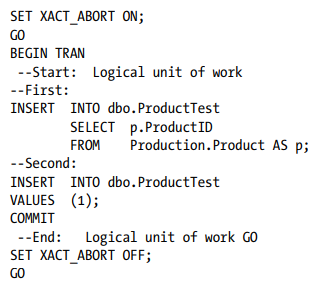
Tính nguyên tử của transaction do người dùng xác định có thể được đảm bảo theo hai cách sau:

• SET XACT\_ABORT ON

• Explicit rollback

**SET XACT\_ABORT ON**

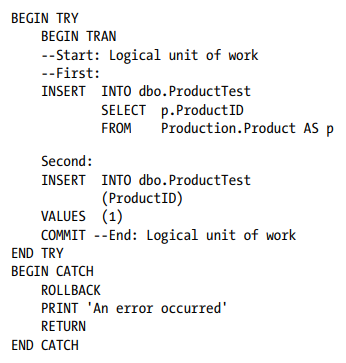
Bạn có thể sửa đổi tính nguyên tử của tác vụ INSERT trong phần trước bằng cách sử dụng câu lệnh SET XACT\_ABORT ON:



Câu lệnh SET XACT\_ABORT chỉ định liệu SQL Server có nên tự động quay lại và hủy bỏ toàn bộ transaction hay không khi một câu lệnh trong transaction không thành công. Thất bại của câu lệnh INSERT đầu tiên sẽ tự động tạm dừng toàn bộ transaction và do đó, câu lệnh INSERT thứ hai sẽ không được thực thi. Ảnh hưởng của SET XACT\_ABORT ở cấp kết nối và nó vẫn được áp dụng cho đến khi được cấu hình lại hoặc kết nối bị đóng. Theo mặc định, SET XACT\_ABORT OFF

**Explicit Rollback**

Bạn cũng có thể quản lý tính nguyên tử của transaction do người dùng xác định bằng cách sử dụng cơ chế bẫy lỗi TRY / CATCH trong SQL Server. Nếu một câu lệnh trong khối mã TRY tạo ra lỗi, thì khối mã CATCH sẽ xử lý lỗi. Nếu xảy ra lỗi và khối CATCH được kích hoạt, thì toàn bộ công việc của transaction do người dùng xác định có thể được khôi phục và có thể ngăn chặn các báo cáo tiếp theo từ thực thi, như sau:



Câu lệnh ROLLBACK cuộn lại tất cả các hành động được thực hiện trong transaction cho đến thời điểm đó. Để biết mô tả chi tiết về cách triển khai xử lý lỗi trong các ứng dụng dựa trên SQL Server, vui lòng tham khảo bài viết của Thư viện MSDN có tiêu đề “Using TRY . . . CATCH in Transact SQL” (http://bit.ly/PNlAHF) hoặc bài viết giới thiệu có tiêu đề "SQL Server Error Handling Workbench “ (http://bit.ly/1nC3VBt). Vì thuộc tính nguyên tử yêu cầu tất cả các các hành động của một đơn vị công việc logic được hoàn thành hoặc không có tác dụng nào được giữ lại, SQL Server tách biệt công việc của một transaction với các transaction khác bằng cách cấp cho nó quyền độc quyền đối với các tài nguyên bị ảnh hưởng. Điều này có nghĩa là transaction có thể khôi phục một các an toàn tất cả tác động từ hành động của nó, nếu được yêu cầu. Các quyền độc quyền được cấp cho một transaction trên các tài nguyên bị ảnh hưởng sẽ chặn tất cả các transaction khác (hoặc yêu cầu cơ sở dữ liệu) cố gắng truy cập các tài nguyên đó trong khoảng thời gian đó. Do đó, mặc dù nguyên tử là bắt buộc để duy trì tính toàn vẹn của dữ liệu, nó đã đưa ra tác dụng phụ không mong muốn của blocking

**Consistency**

Một đơn vị công việc logic sẽ làm cho trạng thái của cơ sở dữ liệu đi từ trạng thái nhất quán này sang trạng thái nhất quán khác. Khi kết thúc một transaction, trạng thái của cơ sở dữ liệu phải hoàn toàn nhất quán. SQL Server luôn đảm bảo rằng trạng thái bên trong của cơ sở dữ liệu là chính xác và hợp lệ bằng cách tự động áp dụng tất cả các ràng buộc của tài nguyên cơ sở dữ liệu bị ảnh hưởng như một phần của transaction. SQL Server đảm bảo rằng trạng thái của các cấu trúc bên trong, chẳng hạn như bố cục dữ liệu và chỉ mục, là chính xác sau khi transaction. Chẳng hạn, khi dữ liệu của bảng được sửa đổi, SQL Server sẽ tự động xác định tất cả các chỉ mục, các ràng buộc và các đối tượng phụ thuộc khác trên bảng và áp dụng các sửa đổi cần thiết cho tất cả các đối tượng cơ sở dữ liệu phụ thuộc như một phần của transaction.

Tính nhất quán hợp lý của dữ liệu theo yêu cầu của quy tắc kinh doanh phải được đảm bảo bởi nhà phát triển cơ sở dữ liệu.

Một quy tắc kinh doanh có thể yêu cầu thay đổi được áp dụng trên nhiều bảng. Nhà phát triển cơ sở dữ liệu nên xác định một đơn vị công việc hợp lý để đảm bảo rằng tất cả các tiêu chí của quy tắc kinh doanh được quan tâm. SQL Server cung cấp các tính năng quản lý transaction khác nhau mà nhà phát triển cơ sở dữ liệu có thể sử dụng để đảm bảo tính thống nhất logic của dữ liệu.

Như đã giải thích, việc duy trì trạng thái logic nhất quán đòi hỏi phải sử dụng các transaction để xác định đơn vị công việc logic theo quy tắc kinh doanh. Ngoài ra, để duy trì trạng thái vật lý nhất quán, SQL Server xác định và hoạt động trên các đối tượng cơ sở dữ liệu phụ thuộc như một phần của đơn vị công việc logic. Đặc tính nguyên tử của đơn vị công việc logic chặn tất cả các transaction khác (hoặc yêu cầu cơ sở dữ liệu) cố gắng truy cập các đối tượng bị ảnh hưởng trong khoảng thời gian đó.

Do đó, mặc dù tính nhất quán là cần thiết để duy trì trạng thái logic và vật lý hợp lệ của cơ sở dữ liệu, nó cũng đưa ra tác dụng phụ không mong muốn của việc blocking.

**Isolation**

Trong môi trường nhiều người dùng, nhiều transaction có thể được thực hiện đồng thời. Các transaction đồng thời này nên được cách ly với nhau để các thay đổi trung gian được thực hiện bởi một transaction không ảnh hưởng đến tính nhất quán dữ liệu của các transaction khác. Mức độ cô lập được yêu cầu bởi một transaction có thể khác nhau. SQL Server cung cấp các tính năng cách ly transaction khác nhau để thực hiện mức độ cô lập theo yêu cầu của transaction.

Các yêu cầu cách ly của một transaction hoạt động trên tài nguyên cơ sở dữ liệu có thể chặn các transaction khác cố gắng truy cập tài nguyên. Trong môi trường cơ sở dữ liệu nhiều người dùng, nhiều transaction thường được thực hiện đồng thời. Điều bắt buộc là các sửa đổi dữ liệu được thực hiện bởi một transaction đang diễn ra phải được bảo vệ khỏi các sửa đổi được thực hiện bởi các transaction khác. Chẳng hạn, giả sử một transaction ở giữa sửa đổi một vài hàng trong bảng. Trong khoảng thời gian đó, để duy trì tính nhất quán của cơ sở dữ liệu, bạn phải đảm bảo rằng các transaction khác không sửa đổi hoặc xóa cùng một hàng. SQL Server cách ly một cách hợp lý các hoạt động của một transaction với các transaction khác bằng cách chặn chúng một cách thích hợp, cho phép nhiều transaction thực hiện đồng thời mà không làm hỏng một công việc khác.

Blocking quá mức gây ra bởi sự cô lập có thể ảnh hưởng xấu đến khả năng mở rộng của ứng dụng cơ sở dữ liệu. Một transaction có thể vô tình chặn các transaction khác trong một thời gian dài, do đó làm tổn hại đến sự tương tranh của cơ sở dữ liệu. Vì SQL Server quản lý sự cô lập bằng cách sử dụng các khóa, điều quan trọng là phải hiểu kiến trúc khóa của SQL Server. Điều này giúp bạn phân tích một kịch bản chặn và thực hiện các nghị quyết.

**Durability**

Khi một transaction được hoàn thành, những thay đổi được thực hiện bởi transaction sẽ rất bền. Ngay cả khi nguồn điện cho máy bị ngắt ngay lập tức sau khi transaction được hoàn thành, hiệu ứng của tất cả các hành động trong transaction vẫn được giữ lại. SQL Server đảm bảo độ bền bằng cách theo dõi tất cả các hình ảnh trước và sau của dữ liệu được sửa đổi trong nhật ký transaction khi các thay đổi được thực hiện. Ngay sau khi hoàn thành transaction, SQL Server đảm bảo rằng tất cả các thay đổi được thực hiện trong transaction đều được giữ lại ngay cả khi SQL Server, hệ điều hành hoặc phần cứng bị lỗi (không bao gồm đĩa nhật ký). Trong quá trình khởi động lại, SQL Server chạy tính năng khôi phục cơ sở dữ liệu của nó, xác định các thay đổi đang chờ xử lý từ nhật ký transaction cho các transaction đã hoàn thành và áp dụng chúng cho các tài nguyên cơ sở dữ liệu. Tính năng cơ sở dữ liệu này được gọi là roll forward.

Khoảng thời gian phục hồi phụ thuộc vào số lượng thay đổi đang chờ xử lý cần được áp dụng cho tài nguyên cơ sở dữ liệu trong khi khởi động lại. Để giảm khoảng thời gian phục hồi, SQL Server không ngừng áp dụng các thay đổi trung gian được thực hiện bởi các transaction đang chạy như được định cấu hình bởi tùy chọn khoảng thời gian khôi phục. Tùy chọn khoảng thời gian phục hồi có thể được cấu hình bằng cách sử dụng câu lệnh spconfigure. Quá trình áp dụng không liên tục các thay đổi trung gian được gọi là quá trình checkpoint. Trong quá trình khởi động lại, quá trình khôi phục xác định tất cả các thay đổi không được cam kết và xóa chúng khỏi tài nguyên cơ sở dữ liệu bằng cách sử dụng hình ảnh trước của dữ liệu từ nhật ký transaction.

Thuộc tính độ bền không phải là nguyên nhân trực tiếp của hầu hết các trường hợp blocking vì nó không yêu cầu các hành động của transaction phải được cách ly với các transaction khác. Nhưng theo một cách gián tiếp, nó làm tăng thời gian blocking. Vì thuộc tính độ bền yêu cầu lưu hình ảnh trước và sau của dữ liệu được sửa đổi vào nhật ký transaction trên đĩa, nó làm tăng thời lượng của transaction và blocking

Được giới thiệu trong SQL Server 2014 là khả năng giảm độ trễ, thời gian chờ truy vấn để cam kết và ghi vào nhật ký, bằng cách sửa đổi hành vi độ bền của cơ sở dữ liệu đã cho. Bây giờ bạn có thể sử dụng độ bền chậm. Điều này có nghĩa là khi một transaction hoàn thành, nó báo cáo ngay cho ứng dụng là một transaction thành công, giảm độ trễ.

Nhưng việc ghi vào nhật ký vẫn chưa xảy ra. Điều này cũng có thể cho phép nhiều transaction được hoàn thành trong khi vẫn chờ hệ thống ghi tất cả đầu ra vào nhật ký transaction. Mặc dù điều này có thể làm tăng tốc độ rõ ràng trong hệ thống, cũng như có thể làm giảm sự tranh chấp trong I / O nhật ký transaction, nhưng nó vốn là một lựa chọn nguy hiểm. Đây là một khuyến nghị khó thực hiện. Microsoft gợi ý ba tình huống có thể khiến nó hấp dẫn.

Note: Trong số bốn thuộc tính ACID, thuộc tính cách ly, cũng được sử dụng để đảm bảo tính nguyên tử và tính nhất quán, là nguyên nhân chính của việc chặn trong cơ sở dữ liệu SQL Server. Trong SQL Server, sự cô lập được thực hiện bằng cách sử dụng các khóa, như được giải thích trong phần tiếp theo.

**Locks**

Khi một phiên thực hiện một truy vấn, SQL Server xác định tài nguyên cơ sở dữ liệu cần được truy cập; và, nếu được yêu cầu, trình quản lý khóa cấp các loại khóa khác nhau cho phiên. Truy vấn bị chặn nếu phiên khác đã được cấp khóa; tuy nhiên, để cung cấp cả cách ly transaction và đồng thời, SQL Server sử dụng các mức độ chi tiết khóa khác nhau, như được giải thích trong các phần sau

**Lock Granularity**

Cơ sở dữ liệu SQL Server được duy trì dưới dạng tệp trên đĩa vật lý. Trong trường hợp tệp không phải là cơ sở dữ liệu truyền thống, chẳng hạn như tệp Excel trên máy tính để bàn, tệp chỉ có thể được ghi bởi một người dùng tại một thời điểm. Bất kỳ nỗ lực nào để ghi vào tệp bởi người dùng khác đều thất bại. Tuy nhiên, không giống như giới hạn đồng thời trên một tệp không có cơ sở dữ liệu, SQL Server cho phép nhiều người dùng sửa đổi (hoặc truy cập) nội dung cùng một lúc, miễn là họ không có ảnh hưởng đến tính nhất quán của dữ liệu khác. Điều này làm giảm việc chặn và cải thiện sự tương tranh giữa các transaction.

Để cải thiện sự tương tranh, SQL Server triển khai các mức độ chi tiết khóa ở các mức tài nguyên sau và theo thứ tự sau:

• Row (RID)

• Key (KEY)

• Page (PAG)

• Extent (EXT)

• Heap or B-tree (HoBT)

• Table (TAB)

• File (FIL)

• Application (APP)

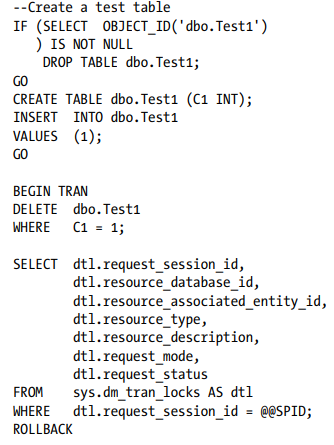
• MetaData (MDT)

• Allocation Unit (AU)

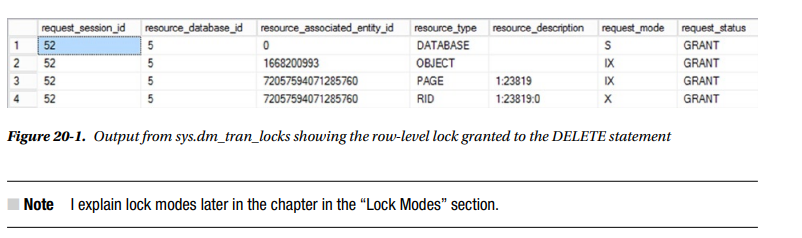
• Database (DB)

**Row-Level Lock**

Khóa này được duy trì trên một hàng trong bảng và là mức khóa thấp nhất trên bảng cơ sở dữ liệu. Khi truy vấn sửa đổi một hàng trong bảng, khóa RID được cấp cho truy vấn trên hàng. Ví dụ: xem xét transaction trên bảng thử nghiệm sau:



Chế độ xem quản lý động, sys.dm\_tran\_locks, có thể được sử dụng để hiển thị trạng thái khóa. Truy vấn đối với sys.dm\_tran\_locks trong Hình 20-1 cho thấy câu lệnh DELETE thu được, trong số các khóa khác, một khóa RID độc quyền trên hàng sẽ bị xóa.



Cấp khóa RID cho câu lệnh DELETE ngăn các transaction khác truy cập vào hàng.



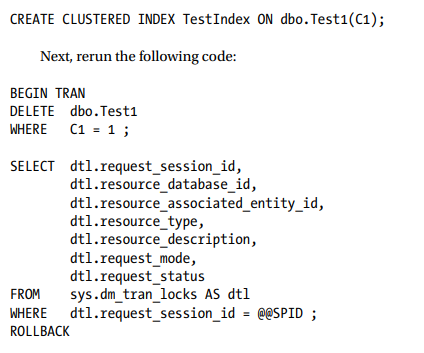
Trong đầu ra từ truy vấn đối với sys.dm\_tran\_locks trong Hình 20-1, DatabaseID được hiển thị riêng dưới cột resource\_database\_id. Giá trị cột resource\_description cho loại RID đại diện cho phần còn lại của tài nguyên RID là 1: 23819: 0. Trong trường hợp này, FileID của 1 là tệp dữ liệu chính, PageID của 23819 là một trang thuộc bảng dbo.Test1 được xác định bởi cột C1 và một Slot (hàng) 0 đại diện cho vị trí hàng trong trang. Bạn có thể lấy tên bảng và tên cơ sở dữ liệu bằng cách thực hiện các câu lệnh SQL sau:



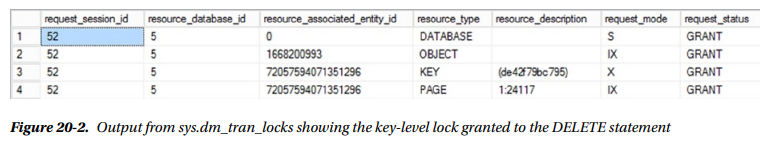
Row-level lock cung cấp đồng thời rất cao do việc blocking bị hạn chế ở hàng có hiệu lực.

**Key-Level Lock**

Đây là một row lock trong một chỉ mục và nó được xác định là khóa KEY. Như bạn đã biết, đối với một bảng có clustered index, các trang dữ liệu của bảng và các trang lá của clustered index là như nhau. Vì cả hai hàng đều giống nhau cho một bảng có clustered index, nên chỉ có khóa KEY trên hàng clustered index hoặc phạm vi hàng giới hạn trong khi truy cập các hàng từ bảng (hoặc chỉ mục được nhóm). Ví dụ, hãy xem xét có một chỉ mục được nhóm trên bảng Test1.



Đầu ra tương ứng từ sys.dm\_tran\_locks hiển thị khóa KEY thay vì khóa RID, như bạn có thể thấy trong Hình 20-2.



Khi bạn đang truy vấn sys.dm\_tran\_locks, bạn sẽ có thể truy xuất định danh cơ sở dữ liệu, resource\_ database\_id. Bạn cũng có thể nhận thông tin về những gì đang bị khóa từ resource\_associated\_entity\_id; tuy nhiên, để có được tài nguyên cụ thể (trong trường hợp này là trang trên key), bạn phải chuyển đến cột resource\_description cho giá trị, là 1: 24117. Trong trường hợp này, ID chỉ mục của 1 là clustered index trên bảng dbo.Test1. Bạn cũng thấy các loại yêu cầu được thực hiện: S, Sch-S, X, v.v. Tôi trình bày những điều này chi tiết hơn trong phần “Lock Modes” sắp tới.

Giống như row-level lock, the key-level lock cung cấp tính đồng thời rất cao

**Page-Level Lock**

Khóa cấp độ trang được duy trì trên một trang trong bảng hoặc chỉ mục và được xác định là khóa PAG. Khi truy vấn yêu cầu nhiều hàng trong một trang, tính nhất quán của tất cả các hàng được yêu cầu có thể được duy trì bằng cách lấy khóa RID / KEY trên các hàng riêng lẻ hoặc khóa PAG trên toàn bộ trang. Từ kế hoạch truy vấn, trình quản lý khóa xác định áp lực tài nguyên khi có được nhiều khóa RID / KEY và nếu áp lực cao, lock manager yêu cầu PAG thay thế.

Tài nguyên bị khóa bởi khóa PAG có thể được biểu diễn theo định dạng sau trong cột resource\_description của sys.dm\_tran\_locks.



Khóa cấp độ trang có thể tăng hiệu suất của một truy vấn riêng lẻ bằng cách giảm chi phí khóa của nó, nhưng nó làm tổn hại đến tính đồng thời của cơ sở dữ liệu bằng cách chặn truy cập vào tất cả các hàng trong trang.

**Extent-Level Lock**

Extent-level lock được duy trì ở mức độ (một nhóm gồm tám trang dữ liệu liền kề hoặc index pages) và được xác định là khóa EXT. Khóa này được sử dụng, ví dụ, khi lệnh ALTER INDEX REBUILD được thực thi trên một bảng và các trang của bảng có thể được chuyển từ một phạm vi hiện tại sang một phạm vi mới. Trong giai đoạn này, tính toàn vẹn của các khu vực được bảo vệ bằng các khóa EXT.

**Heap or B-tree Lock**

Heap hoặc B-tree lock được sử dụng để mô tả khi khóa đến loại khác của đối tượng có thể được thực hiện. Đối tượng đích có thể là một bảng heap không có thứ tự, một bảng không clustered index hoặc đối tượng B-tree, thường đề cập đến các phân vùng. Một cài đặt trong chức năng ALTER TABLE cho phép bạn thực hiện một mức độ kiểm soát đối với việc leo thang khóa (được bao phủ trong phần “Lock Escalation” khóa) bị ảnh hưởng với các phân vùng. Bởi vì các phân vùng có thể nằm trên nhiều nhóm file, mỗi nhóm phải có định nghĩa phân bổ dữ liệu riêng. Đây là nơi khóa HoBT phát huy tác dụng. Nó hoạt động như một khóa cấp bảng nhưng trên một phân vùng thay vì trên chính bảng.

**Table-Level Lock**

Đây là mức khóa cao nhất trên bảng và được xác định là khóa TAB. Khóa cấp bảng trên bảng dự trữ quyền truy cập vào bảng hoàn chỉnh và tất cả các chỉ mục của bảng.

Khi một truy vấn được thực thi, trình quản lý khóa sẽ tự động xác định chi phí khóa của việc có được nhiều khóa ở các cấp thấp hơn. Nếu áp lực tài nguyên của việc thu được các khóa ở cấp hàng hoặc cấp trang được xác định là cao, thì trình quản lý khóa sẽ trực tiếp lấy khóa cấp bảng cho truy vấn.

Tài nguyên bị khóa bởi khóa TAB sẽ được trình bày dưới dạng resource\_description theo định dạng sau:



Khóa cấp bảng yêu cầu ít chi phí nhất so với các khóa khác và do đó cải thiện hiệu suất của truy vấn riêng lẻ. Mặt khác, vì khóa cấp bảng chặn tất cả các yêu cầu ghi trên toàn bộ bảng (bao gồm các chỉ mục), nên nó có thể làm tổn hại đáng kể đến sự tương tranh của cơ sở dữ liệu.

Đôi khi một tính năng ứng dụng có thể được hưởng lợi từ việc sử dụng một mức khóa cụ thể cho một bảng được đề cập trong một truy vấn.

Chẳng hạn, nếu một truy vấn quản trị được thực thi trong những giờ không thường xuyên, thì khóa cấp bảng có thể không ảnh hưởng quá nhiều đến người dùng hệ thống; tuy nhiên, nó có thể giảm chi phí khóa của truy vấn và do đó cải thiện hiệu suất của nó. Trong các trường hợp như vậy, nhà phát triển truy vấn có thể ghi đè lên lựa chọn cấp khóa của Trình quản lý khóa cho một bảng được đề cập trong truy vấn bằng cách sử dụng các gợi ý khóa.



Nhưng, hãy thận trọng khi lấy quyền kiểm soát khỏi SQL Server như thế này. Kiểm tra kỹ lưỡng trước khi thực hiện

**Database-Level Lock**

Khóa cấp cơ sở dữ liệu được duy trì trên cơ sở dữ liệu và được xác định là khóa DB. Khi một ứng dụng tạo kết nối cơ sở dữ liệu, trình quản lý khóa sẽ gán database-level shared lock cho session\_id tương ứng. Điều này ngăn người dùng vô tình dropping hoặc restoring cơ sở dữ liệu trong khi những người dùng khác được kết nối với nó.

SQL Server đảm bảo rằng các khóa được yêu cầu ở một cấp tôn trọng các khóa được cấp ở các cấp khác. Chẳng hạn, một khi người dùng có được row-level lock trên một hàng của bảng, một người dùng khác không thể nhận được khóa ở bất kỳ cấp nào khác có thể ảnh hưởng đến tính toàn vẹn của hàng. Người dùng thứ hai có thể có được khóa cấp hàng trên các hàng khác hoặc khóa cấp trang trên các trang khác, nhưng khóa cấp độ trang hoặc bảng không tương thích có chứa hàng sẽ không được được cấp cho người dùng khác.

Mức độ mà các khóa nên được áp dụng không cần phải được chỉ định bởi người dùng hoặc database administrato; lock manager xác định điều đó tự động. Nó thường prefer row-level và key-level khi truy cập vào một số lượng nhỏ hàng để hỗ trợ đồng thời. Tuy nhiên, nếu chi phí khóa của nhiều khóa cấp thấp hóa ra rất cao, trình quản lý khóa sẽ tự động chọn khóa cấp cao hơn phù hợp.

**Lock Operations and Modes**

Do tính đa dạng của các hoạt động mà SQL Server cần thực hiện, một bộ cơ chế khóa lớn và phức tạp không kém được duy trì. Ngoài các loại khóa khác nhau, có một đường dẫn leo thang để thay đổi từ loại khóa này sang loại khóa khác. Các phần sau đây mô tả các chế độ và quy trình này, cũng như việc sử dụng chúng.

**Lock Escalation**

Khi một truy vấn được thực thi, SQL Server xác định mức khóa yêu cầu cho các đối tượng cơ sở dữ liệu được đề cập trong truy vấn và nó bắt đầu thực hiện truy vấn sau khi có được các khóa được yêu cầu. Trong quá trình thực hiện truy vấn, trình quản lý khóa theo dõi số lượng khóa được yêu cầu bới truy vấn để xác định nhu cầu nâng mức khóa từ cấp hiện tại lên cấp cao hơn.

Ngưỡng leo thang khóa được xác định bởi SQL Server trong quá trình transaction. Row lock và page lock được tự động chuyển sang table lock khi transaction vượt quá ngưỡng của nó. Sau khi mức khóa được chuyển lên table lock, tất cả các khóa cấp thấp hơn trên bảng sẽ tự động được giải phóng. Tính năng leo thang khóa động này của trình quản lý khóa sẽ tối ưu hóa chi phí khóa của truy vấn.

Có thể thiết lập một mức độ kiểm soát các cơ chế khóa trên một bảng nhất định. Ví dụ, bạn có thể kiểm soát việc leo thang khóa xảy ra. Sau đây là cú pháp T-SQL để thực hiện thay đổi đó:



Cú pháp này sẽ vô hiệu hóa leo thang khóa trên toàn bộ bảng(ngoại trừ một vài trường hợp đặc biệt). Bạn cũng có thể đặt nó thành TABLE, điều này sẽ khiến việc leo thang đi đến table lock mỗi lần. Bạn cũng có thể đặt leo thang khóa trên bảng thành AUTO, điều này sẽ cho phép SQL Server đưa ra quyết định cho lược đồ khóa và bất kỳ sự leo thang cần thiết. Nếu bảng đó được phân vùng, bạn có thể thấy sự thay đổi leo thang đến cấp phân vùng. Một lần nữa, hãy thận trọng khi sử dụng các loại sửa đổi này đối với hành vi của SQL Server tiêu chuẩn.

Bạn cũng có tùy chọn vô hiệu hóa leo thang khóa trên cơ sở rộng hơn bằng cách sử dụng cờ theo dõi 1224. Điều này vô hiệu hóa leo thang khóa dựa trên số lượng khóa nhưng vẫn giữ nguyên mức leo thang khóa dựa trên áp lực bộ nhớ. Bạn cũng có thể vô hiệu hóa pressure lock escalation cũng như số lượng khóa bằng cách sử dụng cờ theo dõi 1211, nhưng đó là một lựa chọn nguy hiểm và có thể dẫn đến lỗi trên hệ thống của bạn. Tôi thực sự khuyên bạn nên kiểm tra kỹ lưỡng trước khi sử dụng một trong các tùy chọn này.

**Lock Modes**

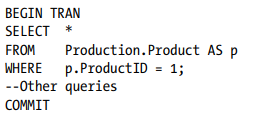
Mức độ cô lập được yêu cầu bởi các transaction khác nhau có thể khác nhau. Chẳng hạn, tính nhất quán của dữ liệu không bị ảnh hưởng nếu hai transaction đọc dữ liệu cùng một lúc; tuy nhiên, tính nhất quán bị ảnh hưởng nếu hai transaction được phép sửa đổi dữ liệu cùng một lúc. Tùy thuộc vào loại quyền truy cập được yêu cầu, SQL Server sử dụng các chế độ khóa khác nhau trong khi khóa tài nguyên:

* Shared (S)
* Update (U)
* Exclusive (X)
* Intent:
  + Intent Shared (IS)
  + Intent Exclusive (IX)
  + Shared with Intent Exclusive (SIX)
  + Schema:
  + Schema Modification (Sch-M)
  + Schema Stability (Sch-S)
  + Bulk Update (BU)
  + Key-Range

**Shared (S) Mode**

Chế độ chia sẻ được sử dụng cho các truy vấn read-only, chẳng hạn như câu lệnh SELECT. Nó không ngăn các truy vấn chỉ đọc khác truy cập dữ liệu đồng thời vì tính toàn vẹn của dữ liệu không bị xâm phạm bởi các lần đọc đồng thời.

Tuy nhiên, các truy vấn sửa đổi dữ liệu đồng thời trên dữ liệu được ngăn chặn để duy trì tính toàn vẹn dữ liệu. Khóa (S) được giữ trên dữ liệu cho đến khi dữ liệu được đọc. Theo mặc định, khóa (S) có được bởi câu lệnh SELECT được giải phóng ngay sau khi đọc dữ liệu. Ví dụ: xem xét các transaction sau:



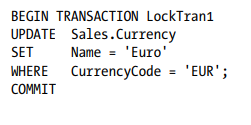
Khóa (S) có được bởi câu lệnh SELECT không được giữ cho đến khi kết thúc transaction; thay vào đó, nó được giải phóng ngay lập tức sau khi dữ liệu được đọc bởi câu lệnh SELECT trong read\_ commit, mức cô lập mặc định. Hành vi này của khóa (S) có thể được thay đổi bằng cách sử dụng mức cách ly cao hơn hoặc gợi ý khóa.

**Update (U) Mode**

Update mode có thể được coi là tương tự như khóa (S), nhưng nó cũng bao gồm một mục tiêu để sửa đổi dữ liệu như là một phần của cùng một truy vấn. Không giống như khóa (S), khóa (U) chỉ ra rằng dữ liệu được đọc để sửa đổi. Vì dữ liệu được đọc với mục tiêu để sửa đổi nó, SQL Server không cho phép nhiều hơn một (U) khóa trên dữ liệu cùng một lúc. Quy tắc này giúp duy trì tính toàn vẹn dữ liệu. Lưu ý rằng khóa (S) đồng thời trên dữ liệu được cho phép. Khóa (U) được liên kết với câu lệnh UPDATE và hành động của câu lệnh UPDATE thực sự bao gồm hai bước trung gian.

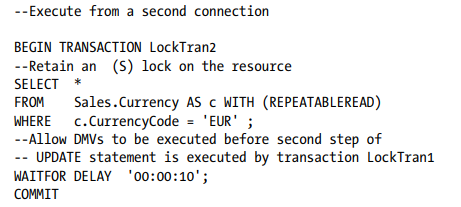
1. Read the data to be modified.
2. Modify the data.

Các chế độ khóa khác nhau được sử dụng trong hai bước trung gian để tối đa hóa đồng thời. Thay vì có được quyền độc quyền trong khi đọc dữ liệu, bước đầu tiên có được khóa (U) trên dữ liệu. Trong bước thứ hai, khóa (U) được chuyển đổi thành khóa độc quyền để sửa đổi. Nếu không cần sửa đổi, thì khóa (U) được giải phóng; nói cách khác, nó không được giữ cho đến khi kết thúc transaction. Xem xét ví dụ sau, thể hiện hành vi khóa của câu lệnh UPDATE:



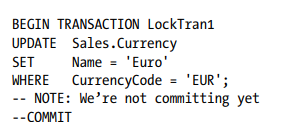
Để hiểu hành vi khóa của các bước trung gian của câu lệnh UPDATE, bạn cần lấy dữ liệu từ sys.dm\_tran\_locks ở cuối mỗi bước. Bạn có thể có được trạng thái khóa sau mỗi bước của câu lệnh UPDATE bằng cách làm theo các bước được nêu tiếp theo. Bạn có thể mở ba kết nối mà tôi sẽ gọi là Kết nối 1, Kết nối 2 và Kết nối 3. Điều này sẽ yêu cầu ba cửa sổ truy vấn khác nhau trong Management Studio. Bạn sẽ chạy các truy vấn trong các kết nối tôi liệt kê theo thứ tự mà tôi chỉ định để đến tình huống blocking; điểm của điều này là để quan sát những khối khi chúng xảy ra. Truy vấn ban đầu, được liệt kê trước đây, nằm trong Kết nối 1:

1. Chặn bước thứ hai của câu lệnh UPDATE bằng cách thực hiện transaction đầu tiên từ kết nối thứ hai, Kết nối 2.

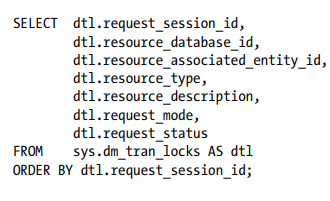


Gợi ý khóa REPEATABLEREAD, chạy trong Kết nối 2, cho phép câu lệnh SELECT giữ lại khóa (S) trên tài nguyên.

1. Trong khi transaction LockTran2 đang thực thi, hãy thực hiện transaction UPDATE, updatelock, từ kết nối đầu tiên (được lặp lại ở đây cho rõ ràng), Kết nối 1.



1. Trong khi câu lệnh UPDATE bị chặn, hãy truy vấn sys.dm\_tran\_locks DMV từ kết nối thứ ba, Kết nối 3, như sau:

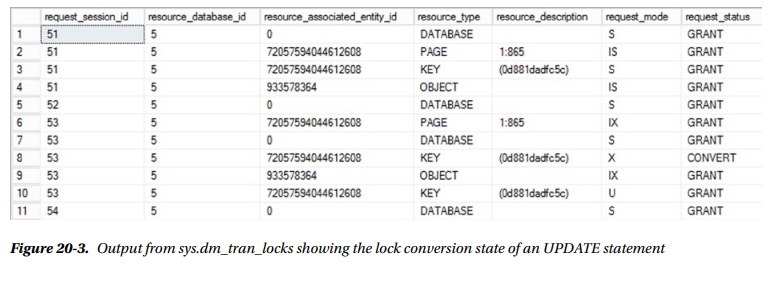


Đầu ra từ sys.dm\_tran\_locks trong Kết nối 3 sẽ cung cấp trạng thái khóa sau bước đầu tiên của câu lệnh UPDATE kể từ khi chuyển đổi khóa thành khóa độc quyền (X) bởi câu lệnh UPDATE bị chặn bởi câu lệnh SELECT.

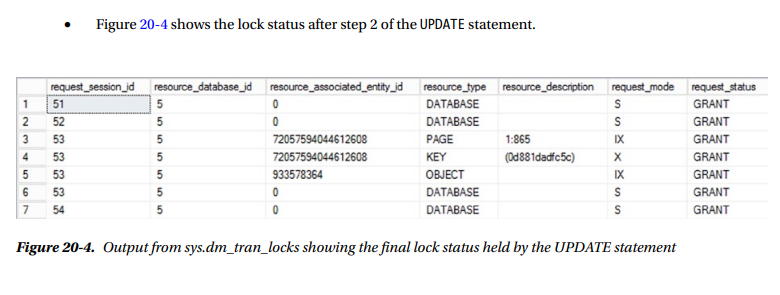
1. Trạng thái khóa sau bước thứ hai của câu lệnh UPDATE sẽ được cung cấp bằng cách chạy lại truy vấn đối với sys.dm\_tran\_locks trong Kết nối 3.

Tiếp theo, hãy để xem xét trạng thái khóa được cung cấp bởi sys.dm\_tran\_locks khi bạn thực hiện các bước riêng lẻ của câu lệnh UPDATE.

Hình 20-3 hiển thị trạng thái khóa sau bước 1 của câu lệnh UPDATE (thu được từ đầu ra từ sys.dm\_tran\_locks được thực hiện trên kết nối thứ ba, Kết nối 3, như đã giải thích trước đó).



Thứ tự của các hàng này không quan trọng. Tôi đã đặt hàng bởi session\_id để nhóm các khóa từ mỗi truy vấn.



Từ đầu ra sys.dm\_tran\_locks sau bước đầu tiên của câu lệnh UPDATE, bạn có thể lưu ý những điều sau:

* Khóa (U) được cấp cho SPID trên hàng dữ liệu.
* Yêu cầu chuyển đổi sang khóa (X) trên hàng dữ liệu.

Từ đầu ra của sys.dm\_tran\_locks sau bước thứ hai của câu lệnh UPDATE, bạn có thể thấy rằng câu lệnh UPDATE chỉ giữ một khóa (X) trên hàng dữ liệu. Về cơ bản, khóa (U) trên hàng dữ liệu được chuyển đổi thành khóa (X).

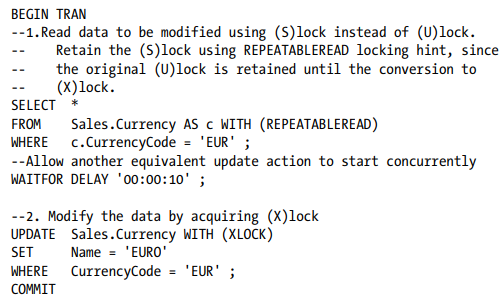
Bằng cách không có được khóa độc quyền ở bước đầu tiên, câu lệnh CẬP NHẬT cho phép các transaction khác đọc dữ liệu bằng cách sử dụng câu lệnh CHỌN trong khoảng thời gian đó. Điều này là có thể bởi vì (U) và (S) khóa tương thích với nhau. Điều này làm tăng sự tương tranh cơ sở dữ liệu.

Bạn có thể tò mò tìm hiểu tại sao khóa (U) được sử dụng thay vì khóa (S) trong bước đầu tiên của câu lệnh CẬP NHẬT.

Để hiểu nhược điểm của việc sử dụng khóa (S) thay vì khóa (U) trong bước đầu tiên của câu lệnh CẬP NHẬT, hãy chia câu lệnh CẬP NHẬT thành hai bước.

* Đọc dữ liệu cần sửa đổi bằng khóa (S) thay vì khóa (U).
* Sửa đổi dữ liệu bằng cách lấy khóa (X).

Consider the following code:



Nếu transaction này được thực hiện đồng thời từ hai kết nối, thì nó sẽ gây ra bế tắc, như sau:



Cả hai transaction đều đọc dữ liệu cần sửa đổi bằng khóa (S) và sau đó yêu cầu khóa (X) để sửa đổi.

Khi transaction đầu tiên thử chuyển đổi sang khóa (X), nó bị chặn bởi khóa (S) được giữ bởi transaction thứ hai. Tương tự, khi transaction thứ hai thử chuyển đổi từ khóa (S) sang khóa (X), nó bị chặn bởi khóa (S) được giữ bởi transaction đầu tiên, lần lượt bị chặn bởi transaction thứ hai. Điều này gây ra một khối tròn tròn và do đó, bế tắc.

Để tránh bế tắc điển hình này, câu lệnh UPDATE sử dụng khóa (U) thay vì khóa (S) ở bước trung gian đầu tiên. Không giống như khóa (S), khóa (U) không cho phép khóa (U) khác trên cùng một tài nguyên. Điều này buộc câu lệnh UPDATE đồng thời thứ hai phải đợi cho đến khi câu lệnh UPDATE đầu tiên hoàn thành.

**Exclusive (X) Mode**

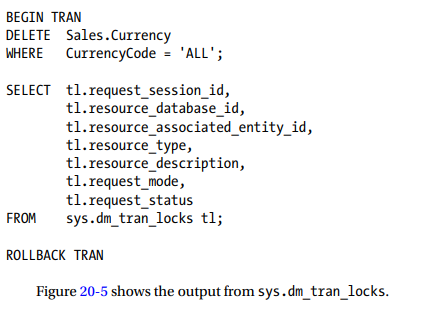
Chế độ độc quyền cung cấp quyền độc quyền trên tài nguyên cơ sở dữ liệu để sửa đổi bằng các truy vấn thao tác dữ liệu như INSERT, UPDATE, và DELETE. Nó ngăn chặn các transaction đồng thời khác truy cập vào tài nguyên theo sửa đổi. Cả hai câu lệnh INSERT và DELETE thu được khóa (X) ngay khi bắt đầu thực thi. Như đã giải thích trước đó, câu lệnh UPDATE chuyển đổi sang khóa (X) sau khi dữ liệu cần sửa đổi được đọc. Các khóa (X) được cấp trong một transaction được giữ cho đến khi kết thúc transaction.

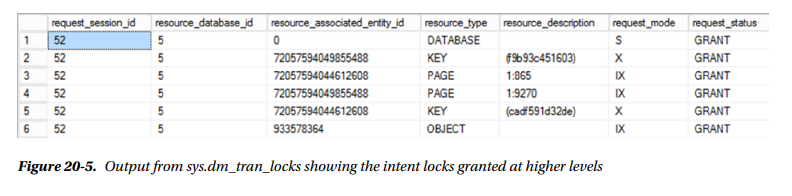
Khóa (X) phục vụ hai mục đích.

* Nó ngăn các transaction khác truy cập vào tài nguyên đang được sửa đổi để họ thấy một giá trị trước hoặc sau khi sửa đổi, không phải là một giá trị đang được sửa đổi.
* Nó cho phép transaction sửa đổi tài nguyên để quay trở lại giá trị ban đầu một cách an toàn trước khi sửa đổi, nếu cần, vì không có transaction nào khác được phép sửa đổi tài nguyên đồng thời.

**Intent Shared (IS), Intent Exclusive (IX and Shared with Intent Exclusive (SIX) Modes**

Intent Shared, Intent Exclusive và Shared with Intent Exclusive chỉ ra rằng truy vấn có ý định lấy khóa (S) hoặc (X) tương ứng ở mức khóa thấp hơn. Ví dụ: hãy xem xét transaction sau trên bảng Sales.Currency:





Khóa (IX) ở cấp độ bảng (PAGE) biểu thị rằng câu lệnh DELETE có ý định lấy khóa (X) ở cấp độ page, row, or key. Tương tự, khóa (IX) ở cấp độ trang (PAGE) cho biết rằng truy vấn có ý định lấy khóa (X) trên một hàng trong trang. Khóa (IX) ở các cấp cao hơn ngăn không cho transaction khác có được khóa không tương thích trên bảng hoặc trên trang có chứa hàng.

Việc gắn cờ khóa ý định khác (IS) hoặc (IX) hội tụ một mức cao hơn tương ứng bằng một transaction, trong khi giữ khóa ở mức thấp hơn, ngăn các transaction khác có được incompatible lock ở cấp cao hơn. Nếu intent lock không được sử dụng, thì transaction cố gắng lấy khóa ở cấp cao hơn sẽ phải quét qua các cấp thấp hơn để phát hiện sự hiện diện của khóa cấp thấp hơn. Mặc dù intent lock ở các cấp cao hơn cho thấy sự hiện diện của khóa cấp thấp hơn, nhưng chi phí khóa của việc mua khóa ở cấp cao hơn được tối ưu hóa. Các intent lock được cấp cho một transaction được giữ cho đến khi kết thúc transaction.

Chỉ một khóa (SIX) có thể được đặt trên một tài nguyên nhất định cùng một lúc. Điều này ngăn chặn cập nhật được thực hiện bởi các transactionkhác. Các transactionkhác có thể đặt khóa (IS) trên các tài nguyên cấp thấp hơn trong khi khóa (SIX) được đặt đúng chỗ.

Hơn nữa, có thể có sự kết hợp của các khóa được yêu cầu (hoặc có được) ở một mức độ nhất định và ý định có một khóa (hoặc khóa) ở mức thấp hơn. Ví dụ: có thể có các kết hợp khóa (SIU) và (UIX) chỉ ra rằng khóa (S) hoặc (U) đã được mua ở cấp độ tương ứng và (các) khóa (U) hoặc (X) được dự định một cấp độ thấp hơn.

**Schema Modification (Sch-M) and Schema Stability (Sch-S) Modes**

Schema Modification và Schema Stability được lấy trên một bảng bằng các câu lệnh SQL phụ thuộc vào lược đồ của bảng. Một tuyên bố DDL, làm việc trên lược đồ của bảng, có được khóa (Sch-M) trên bảng và ngăn các transactionkhác truy cập vào bảng. (Sch-S) lock có được cho các hoạt động cơ sở dữ liệu phụ thuộc vào lược đồ nhưng không sửa đổi lược đồ, chẳng hạn như query compilation. Nó ngăn chặn khóa (Sch-M) trên bảng, nhưng nó cho phép các khóa khác được cấp trên bảng.

Vì trên production database, các sửa đổi lược đồ không thường xuyên, các khóa (Sch-M),thường không trở thành một vấn đề chặn. Và bởi vì (Sch-S) lock không chặn các khóa khác ngoại trừ khóa (Sch-M), đồng thời thường không bị ảnh hưởng bởi khóa (Sch-S)

**Bulk Update (BU) Mode**

Bulk Update lock mode là duy nhất cho các hoạt động tải số lượng lớn. Các hoạt động này là bcp kiểu cũ (bản sao số lượng lớn), câu lệnh BULK INSERT và chèn từ OPENROWSET bằng tùy chọn BULK. Là một cơ chế để tăng tốc các quy trình này, bạn có thể cung cấp gợi ý TABLOCK hoặc đặt tùy chọn trên bảng để nó khóa khi tải hàng loạt. Chìa khóa cho chế độ khóa (BU) là nó sẽ cho phép nhiều hoạt động hàng loạt chống lại bảng bị khóa nhưng ngăn các hoạt động khác trong khi quy trình hàng loạt đang chạy.

**Key-range Mode**

Chế độ Key-Range chỉ có thể áp dụng trong khi mức cô lập được đặt thành Serializable (bạn có thể tìm hiểu thêm về các mức cô lập transaction trong phần “Isolation Levels”). Các khóa Key-Range được áp dụng cho một loạt hoặc phạm vi các giá trị khóa sẽ được sử dụng nhiều lần trong khi transaction được mở. Khóa một phạm vi trong một transaction tuần tự hóa đảm bảo rằng các hàng khác không được chèn trong phạm vi, có thể thay đổi các tập kết quả trong giao dịch. Phạm vi có thể được khóa bằng các chế độ khóa khác, làm cho nó giống như chế độ khóa kết hợp hơn là chế độ khóa riêng biệt. Để chế độ Key-Range hoạt động, chỉ mục phải được sử dụng để xác định các giá trị trong phạm vi

**Lock Compatibility**

SQL Server cung cấp sự cô lập cho một transaction bằng cách ngăn các transaction khác truy cập vào cùng một tài nguyên theo cách không tương thích. Tuy nhiên, nếu một transaction cố gắng thực hiện một nhiệm vụ tương thích trên cùng một tài nguyên, thì, để tăng tính đồng thời, nó sẽ không bị chặn bởi transaction đầu tiên. SQL Server đảm bảo loại chặn chọn lọc này bằng cách ngăn chặn transactioncó được khóa không tương thích trên tài nguyên do transaction khác nắm giữ. Ví dụ: khóa (S) có được trên một tài nguyên bởi một transaction cho phép các transaction khác có được khóa (S) trên cùng một tài nguyên. Tuy nhiên, khóa (Sch-M) trên tài nguyên bằng transaction sẽ ngăn các transaction khác có được bất kỳ khóa nào trên tài nguyên đó.

**Isolation Levels**

Các chế độ khóa được giải thích trong phần trước giúp giao dịch bảo vệ tính nhất quán dữ liệu của nó khỏi các giao dịch đồng thời khác. Mức độ bảo vệ dữ liệu hoặc cách ly một giao dịch được phụ thuộc, không chỉ vào các lock mode mà còn phụ thuộc vào mức độ cô lập của giao dịch. Mức này ảnh hưởng đến hành vi của các lock mode.

Ví dụ, theo mặc định, khóa (S) được giải phóng ngay sau khi dữ liệu được đọc; nó được tổ chức cho đến khi kết thúc giao dịch. Hành vi này có thể không phù hợp với một số chức năng ứng dụng. Trong những trường hợp như vậy, bạn có thể định cấu hình mức cô lập của giao dịch để đạt được mức độ cô lập mong muốn. SQL Server thực hiện sáu cấp độ cách ly, bốn trong số chúng theo định nghĩa của ISO:

* Read Uncommitted
* Read Committed
* Repeatable Read
* Serializable

Hai mức cô lập khác cung cấp row versioning, đó là một cơ chế theo đó một phiên bản của hàng được tạo như một phần của các truy vấn thao tác dữ liệu. Phiên bản bổ sung này của hàng cho phép đọc các truy vấn để truy cập dữ liệu mà không cần lấy khóa chống lại nó. Hai mức cô lập thêm như sau:

* Read Committed Snapshot (actually part of the Read Committed isolation)
* Snapshot

Bốn mức cách ly ISO được liệt kê theo thứ tự tăng dần mức độ cô lập. Bạn có thể định cấu hình chúng ở cấp độ kết nối hoặc truy vấn bằng cách sử dụng câu lệnh SET TRANSACTION ISOLATION LEVEL hoặc gợi ý khóa tương ứng. Cấu hình mức cô lập ở mức kết nối vẫn có hiệu lực cho đến khi mức cô lập được cấu hình lại bằng cách sử dụng câu lệnh SET hoặc cho đến khi kết nối được đóng lại. Tất cả các mức cô lập được giải thích trong các phần sau

**Read Uncommitted**

Read Uncommitted là mức thấp nhất trong bốn mức cô lập và nó cho phép các câu lệnh SELECT đọc dữ liệu mà không yêu cầu khóa (S). Do khóa (S) không được yêu cầu bởi câu lệnh SELECT, nên nó không block và cũng không bị block bởi khóa (X). Nó cho phép một câu lệnh SELECT để đọc dữ liệu trong khi dữ liệu đang được sửa đổi. Loại dữ liệu đọc này được gọi là đọc bẩn.

Giả sử bạn có một ứng dụng trong đó số lượng sửa đổi dữ liệu là cực kỳ nhỏ và ứng dụng của bạn không đòi hỏi nhiều về cách lấy từ câu lệnh SELECT để đọc dữ liệu. Trong trường hợp này, bạn có thể sử dụng mức cô lập Read Uncommitted để tránh việc một số hoạt động sửa đổi dữ liệu khác block câu lệnh SELECT.

Bạn có thể sử dụng câu lệnh SET sau để định cấu hình mức cô lập của kết nối cơ sở dữ liệu thành Read Uncommitted:



Bạn cũng có thể đạt được mức độ cô lập này trên cơ sở truy vấn bằng cách sử dụng gợi ý khóa NOLOCK.



Hiệu ứng của gợi ý khóa vẫn được áp dụng cho truy vấn và không thay đổi mức độ cô lập của kết nối.

Mức cô lập Read Uncommitted sẽ tránh việc block gây ra bởi câu lệnh SELECT, nhưng bạn không nên sử dụng nó nếu giao dịch phụ thuộc vào độ chính xác của dữ liệu được đọc bởi câu lệnh SELECT hoặc nếu giao dịch không thể chịu được sự thay đổi dữ liệu đồng thời bởi một giao dịch khác.

Nó rất quan trọng để hiểu ý nghĩa của việc đọc bẩn. Nhiều người nghĩ rằng điều này có nghĩa là, trong khi một trường đang được cập nhật từ Tusa sang Tulsa, một truy vấn vẫn có thể đọc giá trị trước đó hoặc thậm chí giá trị được cập nhật, trước khi commit. Mặc dù đó là sự thật, nhưng nhiều vấn đề dữ liệu nghiêm trọng hơn có thể xảy ra. Vì không có khóa được đặt trong khi đọc dữ liệu, các chỉ mục có thể được phân chia. Điều này có thể dẫn đến các hàng dữ liệu thừa hoặc thiếu được trả về truy vấn. Để rõ ràng, việc sử dụng Read Uncommitted trong bất kỳ môi trường nào xảy ra thao tác dữ liệu cũng như đọc dữ liệu có thể dẫn đến các hành vi không lường trước được. Mục đích của mức cô lập này là dành cho các hệ thống chủ yếu tập trung vào báo cáo và kinh doanh thông minh, không phải xử lý giao dịch trực tuyến

**Read Committed**

Mức cô lập Read Committed ngăn chặn việc đọc bẩn gây ra bởi mức cô lập Read Uncommitted. Điều này có nghĩa là các khóa (S) được yêu cầu bởi các câu lệnh SELECT ở mức cô lập này. Đây là mức cô lập mặc định của SQL Server. Nếu cần, bạn có thể thay đổi mức cô lập của kết nối thành Read Committed bằng cách sử dụng câu lệnh SET sau:



Mức cô lập Read Committed là tốt cho hầu hết các trường hợp, nhưng vì khóa (S) có được bởi câu lệnh SELECT không được giữ cho đến khi kết thúc giao dịch, nó có thể gây ra các sự cố đọc hoặc đọc ảo không thể lặp lại, như được giải thích trong các phần theo.

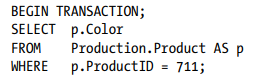
Hành vi của mức cô lập Read Committed có thể được thay đổi bằng tùy chọn cơ sở dữ liệu READ\_COMMITTED\_SNAPSHOT. Khi điều này được đặt thành BẬT, row versioning được sử dụng bởi các giao dịch thao tác dữ liệu. Điều này đặt thêm một tải cho tempdb vì các phiên bản trước của các hàng được thay đổi được lưu trữ ở đó trong khi giao dịch không được cam kết.

Điều này cho phép các giao dịch khác truy cập dữ liệu để đọc mà không cần phải khóa dữ liệu, điều này có thể cải thiện tốc độ và hiệu quả của tất cả các truy vấn trong hệ thống mà không dẫn đến các sự cố do chia tách trang với NOLOCK hoặc READ UNCOMMITTED

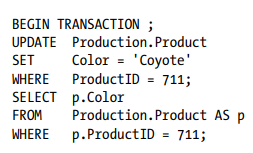
Tiếp theo, sửa đổi cơ sở dữ liệu AdventureWorks2012 để READ\_COMMITTED\_SNAPSHOT được bật.



Bây giờ hãy tưởng tượng một tình huống kinh doanh. Kết nối và giao dịch đầu tiên sẽ lấy dữ liệu từ bảng Production.Product, có được màu của một mặt hàng cụ thể.



Kết nối thứ hai được thực hiện với một giao dịch mới sẽ sửa đổi màu của cùng một mặt hàng



Chạy câu lệnh SELECT sau khi cập nhật màu, bạn có thể thấy màu đó đã được cập nhật. Nhưng nếu bạn quay lại kết nối đầu tiên và chạy lại câu lệnh SELECT ban đầu (không nên chạy lại câu lệnh BEGIN TRAN), bạn vẫn sẽ thấy màu đó là Blue. Chuyển về kết nối thứ hai và kết thúc giao dịch.



Chuyển lại giao dịch đầu tiên, commit giao dịch đó và sau đó chạy lại câu lệnh SELECT ban đầu. Bạn sẽ thấy màu mới được cập nhật cho vật phẩm, Coyote. Bạn có thể đặt lại mức cô lập trên AdventureWorks2012 trước khi tiếp tục.



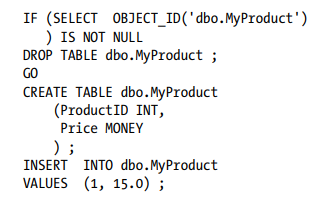
Nếu tempdb được điền, sửa đổi dữ liệu bằng cách sử dụng row versioning sẽ tiếp tục thành công, nhưng việc đọc có thể thất bại do hàng được phiên bản sẽ không có sẵn. Nếu bạn kích hoạt bất kỳ loại cách ly phiên bản hàng nào trong cơ sở dữ liệu của mình, bạn phải cẩn thận hơn để duy trì không gian trống trong tempdb.

**Repeatable Read**

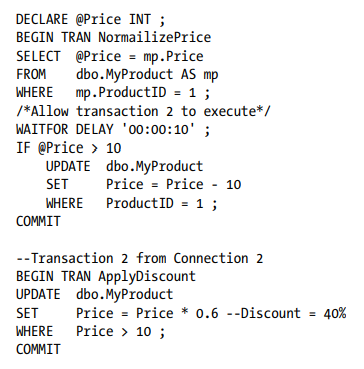
Mức cô lập Repeatable Read cho phép câu lệnh SELECT giữ khóa (S) của nó cho đến khi kết thúc giao dịch, do đó ngăn các giao dịch khác sửa đổi dữ liệu trong thời gian đó. Chức năng cơ sở dữ liệu có thể thực hiện quyết định hợp lý bên trong giao dịch dựa trên dữ liệu được đọc bởi câu lệnh SELECT trong giao dịch. Nếu kết quả của quyết định phụ thuộc vào dữ liệu được đọc bởi câu lệnh SELECT, thì bạn nên xem xét việc ngăn chặn sửa đổi dữ liệu bằng các giao dịch đồng thời khác. Ví dụ: xem xét hai giao dịch sau:

* Normalize the price for ProductID = 1: For ProductID = 1, if Price > 10, then decrease the price by 10.
* Apply a discount: For products with Price > 10, apply a discount of 40 percent.

Bây giờ hãy xem xét bảng thử nghiệm sau:



You can write the two transactions like this:



Nhìn bề ngoài, các giao dịch trước có thể trông tốt và vâng, chúng hoạt động trong môi trường một người dùng. Nhưng trong một môi trường nhiều người dùng, nơi nhiều giao dịch có thể được thực hiện đồng thời, bạn có một vấn đề ở đây!

Để tìm ra vấn đề, hãy thực hiện hai giao dịch từ các kết nối khác nhau theo thứ tự sau:

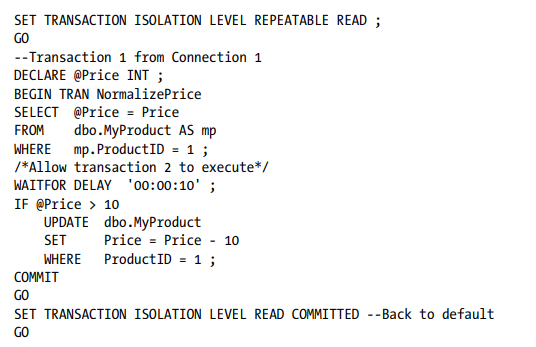
* Start transaction 1 first.
* Start transaction 2 within ten seconds of the start of transaction 1.

Như bạn có thể đoán, khi kết thúc giao dịch, giá mới của sản phẩm (có ID sản phẩm = 1) sẽ là -1.0. Có vẻ như bạn đã sẵn sàng ra khỏi doanh nghiệp!

Sự cố xảy ra do giao dịch 2 được phép sửa đổi dữ liệu trong khi giao dịch 1 đã đọc xong dữ liệu và sắp đưa ra quyết định về nó. Giao dịch 1 yêu cầu mức độ cô lập cao hơn so với mức cô lập mặc định (Read Committed).

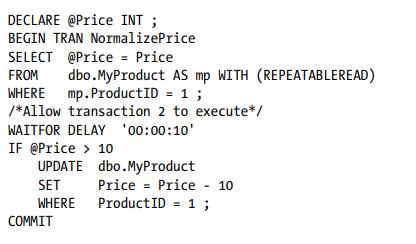
Là một giải pháp, bạn muốn ngăn giao dịch 2 sửa đổi dữ liệu trong khi giao dịch 1 đang hoạt động trên đó.

Nói cách khác, cung cấp cho giao dịch 1 khả năng đọc lại dữ liệu sau này trong giao dịch mà không bị người khác sửa đổi. Tính năng này được gọi là đọc lặp lại. Xem xét bối cảnh, việc thực hiện các giải pháp có lẽ là rõ ràng. Sau khi tạo lại bảng mẫu, bạn có thể viết điều này:



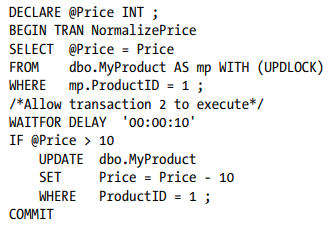
Việc tăng mức độ cô lập của giao dịch 1 thành Repeatable Read sẽ ngăn giao dịch 2 sửa đổi dữ liệu trong quá trình thực hiện giao dịch 1. Do đó, bạn sẽ không có một giá trị không nhất quán về giá của sản phẩm

Vì ý định không giải phóng khóa (S) có được bởi câu lệnh SELECT cho đến khi kết thúc giao dịch, nên hiệu quả của việc đặt mức cô lập thành Repeatable Read cũng có thể được thực hiện ở cấp truy vấn bằng gợi ý khóa.



Giải pháp này ngăn chặn sự không thống nhất dữ liệu của MyProduct.Price, nhưng nó đưa ra một vấn đề khác cho kịch bản này. Khi quan sát kết quả của giao dịch 2, bạn nhận ra rằng nó có thể gây ra deadlock. Do đó, mặc dù giải pháp trước đó đã ngăn chặn sự không thống nhất dữ liệu, nhưng nó không phải là một giải pháp hoàn chỉnh. Nhìn kỹ vào tác động của mức cô lập Repeatable Read đối với các giao dịch, bạn sẽ thấy rằng nó đã đưa ra vấn đề deadlock điển hình được tránh bằng cách triển khai nội bộ của một câu lệnh UPDATE, như đã giải thích trước đây. Câu lệnh SELECT thu được và giữ lại khóa (S) thay vì khóa (U), mặc dù nó có ý định sửa đổi dữ liệu sau này trong giao dịch. Khóa (S) cho phép giao dịch 2 có được khóa (U), nhưng nó đã chặn chuyển đổi (U) khóa thành khóa (X). Việc cố gắng giao dịch 1 để có được khóa (U) trên dữ liệu ở giai đoạn sau đã gây ra một khối tròn, dẫn đến deadlock.

Để ngăn chặn deadlook và vẫn tránh hỏng dữ liệu, bạn có thể sử dụng một chiến lược tương đương như được thông qua khi triển khai nội bộ của câu lệnh UPDATE. Do đó, thay vì yêu cầu khóa (S), giao dịch 1 có thể yêu cầu khóa (U) bằng cách sử dụng gợi ý khóa UPDLOCK khi thực hiện câu lệnh SELECT



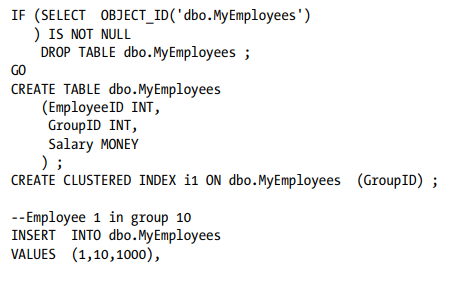
Giải pháp này ngăn ngừa cả sự không nhất quán dữ liệu và khả năng deadlock. Nếu việc tăng mức cô lập thành Repeatable Read không đưa ra deadlock điển hình, thì nó sẽ hoàn thành công việc. Vì có khả năng xảy ra deadlock do việc giữ khóa (S) cho đến khi kết thúc giao dịch, nên thường lấy khóa (U) thay vì giữ khóa (S), như minh họa.

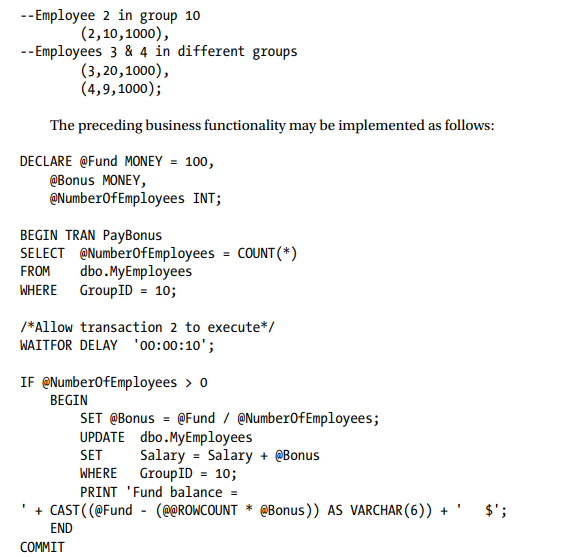
**Serializable**

Serializable là mức cao nhất trong sáu mức cô lập. Thay vì chỉ có được một khóa trên hàng được truy cập, mức cô lập Serializable có được một khóa phạm vi trên hàng và hàng tiếp theo theo thứ tự của tập dữ liệu được yêu cầu.

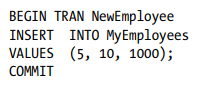
Chẳng hạn, một câu lệnh SELECT được thực thi ở mức cô lập Serializable có được khóa (RangeS-S) trên hàng được truy cập và hàng tiếp theo theo thứ tự. Điều này ngăn việc thêm các hàng bởi các giao dịch khác trong tập dữ liệu được vận hành bởi giao dịch đầu tiên và nó bảo vệ giao dịch đầu tiên tìm thấy các hàng mới trong tập dữ liệu của nó trong phạm vi giao dịch. Tìm các hàng mới trong một tập dữ liệu trong một giao dịch cũng được gọi là *phantom read*.

Để hiểu được sự cần thiết của một mức cô lập Serializable, hãy để xem xét một ví dụ. Giả sử một nhóm (với GroupID = 10) trong một công ty có quỹ 100 đô la được phân phối giữa các nhân viên trong nhóm dưới dạng tiền thưởng. Số dư quỹ sau khi thanh toán tiền thưởng nên là 0 đô la. Hãy xem xét bảng thử nghiệm sau:





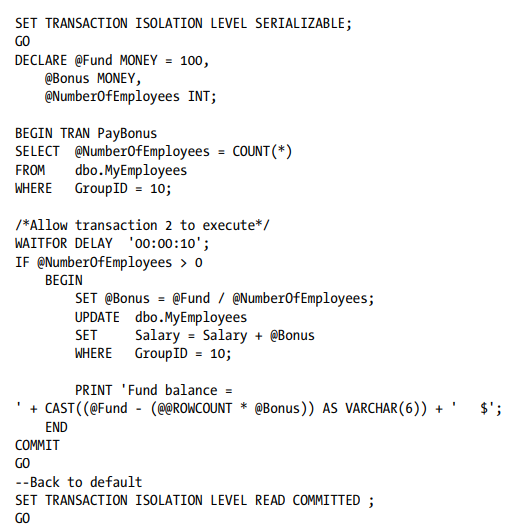
Bạn sẽ thấy giá trị trả về là số dư quỹ 0 đô la kể từ khi cập nhật hoàn tất thành công. Giao dịch PayBonus hoạt động tốt trong môi trường có 1 người dùng. Tuy nhiên, trong một môi trường nhiều người dùng, có một vấn đề. Xem xét một giao dịch khác có thêm một nhân viên mới vào GroupID = 10 như sau và được thực hiện đồng thời (ngay sau khi bắt đầu giao dịch PayBonus) từ kết nối thứ hai:



Số dư quỹ sau giao dịch PayBonus sẽ là - $ 50! Mặc dù nhân viên mới có thể thích nó, quỹ nhóm sẽ chìm trong màu đỏ. Điều này gây ra sự không nhất quán trong trạng thái logic của dữ liệu.

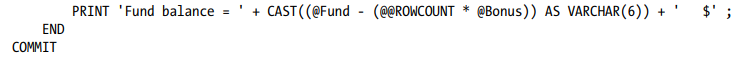
Để ngăn chặn sự không nhất quán dữ liệu này, việc thêm nhân viên mới vào nhóm (hoặc bộ dữ liệu) đang hoạt động nên bị block. Trong năm mức cô lập được thảo luận, chỉ có cách ly Snapshot có thể cung cấp chức năng tương tự, vì giao dịch phải được bảo vệ không chỉ trên dữ liệu hiện tại mà còn khỏi sự xâm nhập của dữ liệu mới trong bộ dữ liệu. Mức cô lập Serializable có thể cung cấp loại cách ly này bằng cách có được một khóa phạm vi trên hàng bị ảnh hưởng và thứ tự hàng tiếp theo theo được xác định bởi chỉ số MyEmprocod.il trên cột GroupID. Do đó, sự không nhất quán dữ liệu của giao dịch PayBonus có thể được ngăn chặn bằng cách đặt mức cô lập giao dịch thành Serializable.

Nhớ tạo lại bảng trước.



Hiệu quả của mức cô lập Serializable cũng có thể đạt được ở cấp truy vấn bằng cách sử dụng gợi ý khóa HOLDLOCK trên câu lệnh CHỌN, như được hiển thị ở đây:



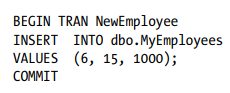


Bạn có thể quan sát các khóa phạm vi có được bởi giao dịch PayBonus bằng cách truy vấn sys.dm\_tran\_locks từ một kết nối khác trong khi giao dịch PayBonus đang thực thi, như trong Hình 20-6.



Đầu ra của sys.dm\_tran\_locks cho thấy các khóa phạm vi dùng chung (RangeS-S) được lấy trên ba hàng chỉ mục: nhân viên đầu tiên trong GroupID = 10, nhân viên thứ hai trong GroupID = 10 và nhân viên thứ ba trong GroupID = 20. khóa phạm vi ngăn chặn sự xâm nhập của bất kỳ nhân viên mới nào trong GroupID = 10. Các khóa phạm vi vừa hiển thị giới thiệu một vài tác dụng phụ thú vị.

* Không có nhân viên mới nào có GroupID trong khoảng từ 10 đến 20 có thể được thêm vào trong giai đoạn này. Chẳng hạn, một nỗ lực để thêm một nhân viên mới với GroupID là 15 sẽ bị chặn bởi giao dịch PayBonus

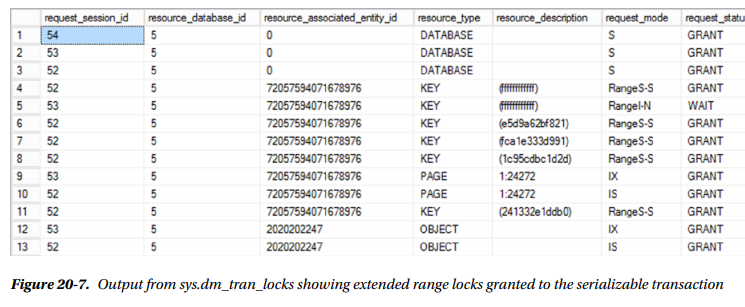


* Nếu tập dữ liệu của giao dịch PayBonus hóa ra là tập cuối cùng trong dữ liệu hiện có theo chỉ mục, thì khóa phạm vi được yêu cầu trên hàng, sau lần cuối cùng trong tập dữ liệu, được lấy trên giá trị dữ liệu cuối cùng có thể trong bảng

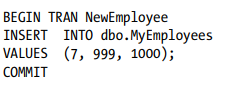
Để hiểu hành vi này, hãy để xóa các nhân viên với GroupID> 10 để tạo dữ liệu GroupID = 10 đặt tập dữ liệu cuối cùng trong clustered index (hoặc bảng).



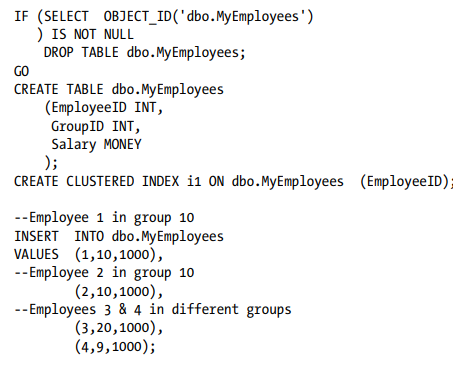
Chạy phần thưởng cập nhật và nhân viên mới một lần nữa. Hình 20-7 cho thấy kết quả đầu ra của sys.dm\_tran\_locks cho giao dịch PayBonus.



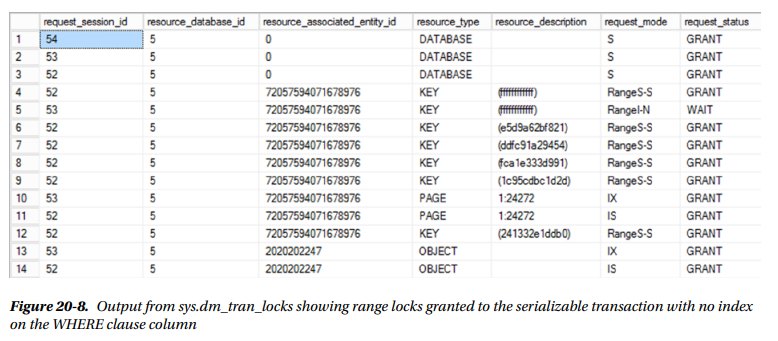
Khóa phạm vi trên hàng có thể cuối cùng (KEY = ffffffffffff) trong clustered index, như trong Hình 20-7, sẽ chặn việc thêm nhân viên với tất cả các GroupID lớn hơn hoặc bằng 10. Bạn biết rằng khóa nằm ở hàng cuối cùng, không phải vì nó được hiển thị theo kiểu hiển thị trong đầu ra của sys.dm\_tran\_locks mà bởi vì bạn đã dọn sạch mọi thứ cho đến hàng đó trước đó. Ví dụ: một nỗ lực để thêm một nhân viên mới với GroupID = 999 sẽ bị chặn bởi giao dịch PayBonus



Đoán xem điều gì sẽ xảy ra nếu bảng không có chỉ mục trên cột GroupID (nói cách khác, cột trong mệnh đề WHERE)? Trong khi bạn suy nghĩ, tôi sẽ tạo lại bảng với clustered index trên một cột khác.



Bây giờ chạy lại truy vấn tiền thưởng được cập nhật và truy vấn nhân viên mới. Hình 20-8 cho thấy kết quả đầu ra của sys.dm\_tran\_locks cho giao dịch PayBonus.



Một lần nữa, khóa phạm vi trên hàng có thể cuối cùng (KEY = ffffffffffff) trong clustered index, như trong Hình 20-8, sẽ chặn việc thêm bất kỳ hàng mới nào vào bảng. Tôi sẽ thảo luận về lý do đằng sau việc khóa rộng rãi này trong phần sau trong “Effect of Indexes on the Serializable Isolation Level”.

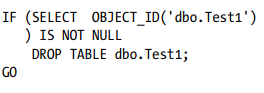
Như bạn đã thấy, mức cô lập Serializable không chỉ giữ các khóa chia sẻ cho đến khi kết thúc giao dịch như mức cô lập Repeatable Read mà còn ngăn bất kỳ hàng mới nào xuất hiện trong tập dữ liệu bằng cách giữ khóa phạm vi. Vì việc chặn tăng này có thể ảnh hưởng đến sự tương tranh của cơ sở dữ liệu, bạn nên tránh mức cô lập Serializable. Nếu bạn phải sử dụng Serializable, thì hãy chắc chắn rằng bạn có các chỉ mục và truy vấn tốt để tối ưu hóa hiệu suất nhằm giảm thiểu kích thước và thời lượng giao dịch của bạn

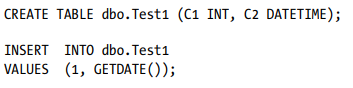
**Snapshot**

Snapshot là mức thứ hai của các mức cô lập row-versioning có sẵn trong SQL Server kể từ SQL Server 2005. Không giống như cách ly Read Committed Snapshot, cách ly Snapshot yêu cầu một cuộc gọi rõ ràng để SET TRANSACTION ISOLATION LEVEL khi bắt đầu giao dịch. Nó cũng yêu cầu thiết lập mức cô lập trên cơ sở dữ liệu. Cách ly Snapshot nhanh có nghĩa là một mức cô lập nghiêm ngặt hơn so với cách ly Read Committed Snapshot. Cách ly Snapshot sẽ cố gắng đặt một khóa độc quyền trên dữ liệu mà nó dự định sửa đổi. Nếu dữ liệu đó đã có khóa, snapshot transaction sẽ thất bại. Nó cung cấp tính nhất quán đọc ở cấp độ giao dịch, giúp nó phù hợp hơn với các hệ thống loại tài chính so với Read Committed Snapshot

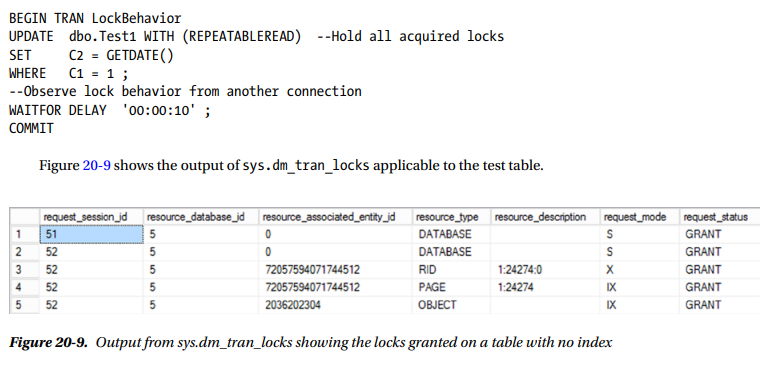
**Effect of Indexes on Locking**

Các chỉ mục ảnh hưởng đến hành vi locking trên một bảng. Trên bảng không có chỉ mục, độ chi tiết của khóa là RID, PAG (trên trang chứa RID) và TAB. Việc thêm các chỉ mục vào bảng sẽ ảnh hưởng đến các tài nguyên bị khóa. Ví dụ, hãy xem xét bảng kiểm tra sau không có chỉ mục:





Tiếp theo, quan sát hành vi khóa trên bảng cho giao dịch (--indexlock trong phần tải xuống).



Các khóa sau được giữ bởi giao dịch:

* An (IX) lock on the table
* An (IX) lock on the page containing the data row
* An (X) lock on the data row within the table

Khi resource\_type là một đối tượng, giá trị cột resource\_associated\_entity\_id trong sys.dm\_tran\_locks chỉ ra objectid của đối tượng mà khóa được đặt. Bạn có thể có được tên đối tượng cụ thể mà khóa được lấy từ bảng hệ thống sys.object, như sau:



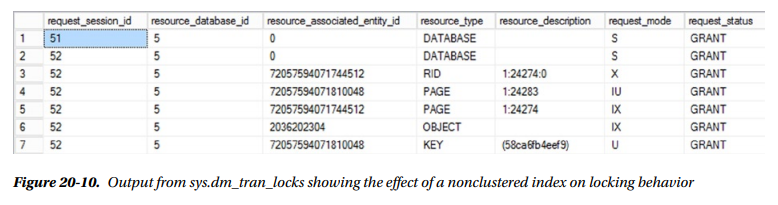
Ảnh hưởng của chỉ mục đến hành vi khóa của bảng thay đổi theo loại chỉ mục trên cột mệnh đề WHERE. Sự khác biệt nảy sinh từ thực tế là các trang lá của các nonclustered và clustered index có mối quan hệ khác với các trang dữ liệu của bảng. Chúng ta hãy xem xét ảnh hưởng của các chỉ số này đến hành vi khóa của bảng.

**Effect of a Nonclustered Index**

Vì các trang lá của chỉ mục không tách rời tách biệt với các trang dữ liệu của bảng, nên các tài nguyên được liên kết với nonclustered index cũng được bảo vệ khỏi tham nhũng. SQL Server tự động đảm bảo điều này. Để thấy điều này trong thực tế, tạo một nonclustered index trên bảng thử nghiệm.



Khi chạy lại giao dịch LockBehavior và truy vấn sys.dm\_ tran\_locks từ một kết nối riêng, bạn sẽ nhận được kết quả như trong Hình 20-10.

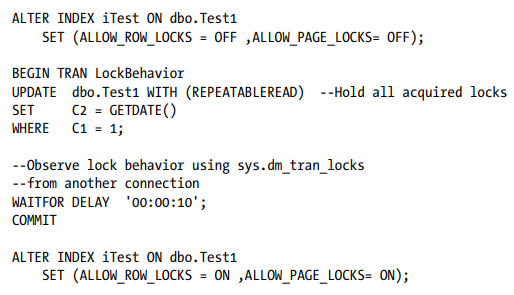


Các khóa sau được giữ bởi giao dịch:

* + An (IU) lock on the page containing the nonclustered index row
  + A (U) lock on the nonclustered index row within the index page
  + An (IX) lock on the table
  + An (IX) lock on the page containing the data row
  + An (X) lock on the data row within the data page

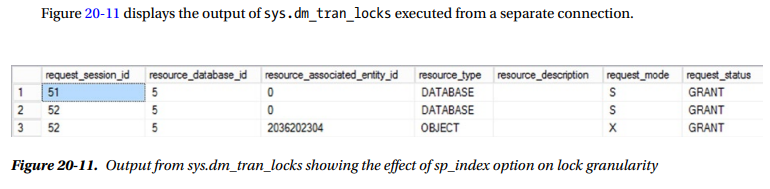
Lưu ý rằng chỉ các khóa cấp độ hàng và cấp độ trang được liên kết trực tiếp với nonclustered index. Mức độ chi tiết khóa cao hơn tiếp theo cho nonclustered index là khóa cấp bảng trên bảng tương ứng.

Do đó, các nonclustered index giới thiệu một chi phí khóa bổ sung trên bảng. Bạn có thể tránh phí khóa trên chỉ mục bằng cách sử dụng các tùy chọn ALLOW\_law\_LOCKS và ALLOW\_PAGE\_LOCKS trong ALTER INDEX. Tuy nhiên, hiểu rằng đây là một sự đánh đổi có thể làm mất hiệu suất và nó yêu cầu kiểm tra cẩn thận để đảm bảo nó không ảnh hưởng tiêu cực đến hệ thống của bạn.



Bạn có thể sử dụng các tùy chọn này khi làm việc với một chỉ mục để bật / tắt các khóa KEY và khóa PAG trên chỉ mục. Chỉ vô hiệu hóa khóa KEY làm cho độ chi tiết của khóa thấp nhất trên chỉ mục là khóa PAG. Cấu hình độ chi tiết của khóa trên chỉ mục vẫn có hiệu lực cho đến khi được cấu hình lại.

Sửa đổi khóa như thế này sẽ là giải pháp cuối cùng sau khi nhiều lựa chọn khác đã được thử. Điều này có thể gây ra chi phí khóa đáng kể sẽ ảnh hưởng nghiêm trọng đến hiệu suất của hệ thống.



Khóa duy nhất có được bởi giao dịch trên bảng thử nghiệm là khóa (X) trên bàn.

Bạn có thể thấy từ hành vi khóa mới vô hiệu hóa khóa KEY nâng mức độ chi tiết của khóa lên mức bảng.

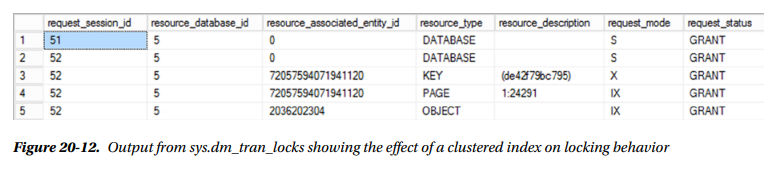
Điều này sẽ chặn mọi truy cập đồng thời vào bảng hoặc các chỉ mục trên bảng; do đó, nó có thể làm tổn thương nghiêm trọng đồng thời cơ sở dữ liệu. Tuy nhiên, nếu chỉ mục không bao gồm trở thành điểm tranh chấp trong kịch bản chặn, thì có thể có ích khi vô hiệu hóa khóa PAG trên chỉ mục, do đó chỉ cho phép khóa KEY trên chỉ mục

**Effect of a Clustered Index**

Vì đối với clustered index, các trang lá của chỉ mục và các trang dữ liệu của bảng là như nhau, nên chỉ mục được phân cụm có thể được sử dụng để tránh chi phí khóa các trang bổ sung (trang lá) và các hàng được giới thiệu bởi một chỉ mục không bao gồm. Để hiểu chi phí khóa liên kết với một chỉ mục được nhóm, hãy chuyển đổi chỉ mục không bao gồm trước đó thành một clustered index.



Nếu bạn chạy lại tập lệnh khóa và truy vấn sys.dm\_tran\_locks trong một kết nối khác, bạn sẽ thấy kết quả đầu ra cho giao dịch LockBehavior trên iTest được hiển thị trong Hình 20-12.



Các khóa sau được mua bởi giao dịch:

* An (IX) lock on the table
* An (IX) lock on the page containing the clustered index row
* An (X) lock on the clustered index row within the table or clustered index

Các khóa trên hàng chỉ mục được nhóm và trang lá thực sự là các khóa trên hàng dữ liệu và trang dữ liệu, vì các trang dữ liệu và trang lá giống nhau. Do đó, chỉ mục được nhóm giảm chi phí khóa trên bảng so với chỉ mục không bao gồm. Giảm chi phí khóa của một chỉ mục được nhóm là một lợi ích khác của việc sử dụng một chỉ mục được nhóm trong một đống

**Effect of Indexes on the Serializable Isolation Level**

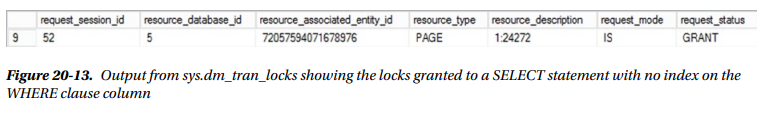
Các chỉ mục đóng một vai trò quan trọng trong việc xác định mức độ chặn gây ra bởi mức cô lập Serializable.

Tính khả dụng của một chỉ mục trên cột mệnh đề WHERE (khiến tập dữ liệu bị khóa) cho phép SQL Server xác định thứ tự của các hàng sẽ bị khóa. Ví dụ, hãy xem xét ví dụ được sử dụng trong phần về mức cô lập Nối tiếp. Câu lệnh SELECT sử dụng bộ lọc trên cột GroupID để tạo tập dữ liệu của nó, như vậy:



Một clustered index có sẵn trên cột GroupID, cho phép SQL Server có được khóa (RangeS-S) trên hàng được truy cập và hàng tiếp theo theo đúng thứ tự.

Nếu chỉ mục trên cột GroupID bị xóa, thì SQL Server không thể xác định các hàng mà các khóa phạm vi sẽ được lấy do thứ tự của các hàng không còn được đảm bảo. Do đó, câu lệnh SELECT thu được khóa (IS) ở cấp độ bảng thay vì có được các khóa có độ chi tiết thấp hơn ở cấp hàng, như trong Hình 20-13.



Bằng cách không có chỉ mục trên filter column, bạn tăng đáng kể mức độ chặn gây ra bởi mức cách ly Serializable. Đây là một lý do tốt để có một chỉ mục trên các cột mệnh đề WHERE.

**Capturing Blocking Information**

Mặc dù việc chặn là cần thiết để cách ly một giao dịch khỏi các giao dịch đồng thời khác, đôi khi nó có thể tăng đến mức quá mức, ảnh hưởng xấu đến sự tương tranh của cơ sở dữ liệu. Trong kịch bản chặn đơn giản nhất, khóa có được bởi một phiên trên tài nguyên sẽ chặn một phiên khác yêu cầu khóa không tương thích trên tài nguyên. Để cải thiện sự tương tranh, điều quan trọng là phân tích nguyên nhân chặn và áp dụng độ phân giải phù hợp.

Trong kịch bản chặn, bạn cần có các thông tin sau để hiểu rõ về nguyên nhân gây ra chặn:

* Thông tin kết nối của các phiên chặn và chặn: Bạn có thể lấy thông tin này từ chế độ xem quản lý động sys.dm\_os\_waiting\_t Nhiệm hoặc quy trình lưu trữ hệ thống sp\_who2.
* Thông tin khóa của các phiên chặn và chặn: Bạn có thể lấy thông tin này từ sys.dm\_tran\_locks DMO.
* Các câu lệnh SQL được thực thi lần cuối bởi các phiên chặn và chặn: Bạn có thể sử dụng sys.dm\_exec numquests DMV kết hợp với sys.dm\_exec\_sql tựa và sys.dm\_exec\_queryplan hoặc Sự kiện mở rộng để có được thông tin này khi chặn:

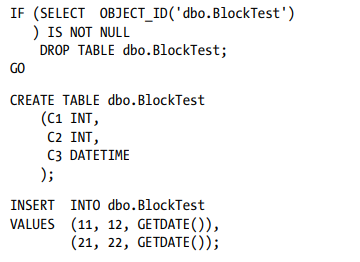
Bạn cũng có thể lấy thông tin sau từ SQL Server Management Studio bằng cách chạy Activity Monitor. Trang Processes cung cấp thông tin kết nối của tất cả các SPID. Điều này cho thấy SPIDS bị chặn, quá trình chặn chúng và người đứng đầu của bất kỳ chuỗi chặn nào có thông tin chi tiết về quá trình đã chạy, SPID của nó và thông tin khác. Có thể đặt Extended Events hoạt động bằng cách sử dụng báo cáo chặn để thu thập nhiều thông tin tương tự. Để kiểm tra ngay lập tức về khóa, sử dụng DMO; để theo dõi mở rộng và theo dõi lịch sử, bạn sẽ muốn sử dụng Extended Events. Bạn có thể tìm thấy nhiều hơn về điều này trong phần “Extended Events and the blocked\_process\_report Event”. Để cung cấp thêm sức mạnh và tính linh hoạt cho quá trình thu thập thông tin chặn, quản trị viên SQL Server có thể sử dụng các tập lệnh SQL để cung cấp thông tin liên quan được liệt kê ở đây

**Capturing Blocking Information with SQL**

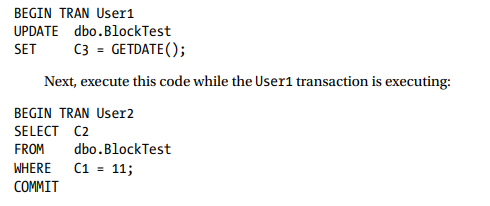
Để có đủ thông tin về các quy trình bị chặn và chặn, bạn có thể đưa một số chế độ xem quản lý động vào chơi. Truy vấn này sẽ hiển thị thông tin cần thiết để xác định các quy trình bị chặn dựa trên những quy trình đang chờ. Bạn có thể dễ dàng thêm bộ lọc để chỉ truy cập vào các quy trình bị chặn trong một khoảng thời gian nhất định hoặc chỉ trong các cơ sở dữ liệu nhất định, trong số các tùy chọn khác



Để hiểu cách phân tích một kịch bản chặn và thông tin liên quan được cung cấp bởi tập lệnh chặn, hãy xem xét ví dụ sau. Đầu tiên, tạo một bảng thử nghiệm.



Bây giờ mở ba kết nối và chạy hai truy vấn sau đây đồng thời. Khi bạn chạy chúng, hãy sử dụng tập lệnh chặn trong kết nối thứ ba. Thực thi mã sau trong một kết nối:



Điều này tạo ra một kịch bản chặn đơn giản trong đó giao dịch User1 chặn giao dịch User2.

Đầu ra của tập lệnh chặn cung cấp thông tin ngay lập tức hữu ích để bắt đầu giải quyết các vấn đề chặn.

Đầu tiên, bạn có thể xác định thông tin phiên cụ thể, bao gồm ID phiên của cả phiên chặn và phiên chờ. Bạn nhận được một mô tả tài nguyên ngay lập tức từ tài nguyên chờ, loại chờ và thời gian tính bằng mili giây mà quy trình đã chờ. Nó có giá trị cho phép bạn cung cấp bộ lọc để loại bỏ các khối ngắn hạn, là một phần của xử lý thông thường.

Tên cơ sở dữ liệu được cung cấp vì việc chặn có thể xảy ra ở bất kỳ đâu trong hệ thống, không chỉ trong AdventureWorks2012. Bạn muốn nhận ra nó ở nơi nó xảy ra. Các tài nguyên và loại từ khóa cơ bản thông tin được lấy cho quá trình chờ đợi.

Loại yêu cầu chặn được hiển thị và cả T-SQL đang chờ và T-SQL đang chờ, nếu có, đều được hiển thị. Khi bạn có đối tượng xảy ra khối, có T-SQL để bạn có thể hiểu chính xác vị trí và cách quá trình chặn hoặc bị chặn là một phần quan trọng của quá trình loại bỏ hoặc

giảm số lượng chặn. Tất cả thông tin này có sẵn từ một truy vấn đơn giản. Hình 20-14 cho thấy đầu ra mẫu từ quá trình bị chặn trước đó.

