. **Chỉ mục (Index) trong SQL**

*Chỉ mục (Index) là bảng tra cứu đặc biệt mà Database Search Engine có thể sử dụng để tăng nhanh thời gian và hiệu suất thu thập dữ liệu. Hiểu đơn giản, một chỉ mục là một con trỏ tới dữ liệu trong một bảng. Một chỉ mục trong một Database là tương tự như một chỉ mục trong Mục lục của cuốn sách.*

Ví dụ, nếu bạn muốn tham chiếu tất cả các trang trong một cuốn sách về một chủ đề nào đó, đầu tiên bạn nghĩ ngay đến mục lục của nó, mà liệt kê tất cả các chương, chủ đề theo thứ tự và sau đó được tham chiếu tới một hoặc nhiều trang cụ thể.

Một chỉ mục giúp tăng tốc các truy vấn SELECT và các mệnh đề WHERE, nhưng nó làm chậm việc dữ liệu nhập vào, với các lệnh UPDATE và INSERT. Các chỉ mục có thể được tạo hoặc xóa mà không ảnh hưởng tới dữ liệu.

Tạo một chỉ mục bởi sử dụng lệnh CREATE INDEX, mà cho phép bạn đặt tên cho chỉ mục, xác định bảng và cột hoặc các cột nào để lập chỉ mục, và để chỉ dẫn chỉ mục là theo thứ tự tăng dần hoặc giảm dần.

Các chỉ mục cũng có thể là duy nhất, tương tự như ràng buộc UNIQUE, trong đó chỉ mục ngăn ngừa các bản sao của các bản ghi trong cột hoặc việc tổ hợp của các cột có chỉ mục.

**Lệnh CREATE INDEX trong SQL**

Cú pháp cơ bản của lệnh CREATE INDEX trong SQL là như sau:

CREATE INDEX ten\_chi\_muc ON ten\_bang;

Single-Column Index trong SQL

Chỉ mục cho một cột đơn là một chỉ mục được tạo dựa trên chỉ một cột trong bảng. Cú pháp cơ bản là:

CREATE INDEX ten\_chi\_muc

ON ten\_bang (ten\_cot);

**Unique index trong SQL**

Unique index được sử dụng không chỉ để tăng hiệu suất, mà còn cho mục đích toàn vẹn dữ liệu. Một Unique index không cho phép bất kỳ bản sao giá trị nào được chèn vào trong bảng. Cú pháp cơ bản là:

CREATE UNIQUE INDEX ten\_chi\_muc

on ten\_bang (ten\_cot);

**Composite Index trong SQL**

Composite Index là một chỉ mục cho hai hoặc nhiều cột trong một bảng. Cú pháp cơ bản của nó như sau:

CREATE INDEX ten\_chi\_muc

on ten\_bang (cot1, cot2);

Việc tạo một Single-column index hoặc một Composite index tùy thuộc vào sự cân nhắc của bạn về các cột bạn có thể sử dụng thường xuyên trong mệnh đề WHERE của một truy vấn như là các điều kiện lọc.

Nếu chỉ có một cột được sử dụng, thì lựa chọn tốt nhất là Single-column index. Nếu có hai hoặc nhiều cột được sử dụng thường xuyên trong mệnh đề WHERE như là các bộ lọc, thì dạng chỉ mục Composite index là lựa chọn tối ưu.

**Implicit Index trong SQL**

Implicit Index (có thể hiểu là chỉ mục ngầm định) là chỉ mục mà được tạo tự động bởi Database Server khi một đối tượng được tạo. Các chỉ mục được tạo tự động cho các ràng buộc Primary key và các ràng buộc Unique.

**Lệnh DROP INDEX trong SQL**

Một chỉ mục có thể bị xóa bởi sử dụng lệnh DROP INDEX trong SQL. Bạn nên cẩn thận trong khi xóa một chỉ mục, bởi vì khi đó hiệu suất có thể chậm hơn hoặc không được cải thiện.

Cú pháp cơ bản của lệnh DROP INDEX là như sau:

DROP INDEX ten\_chi\_muc;

Bạn có thể tìm hiểu về Ràng buộc INDEX trong SQL để theo dõi các ví dụ về chỉ mục.

<https://www.youtube.com/watch?v=n2xum3GsoDY&t=599s>

**Index Giúp Tăng Hiệu Năng Thực Hiện Như Thế Nào ?**

*Index là phương tiện rất mạnh để tăng hiệu năng thực hiện của câu lệnh. Bài post này sẽ cung cấp một ví dụ cho bạn thấy bên trong SQL Server sử dụng index để  tăng hiệu năng như thế nào.*

Dùng database **AdventureWork** là database mẫu đi kèm với SQL Server

Trước hết ta hãy dùng bảng Sale.Customer để tạo ra hai bảng mới là Sale.Customer\_noIndex và Sale.Customer\_Index, đồng thời tạo 1 index trên trường CustomerID cho bảng Sale.Customer\_Index:

SELECT \*

INTO Sales.Customer\_NoIndex

FROM Sales.Customer

SELECT \*

INTO Sales.Customer\_Index

FROM Sales.Customer

GO

CREATE INDEX Idx\_Customer\_Index\_CustomerID ON Sales.Customer\_Index(CustomerID)

Nay ta có hai câu lệnh SELECT sau để truy vấn hai bảng:

-- #1

SELECT CustomerID, CustomerType

FROM Sales.Customer\_NoIndex

WHERE CustomerID = 11001

-- #2

SELECT CustomerID, CustomerType

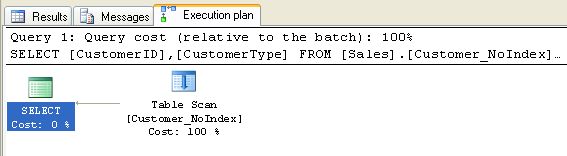
FROM Sales.Customer\_Index

WHERE CustomerID = 11001

Hai câu lệnh này sẽ cho cùng kết quả, khác biệt duy nhất là câu lệnh thứ hai truy vấn bảng Sales.Customer\_Index có index trên trường cần tìm (CustomerID). Ta sẽ xem hai câu lệnh trên được thực hiện như thế nào bằng cách nhìn vào kế hoạch thực thi (execution plan) của chúng. Khi bắt đầu thực hiện một câu lệnh, SQL Server lên một kế hoạch gồm các bước sẽ tiến hành để thực thi câu lệnh đó, gọi là kế hoạch thực thi.

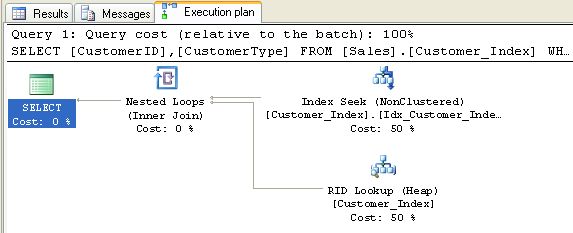
Trên hàng công cụ bạn hãy bấm vào nút “Include Actual Execution Plan”. Khi đó, mỗi lần bạn chạy câu lệnh hệ thống sẽ vừa thực hiện câu lệnh vừa đồng thời trả lại kế hoạch thực thi mà nó đã dùng để thực hiện câu lệnh đó.

Bạn hãy bôi đen **câu lệnh thứ nhất** và thực hiện nó, ở tab “Execution plan” hiện ra kế hoạch thực thi như thế này:

[](http://sqlviet.com/blog/wp-content/uploads/2010/04/tablescan1.jpg)

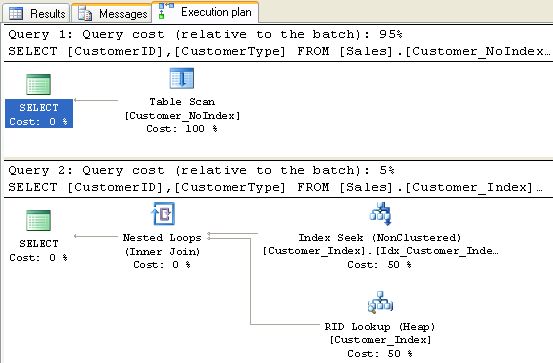
Như vậy ta thấy hệ thống sẽ thực thi câu lệnh bằng cách duyệt qua cả bảng (table scan) và tìm ra các bản ghi thỏa mãn yêu cầu tìm kiếm. Thao tác duyệt bảng có nghĩa là hệ thống cần phải đọc tuần tự từng bản ghi từ đầu đến cuối để tìm ra kết quả. Trong trường hợp này, nó phải đọc toàn bộ 19 185 bản ghi và tìm ra bản ghi có CustomerID=11011. Đây là một thao tác rất chậm vì nó phải xử lý tất cả các bản ghi trong bảng. Nên nhớ hệ thống sẽ không dừng lại khi nó tìm được bản ghi đầu tiên có CustomerID=11011, vì nó không biết liệu còn bản ghi nào khác có giá trị CustomerID tương tự hay không, cho nên để chắc chắn trả lại kết quả đầy đủ hệ thống vẫn phải tiếp tục đọc các bản ghi còn lại. Ta có thể nhận xét thấy chi phí của thao tác duyệt bảng tăng tuyến tính cùng với số lượng bản ghi trong bảng (độ phức tạp là O(n)).

Giờ ta hãy thực hiện **câu lệnh thứ hai**, lần này kế hoạch thực thi sẽ như sau:

[](http://sqlviet.com/blog/wp-content/uploads/2010/04/index-seek.jpg)

Lần này ta không thấy thao tác table scan nữa, mà thay vào đó là index seek và RID lookup. Index seek là khi hệ thống có thể nhảy đến được node trên cây index chứa khóa thỏa mãn yêu cầu tìm kiếm. Index là một cấu trúc dữ liệu có dạng B-tree, nên nó rất thích hợp với các thao tác tìm kiếm theo kiểu *key=value*, chỉ cần vài phép so sánh là hệ thống định vị được node chứa khóa cần tìm. Node này chứa khóa (trường được index, ở đây là giá trị của CustomerID) và RID là ID của bản ghi tương ứng trong bảng (đây là giá trị nội bộ chỉ dùng bên trong hệ thống, ta không truy cập được giá trị này). Vì thế bước tiếp theo là dùng RID này để nhảy đến bản ghi tương ứng trong bảng (RID lookup) để lấy các trường dữ liệu cần thiết. Với index seek, độ phức tạp giảm xuống thành O(logn), một bước tiến vượt bậc so với table scan.

Ta có thể so sánh chi phí của hai câu lệnh trên bằng cách thực hiện cả hai cùng nhau:

[](http://sqlviet.com/blog/wp-content/uploads/2010/04/combine.jpg)

Ta thấy câu lệnh thứ nhất chiếm tới 95% tổng chi phí, trong khi câu lệnh thứ hai chỉ chiếm có 5%. Nói cách khác, index trên trường CustomerID đã giúp cho câu lệnh thực hiện nhanh lên đến 19 lần. Index đã giúp cho lượng dữ liệu hệ thống cần xử lý để tìm ra kết quả giảm xuống đến mức tối thiểu, và điều đó đã tạo ra bước nhảy về tốc độ. Từ đây ta rút ra một bài học quan trọng: Các trường thường được dùng trong mệnh đề WHERE là các ứng cử viên đầu tiên cần được tạo index.

**Khi nào nên tránh sử dụng chỉ mục trong SQL?**

Mặc dù các chỉ mục nhằm mục đích để nâng cao hiệu suất của Database, nhưng đôi khi, bạn nên tránh dùng chúng. Dưới đây là một số điểm bạn cần xem xét để quyết định có nên sử dụng chỉ mục hay không:

Các chỉ mục không nên được sử dụng trong các bảng nhỏ.

Bảng mà thường xuyên có các hoạt động update, insert.

Các chỉ mục không nên được sử dụng trên các cột mà chứa một số lượng lớn giá trị NULL.

Không nên dùng chỉ mục trên các cột mà thường xuyên bị sửa đổi.

**Index không sử dụng trong các trường hợp nào ?**

- Sử dụng Index trong các trường hợp như sau:

+ Đối với bảng mà khi truy vấn mà cần sử dụng câu lệnh WHERE, JOIN với ON hay các câu lệnh so sánh có điều kiện thường xuyên trên một cột nào đó, nên tạo index ở những cột này để truy vấn nhanh hơn.

+ Bản chất của Index là tạo một bảng ánh xạ dữ liệu đến bảng có cột được Index, bảng ánh xạ này đã được sắp xếp có thứ tự để khi truy vấn với câu lệnh có điều kiện so sánh như WHERE hay JOIN sẽ nhanh hơn. Đó là áp dụng thuật toán BTREE ( google để tìm hiểu sự liên quan của thuật toán này với index) . Và vấn đề ở đây đó là việc tại sao trong bài viết có nói đến vấn đề INSERT hay UPDATE. Khi INSERT một bản ghi vào bảng trong database, những cột nào có chứa index thì sẽ có bảng ánh xạ index tương ứng, bảng ánh xạ này như đã nói ở trên thì luôn được sắp xếp có thứ tự, vậy nên khi một bản ghi được INSERT thì hệ thống sẽ phải tìm kiếm thứ tự để chèn bản ghi cho bảng ánh xạ tương ứng .

"Nghĩa là việc INSERT trên một bảng có Index thì đồng nghĩa với việc bảng đó phải thực hiện chức năng tìm kiếm vị trí để INSERT trên bảng ánh xạ tương ứng với cột Index".

Khi trong bảng có càng nhiều bản ghi thì việc INSERT cũng trở nên lâu hơn với những bản ghi được INSERT phía sau cùng , do việc INSERT thì việc tìm kiếm vị trí thích hợp cho bảng ánh xạ Index cũng mất thời gian hơn, do đó bạn có thể thấy là trong hệ thống SQL luôn có cột cơ bản và hay dùng đó là cột "id" với khoá PRIMARY và AUTO INCREMENT hay AUTO IDENTITY hay gọi là khoá chính và mã tự tăng như thường gọi.

Cột PRIMARY là một Index , còn phần tự tăng thì luôn tăng theo số lần INSERT, nghĩa là mã tự tăng bắt đầu từ 1 chẳng hạn, khi INSERT lần thứ 5 thì sẽ mã sẽ là 5, nhưng khi xoá 4 bản ghi thì nó vẫn sẽ tăng tiếp là 6 , nói chung là mã tự tăng sẽ tăng theo số lần INSERT . Vì vậy nên nó cũng INSERT vào bảng ánh xạ luôn, nhưng nó luôn chọn giá trị lớn nhất là số lần INSERT cuối cùng nên nó chẳng cần tìm kiếm vị trí để INSERT mà luôn luôn INSERT vào cuối cùng . Vì vậy lợi thế của việc mã tự tăng là như vậy đó. Bạn cứ INSERT thoải mái đi, không lo về vấn đề hiệu suất, hệ thống database cần INSERT dữ liệu rất lớn, thì bạn có thể sử dụng BIGINT với số lớn nhất có thể lên đến 2^64 .

+ Đối với các cột cần Index thì tuỳ vào kiểu dữ liệu của nó nữa, kiểu có số byte càng lớn thì việc INSERT nhiều dữ liệu về sau càng lâu hơn (trừ cột có kiểu int hay bigint tự tăng) . Kiểu dữ liệu cơ bản cần Index đó là kiểu CHAR hay VARCHAR:

Ví dụ bạn cần Index cho cột "tên khách hàng" cho bảng đơn hàng , bảng đơn hàng thì một cửa hàng lớn trong một ngày bán được khoảng vài nghìn sản phẩm , tính theo năm hay tháng thì số đơn hàng cũng "khá" là lớn đấy . trong đơn hàng thì phải có "khách hàng" để giao hàng , cột khách hàng là quan trọng nhất trong đơn hàng và thường có các câu lệnh truy vấn đơn hàng mà liên quan đến cột khách hàng như "Khách hàng có nhiều đơn hàng nhất", "khách hàng có tổng giá trị đơn hàng lớn nhất" v.v... Vậy thì bạn cần phải Index cột "khách hàng" này . CỘt này có kiểu dữ liệu là CHAR hoặc VARCHAR đi . Mình xin đưa ra một vài lưu ý khi sử dụng Index nên kiểu dữ liệu dạng "text":

- Index giới hạn độ dài của "text", như MYSQL là 767 byte thôi

- Đối với MYSQL, MS SQL, v.v... thì nên đặt kiểu dữ liệu là CHAR(số ký tự) . kiểu CHAR sẽ chiếm không gian định sẵn chứ không giãn nở như kiểu VARCHAR , tuy nhiên thì cột Index không cần có số byte quá lớn nên kiểu CHAR là hợp lý nhất.

\* Đối với các bảng có trường là text và thường xuyên phải thực hiện các câu query tìm kiếm liên quan tới trường text này thì mình khuyên các bạn nên tìm hiểu về một số FTS (Full Text Search) . Là một engine hỗ trợ mạnh trong việc tìm kiếm "text".