**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**CƠ SỞ THÀNH PHỐ HỒ CHÍ MINH**



***Ěề Tài Nghiên Cứu Khoa học***

**ĚIỀU CHỈNH HIỆU SUẤT TRUY VẤN SQL**

Mã số: **18-2023-HV-TH2**

Chủ trì đề tài:

# ThS. LƯU NGUYỄN Kǵ THƯ

Tháng 11-2023

### MỤC LỤC

Phần 1: Mục tiêu – Nội dung đề tài 3

1. Mục tiêu 3
2. [Nội dung 3](#_TOC_250009)

[Phần 2: ĚÁNH GIÁ HIỆU SUẤT TRUY VẤN 4](#_TOC_250008)

* 1. [Kế hoạch thực thi truy vấn (SQL Execution Plan) 4](#_TOC_250007)
  2. [Cấu trúc dữ liệu của Index data 8](#_TOC_250006)
  3. [Ěọc và phân tích execution plans 10](#_TOC_250005)
  4. [Một số phép toán trong Execution Plan ảnh hưởng đến tốc độ truy vấn 12](#_TOC_250004)
  5. [Hiệu chỉnh cách viết truy vấn 15](#_TOC_250003)

[Phần 3: DEMO 20](#_TOC_250002)

[Kết luận 22](#_TOC_250001)

[Tài liệu tham khảo 22](#_TOC_250000)

**PHẦN 1** . **MỤC TIÊU – NỘI DUNG ĚỀ TÀI**

1. **MỤC TIÊU:** Tìm hiểu các phương pháp nâng cao hiệu suất truy vấn dữ liệu trong SQL Server

### NỘI DUNG :

* Tìm hiểu các công cụ theo dõi, phân tích quá trình truy vấn dữ liệu của 1 lệnh Select.
* Ứng dụng các công cụ phân tích vào cơ sở dữ liệu cụ thể, và đưa ra các phương pháp để tối ưu hóa truy vấn theo 2 cách: hiệu chỉnh các option trong cơ sở dữ liệu hỗ trợ truy vấn, và hiệu chỉnh cách viết lệnh truy vấn

### PHẦN 2: ĚÁNH GIÁ HIỆU SUẤT TRUY VẤN

### Kế hoạch thực thi truy vấn (SQL Execution Plan)

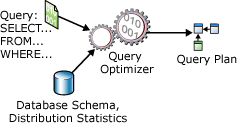
Ěể thực hiện truy vấn, SQL Server Database Engine phải phân tích lệnh để xác định phương pháp hiệu quả truy xuất dữ liệu cần thiết, và xử lý. Quá trình phân tích được quản lý bởi 1 thành phần gọi là tối ưu truy vấn (Query Optimizer). Input của Query Optimizer bao gồm: query, database schema (table and index), và database statistics. Query Optimizer tạo ra kế hoạch thực thi truy vấn (*query execution plans)*, còn được gọi *query plans* hoặc *execution plans.*

Query Optimizer sẽ chọn 1 kế hoạch để cân bằng thời gian dịch lệnh và kế hoạch tối ưu để

tìm ra kế hoạch truy vấn tốt.

Input và Output của Query Optimizer trong lúc tối ưu 1 Query đơn giản được minh họa

trong sơ đồ sau:



Một query execution plan có thể có các phần như:

### Thứ tự các table được truy xuất:

Thông thường, câu lệnh truy vấn có nhiều table thì database server sẽ truy xuất base table để xây dựng tập kết quả.Chẳng hạn như, nếu 1 lệnh SELECT cần 3 tables: đầu tiên, database server truy xuất TableA, dùng dữ liệu từ TableA để trích các mẫu tin khớp từ TableB, và tiếp tục dùng dữ liệu từ TableB để trích dữ liệu khớp từ TableC. Các thứ tự khác mà database server cǜng có thể truy xuất:

TableC, TableB, TableA, hoặc TableB, TableA, TableC, hoặc TableB, TableC, TableA, hoặc TableC, TableA, TableB

### Cách thức dùng để truy xuất dữ liệu từ mỗi table:

Có nhiều cách thức khác nhau để truy xuất dữ liệu trong table: **Table Scan**, **Index Scan, Index Seek, Key Lookup** . Tùy lược đồ cơ sở dữ liệu mà *query plans* dùng phương pháp quét lấy dữ liệu khác nhau.

Nếu chỉ cần một vài hàng có giá trị khóa cụ thể thì máy chủ cơ sở dữ liệu có thể sử dụng chỉ mục. Nếu tất cả các hàng trong bảng là bắt buộc, máy chủ cơ sở dữ liệu có

thể bỏ qua các chỉ mục và thực hiện table scan. Nếu tất cả các hàng trong một bảng là bắt buộc nhưng có một chỉ mục có các cột chính nằm trong ORDER BY, thì việc thực hiện index scan thay vì table scan có thể tiết kiệm thời gian hơn. Nếu một bảng rất nhỏ, table scan có thể là phương pháp hiệu quả nhất cho hầu hết mọi quyền truy cập vào bảng

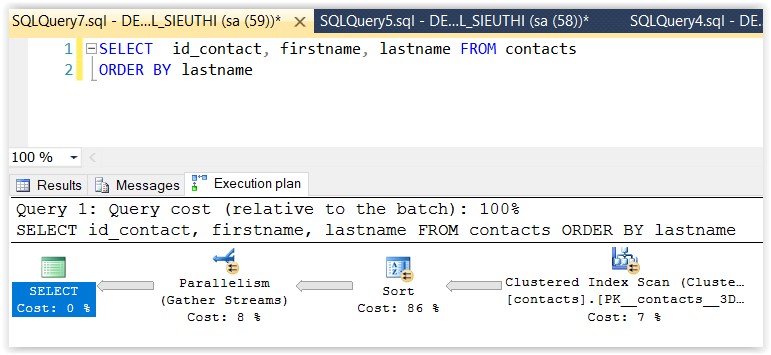
Query optimizer dựa vào số liệu thống kê phân phối khi nó ước tính chi phí tài nguyên của các phương pháp khác nhau để trích xuất thông tin từ một bảng hoặc chỉ mục. Thống kê phân phối được lưu giữ cho các cột và chỉ mục. Chúng chỉ ra tính chọn lọc của các giá trị trong một chỉ mục hoặc cột cụ thể. Ví dụ: trong bảng lưu trữ số liệu về ô tô, nhiều ô tô có cùng một nhà sản xuất nhưng mỗi ô tô có một số nhận dạng xe (vehicle identification number - VIN) duy nhất. Chỉ số trên VIN có tính chọn lọc cao hơn chỉ số về nhà sản xuất. Nếu số liệu thống kê chỉ mục không hiện hành thì trình tối ưu hóa truy vấn có thể không đưa ra lựa chọn tốt nhất cho trạng thái hiện tại của bảng.

### Kế hoạch thực thi truy vấn (SQL Execution Plan):

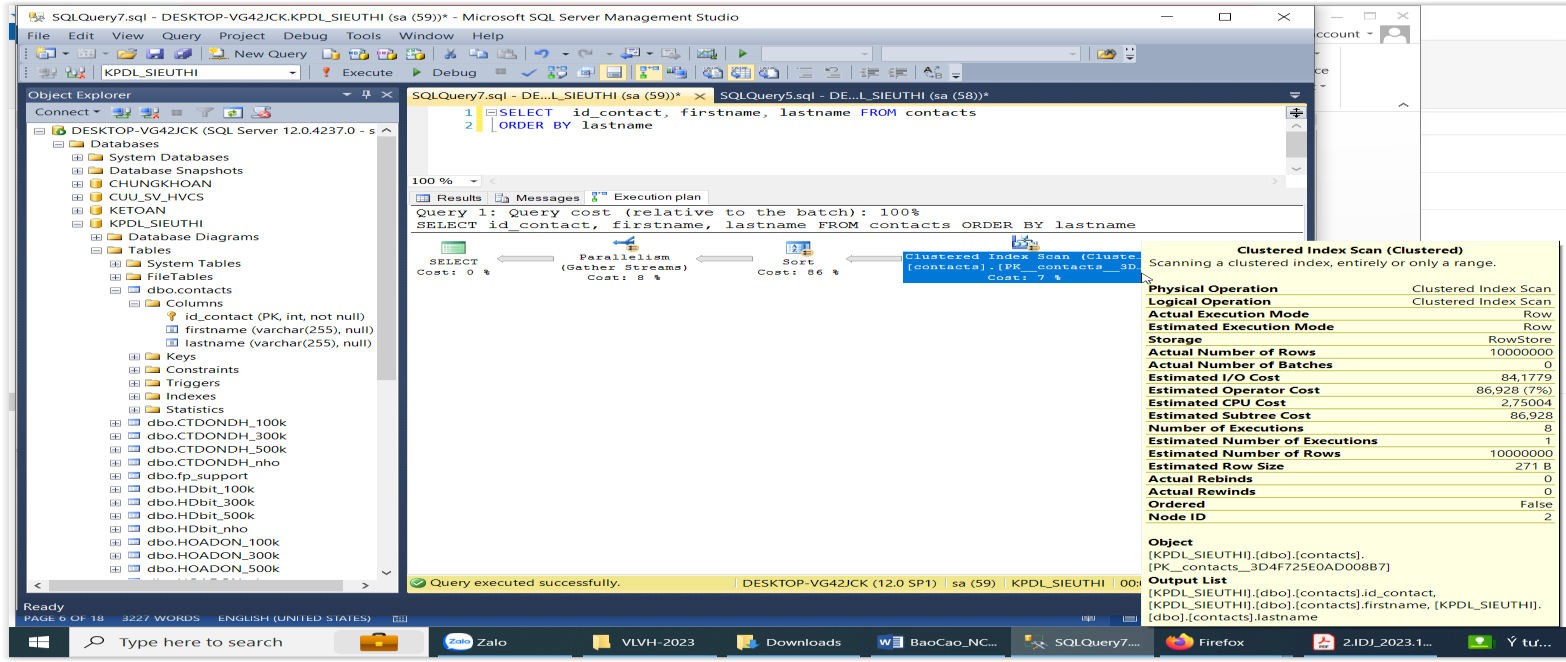
The Execution Plan trong SSMS thể hiện các phép toán được thực hiện bởi bộ xử lý truy vấn SQL. Khi ta thực hiện 1 truy vấn, bộ xử lý truy vấn SQL sẽ khởi tạo 1 kế hoạch thực thi cho truy vấn đó. SQL server cung cấp 2 loại kế hoạch thực thi: Kế hoạch thực thi dự kiến (Estimated Execution plan), Kế hoạch thực thi thực tế (Actual Execution plan) .

* Kế hoạch thực thi dự kiến : Ngay thời điểm ta xác nhận câu truy vấn, SQL Server sẽ xác lập % thời gian thực thi dự kiến của các phép toán trong câu lệnh truy vấn . Kế hoạch thực thi truy vấn được sinh trước khi thực hiện truy vấn. Thông tin về thời gian thực thi chưa có ở kế hoạch này.
* Kế hoạch thực thi thực tế : Loại kế hoạch này chỉ phát sinh khi câu lệnh truy vấn đã thực thi, và hiển thị từng bước mà SQL đã thực hiện. Nó cung cấp thông tin thời gian thực tế của từng bước.

Hình ảnh ví dụ Kế hoạch thực thi dự kiến:



Hình ảnh ví dụ Kế hoạch thực thi thực tế



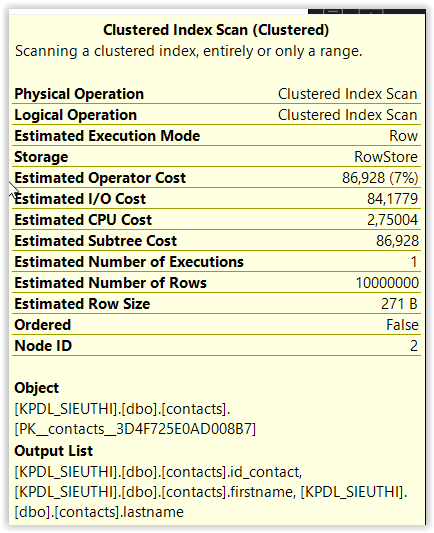
Ta đã thực thi câu truy vấn, và có 3 tab trên câu truy vấn đó. Kế hoạch thực thi câu lệnh trong hình có 3 phần:

1. Clustered Index Scan (Clustered) : cost : 7%
2. Sort Operation : cost 86%
3. Parallelism Operation : 8%

### Các thành phần (Components) của SQL Execution Plan

Vì không có dữ liệu trong table nên Kế hoạch thực thi dự kiến và Kế hoạch thực thi thực tế sẽ giống nhau, nhưng nếu dữ liệu truy vấn lớn thì ta sẽ thấy sự khác biệt. Ta có thể dựa vào đó để tối ưu hóa câu lệnh truy vấn.

Khi ta đưa trỏ chuột đến Clustered Index Scan, ta sẽ thấy chi tiết kết quả truy vấn:



1. Physical Operation: Phép toán thực hiện tên đối tượng có thể là Table Scan, *Index Seek, Clustered Index Scan*, v.v. .
2. Actual Execution Mode: mô tả kế hoạch đã thực thi.
3. Estimated Execution Mode: tương tự kế hoạch ở trên, chỉ khác ở chỗ nó là giá trị ước lượng.
4. Actual Number of Rows: số mẫu tin thực tế đã đọc
5. Estimated I/O Cost: ước lượng chi phí

input and output của tập kết quả

1. Estimated Operational Cost: ước lượng

chi phí phép toán

1. Estimated CPU Cost: đánh giá chi phí CPU thực hiện phép toán
2. Number of Executions: giá trị này chỉ có trong kế hoạch thực thi thực tế.
3. Estimated Number of Execution: tương tự như trên, nhưng đây là giá trị ước lượng.
4. Estimated Number of Rows per Execution: Số dòng sẽ được trả về.
5. Estimated Number of Rows to be Read: ước lượng từ Optimizer số dòng sẽ được đọc.
6. Estimated Row Size: ước lượng kích thước 1 dòng khi xử lý.
7. Actual Rebinds: số lần đối tượng được đánh giá lại để xử lý.
8. Ordered: xác định dữ liệu có được sắp thứ tự hay không.
9. Node ID: This follows a unique type of numbering from Right to left and then the usual Top to bottom. So, we can say that the Bottom Right will have NodeID=1 and the Top Left has Maximum Node based on the Execution Plan Tree.

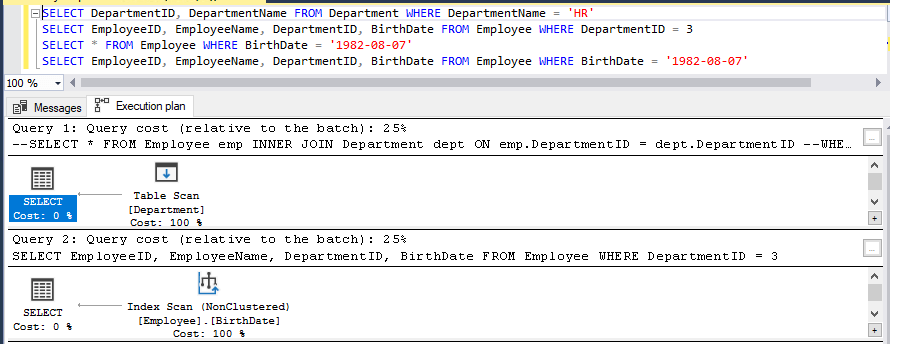
### Search Data in the Table

**Table Scan:** Quét hết tất cả các dòng của table. Table Scan chỉ có hiệu quả với table nhỏ. Ước lượng thời gian quét tỷ lệ thuận với số dòng của table.

**Index Scan:** quét dữ liệu dựa vào index, như vậy không cần quét hết các dòng

**Index Seek, Key Lookup :** đây là cách tìm dữ liệu trong table nhanh nhất, vì SQL sẽ biết dữ liệu cần tìm thuộc các trang nào, và truy xuất trực tiếp tới trang đó.

### Execution Plans giúp ta cải thiện thực hiện truy vấn trong SQL Server: Ta xét 4 lệnh truy vấn sau:



**Query 1:**

SELECT DepartmentID, DepartmentName FROM Department WHERE DepartmentName = ‘HR’

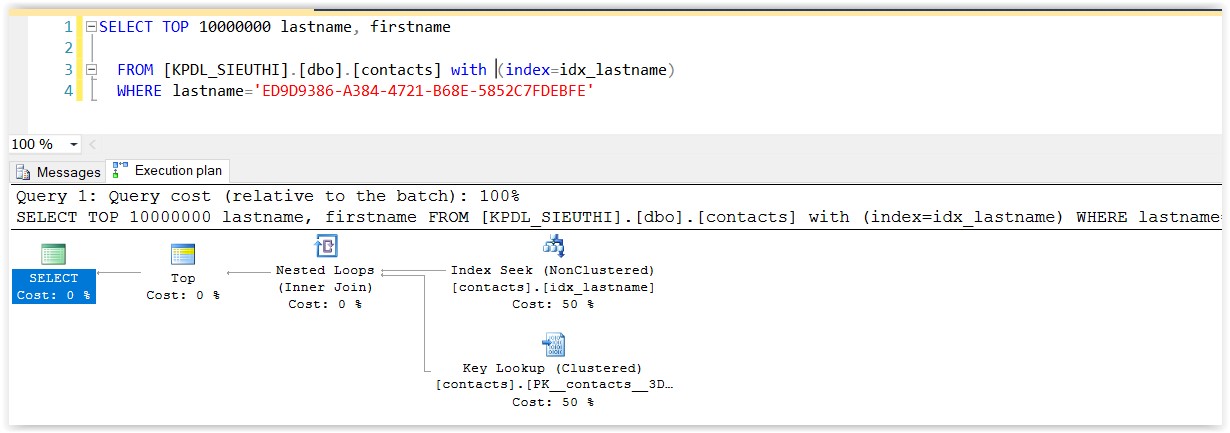
Table Department không có khóa chính, vì vậy không có clustered index. Bộ xử lý truy vấn thực hiện **table scan**. Query này mất nhiều thời gian truy vấn nhất.

**Query 2:**

SELECT EmployeeID, EmployeeName, DepartmentID, BirthDate FROM Employee WHERE DepartmentID = 3

Query này thực hiện Index scan, nhanh hơn **table scan** . Tuy nhiên, nó vẫn bị chậm nếu table có dữ liệu rất lớn.

**Query 3:**



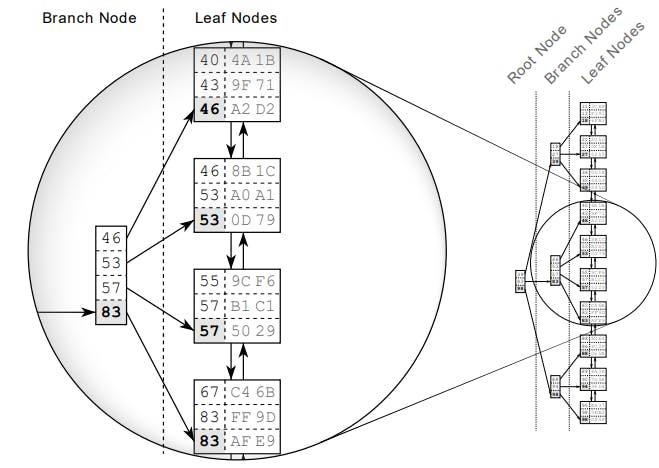
Ta đã tạo non-clustered index trên cột LastName. Lúc này, trong WHERE có điều kiện trên cột LastName thì SQL sẽ tìm dữ liệu trên LastName bằng **the index seeks**.

**Index :** Ěể truy vấn dữ liệu trong DB nhanh, thì 1 trong các yếu tố ta phải thiết lập là tạo index. Vậy Index được SQL tổ chức như thế nào mà hỗ trợ truy vấn nhanh?

Theo tìm hiểu thì các **SQL** database thường sẽ tổ chức **index** dưới dạng **B-tree** - dựa theo kiến trúc của cây cân bằng.

### Cấu trúc dữ liệu của Index data:

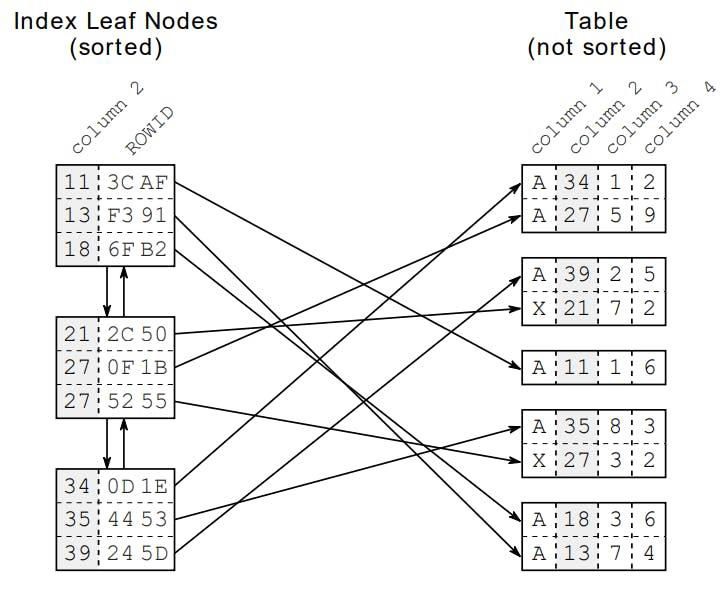
Ěể hiểu được **index**, ta cần phải biết được cấu trúc dữ liệu của một **index**. Giả sử ta tạo một index trên **column 2** kiểu số. **Database** sẽ tạo ra một dạng cấu trúc dữ liệu **B-tree** dựa theo các dữ liệu có trong **column 2** và các dữ liệu này sẽ được **sắp xếp** như hình bên dưới.



Trong hình bên trên, ta sẽ chú ý đến các **Leaf Nodes.** Các **Node** này sẽ cung cấp cho chúng

ta **địa chỉ** để đọc dữ liệu từ bảng được lưu trong cơ sở dữ liệu.

Tất nhiên sẽ có các **Leaf Nodes** không được các **Branch Node** trỏ tới nên các **Leaf Node** nãy sẽ được liên kết với nhau bằng **danh sách liên kết kép** để đảm bảo được việc duyệt dữ liệu trong cây **index**.

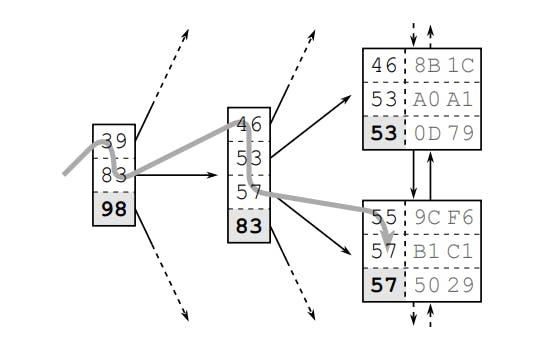


Database dùng Doubly Linked List để kết nối các *index leaf nodes*. Mỗi một *index leaf nodes* sẽ được lưu trong một database block, trong mỗi block sẽ có nhiều dữ liệu của cột được đánh index (index entry). Ěiều này có nghƿa index duy trì trật tự của nó ở 2 mức: giữa các index entry trong cùng 1 block và giữa các block với nhau

### Duyệt dữ liệu theo Index:

Tiếp đến, ta sẽ xem xét đến cách duyệt dữ liệu trên **index** để biết được tại sao nó lại nhanh hơn với việc **scan** table rất nhiều.

Giả sử ta tìm dữ liệu bản ghi có giá trị là 57. SQL sẽ bắt đầu duyệt từ gốc của cây index qua các Branch Node khác nhau cho đến khi đến đầu của Leaf Nodes. Tiếp tục duyệt theo qua các Leaf Node để tìm được giá trị 57.



Minh họa quá trình tìm kiếm một index entry có giá trị 57. Quy trình như sau:

* + - Bắt đầu từ root node với 3 giá trị: 39, 83, 98
    - So sánh 57 với giá trị đầu tiên là 39: 57 > 39, trong khi 39 là giá trị lớn nhất của branch node mà 39 trỏ đến --> 57 sẽ không nằm trong branch node đó
    - So sánh 57 với giá trị thứ hai là 83: 57 < 83 --> 57 *rất có thể* nằm trong branch node mà 83 trỏ đến --> bắt đầu tìm kiếm trong branch node này
    - Quy trình tìm kiếm lại tiếp tục tại branch node mà root node 83 trỏ đến, gồm các giá trị: 46,

53, 57, 83

* + - So sánh 57 với giá trị đầu tiên là 46: 57 > 46 --> 57 không nằm trong leaf node mà 46 trỏ đến
    - Tương tự, 57 > 53 --> 57 không nằm trong leaf node mà 53 trỏ đến
    - 57 <= 57 --> 57 sẽ nằm trong leaf node mà 57 (giá trị ở branch node) trỏ đến
    - Quy trình tìm kiếm lại lặp lại tại leaf node gồm các giá trị: 55, 57, 57
    - 57 > 55 --> không phải giá trị cần tìm --> tìm tiếp
    - 57 = 57 --> đây chính là giá trị cần tìm, ***nhưng quá trình tìm kiếm vẫn tiếp tục, vì biết đâu giá trị tiếp theo trong leaf node cǜng là 57***
    - ***57 = 57 --> Lại tìm thêm được một giá trị 57 nữa***
    - Tất các các giá trị trong leaf node đều đã được check, quá trình tìm kiếm kết thúc. Kết quả thu được là 2 giá trị 57 ứng với 2 ROW\_ID trỏ đến 2 bản ghi trong database

Quá trình tìm kiếm trên được gọi là ***tree traversal - tìm kiếm dọc theo cây.*** Tree traversal đặc biệt hiệu quả vì 2 nguyên nhân:

* + - Thứ nhất: mọi phần tử trong leaf node đều có thể được tìm ra trong cùng 1 số lượng bước nhất định
    - Thứ hai: *Độ sâu* của cây (tree depth) tĕng rất chậm so với tốc độ tĕng của số lượng leaf

nodes.

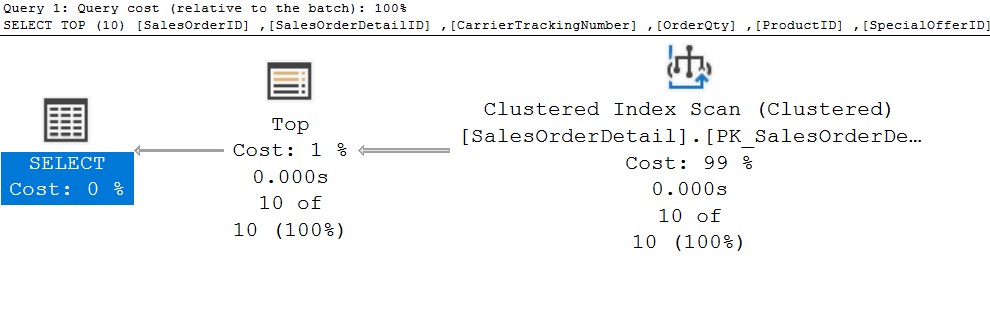
### Ěọc và phân tích execution plans

Ta đọc execution plans từ phải sang trái, bắt đầu từ phép toán ở trên bên phải. Execution plans giúp ta theo dõi luồng dữ liệu thực thi trong query .

Giả sử ta thực thi query sau, và xem kế hoạch thực thi thực tế.

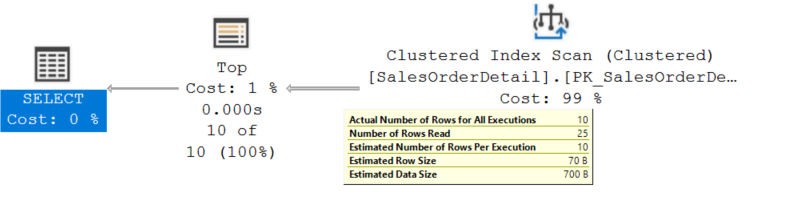
SELECT TOP (10) [SalesOrderID], [SalesOrderDetailID] ,[CarrierTrackingNumber] , [OrderQty] , [ProductID], [SpecialOfferID], [ModifiedDate]

FROM [Sales].[SalesOrderDetail] where [UnitPrice]>1000

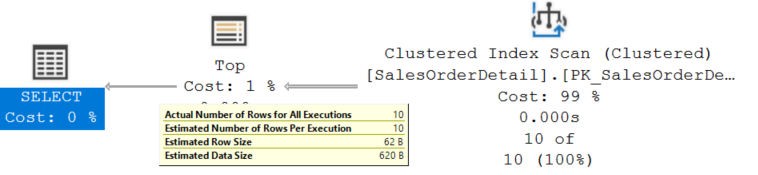


Nếu ta đọc execution plan từ phải sang trái, ta thấy có các bước sau:

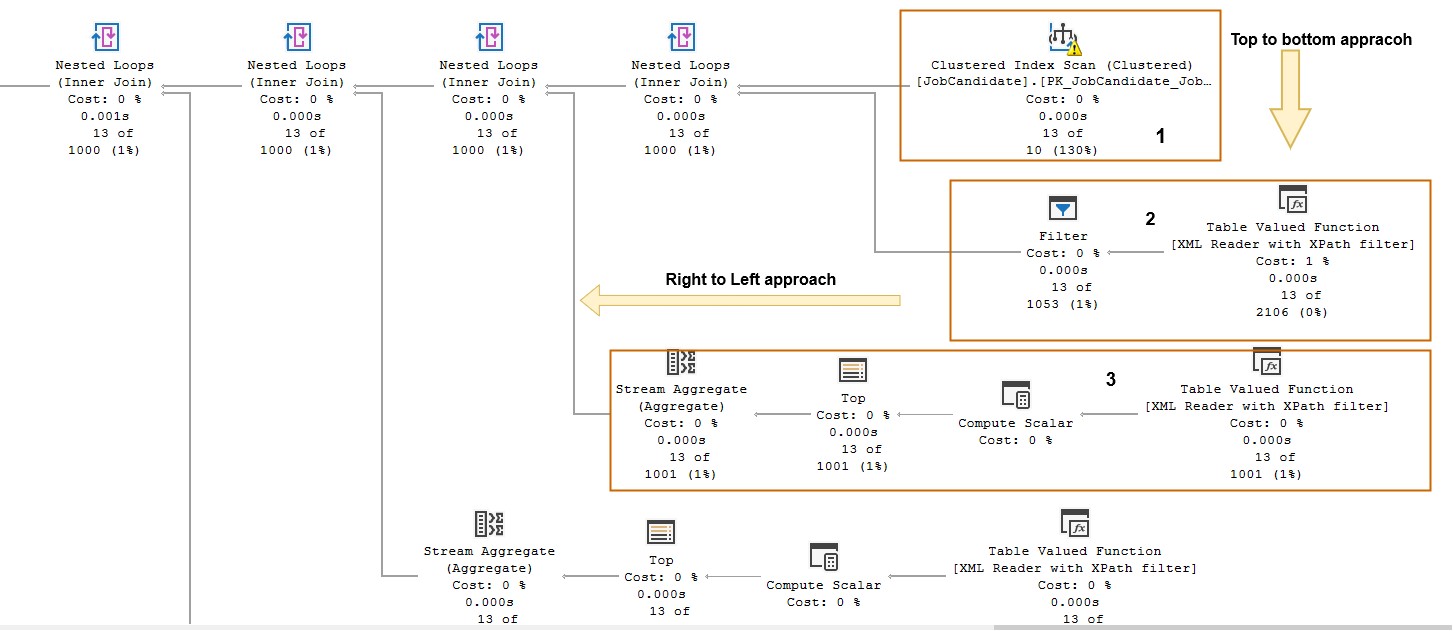
* + - Phép toán đầu tiên là Clustered Index Scan: đọc data từ [Sales].[SalesOrderDetail]. Ta sẽ thấy số dòng ước lượng và thực tế đã thực thi như hình dưới.



* + - Kết tiếp, dữ liệu từ phép toán Clustered Index Scan được chuyển sang phép toán Top với 10 rows.



Trong trường hợp có nhiều nhánh trong một kế hoạch thực hiện, ta sẽ tiếp cận từ phải sang trái, từ trên xuống dưới



### Chi phí của phép toán

Mỗi phép toán trong kế hoạch thực thi SQL Server được liên kết với một chi phí. Chi phí của phép toán có liên quan đến các chi phí khác trong kế hoạch thực hiện. Thông thường, chúng ta cần tập trung vào phép toán có chi phí cao và điều chỉnh truy vấn xung quanh nó.

### Một số phép toán trong Execution Plan ảnh hưởng đến tốc độ truy vấn:

i. **Distribute Streams:** Toán tử Distribute Streams chỉ được sử dụng trong truy vấn song song. Toán tử Distribute Streams lấy một luồng bản ghi đầu vào duy nhất và tạo ra nhiều luồng đầu ra. Nội dung và định dạng bản ghi không thay đổi. Mỗi bản ghi từ luồng đầu vào xuất hiện ở một trong các luồng đầu ra. Toán tử này tự động giữ nguyên thứ tự tương đối của các bản ghi đầu vào trong luồng đầu ra. Thông thường, hàm bĕm được sử dụng để quyết định xem bản ghi đầu vào cụ thể thuộc về luồng đầu ra nào.

1. **Gather Streams**: Toán tử Gather Streams chỉ được sử dụng trong truy vấn song song. Toán tử Gather Streams sử dụng một số luồng đầu vào và tạo ra một luồng bản ghi đầu ra duy nhất bằng cách kết hợp các luồng đầu vào. Nội dung và định dạng bản ghi không thay đổi. Nếu toán tử này bảo toàn thứ tự thì tất cả các luồng đầu vào phải được sắp xếp theo thứ tự
2. **Repartition Streams :** Toán tử **Repartition Streams** chỉ được sử dụng trong truy vấn

song song.

Toán tử **Repartition Streams** sử dụng nhiều luồng và tạo ra nhiều luồng bản ghi. Nội dung và định dạng bản ghi không thay đổi. Nếu query optimizer sử dụng bộ lọc bitmap thì số lượng dòng trong luồng đầu ra sẽ giảm. Mỗi bản ghi từ 1 luồng đầu vào sẽ được đặt vào một luồng đầu ra. Nếu toán tử này bảo toàn thứ tự thì tất cả các luồng đầu vào phải được sắp xếp thứ tự và hợp nhất thành một số luồng đầu ra có thứ tự

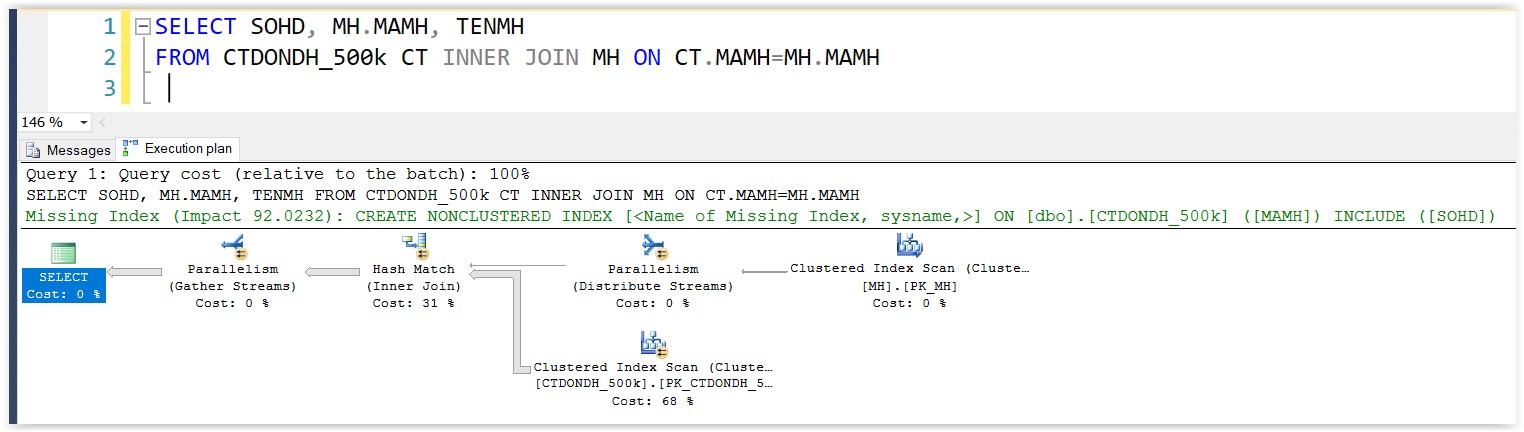
### Các hình thức kết nối:

**a/ Nested Loops Joins:** Nếu một bảng tham gia kết nối nhỏ( ít hơn 10 hàng), bảng thứ 2 khá lớn, và đã được lập chỉ mục trên field kết nối, thì nested loops join là phép toán kết nối nhanh nhất vì nó yêu cầu chi phí I/O ít nhất và ít so sánh nhất.

**The nested loops join** sử dụng một table làm **outer input table** (trong execution plan được hiển thị là table ở trên) và một table làm **inner input table** (trong execution plan được hiển thị là table ở dưới). Vòng lặp bên ngoài quét từng hàng của outer input table. Vòng lặp bên trong, quét từng hàng của inner input table để tìm kiếm các hàng phù hợp trong outer input table.

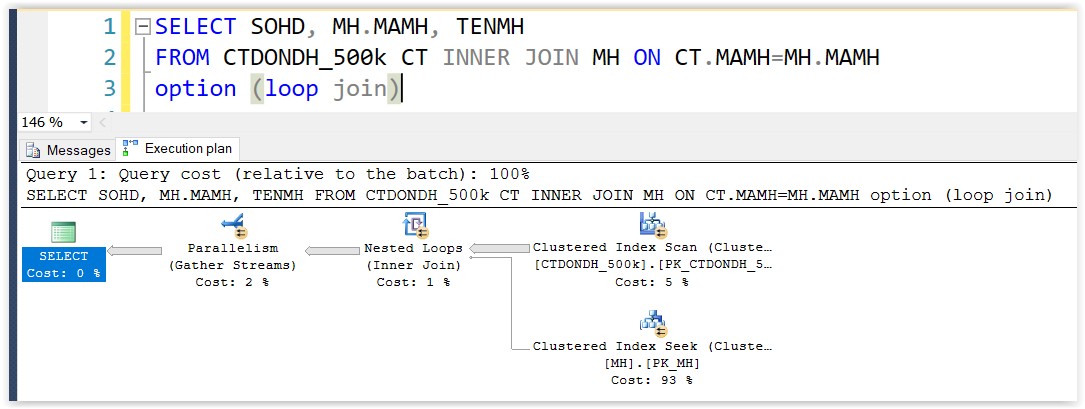
The **nested loops join *đặc biệt hiệu quả*** nếu ***outer input nhỏ và inner input lớn và được lập chỉ mục trước***. Trong nhiều giao tác nhỏ, chẳng hạn như những giao tác chỉ ảnh hưởng đến một tập hợp nhỏ các hàng, index nested loops joins sẽ vượt trội hơn cả các merge joins and hash joins. Tuy nhiên, trong các truy vấn lớn, nested loops joins thường không phải là lựa chọn tối ưu.

* Nếu table CTDONDH\_500K chưa tạo chỉ mục trên MAMH, và trình Query Optimize tự đưa ra chiến lược thực thi, và đề nghị nên tinh chỉnh để cải thiện tốc độ truy vấn



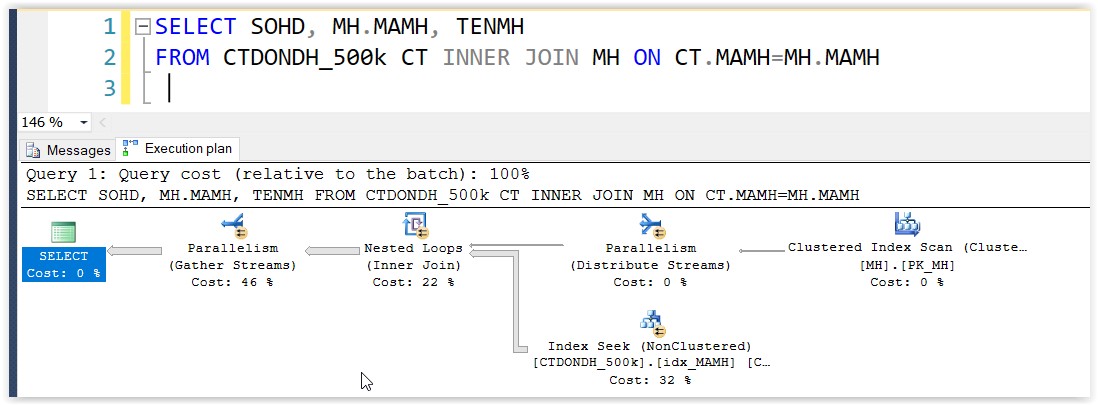
Time: 16s, 17s, 16s

* Nếu table CTDONDH\_500K chưa tạo chỉ mục trên MAMH, và ta chỉ định hình thức kết nối là loop join:



Time: 19s, 20s, 19s

* Nếu table CTDONDH\_500K đã tạo chỉ mục trên MAMH:



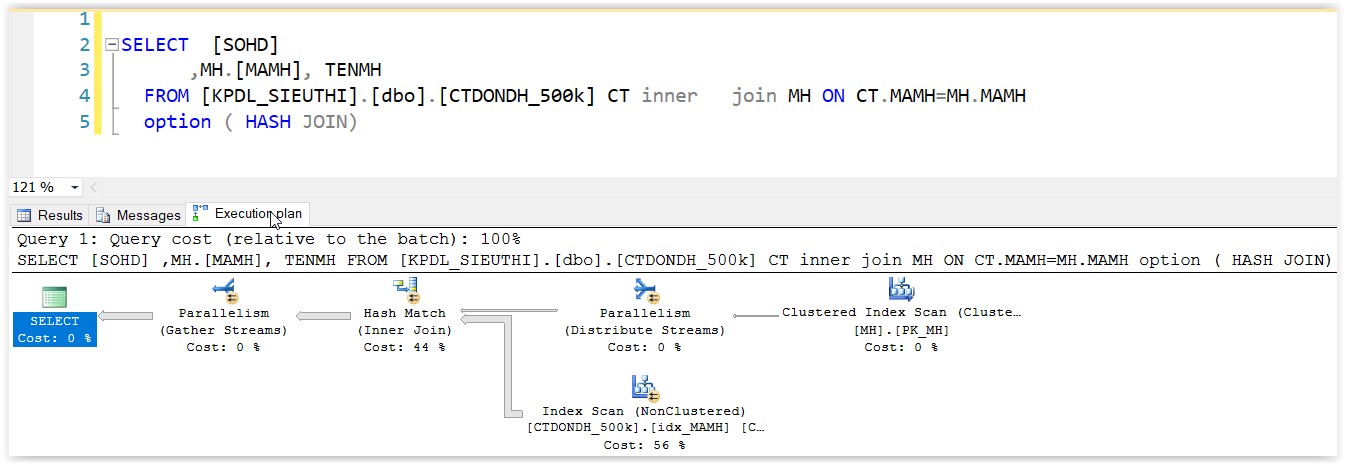
Time: 17s, 15s, 21s

**b/ Hash Joins:** Hash joins xử lý hiệu quả với các table lớn, chưa được sắp xếp, không được lập chỉ mục. Hash joins hữu ích cho các kết quả trung gian trong các truy vấn phức tạp vì:

* + Các kết quả trung gian không được lập chỉ mục và thường không được sắp xếp phù hợp cho các thao tác tiếp theo trong kế hoạch truy vấn.
  + Trình tối ưu hóa truy vấn chỉ ước tính dựa trên kích thước kết quả trung gian. Bởi vì các ước tính có thể không chính xác đối với các truy vấn phức tạp, các thuật toán xử lý kết quả trung gian phải hiệu quả nếu kết quả trung gian hóa ra lớn hơn nhiều so với dự đoán.

Hash join có hai table input: the build input and probe input. The query optimizer sẽ chỉ định table có kích thước nhỏ hơn làm build input.

Hash joins được sử dụng cho nhiều loại phép kết nối: inner join; left, right, and full outer join;

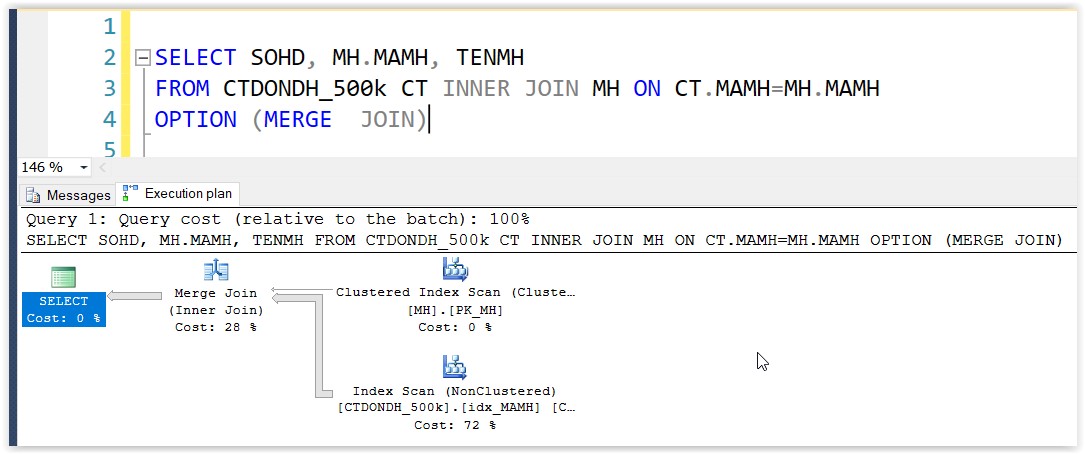


**Time : 19s, 17S, 15S**

**c/ Merge Joins:** Phép merge join yêu cầu cả hai table phải được sắp xếp trên filed quan hệ

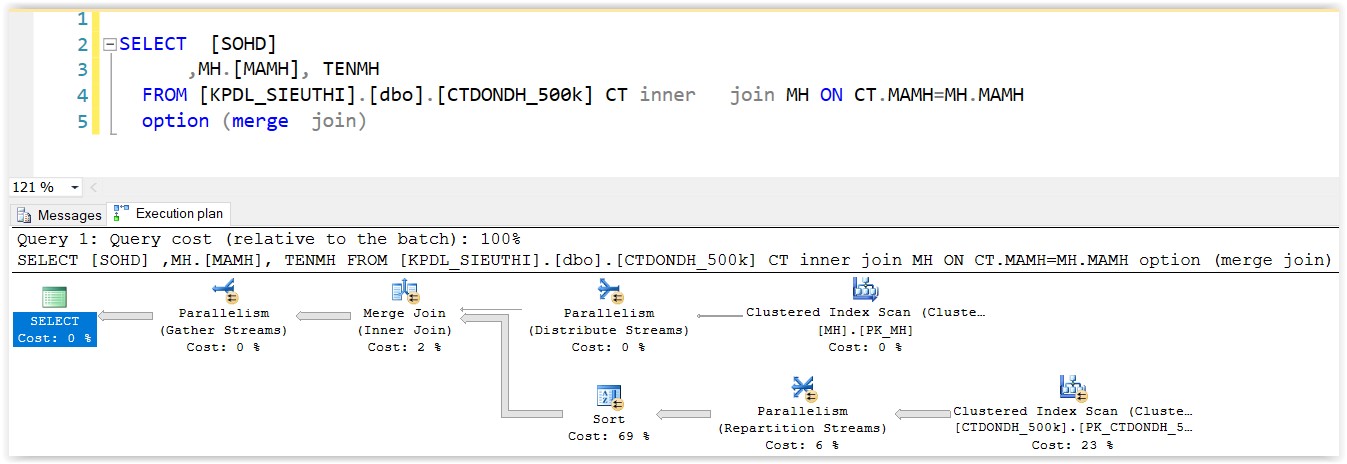
Nếu hai table trong kết nối có số lượng các dòng vừa phải nhưng đã được sắp xếp trên cột nối của chúng, thì merge join là thao tác nối nhanh nhất. Nếu cả hai table trong kết nối đều lớn và có kích thước tương tự nhau, thì merge join với sắp xếp trước và hash join sẽ mang lại hiệu suất tương tự. Tuy nhiên, các thao tác hash join thường nhanh hơn nhiều nếu hai table kết nối có kích thước khác nhau đáng kể

* Nếu table CTDONDH\_500K đã tạo chỉ mục trên MAMH:



Time : 16s, 15s, 15s

* Nếu table CTDONDH\_500K chưa tạo chỉ mục trên MAMH:



Time : 18s, 16s, 16s

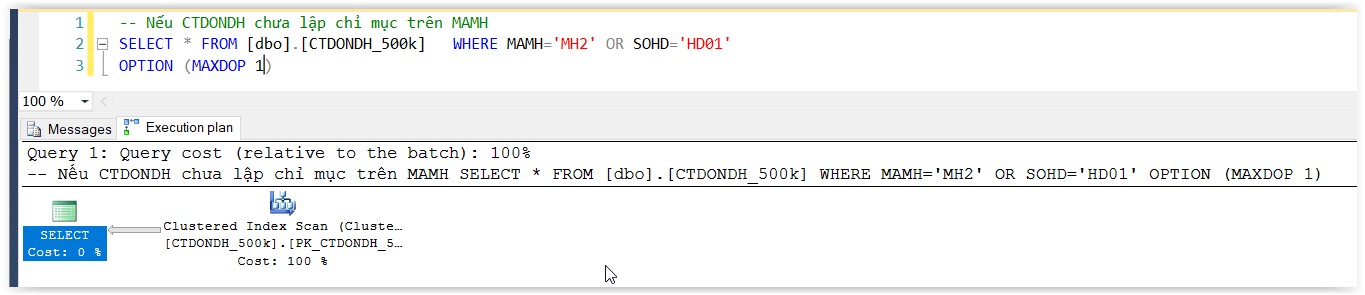
### Hiệu chỉnh cách viết truy vấn:

Một số truy vấn tiêu tốn nhiều tài nguyên hơn những truy vấn khác. Ví dụ: các truy vấn trả về tập hợp kết quả lớn hoặc những truy vấn chứa mệnh đề WHERE trả về các giá trị không duy nhất luôn tiêu tốn nhiều tài nguyên. Query optimizer không đủ thông minh để có thể loại bỏ chi phí tài nguyên của các cấu trúc này. SQL Server sử dụng kế hoạch truy cập tối ưu nhưng việc tối ưu hóa truy vấn vẫn bị giới hạn.

Tuy nhiên, để cải thiện hiệu suất truy vấn, ta có thể:

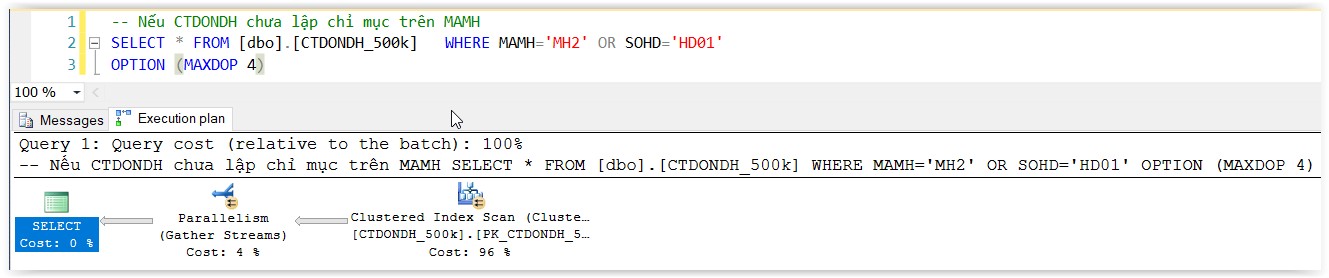
* + 1. Thêm nhiều bộ nhớ hơn. Giải pháp này có thể đặc biệt hữu ích nếu máy chủ chạy nhiều truy vấn phức tạp và một số truy vấn thực thi chậm.
    2. Sử dụng nhiều bộ xử lý. Nhiều bộ xử lý cho phép Database Engine sử dụng các truy vấn song song.

Mức độ song song được xác định bởi số lượng CPU tối đa đang được sử dụng; Mức độ giá trị song song được thiết lập ở Server và có thể được sửa đổi bằng cách sử dụng thủ tục hệ thống sp\_configure. Ta cǜng có thể yêu cầu xử lý song song ngay trong lệnh truy vấn với option MAXDOP (theo ví dụ dưới)



MAXDOP=1 : xử lý tuần tự

Time : 2s



MAXDOP=4 : xử lý song song (tối đa 4 luồng)

Time : 1s

* + 1. Nếu ứng dụng sử dụng vòng lặp, hãy cân nhắc việc đặt lệnh gọi truy vấn bên trong vòng lặp. Thông thường, một ứng dụng có một vòng lặp chứa một truy vấn có tham số, truy vấn được thực thi nhiều lần và như vậy yêu cầu kết nối mạng liên tục giữa máy tính chạy ứng dụng và SQL Server. Thay vào đó, hãy tạo một truy vấn dùng bảng tạm. Ta chỉ cần gọi query 1 lần, và trình query optimizer có thể tối ưu hóa query đó.
    2. Dùng phép chọn/chiếu trước, phép kết sau ;
    3. Khử phép kết (nếu được) : nếu kết quả in ra có 1 cột mà dữ liệu không thay đổi thì bảng cung cấp dữ liệu cho cột có thể truy vấn riêng ;
    4. Nếu 1 điều kiện xuất hiện nhiều lần trong WHERE thì dùng các phép biến đổi tương đương để cho điều kiện đó xuất hiện 1 lần

P1 ^ ( P2 vP3) œ ( P1 ^ P2) v (P1 ^ P3)

P1 ^ (P1 v P2 v P3) œ P1

P1 v (P1 ^ P2 ^ P3) œ P1

* + 1. Tạo INDEX trước cho các field được dùng nhiều và thường xuyên trong điều kiện WHERE

In bảng điểm môn @mamh , @malop , lan =@lan

-- TENLOP TENMH MASV HOTEN DIEM

CREATE PROC SP\_BDMH

@malop NVARCHAR(10), @mamh NVARCHAR(10), @lan INT AS

BEGIN

SELECT TENLOP, TENMH , SV.MASV , HOTEN=HO+' '+TEN, DIEM FROM LOP , SINHVIEN SV, DIEM , MONHOC MH

WHERE SV.MALOP = @malop AND DIEM.MAMH = @mamh AND LAN = @lan AND LOP.MALOP =SV.MALOP AND SV.MASV = DIEM.MASV

AND DIEM.MAMH = MH.MAMH

END

CREATE PROC SP\_BDMH\_XULYTOIUU

@malop NVARCHAR(10), @mamh NVARCHAR(10), @lan INT AS

BEGIN

DECLARE @TENLOP NVARCHAR(100), @TENMH NVARCHAR(100) SELECT @TENLOP= TENLOP FROM LOP WHERE MALOP = @malop SELECT @TENMH = TENMH FROM MONHOC WHERE MAMH = @mamh

SELECT TENLOP=@TENLOP , TENMH=@TENMH , SV.MASV , HOTEN=HO+' '+TEN, DIEM

FROM (SELECT MASV, HO,TEN FROM SINHVIEN WITH (INDEX=IX\_MALOP) WHERE MALOP = @malop) **SV**,

(SELECT MASV , DIEM FROM DIEM WHERE MAMH = @mamh AND

LAN = @lan) **DIEM**

WHERE SV.MASV = DIEM.MASV

END

* + 1. Hạn chế Distinct, Order By: Hạn chế sử dụng 2 từ khóa này. Hai phép toán này chiếm phần lớn thời gian truy vấn vì phải thực hiện lọc bản ghi trùng và sắp xếp các bản ghi.
    2. Tránh sử dụng function bên trong toán tử so sánh

Ěể tối ưu câu truy vấn SQL thì Function là một cách thuận tiện để xử lý các nhiệm vụ phức tạp mà bạn có thể sử dụng nó cả ở mệnh đề FROM hay WHERE. Tuy nhiên, ứng

dụng chúng trong mệnh đề WHERE có thể gây ra những vấn đề không tốt về performance. Hãy xem qua ví dụ dưới đây

SELECT nickname FROM users

WHERE DATEDIFF(MONTH, appointment\_date, '2015-04-28') < 0;

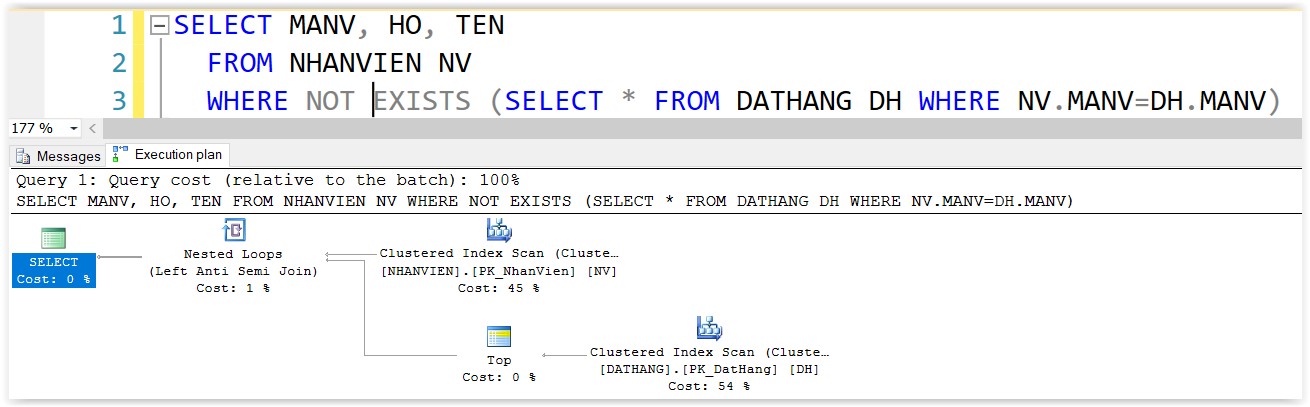
Thậm chí nếu ta đã đánh index cho cột appointment\_date trong bảng users, câu truy vấn vẫn phải scan toàn bộ bảng. Ěó là bởi vì bạn sử dụng hàm DATEDIFF cho cột mà bạn đang xét (appointment\_date).

Giá trị tính toán được của hàm này sẽ được tính lúc runtime, như vậy nó sẽ phải kiểm tra từng record trong bảng, lấy ra giá trị output và đem so sánh với 0. Ěể cải thiện performace, chúng ta có thể đổi câu truy vấn như sau

SELECT nickname FROM users WHERE appointment\_date > '2015-04- 30';

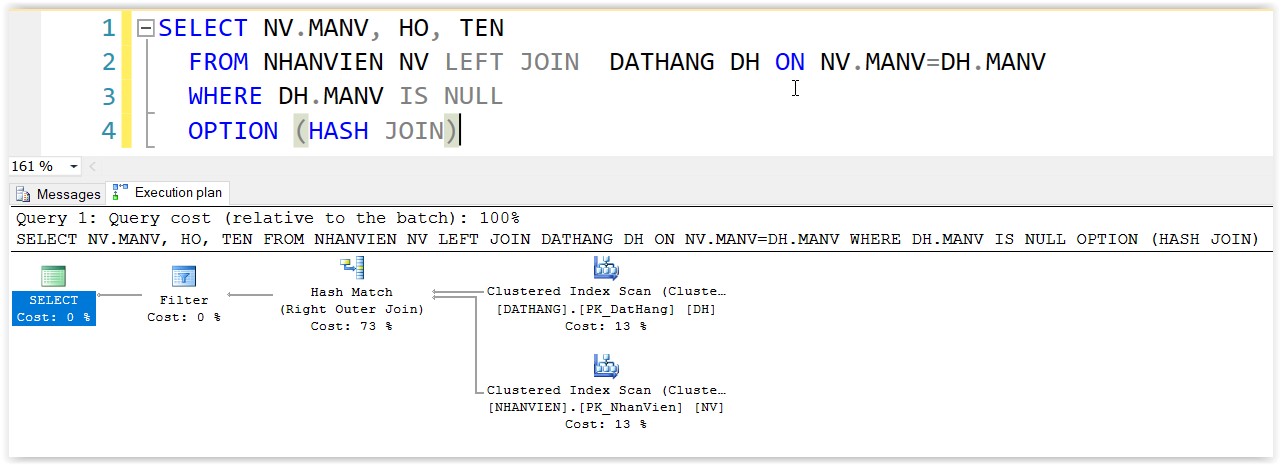
* + 1. Cân nhắc việc loại bỏ các câu truy vấn con tương quan

Câu truy vấn con tương quan là những câu truy vấn con phụ thuộc vào câu truy vấn ngoài. Nó sử dụng dữ liệu lấy được từ câu truy vấn ngoài trong mệnh đề WHERE. Ví dụ bạn muốn lấy danh sách tất cả những user mà đã ủng hộ. Bạn có thể sử dụng câu truy vấn



Trong trường hợp này, câu truy vấn con sẽ chạy 1 lần với mỗi record nhân viên của câu truy vấn chính. Cụ thể là mỗi lần xét tới 1 record của câu truy vấn chính, câu truy vấn con lại được chạy với điều kiện WHERE trong đó là giá trị MANV của record đang được xét của câu truy vấn NHANVIEN. Do đó, câu truy vấn chưa *hiệu quả*

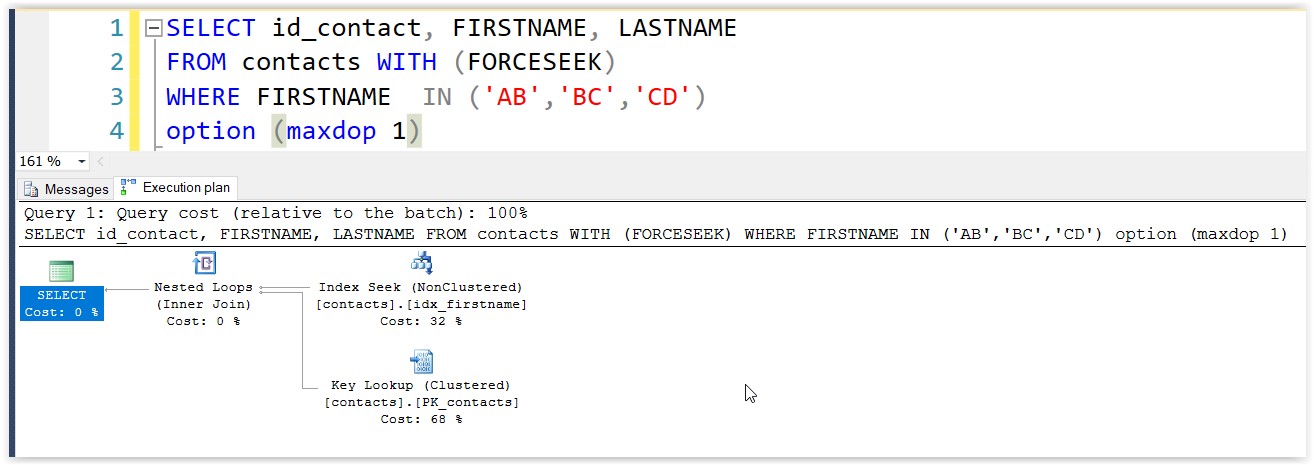
Thay vào đó, ta có thể sử dụng

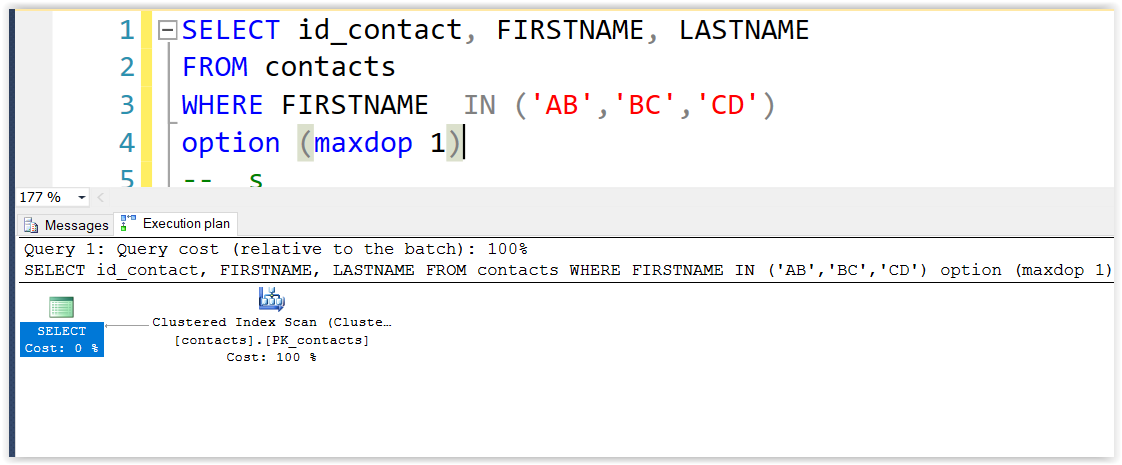


* + 1. Sử dụng FORCESEEK trong các truy vấn dùng toán tử LIKE hoặc IN

Khi truy vấn sử dụng toán tử IN hoặc LIKE để tìm kiếm thì Query optimizer có thể thực hiện table scan hoặc index scan thay vì dùng index seek.

Ví dụ sau đây minh họa cách buộc query optimizer thực hiện phép toán index seek khi dùng toán tử IN hoặc LIKE để tìm kiếm.





### PHẦN 3. DEMO

Ěể demo các kỹ thuật làm tĕng tốc độ xử lý, ta sẽ tạo 1 table contacts với cấu trúc sau:

create table contacts (id\_contact int not null primary key, firstname varchar(255), lastname varchar(255))

Sau đó, tạo giả 10 triệu records theo như Stored Procedure sau:

CREATE PROC sp\_Tao\_du\_lieu\_contacts AS

Begin

declare @i int = 0, @randomString1 varchar(255), @randomString2 varchar(255)

while @i < 10000000 begin

set @randomString1 = CONVERT(varchar(255), NEWID()) set @randomString2 = CONVERT(varchar(255), NEWID())

insert into contacts(id\_contact, firstname, lastname) values (@i, @randomString1, @randomString2)

End

set @i = @i + 1 end

Thời gian tạo dữ liệu mất : 22 phút 47 giây

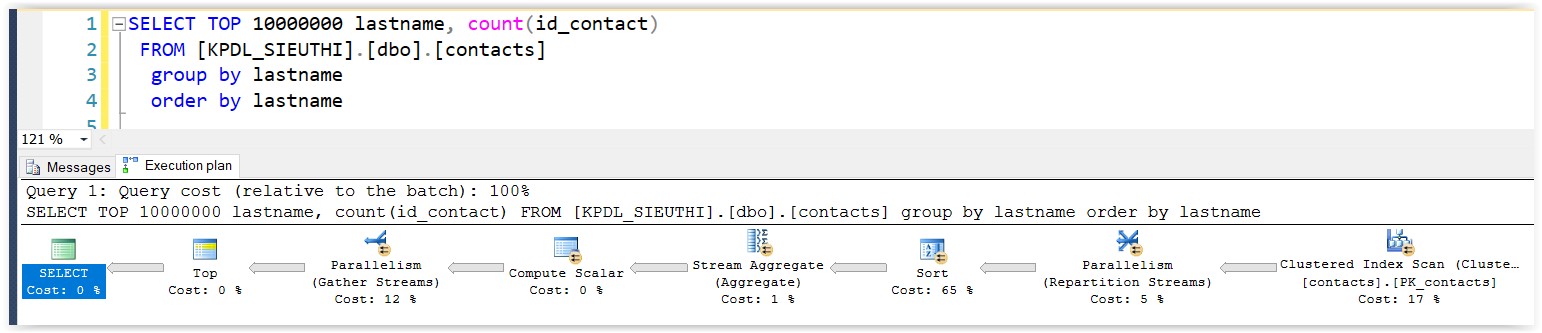
Khi thực hiện lệnh : SELECT \* FROM [dbo].[contacts] Thời gian liệt kê: 2 phút 55 giây.

-- flush query cache on server

dbcc freeproccache with no\_infomsgs;

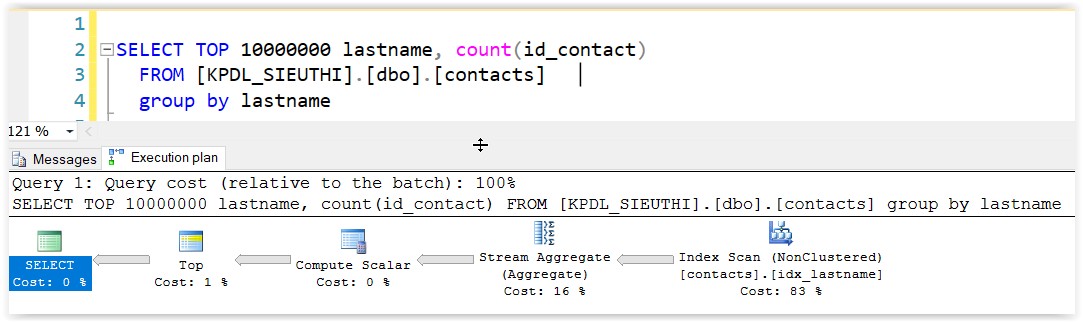
set statistics time on set statistics io on

a/ Sắp thứ tự dùng order by (chưa tạo index trên lastname):



Thời gian: 1:30s; 1:36s; 1:51s

b/ Sắp thứ tự dùng Index (đã tạo index trên lastname):



Thời gian: 1:24s; 1:18s; 1:31s

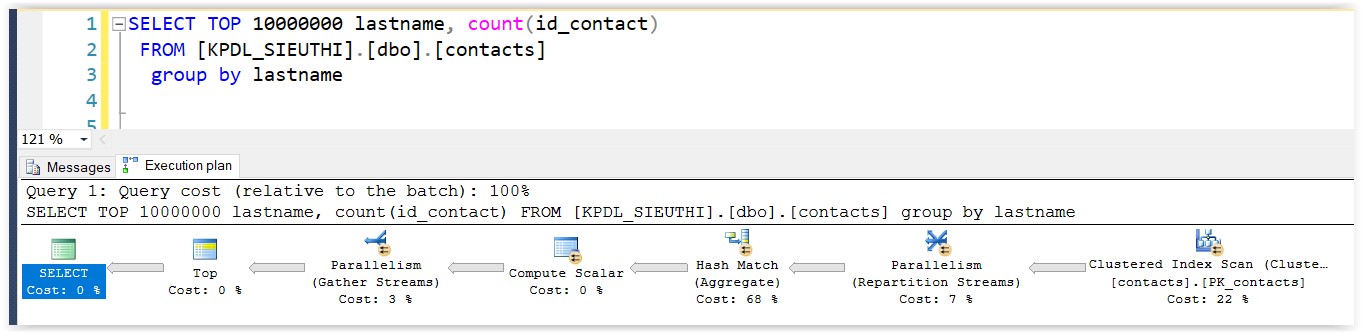
Không sắp thứ tự và không có index dbcc freeproccache

set statistics time on set statistics io on

SELECT TOP 10000000 lastname, count(id\_contact) FROM [KPDL\_SIEUTHI].[dbo].[contacts]

group by lastname

Thời gian : 2 phút 49 giây



# KẾT LUẬN

Ěề tài đã hoàn thiện theo đúng mục tiêu và nội dung như đã đĕng ký .

# TÀI LIỆU THAM KHẢO

* + - 1. [https://ww](http://www.tatvasoft.com/blog/optimize-sql-query/)w.tatva[soft.com/blog/optimize](http://www.tatvasoft.com/blog/optimize-sql-query/)-[sql-query/](http://www.tatvasoft.com/blog/optimize-sql-query/)
      2. https://learn.microsoft.com/en-us/previous-versions/sql/2014/relational- databases/performance/display-the-estimated-execution-plan?view=sql-server-2014
      3. https://learn.microsoft.com/en-us/previous-versions/sql/2014/relational- databases/performance/performance-monitoring-and-tuning-tools?view=sql-server- 2014