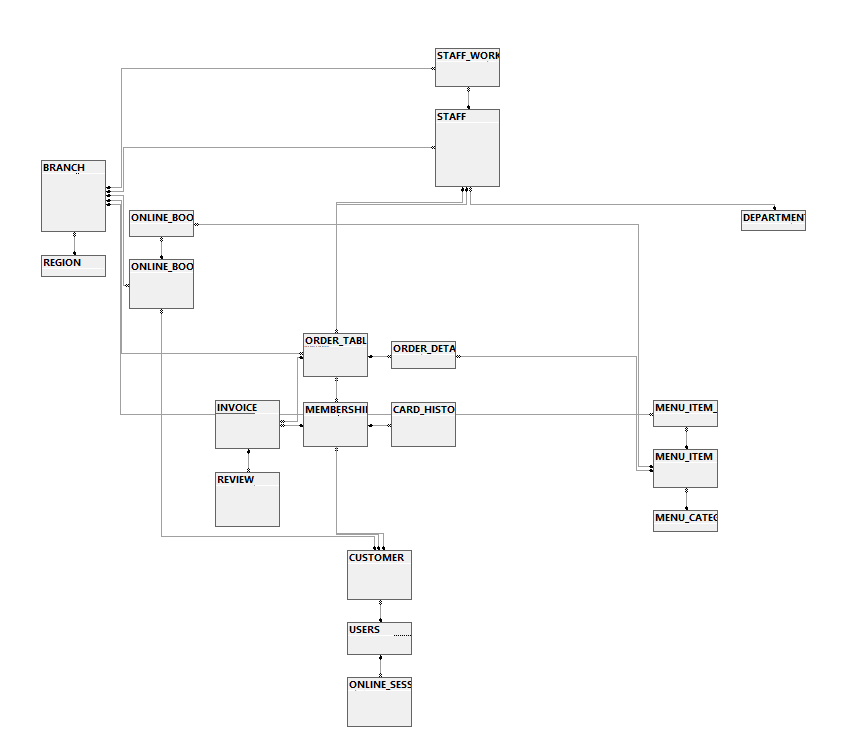
1. **Tạo cơ sở dữ liệu:**

Hình ảnh cơ sơ dữ liệu SuShiX:



Script tạo cơ sở dữ liệu SuShiX:

| **CREATE** **DATABASE** SuShiX; **USE** SuShiX;  **CREATE** **TABLE** REGION (  RegionID INT PRIMARY **KEY**,  RegionName VARCHAR(255) );  **CREATE** **TABLE** BRANCH (  BranchID VARCHAR(255) PRIMARY **KEY**,  RegionID VARCHAR(255),  BranchName VARCHAR(255),  Address VARCHAR(255),  BranchPhoneNumber VARCHAR(20),  OpeningTime **TIME**,  ClosingTime **TIME**,  HasCarParking BIT,  HasBikeParking BIT,  DeliveryService BIT,  ManagerID VARCHAR(255),  FOREIGN **KEY** (RegionID) **REFERENCES** REGION(RegionID) );  **CREATE** **TABLE** DEPARTMENT (  DepartmentID VARCHAR(255) PRIMARY **KEY**,  DepartmentName VARCHAR(255) );  **CREATE** **TABLE** STAFF (  StaffID VARCHAR(255) PRIMARY **KEY**,  DepartmentID VARCHAR(255),  BranchID VARCHAR(255),  FullName VARCHAR(255),  Gender VARCHAR(10),  BirthDate DATE,  StartDate DATE,  EndDate DATE,  Salary DECIMAL(10, 2),  Address VARCHAR(255),  PhoneNumber VARCHAR(20),  UserID VARCHAR(255),  FOREIGN **KEY** (DepartmentID) **REFERENCES** DEPARTMENT(DepartmentID),  FOREIGN **KEY** (BranchID) **REFERENCES** BRANCH(BranchID) );  **CREATE** **TABLE** STAFF\_WORK\_HISTORY (  HistoryID INT PRIMARY **KEY**,  StaffID VARCHAR(255),  BranchID VARCHAR(255),  StartDate DATE,  EndDate DATE,  FOREIGN **KEY** (StaffID) **REFERENCES** STAFF(StaffID),  FOREIGN **KEY** (BranchID) **REFERENCES** BRANCH(BranchID) );  **CREATE** **TABLE** **USERS** (  UserID VARCHAR(255) PRIMARY **KEY**,  Username VARCHAR(255),  **Password** VARCHAR(255),  **Role** VARCHAR(50) );  **CREATE** **TABLE** CUSTOMER (  CustomerID VARCHAR(255) PRIMARY **KEY**,  FullName VARCHAR(255),  PhoneNumber VARCHAR(255),  Email VARCHAR(255),  IDNumber VARCHAR(20),  Gender VARCHAR(10),  UserID VARCHAR(255),  FOREIGN **KEY** (UserID) **REFERENCES** **USERS**(UserID) );  **CREATE** **TABLE** MEMBERSHIP\_CARD (  CardID VARCHAR(255) PRIMARY **KEY**,  CustomerID VARCHAR(255),  StaffID VARCHAR(255),  CardType VARCHAR(50),  RegistrationDate DATE,  AccumulatedPoints FLOAT,  FOREIGN **KEY** (CustomerID) **REFERENCES** CUSTOMER(CustomerID),  FOREIGN **KEY** (StaffID) **REFERENCES** STAFF(StaffID) );  **CREATE** **TABLE** CARD\_HISTORY (  HistoryID INT PRIMARY **KEY**,  CardID INT,  StatusChangeDate DATE,  PreviousCardType VARCHAR(50),  NewCardType VARCHAR(50),  SpendingDuringPeriod FLOAT,  FOREIGN **KEY** (CardID) **REFERENCES** MEMBERSHIP\_CARD(CardID) );  **CREATE** **TABLE** ONLINE\_SESSION (  SessionID VARCHAR(255) PRIMARY **KEY**,  UserID VARCHAR(255),  SessionStart DATETIME,  SessionDuration INT,  DeviceType VARCHAR(50),  IPAddress VARCHAR(50),  InternalSource VARCHAR(50),  FOREIGN **KEY** (UserID) **REFERENCES** **USERS**(UserID) );  **CREATE** **TABLE** MENU\_CATEGORY (  CategoryID VARCHAR(255) PRIMARY **KEY**,  CategoryName VARCHAR(255) );  **CREATE** **TABLE** MENU\_ITEM (  ItemID VARCHAR(255) PRIMARY **KEY**,  CategoryID VARCHAR(255),  ItemName VARCHAR(255),  CurrentPrice FLOAT,  DeliveryAvailable BIT,  FOREIGN **KEY** (CategoryID) **REFERENCES** MENU\_CATEGORY(CategoryID) );  **CREATE** **TABLE** MENU\_ITEM\_AVAILABILITY (  BranchID VARCHAR(255),  ItemID VARCHAR(255),  IsAvailable BIT,  PRIMARY **KEY** (BranchID, ItemID),  FOREIGN **KEY** (BranchID) **REFERENCES** BRANCH(BranchID),  FOREIGN **KEY** (ItemID) **REFERENCES** MENU\_ITEM(ItemID) );  **CREATE** **TABLE** ONLINE\_BOOKING (  BookingID INT PRIMARY **KEY**,  CustomerID VARCHAR(255),  BranchID VARCHAR(255),  GuestCount INT,  BookingDate DATE,  ArrivalTime **TIME**,  Notes VARCHAR(255),  FOREIGN **KEY** (CustomerID) **REFERENCES** CUSTOMER(CustomerID),  FOREIGN **KEY** (BranchID) **REFERENCES** BRANCH(BranchID) );  **CREATE** **TABLE** ONLINE\_BOOKING\_ORDER (  BookingID INT,  ItemID VARCHAR(255),  Quantity INT,  PRIMARY **KEY** (BookingID, ItemID),  FOREIGN **KEY** (BookingID) **REFERENCES** ONLINE\_BOOKING(BookingID),  FOREIGN **KEY** (ItemID) **REFERENCES** MENU\_ITEM(ItemID) );  **CREATE** **TABLE** ORDER\_TABLE (  OrderID VARCHAR(255) PRIMARY **KEY**,  OrderDate DATE,  StaffID VARCHAR(255),  TableNumber INT,  BranchID VARCHAR(255),  CustomerID VARCHAR(255),  FOREIGN **KEY** (StaffID) **REFERENCES** STAFF(StaffID),  FOREIGN **KEY** (BranchID) **REFERENCES** BRANCH(BranchID),  FOREIGN **KEY** (CustomerID) **REFERENCES** CUSTOMER(CustomerID) );  **CREATE** **TABLE** ORDER\_DETAIL (  OrderID VARCHAR(255),  ItemID VARCHAR(255),  Quantity INT,  PRIMARY **KEY** (OrderID, ItemID),  FOREIGN **KEY** (OrderID) **REFERENCES** ORDER\_TABLE(OrderID),  FOREIGN **KEY** (ItemID) **REFERENCES** MENU\_ITEM(ItemID) );  **CREATE** **TABLE** INVOICE (  InvoiceID VARCHAR(255) PRIMARY **KEY**,  OrderID VARCHAR(255),  CardID VARCHAR(255),  TotalAmount FLOAT,  Discount FLOAT,  FinalAmount FLOAT,  CreatedAt DATE,  FOREIGN **KEY** (OrderID) **REFERENCES** ORDER\_TABLE(OrderID),  FOREIGN **KEY** (CardID) **REFERENCES** MEMBERSHIP\_CARD(CardID) );  **CREATE** **TABLE** REVIEW (  ReviewID VARCHAR(255) PRIMARY **KEY**,  InvoiceID VARCHAR(255),  ServiceScore INT,  LocationScore INT,  FoodQualityScore INT,  PriceScore INT,  AmbianceScore INT,  **Comment** VARCHAR(255),  FOREIGN **KEY** (InvoiceID) **REFERENCES** INVOICE(InvoiceID) ); |
| --- |

1. **Dữ liệu cho mỗi table**

Ở repo [bpthien14/SuShiX\_restaurant: Đồ án thực hành csdl nâng cao](https://github.com/bpthien14/SuShiX_restaurant) này, trong folder SAMPLE DATA, có các file python phát sinh 10000 dòng dữ liệu cho mỗi bảng tương ứng với tên file python và có các file csv tương ứng với tên table chứa các dòng dữ liệu đó.

1. **Kịch bản cho SuShiX**

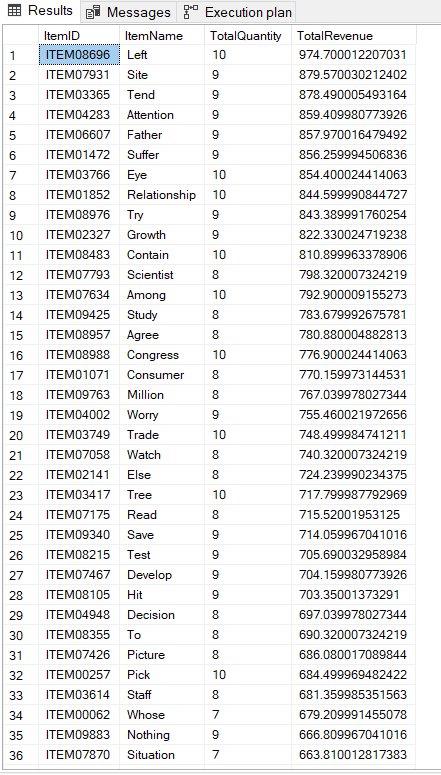
**Kịch bản 1 : Thống kê Doanh Thu Theo Món Ăn**

Mô tả

Quản lý nhà hàng muốn thống kê doanh thu của từng món ăn trong một khoảng thời gian cụ thể để xác định món ăn nào bán chạy nhất và món nào bán chậm nhất. Điều này giúp tối ưu hóa thực đơn và chiến lược kinh doanh.

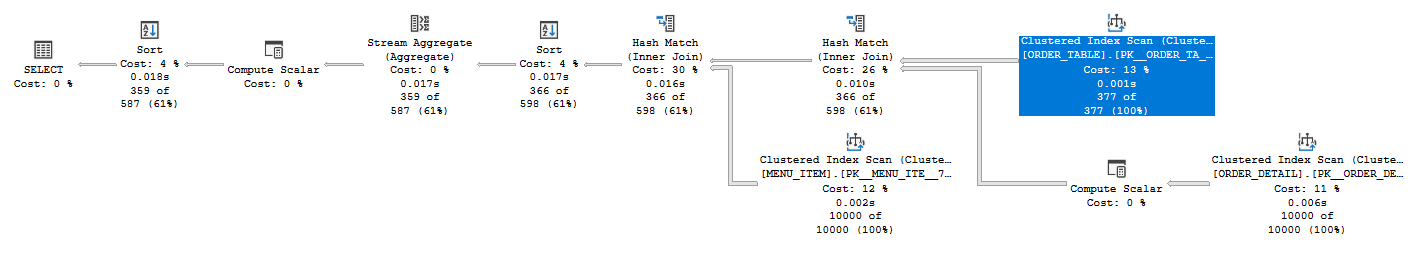
Script:

| **CREATE** **PROCEDURE** GetItemRevenue  @StartDate DATE,  @EndDate DATE **AS** **BEGIN**  **SELECT**   mi.ItemID,   mi.ItemName,   **SUM**(od.Quantity) **AS** TotalQuantity,   **SUM**(od.Quantity \* mi.CurrentPrice) **AS** TotalRevenue  **FROM**   ORDER\_DETAIL od  **INNER** **JOIN**   MENU\_ITEM mi **ON** od.ItemID = mi.ItemID  **INNER** **JOIN**   ORDER\_TABLE ot **ON** od.OrderID = ot.OrderID  **WHERE**   ot.OrderDate **BETWEEN** @StartDate **AND** @EndDate  **GROUP** **BY**   mi.ItemID, mi.ItemName  **ORDER** **BY**   TotalRevenue **DESC**; **END** |
| --- |

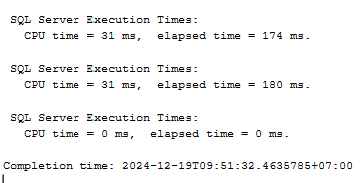
Kết quả: (trong hình là kết quả của 36 trên 359 dòng)

**Phân tích kế hoạch thực thi và tốc độ câu truy vấn trước khi tối ưu:**

Kế hoạch thực thi:



Tốc độ câu truy vấn:



Kết luận:

* Execution plan có sử dụng Hash Match cho các phép nối INNER JOIN. Điều này thường xảy ra khi không có chỉ mục phù hợp cho các cột được nối.
* Có nhiều Clustered Index Scan, điều này có thể chỉ ra rằng SQL Server đang quét toàn bộ bảng để tìm dữ liệu. Điều này thường không hiệu quả cho các bảng lớn.

Cách tối ưu:

* Tạo một view để chứa kết quả của kịch bản 1 và một chỉ mục không phân cụm (nonclustered index) trên cột OrderDate của bảng ORDER\_TABLE.

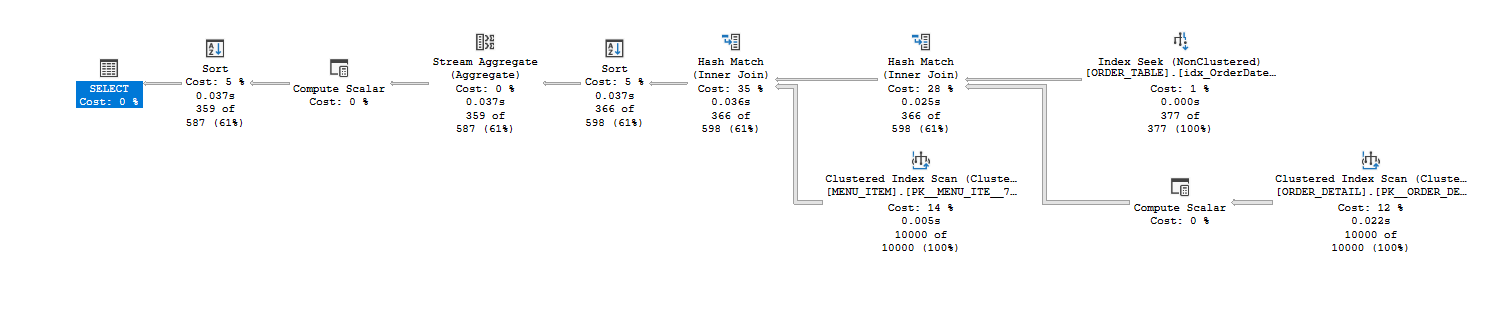
**Tối ưu kịch bản 1**

Scrip tạo một chỉ mục không phân cụm (nonclustered index) trên cột OrderDate của bảng ORDER\_TABLE:

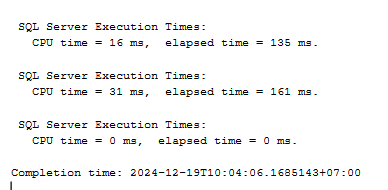
| **CREATE** NONCLUSTERED **INDEX** idx\_OrderDate **ON** ORDER\_TABLE (OrderDate); |
| --- |

Phân tích kế hoạch thực thi và tốc độ câu truy vấn sau khi tối ưu:

Kế hoạch thực thi:



Tốc độ câu truy vấn:

  
Kết luận:

* Trong hình mới, có một Index Seek trên ORDER\_TABLE với chỉ mục idx\_OrderDate. Điều này cho thấy SQL Server đang sử dụng chỉ mục không phân cụm mới để tìm kiếm dữ liệu nhanh hơn, thay vì quét toàn bộ bảng.
* Chi phí cho các phép nối Hash Match đã giảm, cho thấy hiệu suất đã được cải thiện.
* Vẫn còn Clustered Index Scan trên ORDER\_DETAIL, nhưng việc sử dụng Index Seek đã giúp giảm chi phí tổng thể.

**Kịch bản 2: Thống kê Doanh Thu Theo Chi Nhánh**

Mô tả

Quản lý công ty muốn thống kê tổng doanh thu của từng chi nhánh trong một khoảng thời gian cụ thể để đánh giá hiệu suất và đưa ra quyết định chiến lược.

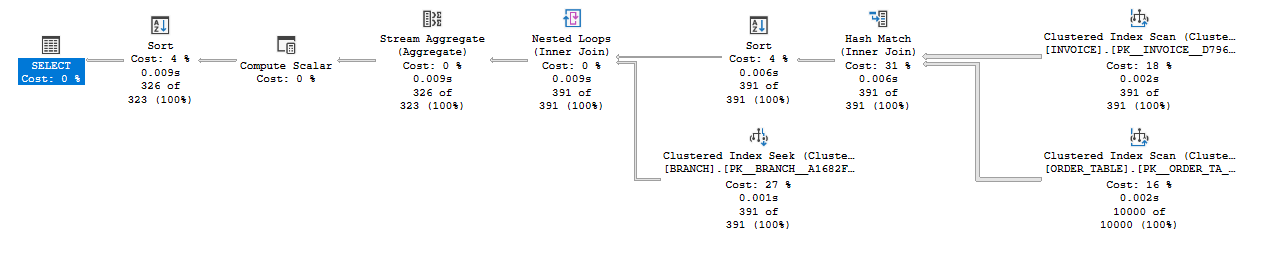
Script:

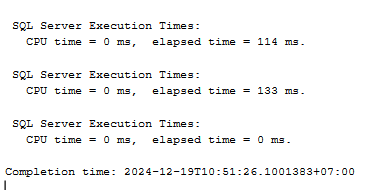
| **CREATE** **PROCEDURE** GetBranchRevenue  @StartDate DATE,  @EndDate DATE **AS** **BEGIN**  **SELECT**   b.BranchID,  b.BranchName,   **SUM**(i.FinalAmount) **AS** TotalRevenue  **FROM**   INVOICE i  **JOIN**   ORDER\_TABLE ot **ON** i.OrderID = ot.OrderID  **JOIN**   BRANCH b **ON** ot.BranchID = b.BranchID  **WHERE**   i.CreatedAt **BETWEEN** @StartDate **AND** @EndDate  **GROUP** **BY**   b.BranchID, b.BranchName  **ORDER** **BY**   TotalRevenue **DESC**; **END** |
| --- |

Kết quả: (36 trên 326 dòng)



**Phân tích kế hoạch thực thi và tốc độ câu truy vấn trước khi tối ưu:**





Kết luận:

* Có Clustered Index Scan trên bảng INVOICE và ORDER\_TABLE. Điều này có thể chỉ ra rằng SQL Server đang quét toàn bộ bảng để tìm dữ liệu. Điều này thường không hiệu quả cho các bảng lớn.
* Sử dụng Hash Match cho phép nối INNER JOIN. Điều này thường xảy ra khi không có chỉ mục phù hợp cho các cột được nối.
* Có Clustered Index Seek trên bảng BRANCH, cho thấy SQL Server đang sử dụng chỉ mục để tìm kiếm dữ liệu nhanh hơn.

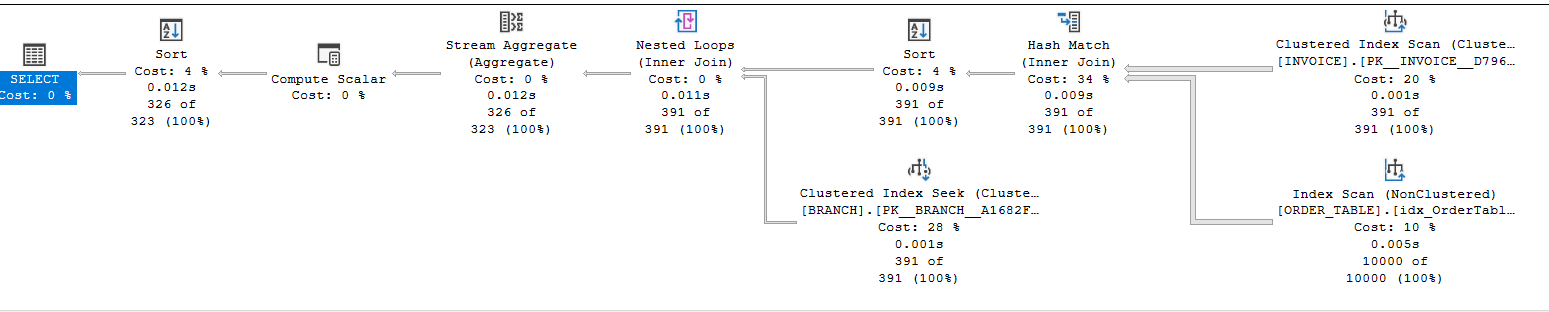
**Tối ưu kịch bản 2:**

Scrip tạo một chỉ mục không phân cụm (nonclustered index) trên cột BranchID của bảng ORDER\_TABLE và INVOICE:

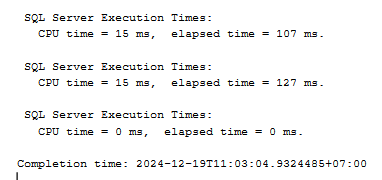
| **CREATE** NONCLUSTERED **INDEX** idx\_OrderTable\_BranchID **ON** ORDER\_TABLE (BranchID); GO  **CREATE** NONCLUSTERED **INDEX** idx\_Invoice\_OrderID **ON** INVOICE (OrderID); GO |
| --- |

**Phân tích kế hoạch thực thi và tốc độ câu truy vấn sau khi tối ưu:**

Kế hoạch thực thi:



Tốc độ câu truy vấn:



Kết luận:

* Chi phí cho các phép nối Hash Match đã giảm, cho thấy hiệu suất đã được cải thiện.

**Kịch bản: Thống kê Đơn Hàng và Doanh Thu Theo Nhân Viên**

Mô tả

Quản lý muốn biết số lượng đơn hàng và tổng doanh thu mà mỗi nhân viên đã xử lý trong một khoảng thời gian cụ thể để đánh giá hiệu suất làm việc.

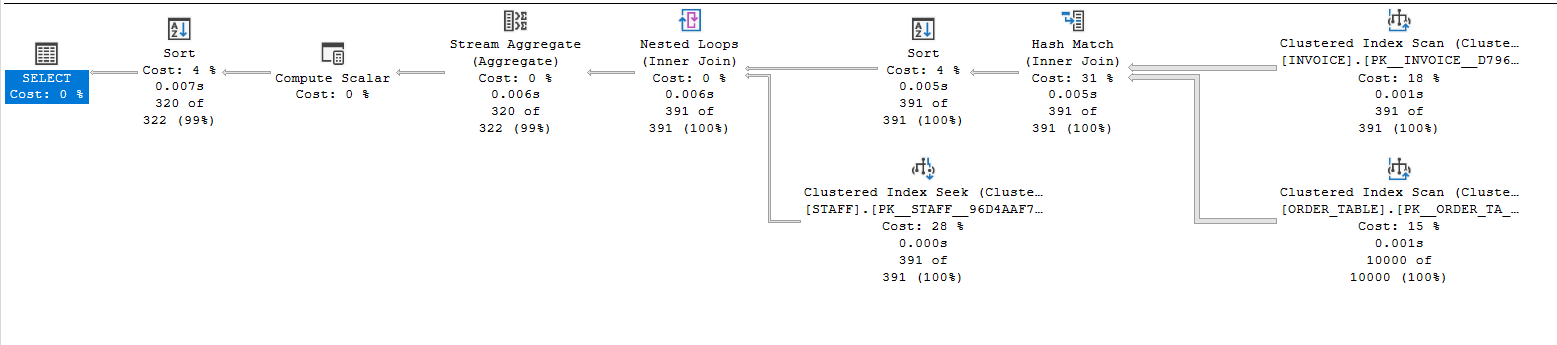
Script

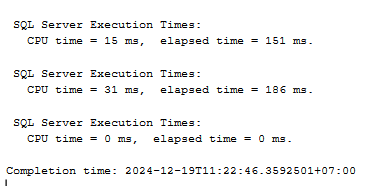
| **CREATE** **PROCEDURE** GetStaffRevenue  @StartDate DATE,  @EndDate DATE **AS** **BEGIN**  **SELECT**   s.StaffID,  s.FullName,  **COUNT**(i.InvoiceID) **AS** TotalOrders,  **SUM**(i.FinalAmount) **AS** TotalRevenue  **FROM**   INVOICE i  **JOIN**   ORDER\_TABLE ot **ON** i.OrderID = ot.OrderID  **JOIN**   STAFF s **ON** ot.StaffID = s.StaffID  **WHERE**   i.CreatedAt **BETWEEN** @StartDate **AND** @EndDate  **GROUP** **BY**   s.StaffID, s.FullName  **ORDER** **BY**   TotalRevenue **DESC**; **END** |
| --- |

Kết quả: (36 trên 320 dòng)



**Phân tích kế hoạch thực thi và tốc độ câu truy vấn trước khi tối ưu:**

****

****

Kết luận:

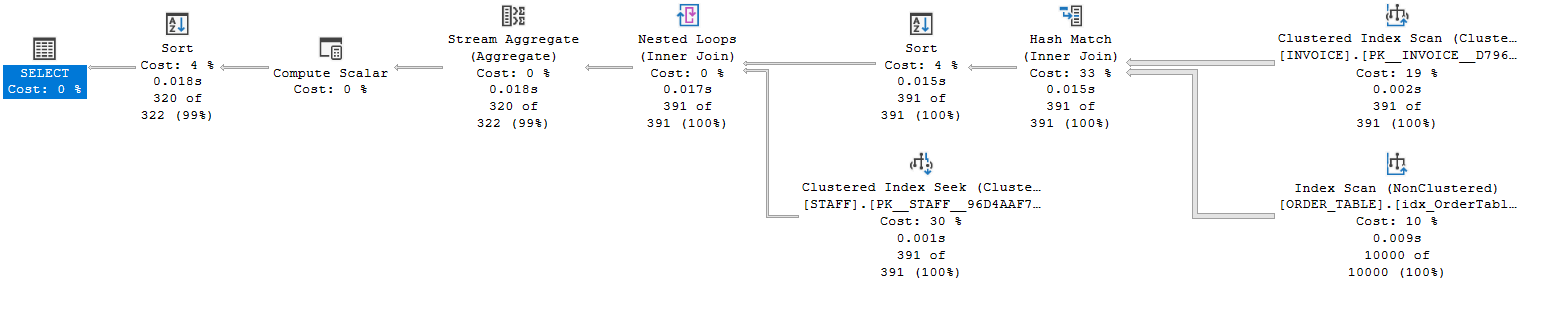
* Có Clustered Index Scan trên bảng INVOICE và ORDER\_TABLE. Điều này cho thấy SQL Server đang quét toàn bộ bảng để tìm dữ liệu, có thể không hiệu quả cho các bảng lớn.
* Sử dụng Hash Match cho phép nối INNER JOIN. Điều này thường xảy ra khi không có chỉ mục phù hợp cho các cột được nối.
* Clustered Index Seek trên bảng STAFF cho thấy SQL Server đang sử dụng chỉ mục để tìm kiếm dữ liệu nhanh hơn.

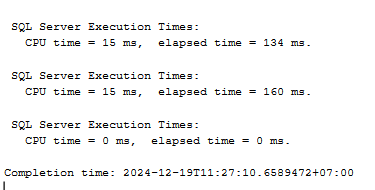
**Tối ưu kịch bản 3:**

Tạo index không phân cụm trên cột StaffID của bảng ORDER\_TABLE:

| **CREATE** NONCLUSTERED **INDEX** idx\_OrderTable\_StaffID **ON** ORDER\_TABLE (StaffID); |
| --- |

**Phân tích kế hoạch thực thi và tốc độ câu truy vấn sau khi tối ưu:**

****

****

**Kết luận:**

* Index Seek trên chỉ mục không phân cụm idx\_Invoice\_CreatedAt cho thấy chỉ mục đang được sử dụng hiệu quả để tìm kiếm dữ liệu.
* Clustered Index Seek trên bảng STAFF cho thấy SQL Server đang sử dụng chỉ mục để tìm kiếm dữ liệu nhanh hơn.

**Kịch Bản 4: Cập Nhật Lương Theo Bộ Phận và Chi Nhánh**

Mô tả :

Mục đích: Cập nhật đồng loạt mức lương cho tất cả nhân viên thuộc cùng một bộ phận tại một chi nhánh cụ thể.

Đối tượng sử dụng: Quản lý nhân sự, Giám đốc chi nhánh

Tần suất sử dụng: Định kỳ (6 tháng/lần) hoặc khi có thay đổi chính sách lương

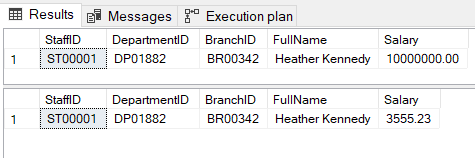
Script:

| **CREATE** **PROCEDURE** UpdateDepartmentSalary  @DepartmentID VARCHAR(255),  @BranchID VARCHAR(255),  @NewSalary DECIMAL(10,2) **AS** **BEGIN**  **BEGIN** TRY  **BEGIN** **TRANSACTION**;  -- Kiểm tra bộ phận và chi nhánh có tồn tại  IF NOT EXISTS (**SELECT** 1 **FROM** DEPARTMENT **WHERE** DepartmentID = @DepartmentID)  **BEGIN**  RAISERROR('Bộ phận không tồn tại!', 16, 1);  RETURN;  **END**   **IF** **NOT** **EXISTS** (**SELECT** 1 **FROM** BRANCH **WHERE** BranchID = @BranchID)  **BEGIN**  RAISERROR('Chi nhánh không tồn tại!', 16, 1);  RETURN;  **END**   -- Cập nhật lương cho nhân viên  **UPDATE** STAFF  **SET** Salary = @NewSalary  **WHERE** DepartmentID = @DepartmentID   **AND** BranchID = @BranchID;   -- Kiểm tra số lượng bản ghi đã cập nhật  IF @@ROWCOUNT = 0  **BEGIN**  RAISERROR('Không có nhân viên nào được cập nhật!', 16, 1);  RETURN;  **END**   **COMMIT** **TRANSACTION**;  **END** TRY  **BEGIN** CATCH  **IF** @@TRANCOUNT > 0  **ROLLBACK** **TRANSACTION**;    **DECLARE** @ErrorMessage **NVARCHAR**(4000) = ERROR\_MESSAGE();  RAISERROR(@ErrorMessage, 16, 1);  **END** CATCH **END** |
| --- |

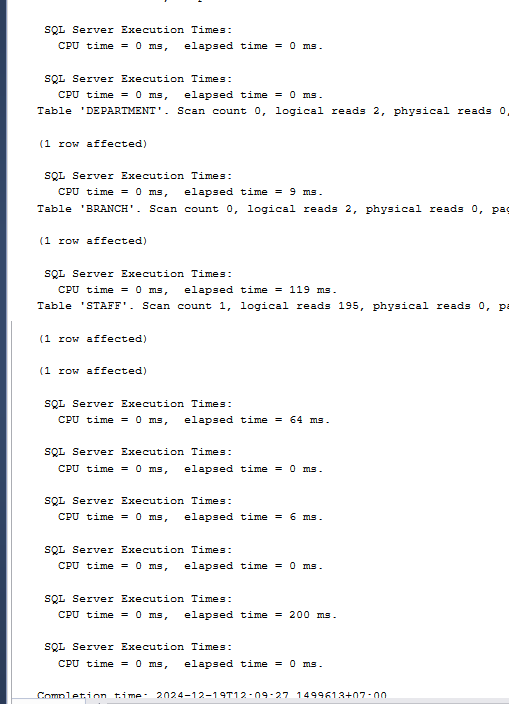
Kiểm tra kết quả câu truy vấn:

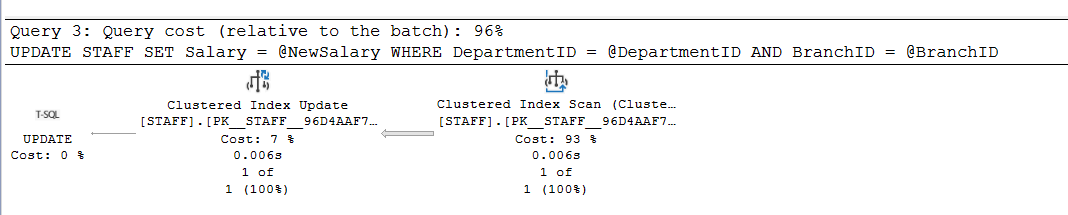
| -- Xem dữ liệu trước khi cập nhật **SELECT** StaffID, DepartmentID, BranchID, FullName, Salary **FROM** STAFF  **WHERE** DepartmentID = 'DP01882' **AND** BranchID = 'BR00342';  -- Thực thi procedure EXEC UpdateDepartmentSalary   @DepartmentID = 'DP01882',  @BranchID = 'BR00342',  @NewSalary = 3555.23;  -- Xem dữ liệu sau khi cập nhật **SELECT** StaffID, DepartmentID, BranchID, FullName, Salary **FROM** STAFF  **WHERE** DepartmentID = 'DP01882' **AND** BranchID = 'BR00342'; |
| --- |

Kết quả:



**Phân tích kế hoạch thực thi và tốc độ câu truy vấn trước khi tối ưu:**

****

****

Kết luận:

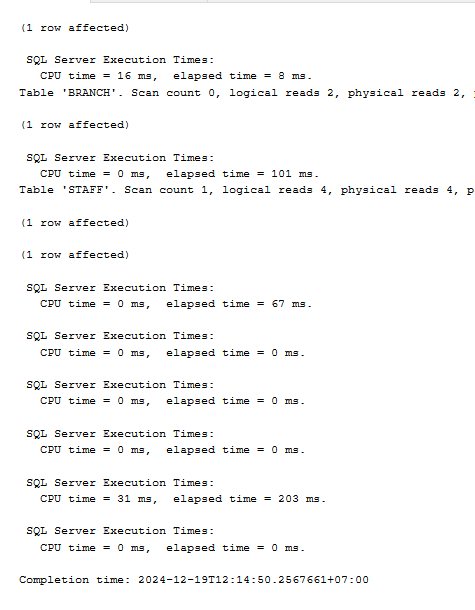
Truy vấn đang sử dụng Clustered Index Scan, có thể không tối ưu nếu bảng STAFF lớn.

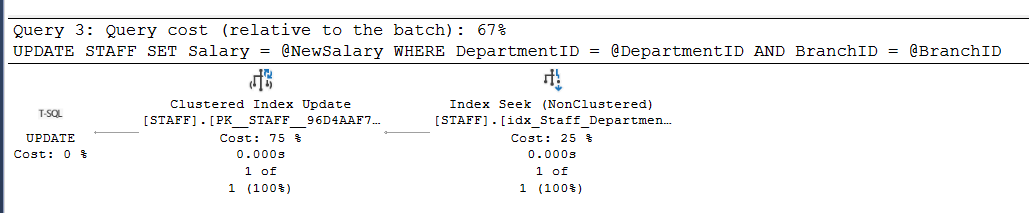
**Tối ưu kịch bản 4**

Tạo chỉ mục không phân cụm trên các cột DepartmentID và BranchID để chuyển từ Index Scan sang Index Seek.

| **CREATE** NONCLUSTERED **INDEX** idx\_Staff\_Department\_Branch  **ON** STAFF (DepartmentID, BranchID); |
| --- |

**Phân tích kế hoạch thực thi và tốc độ câu truy vấn sau khi tối ưu:**

****

****

Kết luận:

* Index Seek cho thấy chỉ mục không phân cụm idx\_Staff\_Department\_Branch đang được sử dụng hiệu quả để tìm kiếm dữ liệu.
* Cập nhật chỉ mục phân cụm vẫn chiếm phần lớn chi phí, điều này là bình thường khi thực hiện cập nhật dữ liệu.

**Kịch Bản 5: Phân Tích Doanh Thu Theo Tháng**

Mô Tả

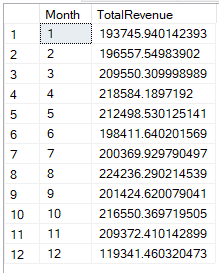
Công ty muốn phân tích doanh thu hàng tháng để đánh giá hiệu suất kinh doanh và lập kế hoạch chiến lược.

Script:

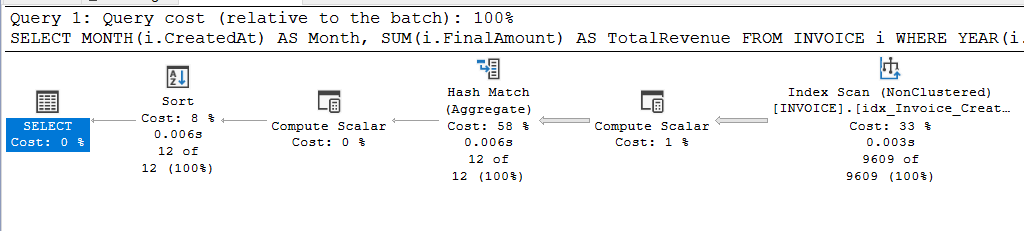
| **CREATE** **PROCEDURE** AnalyzeMonthlyRevenue  @**Year** INT **AS** **BEGIN**  **SELECT**   **MONTH**(i.CreatedAt) **AS** **Month**,  **SUM**(i.FinalAmount) **AS** TotalRevenue  **FROM**   INVOICE i  **WHERE**   **YEAR**(i.CreatedAt) = @**Year**  **GROUP** **BY**   **MONTH**(i.CreatedAt)  **ORDER** **BY**   **Month**; **END** **GO** |
| --- |

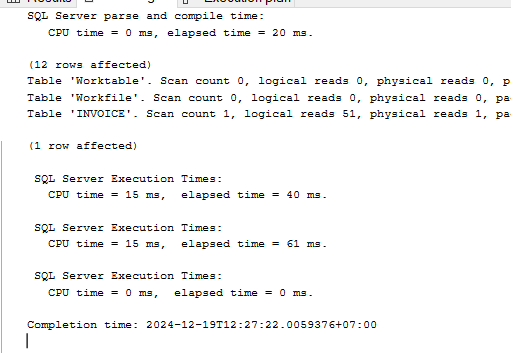
Kết quả:

| -- Phân tích doanh thu cho năm 2024 EXEC AnalyzeMonthlyRevenue @Year = 2024; |
| --- |



**Phân tích kế hoạch thực thi và tốc độ câu truy vấn trước khi tối ưu:**

****

****

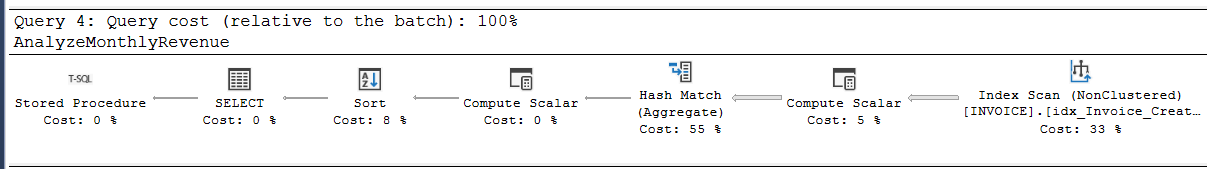
Kết luận:

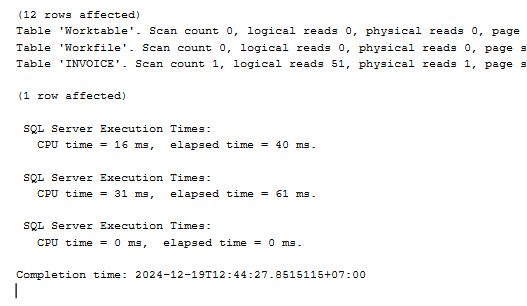
* Index Scan trên chỉ mục không phân cụm idx\_Invoice\_CreatedAt cho thấy chỉ mục đang được sử dụng, nhưng việc quét chỉ mục có thể không tối ưu bằng Index Seek.
* Sử dụng Hash Match để tính toán tổng doanh thu, điều này thường xảy ra khi không có chỉ mục phù hợp cho việc nhóm dữ liệu.

**Tối ưu kịch bản 5**

| **ALTER** **TABLE** INVOICE **ADD** CreatedYear **AS** **YEAR**(CreatedAt); GO **CREATE** NONCLUSTERED **INDEX** idx\_Invoice\_CreatedYear  **ON** INVOICE (CreatedYear); GO **UPDATE** **STATISTICS** INVOICE; GO **ALTER** **PROCEDURE** AnalyzeMonthlyRevenue  @**Year** INT **AS** **BEGIN**  **SELECT**   **MONTH**(i.CreatedAt) **AS** **Month**,  **SUM**(i.FinalAmount) **AS** TotalRevenue  **FROM**   INVOICE i  **WHERE**   i.CreatedYear = @**Year**  **GROUP** **BY**   **MONTH**(i.CreatedAt)  **ORDER** **BY**   **Month**; **END** **GO** |
| --- |

Phân tích kế hoạch thực thi và tốc độ câu truy vấn sau khi tối ưu:





Kết luận:

* Vẫn sử dụng Index Scan, nhưng chi phí đã giảm, cho thấy chỉ mục đang hoạt động hiệu quả hơn.
* Chi phí cho Hash Match vẫn cao, nhưng tổng chi phí đã giảm, cho thấy hiệu suất tổng thể được cải thiện.