**BỘ MÔN HỆ THỐNG THÔNG TIN – KHOA CÔNG NGHỆ THÔNG TIN**

**ĐẠI HỌC KHOA HỌC TỰ NHIÊN THÀNH PHỐ HỒ CHÍ MINH, ĐẠI HỌC QUỐC GIA TP HCM**

GV phụ trách

TS. Nguyễn Trần Minh Thư

ThS. Hồ Thị Hoàng Vy

Nhóm sinh viên thực hiện: Nhóm 11

1. 19127041 – Lê Thị Phương Linh
2. 19127173 – Hoàng Lê Khanh
3. 19127361 – Nguyễn Thị Ngọc Diệu
4. 19127365 – Phan Khải Đông

HỌC KỲ I – NĂM HỌC 2021-2022

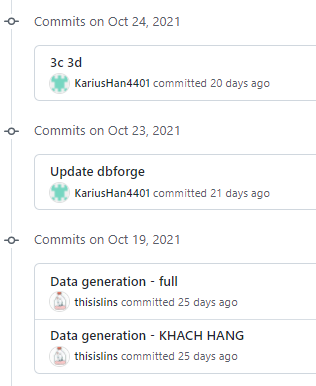
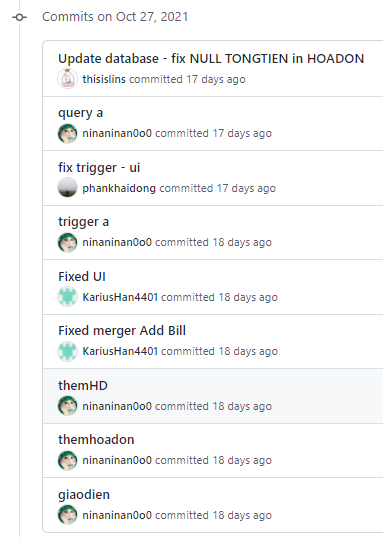
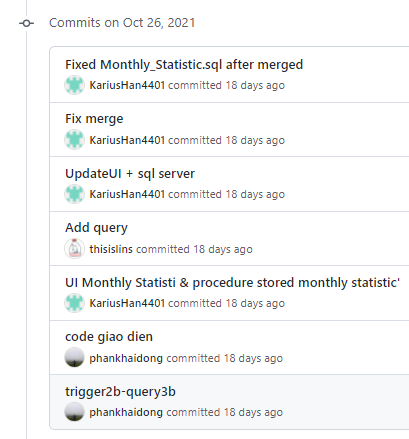
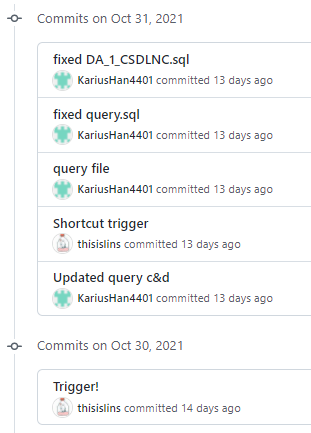
**MÔN CƠ SỞ DỮ LIỆU NÂNG CAO**

**BẢNG THÔNG TIN CHI TIẾT NHÓM**

|  |  |  |
| --- | --- | --- |
| Mã nhóm: | ABD1\_11 | |
| Tên nhóm: | Úm ba la xì bùa | |
| Số lượng: | 4 | |
| MSSV | **Họ tên** | **Email** |
| 19127041 | Lê Thị Phương Linh | 19127041@student.hcmus.edu.vn |
| 19127173 | Hoàng Lê Khanh | 19127173@student.hcmus.edu.vn |
| 19127361 | Nguyễn Thị Ngọc Diệu | 19127361@student.hcmus.edu.vn |
| 19127365 | Phan Khải Đông | 19127365@student.hcmus.edu.vn |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Bảng phân công & đánh giá hoàn thành công việc | | | | |
| Công việc thực hiện | | **Người thực hiện** | **Username github** | **Mức độ**  **hoàn thành** | **Đánh giá của nhóm** |
| Truy vấn e,f; Viết báo cáo | | 19127041 - Lê Thị Phương Linh | thisislins | 100% | 10/10 |
| Truy vấn c, d; Giao diện thống kê doanh thu | | 19127173 - Hoàng Lê Khanh | KariusHan4401 | 100% | 10/10 |
| Truy vấn b; Trigger b; Giao diện thêm hóa đơn | | 19127361 - Nguyễn Thị Ngọc Diệu | Ninaninan0o0 | 100% | 10/10 |
| Truy vấn a; Trigger a; Giao diện xem danh sách hóa đơn | | 19127365 - Phan Khải Đông | phankhaidong | 100% | 10/10 |

* **Minh chứng commits trên github:**



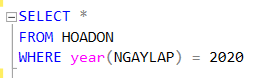
**YÊU CẦU ĐỒ ÁN - BÀI TẬP**

|  |  |
| --- | --- |
| **Loại bài tập** | **Lý thuyết 🗹 Thực hành 🗹 Đồ án Bài tập** |

# GIẢI THÍCH

#### **Giải thích truy vấn**

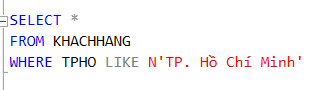
1. *Cho danh sách các hoá đơn lập trong năm 2020.*

**Ý tưởng:** Truy cập vào thuộc tính NGAYLAP của bảng HOADON, lấy ra năm từ NGAYLAP và kiểm tra điều kiện ở WHERE.

Minh họa script SQL

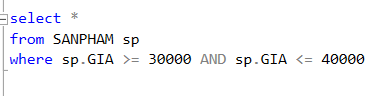
1. *Cho danh sách các khách hàng ở TPHCM.*



**Ý tưởng:** Truy cập vào thuộc tính TPHO của bảng KHACHHANG và đặt điều kiện ‘TPHCM’ ở WHERE.

Minh họa script SQL

1. *Cho danh sách các sản phẩm có giá trong một khoảng từ....đến*

****Ý tưởng:** Tìm danh sách sản phẩm có giá từ 30000 đến 40000, truy cập vào thuộc tính GIA của bảng SANPHAM và đặt điều kiện ở WHERE.

Minh họa script SQL

1. *Cho danh sách các sản phẩm có số lượng tồn <100.*

*Ảnh có chứa văn bản

Mô tả được tạo tự động***Ý tưởng:** Để kiểm tra số lượng tồn, cần truy cập vào bảng SANPHAM và đặt điều kiện ở WHERE.

Minh họa script SQL

1. *Cho danh sách các sản phẩm bán chạy nhất (số lượng bán nhiều nhất).*

Ảnh có chứa văn bản

Mô tả được tạo tự động**Ý tưởng:** Sản phẩm bán chạy nhất là những sản phẩm mà không sản phẩm nào có số lượng bán ra vượt qua nó. Trong bảng CT\_HOADON, một sản phẩm được bán ra ở nhiều hóa đơn, vì vậy, phải nhóm (GROUP) từng sản phẩm lại với nhau để tính được tổng số lượng sản phẩm. Muốn biết thông tin của sản phẩm, sử dụng bảng SANPHAM.

Minh họa script SQL

1. *Cho danh sách các sản phẩm có doanh thu cao nhất.*

*Ảnh có chứa văn bản

Mô tả được tạo tự động***Ý tưởng:** Tương tự câu e, sản phẩm có doanh thu cao nhất là những sản phẩm mà không sản phẩm nào có tổng số tiền thu được vượt qua nó. Trong bảng CT\_HOADON, một sản phẩm được bán ra ở nhiều hóa đơn, vì vậy, phải nhóm (GROUP) từng sản phẩm lại với nhau để tính được doanh thu của sản phẩm đó.

Minh họa script SQL

#### **Giải thích execution plan**

1. Graphical user interface, text, application

   Description automatically generated*Cho danh sách các hoá đơn lập trong năm 2020.*

Logo, company name

Description automatically generated

**Query 2.** Dùng EXCEPT

**Query 1.**

**Excution plan:**

Graphical user interface, text, application

Description automatically generated

**Nhận xét:**

* Estimated subtree cost ở query 1: 2.48217; Estimated subtree cost ở query 2: 6.72092 (gấp 2,7 lần query 1).
* Query cost ở query 2 gấp khoảng 2,7 lần so với query 1 (73% ~ 27%).
* Subtree của query 2 dài hơn query1 1 level.
* Query 1 chỉ thực hiện Clustered Index Scan 1 lần trên bảng HOADON, trong khi đó query 2 có thực hiện thêm Merge Join.
* Tóm lại, sử dụng truy vấn trực tiếp trong trường hợp này tối ưu hơn so với khi dùng loại trừ (EXCEPT).

1. *Cho danh sách các khách hàng ở TPHCM.*Ảnh có chứa văn bản

   Mô tả được tạo tự độngẢnh có chứa văn bản

   Mô tả được tạo tự động

**Query 3.** Sử dụng NOT IN

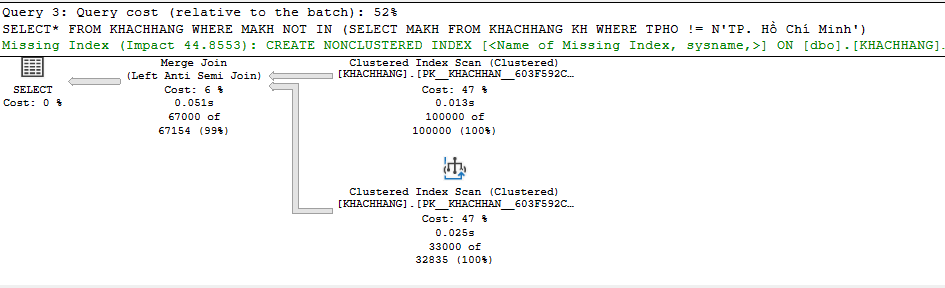
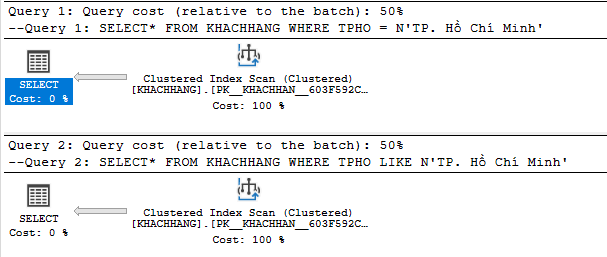
**Query 1.** Sử dụng LOGICAL OPERATOR

**Query 2.** Sử dụng LIKE



Ảnh có chứa văn bản

Mô tả được tạo tự động**Execution plan**



**Nhận xét:**

* Cả hai Query 1 và 2 đều thực hiện clustered index scan trên bảng KHACHHANG sau đó trả về kết quả và có query cost bằng nhau với mức 24% trên tổng tiêu hao khi thực hiện 3 query.
* Query 1, 2 được ước tính sẽ đọc 1 triệu dòng, chi phí cpu 0.110157, subtree cost 2,52587.
* Riêng Query 3 thì query cost lại tốn hơn gấp đôi so với q1 và q2 với tổng thời gian thực thi thực tế sấp xỉ 0.089s còn khi được dự đoán thì query cost chiếm đến 52% trên tổng tiêu hao khi thực hiện 3 query.
* Ghi nhận missing index ở query 3.



1. Ảnh có chứa bàn

   Mô tả được tạo tự độngẢnh có chứa văn bản

   Mô tả được tạo tự động*Cho danh sách các sản phẩm có giá trong một khoảng từ 30.000 đến 40.000*

**Query 2.** Sử dụng BETWEEN AND

**Query 1.** Sử dụng LOGICAL OPERATOR

TH1. Sử dụng LOGICAL OPEARTOR VÀ BETWEEN

Ảnh có chứa văn bản

Mô tả được tạo tự động**Execution plan**

**Nhận xét:**

* Cả hai quy trình giống như nhau.
* Nếu chạy cả hai cùng lúc thì query cost trên mỗi query là 50%.
* Ảnh có chứa văn bản

  Mô tả được tạo tự độngKhi xét trên bước select thì *hiệu suất* và *Estimated Subtree Cost* (0.155023) của cả hai là tương đương nhau.

TH2. Sử dụng NOT IN và NOT EXSITS

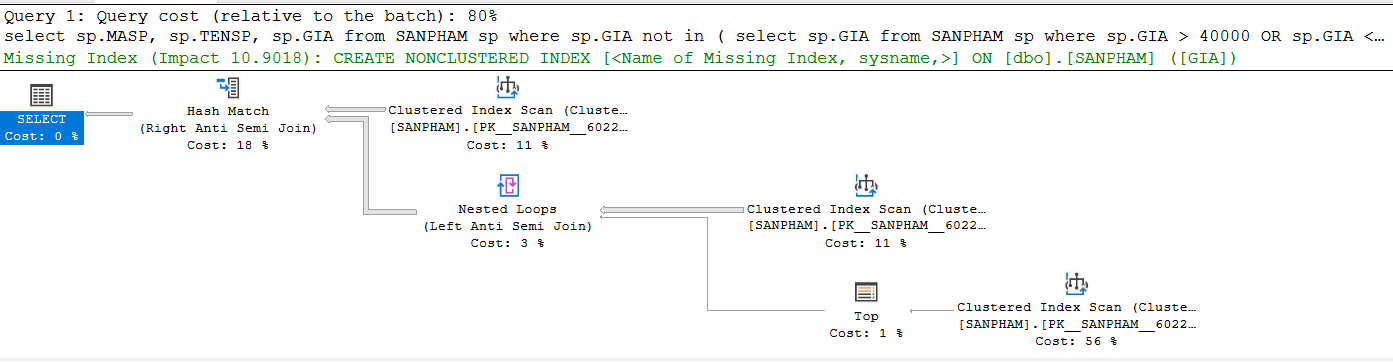
Ảnh có chứa văn bản

Mô tả được tạo tự động

**Query 2.** Sử dụng NOT EXISTS

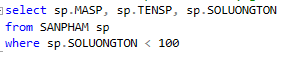
**Query 1.** Sử dụng NOT IN

Ảnh có chứa văn bản

Mô tả được tạo tự động**Execution plan**

**Nhận xét:**

* Ta có thể thấy nếu sử dụng NOT EXISTS thì *Subtree* ngắn hơn khi ta sử dụng NOT IN hai level.
* Nếu chạy cả hai cùng lúc thì query cost của NOT IN gấp 4 lần so với NOT EXISTS (80% - 20%).
* Ở đây ta có thể thấy *execution plan* đề xuất gắn index vào để query sử dụng NOT IN có thể chạy nhanh hơn so với hiện tại.
* Khi xét trên bước select thì *Estimated Subtree Cost* của NOT IN gấp khoảng 4 lần so với của NOT EXISTS (1.41969 và 0.357673) dẫn đến hiệu suất của NOT in sẽ kém hơn so với NOT EXISTS.
* NOT IN phải trải qua nhiều bước lấy dữ liệu trước khi đưa ra kết quả cuối cùng điều đó dẫn đến sử dụng NOT IN kém hiệu quả hơn so với khi sử dụng NOT EXISTS.

1. *Cho danh sách các sản phẩm có số lượng tồn <100.*Ảnh có chứa văn bản

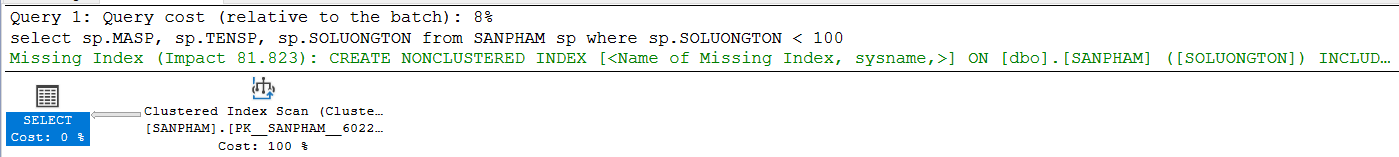
   Mô tả được tạo tự độngẢnh có chứa văn bản

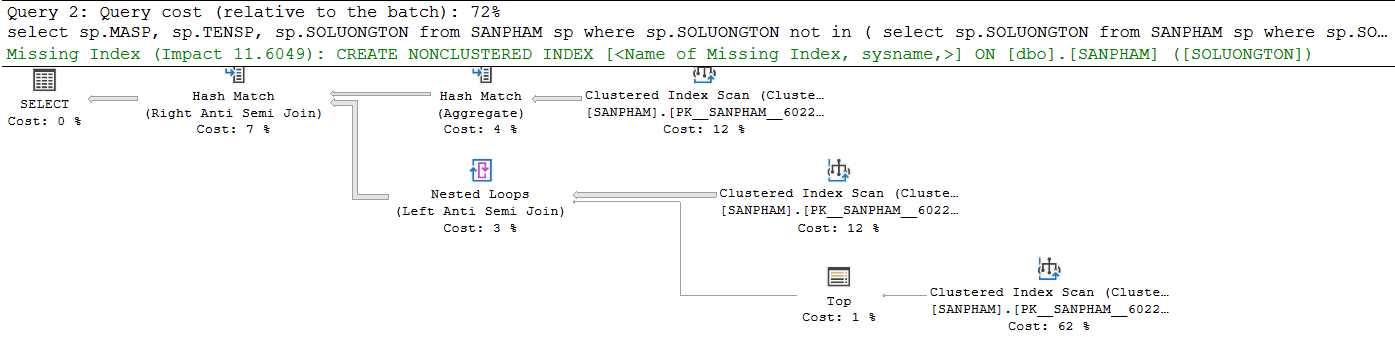
   Mô tả được tạo tự động

**Query 3.** Sử dụng NOT EXISTS

**Query 2.** Sử dụng NOT IN

**Query 1.**

**Execution plan**



Ảnh có chứa văn bản

Mô tả được tạo tự động**Nhận xét:**

* Ta có thể thấy nếu sử dụng NOT EXISTS thì *Subtree* ngắn hơn khi ta sử dụng NOT IN hai level.
* Nếu chạy cả ba cùng lúc thì query cost của NOT IN gấp 3.6 lần so với NOT EXISTS (72% - 20%) và gấp 9 lần so với sử dụng *logical operator “<”* (72% - 8%).
* Ở đây ta có thể thấy *execution plan* đề xuất gắn index vào để query sử dụng NOT IN và *logical operator “<”* có thể chạy nhanh hơn so với hiện tại.
* Khi xét trên bước select thì *Estimated Subtree Cost* của NOT IN gấp khoảng 3.7 lần so với của NOT EXISTS (1.31198 và 0.358161) và gấp khoảng 8.5 lần so với của *logical operator “<”* (1.31198 và 0.155023) dấn đến hiệu suất kém hơn NOT EXISTS và *logical operator “<”.* Còn logical operator “<” nhỉnh hơn khoảng 2.3 lần so với NOT EXISTS.
* NOT IN phải trải qua nhiều bước lấy dữ liệu trước khi đưa ra kết quả cuối cùng điều đó dẫn đến sử dụng NOT IN kém hiệu quả hơn so với khi sử dụng NOT EXISTS.
* Và NOT EXISTS phải truy xuất trên hai bảng dữ liệu sau đó merge chúng lại với nhau để đưa ra kết quả cuối cùng trong khi đó *logical operator “<”* chỉ cần truy xuất thẳng trên một bảng dữ liệu tương ứng với điều kiện.
* Vì thế hiệu suất của ba query trên giảm dần theo thứ tự: *logical operator “<” ->* NOT EXISTS -> NOT IN.

1. **Ảnh có chứa văn bản

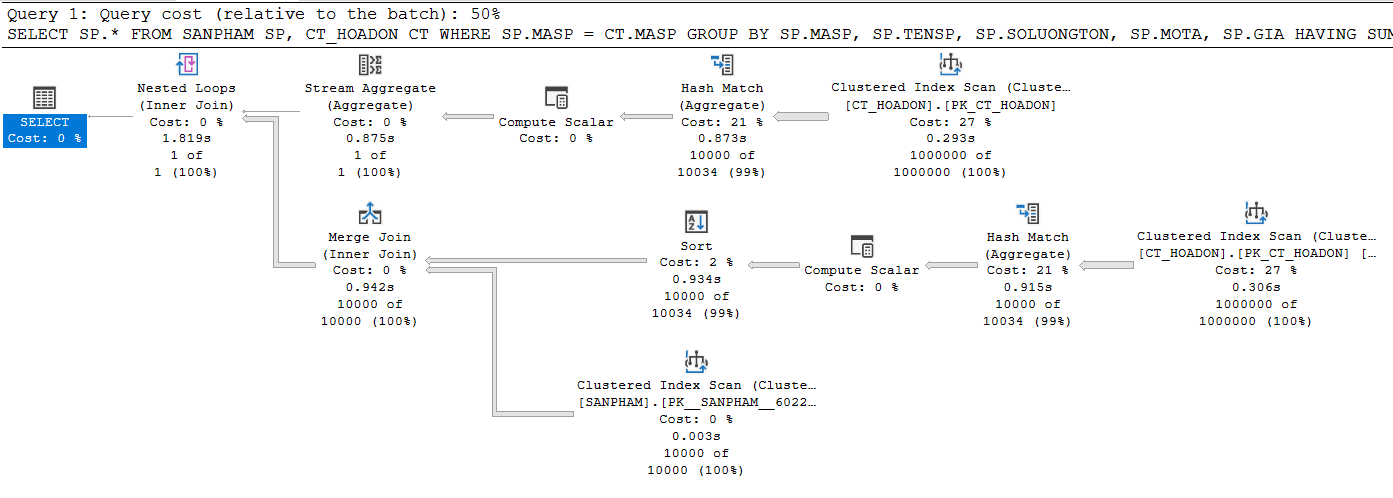
   Mô tả được tạo tự độngẢnh có chứa văn bản, trong nhà, ảnh chụp màn hình

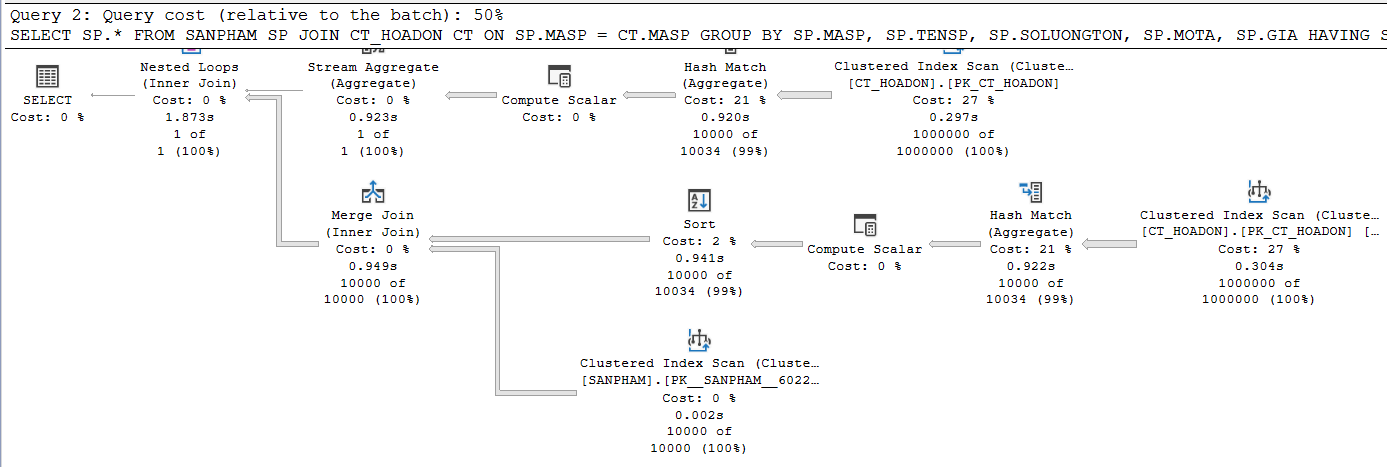
   Mô tả được tạo tự động***Cho danh sách các sản phẩm bán chạy nhất (số lượng bán nhiều nhất).*

TH1. Sử dụng phép kết bảng

**Query 1.** Sử dụng kết bảng thông thường

**Query 2.** Sử dụng kết bảng với JOIN

**Execution plan**

****

**Nhận xét:**

* Query cost trên mỗi truy vấn là 50% / batch.
* Quy trình tương tự nhau.
* Xét trên bước cuối cùng (lệnh SELECT), hiệu suất của 2 truy vấn tương đương nhau, với *estimated subtree cost* gần bằng nhau.
* **Ảnh có chứa văn bản

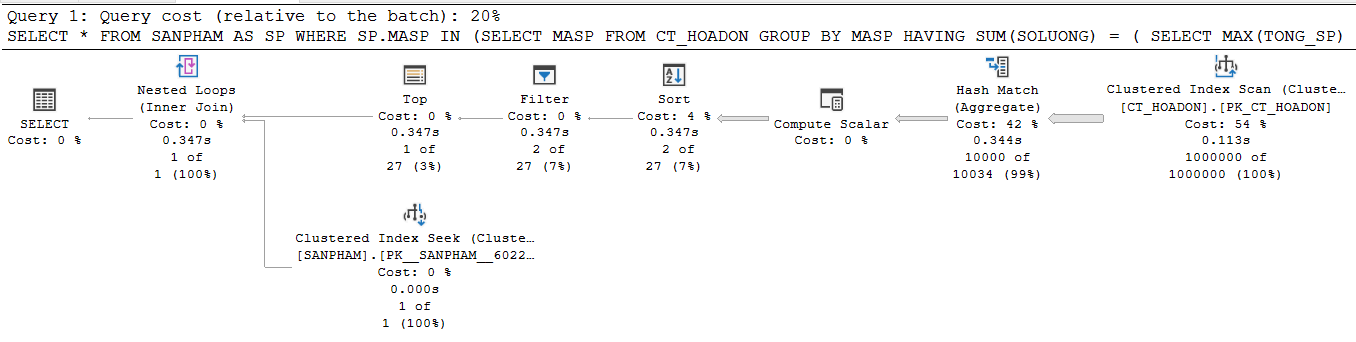
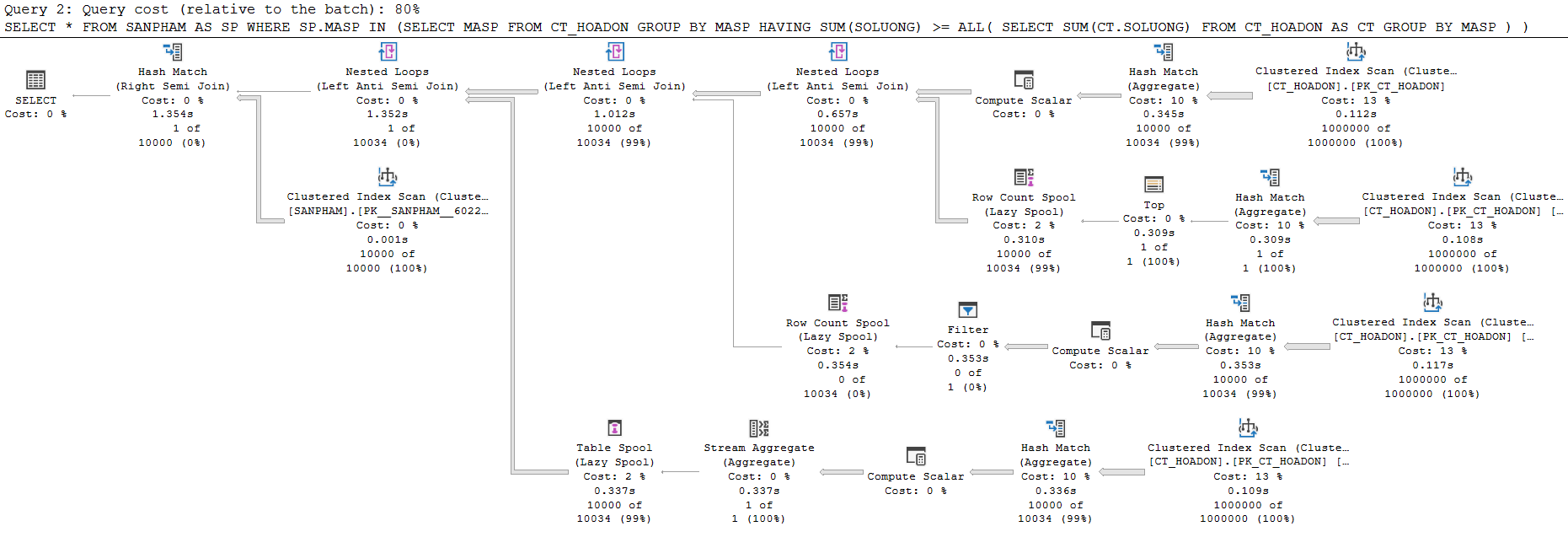
  Mô tả được tạo tự độngẢnh có chứa văn bản

  Mô tả được tạo tự động**Tuy nhiên, trong trường hợp số lượng bản ghi là rất lớn, sử dụng kết bảng với JOIN tối ưu hơn.

TH2. Sử dụng MAX (Aggregate function) và ALL (Logical operator)

**Query 1.** Sử dụng MAX

**Query 2.** Sử dụng ALL

**Execution plan**

**Nhận xét:**

* Khi sử dụng MAX, query cost = 20%. Trong khi đó, sử dụng ALL, query cost chiếm 80%.
* Xét trên bước cuối cùng (SELECT), sử dụng MAX cho kết quả *estimated subtree cost* = 16.3388, nhanh gấp 4 lần so với sử dụng ALL - *estimated subtree cost*  = 66.3215.
* Quy trình khi sử dụng ALL tương đối phức tạp hơn, phải trải qua nhiều vòng lặp scan tất cả bản ghi khi xét một bản ghi bất kỳ.
* **Ảnh có chứa văn bản

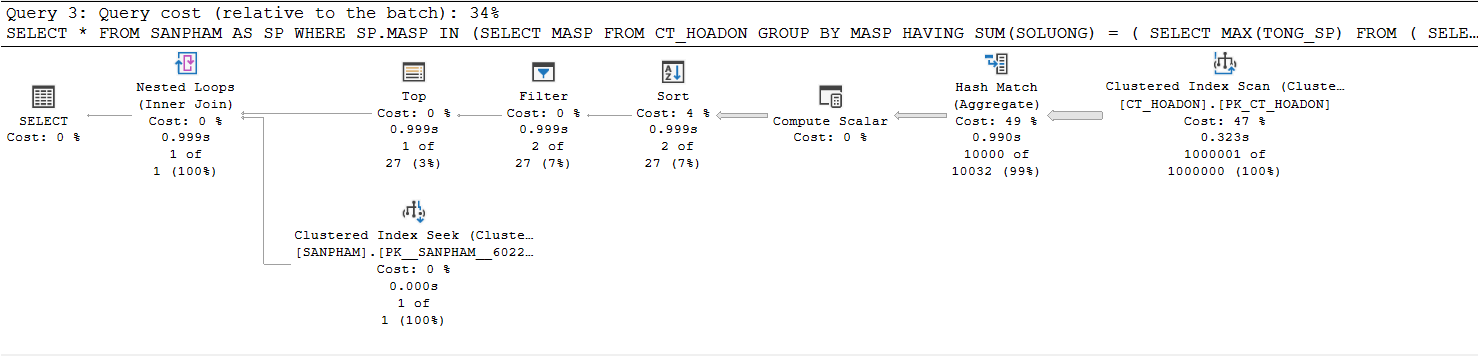
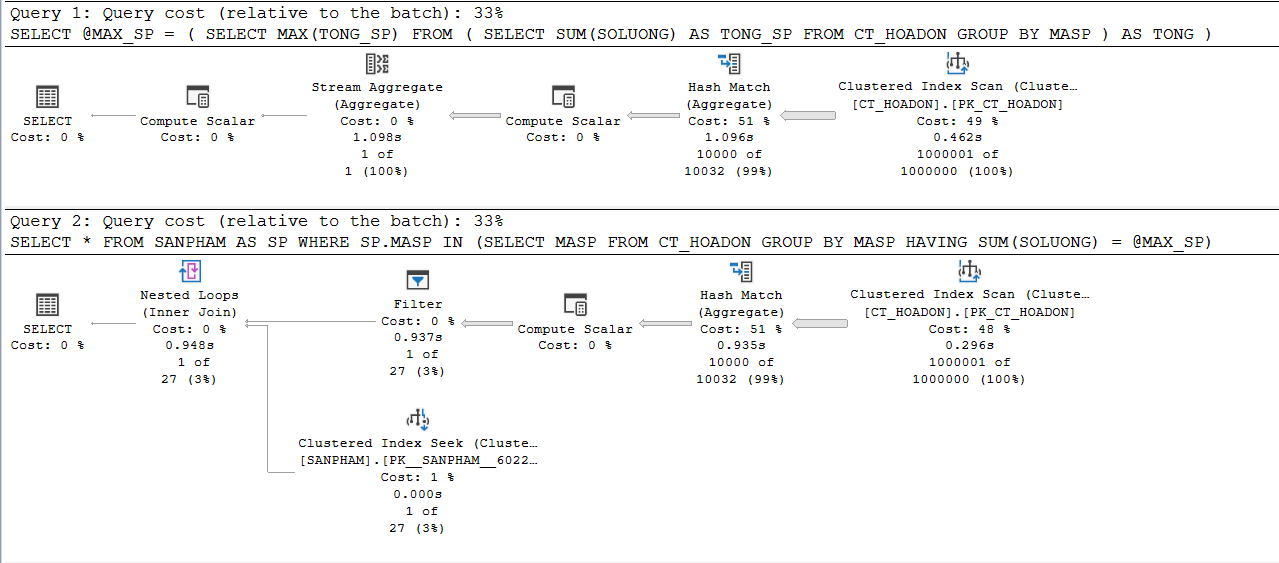
  Mô tả được tạo tự độngẢnh có chứa văn bản

  Mô tả được tạo tự động**Tóm lại, sử dụng aggregate function – MAX tối ưu hơn so với logical operator – ALL.

**Query 1, 2.** Sử dụng biến @MAX\_SP để lưu số lượng nhiều nhất

**Query 3.** Sử dụng truy vấn lồng

TH3. Sử dụng biến tạm và truy vấn lồng

******Execution plan**

**Nhận xét:**

* Khi sử dụng biến @MAX\_SP, phải kết hợp 2 query, vì thế query cost chiếm khoảng 66% / batch.
* Xét trên bước cuối cùng (SELECT), sử dụng @MAX\_SP cho kết quả *estimated subtree cost*  27, gấp 2 lần so với sử dụng truy vấn lồng để tìm số lượng sản phẩm bán nhiều nhất - *estimated subtree cost*  = 14.02.
* Ở bước Clustered Index Seek, truy vấn lồng đã có thể tìm ra kết quả cuối cùng, tuy nhiên, khi sử dụng biến tạm, vẫn còn trả về các kết quả rác.
* Tóm lại, sử dụng truy vấn lồng trong trường hợp này tối ưu hơn so với sử dụng biến tạm.

1. *Cho danh sách các sản phẩm có doanh thu cao nhất.*

Khi tái sử dụng các trường hợp phân tích execution plan như tìm sản phẩm bản chạy nhất, kết quả trả về mang ý nghĩa và giá trị tương đương. Cụ thể, sắp xếp mức tối ưu theo thứ tự từ lớn đến bé: **Sử dụng aggregate function (MAX) > sử dụng biến tạm @MAX\_DT > sử dụng logical operator (ALL).**

Ảnh có chứa văn bản

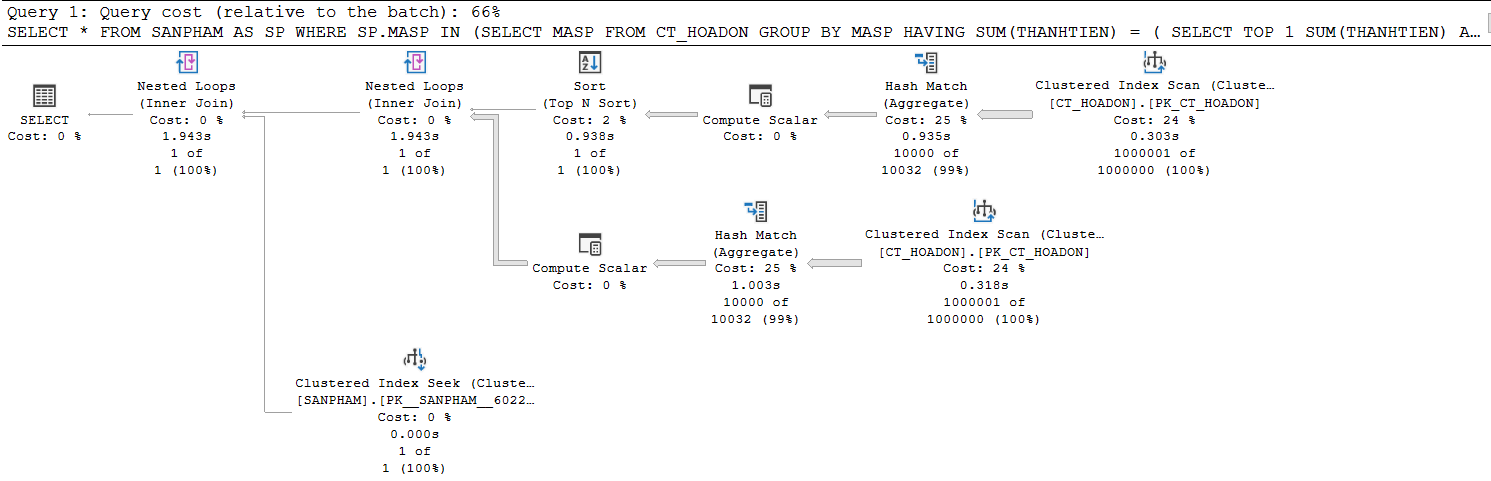
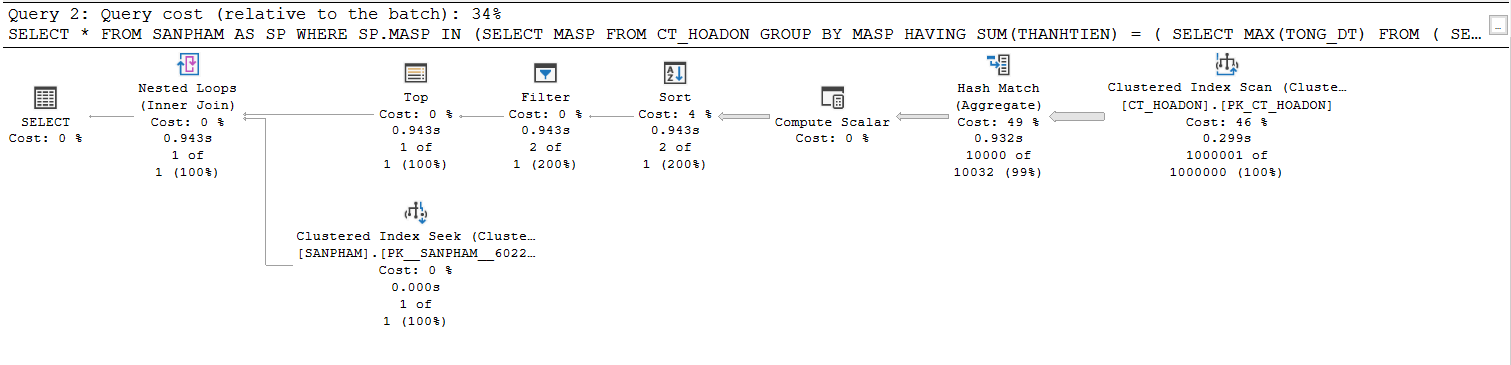
Mô tả được tạo tự độngẢnh có chứa văn bản

Mô tả được tạo tự độngNgoài ra:

**Query 2.** Sử dụng MAX tìm doanh thu cao nhất

**Query 1.** Sử dụng ORDER BY để sắp xếp (SORT) doanh thu

TH. Sử dụng ORDER BY và MAX

**Execution plan**

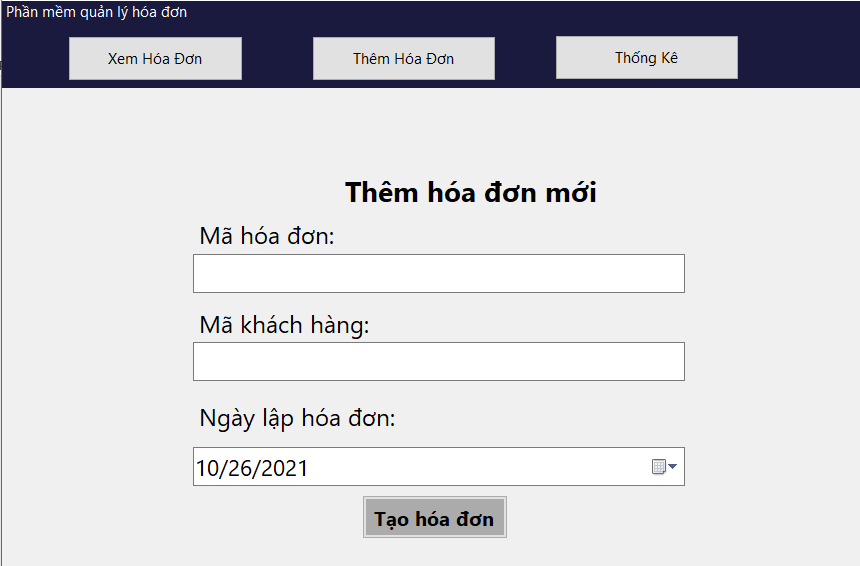
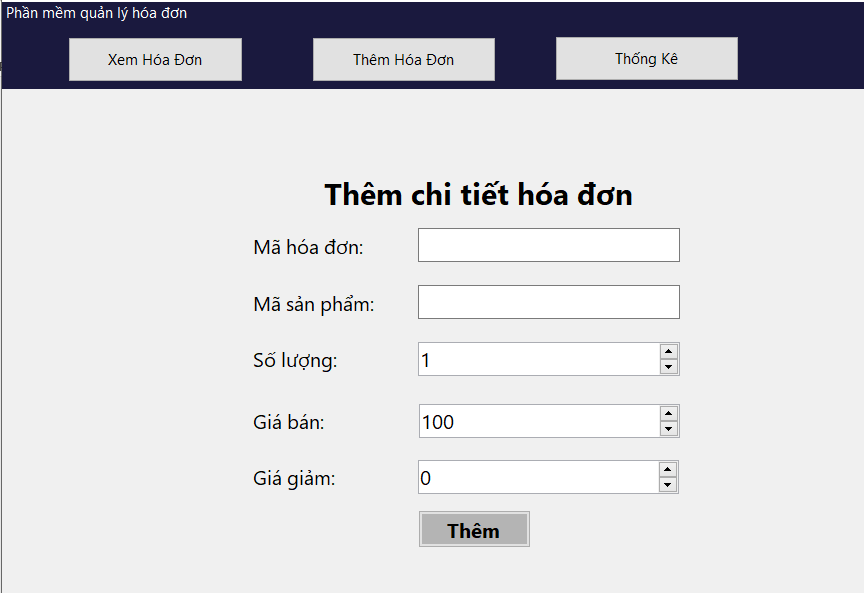
**Nhận xét:**

* Khi sử dụng ORDER BY, query cost = 66%. Trong khi đó, sử dụng MAX, query cost chiếm 34%.
* Xét trên bước cuối cùng (SELECT), sử dụng ORDER BY cho kết quả *estimated subtree cost* = 27.5205, gấp gần 2 lần so với sử dụng MAX - *estimated subtree cost*  = 14.0491.
* Sử dụng ORDER BY phải scan nhiều lần hơn so với khi sử dụng MAX.
* Tóm lại, sử dụng MAX để tìm doanh thu cao nhất tối ưu hơn so với sắp xếp doanh thu.

# MINH HỌA GIAO DIỆN

Với mỗi lần chạy giao diện ở các máy khác nhau, vào ConnectionString.cs để chỉnh sửa đường dẫn liên kết đến database.

#### **Thêm mới hóa đơn**

**Tính năng:** Thêm thông tin hóa đơn mới và chi tiết hóa đơn tương ứng.

Giao diện thêm hóa đơn

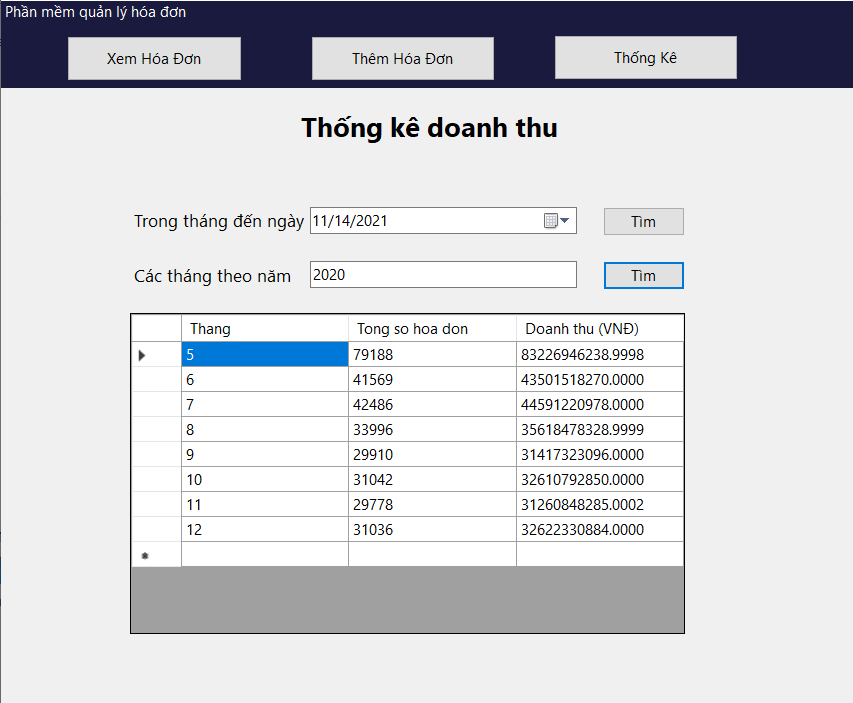
Giao diện thêm chi tiết hóa đơn

#### **Xem danh sách hóa đơn**

Giao diện xem danh sách hóa đơn

**Tính năng:** Xem danh sách thông tin hóa đơn với đầu vào là mã hóa đơn (MAHD), mã khách hàng (MaKH) hoặc ngày, tháng, năm cho trước.

#### **Thống kê doanh thu theo tháng**

**Tính năng:** Xem thống kê doanh thu của các tháng trong năm cho trước hoặc thống kê doanh thu của một tháng tính đến ngày cho trước.

Giao diện xem thống kê doanh thu

# TÀI LIỆU THAM KHẢO

* Dùng C# lập trình giao diện và kết nối dữ liệu: <https://youtu.be/4bQfxbSj4tg>
* Đọc hiểu execution plan trong SQL Server: <https://www.quest.com/community/blogs/b/database-management/posts/sql-server-execution-plan-what-is-it-and-how-does-it-help-with-performance-problems>

# LINK GOOGLE DRIVE + GITHUB NHÓM

* GG Drive:

<https://drive.google.com/drive/folders/1CxGT-sGS7g_NPOHfXfwLkkPa98dntqhz?usp=sharing>

* Github:

<https://github.com/thisislins/Advanced_SQL_DA1_GROUP11.git>