TRƯỜNG ĐẠI HỌC KINH TẾ - LUẬT

**KHOA HỆ THỐNG THÔNG TIN**

**BÁO CÁO GIỮA KỲ**

**HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU**

**MÃ HỌC PHẦN: 242MI6601**

**TỐI ƯU HÓA HIỆU SUẤT TRUY VẤN HỖ TRỢ DOANH NGHIỆP RA QUYẾT ĐỊNH KINH DOANH**

**Giảng viên hướng dẫn:** TS Nguyễn Thôn Dã

**Danh sách thành viên nhóm:**

1. K224060766, Trương Hoàng Anh

2. K224060798, Nguyễn Đặng Hoài Nam

Thành phố Hồ Chí Minh, 2025

# **Bảng tự đánh giá thành viên nhóm**

|  |  |  |  |
| --- | --- | --- | --- |
| *TT* | **MSSV** | **Họ và tên** | **Điểm**  **tự đánh giá**  **(thang điểm 10)** |
| 1 | K224060766 | Trương Hoàng Anh | 10 |
| 2 | K224060798 | Nguyễn Đặng Hoài Nam | 10 |

# **Lời cảm ơn của nhóm**

Trong quá trình thực hiện và cho ra kết quả đồ án, nhóm đã nhận được sự giúp đỡ cũng như là quan tâm. Đồ án được hoàn thành dựa trên sự tham khảo, lược khảo từ các kết quả nghiên cứu đi trước, các tài liệu chuyên môn của các tác giả ở các trường Đại học, các tổ chức nghiên cứu trong và ngoài nước.

Trước hết, nhóm xin gửi lời cảm ơn sâu sắc đến Thầy TS Nguyễn Thôn Dã – người trực tiếp hướng dẫn nhóm nghiên cứu, đã luôn đồng hành cùng nhóm trong suốt quá trình thực hiện phân tích và hoàn thành đề tài.

Nhóm xin trân trọng cảm ơn Ban giám hiệu cùng toàn thể các thầy cô giáo Trường Đại học Kinh tế - Luật, Đại học Quốc Gia Thành phố Hồ Chí Minh đã tận tình truyền đạt những kiến thức hữu ích, giúp đỡ nhóm trong quá trình học tập và nghiên cứu. Tuy có nhiều cố gắng, nhưng đề tài này không tránh khỏi những hạn chế và thiếu sót. Nhóm kính mong Quý thầy cô, những người quan tâm đến đề tài tiếp tục có những ý kiến đóng góp để giúp đề tài ngày càng hoàn thiện.

# **Lời cam kết**

Chúng tôi cam đoan kết quả nghiên cứu này là của riêng chúng tôi, chúng tôi khẳng định không sao chép kết quả nghiên cứu của những cá nhân hoặc nhóm nghiên cứu nào khác.

Ho Chi Minh City, ngày 17 tháng 02 năm 2025

Tập thể thành viên nhóm

**1. Mô tả tình huống thực hành**

**Mô tả tình huống thực hành Case Study**

Trong quá trình sử dụng và phân tích dữ liệu từ bảng **InvoiceLines** của cơ sở dữ liệu **WorldWideImporters**, chúng tôi nhận thấy bảng này đóng vai trò quan trọng trong việc tổng hợp **doanh thu** và **lợi nhuận** theo tháng. Đây là một yêu cầu phổ biến trong các doanh nghiệp, giúp hỗ trợ việc lập **báo cáo tài chính** và **ra quyết định kinh doanh**. Tuy nhiên, khi dữ liệu tăng lên, quá trình truy vấn trở nên chậm chạp, ảnh hưởng đến hiệu suất hệ thống. Vì vậy, chúng tôi quyết định **tối ưu hóa hiệu suất** cho các truy vấn này để cải thiện tốc độ xử lý.

### Lợi ích đạt được

Sau khi thực hiện các tối ưu, hệ thống có thể **giảm gấp 30 lần thời gian truy vấn** , giúp các doanh nghiệp có thể lấy được báo cáo nhanh chóng, hỗ trợ quá trình phân tích dữ liệu một cách hiệu quả hơn.

### Điểm nổi bật và thú vị

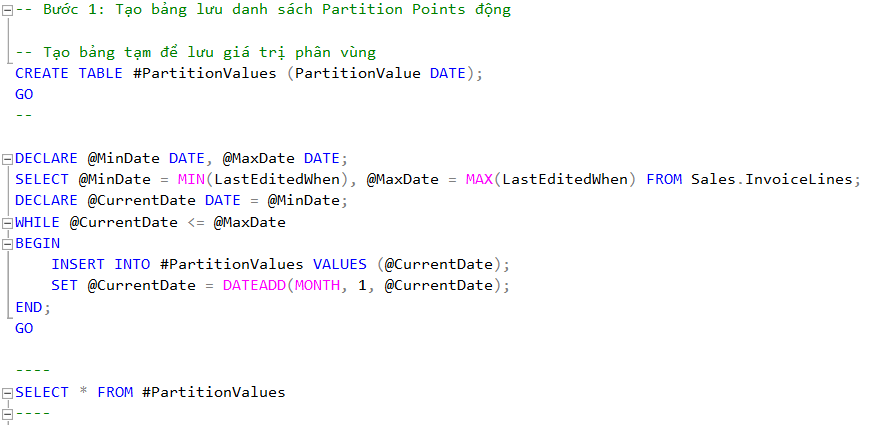
Điểm đặc biệt trong nghiên cứu của chúng tôi là sự kết hợp của ba kỹ thuật mạnh mẽ:

* **Partitioning**: Chia dữ liệu thành từng phân vùng theo tháng giúp truy vấn nhanh hơn.
* **Columnstore Index (CCI)**: Giúp tối ưu hóa bộ nhớ và tăng tốc độ xử lý. Vì dữ liệu của chúng tôi chủ yếu phục vụ cho việc **tổng hợp doanh thu và lợi nhuận theo tháng**, đây là dạng **OLAP (Online Analytical Processing)**, nên CCI là lựa chọn phù hợp nhất. Columnstore Index không chỉ giúp nén dữ liệu hiệu quả, tiết kiệm bộ nhớ mà còn tăng tốc truy vấn dạng tổng hợp như SUM(), AVG(),... - những phép toán thường xuyên được sử dụng trong báo cáo tài chính.
* **Dynamic Partitioning**: Tạo phân vùng động giúp hệ thống có thể tự mở rộng mà không cần can thiệp thủ công.

Ban đầu, chúng tôi thực hiện việc nhập dữ liệu vào từng phân vùng một cách thủ công, đồng thời cũng tính toán báo cáo doanh thu và lợi nhuận theo tháng thủ công. Điều này tuy hiệu quả ở quy mô nhỏ nhưng lại mất nhiều thời gian khi dữ liệu lớn dần. Do đó, chúng tôi đã phát triển thêm một hàm (function) tự động tổng hợp doanh thu và lợi nhuận theo tháng, đồng thời tạo cơ chế phân vùng động, giúp hệ thống linh hoạt và dễ mở rộng hơn.

**2. Trình bày chi tiết nội dung thực hiện**

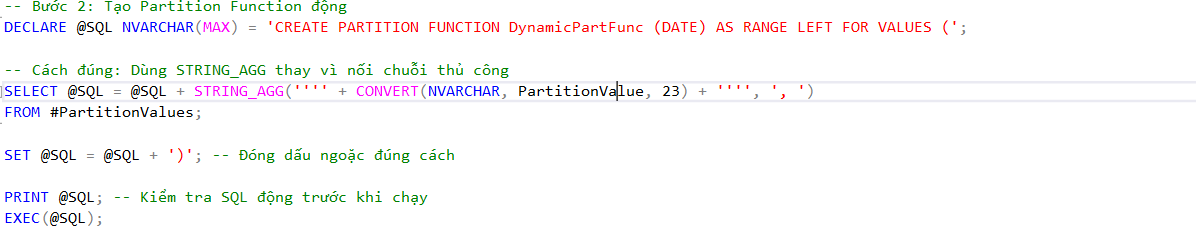
**Action 1: Tạo bảng lưu danh sách Partition Points động**



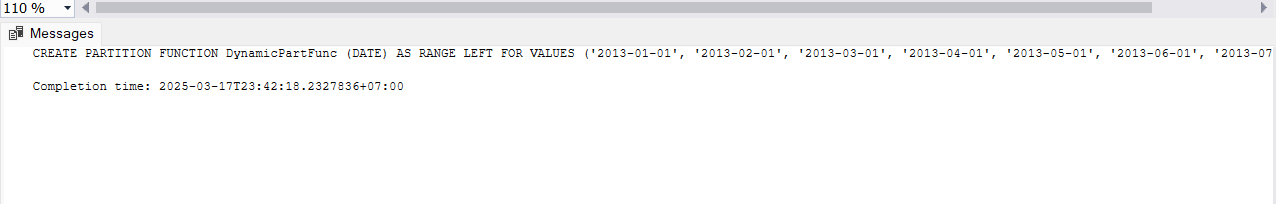
* **Kết quả**



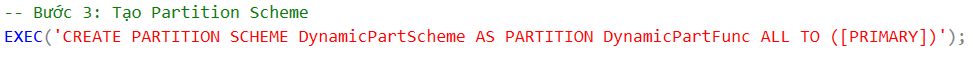
**Action 2: Tạo Partition Function động**



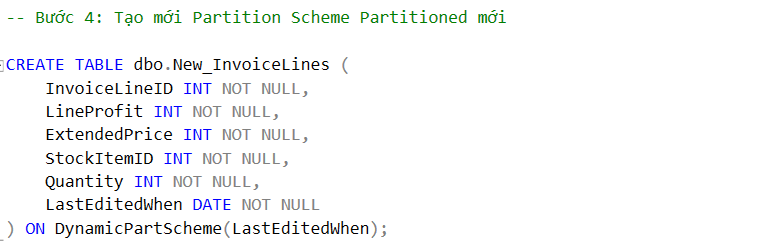
* **Kết quả**



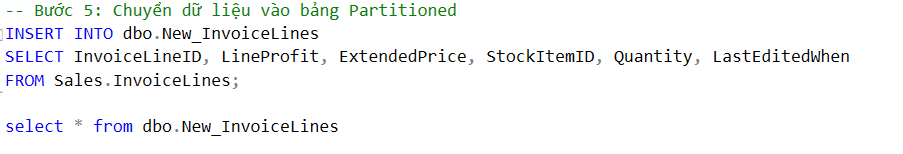
**Action 3: Tạo Partition Scheme**



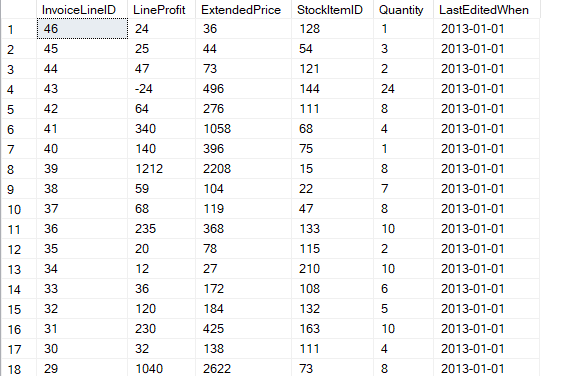
**Action 4: Tạo bảng Partitioned mới**



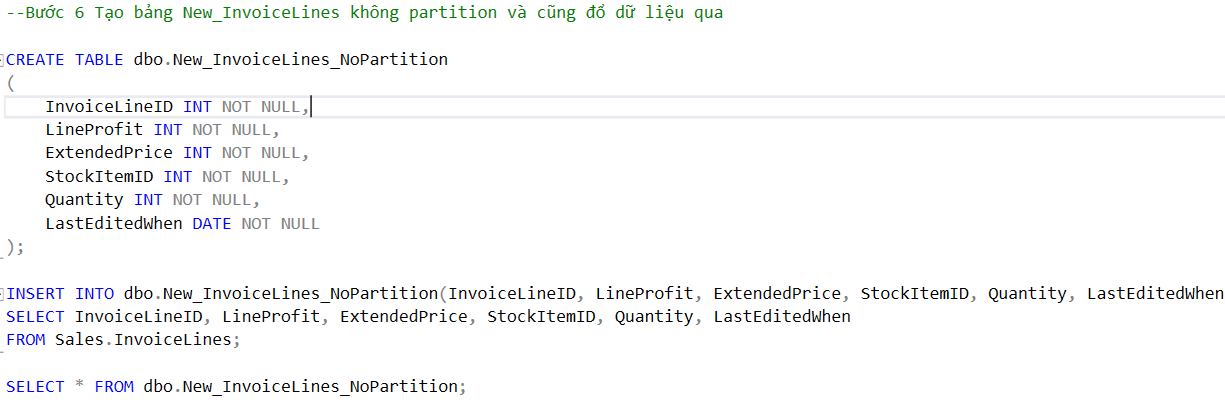
**Action 5: Chuyển dữ liệu vào bảng Partitioned**



* Kết quả:



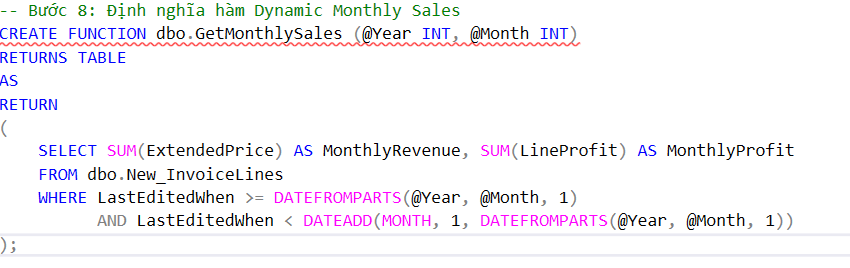
**Action 6: Tạo bảng New\_InvoiceLines không partition và cũng đổ dữ liệu qua**



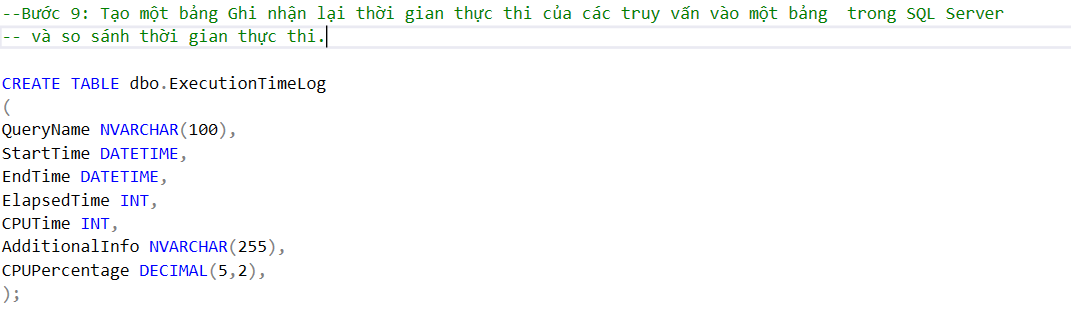
**Action 7: Tạo Clustered Columnstore Index (CCI) trên bảng dbo.New\_InvoiceLines**



**Action 8: Định nghĩa hàm Dynamic Monthly Sales**



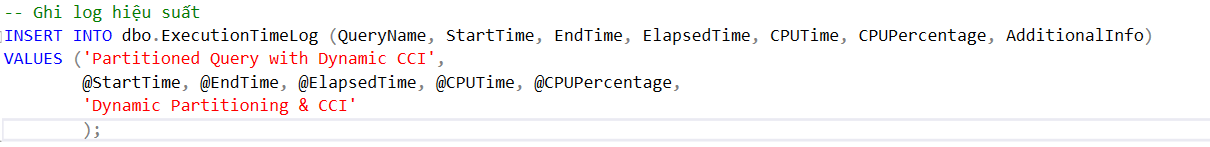
**Action 9: Tạo một bảng Ghi nhận lại thời gian thực thi của các truy vấn vào một bảng trong SQL Server và so sánh thời gian thực thi.**



**Action 10: Ghi nhận thời gian thực thi truy vấn**



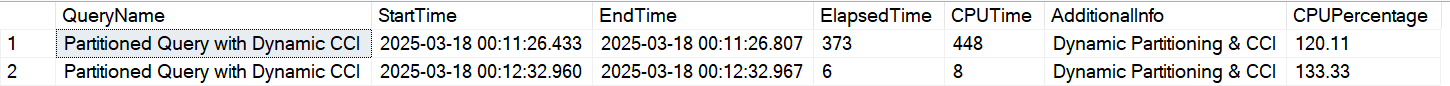
**Ghi log hiệu suất**



**Kiểm tra log**



* **Kết quả:**

****

**3. Nhận xét**

Việc áp dụng Partitioning kết hợp với Columnstore Index đã giúp tối ưu hóa đáng kể hiệu suất truy vấn trên bảng InvoiceLines. Cụ thể, thời gian thực thi đã giảm từ 373 mili giây xuống còn 6 mili giây, tức là nhanh hơn hơn 60 lần so với trước đây.

Nhờ kỹ thuật phân vùng động (Dynamic Partitioning), dữ liệu được tổ chức một cách hợp lý hơn, giúp giảm tải bộ nhớ và tăng tốc độ xử lý truy vấn. Bên cạnh đó, Clustered Columnstore Index (CCI) giúp cải thiện hiệu suất truy vấn phân tích dữ liệu, giảm mức sử dụng I/O và tối ưu hóa CPU.

**4. Mở rộng**

**Yêu cầu:** Mã hóa và tính thời gian của các bảng mã hóa sử dụng kỹ thuật trên và ngược lại  
  
**Code:**

-- Bước 1: Tạo bảng lưu danh sách Partition Points động

CREATE TABLE #PartitionValues (PartitionValue DATE);

GO

DECLARE @MinDate DATE, @MaxDate DATE;

SELECT @MinDate = MIN(LastEditedWhen), @MaxDate = MAX(LastEditedWhen) FROM Sales.InvoiceLines;

DECLARE @CurrentDate DATE = @MinDate;

WHILE @CurrentDate <= @MaxDate

BEGIN

INSERT INTO #PartitionValues VALUES (@CurrentDate);

SET @CurrentDate = DATEADD(MONTH, 1, @CurrentDate);

END;

GO

-- Bước 2: Tạo Partition Function động

DECLARE @SQL NVARCHAR(MAX) = 'CREATE PARTITION FUNCTION DynamicPartFunc (DATE) AS RANGE LEFT FOR VALUES (';

SELECT @SQL = @SQL + STRING\_AGG('''' + CONVERT(NVARCHAR, PartitionValue, 23) + '''', ', ') FROM #PartitionValues;

SET @SQL = @SQL + ')';

PRINT @SQL;

EXEC(@SQL);

-- Bước 3: Tạo Partition Scheme

EXEC('CREATE PARTITION SCHEME DynamicPartScheme AS PARTITION DynamicPartFunc ALL TO ([PRIMARY])');

-- Bước 4: Tạo bảng Partitioned với mã hóa AES-256

CREATE TABLE dbo.New\_InvoiceLines5 (

InvoiceLineID VARBINARY(MAX) NOT NULL,

LineProfit VARBINARY(MAX) NOT NULL,

ExtendedPrice VARBINARY(MAX) NOT NULL,

StockItemID VARBINARY(MAX) NOT NULL,

Quantity VARBINARY(MAX) NOT NULL,

LastEditedWhen DATE NOT NULL

) ON DynamicPartScheme(LastEditedWhen);

-- Bước 5: Chuyển dữ liệu vào bảng Partitioned với mã hóa

OPEN SYMMETRIC KEY MySymmetricKey DECRYPTION BY CERTIFICATE MyCertificate;

INSERT INTO dbo.New\_InvoiceLines5

SELECT

EncryptByKey(Key\_GUID('MySymmetricKey'), CAST(InvoiceLineID AS VARBINARY(MAX))),

EncryptByKey(Key\_GUID('MySymmetricKey'), CAST(LineProfit AS VARBINARY(MAX))),

EncryptByKey(Key\_GUID('MySymmetricKey'), CAST(ExtendedPrice AS VARBINARY(MAX))),

EncryptByKey(Key\_GUID('MySymmetricKey'), CAST(StockItemID AS VARBINARY(MAX))),

EncryptByKey(Key\_GUID('MySymmetricKey'), CAST(Quantity AS VARBINARY(MAX))),

LastEditedWhen -- ❌ Không mã hóa cột này

FROM Sales.InvoiceLines;

CLOSE SYMMETRIC KEY MySymmetricKey;

SELECT \* FROM New\_InvoiceLines5

-- Bước 6: Tạo bảng không Partitioned với mã hóa

CREATE TABLE dbo.New\_InvoiceLines\_NoPartition6 (

InvoiceLineID VARBINARY(MAX) NOT NULL,

LineProfit VARBINARY(MAX) NOT NULL,

ExtendedPrice VARBINARY(MAX) NOT NULL,

StockItemID VARBINARY(MAX) NOT NULL,

Quantity VARBINARY(MAX) NOT NULL,

LastEditedWhen DATE NOT NULL

);

OPEN SYMMETRIC KEY MySymmetricKey DECRYPTION BY CERTIFICATE MyCertificate;

INSERT INTO dbo.New\_InvoiceLines\_NoPartition6

SELECT

EncryptByKey(Key\_GUID('MySymmetricKey'), CAST(InvoiceLineID AS VARBINARY(MAX))),

EncryptByKey(Key\_GUID('MySymmetricKey'), CAST(LineProfit AS VARBINARY(MAX))),

EncryptByKey(Key\_GUID('MySymmetricKey'), CAST(ExtendedPrice AS VARBINARY(MAX))),

EncryptByKey(Key\_GUID('MySymmetricKey'), CAST(StockItemID AS VARBINARY(MAX))),

EncryptByKey(Key\_GUID('MySymmetricKey'), CAST(Quantity AS VARBINARY(MAX))),

LastEditedWhen

FROM Sales.InvoiceLines;

CLOSE SYMMETRIC KEY MySymmetricKey;

-- Bước 7: Tạo Clustered Columnstore Index (CCI)

CREATE CLUSTERED COLUMNSTORE INDEX CCI\_NewInvoiceLines ON dbo.New\_InvoiceLines5;

-- Bước 8: Định nghĩa hàm Dynamic Monthly Sales với giải mã

CREATE FUNCTION dbo.GetMonthlySales5 (@Year INT, @Month INT)

RETURNS TABLE

AS

RETURN

(

SELECT

SUM(CAST(DecryptByKey(CAST(ExtendedPrice AS VARBINARY(MAX))) AS DECIMAL(18,2))) AS MonthlyRevenue,

SUM(CAST(DecryptByKey(CAST(LineProfit AS VARBINARY(MAX))) AS DECIMAL(18,2))) AS MonthlyProfit

FROM dbo.New\_InvoiceLines5

WHERE LastEditedWhen >= DATEFROMPARTS(@Year, @Month, 1)

AND LastEditedWhen < DATEADD(MONTH, 1, DATEFROMPARTS(@Year, @Month, 1))

);

-- Bước 9: Tạo bảng log thời gian thực thi

CREATE TABLE dbo.ExecutionTimeLog5 (

QueryName NVARCHAR(100),

StartTime DATETIME,

EndTime DATETIME,

ElapsedTime INT,

CPUTime INT,

AdditionalInfo NVARCHAR(255),

CPUPercentage DECIMAL(5,2)

);

-- Bước 10: Ghi nhận thời gian thực thi, bao gồm cả thời gian giải mã

DECLARE @StartTime DATETIME, @EndTime DATETIME;

DECLARE @ElapsedTime BIGINT, @CPUTime BIGINT;

DECLARE @CPUPercentage DECIMAL(5,2);

-- Mở symmetric key

IF NOT EXISTS (SELECT \* FROM sys.symmetric\_keys WHERE name = 'AESKey')

BEGIN

CREATE SYMMETRIC KEY AESKey

WITH ALGORITHM = AES\_256

ENCRYPTION BY PASSWORD = 'YourStrongPassword';

END

OPEN SYMMETRIC KEY AESKey DECRYPTION BY PASSWORD = 'YourStrongPassword';

-- Bắt đầu đo thời gian

SET @StartTime = GETDATE();

SET STATISTICS TIME ON;

--SELECT \* FROM dbo.GetMonthlySales5(2015, 2); -- Gọi hàm đã sửa với giải mã

OPEN SYMMETRIC KEY MySymmetricKey DECRYPTION BY CERTIFICATE MyCertificate;

SELECT \* FROM dbo.GetMonthlySales5(2015, 2);

CLOSE SYMMETRIC KEY MySymmetricKey;

SET STATISTICS TIME OFF;

-- Kết thúc đo thời gian

SET @EndTime = GETDATE();

CLOSE SYMMETRIC KEY AESKey;

-- Tính thời gian thực thi

SET @ElapsedTime = DATEDIFF(MILLISECOND, @StartTime, @EndTime);

-- Lấy CPU time

SELECT @CPUTime = CAST(cpu\_time AS BIGINT)

FROM sys.dm\_exec\_requests

WHERE session\_id = @@SPID;

-- Tính % CPU sử dụng

SET @CPUPercentage = CASE

WHEN @ElapsedTime > 0 THEN (CAST(@CPUTime AS DECIMAL(10,2)) / @ElapsedTime) \* 100

ELSE 0

END;

-- Ghi log thời gian thực thi vào bảng

INSERT INTO dbo.ExecutionTimeLog5(QueryName, StartTime, EndTime, ElapsedTime, CPUTime, CPUPercentage, AdditionalInfo)

VALUES ('Partitioned Query with AES-256 Decryption', @StartTime, @EndTime, @ElapsedTime, @CPUTime, @CPUPercentage, 'Includes AES-256 Decryption');

-- Kiểm tra code không p dung kỹ thuật

-- Bước 10: Ghi nhận thời gian thực thi, bao gồm cả thời gian giải mã

DECLARE @StartTime DATETIME, @EndTime DATETIME;

DECLARE @ElapsedTime BIGINT, @CPUTime BIGINT;

DECLARE @CPUPercentage DECIMAL(5,2);

-- Mở symmetric key

IF NOT EXISTS (SELECT \* FROM sys.symmetric\_keys WHERE name = 'AESKey')

BEGIN

CREATE SYMMETRIC KEY AESKey

WITH ALGORITHM = AES\_256

ENCRYPTION BY PASSWORD = 'YourStrongPassword';

END

OPEN SYMMETRIC KEY AESKey DECRYPTION BY PASSWORD = 'YourStrongPassword';

-- Bắt đầu đo thời gian

SET @StartTime = GETDATE();

SET STATISTICS TIME ON;

OPEN SYMMETRIC KEY MySymmetricKey DECRYPTION BY CERTIFICATE MyCertificate;

SELECT

SUM(CAST(DecryptByKey(CAST(ExtendedPrice AS VARBINARY(MAX))) AS DECIMAL(18,2))) AS MonthlyRevenue,

SUM(CAST(DecryptByKey(CAST(LineProfit AS VARBINARY(MAX))) AS DECIMAL(18,2))) AS MonthlyProfit

FROM dbo.New\_InvoiceLines\_NoPartition6

WHERE LastEditedWhen >= DATEFROMPARTS(2015,2, 1)

AND LastEditedWhen < DATEADD(MONTH, 1, DATEFROMPARTS(2015,2, 1))

CLOSE SYMMETRIC KEY MySymmetricKey;

SET STATISTICS TIME OFF;

-- Kết thúc đo thời gian

SET @EndTime = GETDATE();

CLOSE SYMMETRIC KEY AESKey;

-- Tính thời gian thực thi

SET @ElapsedTime = DATEDIFF(MILLISECOND, @StartTime, @EndTime);

-- Lấy CPU time

SELECT @CPUTime = CAST(cpu\_time AS BIGINT)

FROM sys.dm\_exec\_requests

WHERE session\_id = @@SPID;

-- Tính % CPU sử dụng

SET @CPUPercentage = CASE

WHEN @ElapsedTime > 0 THEN (CAST(@CPUTime AS DECIMAL(10,2)) / @ElapsedTime) \* 100

ELSE 0

END;

-- Ghi log thời gian thực thi vào bảng

INSERT INTO dbo.ExecutionTimeLog5(QueryName, StartTime, EndTime, ElapsedTime, CPUTime, CPUPercentage, AdditionalInfo)

VALUES ('No Partitioned Query with AES-256 Decryption', @StartTime, @EndTime, @ElapsedTime, @CPUTime, @CPUPercentage, 'Includes AES-256 Decryption');

-- Kiểm tra log

SELECT \* FROM dbo.ExecutionTimeLog5;

-- Dọn dẹp bảng tạm

DROP TABLE #PartitionValues;

**Kết quả:**



Thời gian của bảng không sử dụng các kỹ thuật trên nhanh hơn.