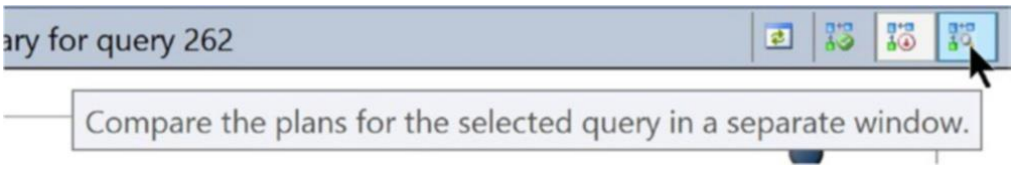
عنوان مقاله: : اشنایی با Query store | قسمت ششم
نویسنده مقاله: تیم فنی آموز
تاریخ انتشار: اسفند ۱۴۰۰
منبع: https://nikamooz.com/query-store-part۰۶

مقدمه

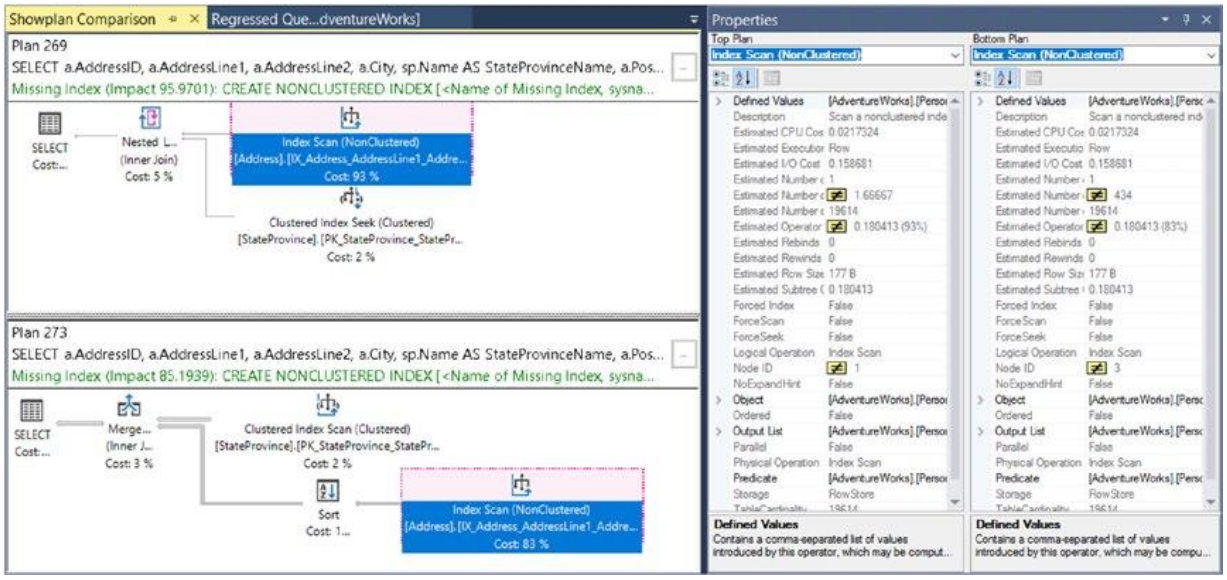
در قسمت پنجم از سلسله مقالات مرتبط با Query Store، در خصوص پلن های اجرایی صحبت کردیم و نکات مرتبط با آنها ارایه گردید . مسلما در بیزینس های مختلف و کوئری های متفاوت ساختار پلن ها عمدتا مشترک هست و اپراتور هایی که استفاده می شود در بیشتر مواقع یکسان هستند . به همین دلیل شناخت اپراتور ها شاید مشکل خاصی نداشته باشد . موضوع اصلی ارتباط بین این اپراتور ها و ورودی ها و خروجی هایی هست که این اپراتور ها با یکدیگر رد و بدل می کنند در این خصوص یکی از بهترین کتاب ها برای بررسی پلن های اجرایی کتاب SQL Server Execution Plans: Third Edition هست که در فصل های مختلف ، دید بسیار جامع و کلی از کار با اپراتور ها را به شما نمایش می دهد . همچنین یکی از بهترین ابزار ها برای بررسی پلن های اجرایی ، ابزاری به اسم SentryOne هست که در زمینه های مختلف ، نرم افزار های کاربردی ارایه کرده است . SentryOne ابزاری برای انالیز و تحلیل پلن های اجرایی هست که می توانید از جنبه های مختلف ، پلن ایجاد شده رو مورد بررسی قرار دهید.

در این مقاله، می خواهیم پلن های اجرایی را در Query Store دقیق تر بررسی کنیم و ببینیم با چه قابلیت هایی می توانیم به صورت خودکار، بهینه ترین پلن ها را همیشه اجرا کنیم . همچنین نگاهی عمیق تر به ساختار کوئری ها خواهیم داشت که در قالب داشبورد های تحلیلی به شما ارایه می شود . شاید نیاز داشته باشید که منطبق با سلیقه یا نیاز خود این داشبورد ها را سفارشی سازی کنید . پس باید بتوانیم Baseline ایی که در این سری از مقالات، بررسی شد را بسته به نیاز خودمان در هر بخش شبیه سازی کنیم و در اصلاح تحلیل هایی دقیق تر و جزیی تری بر روی این فرایند ها داشته باشیم .

در قسمت قبل، در گزارش های مرتبط با پسرفت کوئری که خدمت شما ارایه شد، متوجه شدید که چندین پلن اجرایی ممکن است برای یک کوئری در نظر گرفته شود . اما قابلیت بسیار خوبی که در این گزارش هست ، مقایسه پلن های اجرایی با یکدیگر هست. مطابق با عکسی که در شکل زیر مشاهده می کنید می توانید این مقایسه را انجام دهید:



پس از اعمال گزینه بالا ، می توانیم پلن ها را با یکدیگر مقایسه کنیم:



زمانی که دو پلن اجرایی با یکدیگر مقایسه می شوند ، دو قسمت مجزا به شما نمایش داده خواهد شد .در سمت چپ، اپراتور ها و در سمت راست ویژگی های مرتبط با هر کدام به شما نمایش داده می شود. زمانی که بر روی هر اپراتور کلیک کنیم ، قسمت منتخب با کاور صورتی شکل مطابق با عکس بالا نمایش داده می شود. در سمت راست در قسمت Properties ، در واقع اطلاعات کلی هر پلن قابل نمایش هست. عبارت هایی که ایکون زرد رنگ و نامساوی در کنار آن ها قرار دارند ، نشان دهنده این هست که این مقادیر در دو پلن با یکدیگر مساوی نیستند. این مقایسه ما بین ویژگی ها در دو پلن به ما نشان خواهد داد که مشکل دقیقا در چه بخشی هست. برای این که بتوانیم این مثال را عینا مشاهده کنیم می توانید کوئری های زیر را اجرا کنید:

```
CREATE OR ALTER PROC [dbo].[AddressByCity] @City NVARCHAR(۳۰)
AS
SELECT a.AddressID,
    a.AddressLine۱,
    a.AddressLine۲,
    a.City,
    sp.Name                AS StateProvinceName,
    a.PostalCode
```

```
FROM   Person.Address          AS a
    JOIN Person.StateProvince  AS sp
        ON  a.StateProvinceID = sp.StateProvinceID
WHERE  a.City = @City;
GO
```

در این قسمت بر اساس مقادیر مختلفی که در این جدول هستند می خواهیم Store proc را فرخوانی کنیم:

```
EXEC dbo.AddressByCity @City = N'London';
ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE;
EXEC dbo.AddressByCity @City = N'Mentor';
```

در این قسمت بر اساس مقادیر مختلفی که در این جدول هستند می خواهیم Store proc را فرخوانی کنیم با این تفاوت که در قسمت اول با یک پلن کوئری ما ساخته می شود و در قسمت دوم کلیه پلن ها به ازای دیتابیس مورد نظر حذف شده و بر اساس مقدار شهر 'N'Mentor پلن اجرایی ایجاد می شود . لطفا کلیه این کوئری را به ترتیب اجرا کنیدو در هر مرحله پلن های ان ها را با یکدیگر مقایسه کنید.

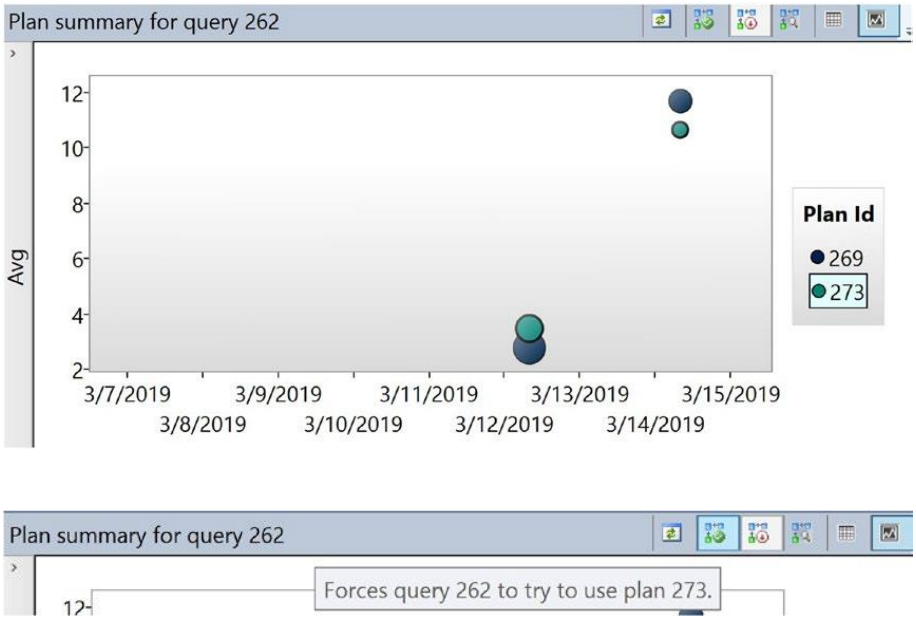
```
--Establish baseline behavior
EXEC dbo.AddressByCity @City = N'London';
GO ۱۰۰
--Remove the plan from cache
ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE;
--Compile a new plan
EXEC dbo.AddressByCity @City = N'Mentor';
GO
--Execute the code to show query regression
EXEC dbo.AddressByCity @City = N'London';
GO ۱۰۰
```

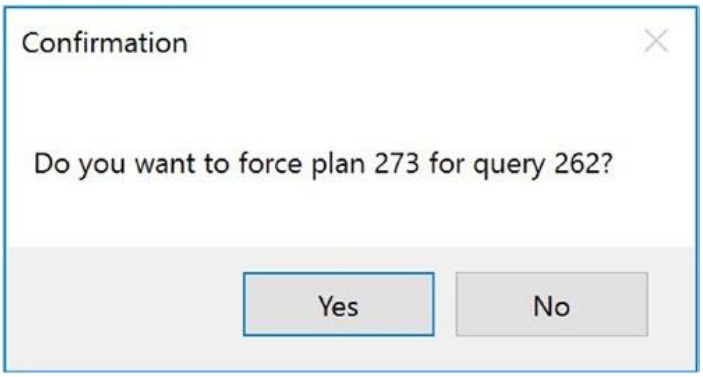
این مثال در قسمت قبل نیز بررسی شده بود. زمانی که قابلیت Query Store بر روی دیتابیس شما فعال باشد کلیه این موارد ذخیره شده و پلن های هر یک به شما نمایش داده می شود . سپس می توانید از طریق این قابلیت، دو پلن فوق را مورد بررسی قرار دهید . دقت داشته باشید در شکل اول اپراتور هایی مورد بررسی قرار گرفت که در دو پلن اجرایی شبیه یکدیگر بودند . عملا قابلیت مقایسه پلن های اجرایی زمانی می تواند مفید و موثر باشد که عمده اپراتور ها در دو پلن با یکدیگر یکسان باشند . در این خصوص می توانید بهترین خروجی ها را از مقایسه دو پلن دریافت کنید .

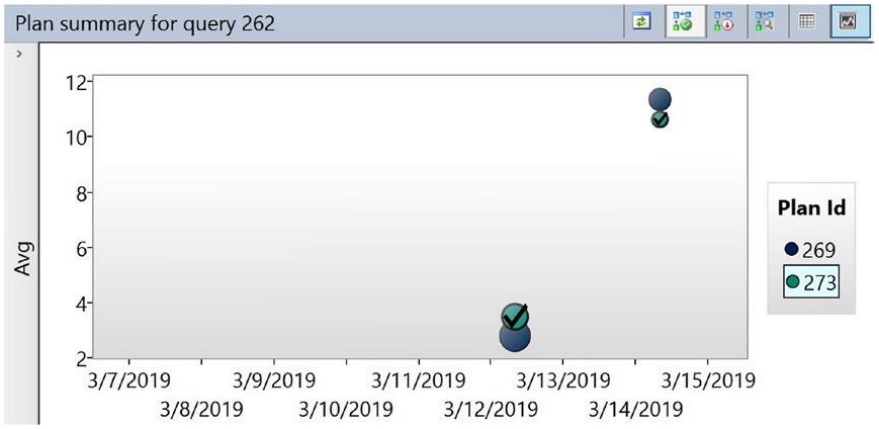
با توجه به سناریو ها و شرایطی که در مقاله های قبل بررسی شد ، مشاهده کردیم که مشکلاتی نظیر Parameter Sniffing باعث ایجاد اختلال در عملکرد سیستم خواهند شد . در این شرایط باید چه کاری انجام دهیم؟ چه تغییری در سیستم اعمال

شود که بتوانیم بهینه ترین پلن را پیدا کنیم؟ یکی از روش های پیشنهادی این کتاب، روش Force کردن این گونه پلن ها و مقایسه امار آنها نسبت به وضعیتی هست که از قبل تنظیم نشده بود . در واقع اعمال یک پلن و اجبار به تحلیل کوئری از روی این مسیر اجرایی که برای ان مشخص شده است ، بازخوردی که برای ما به همراه دارد از گزارشات تحلیلی Query Store به راحتی قابل ارزیابی هست. همچنین قابلیت دیگری که Query Store را در این شرایط نسبت به سایر روش های مشابه ممتاز می کند، قابلیت ردیابی پلن های Force شده هست که می توانیم انالیز دقیق بر روی عملکرد هر پلن داشته باشیم. شرایطی را در نظر بگیرید که سیستم عملیاتی را به شما تحویل داده اند و قرار هست کلیه این کوئری ها را رصد کنید. مطابق با Baseline مسلما به این نتیجه خواهید رسید که انتخاب بعضی از پلن ها در حین اجرا باعث می شود که مصرف منابع توسط برخی از کوئری های سنگین به شدت کاهش پیدا کند یا فرایند های روتین سیستم که شاید شامل گزارشات مختلف و متنوعی باشد را به راحتی از روی جداول عملیاتی تشخیص دهید. پیشنهاد می کنم که برای Force کردن پلن های اجرایی حتما سناریو بالا را در یک محیط تستی یا توسط نرم افزار هایی مانند Query Stress به مراتب و در تعداد درخواست های بالا اجرا کنید و میزان Wait ها و منابع سرور را دقیق تر بررسی نمایید. متوجه خواهید شد، زمانی که سیستم های عملیاتی زیر بار درخواست ها و گزارشات سنگین هستند عملا عدم نادیده گرفتن همین موارد و مشکلاتی جزیی به شدت کارایی سیستم را زیر سوال خواهد برد.

در Query Store این کار به راحتی قابل انجام هست. مطابق با شکل پایین می توانید پلن های منتخب را Force کنید که در کوئری های مرتبط با آنها اجرا شوند :







مراحل بالا به ترتیب برای این منظور استفاده می شود. زمانی که پلن اجرایی شما انتخاب شده باشد تیک مرتبط با پلن ایدی مد نظر در باکس مرتبط نمایش داده می شود. البته این کار را نیز می توانید با استفاده از اسکریپت زیر ، انجام دهید و پلن مد نظر را مجبور به استفاده کنید:

```
EXEC sys.sp_query_store_force_plan ۲۶۲,۲۷۳;
```

همچنین زمانی که می خواهید شرایط را به حالت اول برگردانید می توانید از کوئری زیر استفاده کنید:

```
EXEC sys.sp_query_store_unforce_plan ۲۶۲,۲۷۳;
```

می توانید امار پلن هایی که بین شکل انتخاب شده است را با استفاده از اسکریپت زیر از سیستم استخراج کنید:

```
SELECT qsq.query_id,
    qsp.plan_id,
    qsp.is_forced_plan
    Chapter ۷ ForCing          plans۱۶۶
FROM   sys.query_store_query     AS qsq
```

```
    JOIN sys.query_store_plan AS qsp
        ON qsp.query_id = qsq.query_id
WHERE qsq.query_id = ۲۶۲;
```

	query_id	plan_id	is_forced_plan
1	262	269	0
2	262	273	1

StatementSqlHandle	0x09005BEBBF981
Use plan	True
WaitStats	

زمانی که کوئری بالا اجرا می شود خروجی اول به ما کلیات مرتبط با پلن های Force شده را نمایش می دهد . همچنین در عکس دوم زمانی که کوئری را بررسی کنیم پارامتری وجود دارد که متوجه خواهیم شد از یک پلن Force شده استفاده شه است یا خیر.

تا این جا در خصوص قابلیت های ارزشمند Query Store مطالب مختلفی خدمت شما ارائه شد. اما یکی از قابلیت های بی نظیری که در این قسمت قرار هست معرفی کنیم ، در خصوص بحث بهینه سازی خودکار پلن اجرایی هست. در واقع کلیه مواردی که در این مقاله توضیح داده شد را قرار هست SQL SERVER برای ما به صورت هوشمند انجام بدهد و بر اساس افت عملکرد کوئری ها بهترین تصمیم را اتخاذ کند. این قابلیت تحت عنوان Automatic Plan Correction شناخته می شود . کلیه این فرایند ها تحت شرایطی انجام می شود که پلن های ایجاد شده طی الگوریتم هایی که برای آن نوشته شده است تشخیص داده شود که کارایی آن در حال حاضر اصلا به نفع سیستم نیست و می تواند جایگزین خوبی داشته باشد. برای این که جزییات این موضوع را دقیق تر بررسی کنیم می خواهیم دیتابیس ایی داشته باشیم که شامل رکورد های بیشتری نسبت به دیتابیس های معمولی هست. بدین منظور می توانید از اسکریپت های زیر جهت درج اطلاعات بیشتر در دیتابیس Adventurework استفاده کنید. لطفا اسکریپت های زیر را اجرا کنید:

```
/*
 * ایجاد جدول کالاها با رکورد های بیشتر
 */

USE AdventureWorks۲۰۱۹
GO
```

```
SELECT p.ProductID + (a.number * ۱۰۰۰) AS ProductID,
    p.Name + CONVERT(VARCHAR, (a.number * ۱۰۰۰)) AS NAME,
    p.ProductNumber + '-' + CONVERT(VARCHAR, (a.number * ۱۰۰۰)) AS ProductNumber,
    p.MakeFlag,
    p.FinishedGoodsFlag,
    p.Color,
    p.SafetyStockLevel,
    p.ReorderPoint,
    p.StandardCost,
    p.ListPrice,
    p.Size,
    p.SizeUnitMeasureCode,
    p.WeightUnitMeasureCode,
    p.Weight,
    p.DaysToManufacture,
    p.ProductLine,
    p.Class,
    p.Style,
    p.ProductSubcategoryID,
    p.ProductModelID,
    p.SellStartDate,
    p.SellEndDate,
    p.DiscontinuedDate
INTO   bigProduct
FROM   Production.Product AS p
    CROSS JOIN MASTER..spt_values AS a
WHERE  a.type = 'p'
    AND a.number BETWEEN ۱ AND ۵۰
GO


ALTER TABLE bigProduct
ALTER COLUMN ProductId INT NOT NULL
GO

ALTER TABLE bigProduct
ADD CONSTRAINT pk_bigProduct PRIMARY KEY(ProductId)
GO
```

```
SELECT ROW_NUMBER() OVER(
        ORDER BY
        x.TransactionDate,
        (
           SELECT NEWID()
        )
    ) AS TransactionID,
    p۱.ProductID,
    x.TransactionDate,
    x.Quantity,
    CONVERT(
       MONEY,
       p۱.ListPrice * x.Quantity * RAND(CHECKSUM(NEWID())) * ۲
    ) AS ActualCost
INTO   bigTransactionHistory
FROM   (
        SELECT p.ProductID,
            p.ListPrice,
            CASE
                WHEN p.productid % ۲۶ = ۰ THEN ۲۶
                WHEN p.productid % ۲۵ = ۰ THEN ۲۵
                WHEN p.productid % ۲۴ = ۰ THEN ۲۴
                WHEN p.productid % ۲۳ = ۰ THEN ۲۳
                WHEN p.productid % ۲۲ = ۰ THEN ۲۲
                WHEN p.productid % ۲۱ = ۰ THEN ۲۱
                WHEN p.productid % ۲۰ = ۰ THEN ۲۰
                WHEN p.productid % ۱۹ = ۰ THEN ۱۹
                WHEN p.productid % ۱۸ = ۰ THEN ۱۸
                WHEN p.productid % ۱۷ = ۰ THEN ۱۷
                WHEN p.productid % ۱۶ = ۰ THEN ۱۶
                WHEN p.productid % ۱۵ = ۰ THEN ۱۵
                WHEN p.productid % ۱۴ = ۰ THEN ۱۴
                WHEN p.productid % ۱۳ = ۰ THEN ۱۳
                WHEN p.productid % ۱۲ = ۰ THEN ۱۲
                WHEN p.productid % ۱۱ = ۰ THEN ۱۱
                WHEN p.productid % ۱۰ = ۰ THEN ۱۰
                WHEN p.productid % ۹ = ۰ THEN ۹
                WHEN p.productid % ۸ = ۰ THEN ۸
                WHEN p.productid % ۷ = ۰ THEN ۷
```

```
                WHEN p.productid % ۶ = ۰ THEN ۶
                WHEN p.productid % ۵ = ۰ THEN ۵
                WHEN p.productid % ۴ = ۰ THEN ۴
                WHEN p.productid % ۳ = ۰ THEN ۳
                WHEN p.productid % ۲ = ۰ THEN ۲
                ELSE ۱
            END        AS ProductGroup
        FROM   bigproduct     p
    ) AS p۱
    CROSS APPLY
(
   SELECT transactionDate,
        CONVERT(INT, (RAND(CHECKSUM(NEWID())) * ۱۰۰) + ۱) AS Quantity
   FROM   (
            SELECT DATEADD(dd, number, '۲۰۰۵۰۱۰۱') AS transactionDate,
                NTILE(p۱.ProductGroup) OVER(ORDER BY number) AS groupRange
            FROM   MASTER..spt_values
            WHERE  TYPE = 'p'
        ) AS z
   WHERE  z.groupRange % ۲ = ۱
) AS x

ALTER TABLE bigTransactionHistory
ALTER COLUMN TransactionID INT NOT NULL
GO

ALTER TABLE bigTransactionHistory
ADD CONSTRAINT pk_bigTransactionHistory PRIMARY KEY(TransactionID)
GO

CREATE NONCLUSTERED INDEX IX_ProductId_TransactionDate
ON bigTransactionHistory(ProductId, TransactionDate)
INCLUDE(Quantity, ActualCost)
GO
```

دقت داشته باشید که اسکریپت های فوق بر روی دیتابیس AdventureWorks۲۰۱۹ اجرا شده است . لذا در صورتی که از نسخه های پایین تر استفاده می کنید حتما دقت داشته باشید که دیتابیس مورد نظر به درستی انتخاب شده باشد. پس از اجرای اسکریپت های بالا ، حالا باید رویه ایی ایجاد کنیم:

```
CREATE OR
ALTER PROCEDURE dbo.ProductByCost (@ActualCost MONEY)
AS
        SELECT bth.ActualCost
        FROM   dbo.bigTransactionHistory  AS bth
            JOIN dbo.bigProduct       AS p
                ON  p.ProductID = bth.ProductID
        WHERE  bth.ActualCost = @ActualCost;
GO

--ensuring that Query Store is on and has a clean data set
ALTER DATABASE AdventureWorks۲۰۱۹
SET QUERY_STORE = ON;
ALTER DATABASE AdventureWorks۲۰۱۹
SET QUERY_STORE CLEAR;
GO
```

حالا می خواهیم شرایطی را شبیه سازی کنیم که در آن افت عملکرد در پلن های اجرایی را مشاهده کنیم:

```
 /*
  * قسمت اول
  */
EXEC dbo.ProductByCost @ActualCost = ۸.۲۲۰۵;
GO ۳۰

 /*
  * قسمت دوم
  */
DECLARE @PlanHandle VARBINARY(۶۴);
SELECT @PlanHandle = deps.plan_handle
FROM   sys.dm_exec_procedure_stats AS deps
WHERE  deps.object_id = OBJECT_ID('dbo.ProductByCost');
IF @PlanHandle IS NOT NULL
BEGIN
   DBCC FREEPROCCACHE(@PlanHandle);
END
GO
```

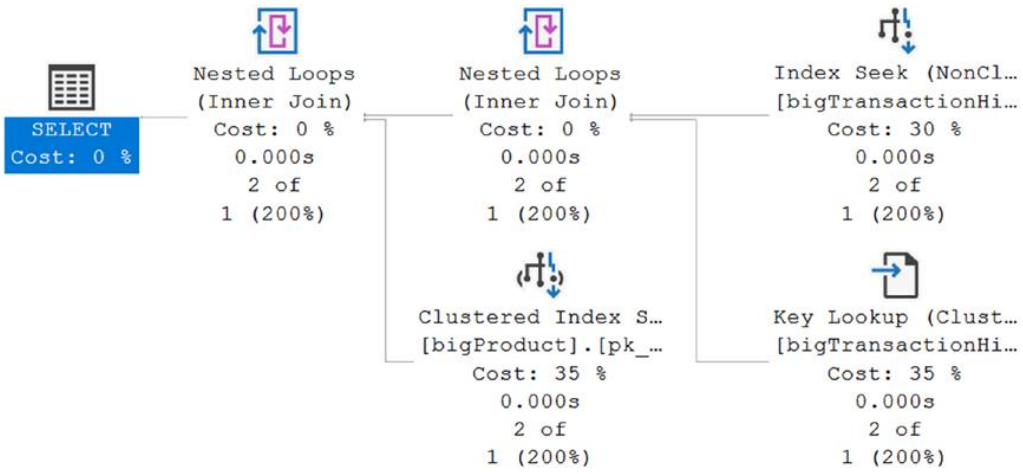
```
/*
 * قسمت سوم
 */
EXEC dbo.ProductByCost @ActualCost = ۰.۰;
GO

/*
 * قسمت چهارم
 */
EXEC dbo.ProductByCost @ActualCost = ۸.۲۲۰۵;
GO ۱۵
```

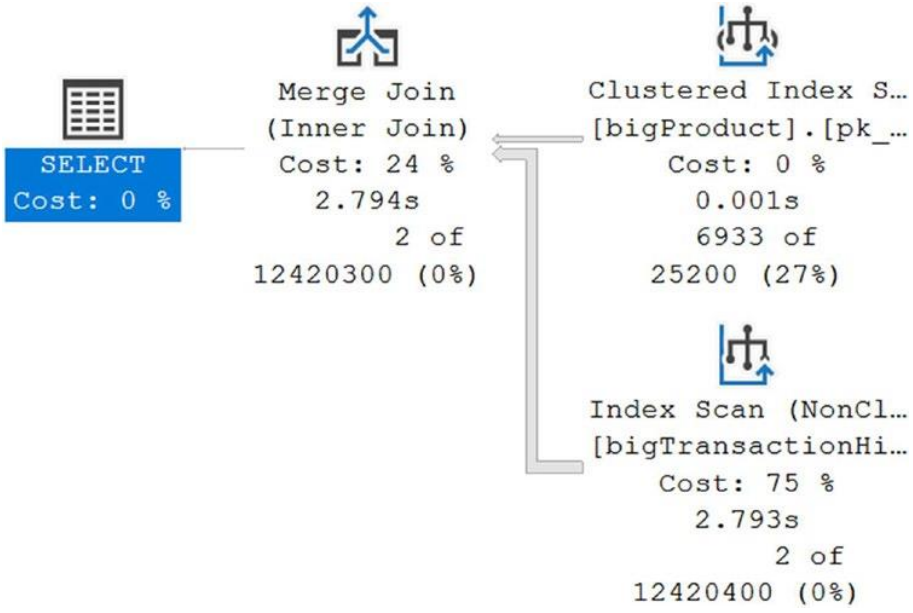
همان طور که در کوئری بالا مشاهده می کنید ، ۴ بخش نوشته شده است که هر بخش برای هدف خاصی در نظر گرفته شده . در بخش اول با اجرای رویه مورد نظر میخواهیم یک Baseline از کوئری مورد نظر ایجاد کنیم و در سیستم داشته باشیم . در بخش دوم کلیه پلن های ایجاد شده را از Plan Cache حذف می کنیم که از این به بعد کوئری های اجرا شده با پلن جدید ، ایجاد شوند. حتما بخش دوم را اجرا کنید، چرا که در صورت عدم اجرا، پلن های قبلی در حافظه باقی می ماند و عملکرد Automatic Plan Correction را نمی توانیم به خوبی بررسی کنیم .در قسمت سوم، رویه را با یک مقدار دیگری اجرا می کنیم که توزیع اطلاعات برای این مقدار با مقدار قسمت اول مقداری متفاوت است .در قسمت چهارم نیز این رویه با مقدار دیگری به تعداد ۱۵ مرتبه اجرا شده است که در خروجی مشاهده خواهید کرد.در نهایت اسکریپت پایین را اجرا کنید :

```
SELECT DDTR.TYPE
    ,DDTR.REASON
    ,DDTR.LAST_REFRESH
    ,DDTR.STATE
    ,DDTR.SCORE
    ,DDTR.DETAILS
FROM   SYS.DM_DB_TUNING_RECOMMENDATIONS AS DDTR;
```

در قسمت اول شاهد این پلن اجرایی هستیم:



و در قسمت دوم زمانی که با مقدار ;۰.۰ = @ActualCost رویه را اجرا کردیم شاهد پلن زیر هستیم:



دقت داشته باشید زمانی که قسمت سوم را اجرا می کنید، در حدود ۱۲ میلیون رکورد، دیتا وارد پلن اجرایی می شود. لذا در اینجا عملکرد اپراتور ها باید نسبت به ظرفیت داده های ورودی، با ظرفیت بالاتر و اپراتور های سنگین تر عملیاتی شود. دلیل این موضوع هم بدین خاطر هست که به نسبت الگوریتم هایی که در پشت هر کدام از این بخش ها نوشته شده است ، هر اپراتور بسته به اطلاعاتی ورودی مختلف ، عملکرد متفاوتی را از خود نشان خواهد داد. به عنوا مثال زمانی که از Merge Join در پلن های اجرایی استفاده شود نشان دهنده این موضوع هست که دو طرف ورودی به این اپراتور باید اطلاعات مرتب شده ایی باشند که منطبق با الگوریتم Merge Join پردازش بر روی ان انجام شود. در فصل دوم و سوم کتابی که در اول مقاله خدمت شما معرفی شد کلیه این جزییات مطرح شده است و با الگوریتم های مختلفی که پشت سر هر اپراتور هست اشنا خواهید شد.

زمانی که خروجی اسکریپت بالا را ملاحظه کنید بدین شکل پیغامی به شما نمایش داده می شود:

Average query CPU time changed from ۰.۱۶ms to ۴۹۰۹.۴۱ms

پیامی که در بالا مشاهده می کنید توسط این DMV به ما نمایش داده شده. مسلما تغییر پلن در این حالت اصلا به نفع سیستم نیست که بخواهیم این پلن اجرایی را بر روی کلیه کوئری ها با مقادیر مختلف Force کنیم. همچنین می توانید با استفاده از اسکریپت زیر در صورتی که در این DMV پیغامی را مشاهده کردید جزییات مختلف با متن آنرا مشاهده کنید. خروجی این قسمت به شکل Json هست و اطلاعات خوبی در اختیار شما قرار خواهد داد.

```
WITH DbTuneRec
    AS (
       SELECT ddtr.reason,
              ddtr.score,
              pfd.query_id,
              pfd.regressedPlanId,
              pfd.recommendedPlanId,
              JSON_VALUE(ddtr.state, '$.currentValue') AS CurrentState,
              JSON_VALUE(ddtr.state, '$.reason') AS CurrentStateReason,
              JSON_VALUE(ddtr.details, '$.implementationDetails.script') AS ImplementationScript
       FROM   sys.dm_db_tuning_recommendations AS ddtr
              CROSS APPLY
       OPENJSON(ddtr.details, '$.planForceDetails')
       WITH (
          query_id INT '$.queryId',
          regressedPlanId INT '$.regressedPlanId',
          recommendedPlanId INT '$.recommendedPlanId'
       ) AS pfd
    )
    WITH DbTuneRec
    AS (
       SELECT ddtr.reason,
              ddtr.score,
              pfd.query_id,
              pfd.regressedPlanId,
              pfd.recommendedPlanId,
              JSON_VALUE(ddtr.state, '$.currentValue') AS CurrentState,
              JSON_VALUE(ddtr.state, '$.reason') AS CurrentStateReason,
              JSON_VALUE(ddtr.details, '$.implementationDetails.script') AS ImplementationScript
       FROM   sys.dm_db_tuning_recommendations AS ddtr
              CROSS APPLY
```

```
    OPENJSON(ddtr.details, '$.planForceDetails')
    WITH (
        query_id INT '$.queryId',
        regressedPlanId INT '$.regressedPlanId',
        recommendedPlanId INT '$.recommendedPlanId'
    ) AS pfd
)
```

همچنین با استفاده از اسکریپت های زیر می توانید این قابلیت را با استفاده از دستورات T-sql فعال یا عیر فعال کنید:

```
ALTER DATABASE AdventureWorks۲۰۱۹ SET AUTOMATIC_TUNING (FORCE_LAST_GOOD_PLAN = ON);
ALTER DATABASE AdventureWorks۲۰۱۹ SET AUTOMATIC_TUNING (FORCE_LAST_GOOD_PLAN = OFF);
```

این قابلیت که با دستورات T-sql بتوانیم این کار را انجام دهیم از نسخه ۲۰۱۷ به بعد شاهد هستیم. پیشنهاد کتاب بدین شکل هست که اجازه دهید این قابلیت بر روی دیتابیس های شما فعال باشد که فرایند بهینه سازی خودکار انجام شود. همچنین جهت تست مجدد بر روی این قابلیت می توانید از اسکریپت زیر استفاده کنید:

```
ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE;
GO
ALTER DATABASE AdventureWorks۲۰۱۹
SET QUERY_STORE CLEAR;
GO
EXEC dbo.ProductByCost @ActualCost = ۸.۲۲۰۵;
GO ۳۰

DECLARE @PlanHandle VARBINARY(۶۴);
SELECT @PlanHandle = deps.plan_handle
FROM   sys.dm_exec_procedure_stats AS deps
WHERE  deps.object_id = OBJECT_ID('dbo.ProductByCost');
IF @PlanHandle IS NOT NULL
BEGIN
   DBCC FREEPROCCACHE(@PlanHandle);
END
GO


EXEC dbo.ProductByCost @ActualCost = ۰.۰;
GO
```

```
EXEC dbo.ProductByCost @ActualCost = ۸.۲۲۰۵;
GO ۱۵
```

در این قسمت ، اسکریپت هایی که در زمینه Query Store و جداول مرتبط می توانید استفاده کنید را معرفی خواهیم کرد که انعطاف گزارش سازی های مرتبط با این موضوع را بالاتر ببرید. با توجه به کلیه بخش هایی که به تفکیک در مقالات قبلی به ان اشاره شد می توانید از طریق کوئری زیر امار مرتبط با پلن های اجرایی، متن کوئری ها و اطلاعات زمان اجرا را علاوه بر داشبورد های تحلیلی Query Store به تفکیک داشته باشید. گاهی نیاز هست که تحلیل ها در سایر نرم افزار ها ها انجام شود . به همین خاطر اسکریپت زیر به شما کمک خواهد کرد که اطلاعات زمان اجرا را بصری سازی کنید.

```
DECLARE @HoursBack smallint = ۸;
DECLARE @StartDate datetime۲ = DATEADD(hour, -@HoursBack, GETUTCDATE());

WITH QueryRuntimeStats AS (
  SELECT
      p.plan_id
     ,q.query_id
     ,q.query_hash
     ,SUM(rs.count_executions) AS total_executions
     ,SUM(rs.count_executions * rs.avg_duration) / ۱۰۰۰ AS total_duration_ms
     ,SUM(rs.count_executions * rs.avg_cpu_time) / ۱۰۰۰ AS total_cpu_ms
     ,SUM(rs.count_executions * rs.avg_clr_time) / ۱۰۰۰ AS total_clr_time_ms
     ,SUM(rs.count_executions * rs.avg_query_max_used_memory) AS total_query_max_used_memory
     ,SUM(rs.count_executions * rs.avg_logical_io_reads) AS total_logi_reads
     ,SUM(rs.count_executions * rs.avg_logical_io_writes) AS total_logi_writes
     ,SUM(rs.count_executions * rs.avg_physical_io_reads) AS total_phys_reads
     ,SUM(rs.count_executions * rs.avg_rowcount) AS total_rowcount
     ,SUM(rs.count_executions * rs.avg_log_bytes_used) AS total_log_bytes_used
     ,SUM(rs.count_executions * rs.avg_tempdb_space_used) AS total_tempdb_space_used
  from sys.query_store_plan p
  join sys.query_store_query q
    on q.query_id = p.query_id
  join sys.query_store_runtime_stats rs
    on rs.plan_id = p.plan_id
  join sys.query_store_runtime_stats_interval rsi
    on rsi.runtime_stats_interval_id = rs.runtime_stats_interval_id
  where rsi.start_time > @StartDate
  group by
```

```
        p.plan_id
       ,q.query_id
       ,q.query_hash
)
,QueryWaitStats AS (
   SELECT
        p.plan_id
       ,q.query_id
       ,q.query_hash
       ,ws.wait_category_desc
       ,SUM(ws.total_query_wait_time_ms) AS total_wait_time_ms
   FROM sys.query_store_plan p
   JOIN sys.query_store_query q
    ON q.query_id = p.query_id
   JOIN sys.query_store_wait_stats ws
    ON ws.plan_id = p.plan_id
   JOIN sys.query_store_runtime_stats_interval rsi
    ON rsi.runtime_stats_interval_id = ws.runtime_stats_interval_id
   WHERE rsi.start_time > @StartDate
   GROUP BY
        p.plan_id
       ,q.query_id
       ,q.query_hash
       ,ws.wait_category_desc
)
,QueryWaitStatsByCategory
AS
(
   SELECT *
   FROM QueryWaitStats
   PIVOT
   (
      SUM(total_wait_time_ms)
      FOR wait_category_desc IN
      (
          [Unknown]
         ,[CPU]
         ,[Worker Thread]
         ,[Lock]
         ,[Latch]
```

```
            ,[Buffer Latch]
            ,[Buffer IO]
            ,[Compilation]
            ,[SQL CLR]
            ,[Mirroring]
            ,[Transaction]
            ,[Idle]
            ,[Preemptive]
            ,[Service Broker]
            ,[Tran Log IO]
            ,[Network IO]
            ,[Parallelism]
            ,[Memory]
            ,[User Wait]
            ,[Tracing]
            ,[Full Text Search]
            ,[Other Disk IO]
            ,[Replication]
            ,[Log Rate Governor]
        )
    ) AS pvt
)
,QueryWaitStatsTotals
AS
(
    SELECT
        plan_id
        ,query_id
        ,query_hash
        ,SUM(total_wait_time_ms) AS total_wait_time_ms
    FROM QueryWaitStats
    GROUP BY
        plan_id
        ,query_id
        ,query_hash
)
SELECT
    rs.plan_id
    , rs.query_id
    , rs.query_hash
```

```
    , rs.total_executions
    , rs.total_duration_ms
    , rs.total_cpu_ms
    , rs.total_clr_time_ms
    , rs.total_query_max_used_memory
    , rs.total_logi_reads
    , rs.total_logi_writes
    , rs.total_phys_reads
    , rs.total_rowcount
    , rs.total_log_bytes_used
    , rs.total_tempdb_space_used
    , ws.total_wait_time_ms
    , wsc.[CPU]
    , wsc.[Lock]
    , wsc.[Latch]
    , wsc.[Buffer Latch]
    , wsc.[Buffer IO]
    , wsc.[Memory]
    , wsc.[Tran Log IO]
    , wsc.[Network IO]
    , wsc.[Worker Thread]
    , wsc.[Unknown]
    , wsc.[Compilation]
    , wsc.[SQL CLR]
    , wsc.[Mirroring]
    , wsc.[Transaction]
    , wsc.[Idle]
    , wsc.[Preemptive]
    , wsc.[Service Broker]
    , wsc.[Parallelism]
    , wsc.[User Wait]
    , wsc.[Tracing]
    , wsc.[Full Text Search]
    , wsc.[Other Disk IO]
    , wsc.[Replication]
    , wsc.[Log Rate Governor]
FROM QueryRuntimeStats rs
LEFT JOIN QueryWaitStatsTotals AS ws
  ON rs.plan_id = ws.plan_id
 AND rs.query_id = ws.query_id
```

```
LEFT JOIN QueryWaitStatsByCategory AS wsc
  ON rs.plan_id = wsc.plan_id
 AND rs.query_id = wsc.query_id;
```

با استفاده از اسکریپت پایین می توانید کلیه امار ثبت شده در Query Store را مشاهده کنید. با توجه به حجم رکورد هایی که ثبت شده است ممکن است این کوئری زمان زیادی را برای ا جرا نیاز داشته باشد. به طور معمول در صورتی که قصد دارید Baseline سفارشی برای خود درست کنید می توانید از این کوئری استفاده کنید:

```
SELECT *
FROM   sys.query_store_query
    LEFT JOIN sys.query_store_query_text
      ON  query_store_query.query_text_id = query_store_query_text.query_text_id
    LEFT JOIN sys.query_store_plan
      ON  query_store_query.query_id = query_store_plan.query_id
    LEFT JOIN sys.query_store_runtime_stats
      ON  query_store_plan.plan_id = query_store_runtime_stats.plan_id
    LEFT JOIN sys.query_store_runtime_stats_interval
      ON  query_store_runtime_stats.runtime_stats_interval_id =
        query_store_runtime_stats_interval.runtime_stats_interval_id
```

با استفاده از اسکریپت زیر می توانیم در یک بازه زمانی مورد نظر امار مرتبط با زمان اجرای هر کوئری را مشاهده کنیم. در صورتی که قصد داشتید از رویه های سفارشی استفاده کنید می توانید از کوئری زیر استفاده کنید . همان طور که مشاهده کردید در داشبورد های تحلیلی Query Store نیز با فیلتر کردن بر روی هر بازه ایی که مد نظر داشتیم ، اطلاعات فوق را به صورت کلی مشاهده می کردیم . همین کار را نیز می توانید به روش دیگری انجام دهید :

```
SELECT TOP ۵۰
    query_store_query_text.query_sql_text,
    CAST(query_store_plan.query_plan AS XML) AS query_plan_xml,
    query_store_runtime_stats.first_execution_time,
    query_store_runtime_stats.last_execution_time,
    query_store_runtime_stats.count_executions,
    query_store_runtime_stats.avg_duration AS avg_duration_microseconds,
    query_store_runtime_stats.last_duration AS last_duration_microseconds,
    query_store_runtime_stats.avg_cpu_time AS avg_cpu_time_microseconds,
    query_store_runtime_stats.last_cpu_time AS last_cpu_time_microseconds,
    query_store_runtime_stats.avg_logical_io_reads,
    query_store_runtime_stats.last_logical_io_reads,
    query_store_runtime_stats.avg_query_max_used_memory AS avg_query_max_used_memory_۸k_pages,
    query_store_runtime_stats.last_query_max_used_memory AS last_query_max_used_memory_۸k_pages,
```

```
    query_store_runtime_stats.avg_rowcount,
    query_store_runtime_stats.last_rowcount,
    query_store_runtime_stats_interval.start_time AS interval_start_time,
    query_store_runtime_stats_interval.end_time AS interval_end_time,
    query_store_query.query_id,
    query_store_query_text.query_text_id,
    query_store_plan.plan_id
FROM   sys.query_store_query
    LEFT JOIN sys.query_store_query_text
        ON  query_store_query.query_text_id = query_store_query_text.query_text_id
    LEFT JOIN sys.query_store_plan
        ON  query_store_query.query_id = query_store_plan.query_id
    LEFT JOIN sys.query_store_runtime_stats
        ON  query_store_plan.plan_id = query_store_runtime_stats.plan_id
    LEFT JOIN sys.query_store_runtime_stats_interval
        ON  query_store_runtime_stats.runtime_stats_interval_id =
          query_store_runtime_stats_interval.runtime_stats_interval_id
WHERE  query_store_runtime_stats_interval.start_time BETWEEN '۱۱/۱۷/۲۰۱۵ ۲۲:۰۰:۰۰' AND '۱۱/۱۷/۲۰۱۵ ۲۳:۰۰:۰۰'
ORDER BY
    query_store_runtime_stats.avg_cpu_time DESC
     ORDER BY query_store_runtime_stats.avg_duration DESC
     ORDER BY query_store_runtime_stats.count_executions DESC
     ORDER BY query_store_runtime_stats.avg_logical_io_reads DESC
     ORDER BY query_store_runtime_stats.avg_rowcount DESC
```

با استفاده از کوئری زیر می توانید وضعیت یک جدول خاص را مورد بررسی قرار دهید. همچنین زمان اجرا بسته به بازه ایی که در نظر دارید را می توانید مشخص کنید. از این کوئری نیز میتوانید به عنوان یک رویه ایی که صرفا با یک متن خاص اجرا می شود به خوبی استفاده کنید:

```
SELECT TOP ۵۰
        query_store_query.query_id,
        query_store_query_text.query_sql_text,
        CAST(query_store_plan.query_plan AS XML) AS query_plan_xml,
        query_store_runtime_stats.first_execution_time,
        query_store_runtime_stats.last_execution_time,
        query_store_runtime_stats.count_executions,
        query_store_runtime_stats.avg_duration AS avg_duration_microseconds,
        query_store_runtime_stats.last_duration AS last_duration_microseconds,
        query_store_runtime_stats.avg_cpu_time AS avg_cpu_time_microseconds,
        query_store_runtime_stats.last_cpu_time AS last_cpu_time_microseconds,
```

```
        query_store_runtime_stats.avg_logical_io_reads,
        query_store_runtime_stats.last_logical_io_reads,
        query_store_runtime_stats.avg_query_max_used_memory AS avg_query_max_used_memory_8k_pages,
        query_store_runtime_stats.last_query_max_used_memory AS last_query_max_used_memory_8k_pages,
        query_store_runtime_stats.avg_rowcount,
        query_store_runtime_stats.last_rowcount
FROM sys.query_store_query
LEFT JOIN sys.query_store_query_text
ON query_store_query.query_text_id = query_store_query_text.query_text_id
LEFT JOIN sys.query_store_plan
ON query_store_query.query_id = query_store_plan.query_id
LEFT JOIN sys.query_store_runtime_stats
ON query_store_plan.plan_id = query_store_runtime_stats.plan_id
LEFT JOIN sys.query_store_runtime_stats_interval
ON query_store_runtime_stats.runtime_stats_interval_id = query_store_runtime_stats_interval.runtime_stats_interval_id
WHERE query_store_runtime_stats_interval.start_time BETWEEN '۱۱/۱۷/۲۰۱۵ ۲۲:۰۰:۰۰' AND '۱۱/۱۸/۲۰۱۵ ۲۲:۰۰:۰۰'
AND query_store_query_text.query_sql_text LIKE '%SELECT * FROM Person.Person%'
ORDER BY query_store_runtime_stats.last_execution_time DESC
```

با استفاده از اسکریپت زیر می توانید ، کوئری هایی که از یک ایندکس خاص استفاده می کنند را به راحتی شناسایی کنید . در بعضی از سناریو ها و دیتابیس های عملیاتی شاهد هستیم که به دلایلی ، در سمت کد برنامه، ایندکس هایی استفاده شده است که حتما باید پلن های اجرایی باید مطابق با ان اجرا شود . لذا این روش به شما این امکان را می دهد که در صورتی که نیاز داشتید ایندکس های مختلف را جهت این موضوع یا دلایل دیگر مورد بررسی قرار دهید، به شکل زیر عمل کنید:

```
SELECT TOP ۵۰
        query_store_query.query_id,
        query_store_query_text.query_sql_text,
        CAST(query_store_plan.query_plan AS XML) AS query_plan_xml,
        query_store_runtime_stats.avg_logical_io_reads
FROM sys.query_store_query
LEFT JOIN sys.query_store_query_text
ON query_store_query.query_text_id = query_store_query_text.query_text_id
LEFT JOIN sys.query_store_plan
ON query_store_query.query_id = query_store_plan.query_id
LEFT JOIN sys.query_store_runtime_stats
ON query_store_plan.plan_id = query_store_runtime_stats.plan_id
LEFT JOIN sys.query_store_runtime_stats_interval
ON query_store_runtime_stats.runtime_stats_interval_id = query_store_runtime_stats_interval.runtime_stats_interval_id
```

```
WHERE query_store_runtime_stats_interval.start_time BETWEEN '۱۱/۱۷/۲۰۱۵ ۲۲:۰۰:۰۰' AND '۱۱/۱۸/۲۰۱۵ ۲۲:۰۰:۰۰'
AND query_store_plan.query_plan LIKE '%IX_Person_LastName_FirstName_MiddleName%'
ORDER BY query_store_runtime_stats.avg_logical_io_reads DESC
```

با استفاده از اسکریپت زیر می توانیم پلن های اجرایی که مشکل مغایرت در نوع دیتا تایپ ها را دارند را به راحتی شناسایی و استخراج کنیم . عبارت convert_implicit در این قسمت نشان دهنده همین مشکل هست که در قسمت های قبلی در این خصوص توضیحات لازم ارایه شد که ممکن است پیغام هایی بر روی پلن هایی اجرایی دریافت کنیم که عملکرد پلن ها دستخوش تغییر شود . لذا این موارد را می توانیم به این روش شناسایی و برطرف کرد

```
SELECT TOP ۵۰
        query_store_query.query_id,
        query_store_query_text.query_sql_text,
        query_store_plan.query_plan AS query_plan_text,
        CAST(query_store_plan.query_plan AS XML) AS query_plan_xml,
        query_store_runtime_stats.last_execution_time
FROM sys.query_store_query
LEFT JOIN sys.query_store_query_text
ON query_store_query.query_text_id = query_store_query_text.query_text_id
LEFT JOIN sys.query_store_plan
ON query_store_query.query_id = query_store_plan.query_id
LEFT JOIN sys.query_store_runtime_stats
ON query_store_plan.plan_id = query_store_runtime_stats.plan_id
LEFT JOIN sys.query_store_runtime_stats_interval
ON query_store_runtime_stats.runtime_stats_interval_id = query_store_runtime_stats_interval.runtime_stats_interval_id
WHERE query_store_runtime_stats_interval.start_time BETWEEN '۱۱/۱۷/۲۰۱۵ ۲۲:۰۰:۰۰' AND '۱۱/۱۸/۲۰۱۵ ۲۲:۰۰:۰۰'
AND query_store_plan.query_plan LIKE '%convert_implicit%'
ORDER BY query_store_runtime_stats.last_execution_time DESC
```

با استفاده از اسکریپت زیر نیز می توانیم کلیه پلن هایی اجرایی که برای یک کوئری ایجاد شده است را مشاهده کنیم. با توجه به گرافی که Query store در اختیار ما قرار میداد و همچنین اماری که از پلن های اجرایی دریافت می کردیم، تصمیم میگرفتیم که بر چه اساسی یک پلن انتخاب شود که بتوانیم کلیه کوئری ها را از طریق ان اجرا کنیم و حتما اجبار به استفاده کنیم . لذا این امار را می توانید از طریق اسکریپت زیر به دست آورید:

```
WITH Query_MultPlans
AS
(
SELECT COUNT(*) AS cnt, q.query_id
FROM sys.query_store_query_text AS qt
```

```
JOIN sys.query_store_query AS q
   ON qt.query_text_id = q.query_text_id
JOIN sys.query_store_plan AS p
   ON p.query_id = q.query_id
GROUP BY q.query_id
HAVING COUNT(distinct plan_id) > ۱
)

SELECT q.query_id, object_name(object_id) AS ContainingObject,
   query_sql_text, plan_id, p.query_plan AS plan_xml,
   p.last_compile_start_time, p.last_execution_time
FROM Query_MultPlans AS qm
JOIN sys.query_store_query AS q
   ON qm.query_id = q.query_id
JOIN sys.query_store_plan AS p
   ON q.query_id = p.query_id
JOIN sys.query_store_query_text qt
   ON qt.query_text_id = q.query_text_id
ORDER BY query_id, plan_id;
```

همچنین اسکریپت دیگری برای این کار وجود دارد که اقای برنت اوزار در سایت خود به شکل پایین معرفی کرده است و می توانید اطلاعاتی خروجی قبلی را در این اسکریپت نیز مشاهده کنید:

جمع بندی

در این مقاله سعی کردیم که قسمت های کاربردی از پلن های اجرایی را در که بخشی از ان در مقاله قبلی معرفی شده بود را خدمت شما عزیزان ارایه دهیم . همچنین اسکریپت هایی معرفی شد که می توانید از طریق ان امار پلن ها و کارکرد کوئری ها و امار مرتبط با وضعیت اجرای آنها را نیز خارج از داشبوردها به دست اوردیم. در فصل اخر این کتاب دو ابزار DBA Tools و SpBlitz معرفی شده است. ابزار اول به صورت دستورات پاورشل قابل استفاده است و می توانید دستورات را در ان محیط استفاده کنید. ابزار دوم نیز تحت عنوان پکیجی به اسم SQL-Server-First-Responder-Kit معرفی شده است که می توانید امار جمع اوری شده توسط Query Store را از طریق اسکریپت های آن تحلیل کنید . در واقع SQL-Server-First-Responder-Kit مجموعه ایی از رویه های مختلفی هست که در ر رویه مجموعه ایی از پارامتر های کاربردی را مشاهده خواهید کرد. به عنوان مثال sp_BlitzQueryStore.sql را زمانی که بر روی دیتابیس Master نصب می کنید قابلیت تحلیل بر روی اطلاعات جمع اوری شده را به راحتی خواهید داشت. حتما جزییات مرتبط با این Package را از سایت گیت هاب مطالعه بفرمایید تا با قابلیت های بی نظیری که به شما ارایه می دهد به خوبی آشنا شوید. منتظر نظرات شما عزیزان در جهت بهبود کیفیت و کمیت مقالات اموزشی هستیم.

لیست منابع

https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-query-store-unforce-plan-transact-sql?view=sql-server-ver۱۵
https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-query-store-remove-query-transact-sql?view=sql-server-ver۱۵
https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-query-store-force-plan-transact-sql?view=sql-server-ver۱۵
https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-query-store-flush-db-transact-sql?view=sql-server-ver۱۵
https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/query-store-stored-procedures-transact-sql?view=sql-server-ver۱۵
https://docs.microsoft.com/en-us/sql/relational-databases/performance/monitoring-performance-by-using-the-query-store?view=sql-server-ver۱۵
https://docs.microsoft.com/en-us/sql/relational-databases/performance/query-store-hints?view=azuresqldb-current&viewFallbackFrom=sql-server-ver۱۵
https://www.red-gate.com/simple-talk/author/enrico-van-de-laar/
https://www.red-gate.com/hub/university/courses/sql-server-query-performance-tuning/built-in-sql-server-tools-make-query-tuning-easier/built-in-sql-server-tools-make-query-tuning-easier/week۱