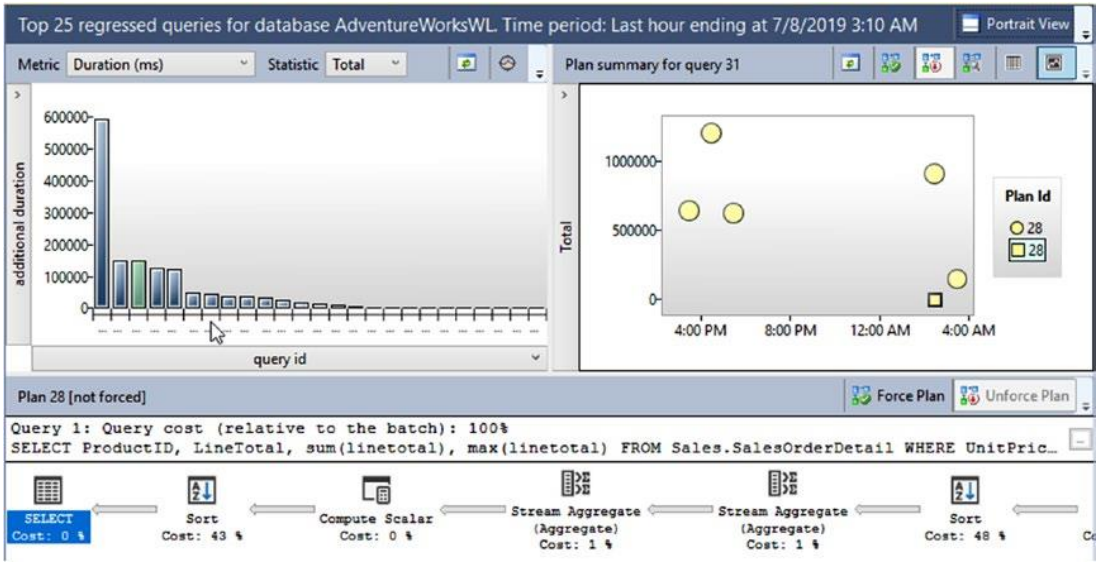
عنوان مقاله: اشنایی با Query Store بخش پنجم
نویسنده مقاله: تیم فنی نیک آموز
تاریخ انتشار: اسفند ماه ۱۴۰۰
منبع: https://nikamooz.com/query-store-part۰۵

مقدمه

در قسمت‌های قبل مفاهیم پایه ایی Query Store و معماری مرتبط با عملکرد این قابلیت مورد بررسی قرار گرفت. همچنین با پارامترها و سناریوهایی که در محیط‌های عملیاتی مورد استفاده قرار میگرفت نیز اشنا شدیم. در این فصل میخواهیم مفاهیم مرتبط با پلن‌های اجرایی را مورد بررسی قرار دهیم که ماهیت و ذات Query Store بر اساس آن ایجاد شده است. اشنایی و تسلط بر پلن‌های اجرایی یکی از عوامل و پیش نیازهای مرتبط با کدنویسی اصولی هست که هر برنامه‌نویس پایگاه داده و DBA باید تسلط کافی بر روی این موضوعات داشته باشند. پلن‌های اجرایی در واقع همانند زبان جدیدی هستند که نحوه تفسیر آن‌ها نیز کمی متفاوت هست. در پلن‌های اجرایی، مفهومی به اسم اپراتور داریم که به ازای سبک‌های مختلف کدنویسی ، چینش این پلن‌ها و نوع پلن‌ها متفاوت خواهد شد. در این مقاله، تمرکز اصلی بر روی این موضوع هست که تا به حال که نسبت به جمع اوری اصولی داده‌ها پرداختیم و Baselineهای خوبی ایجاد کردیم، حالا به چه نحوی می‌توانیم، مطابق با داشبوردهای تحلیلی که داریم، تحلیل جامعی بر روی پلن‌های اجرایی داشته باشیم یا حتی تصمیم بگیریم که در چه شرایطی از چه پلن اجرایی استفاده کنیم. یکی از امتیازات Query Store همین مورد هست که می‌توانیم در شرایطی مطابق با تحلیل‌هایی که انجام می‌شود، پلن‌های اجرایی را برای یک کوئری خاص، مجبور به استفاده کنیم که در بهینه‌ترین حالت ممکن، بهترین استفاده از اپراتورها انجام شود. بررسی مفاهیم مرتبط با پلن های اجرایی مقاله‌های مرتبط با خودش را می‌طلبد ولی پیرو مقالاتی که در خصوص Query Store ارایه شده است، مهمترین بخش‌های آن در این مقاله مورد بررسی قرار خواهد گرفت. ذکر این نکته لازم هست که قبل از شروع مقاله حتما مقالات قبلی را مطالعه کنید چرا که مواردی که مطرح می‌شود با جزییات کاملی در قسمت‌های قبلی مورد بررسی قرار گرفته است.

همان طور که در قسمت سوم مقاله‌های مرتبط با Query Store بررسی گردید، یکی از مهمترین گزارشات این قسمت، گزارش Regressed Queries هست که اطلاعات جامعی در خصوص پسرفت کوئری‌ها به ما ارایه میداد. این گزارش، پلن‌های اجرایی و کوئری‌هایی را به ما نمایش میدهد که پسرفتی در اجرا داشتند. این پسرفت‌ها را می‌توانید از اطلاعاتی که هر اپراتور در پلن اجرایی در اختیار شما قرار می‌دهد مشاهده کنید. محیط کلی این گزارش همانند شکل زیر هست:



در قسمت سمت چپ بالای این عکس، همان طور که مشاهده می‌کنید به ازای کوئری‌های مختلف، نمودارهای میله ایی مختلفی وجود دارد . در صورتی که بر روی هر کدام کلیک کنید جزییات مرتبط با کوئری مورد نظر را در پایین همین صفحه مشاهده خواهید کرد. در قسمت بالا سمت راست، کلیه پلن‌های اجرایی که به ازای این کوئری هست را مشاهده می‌کنیم. ممکن هست در این قسمت دایره یا اشکال مختلف را به رنگ‌های مختلف مشاهده کنید.

رنگ‌های مختلف در واقع به ما این پیام را نشان می‌دهد که این کوئری در زمان‌های مختلف، توسط پلن‌های اجرایی متفاوتی اجرا شده است. لذا می‌توانیم با بررسی هر کدام از این بخش‌ها متوجه شویم که بهینه‌ترین پلن اجرایی برای این قسمت شامل کدام پلن هست و این که می‌توانیم این پلن را به عنوان پلن منتخب همین بخش انتخاب کنیم. پسرفت کوئری‌ها دلایل مختلفی دارد. به عنوان مثال در سیستم‌های عملیاتی که بار ورودی داده‌ها مدام در حال افزایش هست یا فرایندهای نگهداری از ایندکس‌ها و ابجکت‌های مختف به درستی انجام نمی‌شود، تخمین و ساخت پلن‌های اجرایی با مشکلاتی همراه خواهد بود و دقت آن‌ها نیز کمتر خواهد شد. لذا مشاهده خواهیم کرد که پلن‌هایی اجرایی در سیستم داریم که نسبت به ظرفیت ورودی دیتاهایی که ما بین اپراتورها جا به جا می‌شود، اصلا مناسب نیست و بر عکس، پلن‌های اجرایی داریم که بسیار بیشتر از ظرفیت دیتاها از منابع مختلف استفاده می‌کنند. در هر دو این شرایط کندی و مصرف منابع زیادی را مشاهده خواهیم کرد. زمانی که حجم دیتا واکشی شده از سیستم بسیار زیاد باشد، باید اپراتورهایی که مسئولیت انتقال این داده‌ها را دارند به بهترین شکل ممکن انتخاب شود.

همچنین زمانی که ظرفیت دیتا ورودی بسیار کم هست و حجم داده‌ها بسیار کم هست، نباید از اپراتورهایی استفاده شود که منابع زیادی از سیستم دریافت می کنند. این انتخاب البته توسط Optmizer و انجین Sql server انجام می‌شود و در بعضی از شرایط به خصوص می‌توانیم فورس کنیم که فلان پلن حتما اجرایی شود. با توجه به هوشمندی بالایی که در ساخت پلن‌های اجرایی هست و موتور Sql server کلیه این موارد را در نظر میگیرد لذا مبحث نگهداری از دیتابیس بسیار اولویت پیدا می‌کند که به بهترین شکل و مطابق با یک Baseline مشخص این فرایندها انجام شود. به عنوان مثال به روز نگه داشتن Statisticsها یکی از مهمترین عوامل در بالابردن دقت پلن های اجرایی هست. یاعدم Fragmentation در سطح ایندکس‌ها باعث می‌شود که مسیر مورد نظر برای پیدا کردن داده درخواست شده سریع‌تر انجام شود. برای کلیه این قسمت‌ها

اسکریپت‌های متفاوتی وجود دارد که می‌توانید از جنبه‌های مختلف وضعیت ابجکت‌های مختلف را در دیتابیس به صورت دقیق بررسی کنید. در این مقاله سعی می‌کنیم که این گونه اسکریپت‌ها را نیز معرفی کنیم.

بررسی وضعیت Statistics ها:

توسط اسکریپت زیر می توانید اخرین وضعیت Statistics ها را بررسی کنید:

```
SELECT [sp].[object_id],
    OBJECT_NAME([sp].[object_id])  AS 'ObjectName',
    [sp].[stats_id]            AS 'StatID',
    [st].[name]                AS 'StatName',
    [sp].[last_updated]          AS 'LastTimeStatUpdated',
    [sp].[modification_counter]    AS 'NumberOfModifications'
FROM   [sys].[stats]              AS [st]
    OUTER APPLY sys.dm_db_stats_properties ([st].[object_id], [st].[stats_id]) AS [sp]
WHERE  OBJECT_NAME([sp].[object_id]) IS NOT NULL
    AND OBJECT_NAME([sp].[object_id]) NOT LIKE 'sys%'
```

همچنین در صورتی که قصد داشتید به ازای جدول خاصی کلیه Statistics ها را بررسی کنید می توانید از اسکریپت زیر استفاده کنید:

```
SELECT [sp].[object_id],
    OBJECT_NAME([sp].[object_id])  AS 'ObjectName',
    [sp].[stats_id]            AS 'StatID',
    [st].[name]                AS 'StatName',
    [sp].[last_updated]          AS 'LastTimeStatUpdated',
    [sp].[modification_counter]    AS 'NumberOfModifications'
FROM   [sys].[stats]              AS [st]
    OUTER APPLY sys.dm_db_stats_properties ([st].[object_id], [st].[stats_id]) AS [sp]
WHERE  OBJECT_NAME([sp].[object_id]) IS NOT NULL
    AND OBJECT_NAME([sp].[object_id]) NOT LIKE 'sys%'
    AND OBJECT_NAME([sp].[object_id]) IN ('WorkOrder')
```

در اسکریپت دوم جدول 'WorkOrder' جزو جداول عملیاتی دیتابیس AdventureWorks۲۰۱۹ هست که بررسی کلیه Statistics ها را به این Object محدود کردیم .

در صورتی که میخواهید کل Statistics های یک جدول را که به دفعات زیادی مورد استفاده قرار گرفته است را مشاهده می کنید می توانید از طریق اسکریپت زیر این کار را انجام دهید.

```
SELECT OBJ.NAME,
OBJ.OBJECT_ID,
STAT.NAME,
STAT.STATS_ID,
LAST_UPDATED,
MODIFICATION_COUNTER
FROM   SYS.OBJECTS          AS OBJ
INNER JOIN SYS.STATS  AS STAT
ON  STAT.OBJECT_ID = OBJ.OBJECT_ID
CROSS APPLY SYS.DM_DB_STATS_PROPERTIES(STAT.OBJECT_ID, STAT.STATS_ID) AS SP
WHERE  MODIFICATION_COUNTER > ۱۰۰۰;
```

به صورت کلی DMV مرتبط با sys. dm_db_stats_properties اطلاعات جامع و کاملی را در ا ختیار شما قرار خواهد داد که وضعیت Statها را می‌توانید به صورت کلی از طریق اطلاعاتی که این View در ا ختیار شما قرار می‌دهد مشاهده کنید. همواره سعی کنید که وضعیت Statها را به روز نگه دارید که پلن تخمینی و پلن واقعی شما نزدیک به یکدیگر باشد که نسبت به ظرفیت اطلاعات درخواست شده ازکلاینت‌ها اپراتورهای مرتبط نیز ساخته شود. مباحث مرتبط با عملکرد Statisticsها و نوع به روزرسانی خودکار آن‌ها به صورت کامل در مستندات ماکروسافت ارایه شده است. همچنین فرمول‌های محاسبه مرتبط با پلن‌های تخمینی نیز مباحث جالبی هستند که پیشنهاد می‌شود در ادامه این قسمت مطالعه گردد.

بررسی وضعیت Indexها:

مبحث ایندکس‌ها در Sql server بسیار مبحث گسترده ایی هست. زیرا انواع مختلفی از ایندکس‌ها را شاهد هستیم که هر یک در سناریوهای مختلف مورد استفاده قرار می‌گیرد. رفرنس‌های مرتبط با این موضوع از جنبه‌های مختلف تمامی موارد مرتبط با بحث ایندکس‌ها را به صورت کامل و جامع مورد بررسی قرار داده‌اند. یکی از بهترین کتاب‌ها برای بررسی ایندکس‌ها، کتاب Expert Performance Indexing in SQL Server ۲۰۱۹ هست که حتما مطالعه فرمایید. در این قسمت مباحث مهم مرتبط با نگه داری ایندکس‌ها و حذف ایندکس‌های بلا استفاده مطرح خواهد شد که در اجرای کوئری‌ها از ظرفیت پلن‌های اجرایی هم به درستی استفاده شود. یکی از مشکلاتی که بسیاری از سیستم‌های عملیاتی با آن درگیر هستند، وجود ایندکس‌های اشتباهی هست که فقط در سیستم موجود و در کل بلا استفاده هستند. ایندکس‌های بلا استفاده خود به عنوان سربار به سیستم تحمیل می‌شود چون که در هر بار عملیات ورود و حذف دیتا به جداول مرتبط باید کلیه ایندکس‌ها به روز شود. از طرفی ممکن هست در این ساختار که بر مبنای B-Tree هست گاهی در سطح LEAF LEVEL در مکان‌های مختلفی، داده‌های ورودی ذخیره شوند که همین عامل باعث به وجود آمدن مبحثی به اسم Fragmentation خواهد شد. برای این که از اخرین وضعیت Fragmentation ایندکس‌ها مطلع شویم می‌توانیم از اسکریپت زیر استفاده کنیم:

```
SELECT OBJECT_SCHEMA_NAME(ips.object_id)  AS SCHEMA_NAME,
      OBJECT_NAME(ips.object_id)        AS OBJECT_NAME,
      i.name                            AS index_name,
      i.type_desc                       AS index_type,
      ips.avg_fragmentation_in_percent,
      ips.avg_page_space_used_in_percent,
      ips.page_count,
      ips.alloc_unit_type_desc
FROM   sys.dm_db_index_physical_stats(
         DB_ID(N'AdventureWorks۲۰۱۹'),
         DEFAULT,
         DEFAULT,
         DEFAULT,
         'SAMPLED'
      )                                 AS ips
      INNER JOIN sys.indexes            AS i
         ON  ips.object_id = i.object_id
         AND ips.index_id = i.index_id
ORDER BY
      page_count DESC;
```

دقت داشته باشید که اسکریپت بالا را می توانیم به شکل دیگری نیز بنویسیم . در این قسمت می توانیم در حالت دیگری وضعیت ایندکس ها را بررسی کنیم.

```
SELECT OBJECT_NAME(IPS.object_id)  AS [TableName],
      SI.name                      AS [IndexName],
      IPS.Index_type_desc,
      IPS.avg_fragmentation_in_percent,
      IPS.avg_fragment_size_in_pages,
      IPS.avg_page_space_used_in_percent,
      IPS.record_count,
      IPS.ghost_record_count,
      IPS.fragment_count,
      IPS.avg_fragment_size_in_pages
FROM   sys.dm_db_index_physical_stats(DB_ID(N'AdventureWorks۲۰۱۹'), NULL, NULL, NULL, 'DETAILED') IPS
      JOIN sys.tables ST WITH (NOLOCK)
         ON  IPS.object_id = ST.object_id
      JOIN sys.indexes SI WITH (NOLOCK)
```

```
        ON  IPS.object_id = SI.object_id
        AND IPS.index_id = SI.index_id
WHERE  ST.is_ms_shipped = ۰
```

برای پیدا کردن ایندکس‌هایی که به صورت مرتب یا دوره ایی در سیستم استفاده می‌شود یا حتی کلا بلااستفاده هستند نیز می‌توانید از اسکریپت زیر استفاده کنید. ستون‌های مختلفی که نمایش داده شده است، میزان استفاده از ایندکس‌ها را در حالت‌های مختلف به شما نمایش می‌دهد. به عنوان مثال ایندکسی در حالت SEEK بیشترین استفاده را داشته و از طرفی ایندکس دیگری وجود دارد که نسبت INDEX SCAN آن به مراتب بالاتر از حالت INDEX SEEK آن بوده و باید به جزییات بیشتری مورد استفاده قرار گیرد. همچنین ایندکس‌های بلا استفاده که فقط در فرایند Update و به روزرسانی مورد استفاده قرار میگردند و عملا به عنوان یک سربار در سیستم شناخته می‌شوند:
تفاوتی که در دو اسکریپت بالا مشاهده می‌کنید برای Scanning Mode هر کدام هست که شاهد هستیم. به عنوان مثال زمانی که در حالت «DETAILED» سناریو بالا را بررسی می‌کنیم کلیه Pageها در همه سطوح اسکن می‌شود و خروجی را با جزییات کاملتری مشاهده می‌کنیم در صورتی که زمانی که بر روی «SAMPLED» تنظیم کنیم مسلما Pageهای کمتری درگیر این فرایند خواهد شد و خروجی به صورت خلاصه ارزیابی می‌شود. در حالت‌های مختلف علاوه بر بررسی Pageهای مرتبط با ایندکس‌ها، Pageهای سیستمی هم مورد بررسی قرار خواهد گرفت. موضوع دیگری که حائز اهمیت هست، بحث Missing indexهای موجود در دیتابیس هست. برای این بحث از DMVهای زیر بیشترین استفاده می‌شود.
برای اینکه این قسمت را به شکل دقیق‌تری بررسی کنیم در چند قسمت Queryهای مختلفی نوشته شده است که اطلاعات جامعی از بحث ایندکس‌ها و نوع استفاده آن‌ها به شما نمایش داده شده است:

```
SELECT @@SERVERNAME                    AS [Server Name],
    DB_NAME()                      AS [Database Name],
    OBJECT_SCHEMA_NAME(i.OBJECT_ID)   AS [Schema Name],
    OBJECT_NAME(i.OBJECT_ID)          AS [Table Name],
    i.name                         AS [Index Name],
    ps.partition_number              AS [Partition Number],
    i.is_primary_key                 AS [Is Primary Key],
    i.is_unique                    AS [Is Unique],
    ips.index_type_desc              AS [Type],
    ips.alloc_unit_type_desc         AS [Allocation Unit Type],
    p.data_compression_desc          AS [Compression Type],
    ps.row_count                   AS [Row Count],
    ips.record_count                 AS [Record Count],
    ips.ghost_record_count           AS [Ghost Record Count],
    ips.avg_fragmentation_in_percent  AS [Avg Fragmentation Pct],
    ips.avg_page_space_used_in_percent AS [Avg Page Space Used Pct],
```

```
    ps.used_page_count              AS [Used Page Count],
    (ps.used_page_count * ۸) / POWER(۱۰۲۴.۰, ۱) AS [Used Page Count (MB)],
    (ps.used_page_count * ۸) / POWER(۱۰۲۴.۰, ۲) AS [Used Page Count (GB)],
    ps.reserved_page_count          AS [Reserved Page Count],
    (ps.reserved_page_count * ۸) / POWER(۱۰۲۴.۰, ۱) AS [Reserved Page Count (MB)],
    (ps.reserved_page_count * ۸) / POWER(۱۰۲۴.۰, ۲) AS [Reserved Page Count (GB)],
    ius.user_seeks                  AS [User Seeks],
    ius.user_scans                  AS [User Scans],
    ius.user_lookups                AS [User Lookups],
    ius.user_updates                AS [User Updates],
    ius.system_seeks                AS [System Seeks],
    ius.system_scans                AS [System Scans],
    ius.system_lookups              AS [System Lookups],
    ius.system_updates              AS [System Updates],
    CASE
        WHEN (
                ius.user_seeks + ius.user_scans + ius.user_lookups + ius.user_updates
            ) > ۰ THEN (
                ۱.۰
                -(
                    CAST(ius.user_updates AS FLOAT)
                    / CAST(
                        (
                            ius.user_seeks + ius.user_scans + ius.user_lookups + ius.user_updates
                        ) AS FLOAT
                    )
                )
            )
        ELSE ۰
    END                             AS [% Reads]
FROM  sys.indexes                   AS [i]
    INNER JOIN sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, 'LIMITED') AS [ips]
        ON  i.OBJECT_ID = ips.OBJECT_ID
        AND i.index_id = ips.index_id
    INNER JOIN sys.dm_db_partition_stats AS [ps]
        ON  i.OBJECT_ID = ps.OBJECT_ID
        AND i.index_id = ps.index_id
        AND ips.partition_number = ps.partition_number
    INNER JOIN sys.partitions       AS p
        ON  i.index_id = p.index_id
```

```
        AND i.object_id = p.object_id
        AND ps.partition_id = p.partition_id
    LEFT OUTER JOIN sys.dm_db_index_usage_stats AS [ius]
        ON  i.object_id = ius.object_id
        AND i.index_id = ius.index_id
        AND ius.database_id = DB_ID()
ORDER BY
    ۱,
    ۲,
    ۳,
    ۴,
    ۵;
```

اسکریپت بالا به صورت کامل کلیه اطلاعات لازم در خصوص ایندکس‌ها را به شما نمایش می‌دهد. همچنین برای انالیز دقیق‌تر ایندکس‌ها می‌توانید از Packageهای دیگری که برای این کار هستند استفاده کنید. یکی از بهترین سورس کدها که در این زمینه نوشته شده است و به صورت رایگان در اختیار عموم هست Package اقای برنت اوزار هست با نام First Responder Kit که پارامترهای مختلفی را جهت انالیز ایندکس‌ها و مشکلات آن در اختیار شما قرار قرار خواهد داد. با توجه به گستردگی این مبحث، در مقالاتی که در اینده در خصوص این Package نوشته می‌شود صحبت خواهیم کرد. در قسمت بعدی می‌خواهیم حالتی را بررسی کنیم که وجود ایندکس‌هایی در سیستم ضروری هست در صورتی که چنین ایندکسی وجود ندارد. دقت داشته باشید ایندکس‌هایی که در این حالت به شما نمایش داده می‌شود را باید به صورت دقیق و جامع بررسی کنید. به عنوان مثال در قدم اول درصد کارایی ایندکس مورد نظر را در ابتدا بررسی کنید و در صورتی که درصد بالایی داشت برای مدتی توسط Query Store، کوئری‌های مرتبط با ان را بررسی کنید. با توجه به نگهداری سوابق حاصل از کوئری‌هایی که ثبت شده است می‌توانید عملکرد کلیه پلن‌ها را از طریق گزارش Regressed Queries بررسی نمایید. این یکی از بهترین روش‌ها برای حذف و اضافه کردن ایندکس‌ها در سیستم هست. به صورت کلی DMVهای زیر برای این منظور استفاده می‌شود.

- sys.dm_db_missing_index_details
- sys.dm_db_missing_index_group_stats
- sys.dm_db_missing_index_groups
- sys.dm_db_missing_index_columns

مطابق با اسکریپت زیر در صورتی که Missing index ایی داشته باشید به شما نمایش داده خواهد شد:

```
SELECT CONVERT(VARCHAR, GETDATE(), ۱۲۶)  AS runtime,
    mig.index_group_handle,
    mid.index_handle,
```

```
        CONVERT(
            DECIMAL(۲۸, ۱),
            migs.avg_total_user_cost *
            migs.avg_user_impact *(migs.user_seeks + migs.user_scans)
        )                           AS improvement_measure,
        'CREATE INDEX missing_index_' +
        CONVERT(VARCHAR, mig.index_group_handle) +
        '_' +
        CONVERT(VARCHAR, mid.index_handle) +
        ' ON ' +
        mid.statement +
        ' (' + ISNULL(mid.equality_columns, '') +
        CASE
            WHEN mid.equality_columns IS NOT NULL
                AND mid.inequality_columns IS NOT NULL THEN ','
            ELSE ''
        END + ISNULL(mid.inequality_columns, '') +
        ')' +
        ISNULL(' INCLUDE (' + mid.included_columns + ')', '') AS create_index_statement,
        migs.*,
        mid.database_id,
        mid.[object_id]
FROM   sys.dm_db_missing_index_groups mig
        INNER JOIN sys.dm_db_missing_index_group_stats migs
            ON  migs.group_handle = mig.index_group_handle
        INNER JOIN sys.dm_db_missing_index_details mid
            ON  mig.index_handle = mid.index_handle
WHERE  CONVERT(
            DECIMAL(۲۸, ۱),
            migs.avg_total_user_cost *
            migs.avg_user_impact *(migs.user_seeks + migs.user_scans)
        ) > ۱۰
ORDER BY
        migs.avg_total_user_cost *
        migs.avg_user_impact *(migs.user_seeks + migs.user_scans) DESC
```

برای این که سناریو بالا را شبیه سازی کنیم می توانی مطابق با اسکریپت های زیر به این گونه ایندکس ها برسیم:

در ابتدا باید دیتابیس ایی برای این سناریو درست می کنیم:

```
CREATE DATABASE TEST_MISSING_INDEX
GO
USE TEST_MISSING_INDEX
GO
```

سپس در مرحله بعد چک می کنیم در صورتی که جدولی مانند زیر در دیتابیس وجود داشت باید حذف گردد و مجددا ایجاد می کنیم:

```
DROP TABLE IF EXISTS EMPLOYEE ;
GO
```

در مرحله بعد جدول مورد نظر را ایجاد می کنیم:

```
CREATE TABLE EMPLOYEE(ID INT PRIMARY KEY CLUSTERED, MID INT, SALARY INT, JOINING_DATE DATETIME)
GO
```

در مرحله بعد اطلاعاتی را به صورت شبیه سازی شده وارد سیستم می کنیم :

```
DECLARE @I INT = ۱
WHILE (@I < ۱۰۰۰۰۰۰)
BEGIN
INSERT INTO EMPLOYEE
SELECT @I,
                CASE WHEN @I > ۵ THEN @I%۵ + ۱
                ELSE ۱ END,
                (@I/۱۰۰۰۰۰)*۱۰۰۰۰,
                DATEADD(YY,-@I%۱۰,GETDATE())
SET @I = @I + ۱
END
```

در مرحله بعدی قبل از هر تغییری یک بار DMV مرتبط با این کار را بررسی می کنیم . در این حالت نباید رکوردی به شما نمایش داده شود.

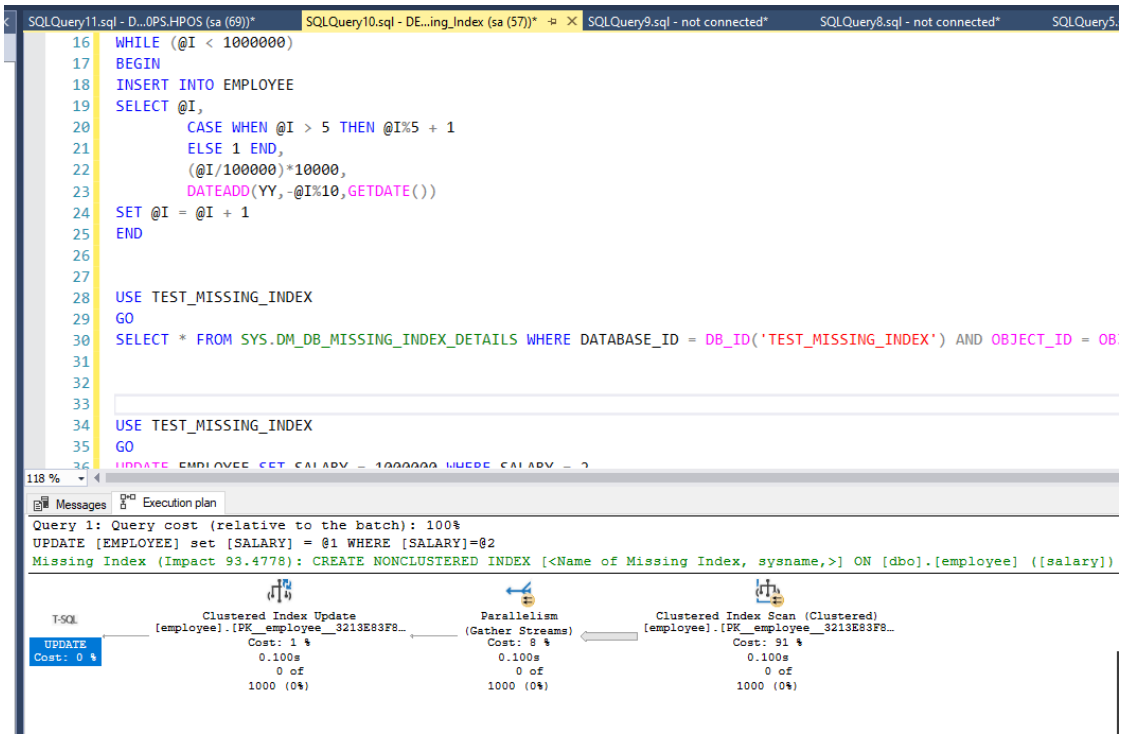
```
USE TEST_MISSING_INDEX
GO
SELECT * FROM SYS.DM_DB_MISSING_INDEX_DETAILS WHERE DATABASE_ID = DB_ID('TEST_MISSING_INDEX') AND
OBJECT_ID = OBJECT_ID('EMPLOYEE')
```

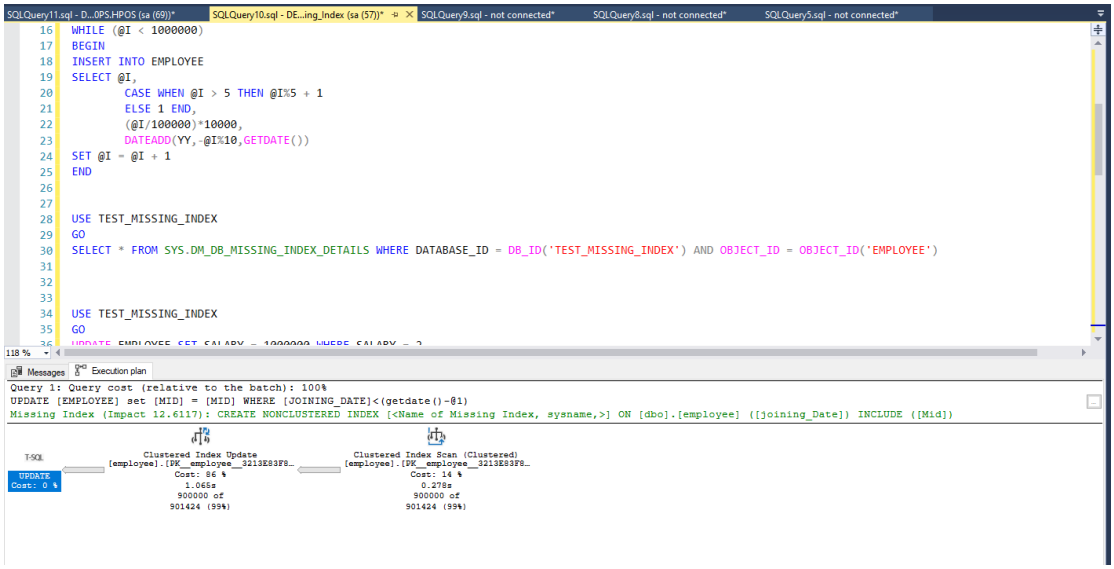
در مرحله بعد بر روی دو ستون از این ، ستون هایی رو update می کنیم:

```
USE TEST_MISSING_INDEX
GO
UPDATE EMPLOYEE SET SALARY = ۱۰۰۰۰۰۰ WHERE SALARY = ۰
UPDATE EMPLOYEE SET MID = MID WHERE JOINING_DATE < GETDATE() - ۱۰
```

در مرحله بعد مجددا اسکریپت بالا را اجرا می کنیم:

```
USE TEST_MISSING_INDEX
GO
SELECT * FROM SYS.DM_DB_MISSING_INDEX_DETAILS WHERE DATABASE_ID = DB_ID('TEST_MISSING_INDEX') AND
OBJECT_ID = OBJECT_ID('EMPLOYEE')
SELECT * FROM SYS.DM_DB_MISSING_INDEX_COLUMNS(۱)
SELECT * FROM SYS.DM_DB_MISSING_INDEX_COLUMNS(۳)
```





همان طور که در عکس‌های بالا مشخص شده است، در دو حالت ایندکس‌های مختلفی پیشنهاد شده است که اعمال گردد. ولی با توجه به درصد استفاده و مفید بودن ایندکس در این بخش می‌توان متوجه شد که ایندکس اول به میزان ۹۳ درصد و ایندکس دوم به میزان ۱۲ درصد کارایی خواهد داشت. لذا در همین مرحله می‌توانیم به این نتیجه برسیم که ایندکس دوم به نوعی کارایی چندانی نداشته و بلا استفاده خواهد بود. همان طور که ملاحضه فرمودید دو ردیف به شما نمایش داده می‌شود. علت این موضوع هم بدین خاطر هست که با توجه به هزینه به روزرسانی کوئری در این قسمت، انجین Sql server به شما این پیشنهاد را می‌دهد که وجود این دو ایندکس باعث تسریع در عملیات فوق خواهد شد. لذا در این DMV نیز قابل نمایش هستند:

```
CREATE NONCLUSTERED INDEX MISSING_INDEX_۱
ON [DBO].[EMPLOYEE] ([SALARY])
INCLUDE ([ID])
GO
CREATE NONCLUSTERED INDEX MISSING_INDEX_۳
ON [DBO].[EMPLOYEE] ([JOINING_DATE])
INCLUDE ([ID],[MID])
GO
```

قسمت سوم که در رابطه با بحث ایندکس ها دارای اهمیت زیادی هست ، ایندکس های بلا استفاده هستند . کوئری زیر شبیه به کوئری اولی هست که در همین قسمت مورد بررسی قرار گرفت . با این تفاوت که با توجه به شرط های اعمال شده در انتهای کوئری می توانیم دقیقا به این گونه ایندکس ها در سیستم دست پیدا کنیم:

```
SELECT TOP ۲۵
    O.NAME              AS OBJECTNAME,
    I.NAME              AS INDEXNAME,
    I.INDEX_ID          AS INDEXID,
    DM_IUS.USER_SEEKS    AS USERSEEK,
    DM_IUS.USER_SCANS    AS USERSCANS,
    DM_IUS.USER_LOOKUPS  AS USERLOOKUPS,
    DM_IUS.USER_UPDATES  AS USERUPDATES,
    P.TABLEROWS,
    'DROP INDEX ' + QUOTENAME(I.NAME)
    + ' ON ' + QUOTENAME(S.NAME) + '.'
    + QUOTENAME(OBJECT_NAME(DM_IUS.OBJECT_ID)) AS 'DROP STATEMENT'
FROM   SYS.DM_DB_INDEX_USAGE_STATS DM_IUS
    INNER JOIN SYS.INDEXES I
        ON  I.INDEX_ID = DM_IUS.INDEX_ID
        AND DM_IUS.OBJECT_ID = I.OBJECT_ID
    INNER JOIN SYS.OBJECTS O
        ON  DM_IUS.OBJECT_ID = O.OBJECT_ID
    INNER JOIN SYS.SCHEMAS S
        ON  O.SCHEMA_ID = S.SCHEMA_ID
    INNER JOIN (
            SELECT SUM(P.ROWS)       TABLEROWS,
                P.INDEX_ID,
                P.OBJECT_ID
            FROM   SYS.PARTITIONS     P
            GROUP BY
                P.INDEX_ID,
                P.OBJECT_ID
        ) P
        ON  P.INDEX_ID = DM_IUS.INDEX_ID
        AND DM_IUS.OBJECT_ID = P.OBJECT_ID
WHERE  OBJECTPROPERTY(DM_IUS.OBJECT_ID, 'ISUSERTABLE') = ۱
    AND DM_IUS.DATABASE_ID = DB_ID()
    AND I.TYPE_DESC = 'NONCLUSTERED'
    AND I.IS_PRIMARY_KEY = ۰
    AND I.IS_UNIQUE_CONSTRAINT = ۰
    AND DM_IUS.USER_SEEKS = ۰
    AND DM_IUS.USER_SCANS = ۰
    AND DM_IUS.USER_LOOKUPS = ۰
ORDER BY
```

```
    DM_IUS.USER_UPDATES     DESC
GO
```

در این کوئری ، ایندکس های بلااستفاده مشخص شده و همچنین می توانید به صورت داینامیک اسکریپت های حذف ایندکس را نیز مشاهده کنید . در صورتی که بخواهید اسکریپت های حذف و ساخت هر ایندکس را با یکدیگر مشاهده کنید می توانید از اسکریپت زیر استفاده کنید که خروجی کامل و جامعی نسبت به اسکریپت بالا به شما نمایش خواهد داد :

```
SELECT o.name                   AS TableName,
    i.name                  AS IndexName,
    dm_ius.user_seeks           AS UserSeek,
    dm_ius.user_scans           AS UserScans,
    dm_ius.user_lookups          AS UserLookups,
    dm_ius.user_updates          AS UserUpdates,
    p.TableRows,
    dm_ius.last_user_scan,
    dm_ius.last_user_seek,
    dm_ius.last_user_update,
    'DROP INDEX ' + QUOTENAME(i.name)
    + ' ON ' + QUOTENAME(s.name) + '.' + QUOTENAME(OBJECT_NAME(dm_ius.OBJECT_ID)) AS 'drop statement',
    index_create_script.Statement  AS 'Create Index Statement'
FROM   sys.dm_db_index_usage_stats dm_ius
    INNER JOIN sys.indexes i
        ON  i.index_id = dm_ius.index_id
        AND dm_ius.OBJECT_ID = i.OBJECT_ID
    INNER JOIN sys.objects o
        ON  dm_ius.OBJECT_ID = o.OBJECT_ID
    INNER JOIN sys.schemas s
        ON  o.schema_id = s.schema_id
    INNER JOIN (
            SELECT SUM(p.rows)       TableRows,
                p.index_id,
                p.OBJECT_ID
            FROM   sys.partitions     p
            GROUP BY
                p.index_id,
                p.OBJECT_ID
        ) p
        ON  p.index_id = dm_ius.index_id
        AND dm_ius.OBJECT_ID = p.OBJECT_ID
    INNER JOIN (
```

```
SELECT 'CREATE INDEX ' + IndexName + ' ON ' + TableName + ' (' + KeyCols + ' ) ' + CASE
                                                                WHEN IncludeCols
                                                                   IS NOT NULL THEN
                                                                   ' INCLUDE ('
                                                                   +
                                                                   IncludeCols
                                                                   + ' )'
                                                                ELSE ''
                                                              END AS Statement,
    IndexName
FROM   (
        SELECT '[' + Sch.name + '].[' + Tab.[name] + ']' AS TableName,
            Ind.Name AS IndexName,
            SUBSTRING(
              (
                SELECT ', ' + AC.name
                FROM   sys.[tables] AS T
                    INNER JOIN sys.[indexes] I
                        ON  T.[object_id] = I.[object_id]
                    INNER JOIN sys.[index_columns] IC
                        ON  I.[object_id] = IC.[object_id]
                        AND I.[index_id] = IC.[index_id]
                    INNER JOIN sys.[all_columns] AC
                        ON  T.[object_id] = AC.[object_id]
                        AND IC.[column_id] = AC.[column_id]
                WHERE  Ind.[object_id] = I.[object_id]
                    AND Ind.index_id = I.index_id
                    AND IC.is_included_column = ۰
                ORDER BY
                    IC.key_ordinal
                    FOR
                    XML PATH('')
              ),
              ۲,
              ۸۰۰۰
            )  AS KeyCols,
            SUBSTRING(
              (
                SELECT ', ' + AC.name
                FROM   sys.[tables] AS T
```

```
                    INNER JOIN sys.[indexes] I
                        ON  T.[object_id] = I.[object_id]
                    INNER JOIN sys.[index_columns] IC
                        ON  I.[object_id] = IC.[object_id]
                        AND I.[index_id] = IC.[index_id]
                    INNER JOIN sys.[all_columns] AC
                        ON  T.[object_id] = AC.[object_id]
                        AND IC.[column_id] = AC.[column_id]
                WHERE  Ind.[object_id] = I.[object_id]
                    AND Ind.index_id = I.index_id
                    AND IC.is_included_column = ۱
                ORDER BY
                    IC.key_ordinal
                    FOR
                    XML PATH('')
                ),
                ۲,
                ۸۰۰۰
            )  AS IncludeCols
        FROM   sys.[indexes] Ind
            INNER JOIN sys.[tables] AS Tab
                ON  Tab.[object_id] = Ind.[object_id]
            INNER JOIN sys.[schemas] AS Sch
                ON  Sch.[schema_id] = Tab.[schema_id]
    ) index_create_script
  ) index_create_script
  ON  i.name = index_create_script.Indexname
WHERE  OBJECTPROPERTY(dm_ius.OBJECT_ID, 'IsUserTable') = ۱
    AND dm_ius.database_id = DB_ID()
    AND i.type_desc = 'nonclustered'
    AND i.is_primary_key = ۰
    AND i.is_unique_constraint = ۰
    AND i.is_unique = ۰
ORDER BY
    last_user_seek,
    last_user_scan
```

دقت داشته باشید ستون‌هایی که در عکس زیر مشخص شده است باید به صورت کامل مورد بررسی شود. طبق تجربه ایی که از سناریوها و بیزینس‌های محیط‌های عملیاتی حاصل شده است، در این سناریوها از نرم‌افزارهای مختلفی جهت

یکپارچه‌سازی سازمان استفاده می‌شود. به عنوان مثال نرم افزاری شبیه به نرم‌افزارهای ERP را در نظر بگیرید که در شرکت‌های مختف جهت یکپارچه‌سازی اطلاعات بین واحدهای مختلف استقرار داده شده است. با توجه به این که در این گونه نرم‌افزارها گزارشات مختلف ممکن است در بازه‌های زمانی متفاوتی از سیستم استخراج شود، حتما باید به این نکته دقت داشته باشید که قبل از حذف ایندکس‌ها، کلیه فرایندهای سیستم یا فرایندهای روتین که مورد استفاده قرار میگیرد را به خوبی بشناسید. به عنوان مثال در شرکت‌های که در زمینه خرده فروشی فعالیت می‌کنند گزارش موجودی یکی از مهمترین گزارشات سیستمی هست که با ماژول‌های مختلف به شدت درگیر هست. لذا کلیه جداولی که مربوط به سوابق تراکنش‌های ورودی و خروجی و برگه‌های مرتبط با آن‌ها هستند در این گزارشات استفاده می‌شود. از طرفی در شرکت‌هایی که در زمینه‌های پخش مویرگی فعالیت می‌کنند، گزارشات فصلی در بازه‌های بزرگ‌تر از سیستم گرفته می‌شود. لذا فیلترها و ایندکس‌هایی که برای این تیپ گزارشات مورد استفاده قرار میگیرند را نیز باید با دقت بیشتری بررسی کنید. شاید مدت زمان زیادی به اصلاح این تیپ ایندکس‌ها مورد استفاده قرار نگیرد ولی در فواصل مختلف زمانی در نهایت استفاده می‌شود. لذا شناخت فرایندها در هر شرکتی از اهمیت بالایی برخوردار هست که فرایندها به خوبی شناخته شود. در این قسمت می خواهیم عواملی را بررسی کنیم که در پلن های اجرایی از اهمیت بالایی برخوردار هستند:

- Warning Signs in Execution Plans
- First Operator
- Most Costly Operator
- Fat Pipes
- Scans
- Extra Operators
- Estimated vs. Actual

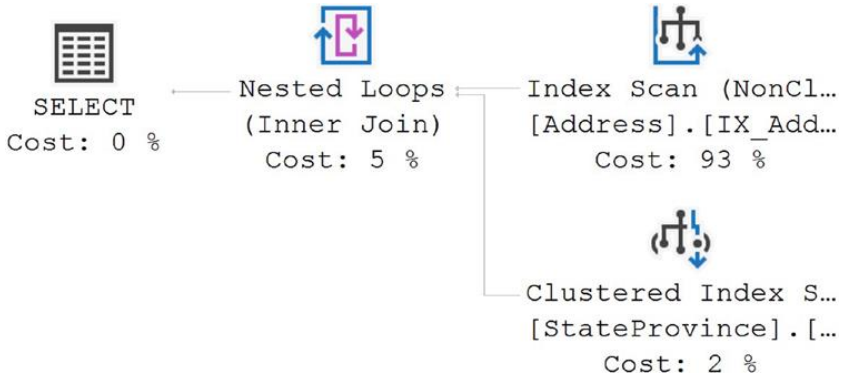
Warning Signs in Execution Plans

در پلن های اجرایی ، بعضا بر روی اپراتور های مختلف ، علامت های مشاهده می شود که اخطاری را به کاربر نمایش می دهد مبنی بر این که در این سناریو استفاده از این اپراتور اصلا مناسب نیست چرا باعث به وجود امدن مشکلاتی در روند خروجی سیستم شده است . این عوامل مطابق با مواردی که گفته شد عمدتا مرتبط با اپراتور هایی هست که نباید در این شرایط انتخاب شوند و عملا برای دیتا فعلی از ظرفیت بسیار بزرگی برخوردار هستند و صرفا باعث اتلاف منابع مصرفی می شوند . لذا شناخت دقیق این عوامل باعث بهبود در عمکرد انتقال اطلاعات ما بین ورودی اپراتور های مختلف خواهد شد .

First Operator

معمولا پلن های اجرایی را از سمت راست به چپ ، از بالا به پایین بررسی می کنند . در این شرایط این بررسی ها از ابتدای کوئری باعث می شود که قدم به قدم اطلاعات ورودی به هر مرحله با دقت مورد بررسی قرار گیرد تا به اخرین اپراتور که در سمت چپ ترین مکان قرار دارد برسیم . در واقع اطلاعاتی که این اپراتور در اختیار ما قرار می دهد ، بسیار اطلاعات مفید و

تجمیع شده ایی از ساختار Query و کلیات پلن در اختیار ما قرار خواهد داد . لذا به خاطر اهمیتی که این اپراتور دارد ، به صورت مجزایی در کتاب ها و منابع زبان اصلی به صورت تخصصی بررسی خواهد شد . همان طور که در عکس پایین نیز ملاحظه می کنید ، اطلاعاتی که در کلیه فیلد ها در این اپراتور هست در سایر اپراتور ها دیده نمی شود :



Cached plan size	32 KB
CardinalityEstimationModelVersion	140
CompileCPU	78
CompileMemory	424
CompileTime	106
DatabaseContextSettingsId	1
Estimated Number of Rows	1.66667
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	0.193771
MemoryGrantInfo	
MissingIndexes	
Optimization Level	FULL
OptimizerHardwareDependentProperties	
OptimizerStatsUsage	
Parameter List	@City
Column	@City
Parameter Compiled Value	N'Mentor'
Parameter Data Type	nvarchar(30)
ParentObjectId	1271675578
QueryHash	0xDD75E124763781F2
QueryPlanHash	0x84AED2D3F83C43D2
Reason For Early Termination Of Statement Optir	Good Enough Plan Found

Most Costly Operator

در پلن های اجرایی همواره اپراتور هایی وجود دارد که در کل پلن همواره درصد بیشتری را نسبت به اپراتور های دیگر گرفته باشد . به عنوان مثال زمانی که Lookup توسط پلن اجرایی استفاده شده باشد عمدتا درصد بالایی از Cost پلن اجرایی مرتبط با همین اپراتور ها هست .

Fat Pipes

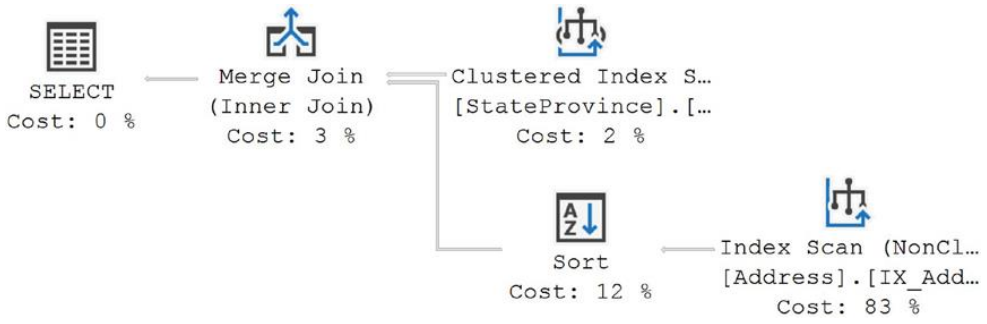
ما بین اپراتور ها در پلن های اجرایی ، خط لوله هایی جهت انتقال دیتا وجود دارد . زمانی که پلن های اجرایی به دقت بررسی کنید متوجه خواهید شد که ما بین این اپراتور ها بعضا این خط لوله ها به ضخامت های مختلف مورد استفاده قرار خواهد گرفت . مسلما هر چه قدر ضخامت این لوله های انتقالی بیشتر باشد دیتا بیشتر در حال انتقال هست . شاید اصلا نیاز به انتقال این حجم از دیتا نباشیم . در این شرایط باید به اپراتور های قبل تر برگشت و علت درخواست این حجم از ورودی را توسط سایر اپراتور ها بررسی کنیم . لذا یکی از مهمترین بخش ها در این قسمت بررسی این راه های ارتباطی هست .

Scans

در پلن های اجرایی مفهوم Scan عمدتا به این معناست که کلیه دیتاها استخراج شود . در صورتی که در بعضی شرایط با ایندکس گذاری مناسب ، نیاز به دریافت کلیه اطلاعات و داده ها نیست . می توانیم حجم زیادی از دیتا ورودی را کاهش دهیم و دقیقا به اطلاعاتی که نیاز هست دسترسی پیدا کنیم . لذا مطابق با توضیحی که در این بخش از کتاب به آن اشاره شده است large amounts of I/O مطرح گردیده است که مسلما باعث کندی در عملکرد کوئری ها شده است .

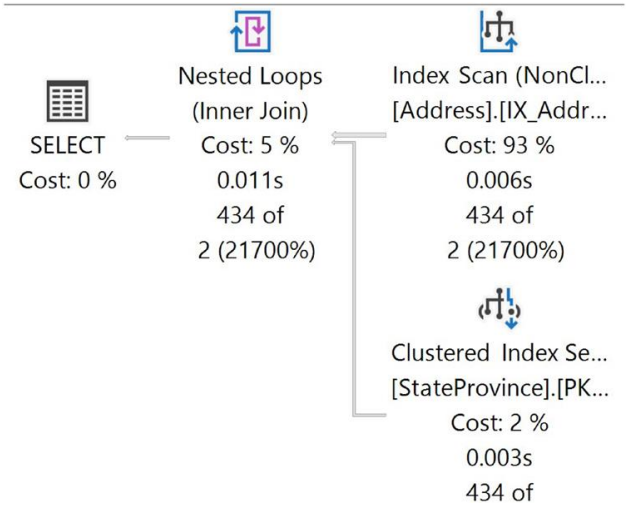
Extra Operators

بعضا در پلن‌های اجرایی اپراتورهایی دیده می‌شود که علاوه بر هزینه زیادی که بر روی پلن اعمال می‌کنند عملا پیچیدگی پلن‌ها را نیز افزایش می‌دهند. به عنوان مثال زمانی که از عبارت ORDER BY در قسمت‌های مختلف کد استفاده می‌کنید، این اپراتور دیده می‌شود. این اپراتور یکی از سنگین‌ترین اپراتورهایی هست که در این گونه پلن‌ها دیده می‌شود. دقت داشته باشید پلن‌های اجرایی با توجه به شرایط خاصی که در کوئری ایجاد می‌شود این گونه اپراتورها را ایجاد می‌کند که در نتیجه اپراتورهای ورودی بعدی نیز به صورت کلی تغییر می‌کند. در شکل زیر اپراتور Sort را مشاهده می‌کنید که به دلیل اینکه Optimizer تشخیص داده دو طرف ورودی اطلاعات در صورتی که مرتب شده باشد بهترین پلن ایجاد می‌شود لذا این اپراتور ایجاد شده است که در نتیجه منجر به اپراتور Merge Join خواهد شد. در صورتی که اگر دیتا ورودی از قسمت پایین مرتب شده باشد، هزینه ایی بابت این پلن و مرتب‌سازی ان شاهد نبودیم. لذا در این قسمت می‌توانیم متوجه شویم که وجود ایندکس‌هایی در جدول Address می‌توانید باعث حذف این گونه اپراتورهای اضافی شود. این اپراتورها البته محدود به Sort نیستند. بلکه زمانی که ایندکس‌گذاری مناسب نداشته باشید جداولی تحت عنوان Table spool ایجاد می‌شود که چندین بار مورد استفاده قرار خواهد گرفت. در کل بررسی پلن‌های متعدد باعث می‌شود که در نگاه اول کلیه اپراتورهای اضافی را به راحتی تشخیص دهید و ساختار کدنویسی درست را پیاده‌سازی کنید.



Estimated vs. Actual

در پلن‌های اجرایی دو مفهوم بسیار مهم وجود دارد. پلن‌های تخمینی و پلن‌های واقعی. پلن‌های تخمینی بر اساس امار Statها و ایندکس‌ها به صورت تقریبی، یک پلن از کوئری شما نمایش می‌دهد. زمانی که کوئری برای اجرا می‌رود پلن واقعی یا Actual آن ایجاد می‌شود که بر اساس اپراتورهایی که در آن مشاهده می‌کنید دیتا به اولین اپراتور می‌رسد. نکته مهمی که در این قسمت هست، در نوع تخمین و دقت تخمین این پلن‌ها هست. هر چه قدر این دو عدد، که در مشخصات هر پلن قابل نمایش هست به یکدیگر نزدیک‌تر باشد باعث می‌شود که هدررفت منابع یا مشکلات کوئری کمتر شود. همان طور که قبلا صحبت شد در صورتی که پلن تخمینی مقدار کمتری از پلن واقعی ارایه دهد، یا برعکس آن، پلن تخمینی مقدار بیشتری از پلن واقعی ارایه دهد، پلن اجرایی شما فاقد اعتبار هست. چرا که در حالت اول در زمان اجرا باید یک سرباری بر روی سایر قسمت‌های سیستم ایجاد کند که فضای کافی برای اجرای کوئری ایجاد شود و در حالت دوم نسبت حجم ورودی داده‌ها به نسبت ظرفیت اپراتورها بیشتر است که باعث بالا رفتن فضای ذخیره‌سازی پلن و بعضا مشکلات دیگری خواهد شد. لذا سعی کنید این دو مقدار را همیشه به یکدیگر نزدیک نگه دارید.



به عنوان مثال پلن بالا را در نظر بگیرید . این پلن را می توانید به از طریق اسکریپت های زیر ایجاد 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
```

مثالی که در این قسمت ارایه گردید یکی از مشکلات رایجی از که در بحث رویه‌ها یا Store procها شاهد ان هستیم. در ابتدا پلن اجرایی بر اساس یک مقدار ورودی ایجاد می‌شود. سپس در صورتی که مقدار ورودی آن عوض شود، بر اساس پلن قبلی که ایجاد شده است دیتاها از همان اپراتور استفاده می‌کنند. مشکل زمانی ایجاد می‌شود که چگالی اطلاعات بر اساس مقادیر مختلفی که در این جداول هستند متفاوت هست. به عنوان مثال اگر بیزینسی را در نظر بگیریم که شعب مختلفی در شهر تهران دارد مسلما تراکنش‌های ورودی بر اساس شهر تهران، یا مشتریانی که در شهر تهران اقدام به خرید می‌کنند زیاد هستند. لذا اطلاعات هر کدام از این جداول نسبت به سایر شهرها از کمیت بیشتری برخوردار هستند و مسلما اپراتورهای مختلفی را در سیستم مشاهده خواهید کرد. در مثالی که عنوان گردید، اپراتوری را مشاهده می‌کنید به اسم Nested loop که برای دیتاهایی با حجم پایین استفاده می‌شود. از طرفی اطلاعات مشتری مرتبط با شهر Mentor در این مثال با مقادیر کمتری نسبت به اطلاعات مشتریان شهر London در سیستم ثبت شده است. لذا این اپراتور با این که برای شهر Mentor مناسب هست، برای شهر London مناسب نبوده و ظرفیت اپراتورها نسبت به اطلاعات جدولی که توسط این مقدار فراخوانی شده است بسیار پایین‌تر هست.

118 %

Results | Messages

	AddressID	AddressLine1	AddressLine2	City	StateProvinceName	PostalCode
1	28674	1005 Valley Oak Plaza	NULL	London	England	SW6 SBY
2	17038	1019 Mt. Davidson Court	NULL	London	England	SW8 4BG
3	20210	1040 Northridge Road	NULL	London	England	W1X3SE
4	25816	1050 Creed Ave	NULL	London	England	W10 6BL
5	27652	1055 Horseshoe Road	NULL	London	England	SW1P 2NU
6	22559	1085 Ash Lane	NULL	London	England	W1N 9FA
7	20912	109 Clayton Road	NULL	London	England	W1N 9FA
8	23458	1097 Kulani Lane	NULL	London	England	SW19 3RU
9	25050	1099 C Street	NULL	London	England	SW1P 2NU
10	11907	1105 N. 48th St	NULL	London	England	E17 6JF
11	17096	112 Kathleen Drive	NULL	London	England	EC1R 0DU
12	21985	1141 Panoramic Dr.	NULL	London	England	SW1P 2NU
13	14692	1172 Flamingo Dr.	NULL	London	England	C2H 7AU

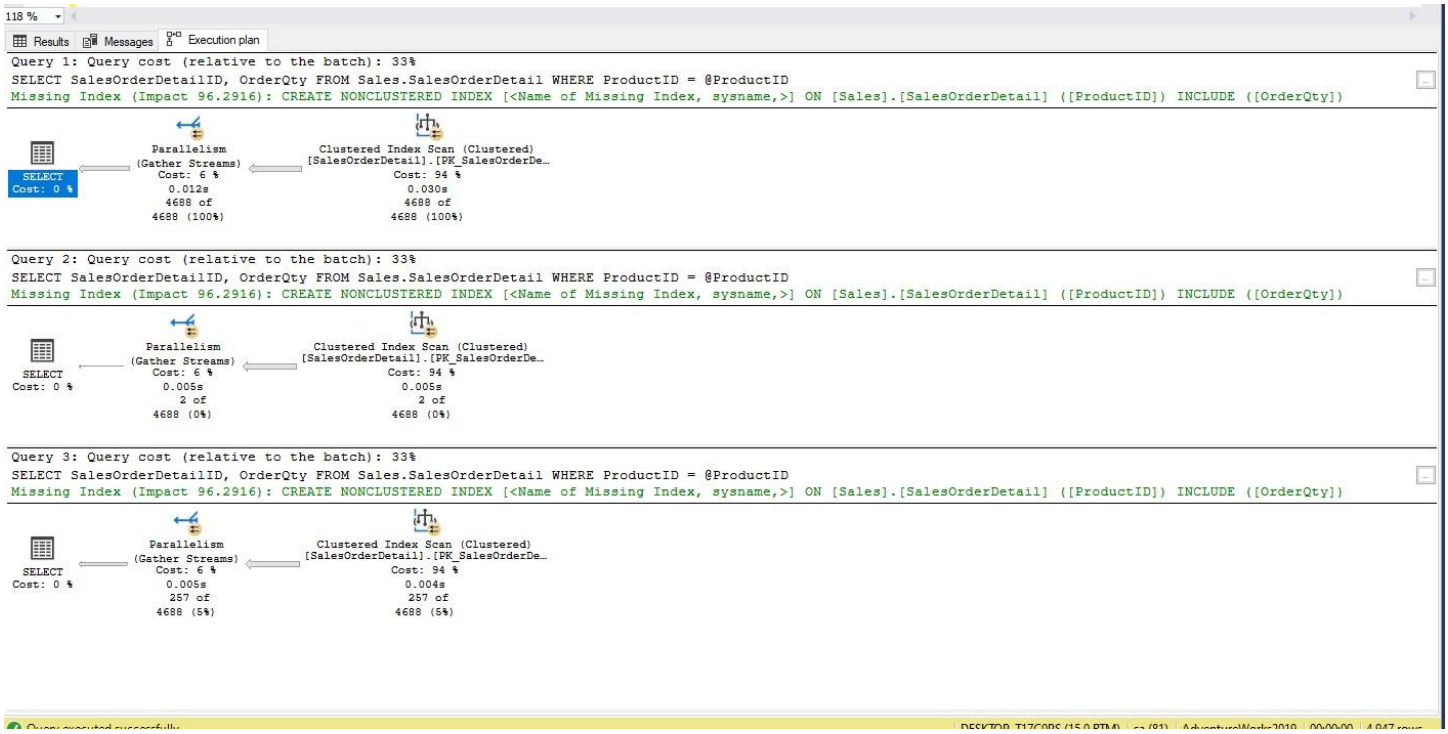
	AddressID	AddressLine1	AddressLine2	City	StateProvinceName	PostalCode
1	773	77993 Mentor Ave.	NULL	Mentor	Ohio	44060

Query executed successfully.

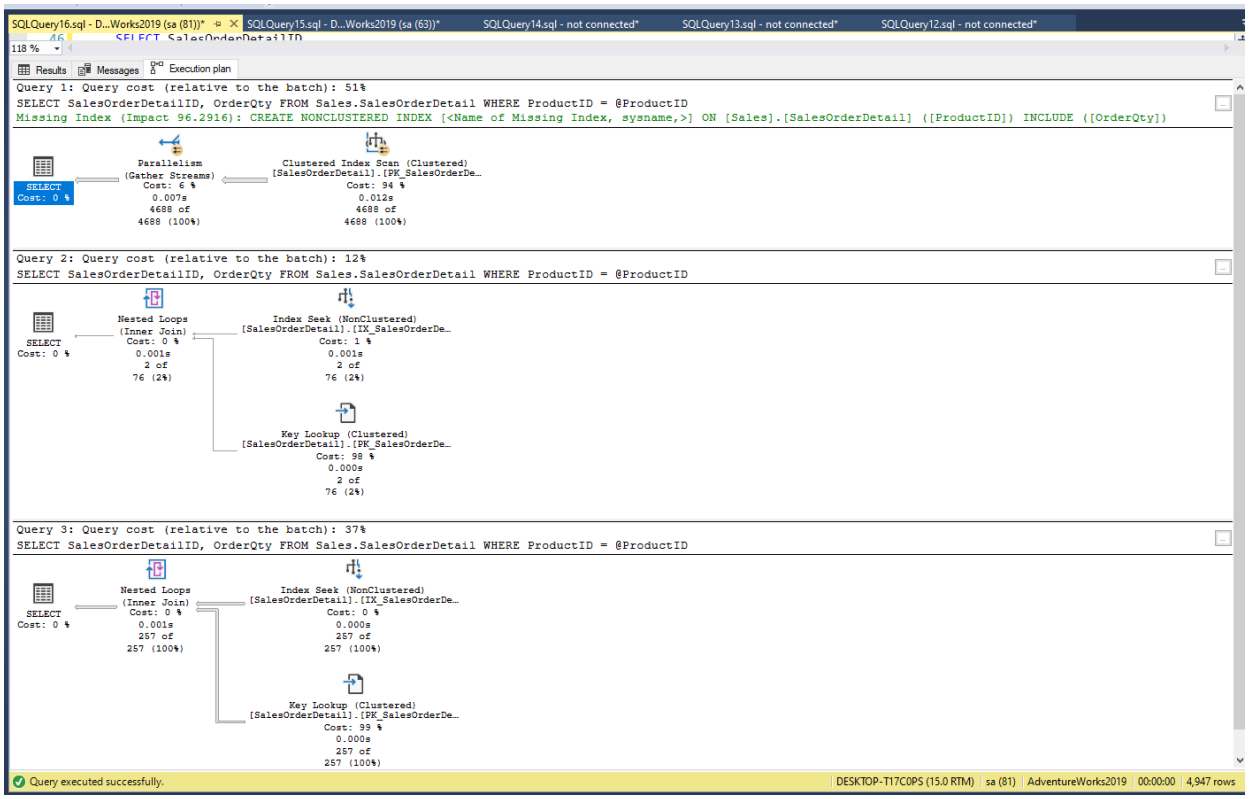
مثال دومی که برای بحث parameter sniffing می توانید در نظر به شرح زیر هست . اسکریپت ها را با روشن کردن Actual plan اجرا کنید که پلن های اجرایی هر بخش را مشاهده کنید :

```
/*
 * بررسی تراکنش های کد کالا شماره ۸۹۷
 */
SELECT SalesOrderDetailID,
       OrderQty
FROM   Sales.SalesOrderDetail
WHERE  ProductID = ۸۹۷;

/*
 * بررسی تراکنش های کد کالا ۹۴۵
 */
SELECT SalesOrderDetailID,
       OrderQty
FROM   Sales.SalesOrderDetail
WHERE  ProductID = ۹۴۵;


/*
 * بررسی تراکنش های کد کالا ۸۷۰
 */
SELECT SalesOrderDetailID,
       OrderQty
FROM   Sales.SalesOrderDetail
WHERE  ProductID = ۸۷۰;



DROP PROC
IF EXISTS Get_OrderID_OrderQty
   GO

/*
 * ساخت یک رویه برای مشاهده مقادیر مختلف به ازای هر کالا
 */

CREATE PROCEDURE Get_OrderID_OrderQty
          @ProductID INT
AS
```

```
        SELECT SalesOrderDetailID,
               OrderQty
        FROM   Sales.SalesOrderDetail
        WHERE  ProductID = @ProductID;

        /*
        * مشاهده تراکنش های کالاهای منتخب توسط رویه نوشته شده
        */
        /*
        * فراخوانی تراکنش های مرتبط با کالاهای مختلف
        */
EXEC Get_OrderID_OrderQty @ProductID = ۸۷۰
GO
EXEC Get_OrderID_OrderQty @ProductID = ۸۹۷
GO
EXEC Get_OrderID_OrderQty @ProductID = ۹۴۵
GO
```

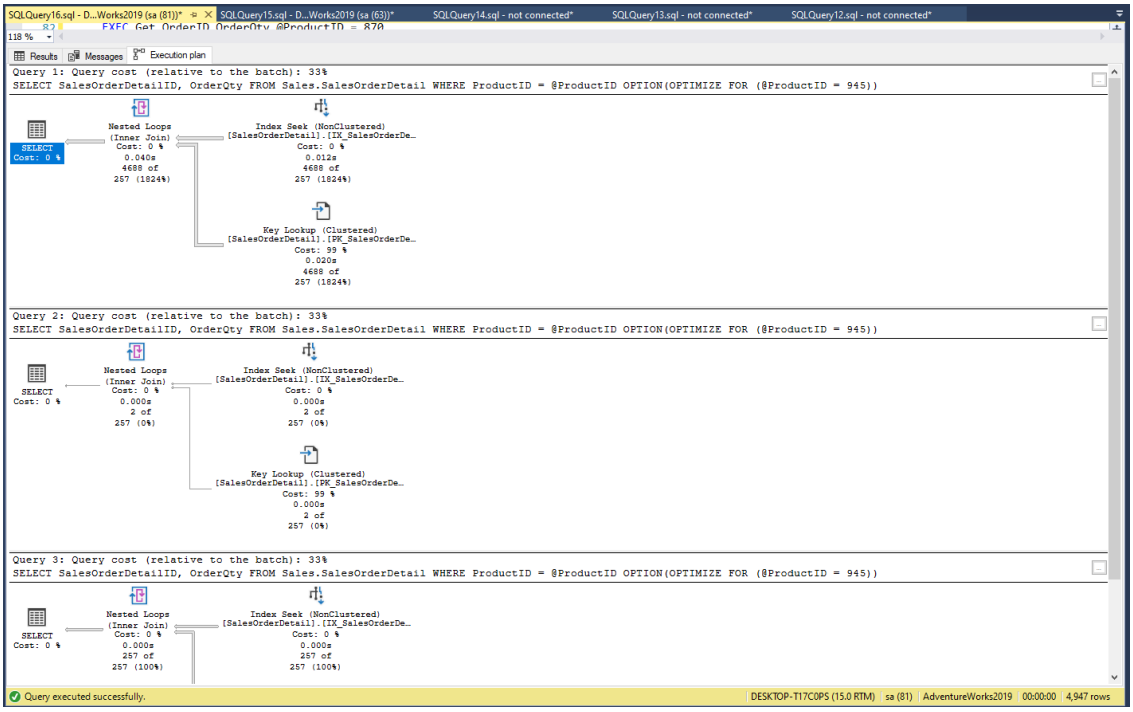


```
/*
 * WITH RECOMPILE اصلاح رویه مورد نظر با استفاده از قابلیت
 */
ALTER PROCEDURE Get_OrderID_OrderQty
        @ProductID INT
WITH RECOMPILE
AS
        SELECT SalesOrderDetailID,
            OrderQty
        FROM   Sales.SalesOrderDetail
        WHERE  ProductID = @ProductID;



*/
فراخوانی تراکنش های مرتبط با کالاهای مختلف
*/

EXEC Get_OrderID_OrderQty @ProductID = ۸۷۰
GO
EXEC Get_OrderID_OrderQty @ProductID = ۸۹۷
GO
EXEC Get_OrderID_OrderQty @ProductID = ۹۴۵
GO
```




```
/*
 * OPTIMIZE FOR اصلاح رویه مورد نظر  با استفاده از قابلیت
 */

ALTER PROCEDURE Get_OrderID_OrderQty
        @ProductID INT
AS
        SELECT SalesOrderDetailID,
            OrderQty
        FROM   Sales.SalesOrderDetail
        WHERE  ProductID = @ProductID
            OPTION(OPTIMIZE FOR (@ProductID = ۹۴۵));
GO


/*
 * فراخوانی تراکنش های مرتبط با کالاهای مختلف
 */
EXEC Get_OrderID_OrderQty @ProductID = ۸۷۰
GO
```

```
EXEC Get_OrderID_OrderQty @ProductID = ۸۹۷
GO
EXEC Get_OrderID_OrderQty @ProductID = ۹۴۵
GO
```



همان طور که شکل‌های بالا به ازای هر کوئری مشاهده کردید، پلن‌های اجرایی مختلفی در شرایط مختلفی ایجاد شده است که بسته به محیط‌های عملیاتی باید با دقت بالایی این موارد مورد بررسی قرار گیرد تا شاهد مشکل parameter sniffing نباشیم. همان طور که در مثال اول مشاهده کردید تراکنش‌ها به ازای شهرها در یک جدول دارای مقادیر مختلفی بود که پلن‌های اجرایی به درستی برای شهر مورد نظر ایجاد نشده بود. در مثال دوم هم مشاهده کردیم که پلن‌های اجرایی برای کالاهایی که تراکنش‌های مختلفی در ورودی و خروجی ان‌ها بود به درستی ایجاد نشده بود. مثالاهای مختلفی از این مبحث در سایت‌های مختلف ارایه شده است که حتما پیشنهاد می‌کنم مطالعه بفرمایید. همواره سعی کنید، جداول عملیاتی و اطلاعات پایه بیزینس‌ها را به خوبی بشناسید و نسبت به حجم دیتاها بر اساس ایتم‌های مختلفی که در آن دیتابیس هستند شناخت کافی پیدا کنید که شاهد این مشکلات نباشید.

جمع‌بندی

در این مقاله سعی کردیم کمی عمیق‌تر گزارشات مرتبط با پسرفت کوئری‌ها را مورد بررسی قرار دهیم. پسرفت کوئری‌ها دلایل و علت‌های مختلفی داشت که چند مورد از آن‌ها را با یکدیگر بررسی کردیم و اسکریپت‌های مرتبط با هر بخش سعی شد که به صورت کامل ارایه شود. با توجه به این که بخش مهمی از مبحث Query Store مرتبط با همین پلن‌های اجرایی و

نحوه استفاده از اپراتورهای ان باشد سعی بر ان بود که مفاهیم مرتبط با پلن‌های اجرایی، با جزییات بیشتری مطرح شود که سریعا بتوانید بر اساس این پلن‌ها مشکلات را به اصولی‌ترین شکل برطرف کنید. در قسمت‌های بعد، در مورد قابلیت‌های ویژه ایی که Query Store در اختیار ما قرار می‌دهد که بتوانیم کنترل دقیق‌تری بر روی این پلن‌ها داشته باشیم یا به صورت هوشمند، بهترین پلن اجرایی انتخاب شود صحبت خواهیم کرد. لینک‌ها و مستندات هر بخش نیز در اخر این مقاله خدمت شما عزیزان ارایه شده است که می‌توانید موارد مطرح شده را با جزییات بیشتری مطالعه فرمایید.

لینک ها و رفرنس ها

https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-db-index-usage-stats-transact-sql?view=sql-server-ver۱۵
https://www.mssqltips.com/sqlservertip/۱۶۳۴/find-sql-server-missing-indexes-with-dmvs
https://dba.stackexchange.com/questions/۲۰۶۱۹/improve-performance-of-sys-dm-db-index-physical-stats
https://www.mssqltips.com/sqlservertip/۴۳۳۱/sql-server-index-fragmentation-overview/
https://www.mssqltips.com/sqlservertip/۱۷۰۸/index-fragmentation-report-in-sql-server/
https://github.com/MicrosoftDocs/sql-docs/tree/live/docs/relational-databases/system-dynamic-management-views
https://github.com/kendalvandyke/SQL-Server-Scripts/blob/master/Performance۲۰%-۲۰%Find۲۰%Index۲۰%Seeks۲%C۲۰%Scans۲%C۲۰%۲۶%۲۰%Lookups۲۰%(۲%۲۰۰۵B).sql
https://github.com/MicrosoftDocs/sql-docs/blob/live/docs/relational-databases/system-dynamic-management-views/sys-dm-db-index-usage-stats-transact-sql.md
https://sqlservergeeks.com/sys-dm_db_missing_index_details/
https://www.sqlservercentral.com/scripts/unused-indexes
https://www.mssqltips.com/sqlservertip/۱۵۴۵/deeper-insight-into-used-and-unused-indexes-for-sql-server/
https://www.brentozar.com/archive/۲۰۱۳/۰۶/the-elephant-and-the-mouse-or-parameter-sniffing-in-sql-server/
https://www.mssqltips.com/sqlservertip/۳۲۵۷/different-approaches-to-correct-sql-server-parameter-sniffing/