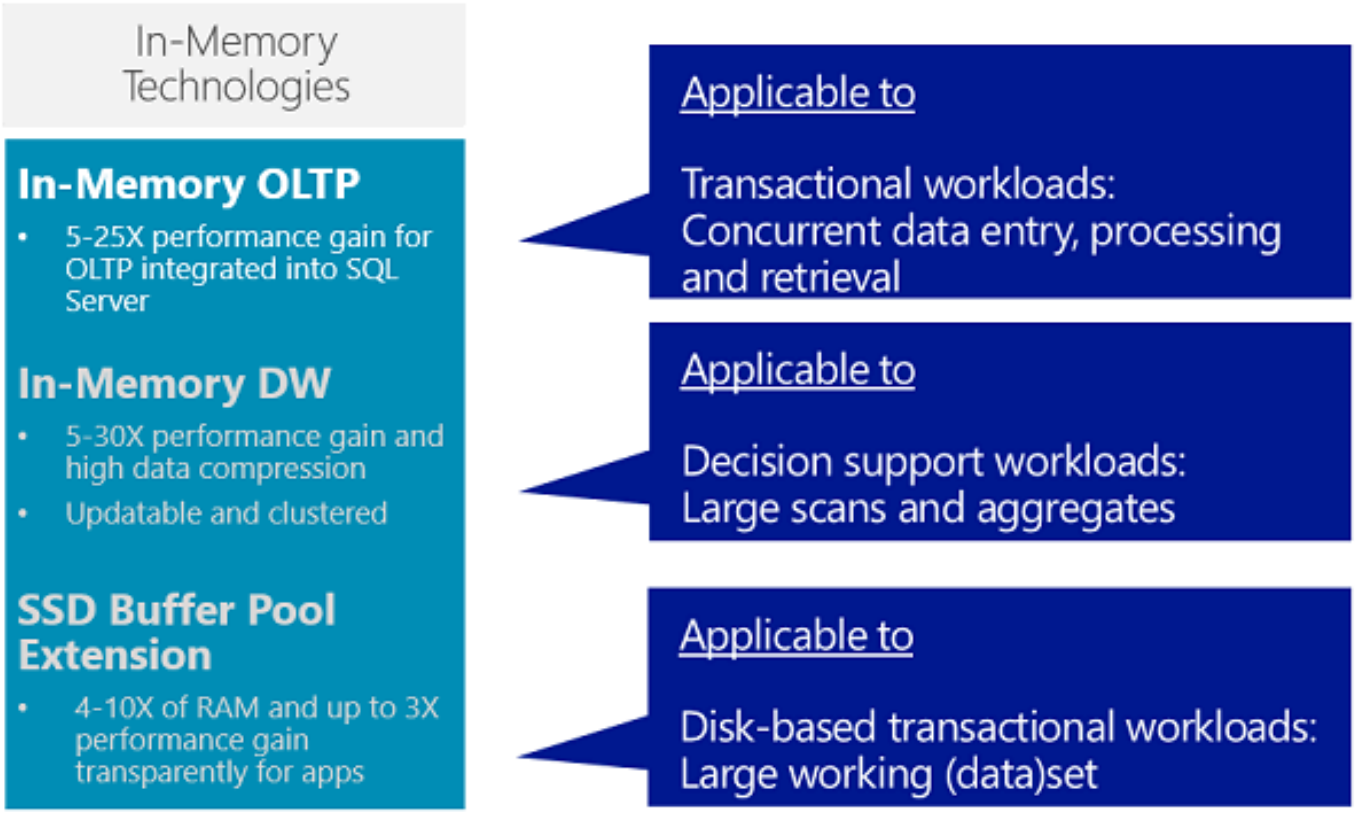
عنوان: **معرفی OLTP درون حافظه‌ای در SQL Server 2014**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۳/۱۱ ۰:۵۵
آدرس: www.dotnettips.info
گروه‌ها: SQL Server, In-Memory OLTP

OLTP درون حافظه‌ای، مهم‌ترین ویژگی جدید SQL Server 2014 است. موتور بانک اطلاعاتی disk based اس کیوال سرور، حدود 15 تا 20 سال قبل تهیه شده‌است و موتور جدید درون حافظه‌ای OLTP آن، بزرگترین بازنویسی این سیستم از زمان ارائه‌ی آن می‌باشد و شروع این پروژه به 5 سال قبل بر می‌گردد. علت تهیه‌ی آن نیز به نیازهای بالای پردازش‌های همزمان مصرف کنندگان این محصول در سال‌های اخیر، نسبت به 15 سال قبل مرتبط است. با استفاده از امکانات OLTP درون حافظه‌ای، امکان داشتن جداول معمولی disk based و جداول جدید memory optimized با هم در یک بانک اطلاعاتی میسر است؛ به همراه مهیا بودن تمام زیرساخت‌هایی مانند تهیه بک آپ، بازیابی آن‌ها، امنیت و غیره برای آن‌ها.



آیا جداول بهینه سازی شده‌ی برای حافظه، همان DBCC PINTABLE منسوخ شده هستند؟

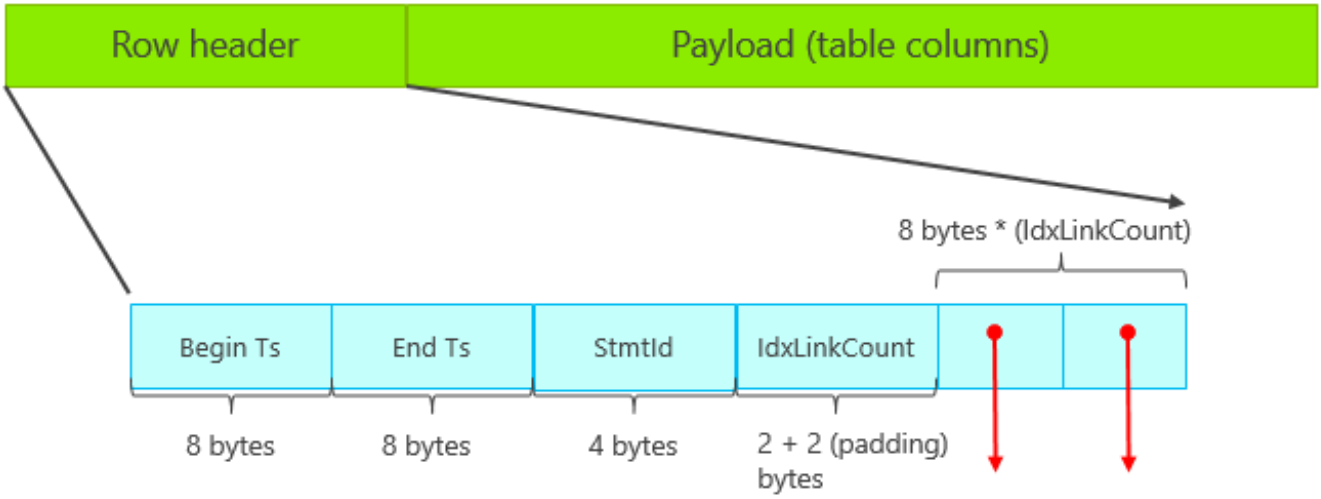
در نگارش‌های قدیمی‌تر اس کیوال سرور، دستوری وجود داشت به نام DBCC PINTABLE که سبب ثابت نگه داشتن صفحات جداول مبتنی بر دیسک یک دیتابیس، در حافظه می‌شد. به این ترتیب تمام خواندن‌های مرتبط با آن جدول، از حافظه صورت می‌گرفت. مشکل این روش که سبب منسوخ شدن آن گردید، اثرات جانبی آن بود؛ مانند خوانده شدن صفحات جدیدتر (با توجه به اینکه ساختار پردازشی و موتور بانک اطلاعاتی تغییری نکرده بود) و نیاز به حافظه‌ی بیشتر تا حدی که کل کش بافر سیستم را پر می‌کرد و امکان انجام سایر امور آن مختل می‌شدند. همچنین اولین ارجاعی به یک جدول، سبب قرار گرفتن کل آن در حافظه می‌گشت. به علاوه ساختار این سیستم نیز همانند روش مبتنی بر دیسک، بر اساس همان روش‌های قفل گذاری، ذخیره سازی اطلاعات و تهیه ایندکس‌های متداول بود.

اما جداول بهینه سازی شده‌ی برای حافظه، از یک موتور کاملا جدید استفاده می‌کنند؛ با ساختار جدیدی برای ذخیره سازی اطلاعات و تهیه ایندکس‌ها. دسترسی به اطلاعات آن‌ها شامل قفل گذاری‌های متداول نیست و در آن حداقل زمان دسترسی به

اطلاعات درنظر گرفته شده‌است. همچنین در آن‌ها data pages یا index pages و کش بافر نیز وجود ندارد.

نحوه‌ی ذخیره سازی و مدیریت اطلاعات جداول بهینه سازی شده برای حافظه

جداول بهینه سازی شده برای حافظه، فرمت ردیف‌های کاملا جدیدی را نیز به همراه دارند و جهت قرارگرفتن در حافظه ودسترسی سریع به آن‌ها بهینه سازی شده‌اند. برخلاف جداول مبتنی بر دیسک سخت که اطلاعات آن‌ها در یک سری صفحات خاص به نام‌های data or index pages ذخیره می‌شوند، اینگونه جداول، دارای ظروف مبتنی بر صفحه نیستند و از مفهوم چند نگارشی برای ذخیره سازی اطلاعات استفاده می‌کنند؛ به این معنا که ردیف‌ها به ازای هر تغییری، دارای یک نگارش جدید خواهند بود و بلافاصله در همان نگارش اصلی به روز رسانی نمی‌شوند.
در اینجا هر ردیف دارای یک timestamp شروع و یک timestamp پایان است. timestamp شروع بیانگر تراکنشی است که ردیف را ثبت کرده و timestamp پایان برای مشخص سازی تراکنشی بکار می‌رود که ردیف را حذف کرده است. اگر timestamp پایان، دارای مقدار بی‌نهایت باشد، به این معنا است که ردیف متناظر با آن هنوز حذف نشده‌است. به روز رسانی یک ردیف در اینجا، ترکیبی است از حذف یک ردیف موجود و ثبت ردیفی جدید. برای یک عملیات فقط خواندنی، تنها نگارش‌هایی که timestamp معتبری داشته باشند، قابل مشاهده خواهند بود و از مابقی صرفنظر می‌گردد.
در OLTP درون حافظه‌ای که از روش چندنگارشی همزمانی استفاده می‌کند، برای یک ردیف مشخص، ممکن است چندین نگارش وجود داشته باشند؛ بسته به تعداد باری که یک رکورد به روز رسانی شده‌است. در اینجا یک سیستم garbage collection همیشه فعال، نگارش‌هایی را که توسط هیچ تراکنشی مورد استفاده قرار نمی‌گیرند، به صورت خودکار حذف می‌کند؛ تا مشکل کمبود حافظه رخ ندهد.



آیا می‌توان به کارآیی جداول بهینه سازی شده برای حافظه با همان روش متداول مبتنی بر دیسک اما با بکارگیری حافظه‌ی بیشتر و استفاده از یک SSD RAID رسید؟

خیر! حتی اگر کل بانک اطلاعاتی مبتنی بر دیسک را در حافظه قرار دهید به کارآیی روش جداول بهینه سازی شده‌ی برای حافظه نخواهید رسید. زیرا در آن هنوز مفاهیمی مانند data pages و index pages به همراه یک buffer pool پیچیده وجود دارند. در روش‌های مبتنی بر دیسک، ردیف‌ها از طریق page id و row offset آن‌ها قابل دسترسی می‌شوند. اما در جداول بهینه سازی شده‌ی برای حافظه، ردیف‌های جداول با یک B-tree خاص به نام Bw-Tree در دسترس هستند.

میزان حافظه‌ی مورد نیاز برای جداول بهینه سازی شده‌ی برای حافظه

باید درنظر داشت که تمام جداول بهینه سازی شده‌ی برای حافظه، به صورت کامل در حافظه ذخیره خواهند شد. بنابراین بدیهی

است که نیاز به مقدار کافی حافظه در اینجا ضروری است. توصیه صورت گرفته، داشتن حافظه‌ای به میزان دو برابر اندازه‌ی اطلاعات است. البته در اینجا چون با یک سیستم هیبرید سر و کار داریم، حافظه‌ی کافی جهت کار buffer pool مختص به جداول مبتنی بر دیسک را نیز باید درنظر داشت.
همچنین اگر به اندازه‌ی کافی حافظه در سیستم تعبیه نشود، شاهد شکست مداوم تراکنش‌ها خواهید بود. به علاوه امکان بازیابی و restore جداول را نیز از دست خواهید داد.
البته لازم به ذکر است که اگر کل بانک اطلاعاتی شما چند ترابایت است، نیازی نیست به همین اندازه یا بیشتر حافظه تهیه کنید. فقط باید به اندازه‌ی جداولی که قرار است جهت قرار گرفتن در حافظه بهینه سازی شوند، حافظه تهیه کنید که حداکثر آن 256 گیگابایت است.

چه برنامه‌هایی بهتر است از امکانات OLTP درون حافظه‌ای SQL Server 2014 استفاده کنند؟

- برنامه‌هایی که در آن‌ها تعداد زیادی تراکنش کوتاه مدت وجود دارد به همراه درجه‌ی بالایی از تراکنش‌های همزمان توسط تعداد زیادی کاربر.
- اطلاعاتی که توسط برنامه زیاد مورد استفاده قرار می‌گیرند را نیز می‌توان در جداول بهینه سازی شده جهت حافظه قرار داد.
- زمانیکه نیاز به اعمال دارای write بسیار سریع و با تعداد زیاد است. چون در جداول بهینه سازی شده‌ی برای حافظه، صفحات داده‌ها و ایندکس‌ها وجود ندارند، نسبت به حالت مبتنی بر دیسک، بسیار سریعتر هستند. در روش‌های متداول، برای نوشتن اطلاعات در یک صفحه، مباحث همزمانی و قفل‌گذاری آن‌را باید در نظر داشت. در صورتیکه در روش بهینه سازی شده‌ی برای حافظه، به صورت پیش فرض از حالتی همانند snapshot isolation و همزمانی مبتنی بر نگارش‌های مختلف رکورد استفاده می‌شود.
- تنظیم و بهینه سازی جداولی با تعداد Read بالا. برای مثال، جداول پایه سیستم که اطلاعات تعاریف محصولات در آن قرار دارند. این نوع جداول عموما با تعداد Readهای بالا و تعداد Write کم شناخته می‌شوند. چون طراحی جداول مبتنی بر حافظه از hash tables و اشاره‌گرهایی برای دسترسی به رکوردهای موجود استفاده می‌کند، اعمال Read آن نیز بسیار سریعتر از حالت معمول هستند.
- مناسب جهت کارهای data warehouse و ETL Staging Table. در جداول مبتنی بر حافظه امکان عدم ذخیره سازی اطلاعات بر روی دیسک سخت نیز پیش بینی شده‌است. در این حالت فقط اطلاعات ساختار جدول، ذخیره‌ی نهایی می‌گردد و اگر سرور نیز ری استارت گردد، مجددا می‌تواند اطلاعات خود را از منابع اصلی data warehouse تامین کند.

محدودیت‌های جداول بهینه سازی شده‌ی برای حافظه در SQL Server 2014

- تغیر اسکیما و ساختار جداول بهینه سازی شده‌ی برای حافظه مجاز نیست. به بیان دیگر دستور ALTER TABLE برای اینگونه جداول کاربردی ندارد. این مورد جهت ایندکس‌ها نیز صادق است. همان زمانیکه جدول ایجاد می‌شود، باید ایندکس آن نیز تعریف گردد و پس از آن این امکان وجود ندارد.
تنها راه تغییر اسکیمای اینگونه جداول، Drop و سپس ایجاد مجدد آن‌ها است.
البته باید درنظر داشت که SQL Server 2014، اولین نگارش این فناوری را ارائه داده‌است و در نگارش‌های بعدی آن، بسیاری از این محدودیت‌ها قرار است که برطرف شوند.
- جداول بهینه سازی شده‌ی برای حافظه حتما باید دارای یک ایندکس باشند. البته اگر یک primary key را برای آن‌ها تعریف نمائید، کفایت می‌کند.
- از unique indexها پشتیبانی نمی‌کند، مگر اینکه از نوع primary key باشد.
- حداکثر 8 ایندکس را می‌توان بر روی اینگونه جداول تعریف کرد.
- امکان تعریف ستون identity در آن وجود ندارد. اما می‌توان از قابلیت sequence برای رسیدن به آن استفاده کرد.
- DML triggers را پشتیبانی نمی‌کند.
- کلیدهای خارجی و قیود را پشتیبانی نمی‌کند.
- حداکثر اندازه‌ی یک ردیف آن 8060 بایت است. بنابراین از نوع‌های داده‌ای max دار و XML پشتیبانی نمی‌کند.
این مورد در حین ایجاد جدول بررسی شده و اگر اندازه‌ی ردیف محاسبه‌ی شده‌ی آن توسط SQL Server 2014 بیش از 8060 بایت باشد، جدول را ایجاد نخواهد کرد.

اگر سرور را ری استارت کنیم، چه اتفاقی برای اطلاعات جداول بهینه سازی شده‌ی برای حافظه رخ می‌دهد؟

حالت DURABILTY انتخاب شده‌ی در حین ایجاد جدول بهینه سازی شده‌ی برای حافظه، تعیین کننده‌ای این مساله است. اگر SCHEMA_ONLY انتخاب شده باشد، کل اطلاعات شما با ری استارت سرور از دست خواهد رفت؛ البته اطلاعات ساختار جدول حفظ خواهد گردید. اگر حالت SCHEMA_AND_DATA انتخاب شود، اطلاعات شما پس از ری‌استارت سرور نیز در دسترس خواهد بود. این اطلاعات به صورت خودکار از لاگ تراکنش‌ها بازیابی شده و مجددا در حافظه قرار می‌گیرند.
حالت SCHEMA_ONLY برای مصارف برنامه‌های data warehouse بیشتر کاربرد دارد. جایی که اطلاعات قرار است از منابع داده‌ی مختلفی تامین شوند.

برای مطالعه بیشتر

SQL Server 2014: NoSQL Speeds with Relational Capabilities
SQL Server 2014 In-Memory OLTP Architecture and Data Storage
Overview of Applications, Indexes and Limitations for SQL Server 2014 In-Memory OLTP Tables
Microsoft SQL Server 2014: In-Memory OLTP Overview
SQL Server in Memory OLTP for Database Developers
Exploring In-memory OLTP Engine (Hekaton) in SQL Server 2014 CTP1

نظرات خوانندگان

نویسنده: فرید طاهری
تاریخ: ۱۳۹۳/۰۳/۱۱ ۱۰:۳

جناب نصیری با تشکر از مقاله مفیدتون لازم می‌دونم در جهت تکمیل مباحث به چند نکته اشاره کنم

1- « *اگر حالت SCHEMA_AND_DATA انتخاب شود، اطلاعات شما پس از ری‌استارت سرور نیز در دسترس خواهد بود. این اطلاعات به صورت خودکار از لاگ تراکنش‌ها بازیابی شده و مجددا در حافظه قرار می‌گیرند* ».

بازیابی اطلاعات مربوط به تراکنش‌هایی که به ازای In Memory OLTP است بوسیله Data File + Delta File و Log File می‌باشد. در صورتیکه Schema_AND_Data را به ازای این نوع جداول فعال کنید داده‌های شما در Data File و داده‌های حذف شده در Delta File ثبت می‌گردد. مکانیزم Log File برای In Memory OLTP همچنان مانند جداول Disk base وجود دارد اما با بهینه سازی مناسب مانند ثبت Log Record کمتر به ازای عملیات کاربران و...

2- در جایی دیگر در متن اشاره شده که In Memory OLTP اجازه استفاده از Identity را به کاربر نمی‌دهد باید اشاره کنم که این موضوع برای نسخه CTP بوده است در نسخه RTM این قابلیت وجود دارد . لازم می‌دانم اشاره کنم که در Books Online جایی گفته شده که امکان استفاده وجود ندارد و در جایی هم گفته شده وجود دارد .

به مثال زیر دقت کنید

```
CREATE TABLE test(
[ID] BIGINT IDENTITY(1,1) NOT NULL PRIMARY KEY NONCLUSTERED HASH WITH (BUCKET_COUNT=10000),
N1 NVARCHAR(100),
N2 NVARCHAR(100),
N3 NVARCHAR(100)
) WITH (MEMORY_OPTIMIZED=ON,DURABILITY = SCHEMA_AND_DATA)
GO
```

این مثال در SQL Server 2014 RTM Edition قابل اجرا است اما یکسری محدودیت داریم . مثلاً مقدار شروع و گام افزایش باید 1 باشد .
باز هم از مطالب خوب شما متشکرم مقاله مفیدی بود .

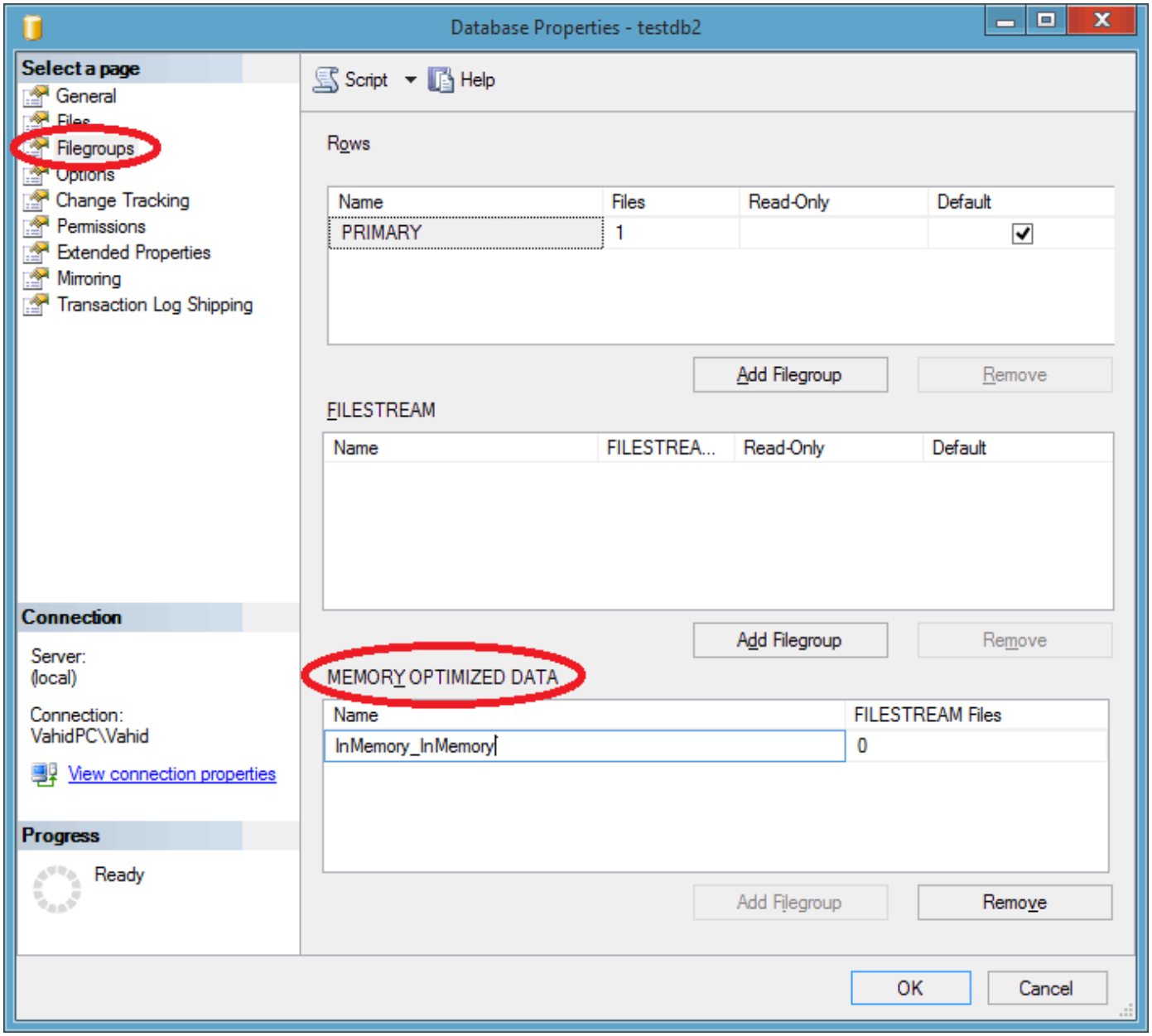
عنوان: **ایجاد جداول بهینه سازی شده برای حافظه در SQL Server 2014**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۳/۱۱ ۲۱:۵
آدرس: www.dotnettips.info
گروه‌ها: SQL Server, In-Memory OLTP

پس از نگاهی به مفاهیم مقدماتی OLTP درون حافظه‌ای در SQL Server 2014 ، در ادامه به نحوه‌ی انجام تنظیمات خاص جداول بهینه سازی شده برای حافظه خواهیم پرداخت.

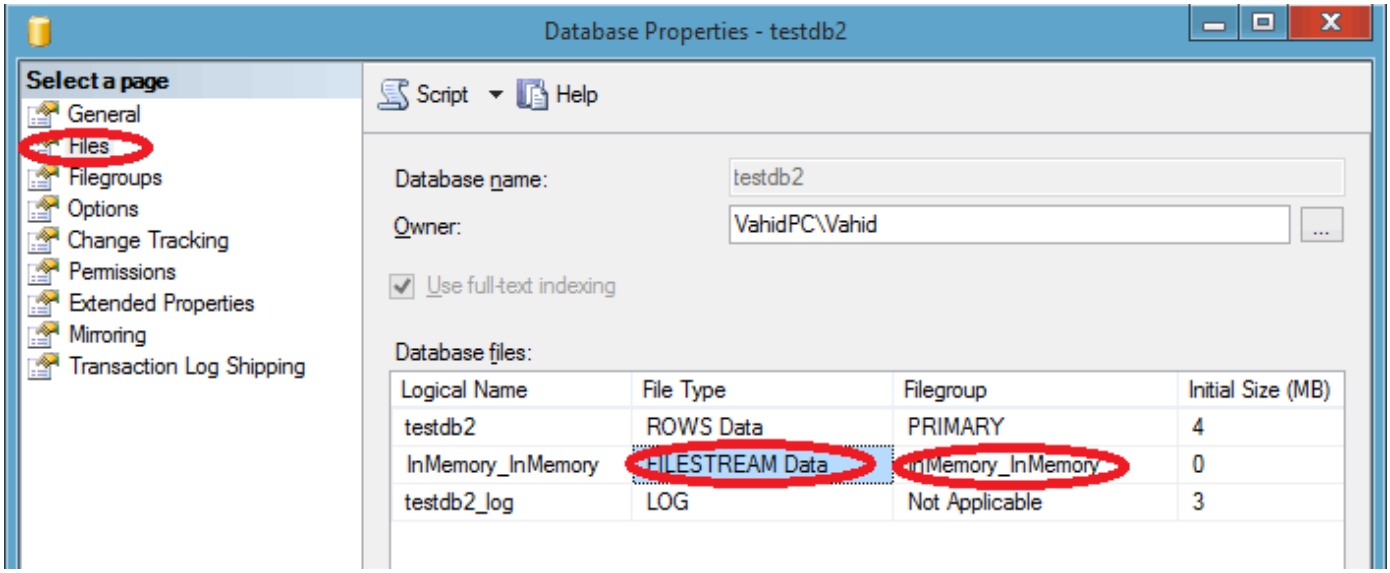
ایجاد یک بانک اطلاعاتی با پشتیبانی از جداول بهینه سازی شده برای حافظه

برای ایجاد جداول بهینه سازی شده برای حافظه، ابتدا نیاز است تا تنظیمات خاصی را به بانک اطلاعاتی آن اعمال کنیم. برای اینکار می‌توان یک بانک اطلاعاتی جدید را به همراه یک filestream filegroup ایجاد کرد که جهت جداول بهینه سازی شده برای حافظه، ضروری است؛ یا اینکه با تغییر یک بانک اطلاعاتی موجود و افزودن filegroup یاد شده نیز می‌توان به این مقصود رسید.
در اینگونه جداول خاص، اطلاعات در حافظه‌ی سیستم ذخیره می‌شوند و برخلاف جداول مبتنی بر دیسک سخت، صفحات اطلاعات وجود نداشته و نیازی نیست تا به کش بافر وارد شوند. برای مقاصد ذخیره سازی نهایی اطلاعات جداول بهینه سازی شده برای حافظه، موتور OLTP درون حافظه‌ای آن، فایل‌های خاصی را به نام checkpoint در یک filestream filegroup ایجاد می‌کند که از آن‌ها جهت ردیابی اطلاعات استفاده خواهد کرد و نحوی ذخیره سازی اطلاعات در آن‌ها از شیوه‌ی با کارآیی بالایی به نام append only mode پیروی می‌کند.
با توجه به متفاوت بودن نحوه‌ی ذخیره سازی نهایی اطلاعات اینگونه جداول و دسترسی به آن‌ها از طریق استریم‌ها، توصیه شده‌است که filestream filegroupهای تهیه شده را در یک SSD یا Solid State Drive قرار دهید.

پس از اینکه بانک اطلاعاتی خود را به روش‌های معمول ایجاد کردید، به برگه‌ی خواص آن در management studio مراجعه کنید. سپس صفحه‌ی file groups را در آن انتخاب کرده و در پایین برگه‌ی آن، در قسمت جدید memory optimized data، بر روی دکمه‌ی Add کلیک کنید. سپس نام دلخواهی را وارد نمائید.



پس از ایجاد یک گروه فایل جدید، به صفحه‌ی `files` خواص بانک اطلاعاتی مراجعه کرده و بر روی دکمه‌ی `Add` کلیک کنید. سپس `File type` این ردیف اضافه شده را از نوع `file stream data` و `file group` آن‌را همان گروه فایلی که پیشتر ایجاد کردیم، تنظیم کنید. در ادامه `logical name` دلخواهی را وارد کرده و در آخر بر روی دکمه‌ی `Ok` کلیک کنید تا تنظیمات مورد نیاز جهت تعاریف جدول بهینه سازی شده برای حافظه به پایان برسد.



این مراحل را توسط دو دستور T-SQL ذیل نیز می‌توان سریعتر انجام داد:

```
USE [master]
GO
ALTER DATABASE [testdb2]
      ADD FILEGROUP [InMemory_InMemory] CONTAINS MEMORY_OPTIMIZED_DATA
GO
ALTER DATABASE [testdb2]
      ADD FILE ( NAME = N'InMemory_InMemory', FILENAME =
N'D:\SQL_Data\MSSQL11.MSSQLSERVER\MSSQL\DATA\InMemory_InMemory' )
      TO FILEGROUP [InMemory_InMemory]
GO
```

ساختار گروه فایل بهینه سازی شده برای حافظه

گروه فایل بهینه سازی شده برای حافظه، دارای چندین دربرگیرنده است که هر کدام چندین فایل را در خود جای خواهند داد:
- Root File که در برگیرنده‌ی متادیتای اطلاعات است.
- Data File که شامل ردیف‌های اطلاعات ثبت شده در جداول بهینه سازی شده‌ی برای حافظه هستند. این ردیف‌ها همواره به انتهای data file اضافه می‌شوند و دسترسی به آن‌ها ترتیبی است. کارآیی IO این روش نسبت به روش دسترسی اتفاقی به مراتب بالاتر است. حداکثر اندازه این فایل 128 مگابایت است و پس از آن یک فایل جدید ساخته می‌شود.
- Delta File شامل ردیف‌هایی است که حذف شده‌اند. به ازای هر ردیف، حداقل اطلاعاتی از آن را در خود ذخیره خواهد کرد؛ شامل ID ردیف حذف شده و شماره تراکنش آن. همانطور که پیشتر نیز ذکر شد، این موتور جدید درون حافظه‌ای، برای یافتن راه چاره‌ای جهت به حداقل رسانی قفل گذاری بر روی اطلاعات، چندین نگارش از ردیف‌ها را به همراه timestamp آن‌ها در خود ذخیره می‌کند. به این ترتیب، هر به روز رسانی به همراه یک حذف و سپس ثبت جدید است. به این ترتیب دیگر بانک اطلاعاتی نیازی نخواهد داشت تا به دنبال رکورد موجود برگردد و سپس اطلاعات آن‌را به روز نماید. این موتور جدید فقط اطلاعات به روز شده را در انتهای رکوردهای موجود با فرمت خود ثبت می‌کند.

ایجاد جداول بهینه سازی شده برای حافظه

پس از آماده سازی بانک اطلاعاتی خود و افزودن گروه فایل استریم جدیدی به آن برای ذخیره سازی اطلاعات جداول بهینه سازی شده برای حافظه، اکنون می‌توانیم اینگونه جداول خاص را در کنار سایر جداول متداول موجود، تعریف و استفاده نمائیم:

```
-- It is not a Memory Optimized
CREATE TABLE tblNormal
(
   [CustomerID] int NOT NULL PRIMARY KEY NONCLUSTERED,
```

```
   [Name] nvarchar(250) NOT NULL,
   CustomerSince DATETIME not NULL
      INDEX [ICustomerSince] NONCLUSTERED
)

--  DURABILITY = SCHEMA_AND_DATA
CREATE TABLE tblMemoryOptimized_Schema_And_Data
(
    [CustomerID] INT NOT NULL
PRIMARY KEY NONCLUSTERED HASH WITH (BUCKET_COUNT = 1000000),
    [Name] NVARCHAR(250) NOT NULL,
    [CustomerSince] DATETIME NOT NULL
INDEX [ICustomerSince] NONCLUSTERED
) WITH (MEMORY_OPTIMIZED = ON, DURABILITY = SCHEMA_AND_DATA)


-- DURABILITY = SCHEMA_ONLY
CREATE TABLE tblMemoryOptimized_Schema_Only
(
    [CustomerID] INT NOT NULL
PRIMARY KEY NONCLUSTERED HASH WITH (BUCKET_COUNT = 1000000),
    [Name] NVARCHAR(250) NOT NULL,
    [CustomerSince] DATETIME NOT NULL
INDEX [ICustomerSince] NONCLUSTERED
) WITH (MEMORY_OPTIMIZED = ON, DURABILITY = SCHEMA_ONLY)
```

در اینجا سه جدول را مشاهده می‌کنید که در بانک اطلاعاتی آماده شده در مرحله‌ی قبل، ایجاد خواهند شد. مورد اول یک جدول معمولی است که از آن برای مقایسه سرعت ثبت اطلاعات با سایر جداول ایجاد شده، استفاده خواهد شد.

همانطور که مشخص است، دو جدول بهینه سازی شده برای حافظه، همان سه ستون جدول معمولی مبتنی بر دیسک سخت را دارا هستند؛ اما با این تفاوت‌ها:

- دارای ویژگی **MEMORY_OPTIMIZED = ON** می‌باشند. به این ترتیب اینگونه جداول نسبت به جداول متداول مبتنی به دیسک سخت متمایز خواهند شد.

- دارای ویژگی **DURABILITY** بوده و توسط مقدار SCHEMA_AND_DATA آن مشخص می‌کنیم که آیا قرار است اطلاعات و ساختار جدول، ذخیره شوند یا تنها قرار است ساختار جدول ذخیره گردد (حالت SCHEMA_ONLY).

- بر روی ستون Id آن‌ها یک hash index ایجاد شده‌است که وجود آن ضروری است و در کل بیش از 8 ایندکس را نمی‌توان تعریف کرد.

برخلاف ایندکس‌های B-tree جداول مبتنی بر سخت دیسک، ایندکس‌های جداول بهینه سازی شده برای حافظه، اطلاعات را تکرار نمی‌کنند. این‌ها صرفا اشاره‌گرهایی هستند به ردیف‌های اصلی اطلاعات. به این معنا که این ایندکس‌ها لاگ نشده و همچنین بر روی سخت دیسک ذخیره نمی‌شوند. کار بازسازی مجدد آن‌ها در اولین بار بازیابی بانک اطلاعاتی و آغاز آن به صورت خودکار انجام می‌شود. به همین جهت مباحثی مانند index fragmentation و نگهداری ایندکس‌ها دیگر در اینجا معنا پیدا نمی‌کنند.

دو نوع ایندکس را در اینجا می‌توان تعریف کرد. اولین آن‌ها hash index است و دومین آن‌ها range index. هش ایندکس‌ها برای حالاتی که در کوئری‌ها از عملگر تساوی استفاده می‌شود بسیار مناسب هستند. برای عملگرهای مقایسه‌ای از ایندکس‌های بازه‌ای استفاده می‌شود.

همچنین باید دقت داشت که پس از ایجاد ایندکس‌ها، دیگر امکان تغییر آن‌ها و یا تغییر ساختار جدول ایجاد شده نیست.

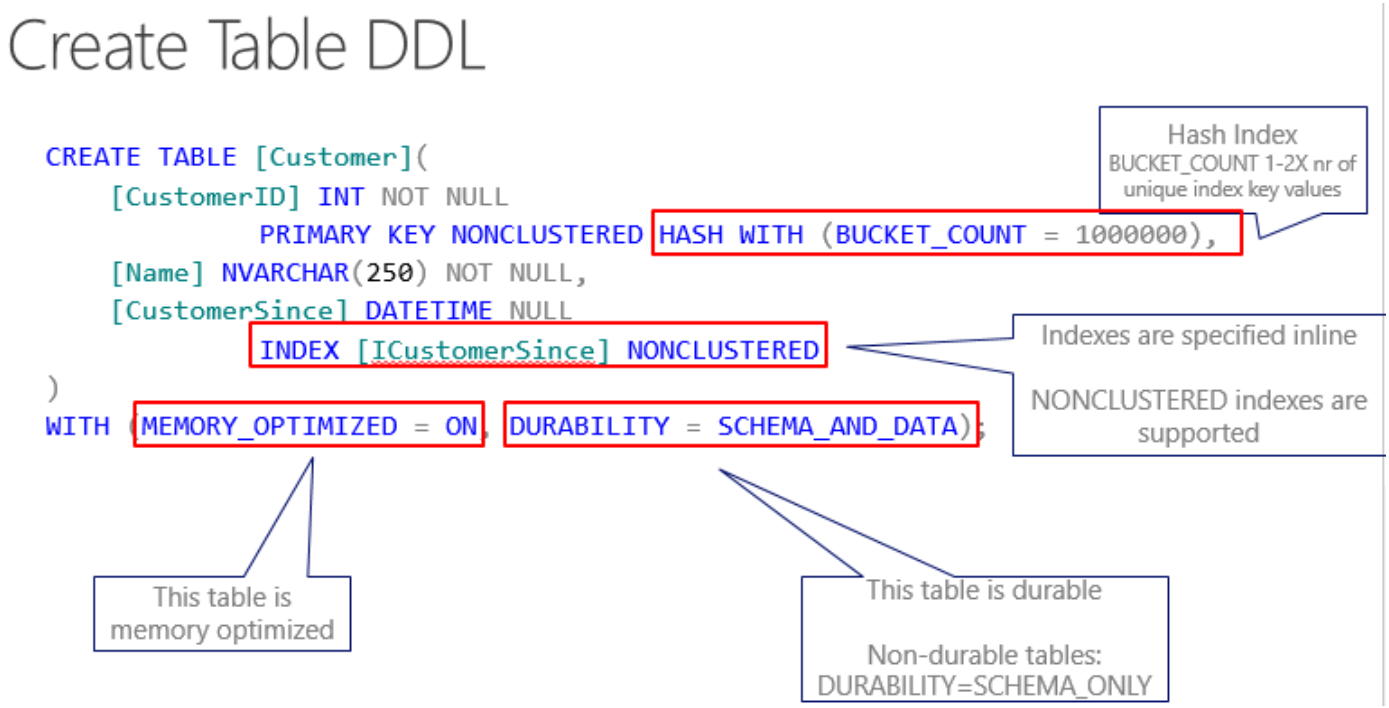
همچنین ایندکس‌های تعریف شده در جداول بهینه سازی شده برای حافظه، تنها بر روی ستون‌هایی غیرنال پذیر از نوع BIN2 collation مانند int و datetime قابل تعریف هستند. برای مثال اگر سعی کنیم بر روی ستون Name ایندکسی را تعریف کنیم، به این خطا خواهیم رسید:

```
Indexes on character columns that do not use a *_BIN2 collation are not supported with indexes on
memory optimized tables.
```

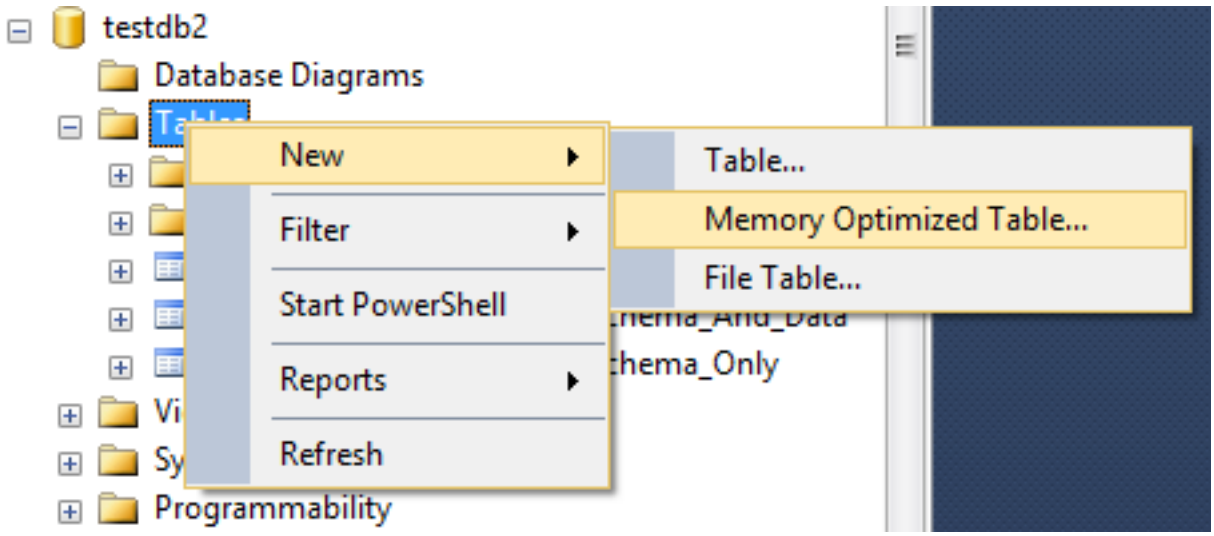
- در حین تعریف هش ایندکس‌ها، مقدار BUCKET_COUNT نیز باید تنظیم شود. هر bucket توسط مقداری که حاصل هش کردن یک ستون است مشخص می‌شود. کلیدهای منحصربفرد دارای هش‌های یکسان در bucketهای یکسانی ذخیره می‌شوند. به همین جهت توصیه شده‌است که حداقل مقدار bucket تعیین شده در اینجا مساوی یا بیشتر از مقدار تعداد کلیدهای منحصربفرد یک جدول باشد؛ مقدار پیش فرض 2 برابر توسط مایکروسافت توصیه شده‌است.

- نوع‌های قابل تعریف ستون‌ها نیز در اینجا به موارد ذیل محدود هستند و جمع طول آن‌ها از 8060 نباید بیشتر شود:

```
bit, tinyint, smallint, int, bigint, money, smallmoney, float, real, datetime, smalldatetime,
datetime2,
date, time, numberic, decimal, char(n),  varchar(n) ,nchar(n),  nvarchar(n), sysname, binary(n),
varbinary(n), and Uniqueidentifier
```



همچنین در management studio، گزینه‌ی جدید memory optimized table -> new نیز اضافه شده‌است و انتخاب آن سبب می‌شود تا قالب T-SQL ایی برای تهیه این نوع جداول، به صورت خودکار تولید گردد.



البته این گزینه تنها برای بانک‌های اطلاعاتی که دارای گروه فایل استریم مخصوص جداول بهینه سازی شده برای حافظه هستند،

فعال می‌باشد.

ثبت اطلاعات در جداول معمولی و بهینه سازی شده برای حافظه و مقایسه کارآیی آن‌ها

در مثال زیر، 100 هزار رکورد را در سه جدولی که پیشتر ایجاد کردیم، ثبت کرده و سپس مدت زمان اجرای هر کدام از مجموعه عملیات را بر حسب میلی ثانیه بررسی می‌کنیم:

```
set statistics time off
SET STATISTICS IO Off
set nocount on
go
-----------------------------

Print 'insert into tblNormal'

DECLARE @start datetime = getdate()
declare @insertCount int = 100000
declare @startId int = 1
declare @customerID int = @startId

while @customerID < @startId + @insertCount
begin
    insert into tblNormal values (@customerID, 'Test', '2013-01-01T00:00:00')
    set @customerID +=1
end

Print DATEDIFF(ms,@start,getdate());
go
-----------------------------

Print 'insert into tblMemoryOptimized_Schema_And_Data'

DECLARE @start datetime = getdate()
declare @insertCount int = 100000
declare @startId int = 1
declare @customerID int = @startId

while @customerID < @startId + @insertCount
begin
    insert into tblMemoryOptimized_Schema_And_Data values (@customerID, 'Test', '2013-01-01T00:00:00')
    set @customerID +=1
end
Print DATEDIFF(ms,@start,getdate());
Go
-----------------------------

Print 'insert into tblMemoryOptimized_Schema_Only'

DECLARE @start datetime = getdate()
declare @insertCount int = 100000
declare @startId int = 1
declare @customerID int = @startId

while @customerID < @startId + @insertCount
begin
    insert into tblMemoryOptimized_Schema_Only values (@customerID, 'Test', '2013-01-01T00:00:00')
    set @customerID +=1
end
Print DATEDIFF(ms,@start,getdate());

Go
```

با این خروجی تقریبی که بر اساس توانمندی‌های سخت افزاری سیستم می‌تواند متفاوت باشد:

```
insert into tblNormal
36423

insert into tblMemoryOptimized_Schema_And_Data
30516

insert into tblMemoryOptimized_Schema_Only
3176
```

و برای حالت select خواهیم داشت:

```
set nocount on
print 'tblNormal'
set statistics time on
select count(CustomerID) from tblNormal
set statistics time off
go
print 'tblMemoryOptimized_Schema_And_Data'
set statistics time on
select count(CustomerID) from tblMemoryOptimized_Schema_And_Data
set statistics time off
go
print 'tblMemoryOptimized_Schema_Only'
set statistics time on
select count(CustomerID) from tblMemoryOptimized_Schema_Only
set statistics time off
go
```

با این خروجی

```
tblNormal
 SQL Server Execution Times:
CPU time = 46 ms,  elapsed time = 52 ms.

tblMemoryOptimized_Schema_And_Data
 SQL Server Execution Times:
CPU time = 32 ms,  elapsed time = 33 ms.

tblMemoryOptimized_Schema_Only
 SQL Server Execution Times:
CPU time = 31 ms,  elapsed time = 30 ms.
```

تاثیر جداول بهینه سازی شده برای حافظه را در 350K inserts بهتر می‌توان با نمونه‌های متداول مبتنی بر دیسک مقایسه کرد.

برای مطالعه بیشتر

Getting started with SQL Server 2014 In-Memory OLTP

Introduction to SQL Server 2014 CTP1 Memory-Optimized Tables

Overcoming storage speed limitations with Memory-Optimized Tables for SQL Server

Memory-optimized Table – Day 1 Test

Memory-Optimized Tables – Insert Test

Memory Optimized Table – Insert Test ...Again

نظرات خوانندگان

نویسنده: علی رضایی
تاریخ: ۱۳۹۳/۰۳/۱۲ ۱۹:۲۲

با سلام و تشکر فراوان جهت این آموزش.
میشه لطفاً بررسی کنید چرا نتیجه نهایی برای من متفاوت شده، طوری که زمان جستجو برای جدول نرمال کمتره!
راستی زمان ورود اطلاعات (100 رکورد) شبیه به زمان‌های مثال شما است و من از یک هارد SSD Corsair استفاده میکنم.

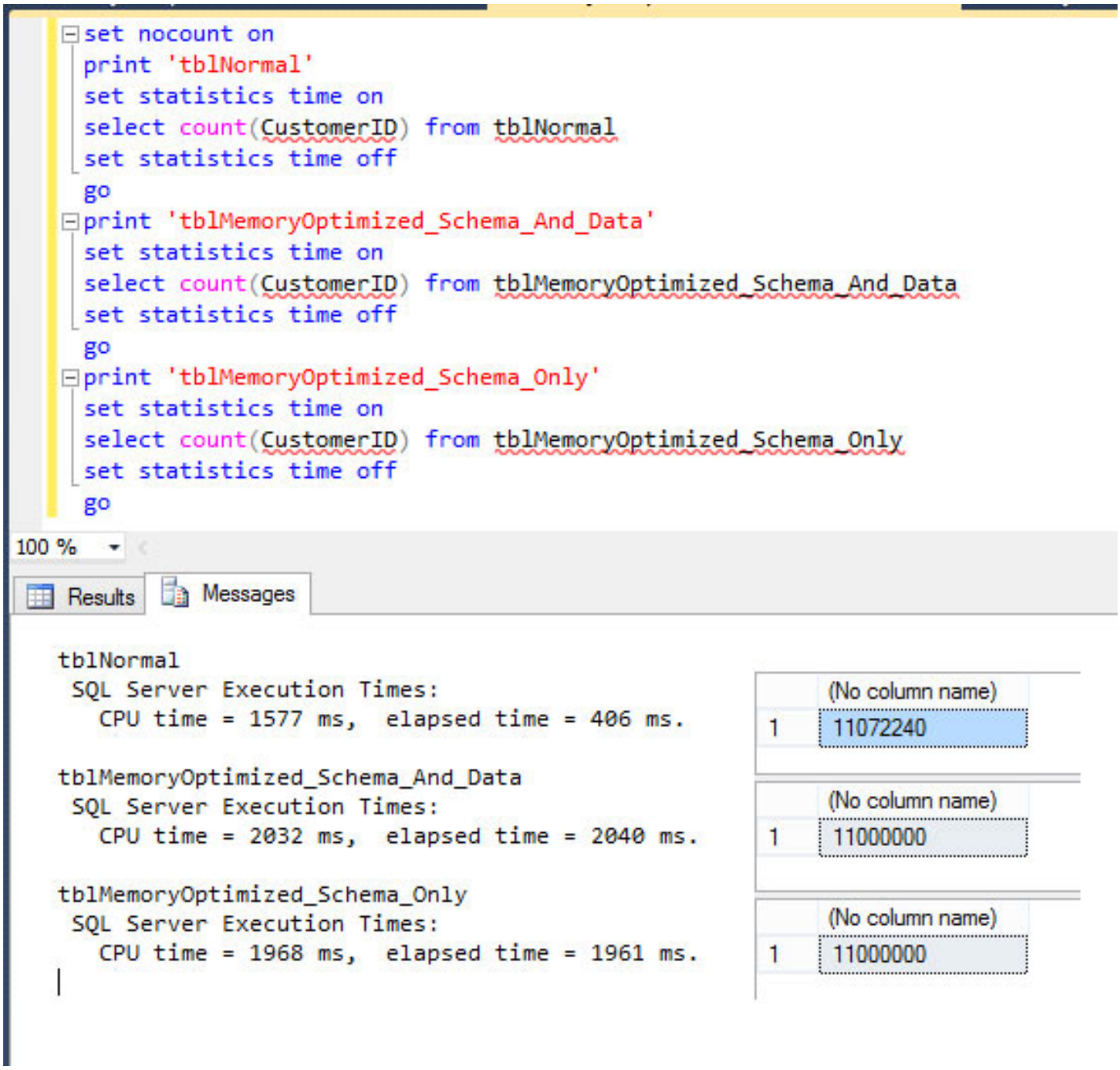
```
set nocount on
print 'tblNormal'
set statistics time on
select count(CustomerID) from tblNormal
set statistics time off
go
print 'tblMemoryOptimized_Schema_And_Data'
set statistics time on
select count(CustomerID) from tblMemoryOptimized_Schema_And_Data
set statistics time off
go
print 'tblMemoryOptimized_Schema_Only'
set statistics time on
select count(CustomerID) from tblMemoryOptimized_Schema_Only
set statistics time off
go
```

100 %

Results Messages

```
tblNormal

 SQL Server Execution Times:
   CPU time = 16 ms,  elapsed time = 6 ms.
tblMemoryOptimized_Schema_And_Data

 SQL Server Execution Times:
   CPU time = 15 ms,  elapsed time = 18 ms.
tblMemoryOptimized_Schema_Only

 SQL Server Execution Times:
   CPU time = 16 ms,  elapsed time = 17 ms.
```

ویرایش:
تست جدید با بیش از 11 میلیون رکورد و خاموش کردن فایروال کومودو و خاموش کردن windows defender



نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۳/۱۳ ۰:۳۷

- بستگی دارد. چقدر سیستم شما RAM دارد. مشخصات CPU آن چیست و خیلی از مسایل جانبی دیگر (مانند تحت نظر بودن پوشه‌های فایل استریم‌ها توسط آنتی ویروس یا خیر).
- هدف اصلی از این تحولات، کارهای همزمان و بررسی تفاوت بهبود در کارهای چند ریسمانی و چند کاربری است. مثال فوق یک مثال تک ریسمانی است.
- یک آزمایش را باید چندبار تکرار کرد و بعد میانگین گرفت. برای تکرار هم نیاز است کش‌های سیستم را هربار حذف کنید:

```
set nocount on
CHECKPOINT
DBCC FREEPROCCACHE()
DBCC DROPCLEANBUFFERS
```

- بعد از پایان Insert تعداد ردیف‌های زیاد در یک جدول درون حافظه‌ای باید اطلاعات آماری آن‌را دستی به روز کرد (^).

```
UPDATE STATISTICS tblNormal WITH FULLSCAN, NORECOMPUTE;
UPDATE STATISTICS tblMemoryOptimized_Schema_And_Data WITH FULLSCAN, NORECOMPUTE;
UPDATE STATISTICS tblMemoryOptimized_Schema_Only WITH FULLSCAN, NORECOMPUTE;
```

این دستورات باید قبل از اجرای سه کوئری آخر قرار گیرند.

- تعداد bucket count هم مهم است. در اینجا فرمولی برای محاسبه آن ارائه شده:

```
Select  POWER(
    2,
    CEILING( LOG( COUNT( 0)) / LOG( 2)))
    AS 'BUCKET_COUNT'
FROM
  (SELECT DISTINCT CustomerId
      FROM tblMemoryOptimized_Schema_And_Data) T
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۳/۱۴ ۰:۵۹

ارزش واقعی جداول درون حافظه‌ای را باید با اعمال تراکنش‌های همزمان و بررسی میزان پاسخگویی سیستم بررسی کرد و نه صرفا با یک آزمایش ساده تک ریسمانی. برای این منظور برنامه‌ای به نام ostress.exe توسط مایکروسافت تهیه شده‌است که امکان انجام یک چنین آزمایشاتی را میسر می‌کند. برای دریافت آن به آدرس‌های ذیل مراجعه کنید:

RML Utilities X64

RML Utilities X86

که نهایتا در این مسیر C:\Program Files\Microsoft Corporation\RMLUtils نصب خواهد شد.

سپس در خط فرمان این سه دستور را امتحان کنید:

```
-- Insert 10000 records using 20 threads, Repeat Execution 3 times

-- disk-based
"C:\Program Files\Microsoft Corporation\RMLUtils\ostress.exe" -n20 -r3 -S. -E -dTestdb2 -q -Q"set
statistics time off; SET STATISTICS IO Off; set nocount on; DECLARE @start datetime = getdate();
declare @insertCount int = 10000; declare @startId int = 1; while @startId < @insertCount begin insert
into tblNormal values ('Test', '2013-01-01T00:00:00') set @startId +=1 end; Print
DATEDIFF(ms,@start,getdate());" -oc:\temp\output

-- memory-optimized, tblMemoryOptimized_Schema_And_Data
"C:\Program Files\Microsoft Corporation\RMLUtils\ostress.exe" -n20 -r3 -S. -E -dTestdb2 -q -Q"set
statistics time off; SET STATISTICS IO Off; set nocount on; DECLARE @start datetime = getdate();
declare @insertCount int = 10000; declare @startId int = 1; while @startId < @insertCount begin insert
into tblMemoryOptimized_Schema_And_Data values ('Test', '2013-01-01T00:00:00') set @startId +=1 end;
Print DATEDIFF(ms,@start,getdate());" -oc:\temp\output

-- memory-optimized, tblMemoryOptimized_Schema_Only
"C:\Program Files\Microsoft Corporation\RMLUtils\ostress.exe" -n20 -r3 -S. -E -dTestdb2 -q -Q"set
statistics time off; SET STATISTICS IO Off; set nocount on; DECLARE @start datetime = getdate();
declare @insertCount int = 10000; declare @startId int = 1;  while @startId <  @insertCount begin
insert into tblMemoryOptimized_Schema_Only values ('Test', '2013-01-01T00:00:00') set @startId +=1 end;
Print DATEDIFF(ms,@start,getdate());" -oc:\temp\output
```

زمانیکه را که در پایان کار نمایش می‌دهد، مبنای واقعی مقایسه است.

البته برای اجرای این دستورات نیاز است فیلد CustomerID را identity تعریف کنید (در هر سه جدول مطلب جاری).

```
-- It is not Memory Optimized
CREATE TABLE tblNormal
(
       [CustomerID] int identity NOT NULL PRIMARY KEY NONCLUSTERED,
       [Name] nvarchar(250) NOT NULL,
   CustomerSince DATETIME not NULL
      INDEX [ICustomerSince] NONCLUSTERED
)

--  DURABILITY = SCHEMA_AND_DATA
CREATE TABLE tblMemoryOptimized_Schema_And_Data
(
    [CustomerID] INT  identity NOT NULL
PRIMARY KEY NONCLUSTERED HASH WITH (BUCKET_COUNT = 131072),
    [Name] NVARCHAR(250) NOT NULL,
    [CustomerSince] DATETIME NOT NULL
INDEX [ICustomerSince] NONCLUSTERED
) WITH (MEMORY_OPTIMIZED = ON, DURABILITY = SCHEMA_AND_DATA)


-- DURABILITY = SCHEMA_ONLY
CREATE TABLE tblMemoryOptimized_Schema_Only
(
    [CustomerID] INT  identity NOT NULL
PRIMARY KEY NONCLUSTERED HASH WITH (BUCKET_COUNT = 131072),
    [Name] NVARCHAR(250) NOT NULL,
    [CustomerSince] DATETIME NOT NULL
INDEX [ICustomerSince] NONCLUSTERED
) WITH (MEMORY_OPTIMIZED = ON, DURABILITY = SCHEMA_ONLY)
```

نویسنده: حمید رضا نظری
تاریخ: ۱۳۹۳/۰۹/۱۰ ۱۰:۴۰

موقع اجرای دستور ALTER DATABASE SQL2014_Demo

ADD FILEGROUP MemFG CONTAINS MEMORY_OPTIMIZED_DATA

GO

این خطا رو میده لطفا راهنمایم کنید

Msg 534, Level 15, State 72, Line 37

FILEGROUP ... CONTAINS MEMORY_OPTIMIZED_DATA' failed because it is not supported in the edition of this SQL'

Server instance 'TFS-SERVER'. See books online for more details on feature support in different SQL Server

.editions

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۹/۱۰ ۱۰:۵۰

- پیام « it is not supported » به معنای عدم پشتیبانی این قابلیت در نگارش SQL Server ایی است که از آن استفاده می‌کنید.
- و فقط در نگارش‌های editions SQL Server *2014* **Enterprise** , **Developer** , and **Evaluation** پشتیبانی می‌شود. برای مثال نگارش Standard این قابلیت را ندارد.
- هنگام نصب هم باید گزینه‌ی «Database Engine Services -> install support for In-Memory OLTP engine» انتخاب شده باشد.

نویسنده: حمید رضا نظری
تاریخ: ۱۳۹۳/۰۹/۱۰ ۱۳:۲۴

من نگارش Developer 64 را نصب کردم ولی باز خطا میده

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۹/۱۰ ۱۳:۵۵

مباحث جداول بهینه سازی شده برای حافظه، روی این سیستم تهیه شدند:

```
select @@VERSION
```

با این خروجی (که مشخصات instance جاری را بر می‌گرداند؛ در مثال شما 'TFS-SERVER' ' instance باید بررسی شود که چه نگارشی دارد)

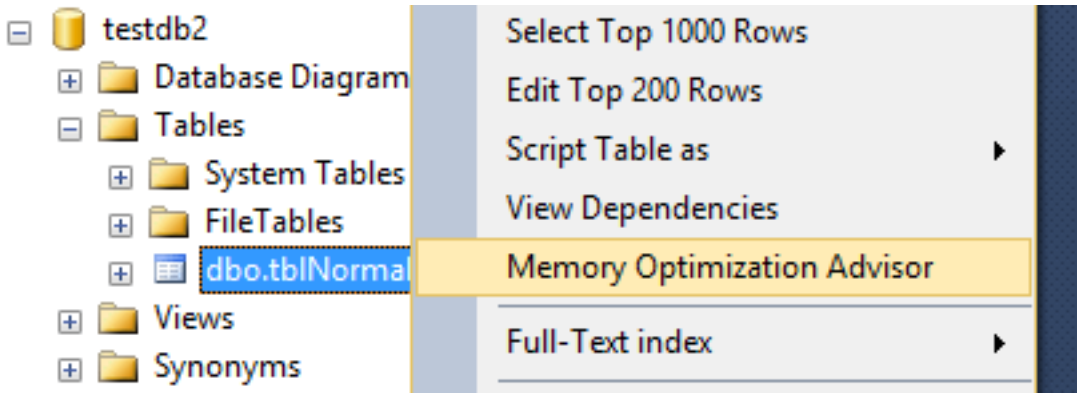
```
Microsoft SQL Server 2014 - 12.0.2254.0 (X64)
Jul 25 2014 18:52:51
Copyright (c) Microsoft Corporation
Developer Edition (64-bit) on Windows NT 6.3 <X64> (Build 9600: )
```

عنوان: **ابزارهای مهاجرت به OLTP درون حافظه‌ای در SQL Server 2014**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۳/۱۲ ۲۱:۵
آدرس: www.dotnettips.info
گروه‌ها: SQL Server, In-Memory OLTP

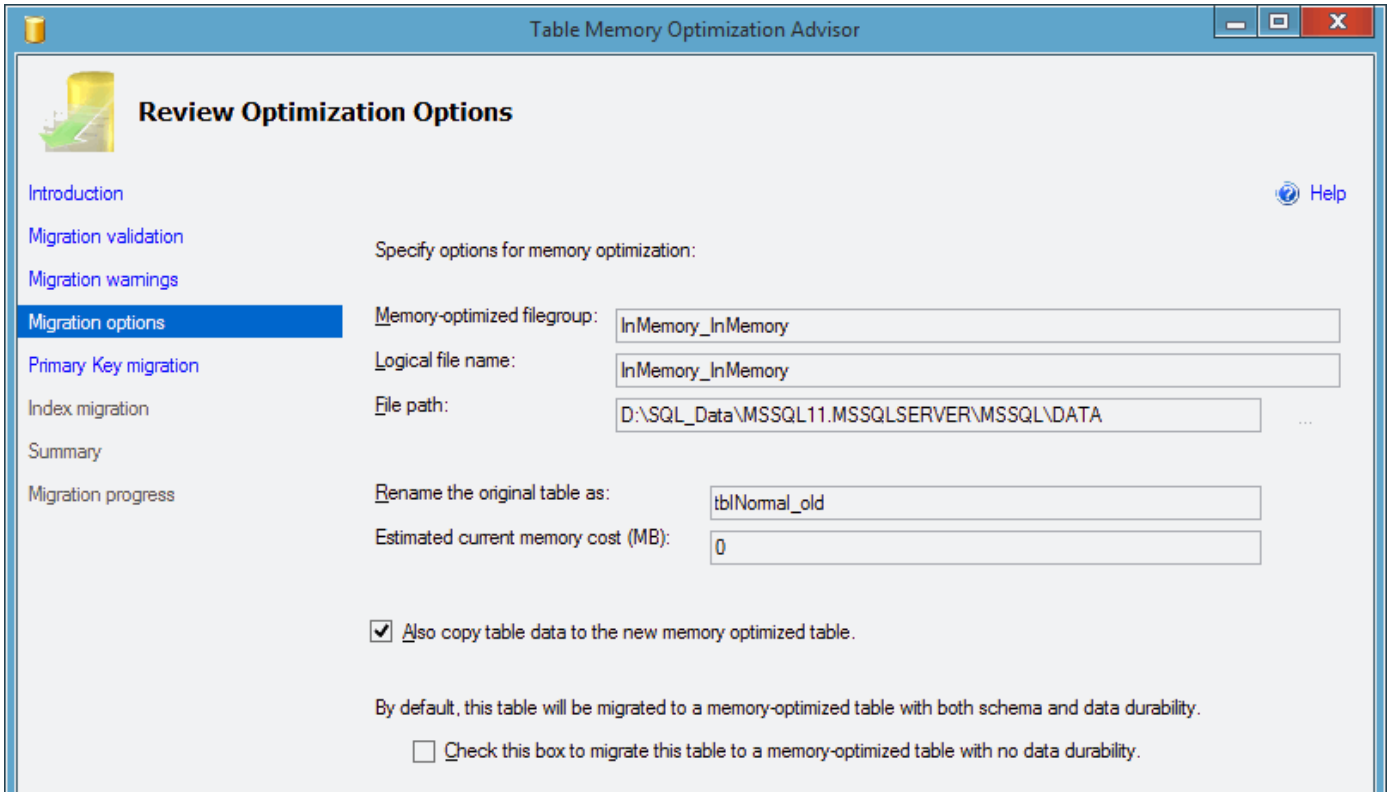
در SQL Server 2014، به Management studio آن ابزارهای جدیدی اضافه شده‌اند تا کار تبدیل و مهاجرت جداول معمولی، به جداول بهینه سازی شده‌ی برای حافظه را ساده‌تر کنند. برای مثال امکان جدیدی به نام Transaction performance collector جهت بررسی کارآیی تراکنش‌های جداول و یا رویه‌های ذخیره شده در محیط کاری جاری، طراحی شده‌است. پس از آن، این اطلاعات را آنالیز کرده و بر اساس میزان استفاده از آن‌ها، توصیه‌هایی را در مورد مهاجرت یا عدم نیاز به مهاجرت به سیستم جدید OLTP درون حافظه‌ای ارائه می‌دهد. در ادامه این ابزارهای جدید را بررسی خواهیم کرد.

ابزار Memory Optimization Advisor

Memory Optimization Advisor یک Wizard مانند است که از آن برای گرفتن مشاوره در مورد تبدیل جداول موجود مبتنی بر دیسک سخت، به نمونه‌های بهینه سازی شده برای حافظه می‌توان استفاده کرد. کار آن بررسی ساختار جداولی است که قصد مهاجرت آن‌ها را دارید. برای مثال همانطور که پیشتر نیز عنوان شد ، جداول بهینه سازی شده برای حافظه محدودیت‌هایی دارند؛ مثلا نباید کلید خارجی داشته باشند. این Wizard یک چنین مواردی را آنالیز کرده و گزارشی را ارائه می‌دهد. پس از اینکه مراحل آن‌را به پایان رساندید و مشکلاتی را که گزارش می‌دهد، برطرف نمودید، کد تبدیل جدول را نیز به صورت خودکار تولید می‌کند.
برای دسترسی به آن، فقط کافی است بر روی نام جدول خود کلیک راست کرده و گزینه‌ی memory optimization advisor را انتخاب کنید.

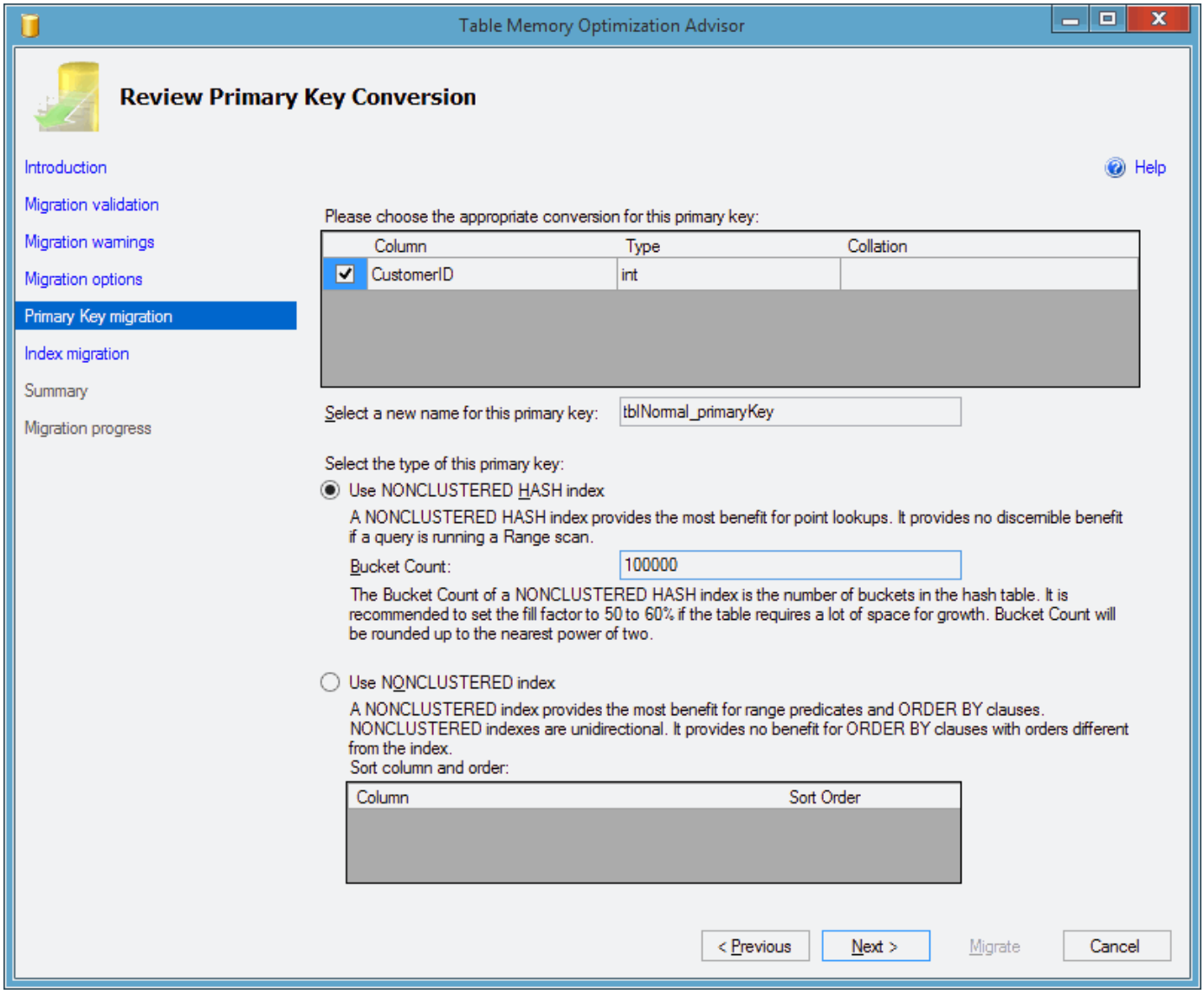


در دو قسمت اول این Wizard، کار بررسی ساختار جدول در حال مهاجرت صورت می‌گیرد. اگر نوع داده‌ای در آن پشتیبانی نشود یا قیود ویژه‌ای در آن تعریف شده باشند، گزارشی را جهت رفع، دریافت خواهید کرد. پس از رفع آن، به صفحه‌ی گزینه‌های مهاجرت می‌رسیم:

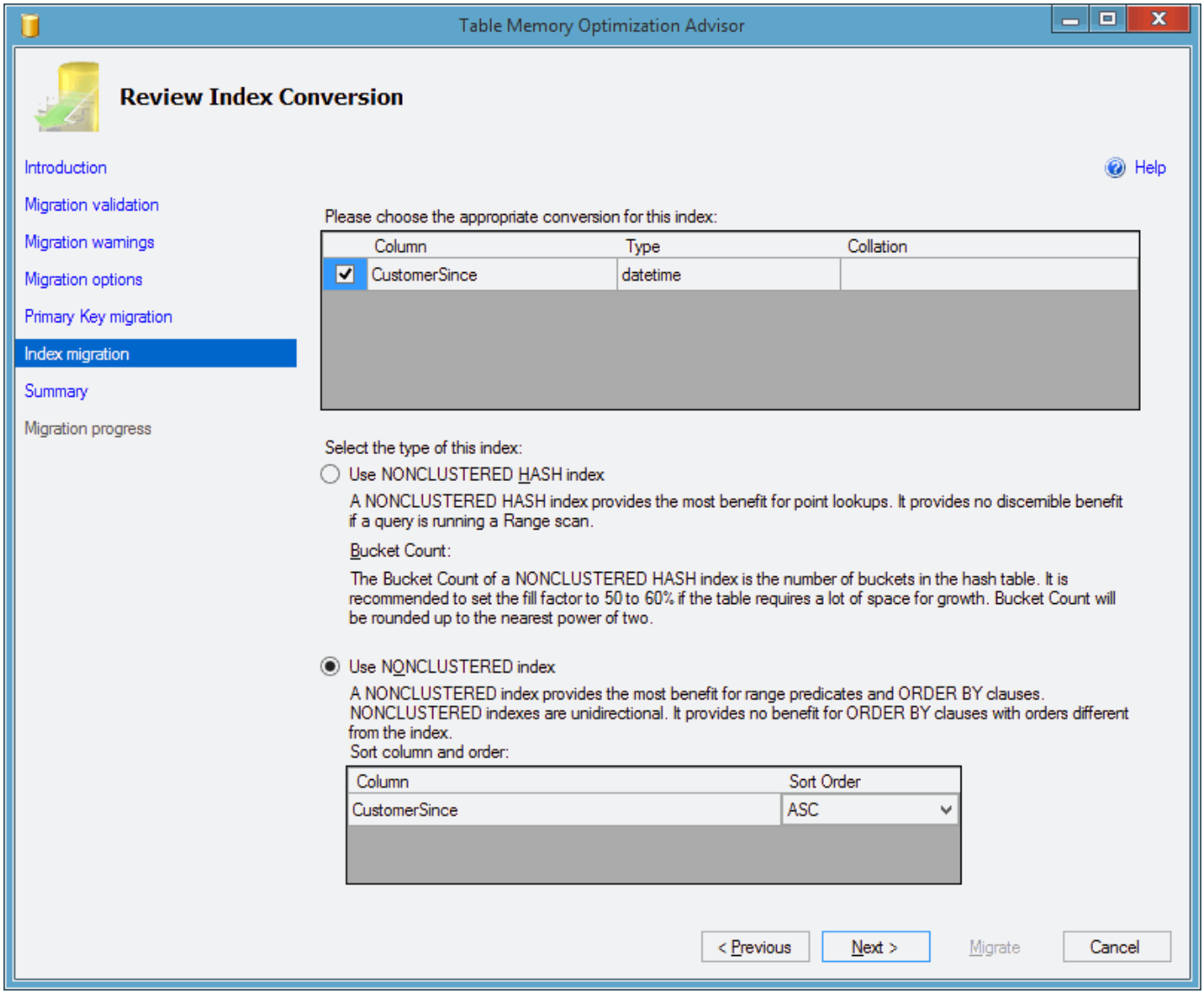


همانطور که ملاحظه می‌کنید، گروه فایل ایجاد شده در قسمت قبل ، به صورت خودکار انتخاب شده‌است.
در ادامه می‌توان نام دیگری را برای جدول مبتنی بر دیسک وارد کرد. در اینجا به صورت خودکار کلمه‌ی old به آخر نام جدول اضافه شده‌است. در حین تولید جدول جدید بهینه سازی شده‌ی بر اساس ساختار جدول فعلی، این جدول قدیمی به صورت خودکار تغییر نام خواهد یافت و کلیه اطلاعات آن حفظ می‌شود.
همچنین تخمینی را نیز از مقدار حافظه‌ی مورد نیاز برای نگهداری این جدول جدید درون حافظه‌ای نیز ارائه می‌دهد. در این مثال چون رکوردی در جدول انتخابی وجود نداشته‌است، تخمین آن صفر است. عدد ارائه شده توسط آن بسیار مهم است و باید به همین میزان برای سیستم خود حافظه تهیه نمائید و یا از حافظه‌ی موجود استفاده کنید.
در پایین صفحه می‌توان انتخاب کرد که آیا داده‌های جدول فعلی، به جدول درون حافظه‌ای انتقال یابند یا خیر. به علاوه نوع ماندگاری اطلاعات آن نیز قابل تنظیم است. اگر گزینه‌ی آخر را انتخاب کنید به معنای حالت SCHEMA_ONLY است. حالت پیش فرض آن SCHEMA_AND_DATA می‌باشد که در قسمت‌های قبل بیشتر در مورد آن بحث شد.

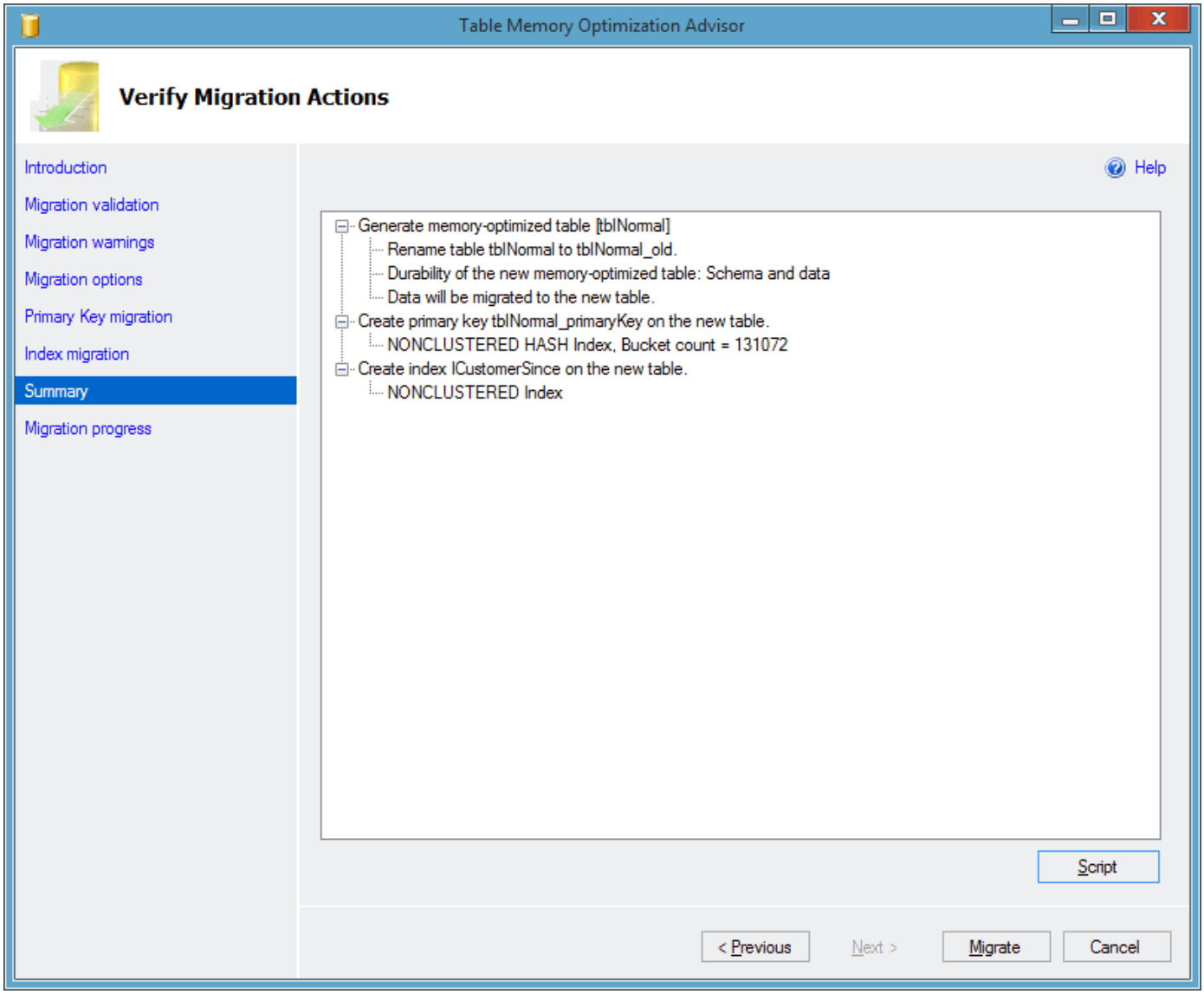
در دو صفحه‌ی بعد، کار انتخاب hash index و range index انجام می‌شود:



در اینجا hash index بر روی فیلد ID تولید شده‌است، به همراه تعیین bucket count آن و در صفحه‌ی بعدی range index بر روی فیلد تاریخ تعریف گردیده‌است:



در آخر می‌توان با کلیک بر روی دکمه‌ی Script، صرفا دستورات T-SQL تغییر ساختار جدول را دریافت کرد و یا با کلیک بر روی دکمه‌ی migrate به صورت خودکار کلیه موارد تنظیم شده را اجرا نمود.



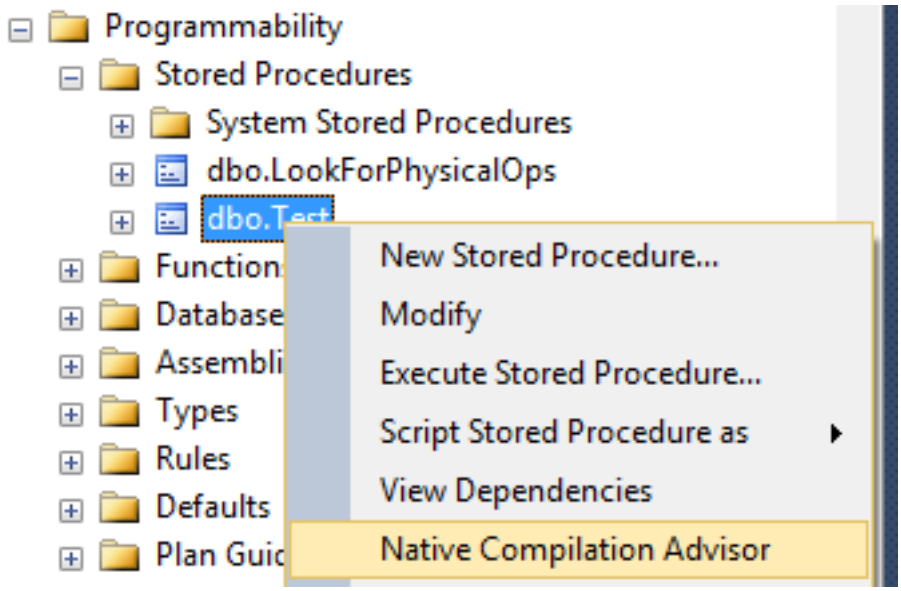
خلاصه‌ی این مراحل که توسط دکمه‌ی Script آن تولید می‌شود، به صورت زیر است:

```
USE [testdb2]
GO

EXEC dbo.sp_rename @objname = N'[dbo].[tblNormal]', @newname = N'tblNormal_old', @objtype = N'OBJECT'
GO

USE [testdb2]
GO

SET ANSI_NULLS ON
GO

CREATE TABLE [dbo].[tblNormal]
(
[CustomerID] [int] NOT NULL,
[Name] [nvarchar](250) COLLATE Persian_100_CI_AI NOT NULL,
[CustomerSince] [datetime] NOT NULL,

INDEX [ICustomerSince] NONCLUSTERED
(
[CustomerSince] ASC
),
CONSTRAINT [tblNormal_primaryKey] PRIMARY KEY NONCLUSTERED HASH
(
[CustomerID]
)WITH ( BUCKET_COUNT = 131072)
```

```
)WITH ( MEMORY_OPTIMIZED = ON , DURABILITY = SCHEMA_AND_DATA )

GO

INSERT INTO [testdb2].[dbo].[tblNormal] ([CustomerID], [Name], [CustomerSince]) SELECT [CustomerID],
[Name], [CustomerSince] FROM [testdb2].[dbo].[tblNormal_old]

GO
```

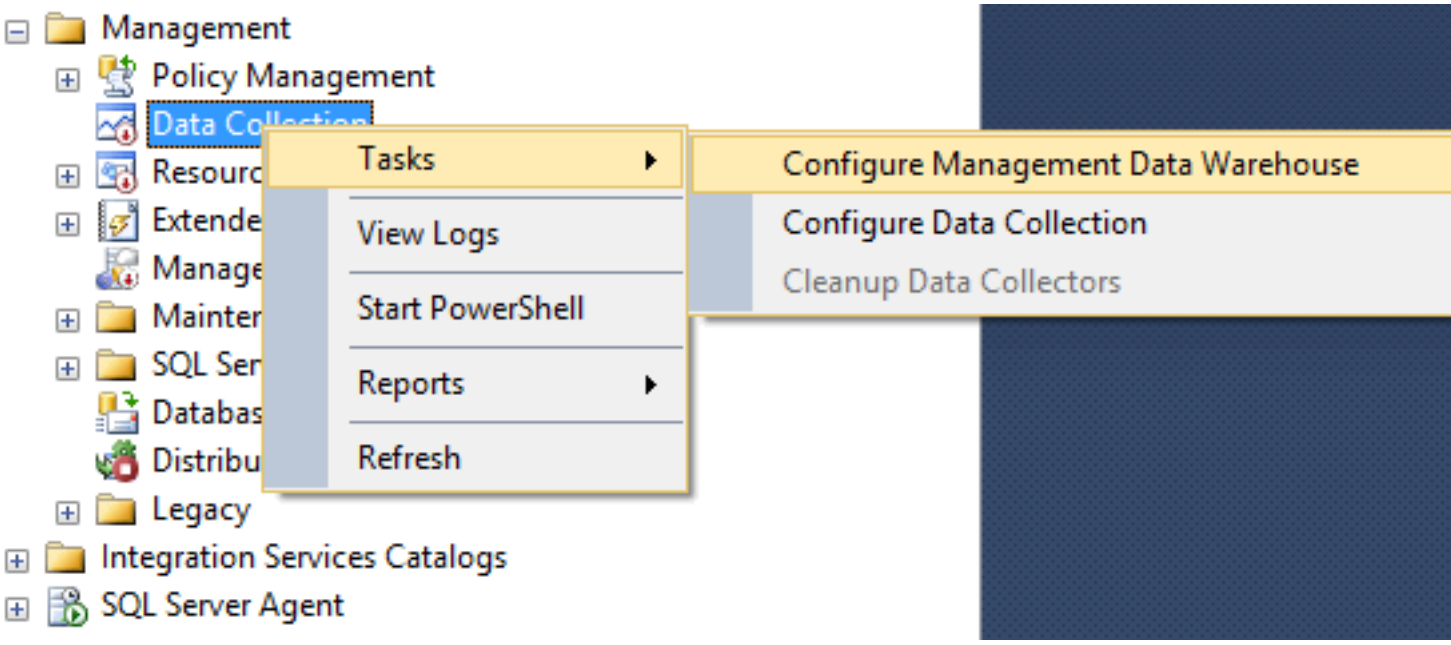
که در آن ابتدا کار تغییر نام جدول قبلی صورت می‌گیرد. سپس یک جدول جدید با ویژگی MEMORY_OPTIMIZED = ON را ایجاد می‌کند. در ساختار این جدول، hash index و range index تعریف شده، قابل مشاهده هستند. در آخر نیز کلیه اطلاعات جدول قدیمی را به جدول جدید منتقل می‌کند.

علاوه بر memory optimization advisor مخصوص جداول، ابزار دیگری نیز به نام **Native compilation advisor** برای آنالیز رویه‌های ذخیره شده تهیه شده‌است:

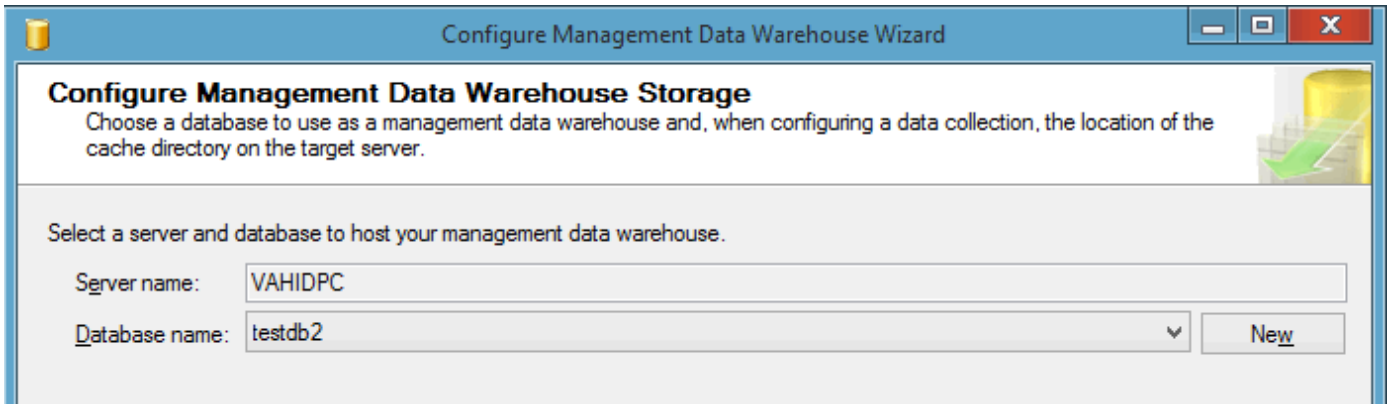


آیا سیستم فعلی ما واقعا نیازی به ارتقاء به جداول درون حافظه‌ای دارد؟

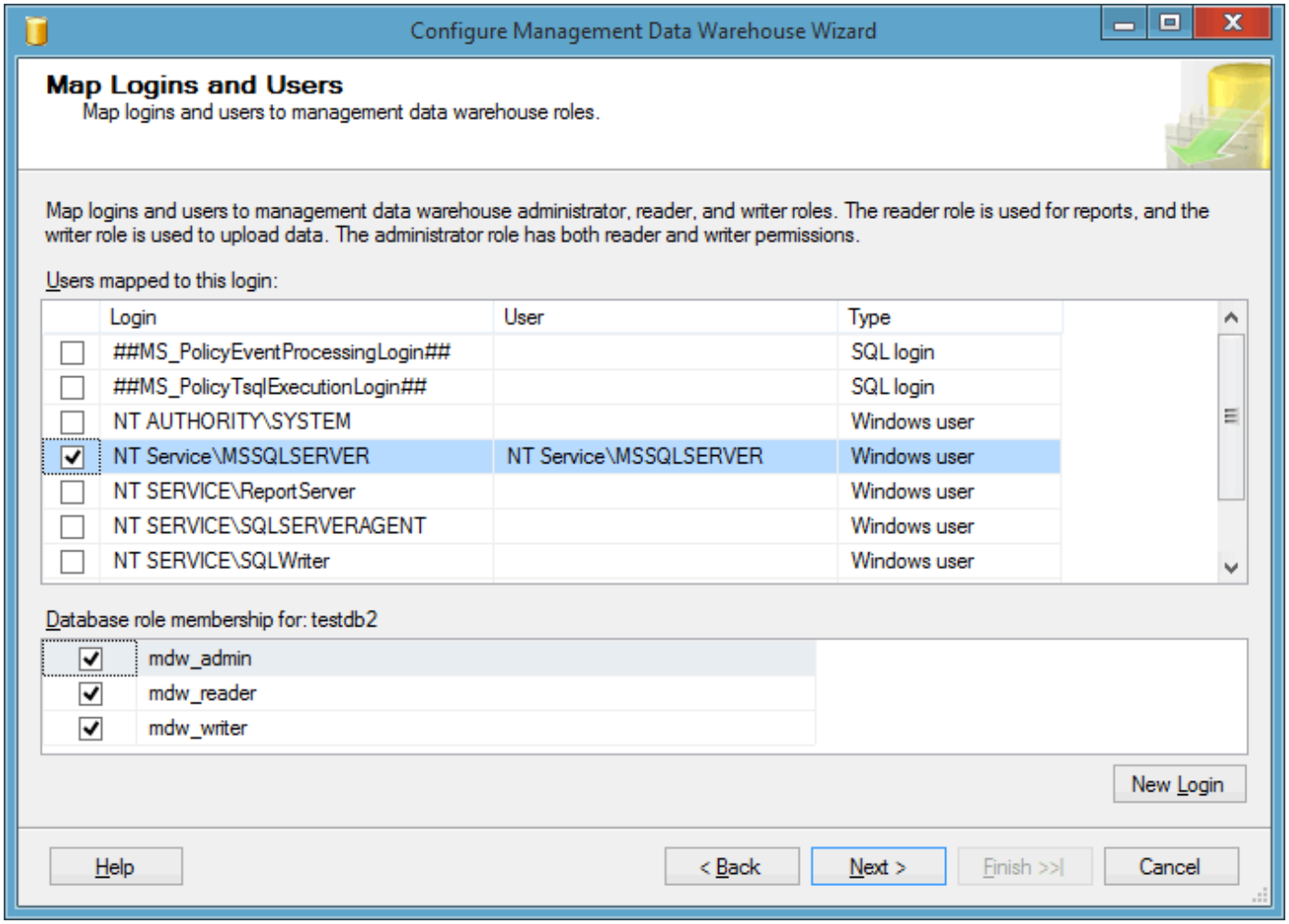
تا اینجا در مورد نحوه‌ی ایجاد جداول درون حافظه‌ای و یا نحوه‌ی تبدیل جداول موجود را به ساختار جدید بررسی کردیم. ولی آیا واقعا یک چنین تغییراتی برای ما سودمند هستند؟ برای پاسخ دادن به این سؤال ابزاری به نام AMR به management studio 2014 اضافه شده‌است (Analyze, Migrate, Report). کار آن تحت نظر قرار دادن جداول و رویه‌های ذخیره شده‌ی بانک اطلاعاتی است و سپس بر اساس بار سیستم، تعداد درخواست‌های همزمان و میزان استفاده از جداول و تراکنش‌های مرتبط با آن‌ها، گزارشی را ارائه می‌دهد. بر این اساس بهتر می‌توان تصمیم گرفت که کدام جداول بهتر است به جداول درون حافظه‌ای تبدیل شوند.
برای تنظیم آن باید مراحل ذیل طی شوند:
در Management Studio، به برگه‌ی Object Explorer آن مراجعه کنید. سپس پوشه‌ی Management آن‌را یافته و بر روی گزینه‌ی Data Collection کلیک راست نمائید:



در اینجا گزینه‌ی Configure Management Data Warehouse را انتخاب نمائید. در صفحه‌ی باز شده، ابتدا بانک اطلاعاتی مدنظر را انتخاب نمائید. همچنین بهتر است بر روی دکمه‌ی new کلیک کرده و یک بانک اطلاعاتی جدید را برای آن ایجاد نمائید، تا دچار تداخل اطلاعاتی و ساختاری نگردد:

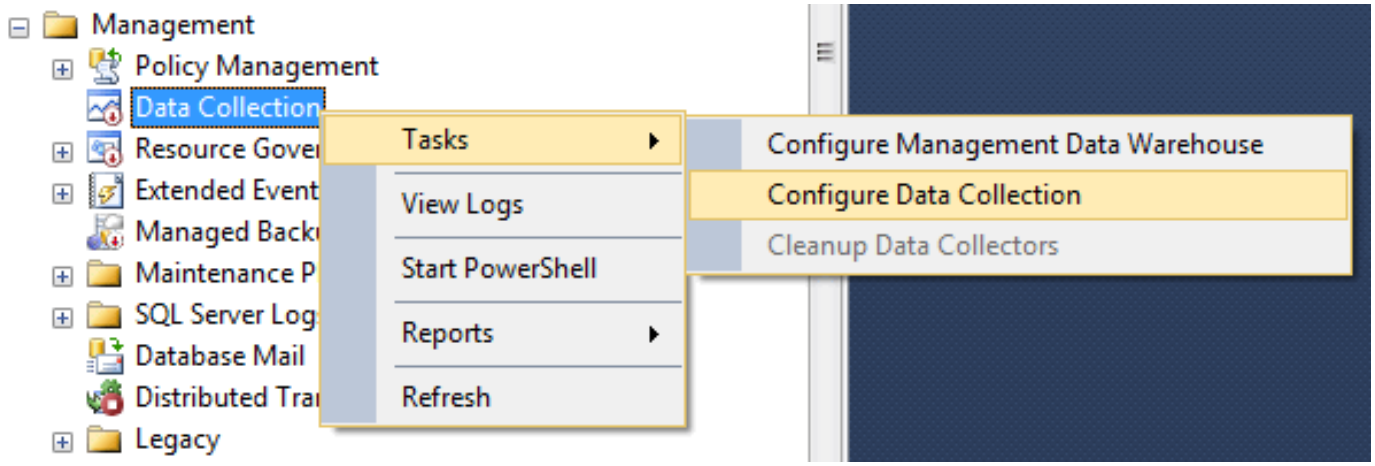


در ادامه نام کاربری را که قرار است کار مدیریت ثبت و جمع آوری اطلاعات را انجام دهد، به همراه نقش‌های آن انتخاب نمائید:



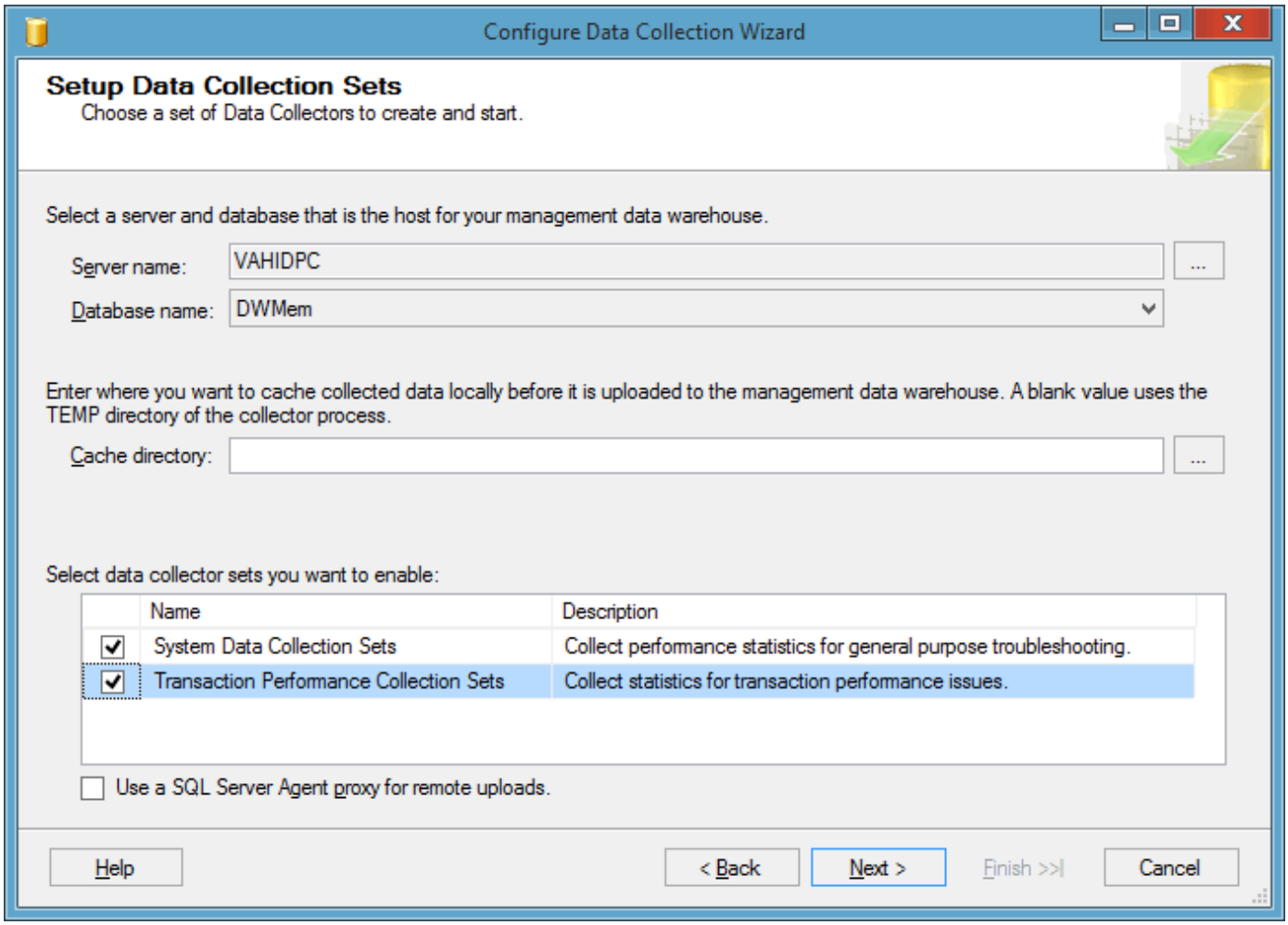
و در آخر در صفحه‌ی بعدی بر روی دکمه‌ی Finish کلیک کنید.

پس از ایجاد و انتخاب بانک اطلاعاتی Management Data Warehouse، نوبت به تنظیم گزینه‌های جمع آوری اطلاعات است:



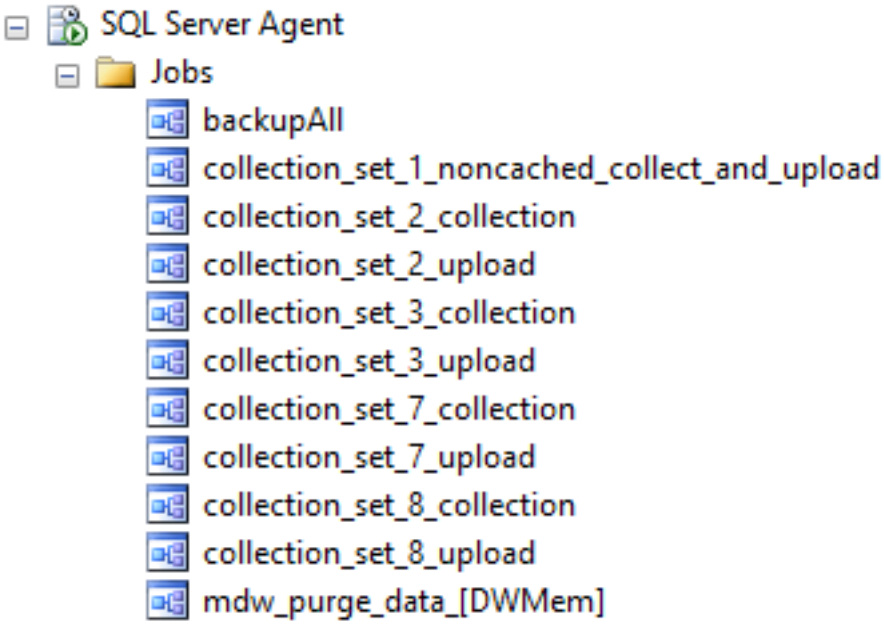
در اینجا ابتدا سرور جاری را انتخاب کنید. پس از آن به صورت خودکار در لیست بانک‌های اطلاعاتی قابل انتخاب، تنها همان بانک

اطلاعاتی جدیدی را که برای مرحله‌ی قبل ایجاد کردیم، می‌توان مشاهده کرد.



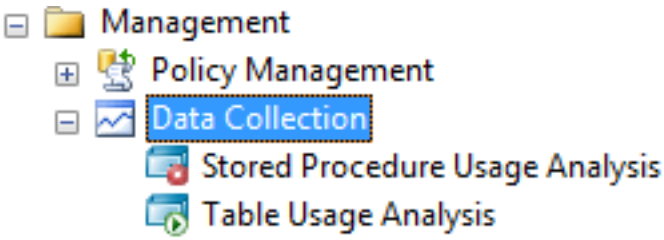
در صفحه‌ی بعد، گزینه‌ی «Transaction Performance Collection Sets» را انتخاب نمائید که دقیقا گزینه‌ی مدنظر ما جهت یافتن آماری از وضعیت تراکنش‌های سیستم است.
در ادامه بر روی گزینه‌های next و finish کلیک کنید تا کار تنظیمات به پایان برسد.

اکنون اگر به لیست وظایف تعریف شده در SQL Server agent مراجعه کنید، می‌توانید، وظایف مرتبط با جمع آوری داده‌ها را نیز مشاهده نمائید:

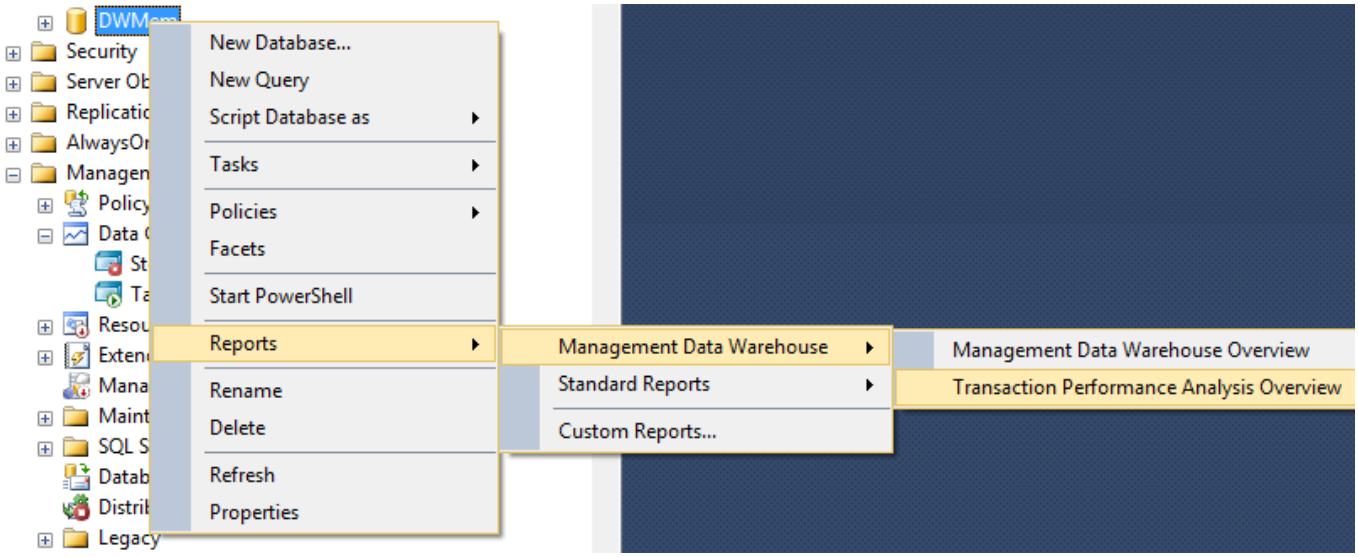


وظایف Stored Procedure Usage Analysis هر نیم ساعت یکبار و وظایف Table Usage Analysis هر 15 دقیقه یکبار اجرا می‌شوند. البته امکان اجرای دستی این وظایف نیز مانند سایر وظایف SQL Server وجود دارند.

همچنین در پوشه‌ی management، گزینه‌ی Data collection نیز دو زیر شاخه اضافه شده‌اند که نمایانگر آنالیز میزان مصرف جداول و رویه‌های ذخیره شده می‌باشند:

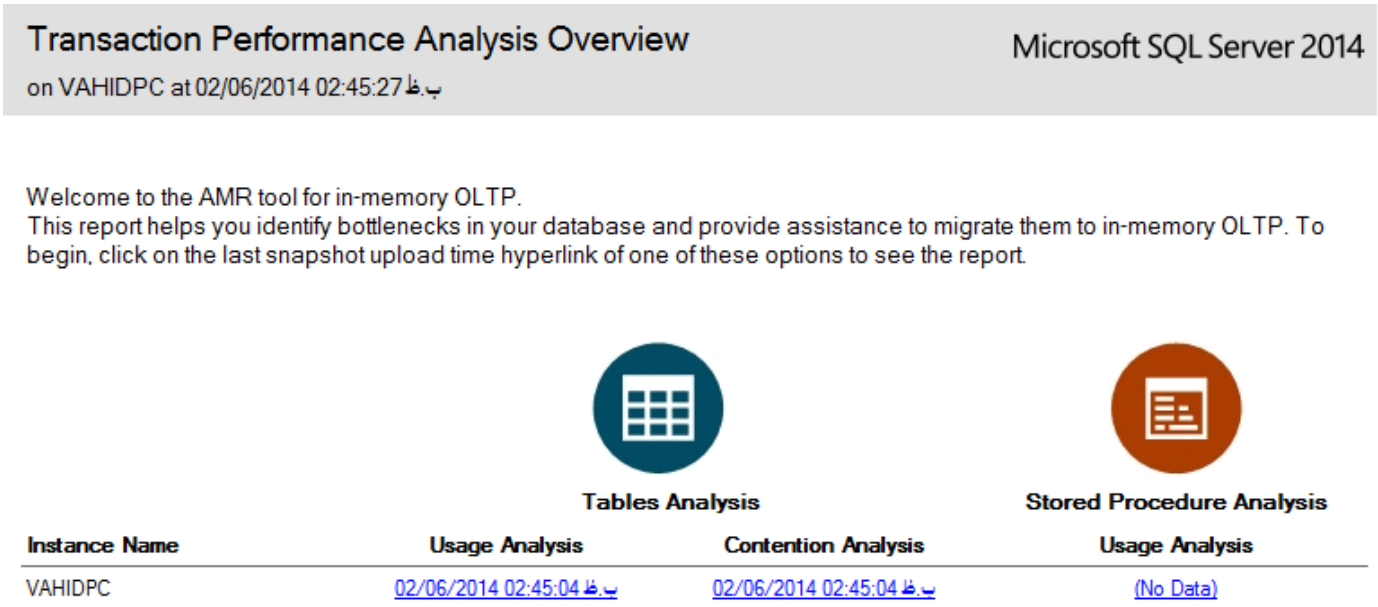


پس از این کارها باید مدتی صبر کنید (مثلا یک ساعت) تا سیستم به صورت معمول کارهای متداول خودش را انجام دهد. پس از آن می‌توان به گزارشات AMR مراجعه کرد.



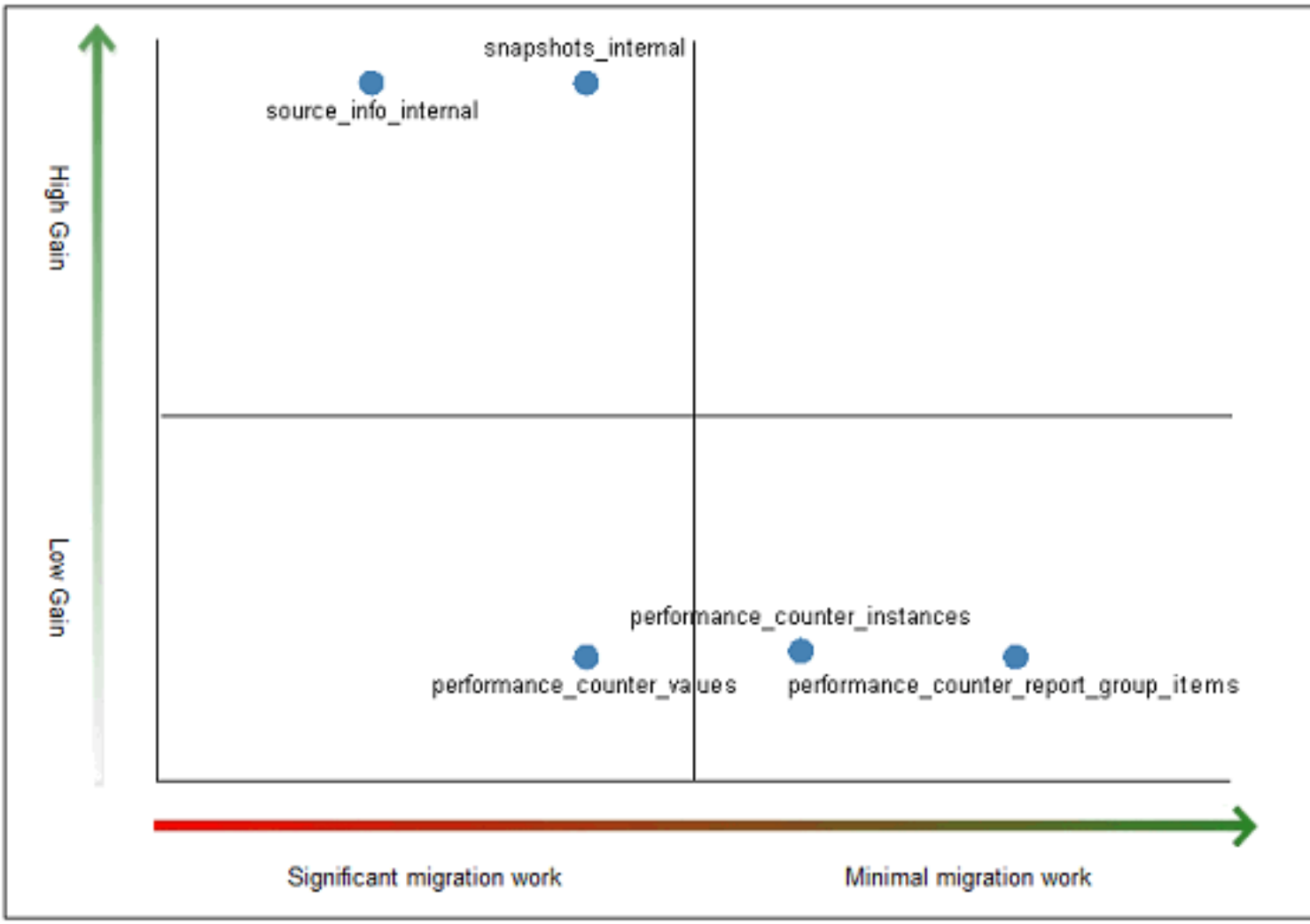
برای اینکار بر روی بانک اطلاعاتی Management Data Warehouse که در ابتدای عملیات ایجاد شد، کلیک راست نمائید و سپس مراحل ذیل را طی کنید:

Reports > Management Data Warehouse > Transaction Performance Analysis Overview



در گزارش ایجاد شده، ذیل گزینه‌ی usage analysis لینک‌هایی وجود دارند که با مراجعه به آن‌ها، چارت‌هایی از میزان مصرف بانک‌های اطلاعاتی مختلف سیستم ارائه می‌شود. اگر پیام No data available را مشاهده کردید، یعنی هنوز باید مقداری صبر کنید تا کار جمع آوری اطلاعات به پایان برسد.

در این چارت‌ها بانک‌های اطلاعاتی که در سمت راست، بالای تصویر قرار می‌گیرند، انتخاب مناسبی برای تبدیل به بانک‌های اطلاعاتی درون حافظه‌ای هستند. محور افقی آن از چپ به راست بیانگر میزان کاهش سختی انتقال یک جدول به جدول درون حافظه‌ای است (با درنظر گرفتن تمام مسایلی که باید تغییر کنند یا نوع‌های داده‌ای که باید اصلاح شوند) و محور عمودی آن نمایانگر میزان بالا رفتن پاسخ دهی سیستم در جهت انجام کار بیشتر است.



هر زمان هم که کار تصمیم‌گیری شما به پایان رسید، می‌توانید بر روی گزینه‌ی Data collection کلیک راست کرده و آن‌را غیرفعال نمائید.

برای مطالعه بیشتر

SQL Server 2014 Field Benchmarking In-Memory OLTP and Buffer Pool Extension Features

New AMR Tool: Simplifying the Migration to In-Memory OLTP

A Tour of the Hekaton AMR Tool

SQL Server 2014 Memory Optimization Advisor

Getting started with the AMR tool for migration to SQL Server In-memory OLTP Tables

How to Use Microsoft's AMR Tool

SQL Server 2014's Analysis, Migrate, and Report Tool

عنوان: ماندگاری با تاخیر در SQL Server 2014
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۳/۱۴ ۱۳:۳۵
آدرس: www.dotnettips.info
گروه‌ها: SQL Server, Performance, In-Memory OLTP

به صورت پیش فرض SQL Server از روش write-ahead log - WAL استفاده می‌کند. به این معنا که کلیه تغییرات، پیش از commit نهایی باید در لاگ فایل آن نوشته شوند. این مساله با تعداد بالای تراکنش‌ها تا حدودی بر روی سرعت سیستم می‌تواند تاثیرگذار باشد. برای بهبود این وضعیت، در SQL Server 2014 قابلیتی به نام delayed_durability اضافه شده‌است که با فعال سازی آن، کلیه اعمال مرتبط با لاگ‌های تراکنش‌ها به صورت غیرهمزمان انجام می‌شوند. به این ترتیب تراکنش‌ها زودتر از معمول به پایان خواهد رسید؛ با این فرض که نوشته شدن تغییرات در لاگ فایل‌ها، در آینده‌ای محتمل انجام خواهند شد. این مساله به معنای فدا کردن D در ACID است (Atomicity, Consistency, Isolation, Durability). البته باید دقت داشت که رسیدن به ACID کامل هزینه‌بر است و شاید خیلی از اوقات تمام اجزای آن نیازی نباشند یا حتی بتوان با اندکی تخفیف آن‌ها را اعمال کرد؛ مانند D به تاخیر افتاده.

برای اینکار SQL Server از یک بافر 60 کیلوبایتی برای ذخیره سازی اطلاعات لاگ‌هایی که قرار است به صورت غیرهمزمان با تراکنش‌ها نوشته شوند، استفاده می‌کند. هر زمان که این 60KB پر شد، آن‌را flush کرده و ثبت خواهد نمود. به این ترتیب به دو مزیت خواهیم رسید:

- پردازش تراکنش‌ها بدون منتظر شدن جهت commit نهایی در دیسک سخت ادامه خواهند یافت. صبر کمتر به معنای امکان پردازش تراکنش‌های بیشتری در یک سیستم پر ترافیک است.
- با توجه به بافری که از آن صحبت شد، اینبار اعمال Write به صورت یک سری batch اعمال می‌شوند که کارآیی و سرعت بیشتری نسبت به حالت تکی دارند.

اندکی تاریخچه

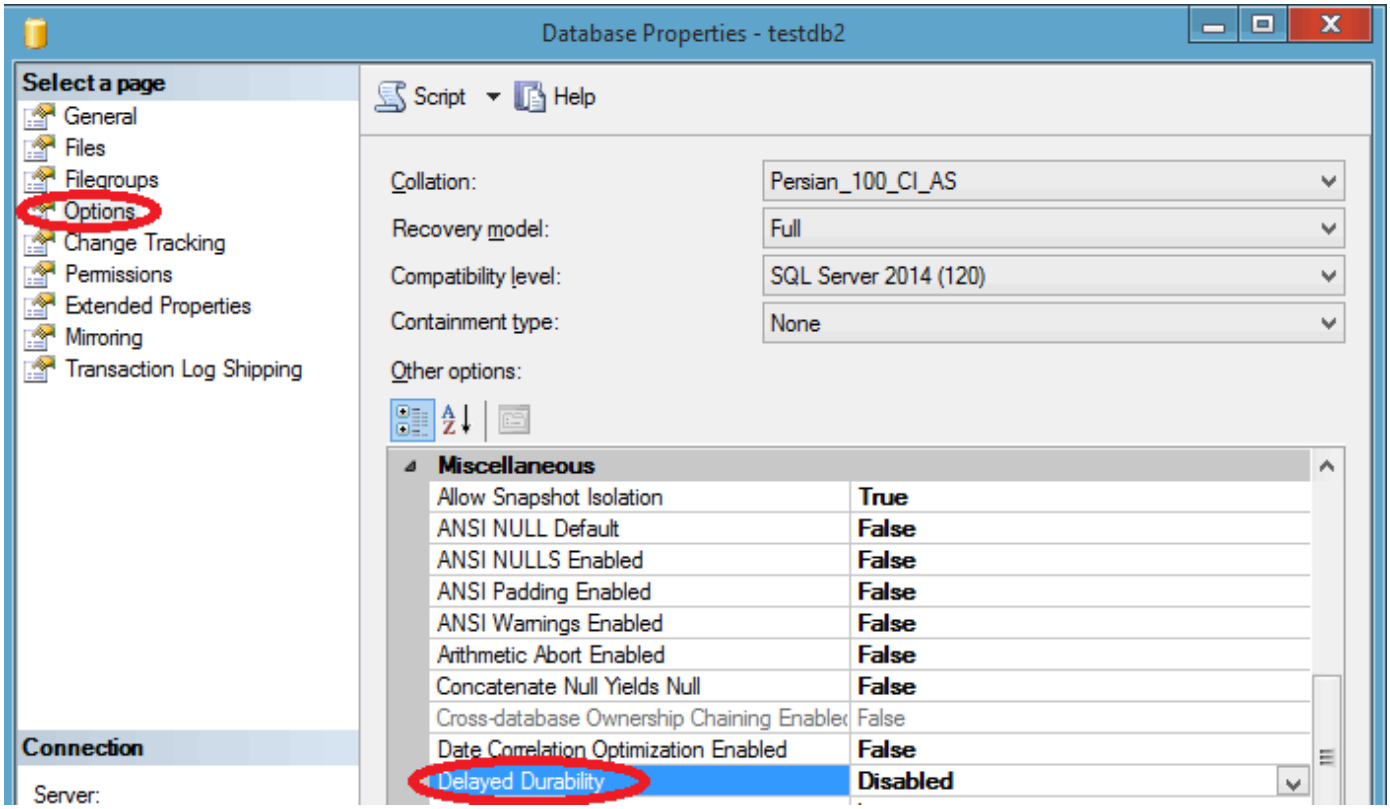
ایده یک چنین عملی 28 سال قبل توسط Hal Berenson ارائه شده‌است! اوراکل آن‌را در سال 2006 تحت عنوان Asynchronous Commit پیاده سازی کرد و مایکروسافت در سال 2014 آن‌را ارائه داده‌است.

فعال سازی ماندگاری غیرهمزمان در SQL Server

فعال سازی این قابلیت در سطح بانک اطلاعاتی، در سطح یک تراکنش مشخص و یا در سطح رویه‌های ذخیره شده کامپایل شده مخصوص OLTP درون حافظه‌ای، میسر است.

برای فعال سازی ماندگاری با تاخیر در سطح یک دیتابیس، خواهیم داشت:

```
ALTER DATABASE dbname SET DELAYED_DURABILITY = DISABLED | ALLOWED | FORCED;
```



در اینجا اگر ALLOWED را انتخاب کنید، به این معنا است که لاگ کلیه تراکنش‌های مرتبط با این بانک اطلاعاتی به صورت غیرهمزمان نوشته می‌شوند. حالت FORCED نیز دقیقا به همین معنا است با این تفاوت که اگر حالت ALLOWED انتخاب شود، تراکنش‌های ماندگار (آن‌هایی که به صورت دستی DELAYED_DURABILITY را غیرفعال کرده‌اند)، سبب flush کلیه تراکنش‌هایی با ماندگاری به تاخیر افتاده خواهند شد و سپس اجرا می‌شوند. در حالت Forced تنظیم دسترسی DELAYED_DURABILITY = OFF در سطح تراکنش‌ها تاثیری نخواهد داشت؛ اما در حالت ALLOWED این مساله به صورت دستی در سطح یک تراکنش قابل لغو است.
البته باید توجه داشت، صرفنظر از این تنظیمات، یک سری از تراکنش‌ها همیشه ماندگار هستند و بدون تاخیر؛ مانند تراکنش‌های سیستمی، تراکنش‌های بین دو یا چند بانک اطلاعاتی و کلیه تراکنش‌هایی که با FileTable، Change Data Capture و Change Tracking سر و کار دارند.

در سطح تراکنش‌های می‌توان نوشت:

```
COMMIT TRANSACTION WITH (DELAYED_DURABILITY = ON);
```

و یا در رویه‌های ذخیره شده کامپایل شده مخصوص OLTP درون حافظه‌ای خواهیم داشت:

```
BEGIN ATOMIC WITH (DELAYED_DURABILITY = ON, ...)
```

سؤال: آیا فعال سازی DELAYED_DURABILITY بر روی مباحث locking و isolation levels تاثیر دارند؟
پاسخ: خیر. کلیه تنظیمات قفل گذاری‌ها همانند قبل و بر اساس isolation levels تعیین شده، رخ خواهند داد. تنها تفاوت در اینجا است که با فعال سازی DELAYED_DURABILITY، کار commit بدون صبر کردن برای پایان نوشته شدن اطلاعات در لاگ سیستم صورت می‌گیرد. به این ترتیب قفل‌های انجام شده زودتر آزاد خواهند شد.

سؤال: میزان از دست دادن اطلاعات احتمالی در این روش چقدر است؟
در صورتیکه سرور کرش کند یا ری‌استارت شود، حداکثر به اندازه‌ی 60KB اطلاعات را از دست خواهید داد (اندازه‌ی بافری که برای اینکار درنظر گرفته شده‌است). البته عنوان شده‌است که اگر ری‌استارت یا خاموشی سرور، از پیش تعیین شده باشد، ابتدا

کلیه لاگ‌های flush نشده، ذخیره شده و سپس ادامه‌ی کار صورت خواهد گرفت؛ ولی زیاد به آن اطمینان نکنید. اما همواره با فراخوانی sys.sp_flush_log، می‌توان به صورت دستی بافر لاگ‌های سیستم را flush کرد.

یک آزمایش

در ادامه قصد داریم یک جدول جدید را در بانک اطلاعاتی آزمایشی testdb2 ایجاد کنیم. سپس یکبار تنظیم DELAYED_DURABILITY = FORCED را انجام داده و 10 هزار رکورد را ثبت می‌کنیم و بار دیگر DELAYED_DURABILITY = DISABLED را تنظیم کرده و همین عملیات را تکرار خواهیم کرد:

```
CREATE TABLE tblData(
    ID INT IDENTITY(1, 1),
    Data1 VARCHAR(50),
    Data2 INT
);
CREATE CLUSTERED INDEX PK_tblData ON tblData(ID);
CREATE NONCLUSTERED INDEX IX_tblData_Data2 ON tblData(Data2);

-------------------------

alter database testdb2 SET DELAYED_DURABILITY = FORCED;

-------------------------

SET NOCOUNT ON
Print 'DELAYED_DURABILITY = FORCED'
DECLARE @counter AS INT = 0
DECLARE @start datetime = getdate()
WHILE (@counter < 10000)
BEGIN
      INSERT INTO tblData (Data1, Data2) VALUES('My Data', @counter)
      SET @counter += 1
END
Print DATEDIFF(ms,@start,getdate());
GO

-------------------------

alter database testdb2 SET DELAYED_DURABILITY = DISABLED;
truncate table tblData;
-------------------------

SET NOCOUNT ON
Print 'DELAYED_DURABILITY = DISABLED'
DECLARE @counter AS INT = 0
DECLARE @start datetime = getdate()
WHILE (@counter < 10000)
BEGIN
      INSERT INTO tblData (Data1, Data2) VALUES('My Data', @counter)
      SET @counter += 1
END
Print DATEDIFF(ms,@start,getdate());
GO

-----------------------
```

با این خروجی:

```
DELAYED_DURABILITY = FORCED
666
DELAYED_DURABILITY = DISABLED
2883
```

در این آزمایش، سرعت insertها در حالت DELAYED_DURABILITY = FORCED حدود 4 برابر است نسبت به حالت معمولی.

برای مطالعه بیشتر

Control Transaction Durability

SQL Server 2014 Delayed Durability/Lazy Commit

Delayed Durability in SQL Server 2014 – Part 1

Is In-Memory OLTP Always a silver bullet for achieving better transactional speed

Delayed Durability in SQL Server 2014