بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

مقدمه ای بر معماری Memory-Optimized Tables

مدرس : مهندس افشین رفوآ

دوره آموزش MVC

مقدمه ای بر معماری Memory-Optimized Tables (جداول بهینه سازی شده بر اساس حافظه)

Memory-optimized tables جداولی هستند که به وسیله ی دستور CREATE TABLE (TransactSQL) ایجاد شده اند.

جداول بهینه سازی شده بر اساس حافظه به صورت پیش فرض کاملا ماندگار (durable) بوده و درست مشابه تراکنش های مبتنی بر دیسک (تراکنش های سنتی)، تراکنش های کاملا پایایی که بر روی جداول Memory-optimized انجام می شوند نیز تجزیه ناپذیر ، قابل اطمینان، ایزوله و ماندگار (ACID) می باشند. جداول بهینه سازی شده بر مبنای حافظه و stored procedure های کامپایل شده به dll. (natively compiled stored procedures) توانایی پشتیبانی از یک زیرمجموعه ای از دستورات Transact-SQL را دارند.

محل ذخیره سازی اولیه برای جداول بهینه سازی شده بر اساس حافظه، حافظه ی اصلی است؛ به عبارت دیگر جداول Memory-optimized بر روی حافظه نگه داری می شوند. سطرهای جداول از حافظه خوانده شده و در آن نیز ثبت و ذخیره می گردند، نتیجه می گیریم که کل جدول مورد نظر در حافظه مقیم می باشد. یک رونوشت عینی نیز از داده ها و اطلاعات جدول بر روی دیسک ذخیره نگه داشته می شود که تنها به مقصود تضمین پایایی و ماندگاری داده ها صورت می پذیرد. اطلاعات تنها در شرایطی که پایگاه داده در

حال بازیابی (recovery) است، از دیسک خوانده می شوند، به عنوان مثال، پس از بازآغازی (restart) سرویس دهنده.

به منظور دستیابی به کارایی بیشتر، معماری In-Memory OLTP از جداول پایا که ماندگارسازی تراکنش درآن به تاخیر می افتد، پشتیبانی می کند. تراکنش های پایای به تاخیر افتاده بلافاصله پس از اینکه transaction تایید ثبت گردیده و کنترل را به کلاینت بازگردانده، بر روی دیسک ثبت (write) می شوند. به ازای کارایی بهبود یافته، تراکنش های تایید ثبت شده که بر دیسک ذخیره نشده اند طی از کار افتادگی سرور (server crash) یا مکانیزم failover، از دست می روند.

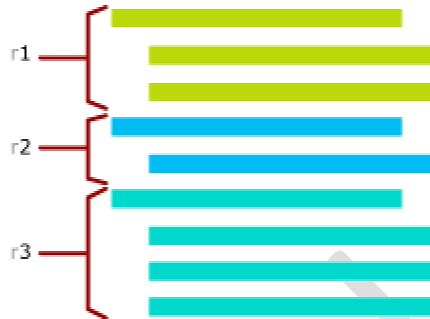
جدا از جداول memory-optimized پایای پیش فرض، SQL Server از جداول بهینه سازی شده که ماندگار نیستند نیز پشتیبانی می کند. این نوع جداول ثبت گزارش (log) نشده و همچنین اطلاعات درون آن بر روی دیسک ماندگار سازی نمی شود. به عبارتی دیگر، تراکنش هایی که بر روی این جداول انجام می شوند نیاز به هیچ disk IO (جریان ورود و خروج به دیسک) نداشته، ولی دیگر در صورت رخداد خرابی سرور یا failover اطلاعات بازیابی نمی شوند (امکان احیای داده ها وجود ندارد).

معماری In-Memory OLTP با SQL Server ترکیب (یکپارچه) شده و تجربه ای روان را در تمامی زمینه ها، همچون برنامه نویسی و توسعه، نصب و مستقرسازی، قابلیت اداره و نیز قابلیت پشتیبانی را به ارمغان می آورد.

یک پایگاه داده می تواند هم دربردارنده ی in-memory objects (اشیای مقیم در حافظه) باشد و هم disk-based object (اشیایی که بر روی دیسک ذخیره می شوند).

سطرهای داخل جداول بهینه سازی شده بر اساس حافظه نسخ سازی (version) می شوند، بدین معنا که هر سطر یا ردیف واقع در جدول ممکن است به صورت نهانی دارای چندین نسخه ی مختلف باشد. تمامی نسخه های سطر مربوطه در یک ساختار داده ای جدول (table data structure) یکسان نگه داری می شوند. Row versioning (نسخه سازی سطرها) به منظور فراهم آوردن امکان خواندن و نوشتن همزمان بر روی سطر یکسان مورد بهره وری قرار می گیرد.

نگاره ی زیر قابلیت multi-versioning را به نمایش می گذارد. این نگاره یک جدول را نمایش می دهد که دارای سه ردیف بوده و هر سطر دارای نسخه های مختلف می باشد.



همان طور که مشاهده می کنید، جدول بالا دارای سه سطر می باشد: r1، r2 و r3. r1 در مجموع سه نسخه دارد، در حالی که r2 تنها دو نسخه و r3 دارای چهار ورژن می باشد. لازم به ذکر است که نسخه های مختلف یک سطر لزوما مکان های پی در پی و متوالی را در حافظه اشغال نمی کنند. از این رو نسخه های مختلف یک ردیف می توانند در سرتاسر ساختار داده ای جدول پراکنده شوند.

می توان به ساختار داده ای جدول memory-optimized به چشم یک مجموعه ای از نسخه های مختلف از یک سطر نگاه کرد. سطرهای موجود در جدول ذخیره شده بر روی دیسک (disk-based) در صفحات و extent هایی (ترکیب 8 صفحه) سازمان دهی شده، سطرهای تکی نیز با استفاده از شماره و زیر آدرس صفحه (page offset) آدرس دهی می شوند. همچنین نسخه های مختلف سطر مورد نظر توسط اشاره گر های حافظه (memory pointer) 8 بایتی آدرس دهی می شوند.

داده های درون جداول memory-optimized به از دو طریق قابل دسترسی می باشند:

از طریق رویه های آماده به صورت اختصاصی کامپایل شده (natively compiled stored procedures).

Tahlildadeh

از طریق افزونه ی Transact-SQL تفسیر شده، خارج از stored-procedure ای که به صورت native (اختصاصی) ترجمه و کامپایل شده است. این دستورات Transact-SQL ممکن است یا داخل stored procedure های تفسیر شده باشند و یا دستورات ویژه ی (ad-hoc) Transact-SQL باشند.

دسترسی به اطلاعات در جداول بهینه سازی شده بر اساس حافظه

بهترین و موثرترین روش برای دسترسی به اطلاعات درون جداول memory-optimized، از طریق natively compiled stored procedure می باشد. natively compiled stored procedure رویه های ذخیره شده ی Transact-SQL ای هستند که به کدهای اختصاصی یا native ترجمه شده و به منظور دسترسی به جداول بهینه سازی شده بر اساس حافظه بکار گرفته می شوند. دسترسی به جداول memory-optimized از طریق Transact-SQL تفسیر شده نیز امکان پذیر می باشد. دسترسی از طریق دستورات Transact-SQL تفسیر شده، اشاره به دستیابی به جداول memory-optimized بدون بهره گیری از natively compiled stored procedure (رویه های ذخیره شده کامپایل شده به dll) دارد. به عنوان مثال می توان به دسترسی به جدول memory-optimized از طریق یک DML trigger، یک مجموعه دستورات ویژه Transact-SQL (ad hoc Transact-SQL batch)، view و توابع مبتنی بر جدول (table-valued function) اشاره کرد.

جدول زیر دسترسی دستورات native و تفسیر شده ی Transact-SQL برای اشیا مختلف را به طور خلاصه تشریح می کند:

ویژگی	Access Using a Natively Compiled Stored Procedure (دسترسی از طریق رویه های آماده ی کامپایل شده به dll)	Interpreted Transact- SQL Access (دسترسی از طریق دستورات تفسیر شده ی -Transact SQL)	CLR Access (دسترسی از طریق برنامه های net.)
Memory-optimized table	Yes	Yes	No 1

Tahlildadeh

Memory-optimized table type	Yes	Yes	No
Natively compiled stored procedure	تودرتو (nesting) کردن رویه های ذخیره شده ی کامپایل شده به dll هم اکنون امکان پذیر می باشد. می توان از ساختار نگارشی EXECUTE در stored procedure ها استفاده کرد، البته مادام اینکه رویه ی ارجاع داده شده نیز به dll کامپایل شود.	Yes	No 1

نمی توان از طریق context connection (منظور اتصال از SQL SERVER زمانی که در حال اجرای یک ماژول CLR می باشد، است) به جدول بهینه سازی شده بر اساس حافظه و رویه های آماده ی کامپایل شده به dll دسترسی داشت. با این حال، می توان یک اتصال دیگر ایجاد و راه اندازی کرد و از طریق آن به جداول memory-optimized و stored procedure های ترجمه شده به dll دسترسی پیدا کرد.

کارایی و مقیاس پذیری

عوامل زیر نقش مهمی در افزایش کارایی (که با بهره وری از معماری In-Memory OLTP صورت می پذیرد) ایفا می کنند:

Communication

یک برنامه که رویه های ذخیره شده ی فراوانی را که کارهای کوچک انجام می دهند، فرا می خواند در مقایسه با برنامه ای که stored procedure های کمتری را صدا زده ولی در هر رویه ی ذخیره شده قابلیت بیشتری را پیاده سازی می کند، افزایش کارایی ناچیزی را تجربه می کند.

Transact-SQL Execution

معماری In-Memory OLTP زمانی بهترین کارایی را ارائه می دهد که بجای رویه های آماده (stored procedure) تفسیر شده یا query execution از stored procedure های ترجمه شده به dll، استفاده کند. به طور قطع دسترسی به جداول memory-optimized از طریق چنین stored procedure هایی فوایدی را به دنبال دارد.

Point Lookup در مقایسه با Range Scan

اندیس های خوشه بندی نشده بهینه سازی شده بر اساس حافظه (Memory-optimized nonclustered indexes) از range scan (پویش محدوده) و ordered scan (پویش مرتب) پشتیبانی می کنند. برای point lookup ها، اندیس های درهم سازی (hash index) بهینه شده بر اساس حافظه کارایی بهتری نسبت به اندیس های خوشه بندی نشده بهینه سازی شده بر اساس حافظه دارند. اندیس های خوشه بندی نشده بهینه سازی شده بر اساس حافظه کارایی بهتری نسبت به اندیس های مبتنی بر دیسک (disk-based index) دارند.

عملیات مربوط به اندیس ها ثبت گزارش (log) نشده و تنها در حافظه موجود می باشند.

Concurrency

برنامه هایی که کارایی آن ها تحت تاثیر همزمانی در سطح موتور (engine-level concurrency) قرار دارد، همچون latch contention (قفل زدن در زمان رقابت) یا blocking، به مجرد اینکه برنامه در حالت In-Memory OLTP (بهینه سازی درون حافظه ای) قرار می گیرد، به طور قابل توجهی بهبود می یابد.

جدول زیر به شرح مشکلات و مسائل مربوط به مقیاس پذیری (scalability) و کارایی را که به صورت مکرر در پایگاه داده های رابطه ای یافت می شوند و همچنین اینکه معماری In-Memory OLTP چگونه قادر به اصلاح آن ها است می پردازد:

مشکل	تاثیر معماری In-Memory OLTP

CPU (تاثیر آن بر پردازنده)	کارایی
رویه های ذخیره شده ی ترجمه شده به dll می توانند استفاده از پردازنده را به طور قابل ملاحظه ای کاهش دهند زیرا که این دست از stored procedure ها به تعداد بسیار کمتری دستورالعمل برای اجرای دستورات SQL Transact- نسبت به stored procedure های تفسیر شده نیاز دارند. معماری In-Memory OLTP قادر است منابع سخت افزاری که در فشار های کاری پروژه های کوچک بکار گرفته می شود را به طور قابل توجهی کاهش دهد زیرا که یک سرویس دهنده، بالقوه می تواند خروجی یا توان عملیاتی پنج تا ده سرور را ارائه دهد. در صورت مواجه شدن با گلوگاه ورودی/خروجی (I/O bottleneck) از پردازش گرفته تا صفحات دربردارنده ی اندیس و اطلاعات، معماری In-Memory OLTP ممکن است باعث رفع مشکل گلوگاه شود. علاوه بر آن، حالت برداری (checkpointing) از اشیا In-Memory OLTP پیوسته بوده و منجر به افزایش ناگهانی عملیات ورودی/خروجی نمی شود. به علاوه، چنانچه مجموعه کاری (working set = مقدار حافظه‌ای که یک فرایند در یک بازه زمانی خاص احتیاج دارد) critical performance tables در حافظه جای نگیرد، در این صورت In-Memory OLTP دیگر به بهبود کارایی کمکی نمی کند، زیرا که شرط مفید بودن این معماری مقیم بودن اطلاعات در حافظه است. اگر در ثبت گزارش (logging) با گلوگاه ورودی/خروجی (I/O bottleneck) مواجه شوید، از آنجایی که معماری In-Memory OLTP ثبت گزارش کمتری انجام می دهد، مشکل گلوگاه تا کمی بر طرف می شود. چنانچه جداول بهینه سازی شده بر اساس حافظه به عنوان جداول ناپایا (non-durable) پیکربندی شده باشد، می توانید ثبت گزارش داده ها را به طور کل حذف کند. شبکه معماری In-Memory OLTP در زمینه ی شبکه هیچ بهبود کارایی ارائه نمی دهد. لازم است اطلاعات از لایه ی data به لایه ی application مخابره شوند.	استفاده ی بیش از حد از منابع موجود (از جمله پردازنده، ورودی/خروجی، شبکه یا حافظه)

آدرس آموزشگاه : تهران - خیابان شریعتی - بالا تر از خیابان ملک - جنب بانک صادرات - پلاک 165 - واحد7
88146323 - 88446780 - 88146330

Latch Contention یک سناریو که به طور معمول با آن مواجه می شویم، رقابت بر سر آخرین صفحه ی یک اندیس، به هنگام درج سطر به صورت همزمان به ترتیب کلید (key order) است. از آنجایی که معماری In-Memory OLTP به هنگام دسترسی به داده latch (قفل) نمی گذارد، مسئله ی مقیاس پذیری مربوط به latch contention کاملا برطرف می شود. Spinlock Contention از آنجایی که معماری In-Memory OLTP به هنگام دسترسی به داده latch (قفل) اعمال نمی کند، مسئله ی مقیاس پذیری مربوط به Spinlock Contention کاملا رفع می شود. Locking Related Contention چنانچه برنامه ی پایگاه داده ی شما با مسئله ی blocking (رخداد انسداد) بین عملیات خواندن و نوشتن روبرو شود، معماری In-Memory OLTP این مشکل را از میان بر می دارد، زیرا که از یک گونه یا روش جدید از رویکرد کنترل همروندی خوشبینانه یا OCC برای مدیریت همزمانی به منظور پیاده سازی تمامی مراحل مجزا یا ایزوله سازی تراکنش بهره می گیرد. معماری مزبور از TempDB به منظور ذخیره سازی نسخه های مختلف سطر استفاده نمی کند. چنانچه مشکل مقیاس دهی (scaling) از تداخل بین دو عملیات نوشتن (write) نشات گیرد، همانند دو تراکنش همزمان که سعی دارند سطری یکسان را بروز رسانی کنند، معماری In-Memory OLTP به یکی از تراکنش ها اجازه ی اجرای با موفقیت را می دهد ولی دیگری را کاملا از کار می اندازد. تراکنش با شکست مواجه شده (failed transaction) باید یا به صورت ضمنی و یا به صورت صریح re-submit شود که باعث می شود تراکنش مجددا تکرار و آزمایش شود. در هر دو مورد، لازم است تغییراتی در برنامه ی مورد نظر اعمال شود. اگر برنامه ی شما به طور مکرر با تداخل بین دو عملیات نوشتن مواجه می شود، در آن صورت ارزش قفل گذاری خوشبینانه (optimistic locking) به مراتب پایین می آید. برنامه مناسب پیاده سازی In-Memory OLTP نیست. بیشتر برنامه های OLTP با write conflict مواجه نمی شوند، مگراینکه تداخل	**مقیاس پذیری** **اغلب مشکلات مقیاس پذیری در برنامه های SQL Server از مشکلات مربوط به همزمانی همچون رقابت در قفل گذاری (spinlock و latch ،lock) نشات می گیرد.**

Tahlildadeh

بر اثر lock escalation (تبدیل قفل های coarse-grain به قفل های fine-grain ، کاهش سربار سیستم و افزایش احتمال concurrency contention) روی دهد.	

امنیت در سطح سطر (row-level security) در جداول بهینه سازی شده بر اساس حافظه (memory-optimized tables)

امنیت در سطح سطر (row-level security) در جداول memory-optimized کاملا پشتیبانی می شود. اعمال سیاست های مربوط به امنیت در سطح سطر به جداول بهینه سازی شده بر اساس حافظه اساسا مشابه جداول ذخیره شده بر روی دیسک (disk-based tables) است، با این تفاوت که توابع درون خطی مبتنی بر جدول (in-line table-valued functions) که به عنوان security predicate مورد استفاده قرار می گیرند باید به dll کامپایل شده (natively compiled) باشند (با استفاده ازگزینه ی WITH NATIVE_COMPILATIONایجاد شده باشند).

بسیاری از توابع امنیتی توکار (built-in security functions) که برای row-level security کاربرد و اهمیت ویژه ای دارند، برای جداول memory-optimized فعال سازی شده اند.

EXECUTE AS CALLER – تمامی ماژول های native هم اکنون از عبارت EXECUTE AS CALLER پشتیبانی کرده آن را به صورت پیش فرض بکار می برند، حتی در شرایطی که hint ای مشخص نشده باشد، زیرا که انتظارمی رود تمامی توابع security predicate مربوط به امنیت در سطح سطر (row-level security) از عبارت EXECUTE AS CALLER استفاده می کنند تا بدین وسیله تابع (و تمامی توابع توکار که در آن بکار گرفته می شوند) در context کاربر فراخواننده ارزیابی شوند.

به دلیل بازرسی مجوز فراخواننده (caller)، EXECUTE AS CALLER تا 10 درصد از دست رفت کارایی (performance hit) را به دنبال دارد. چنانچه خود ماژول EXECUTE AS OWNER یا EXECUTE AS SELF را به طور صریح مشخص کرد، در این صورت از رخ دادن فرایند بازرسی مجوز و افت کارایی حاصل از آن کاملا جلوگیری می شود. با این حال، استفاده از هر یک از گزینه های نام برده به

همراه توابع توکار فوق به خاطر context-switching ای (تغییر بستر اجرایی) الزامی که رخ می دهد، افت کارایی بسیار شدید تری را به دنبال دارد.

Tahlildadeh