بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

رهنمودهای ارائه شده برای جداسازی منابع مورد نیاز تراکنش در رابطه با جداول بهینه سازی شده بر اساس حافظه

مدرس : مهندس افشین رفوآ

دوره آموزش SQL Server Administration

رهنمودهای ارائه شده برای جداسازی منابع مورد نیاز تراکنش (transaction isolation level) در رابطه با جداول بهینه سازی شده بر اساس حافظه

در بسیاری از موارد شما ملزم به مشخص کردن جداسازی بستر تراکنش های در حال اجرا (transaction isolation level) هستید. isolation level تراکنش برای جداول بهینه سازی شده بر اساس حافظه (memory-optimized) با isolation level برای جداول مبتنی بر دیسک طبیعتا فرق دارد. ایزولیشن یا جداسازی، در سامانه های پایگاه داده، مشخص می‌کند که چگونه تراکنشها برای دیگر سیستم‌ها قابل مشاهده باشد. ایزولیشن در سیستم‌هایی که تراکنش‌های بالا و همزمان دارند بسیار مهم است و ایجاد ایزولیشن بالا باعث کم شدن مشکلات همزمانی می‌شود. تراکنش ها یک isolation level تعریف می کنند که میزان جداسازی یک تراکنش از تغییرات اعمال شده به منبع (resource) یا داده ها که توسط دیگر تراکنش ها صورت گرفته را مشخص می کند.

شرایط لازم برای مشخص کردن یا تعریف isolation level تراکنش:

TRANSACTION ISOLATION LEVEL یک گزینه یا option لازم برای بلوک تجزیه ناپذیر (atomic block) که دربردارنده ی محتویات stored procedure کامپایل شده به DLL است، محسوب می شود.

به خاطر محدودیت هایی که در استفاده از isolation level در تراکنش های cross-container (ظرف متقابل) وجود دارد، استفاده از جداول memory-optimized در دستورات Transact-SQL تفسیر شده باید با یک table hint که isolation level ای که برای دسترسی به جدول بکار گرفته می شود را مشخص می کند، همراه شود.

Transaction isolation level دلخواه باید به صورت صریح تعریف شود. نمی توان با استفاده از hint های اعمال قفل همچون XLOCK (lock hints) به منظور تنظیم اختصاصی رفتار sql server از سه جنبه ی granularity، mode و duration بکار می رود. Locking hint ها همراه با دستورات insert، update، select و delete به منظور مطلع ساختن sql server از نحوه ی دلخواه قفل گذاری بر جدول توسط بازنویسی هر گونه transaction isolation level، بکار می رود.) جداسازی سطرها یا جداول مورد نظر را در تراکنش مربوطه تضمین کرد.

برنامه ای که به پایگاه داده دسترسی دارد باید منطق retry را پیاده سازی کرده تا بتواند با خطاها و مشکالات برخواسته از تداخلات اساسی که منجر به از کار افتادگی تراکنش می شود (transaction-dooming failures)، ناموفق بودن اعتبارسنجی (validation failure) و همچنین commit-dependency failures را رفع کند. توجه داشته باشید که خطای commit-dependency failure ممکن است برای تراکنش های read-only (فقط_خواندنی) نیز رخ دهد.

در کار با جداول memory-optimized باید تا حد امکان از تراکنش های طولانی اجتناب کرد. تراکنش هایی از این دست احتمال رخداد تداخل (conflict) و پایان یافتن ناگهانی تراکنش ها که از آن نشات می گیرد را افزایش می دهد. تراکنش های طولانی همچنین فرایند زباله روبی (garbage collection) را به تاخیر می اندازند. هرچه یک تراکنش طولانی تر شود، In-Memory OLTP نیز به مدت طولانی تری نسخه های اخیرا حذف شده ی سطرها را نزد خود نگه می دارد که در نهایت باعث می شود کارایی lookup برای تراکنش های جدید کاهش یابد.

جداول ذخیره شده بر روی دیسک (disk-based) اغلب برای جداسازی منابع مورد نیاز تراکنش (transaction isolation) از قفل گذاری و مسدود سازی (locking&blocking) بهره می گیرند. در حالی که

جداول بهینه سازی شده بر اساس حافظه (memory optimized) از چند نسخه سازی یا multi-versioning و شناسایی تداخلات (conflict detection) برای تضمین جداسازی منابع مورد نیاز تراکنش ها بهره می برد.

جداول ذخیره شده بر روی دیسک استفاده از چند نسخه سازی (multi-versioning) با سطوح SNAPSHOT و READ_COMMITTED_SNAPSHOT را می دهد. برای جداول بهینه سازی شده بر اساس حافظه تمامی isolation level ها مبتنی بر چند_نسخه (multi-version based) هستند از جمله REPEATABLE READ و SERIALIZABLE.

انواع تراکنش

در SQL Server، تمامی query ها در چارچوب یا کانتکست یک تراکنش اجرا می شود.

در SQL Server کلا سه نوع تراکنش وجود دارد:

1. تراکنش های Autocommit: چنانچه هیچ چارچوب تراکنش فعالی (transaction context) وجود نداشته باشد و تراکنش های ضمنی (implicit transaction) نیز در session بر روی ON تنظیم نشده باشند، در آن صورت هر query دارای چارچوب تراکنش مختص به خود خواهد بود. تراکنش همراه با شروع اجرای دستور آغاز شده و با اتمام آن نیز خاتمه می یابد.
2. Explicit transactions (تراکنش های صریح): کاربر تراکنش را از طریق یک BEGIN TRAN یا BEGIN ATOMIC صریح راه اندازی می کند. تراکنش پس از COMMIT و ROLLBACK و یا END (برای یک بلوک تجزیه ناپذیر) پایان می یابد.
3. Implicit transactions (تراکنش های ضمنی): در مواردی که گزینه ی IMPLICIT_TRANSACTIONS بر روی ON تنظیم شده، هر گاه که کاربر یک دستور را اجرا کند و هیچ چارچوب تراکنش فعالی وجود نداشته باشد، تراکنش به صورت ضمنی راه اندازی می شود. چنین تراکنشی توسط یک COMMIT یا ROLLBACK صریح تکمیل شده و پایان می یابد.

Baseline READ COMMITTED Isolation

READ COMMITTED، isolation level پیش فرض در SQL Server است.

این مقدار پیش فرض برای isolation level می باشد. این مقدار تضمین می کند که داده ای که خوانده می شود یک داده commit شده باشد. یعنی مشکل dirty read بوجود نخواهد آمد. نحوه رفع این مشکل نیز بدین صورت است که هر گاه تراکنشی داده ای را تغییر دهد و هنوز commit نکرده باشد، آن داده با قفل exclusive برچسب گذاری می شود. سپس اگر تراکنشی بخواهد داده را بخواند، سیستم اجازه خواندن داده را نمی دهد تا قفل آن داده آزاد شود.

تمامی isolation level ها که برای جداول بهینه سازی شده بر اساس حافظه قابل استفاده می باشند، read committed guarantee را ارائه می دهند (تضمین می کند که اطلاعات خوانده شده است). بنابراین، اگر تراکنش مورد نظر به guarantee های محکم تری نیاز ندارد، در آن صورت می توان از تمامی isolation level هایی که برای جداول memory-optimized قابل استفاده می باشد، بهره گرفت. SNAPSHOT (گرفت رونوشت فوری) در مقایسه با دیگر isolation level به کم ترین میزان منابع سیستم نیاز دارد.

Guarantee که توسط SNAPSHOT isolation level (پایین ترین سطح isolation که برای جداول memory-optimized قابل استفاده می باشد) ارائه می شود READ COMMITTED را نیز شامل می شود. تمامی دستورات موجود در تراکنش، نسخه ی یکسان و یکپارچه از پایگاه داده مد نظر را می خواند. نه تنها تمامی سطرها توسط تراکنش تایید ثبت شده در پایگاه داده، خوانده می شوند بلکه تمامی عملیات خواند شده همان مجموعه تغییراتی را مشاهده می کنند که توسط مجموعه تراکنش های یکسان اعمال شده است.

راهنمایی: اگر تنها READ COMMITTED isolation guarantee مورد نیاز می باشد، از SNAPSHOT isolation به همراه stored procedure های کامپایل شده به dll و برای دسترسی به جداول memory-optimized از طریق دستورات تفسیر شده ی Transact-SQL، استفاده کنید.

برای تراکنش های autocommit، سطح READ COMMITTED به صورت ضمنی (برای جداول بهینه سازی شده بر اساس حافظه) به SNAPSHOT نگاشت می شود. بنابراین، اگر تنظیمات مربوط به session بر روی READ COMMITTED تنظیم شده است، دیگر لازم نیست isolation level را از طریق table hint به هنگام دسترسی به جداول بهینه سازی شده بر اساس حافظه، مشخص کنید.

نمونه تراکنش autocommit زیر یک پیوند (join) را بین جدول بهینه سازی شده بر اساس حافظه به نام Customers و یک جدول معمولی به نام Order History، به عنوان بخشی از یک دسته (batch) ویژه را نمایش می دهد:

```
Transact-SQL
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
GO
SELECT *
FROM dbo.Customers AS c
LEFT JOIN dbo.[Order History] AS oh
  ON c.customer_id = oh.customer_id;
```

نمونه تراکنش های ضمنی و صریح زیر همان پیوند را نمایش می دهد، اما این بار در چارچوب یک تراکنش کاربری صریح. جدول memory-optimized به نام Customers از طریق سطح snapshot مورد دسترسی قرار می گیرد، همان طور که مثال table hint (از hint به منظور اجرای عملیات DML در بیش از یک جدول استفاده می شود) (with (SNAPSHOT نشان می دهد و دسترسی به جدول معمولی [Order History] نیز از طریق read committed isolation صورت می گیرد.

```
Transact-SQL
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
GO
BEGIN TRAN
SELECT * FROM dbo.Customers c with (SNAPSHOT)
LEFT JOIN dbo.[Order History] oh
  ON c.customer_id=oh.customer_id
...
COMMIT
```

تفاوت های عملیاتی

جدا از read committed guarantee، دو نکته ی کلیدی پیاده سازی نیز وجود دارد که برنامه ی استفاده کننده از جداول ذخیره شده بر روی دیسک ممکن است به آن ها نیاز داشته باشد. لازم است به هنگام تبدیل یک جدول ذخیره شده بر روی دیسک که از طریق سطح READ COMMITTED مورد دسترسی قرار می گیرد، به جدول memory-optimized که توسط SNAPSHOT مورد دستیابی قرار می گیرد، دقت لازم را داشته باشید:

پیاده سازی READ COMMITTED برای جداول ذخیره شده بر روی دیسک (فرض بر اینکه READ_COMMITTED_SNAPSHOT بر روی OFF تنظیم شده باشد) از قفل برای جلوگیری از رخداد

تداخل بین reader ها و writer ها بهره می گیرد. هنگامی که writer شروع به بروز رسانی سطر می کند، writer یک قفل گرفته و تا زمانی که تایید ثبت تراکنش به صورت کامل انجام نشده و پایان نیافته آن قفل را رها نمی کند. تمامی عملیات خواندن مسدود شده و منتظر می ماند تا تراکنش نوشتن (write) تایید ثبت شود.

برخی برنامه ها ممکن است فرض را بر این بگذارند که reader ها همیشه منتظر می مانند writer ها تایید ثبت (commit) شوند، به خصوص در مواردی که یک هماهنگ سازی بین دو تراکنش در لایه ی application وجود داشته باشد.

راهنمایی: برنامه ها نمی توانند کاملا به عملکرد blocking تکیه کنند. چنانچه برنامه ای به هماهنگ سازی یا synchronization بین تراکنش های همروند نیاز داشته باشد، می توان این منطق را از طریق (Transact-SQL)sp_getapplock، در لایه ی application و یا database پیاده سازی کرد.

در تراکنش هایی که از READ COMMITTED بهره می گیرند، هر دستور جدیدترین نسخه ی سطرها در پایگاه داده را مشاهده می کنند. از این رو، دستورات بعدی شاهد تغییراتی در وضعیت (state) پایگاه داده خواهند بود.

Poll گرفتن (پرس و جو) از جدول با استفاده از حلقه ی WHILE و یافتن سطرهای اخیرا اضافه شده، نمونه ای از application pattern ای است که از این فرض پیروی می کند. با هر بار تکرار حلقه، query جدیدترین بروز رسانی های انجام شده در پایگاه داده را مشاهده می کند.

راهنمایی: در صورتی که برنامه برای بدست آوردن جدیدترین سطرهای درج شده در جدول، می بایست از جدول بهینه سازی شده بر اساس حافظه poll بگیرد، در چنین شرایطی باید polling loop را به خارج حوزه (scope) تراکنش انتقال داد.

آنچه در زیر مشاهده می کنید، یک نمونه از برنامه ای است که از این فرض پیروی می کند: Poll گرفتن از یک جدول با استفاده از حلقه ی while و بدست آوردن سطر جدید. با هر بار تکرار حلقه، query به آخرین بروز رسانی های صورت گرفته در پایگاه داده دسترسی پیدا می کند.

اسکریپت نمونه زیر از جدول t1، poll گرفته تا یک سطر جدید یافت شود، سپس برای پردازش بیشتر، یک سطر را از جدول حذف می کند.

از آنجایی که polling از snapshot isolation برای دسترسی به جدول t1 استفاده می کند، لازم است توجه داشته باشید که منطق polling باید به بیرون از حوزه (scope) تراکنش انتقال داده شود. استفاده از منطق polling داخل حوزه ی تراکنش، باعث ایجاد تراکنش طولانی می شود که به طور کلی روش نامناسبی تلقی می گردد.

```
Transact-SQL
Copy
-- poll table
WHILE NOT EXISTS (SELECT 1 FROM dbo.t1)
BEGIN
 -- if empty, wait and poll again
 WAITFOR DELAY '00:00:01'
END
BEGIN TRANSACTION
 DECLARE @id int
 SELECT TOP 1 @id=id FROM dbo.t1 WITH (SNAPSHOT)
 DELETE FROM dbo.t1 WITH (SNAPSHOT) WHERE id=@id
 -- insert processing based on @id
COMMIT
```

اعمال قفل بر table hint ها

table hint ها رفتار پیش فرض query optimizer را در طول اجرای دستورات DML بازنویسی می کنند. این کار را از طریق مشخص کردن یک روش معین اعمال قفل/locking method، یک یا چند اندیس، یک عملیات پردازش query مانند پویش جدول (table scan) یا جستجو ی اندیس (index seek) و یا دیگر گزینه های قابل استفاده، انجام می دهد. Table hint ها در عبارت FROM دستور DML تعریف شده و تنها آن جدول یا view ای را تحت تاثیر قرار می دهند که در عبارت مذکور به آن ارجاع داده شده است.

برای اینکه SQL Server بتواند قفل های بیشتری برای یک تراکنش (بیش از آنکه برای یک تراکنش ضروری است) گرفته یا اعمال کند، می توان از locking hint هایی همچون HOLDLOCK و XLOCK همراه با جداول ذخیره شده بر روی دیسک بهره گرفت.

جداول بهینه سازی بر اساس حافظه (memory-optimzed) از قفل استفاده نمی کنند. سطوح بالاتر isolation (جداسازی منابع مورد نیاز تراکنش) همچون REPEATABLE READ و SERIALIZABLE را می توان برای تعریف guarantee های دلخواه بکار برد.

استفاده از locking hint ها مجاز نبوده و پشتیبانی نمی شود. در عوض می توان guarantee های مورد نیاز را از طریق transaction isolation level (سطوح جداسازی منابع موردنیاز تراکنش) تعریف کرد (NOLOCK به این خاطر پشتیبانی می شود که SQL Server اجازه ی اعمال قفل برای جداول memory-optimized را نمی دهد. توجه داشته باشید که بر خلاف جداول ذخیره شده بر روی دیسک، NOLOCK بیانگر این نیست که رفتار READ UNCOMMITTED برای جداول memory-optimized اعمال می شود.