بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

رهنمودهایی در رابطه با منطق Retry برای تراکنش هایی که بر روی جداول memory-optimized انجام می شوند

مدرس : مهندس افشین رفوآ

دوره آموزش SQL Server Administration

رهنمودهایی در رابطه با منطق Retry برای تراکنش هایی که بر روی جداول memory-optimized انجام می شوند

وضعیت هایی وجود دارد که طی دسترسی تراکنش ها به جداول بهینه سازی شده بر اساس حافظه در آن خطا رخ می دهد. به عبارتی دیگر وضعیت خطاهایی وجود دارد که تراکنش ها در دسترسی به جداول memory-optimized با آن ها مواجه می شوند.

41302. تراکنش جاری سعی بر بروز رسانی سطری (رکورد) داشته که از ابتدای شروع تراکنش آپدیت شده است.

41305. تراکنش جاری به دلیل read validation failure تکرار پذیر موفق به commit یا تایید ثبت نشده است.

41325. تراکنش فعلی به خاطر serializable validation failure موفق به تایید ثبت نشده است.

41301. تراکنش قبلی که تراکنش کنونی به آن وابسته است ناگهان متوقف شده و در پی آن تراکنش جاری دیگر قادر به تایید ثبت نیست.

یکی از دلایل شایع روی دادن خطاهایی از این دست، رخداد تداخل بین تراکنش های همروند (تراکنش هایی که به طور هزمان اجرا می شوند) می باشد. حال یکی از اقدامات اصلاحی که می توان ترتیب داد، اجرای مجدد (retry) تراکنش می باشد.

Deadlock یا بنبست (کد خطای 1205) برای جداول بهینه سازی شده بر اساس حافظه رخ نمی دهد. همان طور که پیش تر ذکر شد، قفل برای جداول memory-optimized بکار نمی رود. با این حال اگر برنامه ی مورد نظر از پیش دربردارنده ی منطق retry برای وضعیت بنبست (deadlock) می باشد، می توان با گسترش منطق موجود کدهای خطا (error code) جدید را لحاظ (include) کرد.

ملاحظات در رابطه با پیاده سازی منطق retry

برنامه ها به طور معمول با تداخل بین تراکنش ها مواجه می شوند. به منظور حل این تداخلات برنامه مجبور به پیاده سازی منطق retry می شوند. تعداد دفعاتی که تداخل ممکن است رخ دهد به عوامل مختلفی بستگی دارد:

رقابت بر سر ردیف های فردی (row). احتمال رخداد تداخل با افزایش تعداد تراکنش هایی که سعی بر بروز رسانی سطر مورد نظر دارند، بالا می رود.

تعداد سطرهایی که توسط تراکنش های REPEATABLE READ خوانده می شود. هرچه تعداد سطرهای خوانده شده بیشتر شود، احتمال اینکه برخی سطرها توسط تراکنش های همروند بروز رسانی شود نیز افزایش می یابد. این کار باعث بروز (خطاهای) read validation failure تکرار پذیر می شود.

اندازه ی scan range (محدوده ی پویش) بکار گرفته شده توسط تراکنش SERIALIZABLE. هرچه گستره ی پویش وسیع تر شود (اندازه ی scan range)، احتمال اینکه تراکنش ها سطرهای فرضی (phantom row) وارد کنند نیز افزایش می یابد که در نهایت منجر به رخداد serializable validation failure می شود.

بسیار مهم

تراکنش های Read-write که به جداول memory-optimized دسترسی پیدا می کنند، به منطق retry نیاز دارند.

ملاحظات مربوط به تراکنش های فقط خواندنی (read-only) و stored procedure های کامپایل شده به DLL

تراکنش های read-only که تنها یک بار اجرای stored procedure کامپایل شده به dll را span می کند (شامل می شود)، نیازی به اعتبار سنجی برای تراکنش های REPEATABLE READ و SERIALIZABLE ندارد. تداخل در write به وجود نمی آید، زیرا که تراکنش مربوطه فقط خواندنی می باشد.

با این حال احتمال رخداد dependency failure (ناموفق بودن وابستگی) همواره وجود دارد. Dependency failure در مقایسه با خطاهایی که از تداخلات و conflict ها نشات می گیرند، به مراتب کم تر رخ می دهد. بنابراین در بسیاری از موارد، منطق retry برای تراکنش های فقط خواندنی که تنها یکبار اجرای stored procedure کامپایل شده به dll را شامل می شود، مورد نیاز نمی باشد.

ملاحظاتی در ارتباط با تراکنش های فقط خواندنی و cross-container (ظرف متقابل)

تراکنش های Read-only cross-container (فقط خواندنی ظرف متقابل)، تراکنش هایی هستند که خارج از چارچوب stored procedure کامپایل شده به dll راه اندازی شده و همچنین در صورتی که جداول بهینه سازی شده بر اساس حافظه تحت سطح یا SNAPSHOT isolation level مورد دسترسی قرار گرفته باشند، اعتبار سنجی صورت نمی دهد. با این حال، چنانچه جدول memory-optimized تحت سطوح REPEATABLE READ یا SERIALIZABLE مورد دسترسی قرار بگیرند، اعتبار سنجی در زمان تایید ثبت (commit time) صورت می پذیرد. در چنین مواردی، منطق retry الزامی می باشد.

پیاده سازی منطق retry

مشابه تمامی تراکنش هایی که به جداول memory-optimized دسترسی دارند، باید منطق retry را برای مدیریت خطاهای احتمالی همچون تداخلات write (کد خطای 41302) یا dependency failure (کد خطا 41301) در دستور کار خود قرار دهید. در اغلب برنامه ها نرخ رخداد خطا یا failure پایین خواهد بود، با این وجود نیاز به مدیریت خطاها (failure) با بهره گیری از منطق retry (یا اجرای مجدد تراکنش) از میان برداشته نمی شود. دو روش پیشنهادی برای پیاده سازی منطق retry به شرح زیر می باشد:

اجرای مجدد از سمت سرویس گیرنده (Client-side retry). این روش گزینه ی برتر برای پیاده سازی منطق retry در حالت کلی محسوب می شود. برنامه ی سمت سرویس گیرنده خطاهای ایجاد شده توسط تراکنش را گرفته و تراکنش مورد نظر را مجدد اجرا می کنند. اگر برنامه ی سمت سرویس گیرنده موجود از منطق retry برای مدیریت وضعیت بنبست (deadlock) بهره بگیرد، در آن صورت شما می توانید برنامه را گونه ای گسترش دهید (extend) که قابلیت مدیریت کد خطاهای جدید را نیز پیدا کند.

استفاده از یک wrapper stored procedure. سرویس گیرنده stored procedure تفسیر شده ی Transact-SQL را فرامی خواند که خود stored procedure کامپایل شده به dll را صدا زده و یا تراکنش را اجرا می کند. wrapper procedure (رویه ی دربرگیرنده) سپس با استفاده از منطق try/catch خطا را گرفته (catch) و در صورت نیاز procedure را مجدد فراخوانی می کند. این احتمال وجود دارد که پیش از رخداد خطا (failure) نتایج به سرویس گیرنده برگردانده شده ولی سرویس گیرنده اطلاع ندارد که باید آن ها را پاک (discard) کند. از این رو توصیه می شود از روش نام برده تنها برای stored procedure های کامپایل شده به dll که هیچ مجموعه نتیجه ای را به سرویس گیرنده برنمی گردانند، استفاده کرد.

منطق retry را می توان در Transact-SQL و یا در کد برنامه و در لایه ی میانی (mid-tier) پیاده سازی کرد.

دو دلیل ممکن که در دستور کار قرار دادن منطق retry را توجیح می کنند به شرح زیر است:

1. برنامه ی سمت سرویس گیرنده (client application) دارای منطق retry برای دیگر کدهای خطا (error code) همچون 1205 می باشد که امکان گسترش آن وجود دارد.
2. رخداد تداخل تقریبا نادر بوده و همچنین کاهش زمان تاخیر بین سرویس گیرنده ها (end-to-end latency) با استفاده از execution های از پیش آماده امری ضروری و بسیار مهم می باشد.

مثال زیر استفاده از منطق retry در stored procedure تفسیر شده را نمایش می دهد که دربردارنده ی یک فراخوانی (call) به stored procedure ترجمه شده به dll و یا تراکنش cross-container (ظرف متقابل) می باشد.

```
Transact-SQL
CREATE PROCEDURE usp_my_procedure @param1 type1, @param2 type2, …
AS
```

```
BEGIN
 -- number of retries – tune based on the workload
 DECLARE @retry INT = 10
 WHILE (@retry > 0)
 BEGIN
  BEGIN TRY
   -- exec usp_my_native_proc @param1, @param2, …
   --      or
   -- BEGIN TRANSACTION
   --   …
   -- COMMIT TRANSACTION
   SET @retry = 0
  END TRY
  BEGIN CATCH
   SET @retry -= 1
    -- the error number for deadlocks (1205) does not need to be included for
   -- transactions that do not access disk-based tables
   IF (@retry > 0 AND error_number() in (41302, 41305, 41325, 41301, 1205))
   BEGIN
    -- these error conditions are transaction dooming - rollback the transaction
    -- this is not needed if the transaction spans a single native proc execution
    --  as the native proc will simply rollback when an error is thrown
    IF XACT_STATE() = -1
     ROLLBACK TRANSACTION
    -- use a delay if there is a high rate of write conflicts (41302)
    --  length of delay should depend on the typical duration of conflicting transactions
    -- WAITFOR DELAY '00:00:00.001'
   END
   ELSE
   BEGIN
    -- insert custom error handling for other error conditions here
    -- throw if this is not a qualifying error condition
    ;THROW
   END
  END CATCH
 END
END
```