بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

تراکنش ها در جداول بهینه سازی شده بر اساس حافظه

مدرس : مهندس افشین رفوآ



## دوره آموزش SQL Server Administration

## تراکنش ها در جداول بهینه سازی شده بر اساس حافظه

پیاده کردن row versioning یا نسخه سازی از سطرها بر روی جداول ذخیره شده در دیسک (از طریق SNAPSHOT یا READ_COMMITTED_SNAPSHOT ) یک نوع مدیریت همروندی خوشبینانه را ارائه می دهد. Reader ها و Writer ها یکدیگر را مسدود یا block نمی کنند. در جداول memory-optimized، writer ها به هیچ وجه reader ها را مسدود نمی کنند. با پیاده سازی row versioning بر روی جداول مبتنی بر دیسک (ذخیره شده در دیسک)، یکی از تراکنش های در احال اجرا به سطر مورد نظر قفل زده و دیگر تراکنش های همروند که سعی بر بروز رسانی سطر نام برده دارند مسدود می شوند. برای جداول memory-optimized هیچ گونه اعمال قفلی صورت نمی پذیرد. در عوض اگر دو تراکنش تلاش کنند سطری یکسان را بروز رسانی کنند، تداخل write/write (خطای 41302 ) رخ می دهد.

بر خلاف جداول ذخیره شده بر روی دیسک، جداول بهینه سازی شده بر اساس حافظه اجازه ی استفاده از مدیریت همروندی خوشبینانه (optimistic concurrency control) با سطوح بالا همچون REPEATABLE READ و SERIALIZABLE را می دهد. قفل ها به منظور به حفظ isolation level ها بکار نمی روند. در عوض، در انتهای اعتبار سنجی تراکنش، read تکرار پذیر و قابلیت سریاله شدن (serializability) را تضمین می کند. در صورت رخداد هر گونه نقض، تراکنش خاتمه داده می شود.

واژه های مهم مربوط به تراکنش ویژه ی جداول memory-optimized:

Multi-versioning: چند نسخه سازی

جداسازی منابع مورد نیاز تراکنش (transaction isolation) مبتنی بر snapshot (نسخه ی فوری)

Optimistic (خوشبینانه)

Conflict detection (شناسایی و تشخیص تداخل)

در ادامه ی این مبحث به بسط هر یک از واژه ها و اصطلاحات ذکر شده خواهیم پرداخت.

## Multi-versioning در جداول بهینه سازی شده بر اساس حافظه

سطرهای موجود در جداول memory-optimized می توانند نسخه های مختلف داشته باشند. تراکنش های همروند می توانند به نسخه های مختلف یک سطر دسترسی داشته باشند.

اطلاعات جداول بهینه سازی بر اساس حافظه مبتنی بر نسخه هستند. به ازای هر سطر ممکن است نسخه های مختلفی وجود داشته باشد که هر یک در بازه ی زمانی مشخص معتبر و قابل استفاده باشد. در مواردی که READ_COMMITTED_SNAPSHOT و ALLOW_SNAPSHOT_ISOLATION بر روی ON تنظیم شده ، جداول ذخیره شده بر روی دیسک، نسخه های مختلفی از یک سطر را نزد خود نگه می دارند. جداول memory-optimized حتی در مواردی که READ_COMMITTED_SNAPSHOT و ALLOW_SNAPSHOT_ISOLATION بر روی OFF تنظیم شده اند نیز نسخه های مختلفی از سطر مورد نظر را نزد خود نگه می دارند. نسخه های مختلف سطر متعلق به جداول بهینه سازی شده بر اساس حافظه در tempdb نگه داشته نمی شوند. در عوض، نسخه های مختلف سطر مورد نظر به صورت درون خطی (in-line) نگه داری می شوند (زیرا که ساختار داده ای جداول memory-optimized بدین صورت هست که سطرها را در حافظه ذخیره می کند).

## جداسازی منابع مورد نیاز تراکنش (transaction isolation) مبتنی بر snapshot (نسخه ی فوری)

تمامی عملیاتی که در یک تراکنش منفرد انجام می شود از نسخه ی فوری (snapshot) یکسان و یکپارچه مبتنی بر تراکنش جداول memory-optimized بهره می گیرند. تمامی transaction isolation هایی که برای

جداول بهینه سازی شده بر اساس حافظه صورت می گیرد، مبتنی بر snapshot هست. برای مثال، یک تراکنش که از سطح serializable (serializable isolation level) برای دسترسی به جداول بهینه سازی شده بر اساس حافظه استفاده می کند، کلیه ی عملیات لازم را بر روی یک snapshot که از نظر تراکنش یکپارچه و همخوان می باشد، پیاده می کند.

تراکنش هایی که به جداول بهینه سازی شده بر اساس حافظه دسترسی دارند، از این قابلیت نسخه سازی از سطر (row versioning) برای بدست آوردن یک snapshot که از نظر تراکنش یکپارچه است، کمک می گیرد. اطلاعاتی که توسط یک دستور معین در تراکنش خوانده می شود، در واقع همان نسخه از داده ها و اطلاعات که از نظر تراکنش یکپارچه بوده و از ابتدای شروع تراکنش وجود داشته، خواهد بود.

بنابراین، هرگونه اصلاح یا تغییری که توسط تراکنش های همروند (تراکنش هایی که باهم اجرا می شوند) انجام می پذیرد، برای دستورهای موجود در تراکنش جاری قابل رویت نخواهد بود.

## مدیریت همروندی خوشبینانه (optimistic concurrency control) برای جداول بهینه سازی بر اساس حافظه

تداخل و خرابی به ندرت رخ می دهد و تراکنش های انجام شده بر روی جداول بهینه سازی شده بر اساس حافظه، فرض را بر این می گذارد که هیچ تداخلی بین تراکنش ها وجود نداشته و عملیات با موفقیت انجام می شوند. تراکنش ها اجازه ی استفاده از lock یا latch را بر روی جداول memory-optimized به منظور تضمین و به اجرا در آوردن transaction isolation را نمی دهد. Writer ها به هیچ وجه reader ها را مسدود نمی سازند (block) و reader ها نیز هیچگاه writer ها را مسدود نمی کنند. در عوض، تراکنش ها با این فرض خوشبینانه پیش می روند که هیچ تداخلی با دیگر تراکنش ها رخ نخواهد داد. عدم نیاز به استفاده از lock و latch و نیز عدم نیاز به منتظر ماندن برای اینکه دیگر تراکنش ها پردازش سطرهای یکسان را به اتمام برسانند، خود کارایی را به طور قابل توجهی بهبود می بخشد.

بعلاوه، چنانچه یک تراکنش (TxA) سطرهایی را بخواند که خود توسط یک تراکنش دیگر (TxB) که در حال تایید ثبت (commit) می باشد درج و اصلاح گردد تراکنش اول به صورت خوشبینانه فرض می گیرد که تراکنش دوم بجای اینکه منتظر بماند commit روی دهد، خود تایید ثبت را انجام دهند. در چنین شرایطی، تراکنش TxA برای commit بر روی تراکنش TxB وابسته خواهد بود.

## تشخیص تداخل، اعتبار سنجی و بررسی commit dependency

SQL Server قادر است تداخلات بین تراکنش ها و نقض isolation level ها را شناسایی کرده و در پی آن یکی از تراکنش هایی که باعث ایجاد تداخل شده است را خاتمه دهد. چنین تراکنشی باید مجددا اجرا (retry) شود.

سیستم خوشبینانه فرض می گیرد که هیچ تداخلی وجود نداشته و هیچ نقض transaction isolation ای رخ نداده است. در صورت روی دادن تداخلی که ممکن است باعث ایجاد ناهماهنگی یا تناقض در پایگاه داده شود و یا transaction isolation را نقض کند، تداخلات نام برده بلافاصله شناسایی شده و به دنبال آن تراکنش مورد نظر خاتمه داده می شود.

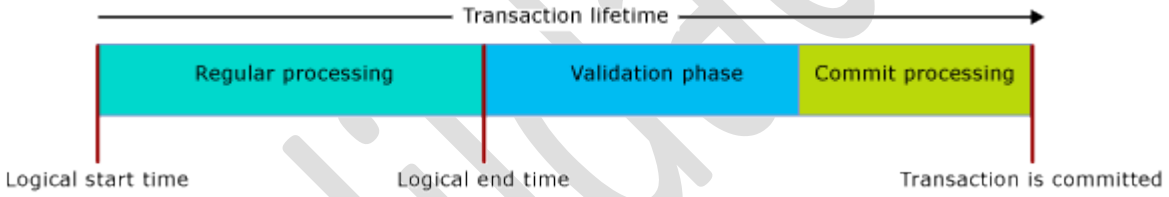
در صورت رخ دادن تداخل، تراکنش پایان داده شده و سرویس گیرنده مجبور به اجرای مجدد (retry) می شود. دئجدول زیر وضعیت هایی که تراکنش ها در دسترسی به جداول بهینه سازی شده بر اساس حافظه در آن با خطا مواجه می شوند را به طور خلاصه شرح می دهد.

| سناریو | خطا |
|---|---|
| یک سطری را UPDATE یا DELETE کند که از پیش توسط تراکنش همزمان دیگری بروز رسانی شده یا حذف گردیده است. | تداخل write. سعی دارد یک سطر را که از ابتدای یا شروع تراکنش آپدیت شده است را بروز رسانی کند. |
| یک سطر که توسط تراکنش مورد نظر خواند شده، از زمان شروع تراکنش تغییر یافته (بروز رسانی شده یا حذف گردیده). Repeatable read validation معمولا زمانی روی می دهد که از جداسازی منابع مورد نیاز تراکنش (transaction isolation levels) REPEATABLE READ و SERIALIZABLE استفاده شود. | Repeatable read validation failure. |
| یک سطر فرضی جدید (phantom row) در یکی از scan range ها در تراکنش مورد نظر، از زمان شروع تراکنش، درج گردیده است. اگر سطر نام برده در پایگاه داده ی مورد نظر پیش از اینکه تراکنش آغاز شود، commit شده بود، این سطر برای تراکنش قابل رویت می شد. SERIALIZABLE validation به طور معمول به هنگام استفاده از SERIALIZABLE | Serializable validation failure. |

| | |
|---|---|
| | isolation و اعتبارسنجی محدودیت های کلید اصلی ( PRIMARY KEY constraint) رخ می دهد. |
| Commit dependency failure. | تراکنش به تراکنش دیگری وابستگی پیدا کرده که موفق به commit نشده است، حال یا به دلیل رخداد خطا (failure) در این جدول، یک out-of-memory condition و یا به دلیل بروز خطا در تایید ثبت در گزارش تراکنش (transaction log). این خطا ممکن است هم با read/write و هم با تراکنش های read-only(فقط_خواندنی) روی دهد. |

## چرخه ی حیات تراکنش

خطاهایی که در جدول بالا به آن اشاره شده است ممکن است در برهه های مختلفی از زمان در طول یک تراکنش رخ دهد. نگاره ی زیر تمامی مراحل یک تراکنش که به جدول بهینه سازی شده بر اساس حافظه دسترسی پیدا می کند، را نمایش می دهد.




### مرحله ی regular processing

در این مرحله، دستورات Transact-SQL صادر شده توسط کاربر (user-issued) اجرا می گردند. سطرهای مورد نظر از جدول خوانده شده و نسخه های جدید از سطرهای موجود در پایگاه داده مربوطه write می شوند. تراکنش مورد نظر از تمامی دیگر تراکنش های همروند جداسازی (ایزوله) می شود. تراکنش مزبور نسخه ی فوری (snapshot) جداول memory-optimized را که در آغاز تراکنش در دسترس (موجود) می باشد را مورد استفاده قرار می دهد.

عملیات Write ای که در جداول در این مرحله انجام می شود، هنوز برای دیگر تراکنش ها قابل مشاهده نمی باشد. در این میان یک استثنا وجود دارد که آن بروز رسانی و حذف سطرها است که برای عملیات delete و update دیگر تراکنش ها قابل رویت می باشد تا از این طریق امکان شناسایی تداخلات write فراهم آید.

چنانچه یک عملیات update و delete متوجه شود که یک سطر از شروع منطقی یک تراکنش بروز رسانی شده یا حذف گردیده، عملیات یاد شده با کد خطای 41302 شکست می خورد. پیام خطای 41302 بدین شرح است:

"The current transaction attempted to update a record in table X that has been updated since this transaction started. The transaction was aborted."

"تراکنش جاری سعی بر بروز رسانی یک رکورد در جدول X داشته که از زمان شروع تراکنش بروز رسانی شده است. تراکنش به طور ناگهانی متوقف شده است."

این خطا تراکنش را محکوم به توقف ناگهانی کرده (حتی اگر هم XACT_ABORT بر روی OFF تنظیم شده باشد)، بدین معنا که با پایان یافتن session کاربر، تراکنش به حالت اولیه باز می گردد (rollback رخ می دهد). تراکنش های محکوم شده به توقف ناگهانی، قادر به commit نبوده و تنها از عملیات read پشتیبانی می کنند که در فایل ثبت گزارش عملیات write انجام نداده و توانایی دسترسی به جداول بهینه سازی شده بر اساس حافظه را ندارد.

## Commit dependency

در مرحله ی regular processing، یک تراکنش می تواند سطرهایی را که توسط دیگر تراکنش ها نوشته شده و در مرحله ی تایید ثبت (commit) یا اعتبار سنجی (validation) به سر می برند و همچنین هنوز تایید ثبت نشده را بخواند. به این خاطر که زمان منطقی اتمام تراکنش از زمان شروع مرحله ی اعتبار سنجی تخصیص داده شده، سطرها قابل مشاهده می باشند.

اگر یک تراکنش چنین سطرهای تایید ثبت نشده را بخواند، در آن صورت برای commit به آن تراکنش وابسته خواهد بود. این اتفاق دو پیامد اصلی زیر را به دنبال دارد:

یک تراکنش نمی تواند تا زمانی که تراکنش هایی که به آن ها وابسته است تایید ثبت نشده اند، commit شود. به عبارتی دیگر، تا زمانی که تمامی dependency ها تایید ثبت نشده اند، تراکنش نمی تواند وارد مرحله ی commit شود.

1. همچنین تا زمانی که dependency ها تایید نشده اند، مجموعه نتایج به سرویس گیرنده برگردانده نمی شوند، این باعث می شود که داده های commit نشده برای سرویس گیرنده نمایش داده نشود.

چنانچه هریک از تراکنش های وابسته موفق به تایید ثبت نشود، یک خطا commit dependency رخ می دهد، بدین معنا که تراکنش با خطای 41301 مواجه شده و موفق به تایید ثبت نمی شود. پیمام خطای نام برده بدین شرح است:

("A previous transaction that the current transaction took a dependency on has aborted, and the current transaction can no longer commit.")

"یک تراکنش که تراکنش جاری به آن وابسته است به طور ناگهانی متوقف شده، از این رو تراکنش فعلی قادر به تایید ثبت نیست. "

## مرحله ی اعتبار سنجی (Validation Phase)

در این مرحله، سیستم بررسی می کند که آیا شرایط transaction isolation level مورد درخواست بین شروع و پایان منطقی تراکنش صحیح بوده یا خیر.

در ابتدای مرحله ی اعتبار سنجی، به تراکنش یک زمان پایان منطقی تخصیص داده می شود. نسخه های سطر نوشته شده در پایگاه داده برای دیگر تراکنش ها در زمان پایان منطقی قابل رویت می باشد.

## مرحله ی Repeatable Read Validation

چنانچه isolation level تراکنش بر روی REPEATABLE READ یا SERIALIZABLE تنظیم شده باشد و یا دسترسی به جداول از طریق REPEATABLE READ یا SERIALIZABLE صورت بگیرد، سیستم بررسی انجام داده و از اینکه عملیات read تکرار پذیر هست یا خیر اطمینان حاصل می کند، بدین معنا که بررسی می کند آیا نسخه های مختلف سطر که توسط تراکنش خوانده می شود در زمان پایان منطقی تراکنش همچنان معتبر هستند یا خیر.

چنانچه هریک از سطرهای موجود بروز رسانی شده یا تغییر یابد، تراکنش با خطای 41305 مواجه شده و موفق به تایید ثبت (commit) نمی شود. پیام خطای نام برده بدین شرح است: "تراکنش فعلی به دلیل یک repeatable read validation failure موفق به تایید ثبت تراکنش نشده است."

این خطا همچنین ممکن است زمانی که یک جدول پس از اجرای عملیات insert ، update یا delete و پیش از اینکه تراکنش تایید ثبت شود رخ دهد. لازم به ذکر است که این رویداد تنها برای عملیات درج، بروز رسانی یا حذف که در stored procedure های کامپایل شده به dll پیاده می شود رخ دهد. عملیات نوشتنی (write) از این دست که از طریق Transact-SQL تفسیر می گردند، باعث می شوند که دستور DROP TABLE مسدود (block) شده و تا اتمام تایید ثبت تراکنش همچنان منتظر مانده و اجرا نشود.

## Serializable Validation

Serializable validation در دو مورد ذکر شده در زیر پیاده می شود:

1. چنانچه isolation level تراکنش بر روی SERIALIZABLE تنظیم (set) شده باشد یا جداول مورد نظر تحت SERIALIZABLE مورد دسترسی قرار گیرد.
2. اگر سطرها در یک اندیس منحصر بفرد درج شود برای مثال می توان به یک اندیس که ویژه ی محدودیت PRIMARY KEY ایجاد شده است، اشاره کرد. سیستم یک بررسی انجام داده و اطمینان حاصل می کند که هیچ سطری با کلید یکسان به طور همزمان درج نشده باشد.

سیستم همچنین بررسی می کند که هیچ سطر فرضی (phantom row) در پایگاه داده نوشته (write) نشده باشد. عملیات خواندنی که توسط تراکنش صورت می پذیرند، برای اینکه مشخص شود هیچ سطر جدیدی در scan range این عملیات read درج نشده، مورد ارزیابی قرار می گیرند.

درج کلید در یک اندیس منحصر بفرد شامل یک عملیات خواندن صریح (implicit read) نیز می شود، بدین وسیله از تکراری نبودن کلید مورد نظر اطمینان حاصل می گردد.

Serializable validation ای که ویژه ی اندیس های منحصر بفرد اجرا می شود، در شرایطی که درج کلید یکسان توسط تراکنش های همروند (به طور همزمان صورت) می گیرد، اطمینان حاصل می کند که اندیس های ایجاد شده دارای نسخه ی تکراری نباشند.

در صورت یافتن هرگونه سطر فرضی (phantom row)، تراکنش با خطای 41325 مواجه شده و موفق به تایید ثبت نمی شود:

("The current transaction failed to commit due to a serializable validation failure.")

("تراکنش کنونی به دلیل یک serializable validation failure موفق به تایید ثبت نشد.")

The system validates that no phantom rows have been written to the database. The read operations performed by the transaction are evaluated to determine that no new rows were inserted in the scan ranges of these read operations.

Insertion of a key in a unique index includes an implicit read operation, to determine that the key is not a duplicate. Serializable validation for unique indexes ensures these indexes cannot have duplicates in case two transactions concurrently insert the same key.

## مرحله ی commit processing

اگر اعتبار سنجی با موفقیت اجرا شده و تمامی dependency های تراکنش آزاد شوند، تراکنش نام برده وارد یک مرحله ی تازه ای از پردازش می شود. در این مرحله کلیه ی تغییراتی که به جداول پایا (ماندگار) اعمال شده اند در فایل ثبت گزارش (log) نوشته می شود، سپس فایل ثبت گزارش یا log بر روی دیسک ذخیره شده تا بدین وسیله ماندگاری آن تضمین گردد. پس از اینکه سطرهای گزارش بر روی دیسک ذخیره شدند، کنترل بار دیگر به سرویس گیرنده باز گردانده می شود.

تمامی وابستگی های متصل به تایید ثبت (dependency commit) در این تراکنش رها شده و کلیه ی تراکنش های دیگری که منتظر بودند این تراکنش commit شود، به کار خود ادامه می دهند.

## محدودیت ها

1. تراکنش های (Cross-database) پایگاه داده ی متقابل برای جداول بهینه سازی شده بر اساس حافظه قابل استفاده نمی باشند. تمامی تراکنش هایی که به جداول بهینه سازی شده بر اساس حافظه دسترسی دارند، قادر به دستیابی به بیش از یک پایگاه داده نیستند. در این میان یک استثنا وجود دارد و آن دسترسی read-write (خواندن-نوشتن) به tempdb و نیز دسترسی read-only (فقط خواندن) به سیستم database master می باشد.

2. تراکنش های توزیع شده (distributed transaction) برای جداول memory-optimized قابل استفاده نبوده و پشتیبانی نمی شود. تراکنش های توزیع شده ای که با BEGIN DISTRIBUTED TRANSACTION راه اندازی شده اند، قادر نخواهند به جداول بهینه سازی شده بر اساس حافظه دسترسی داشته باشند.
3. جداول بهینه سازی شده بر اساس حافظه از قابلیت اعمال قفل پشتیبانی نمی کنند. اعمال قفل های صریح (explicit lock) که از طریق locking hint ها صورت می گیرد (همچون TABLOCK، XLOCK، ROWLOCK) برای جداول بهینه سازی شده بر اساس حافظه قابل استفاده نمی باشد.