Module 8

Plan caching and recompilation

Module Overview

- Plan cache internals
- Troubleshooting the plan cache
- Query Store
- Automatic tuning

Lesson: Plan cache internals

- Query plan caching and retrieval
- Plan cache management
- Queries without plan cache entries
- Maximizing plan cache efficiency
- Examining the plan cache

Query plan caching and retrieval

- Three plan cache stores
  - Object Plans
  - SQL Plans
  - Bound Trees (algebrizer trees for views)
- Compiled plan is held in a hash bucket in the relevant plan cache store, uniquely identified by a plan handle
- A plan can be reused if bucket hash and plan cache key match
- An executable plan is a session-specific instance of a compiled plan
  - It includes working area for holding variable values and such

Demo Plan cache

I SQL Server hanteras frågor genom en process där de kompileras till frågeplaner, som sedan exekveras för att hämta eller manipulera data. För att optimera denna process och undvika att kompilera samma fråga flera gånger, använder SQL Server en mekanism som kallas **plan cache**, där kompilerade planer lagras för potentiell återanvändning. I detta sammanhang finns det tre huvudsakliga cache stores: **Object Plans**, **SQL Plans**, och **Bound Trees**. Varje cache store har sin specifika roll i lagringen av olika typer av kompilerade planer och strukturer.

1. **Object Plans**: Dessa planer lagrar föremål som lagrade procedurer, funktioner och triggers. När SQL Server kompilera en fråga som är relaterad till någon av dessa objekt, sparas den kompilerade planen i Object Plans cache store. Eftersom objekt som lagrade procedurer ofta exekveras flera gånger, är det fördelaktigt att kunna återanvända dessa kompilerade planer, vilket minskar overheaden från ständig nykompilering.

2. **SQL Plans**: Denna cache store innehåller kompilerade planer för ad hoc-SQL-frågor, inklusive frågor som skrivs dynamiskt eller manuellt. Dessa frågor tenderar att vara unika, men SQL Server optimerar ändå genom att spara deras planer om de skulle köras igen med samma struktur eller parametrar. Detta är särskilt användbart i scenarier där samma fråga ställs flera gånger med små justeringar, eftersom en redan kompilerad plan kan återanvändas, så länge frågans struktur och parametrar är desamma.

3. **Bound Trees (Algebrizer Trees)**: Dessa träd används främst för vyer. En vy är en lagrad SQL-fråga som exekveras när den anropas, och algebrizer-träd representerar den interna logiska strukturen av frågan som används för att exekvera vyn. Dessa träd används för att optimera exekveringen av vyer genom att snabbt kunna referera till den underliggande logiken och

strukturen utan att behöva kompilera om varje gång vyn körs.

Plan Cache och Plan Handle

När en plan kompileras, tilldelas den en plats i en **hash-bucket** i den relevanta plan cache store, och den identifieras unikt av ett **plan handle**. Detta handle fungerar som en referens som SQL Server använder för att snabbt hitta planen i cachen när den behöver återanvändas. En **hash-bucket** är en struktur som används för att snabba upp sökningar och matchningar i cachen. Genom att hash-värden används kan SQL Server snabbt avgöra om en specifik plan redan finns lagrad och om den kan återanvändas.

Återanvändning av planer

För att en plan ska kunna återanvändas måste SQL Server säkerställa att både **bucket-hash** och **plan cache-nyckeln** matchar med en redan existerande plan i cachen. Detta innebär att samma fråga, eller en fråga med identiska parametrar och struktur, måste ställas. Om matchningen lyckas, återanvänds planen, vilket drastiskt minskar tiden som krävs för att exekvera frågan. Återanvändningen av planer är avgörande för att upprätthålla en hög prestanda, särskilt i system där många likartade frågor ställs upprepade gånger.

Exekverbara planer

En **exekverbar plan** är en **sessionsspecifik** instans av en kompilerad plan. Detta innebär att när en fråga kompileras och en plan skapas, skapar SQL Server en exekverbar instans av denna plan för varje session där frågan exekveras. Denna exekverbara plan innehåller en **arbetsyta** som används för att lagra variabelvärden och annan sessionsspecifik information under exekveringen av frågan. Detta tillvägagångssätt gör att SQL Server kan hantera frågor som exekveras parallellt av olika användare eller sessioner, utan att skapa konflikter mellan olika exekveringar av samma fråga.

Sammanfattning

Sammanfattningsvis är plan cache och de tre cache stores som nämns (Object Plans, SQL Plans, och Bound Trees) grundläggande för att SQL Server ska kunna optimera exekveringen av frågor. Genom att spara kompilerade planer och återanvända dem vid behov, minskar SQL Server behovet av att kompilera om samma frågor, vilket förbättrar prestanda och minskar belastningen på systemet. Samtidigt möjliggör exekverbara planer flexibel och effektiv hantering av variabler och sessioner under exekvering.

SQL Server, Plan Cache object
https://docs.microsoft.com/en-us/sql/relational-databases/performance-monitor/sql-server-plan-cache-object

Examining the Performance Impact of an Adhoc Workload
https://sqlperformance.com/2019/05/sql-plan/perf-impact-adhoc-workload

Plan cache management

- Plan cache housekeeping will begin when the cache stores hit threshold values
- The least-expensive plans are removed first from the plan cache
- You can manually clear the plan cache in various ways

Att rensa plan cachen i SQL Server kan vara nödvändigt i olika scenarier, till exempel vid prestandaproblem, efter schemaändringar eller när man vill tvinga omkompilering av frågeplaner. Här är de olika metoderna jag känner till för att rensa plan cachen:

1. **DBCC FREEPROCCACHE**

   Detta kommando rensar hela plan cachen på servern, vilket innebär att alla kompilerade frågeplaner tas bort.

   ```sql
   DBCC FREEPROCCACHE;
   ```

   Anmärkning: Använd detta med försiktighet, eftersom det kan påverka prestandan negativt när alla framtida frågor måste kompilera nya planer.

2. **DBCC FLUSHPROCINDB**

   Detta kommando rensar plan cachen för en specifik databas genom att ange databas-ID.

   ```sql
   DECLARE @dbid INT = DB_ID('DittDatabasNamn');
   DBCC FLUSHPROCINDB(@dbid);
   ```

Anmärkning: Detta är användbart när du vill begränsa rensningen till en viss databas.

3. **ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE**

Om du använder SQL Server 2016 eller senare kan du rensa plan cachen för en specifik databas med följande kommando:

```sql
ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE;
```

Anmärkning: Detta påverkar endast den angivna databasen och är användbart för att minimera påverkan på andra databaser på samma server.

4. **sp_recompile**

Denna systemlagrade procedur markerar en specifik lagrad procedur, vy eller funktion för omkompilering nästa gång den körs.

```sql
EXEC sp_recompile N'DittObjektNamn';
```

Anmärkning: Detta tvingar endast omkompilering för det specifika objektet och påverkar inte resten av plan cachen.

5. **WITH RECOMPILE**

När du exekverar en lagrad procedur eller en fråga kan du använda `WITH RECOMPILE` för att säkerställa att den kompileras om varje gång.

För lagrade procedurer:

```sql
EXECUTE DittProcedureNamn WITH RECOMPILE;
```

För ad hoc-frågor:

```sql
SELECT * FROM DinTabell OPTION (RECOMPILE);
```

```

*Anmärkning*: Detta påverkar endast den aktuella exekveringen och inte plan cachen som helhet.

6. **Ändra inställningar som påverkar plan cachen**

Vissa inställningar och åtgärder kan indirekt rensa eller påverka plan cachen:

- **ALTER DATABASE COLLATE**: Ändra kollation på en databas.
- **ALTER DATABASE SET COMPATIBILITY_LEVEL**: Ändra kompatibilitetsnivån för en databas.
- **ALTER SERVER CONFIGURATION**: Ändra vissa serverkonfigurationer.
- **Statistikuppdateringar**: Även om de inte direkt rensar plan cachen, kan uppdatering av statistik leda till att nya frågeplaner genereras.

7. **Återstarta SQL Server-tjänsten**

Att återstarta SQL Server-tjänsten kommer naturligtvis att rensa hela plan cachen eftersom allt i minnet förloras.

*Anmärkning*: Detta är en drastisk åtgärd och påverkar tillgängligheten av databasen.

8. **Droppa och återskapa index eller ändra tabellstruktur**

Strukturförändringar i databasen, såsom att droppa eller skapa index, eller ändra tabellens schema, kan leda till att relaterade frågeplaner tas bort från cachen.

9. **DBCC DROP CLEANBUFFERS**

Även om detta kommando främst används för att rensa data cache (buffertpoolen), används det ofta i kombination med `DBCC FREEPROCCACHE` vid prestandatester.

```sql
DBCC DROP CLEANBUFFERS;
```

Anmärkning: Detta påverkar data cache och kan påverka prestandan för disk I/O.

Viktiga överväganden
```

- **Prestandapåverkan**: Rensning av plan cachen kan leda till ökad CPU-användning och svarstider eftersom frågor måste kompileras om.
- **Begränsa påverkan**: När det är möjligt, rikta in dig på specifika planer eller databaser istället för att rensa hela cachen.
- **Säkerhet**: Se till att du har tillräckliga behörigheter för att utföra dessa åtgärder, vanligtvis krävs `sysadmin` eller `db_owner` rättigheter.

Sammanfattning

Att rensa plan cachen i SQL Server kan göras på flera sätt beroende på behovet. Från att rensa hela cachen med `DBCC FREEPROCCACHE` till att tvinga omkompilering av en enskild lagrad procedur med `sp_recompile`, erbjuder SQL Server flexibla verktyg för cachehantering. Det är viktigt att förstå konsekvenserna av dessa åtgärder och använda dem ansvarsfullt för att undvika oönskad påverkan på systemets prestanda.

SQL Server Plan Cache Limits
https://www.sqlskills.com/blogs/erin/sql-server-plan-cache-limits/

Eight Different Ways to Clear the SQL Server Plan Cache
https://www.sqlskills.com/blogs/glenn/eight-different-ways-to-clear-the-sql-server-plan-cache/

Queries without plan cache entries

- Queries that cannot be cached:
  - Ad hoc and prepared T-SQL statements requiring object name resolution
- Queries marked for recompilation:
  - CREATE…WITH RECOMPILE
  - OPTION (RECOMPILE)
  - EXECUTE…WITH RECOMPILE
- Natively compiled procedures for memory-optimized tables

Maximizing plan cache efficiency

- Auto-parameterization
  - Simple is default, only used for queries with an evenly distributed cardinality
  - Forced can drastically improve cache re-use for some workloads
    - But can suffer from parameter sniffing
    - Consider combining with below
- Optimize for ad hoc workloads
  - Available on both instance and database level
  - Ad hoc plans are cached only on second execution
  - Saves memory but not CPU
- Methods used by various APIs and tools
  - Sp_executesql
    - Does parameter sniffing
  - Sp_prepare, sp_execute etc
    - Doesn't use parameter sniffing, it uses the same mechanism as when you use a T-SQL variable
- Object Plans (stored procedures, triggers, functions)
  - Does parameter sniffing

Auto-parameterization i SQL Server är en funktion som automatiskt ersätter hårdkodade litteraler med parametrar för att öka chansen att återanvända frågeplaner och därmed förbättra prestandan. Det finns två huvudsakliga typer av auto-parameterization: **Simple** och **Forced**.

1. **Simple parameterization**:
   - Detta är standardläget i SQL Server och tillämpas automatiskt på enkla frågor med en jämn fördelning av kardinaliteten (antalet unika värden i en kolumn).
   - Exempelvis, om du kör en enkel SELECT-sats som inkluderar ett konstant värde, kan SQL Server ersätta det konstanta värdet med en parameter och återanvända frågeplanen om samma fråga exekveras igen med ett annat värde.
   - Den här typen är begränsad till enkla frågor och används inte för mer komplexa frågor.

2. **Forced parameterization**:
   - Forced parameterization kan aktiveras på databasnivå och tvingar SQL Server att parametrisera en bredare uppsättning frågor, inklusive mer komplexa.
   - Detta kan drastiskt förbättra cache-återanvändningen för vissa arbetsbelastningar, särskilt om frågor med små variationer körs ofta.
   - Men forced parameterization kan ibland leda till problem med **parameter sniffing**, där SQL Server optimerar en frågeplan baserat på de initiala värdena på parametrarna, vilket kan leda till ineffektiva planer för framtida exekveringar med andra parametrar. Detta problem uppstår eftersom en frågeplan kanske inte är optimal för alla möjliga värden på de parametrar som används.

Optimize for Ad Hoc Workloads

Ett alternativ för att förbättra effektiviteten är att använda inställningen **Optimize for ad hoc workloads**. Detta är tillgängligt både på instans- och databasenivå och hjälper till att minska minnesanvändningen genom att endast cacha frågeplaner efter den andra exekveringen av en fråga. Vid första exekveringen cachas bara en liten stub som fungerar som en pekare till själva frågeplanen. Om samma fråga körs igen, genereras den

fullständiga planen och cachas för framtida användning.

Denna funktion är idealisk för system där många unika ad hoc-frågor körs en gång, vilket kan spara minne. Däremot minskar det inte CPU-kostnaderna, eftersom frågan ändå måste kompileras första gången.

Metoder och API-anrop

SQL Server använder flera olika metoder och API:er för att exekvera frågor, var och en med sin egen hantering av parameterisering och cacheåteranvändning.

1. **sp_executesql**:
   - Denna metod används för att exekvera parametriserade SQL-frågor och återanvänder frågeplaner genom parameterisering.
   - En fördel med `sp_executesql` är att den gör **parameter sniffing**, vilket innebär att frågeplanen optimeras baserat på de faktiska parametrarna som skickas vid första exekveringen.
   - Detta kan vara fördelaktigt om de första parametrarna är representativa för den typ av frågor som kommer att köras framöver. Men det kan också leda till problem om frågeplanen inte är optimal för andra parametrar.

2. **sp_prepare och sp_execute**:
   - Dessa lagrade procedurer används för att förbereda och exekvera parametriserade frågor och är vanliga i applikationer som använder ADO.NET eller ODBC.
   - Till skillnad från `sp_executesql` använder dessa procedurer inte parameter sniffing. Istället behandlar de parametrarna på samma sätt som när man använder variabler i T-SQL, vilket innebär att SQL Server inte försöker optimera frågeplanen baserat på de faktiska parametrarna vid körning.
   - Detta kan leda till mer konsekventa prestanda i scenarier där parameter sniffing kan orsaka problem, men det kan också resultera i mindre optimala frågeplaner i vissa fall.

3. **Object Plans (lagrade procedurer, triggers, funktioner)**:
   - När det gäller objektplaner, såsom lagrade procedurer, triggers och funktioner, använder SQL Server parameter sniffing för att optimera frågeplanen vid första exekveringen.
   - Precis som med `sp_executesql` kan detta leda till prestandaproblem om den initiala frågeplanen inte är lämplig för framtida exekveringar med andra parametrar.

Sammanfattning

Auto-parameterization och cacheåteranvändning är viktiga komponenter för att optimera prestandan i SQL Server. Med **Simple parameterization** är det en standardfunktion som tillämpas på enkla frågor, medan **Forced parameterization** kan tvinga fram en bredare tillämpning av parametrisering, även om den kan leda till problem med parameter sniffing. Funktionen **Optimize for ad hoc workloads** hjälper till att spara minne genom att endast cacha planer vid andra exekveringen. Slutligen hanterar olika metoder som `sp_executesql`, `sp_prepare` och lagrade procedurer parametrisering och cache på olika sätt, vilket påverkar prestandan beroende på arbetsbelastning och användningsfall.

sp_executesql (Transact-SQL)

https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-executesql-transact-sql

sp_prepare (Transact SQL)
https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-prepare-transact-sql

optimize for ad hoc workloads Server Configuration Option
https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/optimize-for-ad-hoc-workloads-server-configuration-option

How to Reduce the CPU Overhead of Dynamic SQL
https://www.brentozar.com/archive/2018/05/using-dynamic-sql-frequently-executed-queries/

Specify Query Parameterization Behavior by Using Plan Guides
https://docs.microsoft.com/en-us/sql/relational-databases/performance/specify-query-parameterization-behavior-by-using-plan-guides

Blitz Result: Forced Parameterization
https://www.brentozar.com/blitz/forced-parameterization/

SQL Server Simple and Forced Parameterization
https://www.mssqltips.com/sqlservertip/2935/sql-server-simple-and-forced-parameterization/

Can Forced Parameterization Go Wrong?
https://www.brentozar.com/archive/2018/09/can-forced-parameterization-go-wrong/

Query parameterization settings
https://sqlblog.karaszi.com/query-parameterization-settings/

Examining the Plan Cache

- sys.dm_exec_cached_plans
  - One row per cached plan
- sys.dm_exec_query_plan
  - Query plan in XML format
- sys.dm_exec_text_query_plan
  - Query plan in text format
- sys.dm_exec_plan_attributes
  - Plan attributes
- sys.dm_exec_query_stats and sys.dm_exec_procedure_stats
  - Plan statistics

Execution Related Dynamic Management Views and Functions (Transact-SQL)
https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/execution-related-dynamic-management-views-and-functions-transact-sql

Lesson: Troubleshooting the plan cache

- Execution plan recompilation
- Recompilation issues
- Problems of the plan cache
- Using the plan cache to guide optimization
- T-SQL query anti-patterns

Execution plan recompilation

- Recompilation occurs when a cached plan is invalidated and a new plan is compiled
- A cached plan may become invalid because:
  - It becomes incorrect (like a schema change)
  - It becomes non-optimal (like a statistics update)
- Recompilation can be tracked using
  - Extended Events
  - Windows Performance Monitor

Understanding SQL Server Recompilations
https://www.mssqltips.com/sqlservertip/5308/understanding-sql-server-recompilations/

Recompilation issues

- Parameter sniffing—fewer recompiles than expected. Address with:
  - OPTION(RECOMPILE)
  - OPTION(OPTIMIZE FOR...)
- Statistics changes
  - KEEP PLAN query hint reduces recompilations caused by statistics changes
  - KEEPFIXED PLAN query hint prevents recompilations caused by statistics changes

Hints (Transact-SQL) – Query
https://docs.microsoft.com/en-us/sql/t-sql/queries/hints-transact-sql-query

KEEP PLAN Demystified
https://straightforwardsql.com/posts/keep-plan-demystified/

Problems of the plan cache

- Plan cache bloat:
  - Caused by multiple copies of the same query execution plan entering the cache
  - Addressed by:
    - Code changes
    - Optimize for ad hoc workloads setting (server or database level)
    - FORCED PARAMETERIZATION database level

Optimize for ad hoc workloads är en inställning i SQL Server som hjälper till att hantera och optimera användningen av plan cache för tillfälliga eller "ad hoc" frågor. När den är aktiverad, lagrar SQL Server en kompakt version av exekveringsplanen för en ad hoc-fråga vid första körningen istället för hela planen. Om samma fråga körs igen, och SQL Server känner igen den, lagras den fullständiga exekveringsplanen i cache. Detta sparar minne eftersom det förhindrar cache från att fyllas med planer som kanske bara används en gång, vilket gör systemet mer effektivt för verkliga arbetsbelastningar.

Forced parameterization i SQL Server är en inställning som ökar databasens förmåga att återanvända exekveringsplaner för liknande frågor. Genom att tvinga parameterisering, behandlar SQL Server konstanta värden i frågor som parametrar, vilket gör att fler frågor kan matcha och återanvända befintliga exekveringsplaner. Detta kan minska belastningen på SQL Server genom att minska antalet gånger den måste skapa nya exekveringsplaner, vilket leder till bättre prestanda. Men det kan också leda till suboptimala exekveringsplaner i vissa fall, så det är viktigt att testa och övervaka dess effekt noggrant.

Using the Plan Cache to Guide Optimization

- Many performance measures are captured for cache plans in sys.dm_exec_query_stats and sys.dm_exec_procedure_stats
- You can use these DMVs to identify candidate statements for optimization and performance tuning
  - Query store is way superior for these purposes, though

T-SQL query anti-patterns

- Avoid below if you care about performance
  - Cursors
  - Using DISTINCT indiscriminately
  - Scalar functions
    - Unless you are on 2019 with database compatibility level 2019
      - Verify that UDF inlining work for you, though
  - Calculations on the column-side
    - YEAR(OrderDate) = 2022
  - Non-matching data types
    - Pass the value from the client with the same type as the column is defined in the table

Demo Match Data Types

T-SQL Anti-Patterns: SQL User Defined Functions (UDFs) that turn your set operation into a cursor
https://sqlsolutionsgroup.com/t-sql-anti-patterns-user-defined-functions/

Scalar functions and improvements in SQL Server 2019
https://sqlblog.karaszi.com/scalar-functions-and-improvements-in-sql-server-2019/

Scalar functions in SQL server 2019, part 2
https://sqlblog.karaszi.com/scalar-functions-in-sql-server-2019-part-2/

Match those types!
https://sqlblog.karaszi.com/match-those-types/

AddWithValue is Evil
https://www.dbdelta.com/addwithvalue-is-evil/

Impact of Eliminating Implicit Conversions in JDBC Driver
https://tracyboggiano.com/archive/2023/01/eliminating-implicit-conversions-jdbc-driver/

Lesson: Query Store

- What is Query Store?
- Enabling Query Store
- Configuring Query Store
- Accessing Query Store Data
- Forcing Query Execution Plans
- Plan hinting through Query Store

What is Query Store?

- Stores queries, plans, and run-time statistics
- Highlights database pattern usage
- Helps troubleshoot query performance issues
- Compatible with on-premises and Azure SQL databases

Monitoring performance by using the Query Store
https://docs.microsoft.com/en-us/sql/relational-databases/performance/monitoring-performance-by-using-the-query-store

Enabling Query Store

- Switch on using ALTER DATABASE
  - `ALTER DATABASE dbname SET QUERY_STORE = ON;`
- To use SSMS, right-click database, click Properties, and then on the Query Store page, change Operation Mode
- Cannot be turned on for system databases
- On by default in SQL Server 2022
  - For new databases
- Should we turn it on?
  - Run sp_BlitzCache, check 2:nd resultset.
  - If we have priority 1 warning, Query Store might not be for us.

Best practices with Query Store
https://docs.microsoft.com/en-us/sql/relational-databases/performance/best-practice-with-the-query-store

Query Store Best Practices
https://www.sqlskills.com/blogs/erin/query-store-best-practices/

Att aktivera Query Store kan vara mycket fördelaktigt för övervakning och optimering av SQL Server-prestanda. Men om sp_BlitzCache identifierar Priority 1-varningar i din miljö, är det viktigt att först åtgärda dessa problem. Annars kan aktivering av Query Store förvärra befintliga prestandaproblem.
Genom att noggrant utvärdera och förbereda din miljö kan du dra nytta av Query Store utan att kompromissa med systemets stabilitet och prestanda.

Configuring Query Store

- View Query Store options using sys.database_query_store_options
- Configure Query Store options using ALTER DATABASE SET QUERY STORE (OPTION = VALUE)
- View and configure Query Store options using SSMS

Query store data cleanup
https://www.scarydba.com/2023/02/06/query-store-data-cleanup/

Accessing Query Store data

- Access query store data through catalog views or SSMS
- SSMS:
  - Regressed Queries
  - Overall Resource Consumption
  - Top Resource Consuming Queries
  - Tracked Queries
  - …
- Consider using Erik Darling's sp_QuickieStore

Demo Query store

The SQL Server 2016 Query Store: Built-in Reporting
https://www.red-gate.com/simple-talk/databases/sql-server/database-administration-sql-server/the-sql-server-2016-query-store-built-in-reporting/

Does Query Store's "Regression" Always Catch Nasty Parameter Sniffing?
https://www.littlekendra.com/2016/01/21/query-store-regression-parameter-sniffing/

Introducing sp_QuickieStore: Find Your Worst Queries In Query Store
https://www.erikdarlingdata.com/sp_quickiestore/introducing-sp_quickiestore-find-your-worst-queries-in-query-store-fast/

List of blog posts, including some that give tips on using sp_QuickieStore for various purposes
https://www.brentozar.com/archive/2022/10/erik-darlings-month-of-free-tools-training/

Forcing Query Execution Plans

- SSMS:
  - Click Force Plan button when viewing a query plan in the Query Store
  - Click Unforce Plan button to undo
- Transact-SQL:
  - Use sp_querystore_force_plan to force a plan
  - Use sp_querystore_unforce_plan to unforce a plan
- Regularly check the performance of forced plans
- Optimized plan forcing (2022)
  - Reduce compile time for forced queries

sp_query_store_force_plan (Transact-SQL)
https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-query-store-force-plan-transact-sql

Why You Need Query Store, Part II: Plan forcing
https://www.sqlskills.com/blogs/erin/why-you-need-query-store-part-ii-plan-forcing/

Forcing Query Execution Plans (continued)

- Optimized plan forcing (2022)
  - Reduce compile time for forced queries
  - On by default
  - Plan has to be produced also for forced plans
  - *Compilation script* is saved with the plan
    - This script is what reduces the compile time
    - Only for queries with "high enough" compile time
  - Can be seen as binary data in query plan in the sys.query_store_plan view
    - But not if you cast the column to XML
    - Is also marked as having a script in the column has_compile_replay_script in sys.query_store_plan
  - Typical load pattern is burst of plan production after mass plan evictions
    - Re-start, failover, severe memory pressure
    - Very high CPU usage, system "unresponsive"

Optimized plan forcing with Query Store
https://learn.microsoft.com/en-us/sql/relational-databases/performance/optimized-plan-forcing-query-store

Plan hinting through Query Store

- Introduced in SQL Server 2022
- Doesn't require a certain compatibility level
- Find the query ID
- Use sp_query_store_set_hints for that ID
  - `EXEC sp_query_store_set_hints 42, N'OPTION(MAXDOP 1, USE HINT(''QUERY_OPTIMIZER_COMPATIBILITY_LEVEL_120''))'`
  - `EXEC sp_query_store_clear_hints 42`

Plan hint Query Store

sp_query_store_force_plan (Transact-SQL)
https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-query-store-force-plan-transact-sql

Why You Need Query Store, Part II: Plan forcing
https://www.sqlskills.com/blogs/erin/why-you-need-query-store-part-ii-plan-forcing/

`sp_query_store_set_hints` är en stored procedure i SQL Server som låter dig ange specifika frågehints för en given fråga som lagras i Query Store. Med den här funktionen kan du manuellt påverka hur databasens optimeringsmotor behandlar en specifik fråga, utan att ändra själva frågetexten. Detta är särskilt användbart för att finjustera prestandan av enskilda frågor genom att ange hur de bör köras, exempelvis genom att tvinga fram en viss join-typ eller använda en specifik index. Det hjälper databasadministratörer att hantera prestandaproblem genom att direkt ingripa och styra frågeexekveringsplanerna på ett mer detaljerat sätt.

Lesson: Automatic tuning

- What is Automatic Tuning?
- sys.dm_db_tuning_recommendations
- Plan choice correction

Automatic Tuning i SQL Server är en kraftfull funktion som syftar till att automatiskt optimera prestandan av din databas. Den identifierar potentiella prestandaproblem, som långsamma frågor, och föreslår eller implementerar automatiskt korrigeringar. De två huvudsakliga funktionerna i Automatic Tuning är **FORCE_PLAN** och **CREATE_INDEX**, där den första tvingar databasen att använda en specifik exekveringsplan för en fråga som historiskt sett har presterat bättre, och den andra automatiskt skapar index baserat på samlade prestandadata för att förbättra frågeexekveringens hastighet.

Ett exempel på hur du kan aktivera Automatic Tuning på databasnivå för att automatiskt tillämpa de bästa exekveringsplanerna och indexförslagen är:

```sql
ALTER DATABASE currentDb
SET AUTOMATIC_TUNING ( FORCE_LAST_GOOD_PLAN = ON, CREATE_INDEX = ON, DROP_INDEX = OFF );
```

I detta exempel:

- `FORCE_LAST_GOOD_PLAN = ON` aktiverar funktionen som automatiskt tvingar fram användningen av den senast observerade "goda" exekveringsplanen för en fråga, om prestandan försämras av någon anledning.
- `CREATE_INDEX = ON` tillåter SQL Server att automatiskt skapa nya index baserat på dess analys av frågeprestanda och optimeringsbehov.
- `DROP_INDEX = OFF` innebär att automatisk borttagning av index är inaktiverat. Du kan

aktivera detta om du vill, men det kan vara klokt att vara försiktig med automatisk indexhantering och istället granska förslagen manuellt.

Automatic Tuning tar en proaktiv approach till databasprestanda, vilket kan spara mycket tid och resurser genom att minska behovet av manuell tuning och övervakning. Dock är det viktigt att övervaka hur dessa automatiska ändringar påverkar databasens prestanda över tid för att säkerställa att de ger önskad effekt.

What is Automatic Tuning?

- Requires Enterprise Edition
- Changes in query execution plans that impact performance can be time consuming to find and fix
- Plan choice regression is a recompiled plan that results in poorer performance
- Automatic tuning
  - Collects data about query performance and execution plans
  - Finds links between plan changes and reduced performance
  - Generates a script to force the previous plan to be used
  - Can be configured to automatically apply the previous script

Automatic tuning
https://docs.microsoft.com/en-us/sql/relational-databases/automatic-tuning/automatic-tuning

Understanding automatic tuning in SQL Server 2017
https://www.sqlshack.com/understanding-automatic-tuning-in-sql-server-2017/

sys.dm_db_tuning_recommendations

- sys.dm_db_tuning_recommendations returns:
  - A score between 0 and 100 to indicate the anticipated performance impact
  - A JSON string containing the recommendations, including:
    - Metrics used to identify plan choice regression
    - Commands used to apply the recommendation
- Contents of sys.dm_db_tuning_recommendations are retained until the instance is restarted

sys.dm_db_tuning_recommendations (Transact-SQL)
https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-db-tuning-recommendations-transact-sql

Sql Server automatic tuning and sys.dm_db_tuning_recommendations
https://www.scarydba.com/2017/12/26/sql-server-automatic-tuning-and-sys-dm_db_tuning_recommendations/

`sys.dm_db_tuning_recommendations` är en dynamisk hanteringsvy i SQL Server som ger insikter om rekommendationer för automatisk optimering av databasen. Denna vy innehåller detaljerad information om potentiella prestandaförbättringar, som Automatic Tuning-funktionen kan identifiera, inklusive rekommendationer för att skapa nya index, ta bort oanvända eller ineffektiva index, eller ändra frågeplaner för att optimera exekveringen. För varje rekommendation tillhandahåller vyn information såsom den berörda frågans identifierare, den specifika åtgärden som föreslås, och en skattning av den potentiella prestandaförbättringen. Genom att analysera dessa rekommendationer kan databasadministratörer fatta informerade beslut om att manuellt eller automatiskt tillämpa föreslagna ändringar för att förbättra databasens övergripande prestanda.

Plan choice correction

- Automatic plan choice correction:
  - Enabled at the database level
  - Plan choice recommendations are forced when:
    - CPU time gain > 10 seconds
    - Fewer errors in the recommended plan
  - Results are monitored
    - If performance does not improve, the plan is recompiled
  - Manual plan choice correction
    - Manually apply recommendations from the DMV
    - Manually monitor performance and act on findings
- Azure SQL Database also has auto create and drop index

Demo Automatic tuning

Automatic Plan Choice Correction, en del av SQL Server's Automatic Tuning-funktionalitet, är en avancerad mekanism designad för att automatiskt identifiera och korrigera suboptimala frågeexekveringsplaner som påverkar databasprestandan negativt. Detta system bygger på förmågan att övervaka och analysera frågeprestanda över tid, jämföra olika exekveringsplaner för samma fråga och identifiera när en tidigare effektiv plan blir mindre optimal på grund av ändringar i databasens tillstånd, som datafördelning eller indexändringar.

När Automatic Plan Choice Correction upptäcker en fråga vars prestanda har försämrats, kan den automatiskt återställa användningen till en tidigare, mer effektiv exekveringsplan, vilket ofta resulterar i omedelbara prestandaförbättringar. Denna process minimerar behovet av manuell ingripande från databasadministratörer, vilket sparar tid och resurser och hjälper till att bibehålla en jämn och förutsägbar prestandanivå.

Genom att använda historisk exekveringsplandata och prestandametriker tillåter Automatic Plan Choice Correction SQL Server att anpassa sig till förändrade förhållanden på ett dynamiskt sätt, vilket säkerställer att databasen kontinuerligt optimeras för de bästa möjliga exekveringstiderna. Denna funktion är särskilt värdefull i stora och komplexa databasmiljöer där små förändringar kan ha stor inverkan på den totala prestandan. Automatic Plan Choice Correction representerar en kraftfull komponent i SQL Server's verktygslåda för databasoptimering, vilket gör att systemet kan upprätthålla optimal effektivitet genom proaktiv och automatisk anpassning.

Lab 8: Plan caching and recompilation

- Ex 1: Fixing stored procedure with plan issues
- Ex 2: Server is gradually slowing down

Estimated Time: 30 minutes