Module 7

Query execution and query plan analysis

Module Overview

- Query execution and query optimizer internals
- Query execution plans
- Analyzing query execution plans
- Intelligent Query Processing

Lesson: Query execution and query optimizer internals

- Phases of query processing
- The optimizer

Phases of query processing

- Parsing:
  - Validate syntax
  - Output: logical query tree (parse tree)
  - Traceflag 8605, 6 and 7 show us various trees
- Binding:
  - Take logical query tree and bind it to database objects
  - Output: algebraizer tree
- Query Optimization:
  - Take algebraizer tree and ***create query execution plan***
  - Output: query execution plan
- Query Execution:
  - Execute query execution plan
  - Output: results

Show query tree

Query Processing Architecture Guide
https://docs.microsoft.com/en-us/sql/relational-databases/query-processing-architecture-guide

Query Optimization in SQL Server for beginners
https://www.sqlshack.com/query-optimization-in-sql-server-for-beginners/

Parsing
Parsingfasen i SQL Server är det första steget i query-processen, där SQL Server granskar och analyserar den inkommande SQL-frågan för att verifiera syntaxen och säkerställa att den följer SQL Server-reglerna. Parsern bryter ned frågan till en serie tokens och kontrollerar för syntaxfel. Om frågan innehåller syntaxfel, genereras ett felmeddelande och processen avbryts. Parsing är kritiskt för att förstå frågans struktur och förbereda för nästa steg. Denna fas handlar inte om dataåtkomst eller hur frågan kommer att utföras, utan endast om att förstå och validera frågans uppbyggnad.

Binding och Algebraizer
Efter parsingfasen kommer binding och skapandet av en algebraizer tree. Bindingfasen involverar att lösa namn och referenser i frågan, såsom tabell- och kolumnnamn, mot databasens schema. SQL Server måste verifiera att alla refererade objekt existerar och att användaren har rätt behörighet att åtkomma dem. När bindingen är klar skapas en algebraizer tree (ibland kallad query tree), som representerar frågan i en strukturerad, hierarkisk form. Denna trädstuktur hjälper SQL Server att förstå relationerna mellan olika delar av frågan och är grundläggande för att optimera och utföra frågan effektivt. Algebraizer-trädet omvandlar den ursprungliga SQL-frågan till en form som mer direkt återspeglar hur data ska hämtas, filtreras och kombineras,

vilket banar väg för den efterföljande optimeringsprocessen.

Efter att en fråga har analyserats och omvandlats till en algebraisk trädstruktur, följer steg två: query optimization. I denna fas använder SQL Server sin frågeoptimerare för att utvärdera olika sätt att utföra frågan med målet att hitta den mest effektiva utförandeplanen. Optimeraren bedömer olika exekveringsstrategier, eller "planer", baserat på kostnadsbaserade algoritmer som tar hänsyn till faktorer såsom antalet rader som måste bearbetas, index som är tillgängliga, och den förväntade resursanvändningen för varje operation. Denna process involverar en omfattande analys av möjliga utförandevägar och slutar med en "bästa" plan som sedan utförs för att hämta eller manipulera data. Optimeringsfasen är avgörande för att säkerställa hög prestanda och effektiv resursanvändning i databasen.

The optimizer

- Algebraizer tree + schema + statistics + transformation rules = potential query execution plan
- Cost-based optimization:
  - Not all plans are considered
  - Of those considered, lowest-cost plan will be returned
  - Multiple optimization phases:
    - Simplification
    - Trivial plan
    - Full optimization
      - Search 0
      - Search 1
      - Search 2

Demo optimizer

Query Processing Architecture Guide
https://docs.microsoft.com/en-us/sql/relational-databases/query-processing-architecture-guide

Ingående förklaring av SQL Servers frågaoptimeringsprocess

När en SQL-fråga skickas till SQL Server, genomgår den flera steg innan den exekveras. Ett av de mest kritiska stegen är frågaoptimering, där målet är att generera den mest effektiva exekveringsplanen. Här är en detaljerad genomgång av processen:

---

1. Generering av Potentiella Exekveringsplaner

Algebraizer Tree

- Efter att frågan har syntaktiskt analyserats, omvandlas den till en intern representation kallad **Algebraizer Tree**.
- Detta träd representerar frågans logiska struktur, inklusive tabeller, kolumner och operatorer.

Schema

- **Schemat** innehåller metadata om databasobjekten: tabeller, index, vyer och deras relationer.
- Det ger information om kolumndatatyper, primär- och främmande nycklar, och andra

constraints som påverkar hur data kan nås och manipuleras.

Statistik

- **Statistik** ger insikt i datafördelningen inom tabeller och index.
- Innehåller information om antal rader, kolumnvärdesfördelningar och korrelationer mellan kolumner.
- Används för att uppskatta kostnader för olika exekveringsplaner.

Transformationsregler

- En uppsättning regler som optimeraren använder för att omvandla frågeträdet till alternativa planer.
- Inkluderar omskrivningar som att byta ordning på joins, använda alternativa index, eller omvandla subqueries till joins.

Sammanfattning: Genom att kombinera Algebraizer Tree, schema, statistik och transformationsregler genererar SQL Server flera potentiella exekveringsplaner som kan användas för att exekvera frågan.

---

2. Kostnadsbaserad Optimering

Inte alla planer övervägs

- På grund av det enorma antalet möjliga exekveringsplaner för komplexa frågor är det inte praktiskt att överväga alla.
- SQL Server använder heuristiker och begränsar sökrymden för att göra optimeringen effektiv.

Lägsta kostnadsplan väljs

- Varje potentiell plan tilldelas en **kostnad** baserad på uppskattad resursförbrukning (CPU, minne, I/O).
- **Kostnadsbaserad optimering** innebär att den plan med lägst uppskattad kostnad väljs för exekvering.
- Kostnaden beräknas med hjälp av statistik och tar hänsyn till faktorer som indexanvändning och join-metoder.

---

3. Flera Optimeringsfaser

Optimeringen delas upp i flera faser för att balansera tid och prestanda:

a. Förenkling (Simplification)

- Initialt förenklas frågan genom att eliminera redundanta uttryck och utvärdera konstanta uttryck.
- Omskrivningar kan inkludera att förenkla WHERE-villkor och eliminera onödiga joins.

b. Trivial Plan

- Om frågan är enkel nog kan en **trivial plan** genereras utan omfattande optimering.
- Används för frågor som enkla SELECT- eller INSERT-operationer på en enda tabell utan komplexa villkor.

c. Fullständig Optimering

Om ingen trivial plan är tillräcklig, genomförs en fullständig optimering i flera sökfaser:

Search 0

- **Grundläggande optimering** med fokus på enkla omskrivningar och planer.
- Optimeraren överväger basala indexanvändningar och join-ordningar.

Search 1

- **Utökad optimering** där fler transformationsregler tillämpas.
- Överväger mer komplexa join-strategier och alternativa index.
- Tar hänsyn till parallellism och partitionering om tillämpligt.

Search 2

- **Avancerad optimering** för mycket komplexa frågor.
- Inkluderar djupgående analys av alla möjliga planer inom rimliga begränsningar.
- Kan vara tidskrävande och används endast om tidigare faser inte hittat en acceptabel plan.

Notering: Optimeraren kan besluta att avbryta optimeringsprocessen efter en viss tid (tidsgräns) för att inte fördröja exekveringen för mycket. Detta innebär att den

bästa möjliga planen inom den tidsramen väljs.

Sammanfattning av Processen

1. **Frågeanalys:** Frågan omvandlas till en Algebraizer Tree baserat på syntaktisk och semantisk analys.

2. **Generering av potentiella planer:** Genom att använda schema, statistik och transformationsregler skapas flera möjliga exekveringsplaner.

3. **Kostnadsberäkning:** Varje plan utvärderas baserat på uppskattad resursförbrukning.

4. **Planval:** Den plan med lägst uppskattad kostnad väljs för exekvering.

5. **Optimeringsfaser:** Processen delas upp i förenkling, trivial plan och fullständig optimering (Search 0, 1, 2) för effektivitet.

Praktiska Implikationer

- **Förståelse för optimeringsprocessen** kan hjälpa utvecklare att skriva mer effektiva frågor.
- **Uppdatering av statistik** och korrekt indexering är kritiskt för att optimeraren ska göra korrekta kostnadsberäkningar.
- **Komplexa frågor** kan ibland behöva omskrivas eller brytas upp för att underlätta optimeringen.

Genom att kombinera dessa komponenter och steg säkerställer SQL Server att frågor exekveras så effektivt som möjligt, vilket förbättrar prestanda och resursutnyttjande i databassystemet.

Lesson: Execution plans

- Estimated execution plans vs actual execution plans
- Execution plan formats
- Capturing execution plans

Estimated execution plans vs actual execution plans

- Each statement in a batch or stored procedure has its own execution plan
- Estimated execution plan
  - Plan is compiled but not executed
- Actual Execution Plan
  - Includes information about estimated and actual behavior
  - Only available when query is executed

Förståelse för Estimated och Actual Execution Plans i SQL Server

När du arbetar med SQL Server och optimerar frågor är **Execution Plans** (exekveringsplaner) ovärderliga verktyg. De ger insikt i hur SQL Server Query Optimizer väljer att exekvera dina frågor, vilket hjälper dig att identifiera och lösa prestandaproblem.

Varje uttalande har sin egen exekveringsplan

- **Individuella planer för varje uttalande:** I en batch eller lagrad procedur har **varje uttalande sin egen exekveringsplan**. Detta innebär att om du har flera SQL-uttryck i en batch, genereras en separat exekveringsplan för var och en. Detta är viktigt eftersom olika uttalanden kan ha olika prestandaegenskaper och kan kräva individuell optimering.

Estimated Execution Plan (Uppskattad exekveringsplan)

- **Planen kompileras men exekveras inte:** När du genererar en **Estimated Execution Plan** analyserar och kompilerar SQL Server frågan för att skapa en plan, men den **exekverar inte själva frågan**. Detta innebär att databasen inte påverkas av några ändringar, och inga data hämtas eller manipuleras.

- **Användning av statistik och metadata:** Den uppskattade planen baseras på tillgänglig statistik, index och metadata om tabeller och kolumner. SQL Server använder denna information för att uppskatta kostnader för olika operationssteg i planen.

- **Visar uppskattade värden:** Den innehåller uppskattade värden såsom antalet rader som förväntas bearbetas, I/O-kostnader och CPU-användning för varje steg i planen.

- **När ska man använda den?:** Den är användbar när du vill förstå hur SQL Server planerar att exekvera din fråga utan att faktiskt köra den. Detta är särskilt praktiskt för komplexa frågor som kan vara resurskrävande eller ändra data.

Actual Execution Plan (Faktisk exekveringsplan)

- **Innehåller både uppskattad och faktisk information:** En **Actual Execution Plan** genereras **efter att frågan har exekverats**. Den inkluderar både de uppskattade värdena (från den initiala planeringen) och de faktiska resultat som uppnåddes under exekveringen.

- **Kräver exekvering av frågan:** Eftersom den faktiska planen innehåller data om vad som verkligen hände under exekveringen, kan den endast genereras när frågan körs.

- **Visar faktiska värden:** Den visar faktiska antal rader bearbetade, verklig I/O, CPU-användning och andra prestandametriker. Detta gör det möjligt att jämföra de uppskattade värdena med de faktiska och identifiera eventuella avvikelser.

- **Inkluderar runtime-varningar:** Den kan också innehålla varningar om problem som uppstod under exekveringen, till exempel spill till tempdb på grund av minnesbrist.

- **När ska man använda den?:** Den är oumbärlig för prestandaoptimering eftersom den visar hur frågan faktiskt kördes, vilket hjälper till att identifiera ineffektiviteter som inte är uppenbara i den uppskattade planen.

Skillnader mellan Estimated och Actual Execution Plans

1. **Exekvering av frågan:**
   - **Estimated Plan:** Frågan kompileras men körs **inte**. Ingen data hämtas eller ändras.
   - **Actual Plan:** Frågan **körs**, och planen innehåller data om den faktiska

exekveringen.

2. **Data som presenteras:**
   - **Estimated Plan:** Visar **uppskattade** värden baserade på statistik och index.
   - **Actual Plan:** Visar både uppskattade och **faktiska** värden från exekveringen.

3. **Användningsområden:**
   - **Estimated Plan:** Bra för att snabbt förstå hur SQL Server planerar att exekvera en fråga utan att påverka databasen.
   - **Actual Plan:** Nödvändig för djupgående prestandaanalys och felsökning efter att ha observerat prestandaproblem.

Varför är detta viktigt?

- **Identifiera avvikelser:** Genom att jämföra uppskattade och faktiska värden kan du identifiera när SQL Servers uppskattningar inte stämmer överens med verkligheten, vilket kan bero på inaktuella statistik eller ooptimerade frågor.

- **Optimering av frågor:** Actual Execution Plans ger insikt i hur dina frågor verkligen körs, vilket är kritiskt för att kunna optimera dem effektivt.

- **Förstå prestandaproblem:** Om en fråga presterar dåligt kan den faktiska exekveringsplanen avslöja flaskhalsar som hög I/O, stora sorteringsoperationer eller oönskade table scans.

Hur du genererar och läser exekveringsplaner i SQL Server Management Studio (SSMS)

Estimated Execution Plan:

1. **Aktivera Estimated Execution Plan:** Klicka på ikonen för "Display Estimated Execution Plan" eller tryck på `Ctrl + L`.
2. **Generera planen:** Kör **inte** frågan; istället kommer SSMS att visa den uppskattade planen direkt.
3. **Analysera planen:** Granska de olika operatorerna och deras uppskattade kostnader.

Actual Execution Plan:

1. **Aktivera Actual Execution Plan:** Klicka på ikonen för "Include Actual Execution Plan" eller tryck på `Ctrl + M`.
2. **Exekvera frågan:** Kör frågan som vanligt.
3. **Visa planen:** Efter exekveringen visas den faktiska exekveringsplanen i ett separat fönster.
4. **Analysera planen:** Jämför uppskattade och faktiska värden för varje operator.

Exempel för att illustrera skillnaderna

Anta att du har en tabell `Orders` med en miljon rader och en fråga som hämtar ordrar för en specifik kund:

```sql
SELECT * FROM Orders WHERE CustomerID = 12345;
```

Estimated Execution Plan:

- **Uppskattad antal rader:** SQL Server kanske uppskattar att 1 000 rader matchar `CustomerID = 12345` baserat på statistik.
- **Accessmetod:** Kan visa en index seek om det finns ett index på `CustomerID`.

Actual Execution Plan:

- **Faktiskt antal rader:** Under exekveringen kanske det visar sig att det bara finns 10 ordrar för `CustomerID = 12345`.
- **Skillnad i uppskattning:** Den faktiska exekveringsplanen visar att SQL Servers uppskattning var mycket högre än det faktiska antalet, vilket kan indikera att statistik behöver uppdateras.

När uppskattningar och verklighet skiljer sig åt

- **Inaktuella statistik:** Om statistik inte är uppdaterade kan SQL Server göra felaktiga uppskattningar, vilket leder till suboptimala exekveringsplaner.
- **Data skew:** Om datafördelningen är ojämn kan det påverka uppskattningarna.

Till exempel kan vissa värden i en kolumn vara mycket vanligare än andra.
- **Komplexa uttryck:** Användning av funktioner eller komplexa uttryck i WHERE-villkor kan göra det svårare för SQL Server att göra korrekta uppskattningar.

Bästa praxis

- **Håll statistik uppdaterad:** Se till att automatisk uppdatering av statistik är aktiverad, eller uppdatera dem manuellt vid behov.
- **Analysera actual execution plans regelbundet:** Använd dem för att identifiera och lösa prestandaproblem.
- **Undvik funktioner i WHERE-villkor:** Om möjligt, undvik att använda funktioner på kolumner i WHERE-villkor, eftersom det kan förhindra indexanvändning och påverka uppskattningar.
- **Använd parametrar med försiktighet:** Parameter sniffing kan ibland leda till ineffektiva planer om parametrarna varierar mycket.

Sammanfattning

- **Estimated Execution Plan:**
  - Genereras utan att exekvera frågan.
  - Baserad på statistik och ger uppskattade värden.
  - Bra för snabb analys utan att påverka databasen.

- **Actual Execution Plan:**
  - Genereras efter att frågan har exekverats.
  - Innehåller både uppskattade och faktiska värden.
  - Nödvändig för djupgående prestandaanalys och felsökning.

Att förstå skillnaden mellan dessa två typer av exekveringsplaner är avgörande för effektiv prestandaoptimering i SQL Server. Genom att använda dem kan du identifiera problem, förstå hur SQL Server exekverar dina frågor och vidta åtgärder för att förbättra effektiviteten och prestandan i dina databassystem.

Comparing Estimated and Actual Execution Plans in SQL Server
https://www.brentozar.com/archive/2014/07/comparing-estimated-actual-execution-plans-sql-server/

Execution plan formats

- Execution plan is a hierarchical tree of operators
- Root operator is the final operator
- Can be visualized in three formats:
  - Graphical
  - XML
  - Text

Display and Save Execution Plans
https://docs.microsoft.com/en-us/sql/relational-databases/performance/display-and-save-execution-plans

Capturing execution plans

- Graphical plan:
  - From SSMS
  - Can be stored as and retrieved from XML
  - Live Query Statistics shows the plan in operation
- XML:
  - SHOWPLAN_XML: estimated plan
  - STATISTICS XML: actual plan
- Text:
  - SHOWPLAN_TEXT: estimated plan
  - SHOWPLAN_ALL: estimated plan with statistics
  - STATISTICS PROFILE: actual plan

Demo Show execution plan and Live query statistics

Lesson: Analyzing execution plans

- Execution plan operators
- Data retrieval operators: scan and seek
- Join operators
- Parallel operators
- Warnings in execution plans

Execution plan operators

- Query plans are made up of one or more logical operators
- All operators have an output; some also support one or two inputs
- Most operators have their own icon in graphical query plans

Showplan Logical and Physical Operators Reference
https://docs.microsoft.com/en-us/sql/relational-databases/showplan-logical-and-physical-operators-reference

SQL Server Execution Plan Operators – Part 1
https://www.sqlshack.com/sql-server-execution-plan-operators-part-1/

SQL Server Execution Plan Reference
https://sqlserverfast.com/epr/

Operator List
https://sqlserverfast.com/epr/operator-list/

SQL Server Execution Plans, 3rd Edition, Free eBook
https://www.red-gate.com/products/dba/sql-monitor/entrypage/execution-plans

Data retrieval operators: scan and seek

- Scan represents the read of a whole table:
  - Can be expensive if the table is large
- Seek represents rows from a table with reference to an index:
  - A scan may be cheaper if many rows are to be returned
  - Can only be used where a suitable index is available

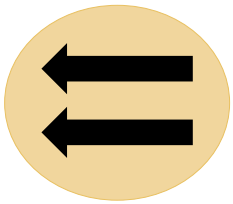
Demo Operators

Join operators

- Nested Loops:
  - The second input is searched once for each value in the first input
  - The second input should be inexpensive to search
- Merge Join:
  - Two sorted inputs are interleaved
  - Both inputs must be sorted
- Hash Match:
  - A hash table built from the first input is compared against hash values from the second input
  - Large unsorted inputs

Parallel operators

- Plans for parallelized queries do not show the activity of individual parallel workers
- Query plan operators that use parallelism are indicated in the query plan
- Parallel plans will have at least one instance of the Gather Streams operator, which combines the results of parallel operators



Warnings in execution plans

- Query plan operators may have associated warnings
- Warnings indicate issues that may have significant negative impact on query performance
- Warnings are serious and should be investigated and addressed

Lesson: Intelligent Query Processing

- About Intelligent Query Processing
- Intelligent Query Processing overview
- Memory Grant Feedback
- Adaptive Joins
- Interleaved Execution
- Other features of Intelligent Query Processing, introduced in SQL Server 2019

Intelligent Query Processing (IQP) i SQL Server är en samling funktioner designade för att förbättra prestandan av databasfrågor genom att automatiskt optimera exekveringen utan att ändra den underliggande SQL-koden. Denna teknik är en del av SQL Server och Azure SQL Databases bredare strategi för att förenkla prestandaförvaltningen och maximera effektiviteten av databasoperationer.

IQP-funktionerna introducerades gradvis från SQL Server 2017 och utökas med varje ny version. Dessa funktioner omfattar ett brett spektrum av optimeringar, som adaptiv frågebearbetning, där SQL Server dynamiskt justerar frågeplaner baserat på den faktiska exekveringsstatistiken; batch mode på radstore-index, som tillåter snabbare bearbetning av analytiska frågor även på icke-kolumnlagrade index; och skalär UDF-inlining, som automatiskt omvandlar vissa skalära användardefinierade funktioner till relationsoperationer för bättre prestanda.

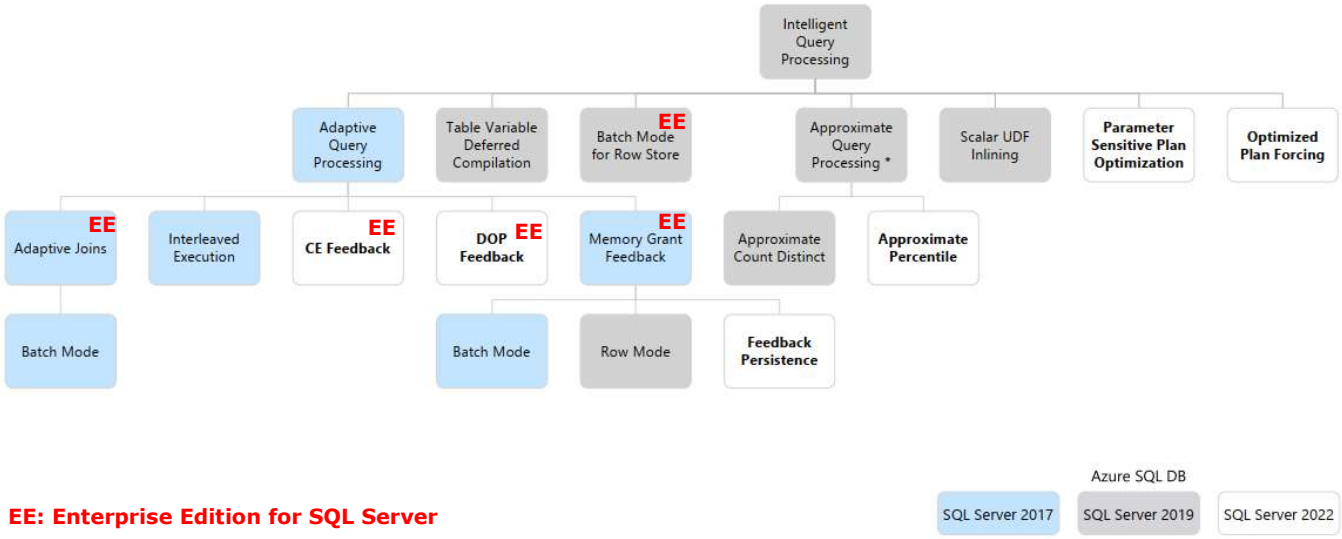
Målet med IQP är att minska behovet av manuell ingripande från databasadministratörer för frågeoptimering och att göra SQL Server smartare om hur den hanterar och exekverar frågor. Detta leder till snabbare responstider och mer effektiv resursanvändning, vilket är avgörande för stora datamängder och komplexa databasoperationer. Genom att dra nytta av dessa intelligenta bearbetningsfunktioner kan organisationer förbättra sina databassystems responsivitet och skalbarhet utan att behöva djupdyka i de tekniska detaljerna för frågeoptimering.

About Intelligent Query Processing

- The terms was introduced in SQL Server 2019
- Builds on ”Adaptive Query Processing”
  - Which was introduced in 2017
- A set of new optimizer features
- Requires database compatibility level
  - that matches the version when the feature was released
- You can disable most features
  - Using database scoped configuration

Intelligent Query Processing overview

- Requires the right database compatibility level
- Most can be turned off using database scoped config options



EE: Enterprise Edition for SQL Server

Azure SQL DB | SQL Server 2017 | SQL Server 2019 | SQL Server 2022

Intelligent query processing in SQL databases
https://docs.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing

Intelligent Query Processing Q&A
https://techcommunity.microsoft.com/t5/azure-sql-blog/intelligent-query-processing-q-amp-a/ba-p/446657

Azure SQL & SQL Server 2022: Intelligent Database Futures
https://techcommunity.microsoft.com/t5/video-hub/azure-sql-amp-sql-server-2022-intelligent-database-futures-data/ba-p/3039415

Memory grant feedback

- Memory grant is stored in the query execution plan
- Inaccurate memory grants can cause problems
  - Too large allocation means other queries might have to wait for memory grants
    - Wait type RESOURCE_SEMAPHORE
  - Too small means spill to tempdb
    - Performance suffers badly
- Requires Enterprise Edition
- Memory grant feedback revises the ideal memory grant when:
  - Insufficient memory has been allocated
  - Excessive memory has been allocated
- XE events available
- Requires batch mode in 2017
  - Available for row store as of 2019

Batch mode memory grant feedback
https://docs.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing#batch-mode-memory-grant-feedback

Row mode memory grant feedback
https://docs.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing#row-mode-memory-grant-feedback

Memory Grant Feedback är ytterligare en nyckelfunktion i Intelligent Query Processing-sviten i SQL Server, som syftar till att optimera minnesanvändningen för frågor. Traditionellt, när SQL Server exekverar en fråga, allokerar den en uppskattad mängd minne baserat på frågeplanens behov. Om denna uppskattning är felaktig – antingen för liten eller för stor – kan det leda till minnesrelaterade prestandaproblem. Memory Grant Feedback adresserar detta genom att automatiskt justera minnestilldelningen för efterföljande körningar av samma fråga, baserat på den faktiska minnesanvändningen observerad under tidigare exekveringar. Om en fråga använder mindre minne än tilldelat justerar funktionen ner minnestilldelningen, och vice versa, vilket leder till effektivare minnesanvändning och förbättrad systemprestanda över tid.

Memory grant feedback, continued

- Enhancements in 2022:
  - Persistence in Query Store
    - Survives cache evictions (fail-over, restart, severe memory pressure, …)
  - Percentiles
    - For "ping-ping" situations
    - Keep a history of prior grants instead of just disabling feedback
- Both above requires db compat 140

Denna bild handlar om förbättringar i **Memory Grant Feedback** i SQL Server 2022. Memory Grant Feedback hjälper SQL Server att justera minnesallokering för frågor baserat på tidigare exekveringar, vilket förbättrar prestanda.

Förbättringar 2022:

1.Persistence in Query Store

1. Tidigare försvann justeringar vid en cache-utkastning (t.ex. vid failover eller serveromstart).
2. Nu lagras dessa justeringar **permanent i Query Store**, vilket gör att systemet kan **lära sig över tid**.

2.Percentiles för "ping-pong" problem

1. Om en fråga växlar mellan **för mycket** och **för lite minne**, hanteras det nu bättre genom att **spara en historik** över tidigare grants istället för att inaktivera feedback.

Krav:

•Båda förbättringarna kräver att databasen har **Compatibility Level 140** eller högre.

Sammanfattningsvis gör dessa ändringar att SQL Server **blir bättre på att hantera minnesanvändning över tid**, särskilt vid fluktuationer i resursbehov. 🚀

Batch mode memory grant feedback
https://docs.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing#batch-mode-memory-grant-feedback

Row mode memory grant feedback
https://docs.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing#row-mode-memory-grant-feedback

Adaptive joins

- Requires Enterprise Edition
- Dynamically select a hash join or nested loop join after the first input has been scanned
- The join operator is determined by the actual number of rows, and not cardinality estimate
- Requires batch mode in 2017
  - Available for row store as of 2019

Adaptive Joins – Förklaring
Adaptive Joins är en **intelligent optimeringsfunktion i SQL Server** som gör att databasmotorn **väljer bästa join-strategi vid körning** istället för att förlita sig enbart på uppskattad kardinalitet.
Hur fungerar det?
•Adaptive Joins kan **växla mellan Hash Join och Nested Loop Join beroende på hur många rader som faktiskt returneras** från den första tabellen i joinen.
•Istället för att lita på **statistiska uppskattningar** väljer SQL Server dynamiskt den bästa metoden vid exekvering.
Krav:
•Kräver **Enterprise Edition**.
•Introducerades i **SQL Server 2017** men krävde **Batch Mode**, vilket då endast fanns för Columnstore Index.
•Från **SQL Server 2019** fungerar det även på **vanliga rowstore-tabeller**.
Fördelar:
✅ Minskar risken för suboptimal join-strategi.
✅ Förbättrar prestanda vid varierande dataset.
✅ Speciellt användbart när **datavolymer förändras över tid**.
En **game-changer** för optimering av dynamiska SQL-frågor! 🚀

Batch mode Adaptive joins
https://docs.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing#batch-mode-adaptive-joins

Adaptive Join

https://sqlserverfast.com/epr/adaptive-join/

Interleaved Execution

- Multi-statement table-valued functions use a guessed cardinality
  - SQL Server 2014 and 2016 – 100 rows
  - Earlier versions - 1 row
- Interleaved execution uses the actual cardinality estimate to process the rest of the query
- The query
  - Must not modify data
  - Not be referenced inside a CROSS APPLY clause
- Do **not** require Enterprise Edition

Interleaved execution for MSTVFs
https://docs.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing#interleaved-execution-for-mstvfs

Interleaved Execution är en funktion inom Intelligent Query Processing (IQP) i SQL Server som ändrar hur frågeplaner hanteras när det gäller multi-statement table-valued functions (MSTVF). Traditionellt, när en frågeplan genereras, används statiska uppskattningar för att bedöma antalet rader som en operation kommer att returnera. Dessa uppskattningar kan vara mycket oexakta för MSTVF, vilket leder till suboptimala frågeplaner. Interleaved Execution löser detta genom att faktiskt pausa frågeplanens generering efter att ha identifierat en MSTVF, köra den relevanta delen av frågan för att få en exakt raduppskattning, och sedan fortsätta med frågeplaneringen med dessa uppdaterade statistikdata. Detta resulterar i mer precisa frågeplaner och kan leda till betydande prestandaförbättringar för berörda frågor.

Other features of Intelligent Query Processing, introduced in SQL Server 2019 1(2)

- Table variable deferred compilation
  - Without this, 1 row is estimated
  - With this, the actual number of rows are known
- Scalar UDF inlining
  - Inline a scalar UDF
  - UDFs are about the worst you can do for performance!
  - Inlining might reduce overhead drastically
    - Or the other way – you can disable this!
- Approximate query processing
  - The new APPROX_COUNT_DISTINCT() aggregate function
  - Requires less memory -> less likely to spill to disk

Table variable deferred compilation
https://docs.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing#table-variable-deferred-compilation

Scalar UDF inlining
https://docs.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing#scalar-udf-inlining

Approximate query processing
https://docs.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing#approximate-query-processing

Scalar UDF (User-Defined Function) Inlining är en funktion i SQL Server som ingår i Intelligent Query Processing (IQP)-paketet. Den transformerar skalära användardefinierade funktioner till relationella uttryck, vilket gör det möjligt för SQL Server att inkludera dessa funktioner direkt i den övergripande frågeplanen istället för att exekvera dem som separata enheter för varje rad. Detta eliminerar den betydande overhead som traditionellt är associerad med skalära UDF:er, där funktionen måste kallas separat för varje rad i resultatet, vilket leder till långsam prestanda vid stora dataset.

Funktionen APPROX_COUNT_DISTINCT() i SQL Server är en prestandaförbättrande funktion som ger en ungefärlig unik värderäkning av en kolumn. Till skillnad från den traditionella COUNT(DISTINCT column_name)-funktionen, som exakt räknar antalet unika värden och kan vara resurskrävande på stora datamängder, använder APPROX_COUNT_DISTINCT() en algoritm som beräknar ett närmevärde av antalet unika värden snabbare och med lägre

minnesanvändning.

Other features of Intelligent Query Processing, introduced in SQL Server 2019 2(2)

- Batch mode over rowstore
  - Batchmode without having a Columnstore index involved
  - Requires Enterprise Edition

Lab 7: Query plan analysis

- Ex 1: Improve performance of the GetOrderDetailsReseller stored procedure
- Ex 2: Improve performance of the GetOrderDetailsDueDate stored procedure

Estimated Time: 30 minutes