SQL Server Modern Query Processing

Gianluca Hotz

Presidente ugiss.org

PASS GLOBAL ITALIAN VIRTUAL GROUP

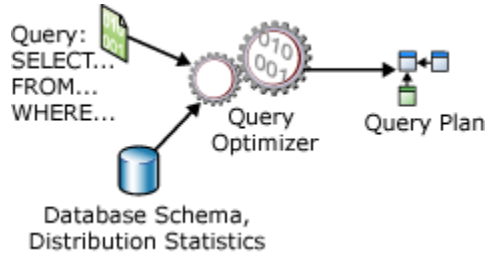
Chi sono?

- Gianluca Hotz | @glhotz | ghotz@ugiss.org
- Consulente indipendente
  - 20+ anni su SQL Server (dalla 4.21 nel 1996)
  - Modellazione e sviluppo database, dimensionamento e amministrazione database server, aggiornamenti e migrazioni, performance tuning
- Community
  - 20+ anni Microsoft MVP SQL Server/Data Platform (dal 1998)
  - VMware Experts SQL Server
  - Fondatore e presidente UGISS (PASS Chapter)
  - Co-organizzatore DAMAG Meetup Community

Agenda

- Introduzione
- Cardinality Estimator (2014+)
- Auto Tuning (2017+)
- Intelligent Query Processing (2017+, 2019+)

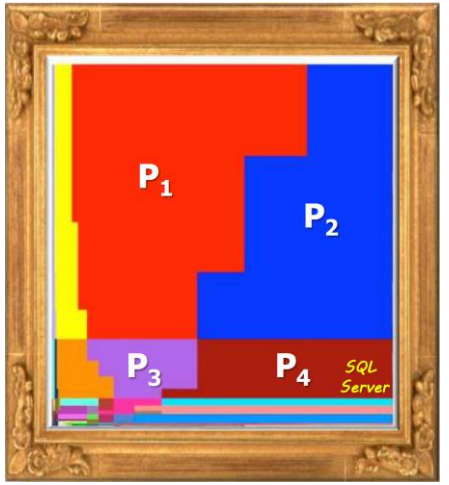
«Query Optimizer»



- SQL è un linguaggio dichiarativo
  - Specifichiamo cosa vogliamo ottenere, non come
- Il QO genera un piano di esecuzione
  - Il come ottenere le cose richieste
  - Tipicamente cercandolo tra molteplici piani possibili
  - Sfruttando modello basato su costi e statistiche distribuzione valori
  - https://docs.microsoft.com/en-gb/sql/relational-databases/query-processing-architecture-guide

Un po’ di storia

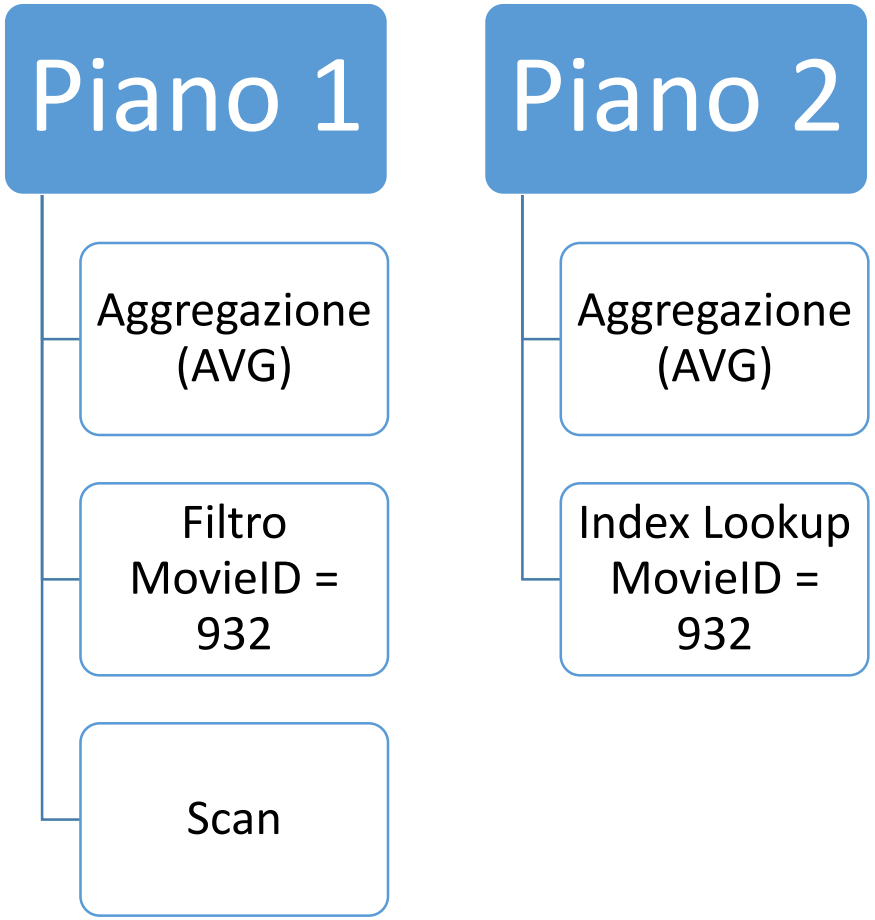
- Ottimizzazione «cost-based» risale 1970
  - Pat Selinger in System R (poi divenuto DB2)
- 50 anni, RDBMS ancora software più difficile da implementare
  - Paura di causare regressioni
- Obiettivo QO non miglior piano in assoluto…
  - Spazio delle soluzioni troppo grande per essere esplorato
  - Miglior piano in un tempo ragionevole (approccio euristico)
  - Sceglie ancora troppi piani inefficienti ☹



Esempio

- SELECT AVG(Rating)
  FROM Reviews
  WHERE MovieID = 932
- Calcolare media punteggi recensioni per film con ID = 932

Esempio

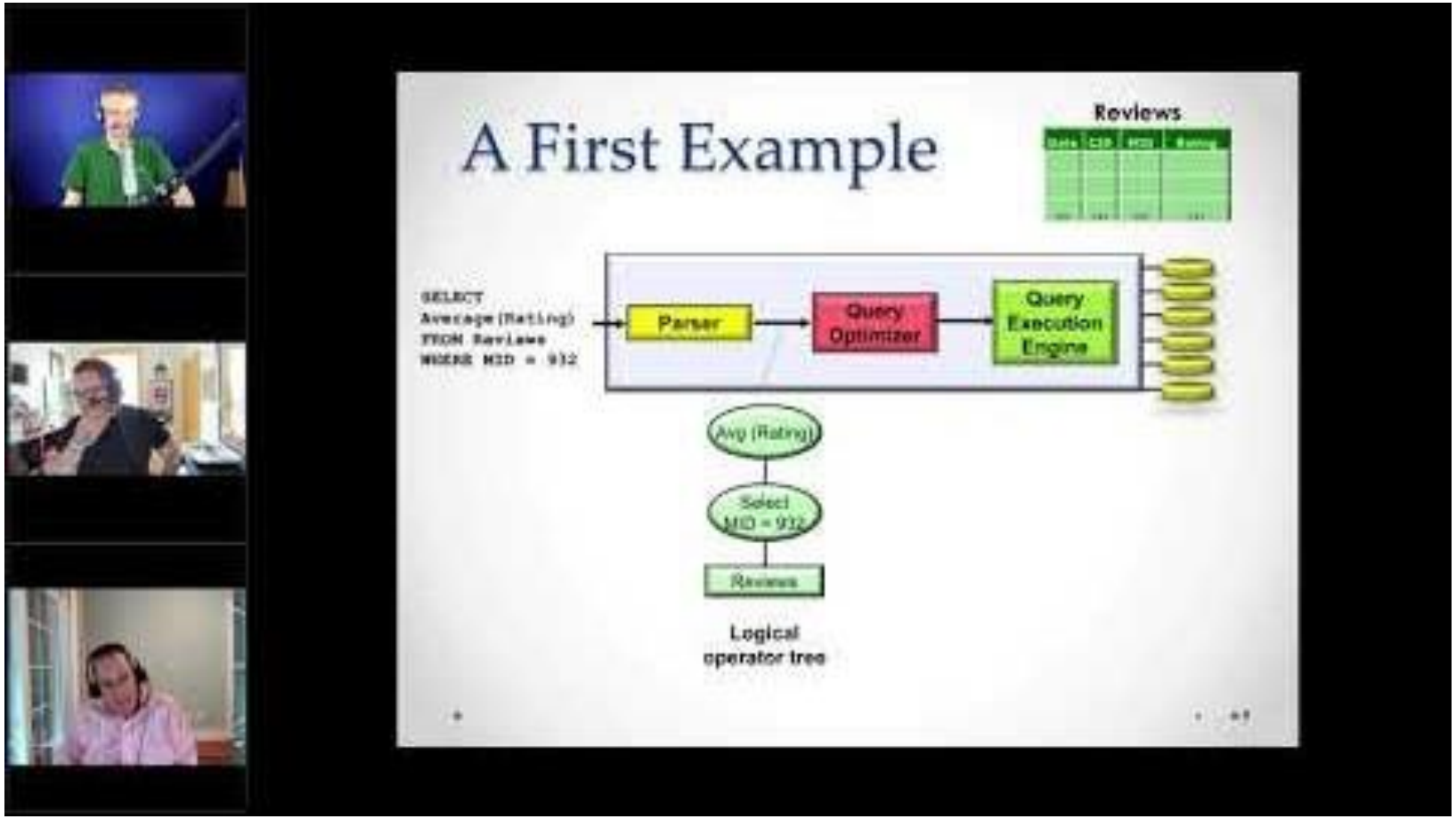

Piano 1
Aggregazione (AVG)
Filtro MovieID = 932
Scan
Piano 2
Aggregazione (AVG)
Index Lookup MovieID = 932

Esempio più complesso

- TPC_H Query 8
  - National Market Share
- Circa 22 milioni di piani possibili!

```
Select o_year,
sum(case
 when nation = 'BRAZIL' then volume
 else 0
end) / sum(volume)
from
(
 select YEAR(O_ORDERDATE) as o_year,
  L_EXTENDEDPRICE * (1 - L_DISCOUNT) as volume,
  n2.N_NAME as nation
 from PART, SUPPLIER, LINEITEM, ORDERS, CUSTOMER, NATION
n1,
 NATION n2, REGION
 where
    P_PARTKEY = L_PARTKEY and S_SUPPKEY = L_SUPPKEY
    and L_ORDERKEY = O_ORDERKEY and O_CUSTKEY = C_CUSTKEY
    and C_NATIONKEY = n1.N_NATIONKEY and n1.N_REGIONKEY =
R_REGIONKEY
    and R_NAME = 'AMERICA‘ and S_NATIONKEY = n2.N_NATIONKEY
    and O_ORDERDATE between '1995-01-01' and '1996-12-31'
    and P_TYPE = 'ECONOMY ANODIZED STEEL'
    and S_ACCTBAL <= constant-1
    and L_EXTENDEDPRICE <= constant-2
) as all_nations
group by o_year  order by o_year
```

SQL Query Optimization: Why Is It So Hard to Get Right?



Introduzione al «Cardinality Estimator»

- Essenziale per generare piani esecuzione, predice
  - Numero righe risultati intermedi (es. join, filtri, aggregazioni)
  - Numero righe finale
- Impatto diretto su iteratori (es. tipo e ordine join)
- In gran parte basato su SQL Server 7.0
  - Abilitazione Fix in QFE tramite Trace Flag per evitare bug di regressione
  - Alcuni problemi necessitavano di un ridisegno rilevante
- SQL Server 2014+ CE può causare regressioni
  - La maggior parte delle query dovrebbe comunque beneficiarne

Comportamento nuovo CE

- Dati ascendenti
  - Es. filtro su «Data Ordine» con statistiche non aggiornate
  - CE 70 assume che nessun nuovo valore esista
  - CE 120+ usa cardinalità media di ogni valore della colonna
- Predicati filtro su colonne stessa tabella
  - Ora assume un certo grado di correlazione
  - Componente esponenziale aggiunto all'equazione
- Predicati filtro su colonne di tabelle diverse
  - «Join Containment»
  - Ora assume maggiore indipendenza

Configurazione nuovo CE

- SQL Server 2014
  - Nessuna opzione indipendente
  - Impostare livello di compatibilità database a 120
- SQL Server 2016+
  - Opzione «database scoped configuration»
    - LEGACY_CARDINALITY_ESTIMATION
  - Indipendente dal livello di compatibilità del database
  - Eventuale impostazione indipendente repliche secondarie AlwaysOn AG
  - Nuovo CE default quando si effettua migrazione

Demo

- Nuovo «Cardinality Estimator»

Troubleshooting nuovo CE

- Trace Flag
  - Trace Flag **2312** usa la versione 70 del CE
  - Trace Flag **9481** usa la versione 120 del CE
  - Globale, sessione o singola query
- USE HINT (SQL Server 2016 SP1+)
  - **`FORCE_DEFAULT_CARDINALITY_ESTIMATION`**
  - **`FORCE_LEGACY_CARDINALITY_ESTIMATION`**
- Extended Events
  - **`query_optimizer_estimate_cardinality`**
  - **`query_optimizer_force_both_cardinality_estimation_behaviors`**
- **CardinalityEstimationModelVersion** nei piani di esecuzione in XML

Risorse nuovo CE

- Articoli
  - Miloš Radivojević Blog Series
    - http://milossql.wordpress.com/tag/cardinality-estimator
  - A First Look at the New SQL Server Cardinality Estimator
    - http://www.sqlperformance.com/2013/12/t-sql-queries/a-first-look-at-the-new-sql-server-cardinality-estimator
  - New functionality in SQL Server 2014 – Part 2 – New Cardinality Estimation
    - http://blogs.msdn.com/b/saponsqlserver/archive/2014/01/16/new-functionality-in-sql-server-2014-part-2-new-cardinality-estimation.aspx
  - Join Containment Assumption and CE Model Variation in SQL Server
    - https://www.sqlshack.com/join-containment-assumption-and-ce-model-variation

«Tuning» automatico

- Correzione dei piani automatica
  - Forza ultimo piano ottimale quando identifica regressione
  - Disponibile in SQL Server 2017+ EE e Azure SQL Database
- Gestione automatica indici
  - Creazione/eliminazione indici
  - Verifica effettivo utilizzo e miglioramento (modello ML)
  - Disponibile solo in Azure SQL Database

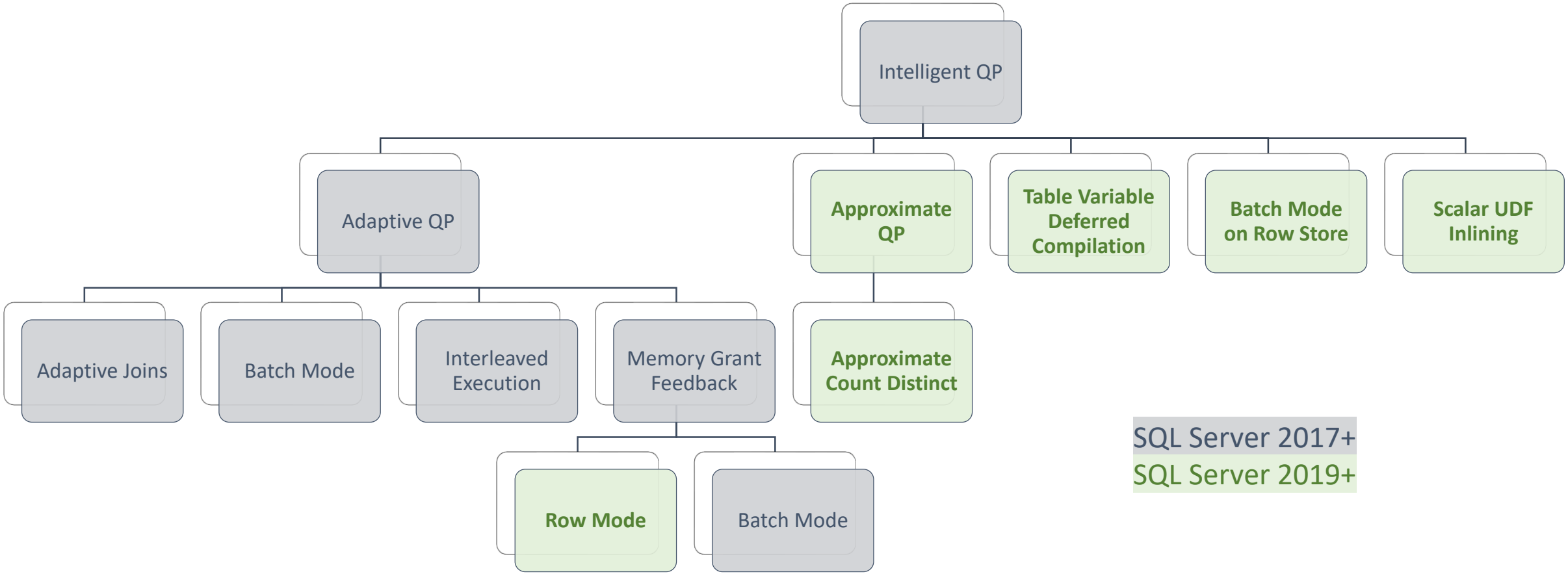
Correzione automatica piani

- Forza ultimo piano ottimale
  - quando identifica regressione
  - contro storico Query Store
  - non sopravvive al ravvio
- Miglioramento minimo
  - di 10 secondi «CPU time»
- Solo Enterprise Edition
- Correzione manuale
  - 2016+ report Query Store
  - 2017+ nuova DMV
    - **sys.dm_db_tuning_recommendations**

Demo

- Correzione dei piani automatica

Intelligent Query Processing



SQL Server 2017+

SQL Server 2019+

«Execution Modes»

- «Row Mode»
  - Iteratori albero piano esecuzione consumano 1 riga alla volta
  - Modalità esecuzione tradizionale con «Rowstore»
- «Batch Mode»
  - Iteratori albero piano esecuzione consumano batch di righe alla volta (1000)
  - Ottimale per operazioni di «scan» (es. aggregati o join tra tabelle grosse)
  - SQL Server 2012: introdotto per sfruttare al meglio indici Columnstore
  - SQL Server 2016/2017: estesi scenari di utilizzo con indici Columnstore
  - SQL Server 2019: estesi scenari di utilizzo con «Rowstore»

«Batch Mode» con «Rowstore»

- Aiuta a ridurre consumo di CPU
- Indice colonnare rimane scelta migliore
  - Per carichi di lavoro OLAP con molto I/O
  - Quando non si può creare (es. impatto su OLTP, funzionalità non supportata)
- Limiti
  - Tabelle «In-Memory» non supportate (solo «Heap» e «B-Tree» su disco)
  - Non usato durante lettura/filtraggio colonne LOB
    - Inclusi «sparse columns sets» e XML

«Batch Mode» con «Rowstore»

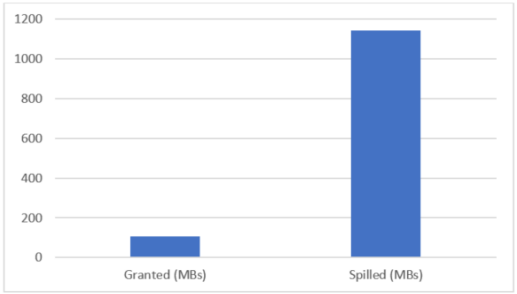
- SQL Server < 2019
  - Alcuni scenari possibili tramite «trucchi» (articoli part1, part2, part3)
- SQL Server 2019+
  - Solo scenari supportati direttamente da «Query Processor»
  - Attivo di default con livello compatibilità database **150+**
  - `ALTER DATABASE SCOPED CONFIGURATION SET BATCH_MODE_ON_ROWSTORE = ON|OFF`
  - `OPTION (USE HINT ('ALLOW_BATCH_MODE'));`
  - `OPTION (USE HINT ('DISALLOW_BATCH_MODE'));`

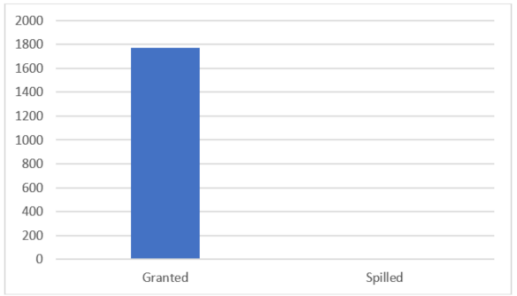
Demo

- «Batch Mode» con «Rowstore»

Memory Grant Feedback

- Valutazione post-esecuzione
  - Aggiorna memoria «granted» per i piani in cache
  - Es. più memoria se «spill», meno se «grant» eccesivo
- Versioni supportate
  - SQL Server 2017+ «Batch Mode»
  - SQL Server 2019+ «Row Mode»
- «Plan caching»
  - Non persistente (non salvato nel Query Store)
  - OPTION(RECOMPILE) previene «caching» e «memory grant feedback»





Controllo «Memory Grant Feedback»

- «Batch Mode»
  - Attivo di default con livello compatibilità database **140+**
  - `ALTER DATABASE SCOPED CONFIGURATION SET BATCH_MODE_MEMORY_GRANT_FEEDBACK = ON|OFF`
  - `OPTION (USE HINT('DISABLE_BATCH_MODE_MEMORY_GRANT_FEEDBACK'));`
- «Row Mode»
  - Attivo di default con livello compatibilità database **150+**
  - ALTER DATABASE SCOPED CONFIGURATION SET **ROW_MODE_MEMORY_GRANT_FEEDBACK** = ON|OFF
  - OPTION (USE HINT ('**DISABLE_ROW_MODE_MEMORY_GRANT_FEEDBACK**'));

Demo

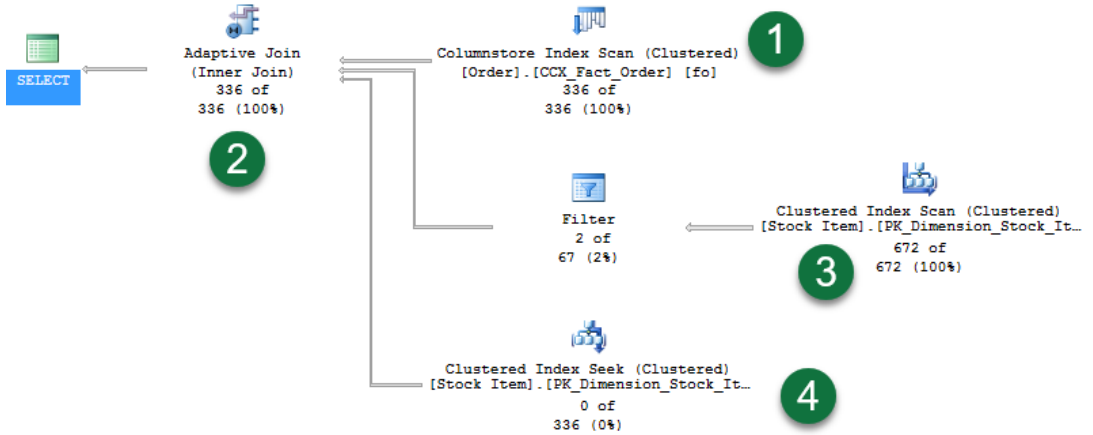
- «Memory Grant Feedback»

Troubleshooting «Memory Grant Feedback»

- Scenari query sensibili a parametri
  - Alcune query richiedono «grant» diversi in base a parametri
  - «Memory Grant Feedback» si disabilita se rileva instabilità
- Extended Events per controllare comportamento
  - SQL Server 2017+ **`memory_grant_feedback_loop_disabled`**
  - SQL Server 2019+ **`memory_grant_updated_by_feedback`**
- SQL Server 2019+ proprietà del piano esecuzione
  - **IsMemoryGrantFeedbackAdjusted**
    - No: First Execution, Accurate Grant, Feedback disabled
    - Yes: Adjusting, Stable
  - **LastRequestedMemory**

«Batch Mode Adaptive Joins»

- «Nested Loop Join» efficiente
  - tabelle input molto piccole
  - una tabella input piccola + indice per «seek» su tabella grande
- «Hash Join» efficiente
  - tabelle di input molto voluminose
  - potenzialmente molta memoria
- «Adaptive Joins»
  - rinvia la scelta dopo scansione primo input



Demo

- «Batch Mode Adaptive Joins»

«Interleaved Execution»

- Problema con funzioni che ritornano tabelle
  - «multi-statement table valued functions» (MSTVF)
  - SQL Server <= 2012 QO assume cardinalità = 1
  - SQL Server 2014/2016 QO assuma cardinalità = 100
- SQL Server >= 2017
  - Inizia ottimizzazione
  - Esegue MSTVF se candidata per «Interleaved Execution»
  - Riprende ottimizzazione con la cardinalità corretta

Demo

- «Interleaved Execution»

Variabili Tabella e Tabelle Temporanee

Area	Tabelle temporanee	Variabili Tabella
Creazione e aggiornamento manuale statistiche	Si	No
Indici	Si	Solo definizioni «in-line»
Vincoli	Si	Solo PRIMARY KEY, UNIQUE e CHECK
Creazione statistiche automatica	Si	No
Creazione e uso di oggetti temporanei in un singolo batch	Compilazione di un comando che referenzia una tabella temporanea che non esiste è differito fino al momento delle prima esecuzione del comando	Un comando che referenzia una variabile tabella è compilato insieme a tutti gli altri comandi prima che uno di questi la popoli, il QO assume cardinalità uguale a 1

Compilazione differita variabili tabella

- Compilazione comando che referenzia variabile tabella
  - SQL Server < 2019 compilato prima che sia popolata con cardinalità fissa = 1
  - SQL Server 2019+ differito a prima esecuzione (come tabelle temporanee)

Demo

- Compilazione differita variabili tabella

Controllo compilazione differita VT

- Attivo di default con livello compatibilità database **150+**
- `ALTER DATABASE SCOPED CONFIGURATION SET` **`DEFERRED_COMPILATION_TV`** `= ON|OFF`
- `OPTION (USE HINT (`**`'DISABLE_DEFERRED_COMPILATION_TV'`**`));`

«Scalar UDF In-lining»

- Funzioni utente T-SQL (UDF) che ritornano un singolo valore scalare
- Problemi di prestazioni
  - Invocazione iterativa linguaggio interpretato
    - Per ogni riga, «context switching» molto oneroso durante esecuzione query
    - Comandi eseguiti in isolamento, no ottimizzazioni «cross-statement
  - Esecuzione seriale
    - Parallelismo «Intra-query» inibito
  - Mancanza di costi
    - Storicamente attribuiti solo a operatori relazioni, assunzione costo UDF basso…

«Scalar UDF Automatic In-lining»

- SQL Server 2019+ UDF scalari automaticamente convertite in
  - Espressioni scalari
  - «Subquery» scalari
- Ottimizzazione piano completa (UDF non più visibili)

IQP: esempio «Scalar UDF in-lining»

```
CREATE FUNCTION dbo.discount_price(@price DECIMAL(12,2), @discount DECIMAL(12,2))
RETURNS DECIMAL (12,2) AS BEGIN RETURN @price * (1 - @discount); END

SELECT L_SHIPDATE, O_SHIPPRIORITY
, SUM(dbo.discount_price(L_EXTENDEDPRICE, L_DISCOUNT))
FROM LINEITEM, ORDERS
WHERE O_ORDERKEY = L_ORDERKEY
GROUP BY L_SHIPDATE, O_SHIPPRIORITY
ORDER BY L_SHIPDATE
```

10GB CCI compressed TPC-H Schema, 2 x CPUs (12 cores), 96GB RAM, SSD storage

	Query senza UDF	Query con UDF (senza in-lining)	Query con UDF (e in-lining)
Tempo esecuzione	1.6 secondi	29 minuti 11 secondi	1.6 secondi

Demo

- «Scalar UDF Automatic In-lining»

Controllo «Scalar UDF Automatic Inlining»

- Attivo di default con livello compatibilità database **150+**
- `ALTER DATABASE SCOPED CONFIGURATION SET **TSQL_SCALAR_UDF_INLINING** = ON|OFF`
- `OPTION (USE HINT ('**DISABLE_TSQL_SCALAR_UDF_INLINING**'));`
- `CREATE FUNCTION … WITH **INLINE** = ON|OFF`

Requisiti «Scalar UDF in-lining»

- Può usare uno dei seguenti costrutti
  - DECLARE, SET
    dichiarazione/assegnazione variabili)
  - SELECT
    assegnazione variabili singole/multiple
  - IF/ELSE con livelli annidamento arbitrari
  - RETURN singoli o multipli
  - Chiamate a UDF annidate/ricorsive
  - Operazioni relazionali es. EXISTS, ISNULL
- Nessuna invocazione di funzioni con
  - dipendenza dal tempo es. GETDATE()
  - effetti collaterali es. NEWSEQUENTIALID()
- Utilizzo di EXECUTE AS CALLER (default)
- No riferimenti a
  - variabili tabella
  - parametri «table-valued»
  - tipi dato «user-defined»
- Non compilate nativamente
  - Supporto a «interop»
- No funzione di partizionamento
- Non referenziate da
  - clausole GROUP BY
  - colonne calcolate
  - vincoli CHECK
- No signatures added to it

Troubleshooting «Scalar UDF in-lining»

- Colonna **`is_inlineable`** in **`sys.sql_modules`**
  - Non implica che sarà sempre «in-lined»! (es. 1000 righe di codice)
- Piano di esecuzione
  - Se «in-lined»! , nodo XML **<UserDefinedFunction>** mancante
- Extended Events
  - **`tsql_scalar_udf_not_inlineable`**

APPROX_COUNT_DISTINCT

- Ritorna il numero **approssimato** di valori univoci non nulli per gruppo
  - Algoritmo «HyperLogLog» garantisce tasso errore <= 2% nel 97% dei casi
- Risultati rapidi con consumo di memoria ridotto
  - Es. dashboards, analisi di trend, «feature selection», ecc.
  - Pensato per grossi volumi di dati e richieste
    - es. 10 miliardi di righe, 1 utente che consuma 1,5GB di memoria per risultato preciso contro 100 utenti in contemporanea che usano 12MB per un risultato approssimato…
- Compromesso
  - precisione, quindi solo scenari dove i valori precisi non sono necessari!

Demo

- APPROX_COUNT_DISTINCT

Disponibilità per edizione

Standard, Web, Express

- New Cardinality Estimator
- Interleaved Execution
- Table variable deferred compilation
- Scalar UDF In-lining
- Approximate Distinct Count

Enterprise, Developer

- Automatic Tuning
- Batch Mode on Rowstore
- Memory Grant Feedback
- Adaptive Joins

GRAZIE!

SQL Server Modern Query Processing