



Best practices on SQL Server

Gianluca Hotz

@glhotz | Presidente UGISS.ORG | Data Platform MVP

- Gianluca Hotz | @glhotz | ghotz@ugiss.org

- Fondatore e Mentor SolidQ

- 20+ anni con SQL Server (dalla 4.21 nel 1996)
- Modellazione basi di dati, dimensionamento e amministrazione, sviluppo, ottimizzazione



- Interessi

- Modello relazionale, architettura DBMS, alta disponibilità e Disaster Recovery

- Community

- 20 anni Microsoft MVP SQL Server (dal 1998)
- Fondatore e presidente [UGISS](#)
 - User Group Italiano SQL Server (PASS Chapter)





Introduzione

Amministrazione

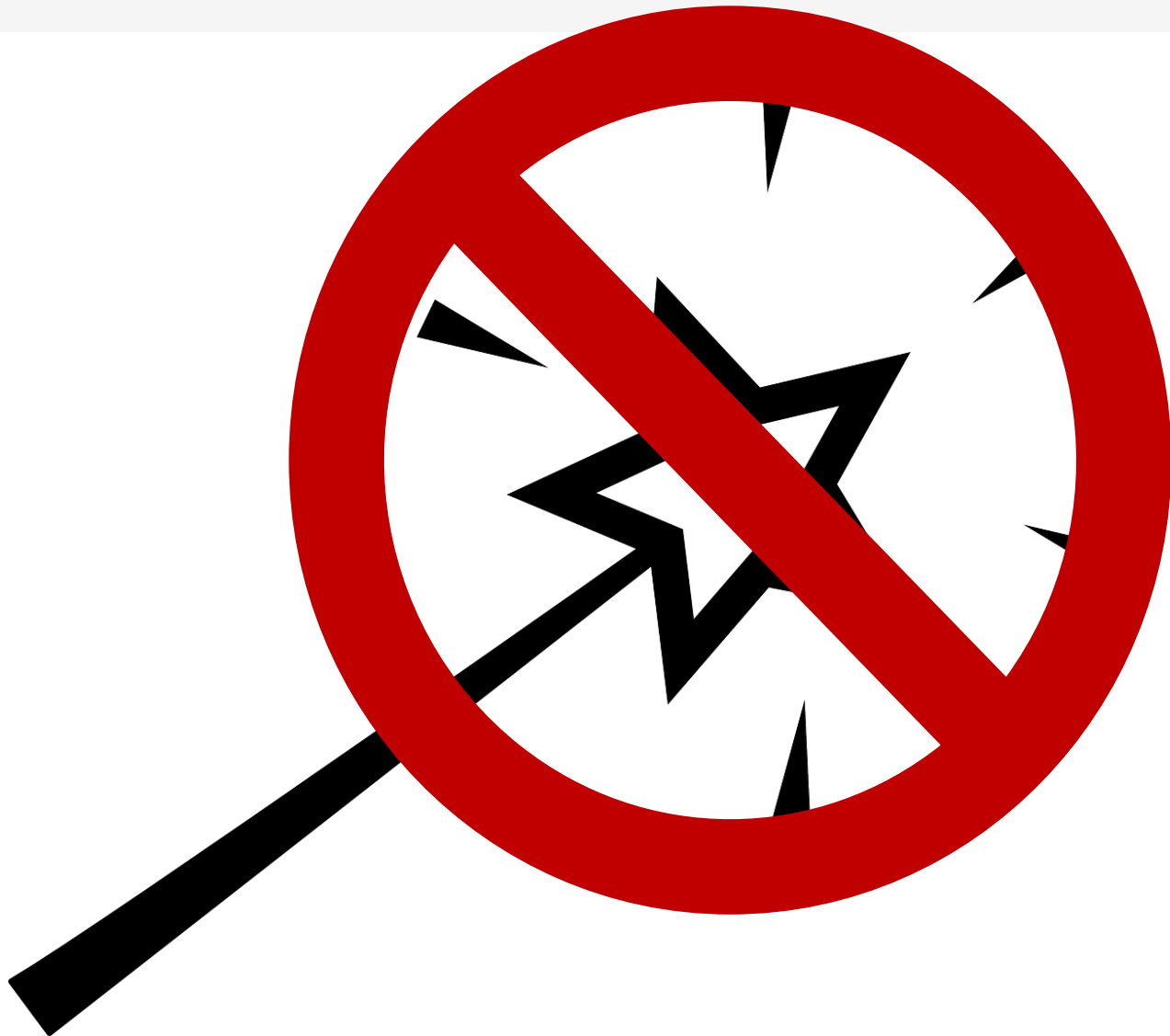
Sviluppo



Introduzione

Best practices on SQL Server

- Compito di preservare i dati e l'integrità fisica/logica
- Vita mediamente più lunga di un'applicazione
 - Nasce OLTP...
 - ... poi arrivano reporting aziendale e BI
 - ... poi iniziano progetti di integrazione tra sistemi
 - ... magari arrivano delle fusioni
- Tipicamente usato da molteplici applicazioni
 - ... le informazioni possono crescere a dismisura
 - ... il carico di lavoro può cambiare frequentemente
 - ... collo di bottiglia rischia di impattarle tutte





- Database Engine
- SQL Server Agent
- Integration Services
- Analysis Services
- Reporting Services
- Master Data Services
- Data Quality Services
- Data Mining
- Edizioni Specifiche
 - PDW/APS
- Edizioni PaaS
 - SQL Database
 - SQL DB Managed Inst.
 - RDS for SQL Server
 - SQL Data Warehouse

- Amministrazione
 - Prestazioni (dimensionamento, configurazione)
 - Disponibilità (HA/DR, backup)
 - Sicurezza (controllo accessi, permessi, crittografia)
- Sviluppo
 - Prestazioni (schema, query, indici)
 - Manutenzione futura
 - Sicurezza (SQL Injection, crittografia)
- Sono tantissime...
 - ... alcune ormai classificate come «miti»
 - ... complessità scenari/RDBMS non aiutano



- Tormentone nella community SQL Server!
- RDBMS è un software estremamente complesso!
- Serve conoscenza generica e specifica
 - architettura
 - modello relazionale e modellazione basi di dati
 - linguaggi di interrogazione
- Best Practice vanno sempre contestualizzate
 - spesso sono eccessive...
 - ... ma questo non significa che vadano ignorate!
 - ... a complicare le cose ci sono le diverse edizioni
 - ... e i diversi ambienti (on-premise, IaaS, PaaS)
 - ... e le diverse tipologie di carico di lavoro (OLTP/OLAP)



Tuttavia qualcosa di generico c'è...



- Move Forward! 😊
- Almeno a SQL Server 2016 SP1!
 - Diagnostica drasticamente migliorata
 - Molti limiti rimossi
 - Molte funzionalità EE disponibili in edizioni inferiori
 - ... ormai meglio direttamente a SQL Server 2017

Supporto Versioni



Versione SQL Server	Supporto "Mainstream"	Supporto "Extended"	Supporto SP
SQL Server 2000 SP4	08/04/2008	09/04/2013	
SQL Server 2005 SP4	12/04/2011	12/04/2016	
SQL Server 2008 SP4	08/07/2014	09/07/2019	
SQL Server 2008R2 SP3	08/07/2014	09/07/2019	
SQL Server 2012 SP3			09/10/2018
SQL Server 2012 SP4	11/07/2017	12/07/2022	
SQL Server 2014 SP2	09/07/2019	09/07/2024	
SQL Server 2016 SP1			09/07/2019
SQL Server 2016 SP2	13/07/2021	14/07/2026	
SQL Server 2017	11/10/2022	12/10/2027	SP non previsti ^[1]

[1] <https://blogs.msdn.microsoft.com/sqlreleaseservices/announcing-the-modern-servicing-model-for-sql-server>



Amministrazione

Best practices on SQL Server



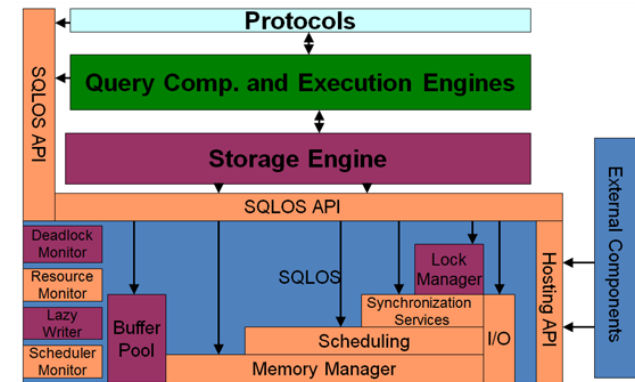


- Solitamente non abbiamo dati sufficienti
 - Storicamente realizzato simulando il sistema: «Benchmark sizing», generalmente molto costoso
- Confronto con benchmark di riferimento
 - TPC-C, TPC-H, SAP-SD ecc.
 - Tipicamente sono poco confrontabili
- Raccolta delle metriche di sistemi simili
 - Non sempre possibile



- Caratteristiche carico di lavoro diverse
 - Transazionali (OLTP)
 - Molte attività lettura/scrittura che coinvolgono pochi dati
 - Decisionali, reporting, Business Intelligence (OLAP)
 - Più attività di lettura che coinvolgono grossi volumi di dati
- Nel modo reale spesso sistemi misti
 - Es. Reporting o BI integrati nel gestionale oppure...
 - Query non performanti o mancanza di indici

- Gestisce risorse
 - Come un Sistema Operativo
 - Memoria, schedulazione task
 - Primitive sincronizzazione ecc.



- Necessaria buona conoscenza
 - Per capire bene parametri di configurazione
 - Per capire quando e come cambiare i «default»
 - <http://blogs.msdn.com/b/sqlosteam/archive/2010/06/23/sqlos-resources.aspx>

- Uso Intensivo
 - Molte connessioni in parallelo
 - Query computazionalmente onerose
 - Risoluzione singole query con forza bruta
 - Compressione (non particolarmente alto)
 - Tecnologie «In-Memory»
- In generale
 - Lasciare i parametri di configurazione di default
- Tuning configurazione per scenari specifici
 - «Max Degree of Parallelism»
 - «Cost Threshold for Parallelism
 - «Max Worker Thread»



- Affinità processori
 - Schedulazione «thread»
 - I/O affinity
- Parametro «Fiber mode»
- Parametro «Boost priority»
- In generale, non toccare!

Opzione «Max Worker Thread»



- Gruppo condiviso di «lavoratori» che eseguono Query
 - Modello concorrenza cooperativo vs. competitivo in Windows
 - Default in base a numero core/architettura CPU (32/64bit)
 - <http://msdn.microsoft.com/en-us/library/ms190219.aspx>
- Consumano memoria «MemToLeave»
 - Non controllabile*
 - x86: 512KB per worker thread
 - X64: 2MB per worker thread (4MB per IA64)



- Inter-query
 - Query distinte in esecuzione parallela
- Intra-query
 - Parallelizzazione attività di una singola query
 - Costoso in termini di inizializzazione/uso risorse



- Massimo grado parallelismo Intra-query
- Alcune regole generiche
 - Abbassare per sistemi OLTP (in teoria fino a 1)
 - Da 1 a numero core se core totali/per nodo NUMA ≤ 8
 - Fisso a 8 se core totali/per nodo NUMA > 8
 - Alcune operazioni possono beneficiare > 8
 - es. reindex ma dipende anche dal «Recovery Model»
- Controllo più granulare con «Resource Governor»
- «Fine-tuning» per singole query



- Costo minimo stimato per considerare parallelismo
 - Espresso in secondi ma...
 - ...stimato per esecuzione su sistema modello
- Regola generica
 - Alzare per sistemi OLTP o misti
 - Indicativamente partire da 20 e tenere sotto controllo



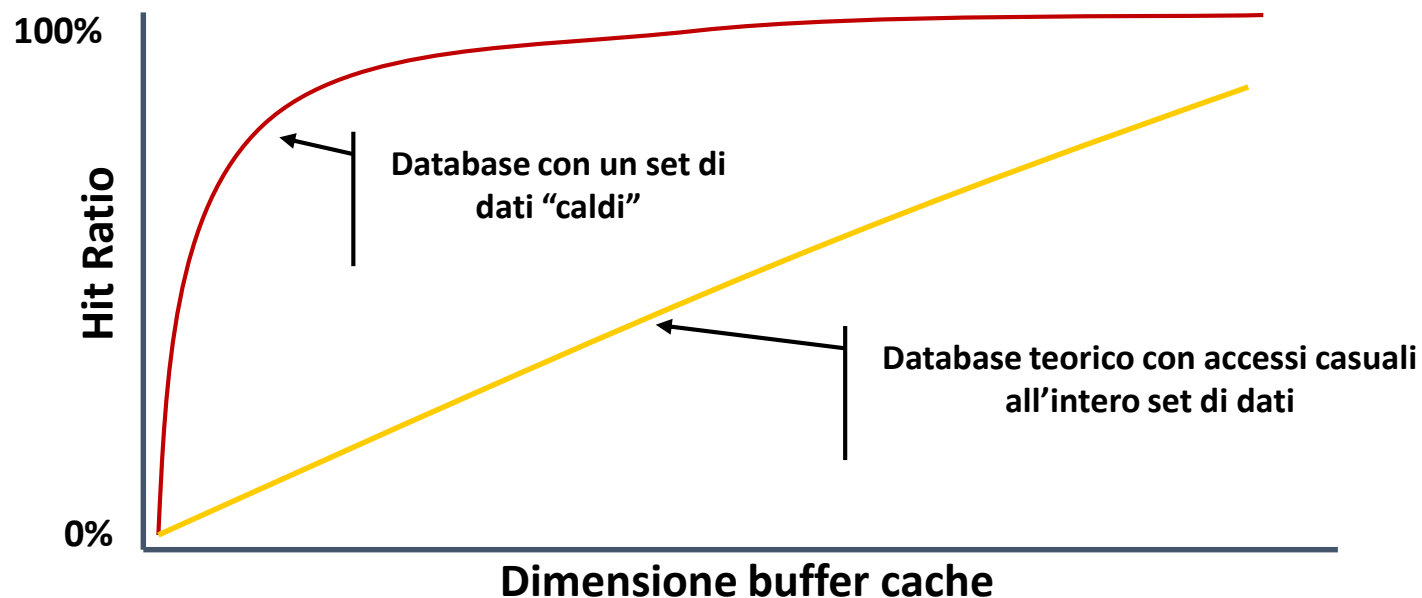
- Tendenza a consolidare più database/istanze
 - Sia per tendenza a virtualizzare in generale
 - Che per aumento costi licenze Enterprise per core!
- Valutare di sfruttare al meglio i core
 - Non lasciarli al 30-50% di utilizzo ma...
 - Attenzione ai picchi di utilizzo!
 - Attenzione a teoria delle code!
 - aumento tempi di risposta esponenziale
- Enterprise Edition
 - svariate tecnologie per sfruttare al meglio risorse

- Uso intensivo
 - Miriadi di cache (principali Buffer Pool/Procedure Cache)
 - «In-Memory OLTP» (storage tabelle «Memory-Mased»)
 - «In-Memory OLAP» (indici ColumnStore)
- In generale
 - Quanta più possibile 😊
 - Soprattutto con tecnologie «In-Memory»
 - Supporto edizione Standard (128GB/32GB/32GB)
- Best Practice: impostazione manuale
 - Parametro «Min Server Memory» solitamente no
 - Parametro «Max Server Memory» quasi sempre si...



- MemToLeave (VAS Reservation)
 - Thread stack
 - Allocazioni >8KB fino a SQL 2008 R2
 - Es. extended & COM procedures, linked servers, OLEDB providers, SQLCLR
- Buffer Pool
 - Organizzata in pagine da 8KB
 - Principalmente Buffer e Procedure Cache (ma anche altre)
 - Obiettivo primario Buffer Cache: ridurre I/O fisico
- In-Memory OLTP Storage

- Quanta più possibile, ma... dipende





- Gestione dinamica
 - Reagisce a pressione esterna (applicazioni o Windows)
 - Problemi velocità deallocazione (Windows non aspetta...)
 - Convivenza con altre applicazione non banale
 - Anche con altri componenti del prodotto: SSIS, AS, RS
- Best Practice: impostazione manuale
 - Parametro «Min Server Memory» solitamente no
 - Parametro «Max Server Memory» quasi sempre si

Parametro «Max Server Memory»



- SQL Server < 2012
 - Controllava praticamente solo il Buffer Pool
 - Tecnicamente solo le allocazioni di singole pagine da 8K
- SQL Server >= 2012
 - Controlla tutta memoria (qualunque tipo di allocazione)
 - Eccezione: memoria per «stack» dei «thread»



- Diverse indicazioni per partire
 - Circa il 10% in meno della memoria disponibile
 - Formule più precise che tengono conto OS/applicazioni
 - Numeri preconfezionati
- Importante tenere sotto controllo!
 - Perfmon
 - SQL Server: Buffer Manager\Page Life Expectancy
 - SQL Server: Buffer Manager\Page reads/sec
 - Physical Disk\Disk Reads/sec
 - DMV
 - sys.dm_os_ring_buffers
 - ring_buffer_type = 'RING_BUFFER_RESOURCE_MONITOR'



- Previene paginazione memoria Buffer Cache
 - Messaggi di errore in ERRORLOG
 - Eventi «ringbuffer» delle risorse
- Privilegio per utente associato al servizio SQL Server
 - Policy locale "Lock pages in memory"
- Può essere sintomo di contesa con altre applicazioni
 - memoria insufficiente
 - configurazione «Max Server Memory» inadatta



- Aggiungere memoria: modo semplice per aumentare le prestazioni (specialmente per sistemi OLTP)
- Può essere problematico
 - Costoso
 - Massima capacità (numero slot/dimensione DIMM)
- Aggiungere storage spesso meno problematico
 - Estensione del «Buffer Pool» su dischi SSD

Buffer Pool Extension: come funziona



- Memoria sotto pressione
- «Clean pages»
 - possono essere liberate
 - o scritte nelle estensioni BP
- «Dirty pages»
 - devono essere scritte nei file dati
 - possono essere scritte nelle estensioni BP
- Simile «Storage Tiering» ma controllato da SQL Server

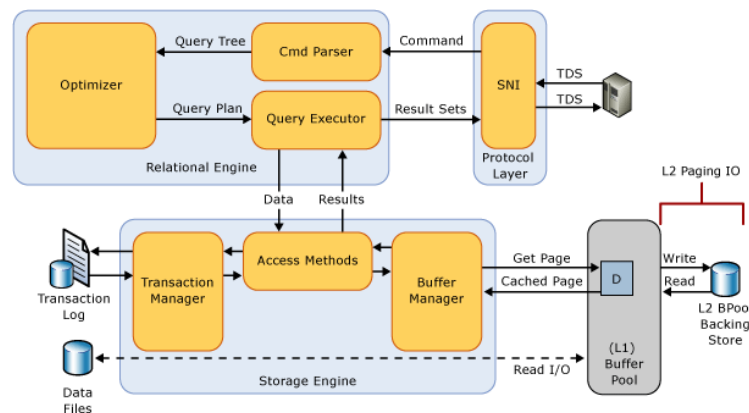


Image source: [http://msdn.microsoft.com/en-us/library/dn133176\(v=sql.120\).aspx](http://msdn.microsoft.com/en-us/library/dn133176(v=sql.120).aspx)

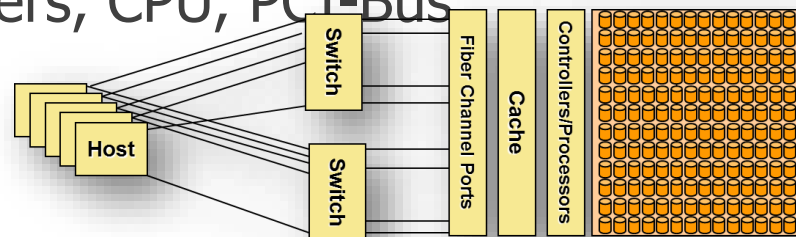


- DBA: «Devo rilasciare un nuovo database»
- SAN Admin: «di quanto spazio hai bisogno?»
- Problema: intendersi sul termine «capacità»
 - Capacità in termini di spazio (Capacity GB/TB)
 - Capacità di trasmissione (Throughput MB/sec o IOPS)
 - Tempi di latenza (Latency ms)



- Scenario classico: SAN monolitica
 - CxO: «La SAN va benissimo! L'abbiamo pagata un occhio della testa e ci girano 200 workstation e una dozzina di altri server senza problemi!»
- Problema: molto spesso sovraccarica
 - Spinto da venditori come sistemi a risorse infinite
 - Tipicamente collo di bottiglia banda o HBA
 - SQL Server con tanti core ma dati non arrivano velocemente

- Parti comuni
 - SQL Server, Windows OS, Drivers, CPU, PCI-Bus
- Storage DAS
 - Controller, Cache, Shelf, Dischi
- Storage SAN
 - HBA, Switch, FC Ports, Storage Processor, Backplane, Dischi
- Qualche livello in più con ambienti virtuali
- Nel Cloud sparisce tutto... meglio o peggio?



Esempio massimo «throughput»

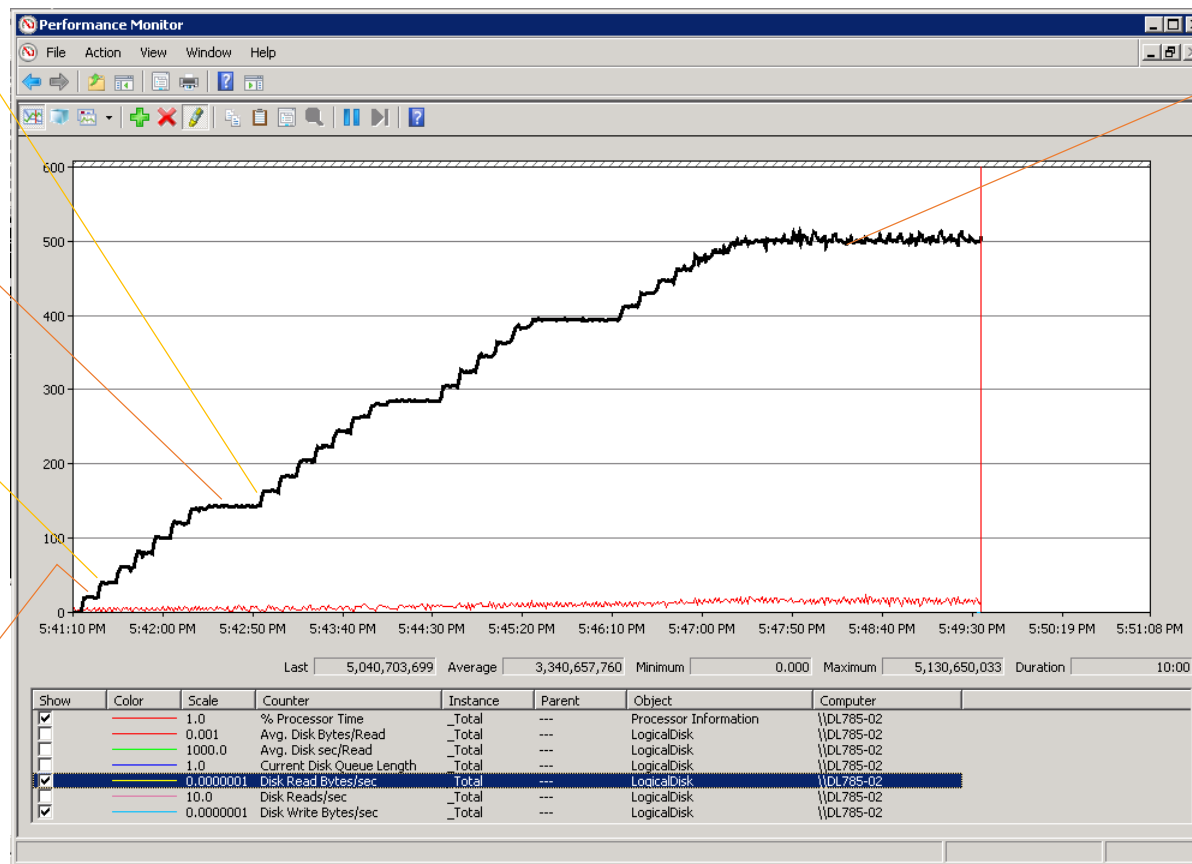


Aggiunta
controller e
disco

Plateau
numero dischi
per controller

Aggiunta disco

Plateau disco



Plateau
numero
controller per
bus/backplane



- Ideale: separare su più volumi dati, log e tempdb
 - Sfruttare al massimo gli accessi in parallelo
 - Ottimizzare diversi pattern di accesso
 - Granularità: tabelle/indici partizionati
 - Ma... deve esserci effettivamente il «ferro» dietro 😊
- Tipo di volumi?
 - Mediazione tra prestazioni e protezione (RAID)
 - Formattazione con «Allocation unit size» da 64KB
 - In genere una partizione per LUN
 - Impatto CHKDSK con volumi di grosse dimensioni



- Diverse tipologie: lettura/scrittura
- Diverse modalità di accesso: sequenziale/random
- Di diverse dimensioni in base all'operazione
- Esempi
 - Seeks: Random Read 8KB
 - Scans: Sequential Read 8KB-512KB (Read-Ahead 1-8MB)
 - Log writes: Sequential Writes up to 60KB
 - Checkpoint/Lazy Writer: Random Writes 8KB-256KB



- OLAP
 - Scan: letture sequenziali 64KB-512KB
 - Caricamenti (bulk load): scritture sequenziali 128KB-256KB
 - Ottimizzare «**throughput**» aggregato (massimizzare MB/sec)
- OLTP
 - Molte operazioni lettura/scrittura random da 8KB
 - Possibile una certa quantità di «read-ahead»
 - Scritture in blocco nei file dati da cache
 - Scritture su transaction log stabili
 - Ottimizzare massimizzando **IOPS** tenendo bassa la **latenza**



- Transaction Log
 - 1-2 millisecondi
 - ideale: ≤ 1 millisecondo
- Dati (sistemi OLTP)
 - 5-20 millisecondi
 - Ideale: ≤ 10 millisecondi
- Dati (sistemi DSS/OLAP)
 - $\leq 25-30$ millisecondi
- Di riferimento! Occorre comprendere i tipi di I/O di SQL



- «Disk Alignment» a 1024K (o multiplo 64K)
 - Default a partire da Windows Vista/Server 2008
 - Default versioni precedenti: disallineato
 - Aumento prestazioni fino al 30%
 - Ormai non se ne vedono più 😊



- Da cosa è usato
 - Oggetti utente (tabelle temporanee, variabili tabella,...)
 - Oggetti interni (row versioning, worktables, sort,...)
- Best Practice
 - Volume dedicato
 - Se non si comprendono a fondo le caratteristiche degli accessi I/O può essere meglio utilizzare un volume condiviso con molte meccaniche
 - Buon candidato SSD locale in molti scenari
 - Trace Flag 1117 e 1118 (SQL Server < 2016)
 - Numero di file

Tempdb: classico esempio di «mito»

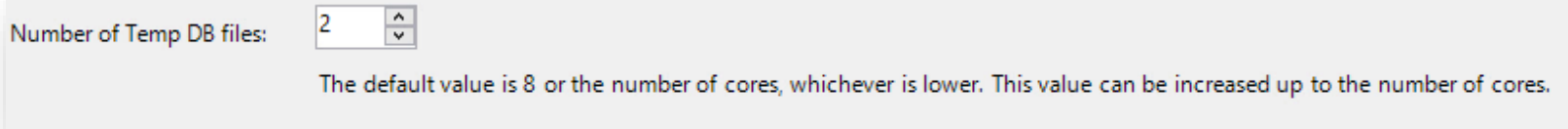
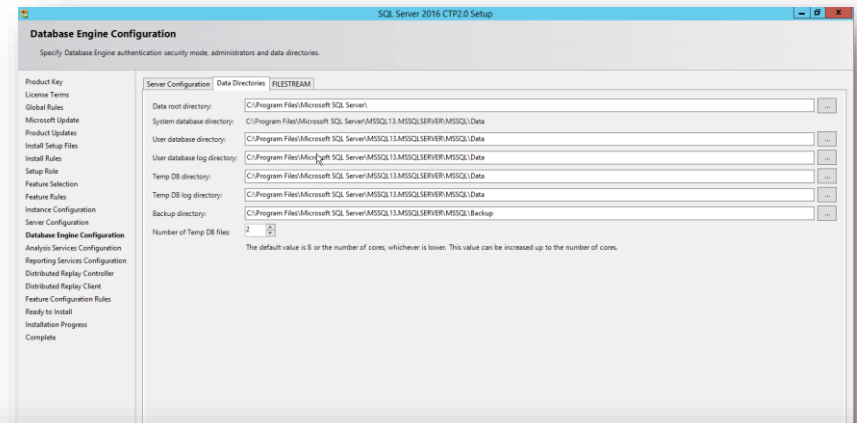


- Numero di file tempdb?
 - «Numero di CPU»?
 - «1/2 numero di CPU»?
 - «1/2 numero di core senza contare HT»?
 - «8»
 - «Nessuno lo sa veramente...»?
 - Dipende!
 - Radice del problema: contesa pagine di allocazione

Numero di file tempdb



- Sessione SQL PASS «inside tempdb»
 - <https://www.youtube.com/watch?v=SvseGMobe2w>
 - Durata 02:42:43! 😊
- Approccio come Setup SQL 2016
 - Numero di core o 8
 - Si può aumentare



Best Practice: Trace Flag 1118



- Impedisce uso di «Mixed Extents»
 - Riduce contesa pagine allocazione SGAM
 - Per tutti i database, problema solitamente con tempdb
 - <https://support.microsoft.com/en-us/kb/2154845>
 - <http://www.sqlskills.com/blogs/paul/misconceptions-around-tf-1118>
- Non più necessario a partire da 2016
 - Sempre attivi per tempdb
 - Estensione ALTER DATABASE per altri database



- In generale l'allocazione va gestita proattivamente!
- «Autogrow» come «paracadute»
- «Autoshrink» solo scenari «mobile» o non presidiati

«Autogrow» e allocazione spazio



- In genere, meglio pre-allocare lo spazio
 - Attesa delle transazioni durante crescita automatica
 - Può introdurre frammentazione a livello di File System
 - Crescita automatica non bilanciata tra file stesso filegroup
 - Buona norma per tutti i database
- Considerare crescita automatica come «paracadute»

Best Practice: Trace Flag 1117



- Forza crescita di tutti i file di un filegroup
 - Non serve per evitare la pre-allocazione ☺
 - Pensata principalmente per tempdb
 - <https://support.microsoft.com/en-us/kb/2154845>
- Non più necessario a partire da 2016
 - Sempre attivi per tempdb
 - Estensione ALTER DATABASE per altri database

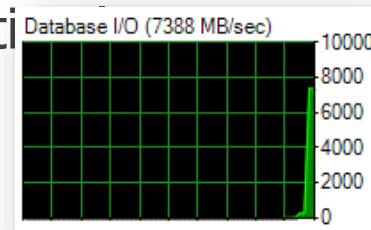


- File non viene inizializzato con zeri
 - Creazione, allocazione e restore se non esiste
 - Valutare rischi di sicurezza (più teorici che altro...)
 - Solo file dati, no log!
- Privilegio per utente associato al servizio SQL Server
 - SE_MANAGE_VOLUME_NAME
 - Policy locale "Perform Volume Maintenance Tasks"
 - Integrato nel Setup da SQL 2016+
- Non serve per evitare pre-allocazione! 😊



- In genere, «shrink» operazione straordinaria
 - Crescita imprevista es. dovuta a query fuori controllo
 - «Autoshrink» pensata per scenari non presidiati
 - Es. mobile, workstation utenti
 - Principalmente per edizione Express
- Introduce frammentazione
- Se log deve ricrescere, deve essere inizializzato
 - Non può usare «Instant File Initialization»
 - Transazioni devono attendere!

- Metodologia architetturale
 - Per rilasciare sistemi configurati in maniera ottimale
 - Legata a carichi di lavoro «Data Warehouse»
- Configurazione Hardware verificata con «vendor/partner»
- Personalizzabile
- Non adatta per carichi di lavoro generici
- Analogo OLTP non esiste





- «Non posso disabilitarlo?»
- «Fortunatamente no!» 😊
- A partire da SQL Server 2014
 - In-Memory OLTP (tabelle persistenti solo in memoria)
 - Delayed Durability / Lazy Commit (commit asincrono)



- Transazione può essere
 - «Fully Durable»
 - protocollo WAL tradizionale, commit sincrono
 - «Delayed Durable»
 - AKA «lazy commit» o commit asincrono
- «Eventual Durability» 😊
 - Record Transaction Log mantenuti nel buffer
 - Buffer scritto quando pieno o evento di «Flush»
 - Possibile perdita della transazione!
 - in modo consistente
 - **Applicazione deve tollerare la perdita di dati!**

«Delayed Durability Flush»



- Buffer Transaction Log pieno
- Commit transazione «fully durable»
 - stesso database
- Esecuzione manuale procedura `sp_flush_log`
- Attenzione! Shutdown SQL Server
 - potrebbe non scrivere il buffer
 - va trattato come un possibile evento di perdita dati

Controllo «Delayed Durability»



- Livello Database

```
ALTER DATABASE ... SET DELAYED_DURABILITY = { DISABLED | ALLOWED | FORCED }
```

- Livello «Atomic block» («Native Compilation»)

```
CREATE PROCEDURE <procedureName> ...  
WITH NATIVE_COMPILATION, SCHEMABINDING, EXECUTE AS OWNER  
AS BEGIN ATOMIC WITH  
(   DELAYED_DURABILITY = ON,  
    TRANSACTION ISOLATION LEVEL = SNAPSHOT,  
    LANGUAGE = N'English'  
    ... )  
END
```

- Livello «Commit»

```
COMMIT [ { TRAN | TRANSACTION } ] [ transaction_name | @tran_name_variable ] ]  
[ WITH ( DELAYED_DURABILITY = { OFF | ON } ) ]
```

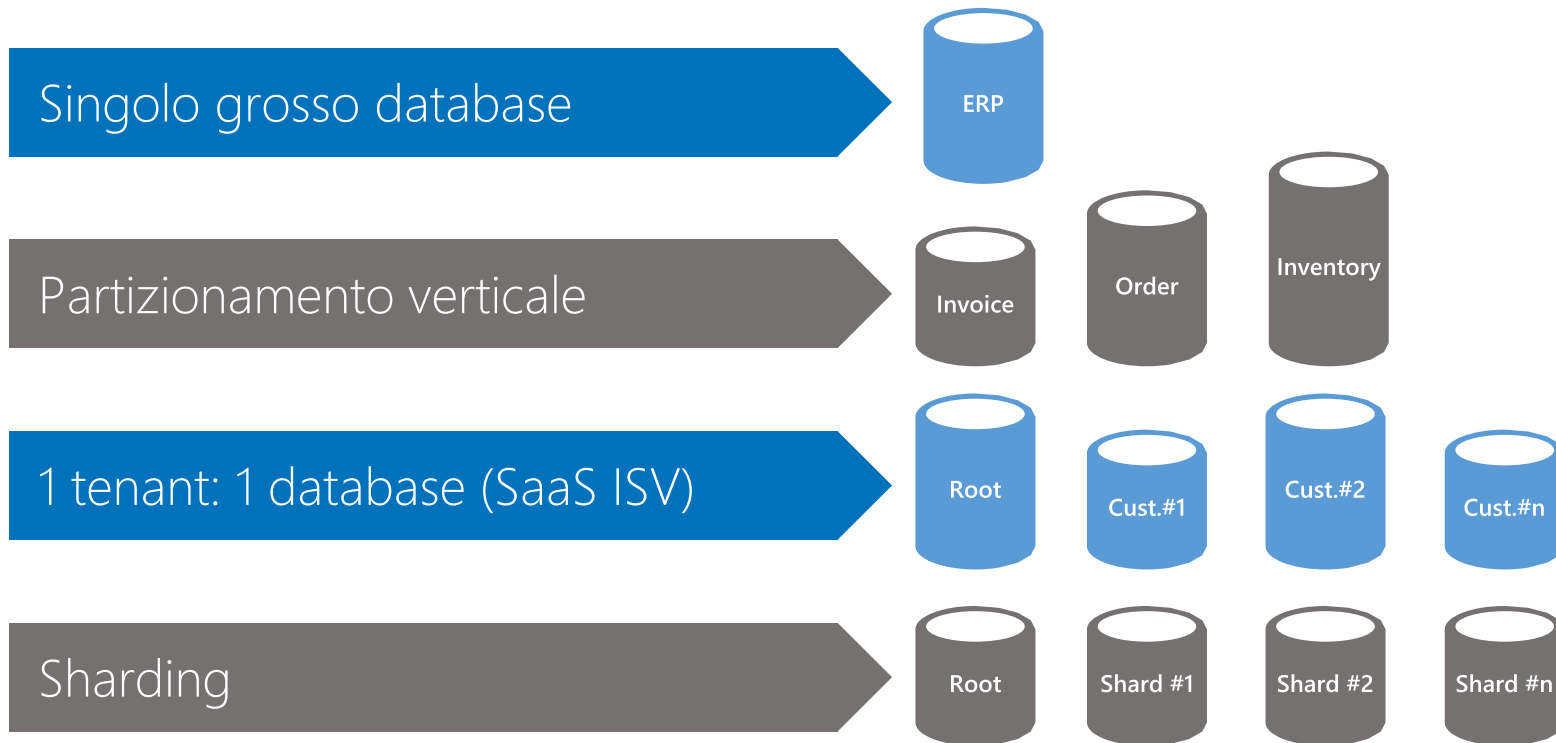



- A livello di riga
 - In realtà nuovo formato più compatto
 - Tipi dato storicamente a lunghezza fissa ora variabile
 - Spazio occupato solo per rappresentare i valori
 - Potenzialmente utile anche per sistemi OLTP
- A livello di pagina
 - Compressione per prefisso (colonna)
 - Compressione a dizionario (intera pagina)
 - Principalmente utile per sistemi OLAP
- Vantaggio
 - più pagine (comprese) nella «Buffer Cache»



- Se già è difficile il dimensionamento per un database...
- Molte domande da porsi
 - Server fisico o virtuale?
 - Server dedicato solo al database Engine?
 - Istanza singola o più istanze?
 - On-premise o Cloud?
 - Sharding?
- In generale per il consolidamento
 - Database Tier 1 richiedono risorse dedicate
 - Database Tier 2 richiedono una buona progettazione
 - Database Tier 3 c'è più flessibilità...

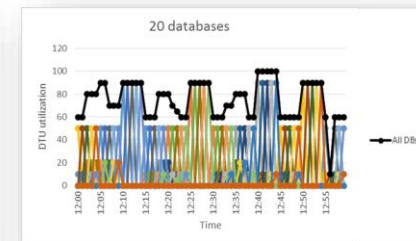
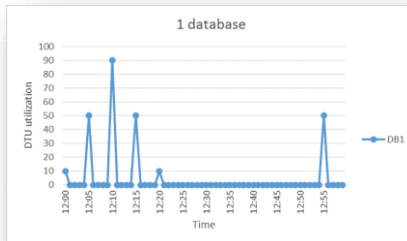
Pattern comuni di scalabilità



Problema tipico SaaS



- Carico di lavoro disomogeneo
 - Clienti di dimensione diversa
 - Collocazione geografica e «Time Zone»
 - Stagionalità diverse
- Spesso «Over-provisioning» risorse
 - Per assorbimento picchi di utilizzo
 - Poco efficiente dal punto di vista dei costi



«Elastic Database Pools»



- Collezione di DTU (eDTUs) e «Storage» (GBs) condivisi
- «Auto-Scale»
- Database aggiunti/rimossi a caldo

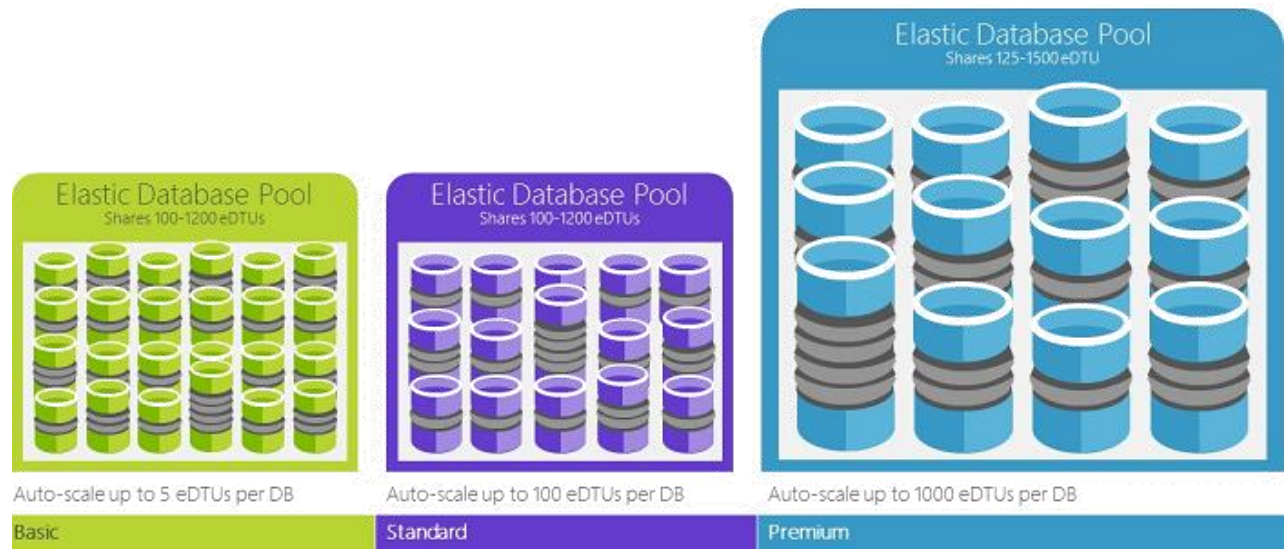


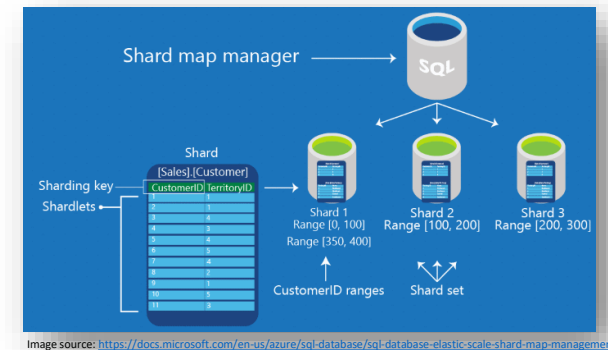
Image source: <https://docs.microsoft.com/en-us/azure/sql-database/sql-database-what-is-a-dtu>

- Molteplici database condivisi da più «tenant»?
- Tecnica «Scale out» distribuzione dati
 - Strutturati in maniera identica
 - In più database indipendenti
 - In base a «Sharding Key»
 - Mappature per intervallo di valori o lista

«Elastic Database client library»



- «Shard Map Management»
 - Mappatura «Shard Keys» e database
 - «Shard Keys» liste o intervalli di valori
- «Data Dependent Routing»
 - Supporto apertura connessione in base a «Shard Key»
- «Multi-Shard Queries»
 - Supporto Query che coinvolge più «Shard»
 - Fusione unico «Result Set» con Semantica UNION ALL



«Elastic Database split-merge tool»



- «Split Range» distribuisce una «Shard» su due «Shard»
- «Merge Range» accorpa due «Shard» in una «Shard»
- «Move Shardlet»

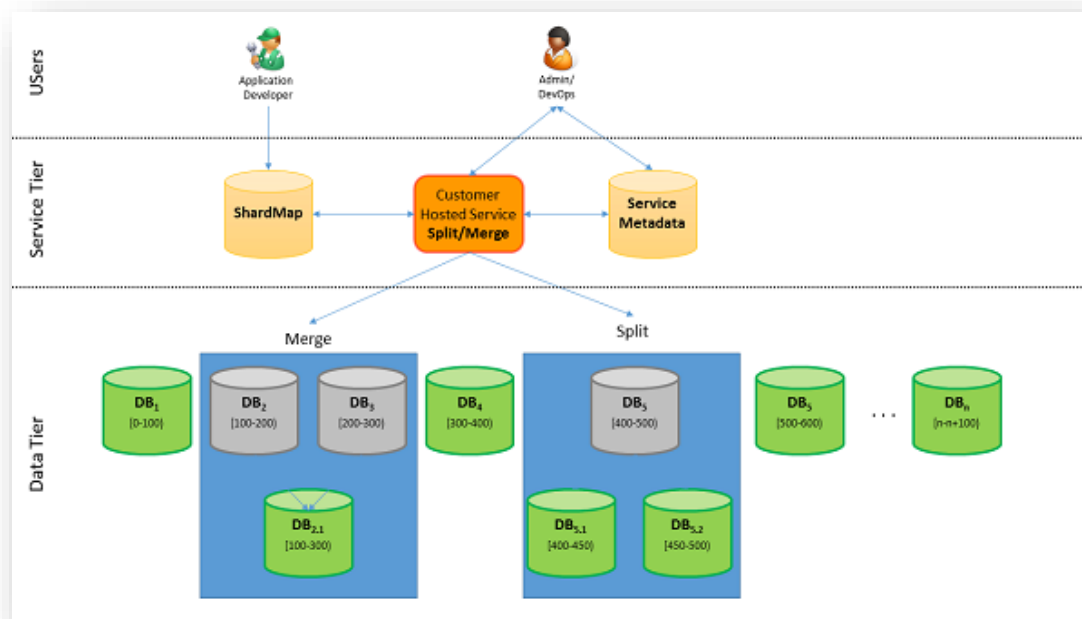


Image source: <https://azure.microsoft.com/en-us/documentation/articles/sql-database-elastic-scale-overview-split-and-merge>



- Esecuzione Script T-SQL o applicazione DACPAC
 - Collezione database personalizzata
 - Tutti i database di «Elastic Database Pool»
 - «Shard Set»
- Ritenta in caso di fallimento
- Schedulazione



- Attività amministrative
 - es. «deploy» nuovo schema
- Ricostruzione degli indici
- Aggiornamento di dati in comune a più database
- Raccolta dati di telemetria in tabella comune

«Elastic Database Queries»



- Supporto T-SQL per query distribuite
 - CREATE EXTERNAL DATA SOURCE
 - CREATE EXTERNAL TABLE
- Partizionamento verticale «cross-database queries»
 - TYPE = RDBMS
- Partizionamento orizzontale «Sharding»
 - TYPE = SHARD_MAP_MANAGER

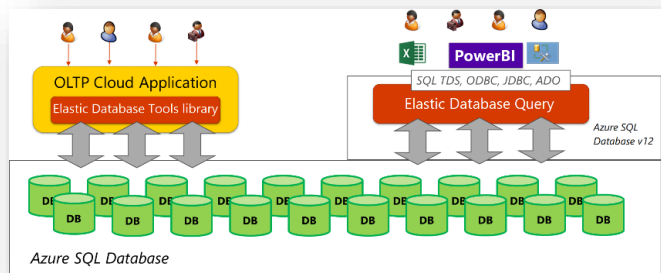


Image source: <https://docs.microsoft.com/en-us/azure/sql-database/sql-database-elastic-query-overview>



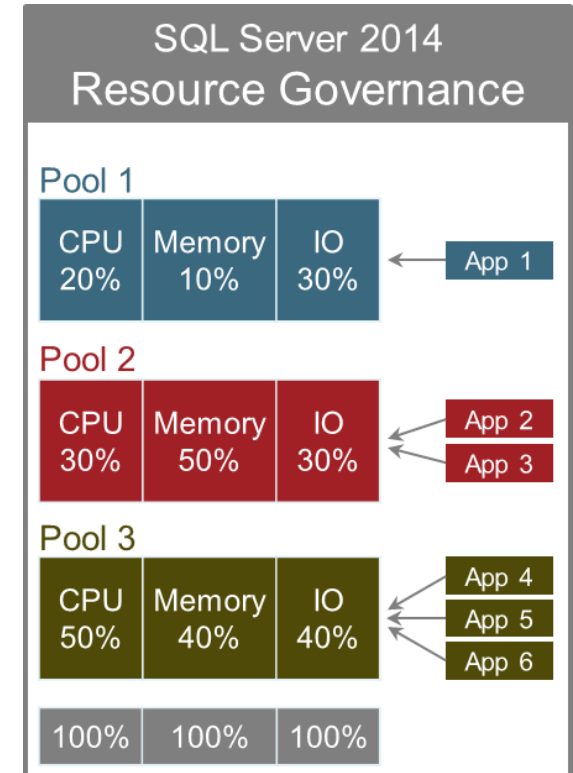
- Best Practice Analyzer (PBA)
 - Verifica conformità installazione
 - Fornisce link a spiegazione
 - Solo vecchie versioni 2008R2/2012 ☹
- Funzionalità Policy Based Management
 - Informalmente sostituisce BPA
 - <https://blogs.msdn.microsoft.com/poojakamath/2018/05/03/where-is-my-sql-bpabest-practice-analyzer-for-sql-201420162017>
- Stored procedure sp_Blitz
 - <https://github.com/BrentOzarULTD/SQL-Server-First-Responder-Kit>
 - altre procedure utili per monitoring/troubleshooting

«Policy Based Management»



- Infrastruttura per
 - Definizione «policy»
 - Controllo conformità installazione con «policy»
- Serie di «policy» conformi a Best Practice
 - Categoria di policy per sicurezza!
- Enterprise Policy Management Framework
 - <http://aka.ms/epmframework>


- Classificazione carico di lavoro
 - In «Workload Group»
 - Tramite funzione T-SQL di classificazione
 - Limiti utilizzo
 - Massimo grado parallelismo
 - Percentuale massima memoria del pool per singola query
 - Numero massimo richieste concorrenti
- Definizione gruppi di risorse («Pool»)
 - Limiti di utilizzo per memoria, CPU, IOPS (da 2014)
- Controllo uso risorse
 - Assegnazione «Pool» a «Workload Group»
 - Alcuni Limiti globali





- Non sottovalutare manutenzione ordinaria!
- Soluzione supportata dalla Community
 - Backup
 - Controllo integrità
 - Manutenzione Indici e Statistiche
 - <https://ola.hallengren.com>
- Altro?
 - SQL Server Agent Alerts
 - Directory Log di SQL Server

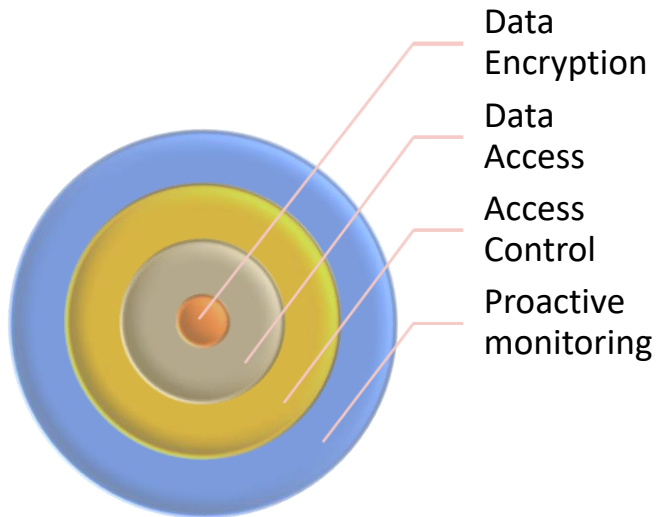
- Directory Log di SQL Server
 - ERRORLOG*
 - SQLAGENT*
 - FDLAUNCHERRORLOG*
 - SQLFT*
 - Stack Dump in caso di crash! Sono importanti!
- ERRORLOG in particolare
 - Limitare le dimensioni!
 - Stored procedure sp_cycle_errorlog
 - Numero di file per rotazione da SSMS

Name	Size	Type	Modified	Attr
 ERRORLOG.1	32,5 GB	1 File	08/08/2016 12:47	-a----



- Avete dato un'occhiata ai report SSMS ultimamente?
 - Availability Group Latency
 - Performance Dashboard
- Glenn Berry Diagnostic Information Queries
 - <https://www.sqlskills.com/blogs/glenn/category/dmv-queries>
- Ma soprattutto.... sp_whoisactive!
 - Wait Stats, blocking, piani esecuzione ecc. ecc.
 - Installare su tutte le istanze! 😊
 - 28 articoli didattici
 - <http://whoisactive.com>

«Security Layering»



Data Encryption

- Transport Layer Security (in transit)
- Transparent Data Encryption (at rest)
- Cell-Level Encryption (at rest)
- Always Encrypted (at rest and in transit)

Data Access

- Dynamic Data Masking
- Row-Level Security

Access Control

- Encrypted Authentication
- SQL Firewall*

Proactive monitoring

- Auditing
- Threat Detection*

Sviluppo

Best practices on SQL Server

- Discorso veramente troppo lungo
 - Complesso a livello concettuale/logico
 - Pieno di dettagli implementativi a livello fisico
- In generale: non perdere di vista le basi
 - Modello relazionale (e limiti di SQL)
 - Normalizzazione



- Tipi dato complessi
 - XML, GEOMETRY, GEOGRAPHY, HIERARCHYID, FILESTREAM, FILETABLE, JSON
 - «Type System» estensibile con integrazione CLR
 - Integrazione SQL (XPATH/XQUERY, funzioni JSON)
- Sparse Column
- In-Memory OLTP/OLAP
- Graph Processing
- Temporal Tables

«In-Memory OLTP»



- Nome in codice «Hekaton»
 - Ancora usato nella documentazione
- Esigenza: sfruttare al meglio RAM/CPU
 - Costo della RAM in costante decrescita
 - Frequenza CPU non cresce più esponenzialmente
 - Anche se numero di core continua a crescere
 - Serviva tecnologia che eseguisse stesse operazioni in meno cicli

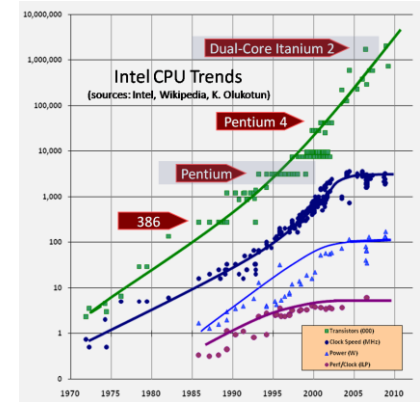
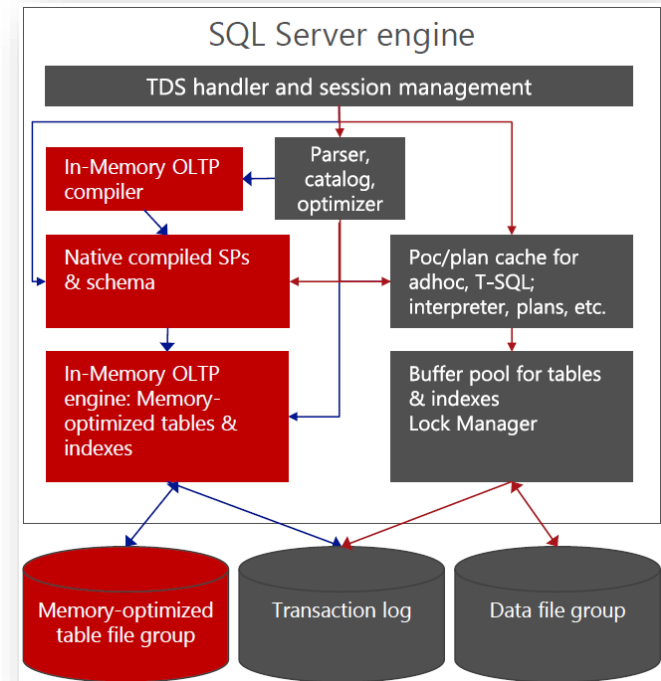


Chart source: <http://gotw.ca/publications/concurrency-ddj.htm>

Componenti principali «In-Memory OLTP»



- Tabelle «Memory-Optimized»
- Moduli T-SQL compilati
- Strutture dati lock e latch free
- Completamente integrate
- Superficie programmabilità
 - V1 in 2014 limitata
 - V2 in 2016 molti limiti rimossi
 - V3 in 2017 ancora meno limiti



«Memory Optimized Tables»



- Struttura
 - Completamente in memoria
 - Indici di tipo hash e range (BW-tree)
 - Compilazione schema e metodi accesso
- Persistenza (durability)
 - Solo Schema o Schema e dati
- Storage
 - Data file e delta file usano tecnologia FILESTREAM
 - Transaction-log



- Compilazione codice nativo
 - «Stored Procedures»
 - «After Triggers»
 - «Inline TVFs»
 - «Scalar UDFs»
- Massime prestazioni ma...
 - svariate limitazioni
 - rimosse in ogni nuova versione



- Nuovo protocollo per transazioni MVCC
 - Conforme proprietà ACID
 - Molteplici versioni delle righe
 - Completamente lock-free (timestamp based)
 - Garbage Collector versioni non più utili
- Metodi di accesso «latch/spinlock-free»
 - Operazioni CAS (Compare & Swap)
 - Logica di «Thread assist/abort»



- Protocollo ottimistico
 - Possibili conflitti tra scritture
 - Validazione prima della commit potrebbe fallire
 - Gestione e logica di «retry» necessaria

«In-Memory OLTP» in SQL 2016



- Formato «Storage» cambiato
- Molti miglioramenti
 - Supporto per aggiornamento e campionamento statistiche
 - ALTER TABLE eseguito in parallelo e scrittura solo metadati nel T-Log
 - Miglioramenti T-SQL
 - Es. UNIQUE, FK, CHECK, TRIGGERS (solo AFTER), LOB (colonne)
 - Esteso supporto moduli T-SQL compilati nativamente
 - Es. UNION, UNION ALL, DISTINCT, OUTER JOIN, «Subqueries», OUTPUT, LOB (variabili/parametri), «Inline-TVF» e «Inline-UDF» compilate
 - Supporto per piani di esecuzione paralleli
 - <https://docs.microsoft.com/en-us/sql/database-engine/whats-new-in-sql-server-2016#InMemory>

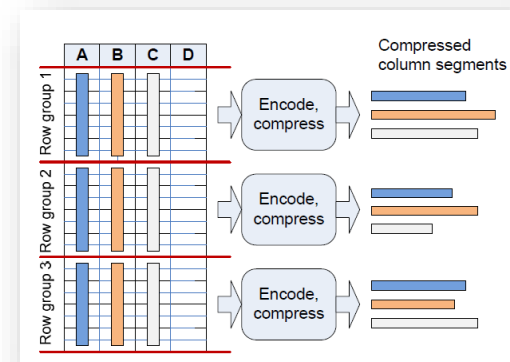


- Supporto colonne calcolate
 - Incluso indici su colonne calcolate
- Supporto funzioni JSON
 - Sia moduli T-SQL compilati nativamente che vincoli CHECK
- Esteso supporto moduli T-SQL compilati nativamente
 - CROSS APPLY, CASE, TOP (N) WITH TIES, sp_spaceused, sp_rename
- Prestazioni
 - Fase REDO ripristino log delle transazioni effettuata in parallelo

«In-Memory OLAP» («ColumnStore indexes»)



- Particolarmente efficienti per DWH
 - Scansione/aggregazione grossi volumi
 - Organizzazione per colonne
 - Compressione a più livelli
 - Modalità di accesso «batch»
- Introdotti in SQL Server 2012
 - Estesi significativamente ad ogni nuova versione
 - Disponibili in tutte le edizioni a partire da
- Forza bruta per query imprevedibili
 - Indice Columnstore non-clustered su tutte le colonne





- Solo un indice «non-Clustered Columnstore» per tabella
- Richiede sempre spazio extra (tabella rimane un B-Tree/Heap)
- Tabella non può esser aggiornata direttamente
- Molte limitazioni
 - Tipi dato
 - Schema
 - Interoperabilità con altre funzionalità



- «Clustered Columnstore»
 - Meccanismo di «storage» primario
 - Aggiornabile
 - Nessun altro indice consentito su tabella
- «Non-Clustered Columnstore»
 - Come in SQL Server 2012
- Opzione di compressione «Archival»
- Più operatori in modalità «Batch»
 - Scan, Filter, Project, Join, Group By, Union All
 - Ancora solo query parallele



- «Clustered Columnstore»
 - Uno o più indici «non-Clustered Rowstore» supportati
 - PK e FK possono essere supportati da indici «non-Clustered Rowstore»
- «Non-Clustered Columnstore»
 - Supporto per un solo indici aggiornabile su tabelle «Rowstore»
 - Supporto per condizione di filtro (es. dati «caldi» di un carico di lavoro OLTP)
- Tabelle «In-Memory» possono avere un indice «Columnstore»
- «Compression Delay»
 - Evita frammentazione quando dati caldi cambiano frequentemente con carichi di lavoro OLTP
- Ancora più operatori in modalità «Batch»
 - sort, aggregates with multiple distinct functions, window aggregate functions, window user-defined aggregates, window aggregate analytic functions
 - Supporto per query «Single-Threaded» (MAXDOP 1)

Indici «Columnstore» in SQL Server 2016



- Supporto per livelli di isolamento transazionale RCSI e SI
- Deframmentazione indici tramite riorganizzazione
- Supporto per repliche secondarie AlwaysOn AG
- «Pushdown» aggregazioni e predicati su stringhe

Feature	RTM				SP1			
	Standard	Web	Express	Local DB	Standard	Web	Express	Local DB
Row-level security	Yes	No	No	No	Yes	Yes	Yes	Yes
Dynamic Data Masking	Yes	No	No	No	Yes	Yes	Yes	Yes
Change data capture*	No	No	No	No	Yes	Yes	No*	No*
Database snapshot	No	No	No	No	Yes	Yes	Yes	Yes
Columnstore	No	No	No	No	Yes	Yes	Yes	Yes
Partitioning	No	No	No	No	Yes	Yes	Yes	Yes
Compression	No	No	No	No	Yes	Yes	Yes	Yes
In Memory OLTP	No	No	No	No	Yes	Yes	Yes	No**
Always Encrypted	No	No	No	No	Yes	Yes	Yes	Yes
PolyBase	No	No	No	No	Yes	Yes	Yes	No
Fine grained auditing	No	No	No	No	Yes	Yes	Yes	Yes
Multiple filestream containers	No	No	No	No	Yes	Yes	Yes	No**

SQL Server 2016 SP1 Edition	Columnstore Index Quota (per Instance)
Express	352 MB
Web	16 GB
Standard	32 GB
Developer	Unlimited
Enterprise	Unlimited

Indici «Columnstore» in SQL 2017

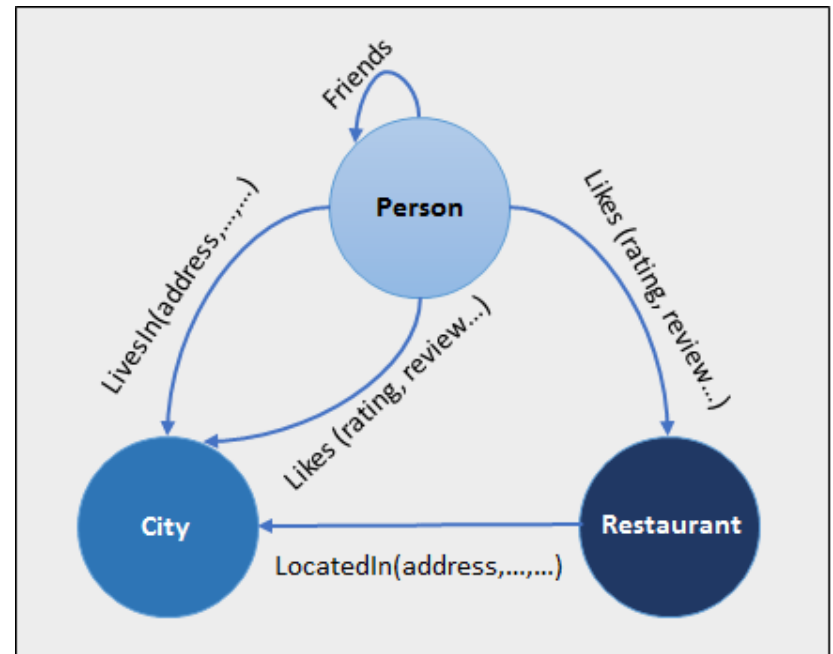
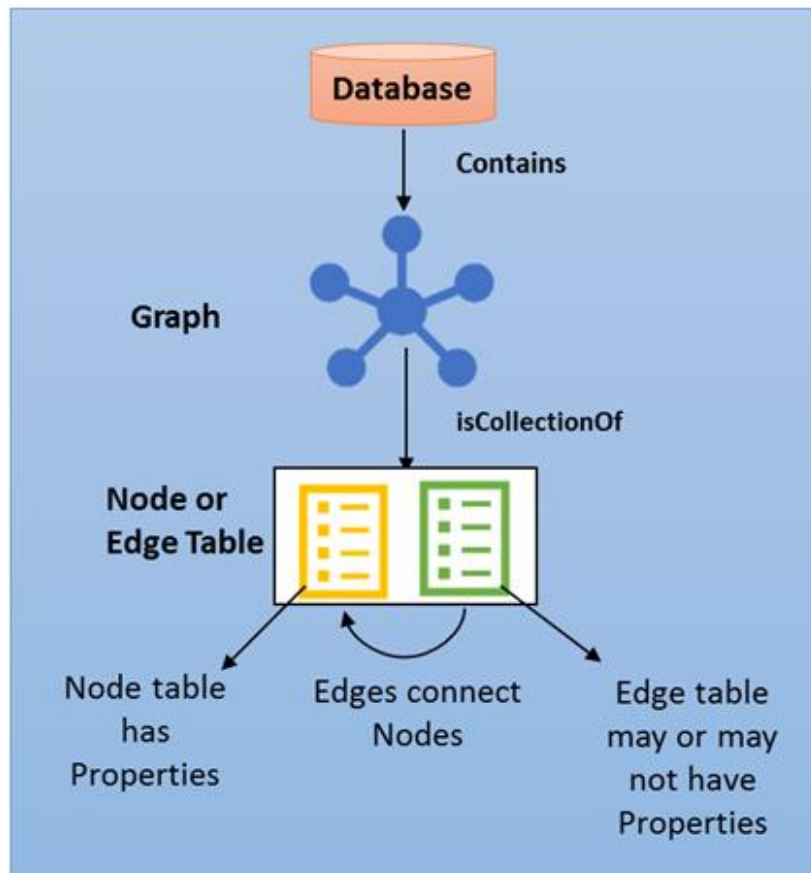


- Supporto per colonne calcolate non persistenti
- Supporto per LOB
 - Solo «Clustered Columnstore»
 - Già disponibile anche in Azure SQL Database (Premium)
 - Es. caso di studio utilizzo dati JSON
 - <https://blogs.msdn.microsoft.com/sqlserverstorageengine/2017/02/09/json-data-in-clustered-column-store-indexes>
- Supporto «online build/rebuild» indici «non-clustered Columnstore»
- Riepilogo funzionalità per versione prodotto
 - <https://docs.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-what-s-new>



- Tabella «Node»
 - Rappresenta un entità
- Tabella «Edge»
 - Rappresenta una relazione «many-to-many»
 - Può avere proprietà
 - Ha un orientamento (digrafo), connette due nodi
- MATCH comando T-SQL
 - Condizione di ricerca per grafi
 - «Pattern matching» e attraversamento

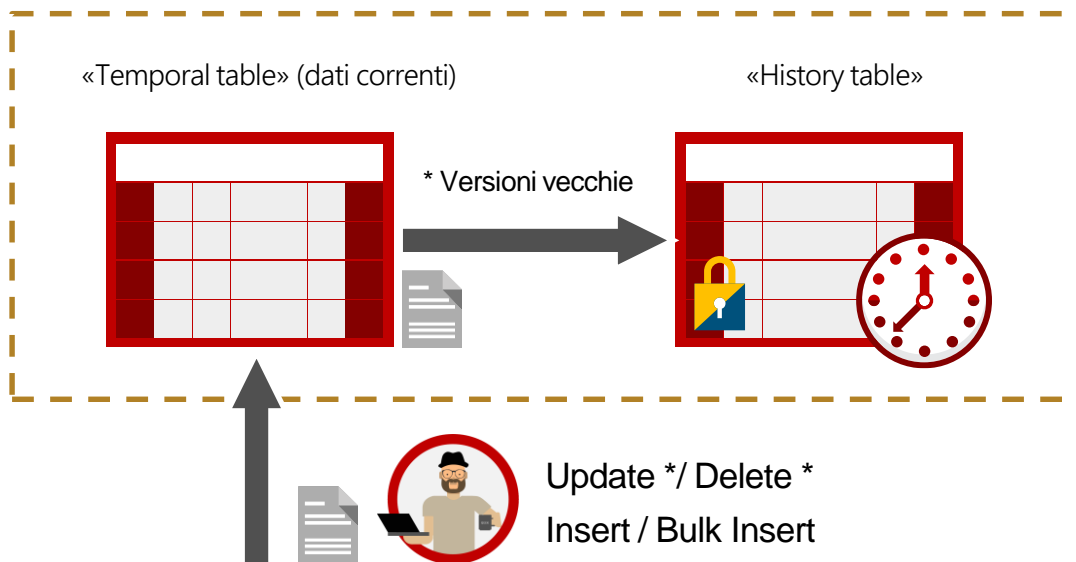
«Graph Processing»



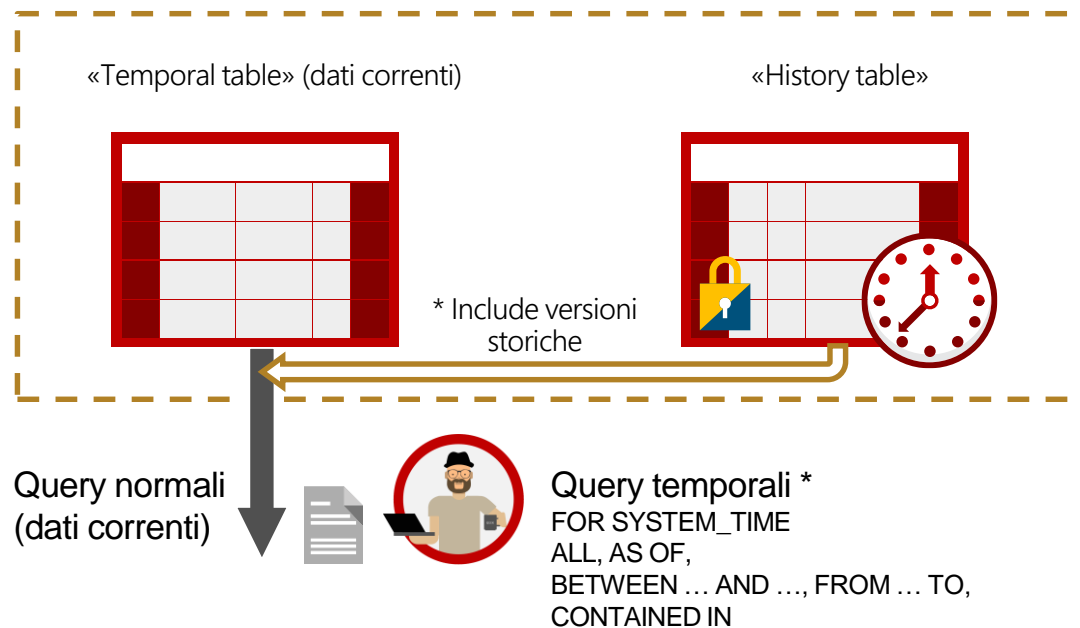


- Tabelle temporanee locali/globali non supportate
- Tipi/variabili tabella non supportate
- Tabelle temporali «system-versioned» non supportate
- Tabella «memory optimized» non supportate
- Non si possono aggiornare direttamente (UPDATE)
 - due «Node» collegati tramite un «Edge», soluzione
 - inserire nuove «Edge» che punta a differenti «Node»
 - cancellare precedente «Edge»
- Query «cross-database» con oggetti «Graph» non supportate

Modifiche Database Temporali



Interrogazioni Database Temporali



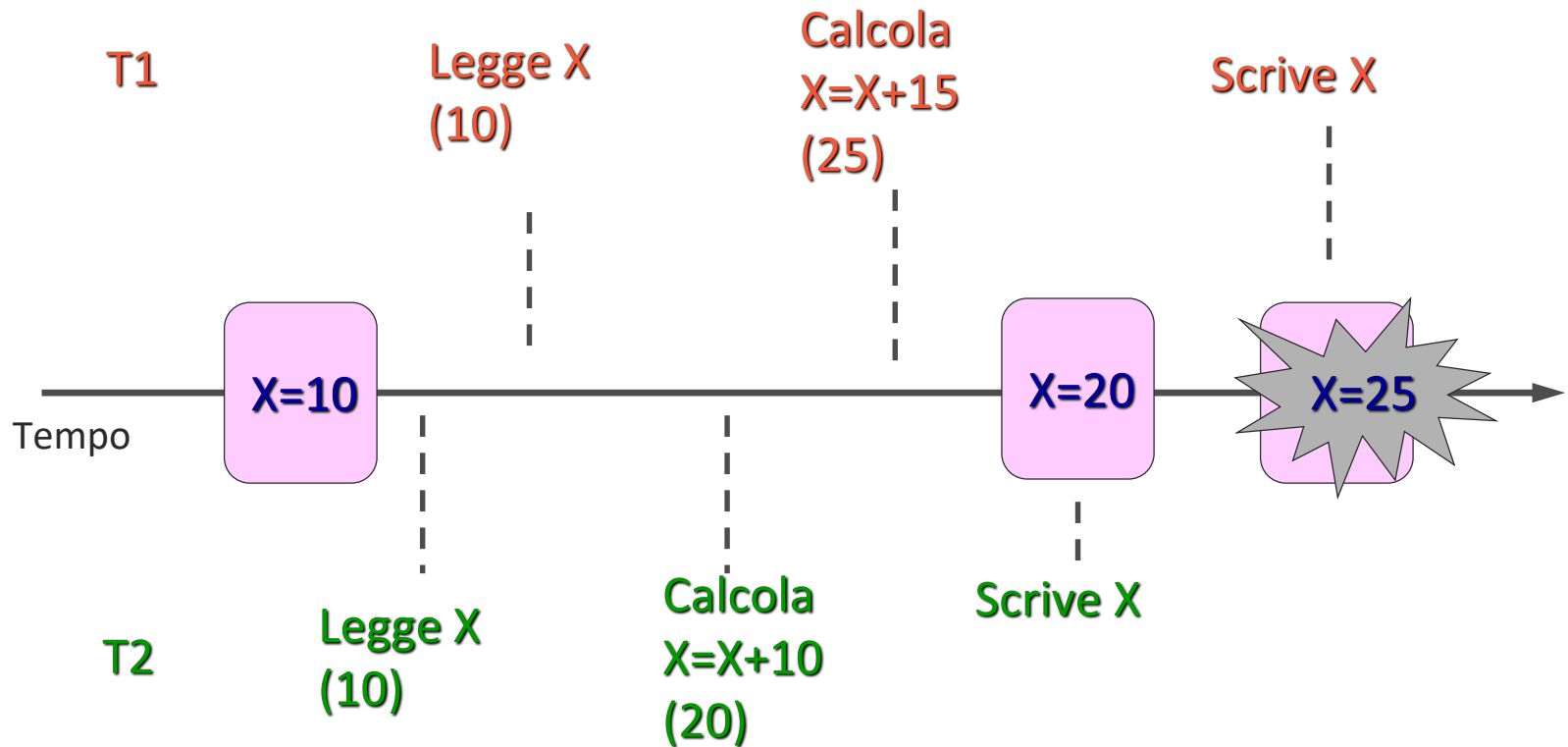


- Idealmente: serializzazione delle transazioni
 - massima consistenza
 - minor concorrenza
- Se le transazioni non sono coordinate
 - perdita di aggiornamenti (lost updates)
- Se transazioni sono parzialmente isolate
 - letture inconsistenti («dirty reads»)
 - letture non ripetibili («non-repeatable reads»)
 - letture fantasma («phantoms»)
 - conflitto in aggiornamento («update conflict»)

Perdita di aggiornamenti



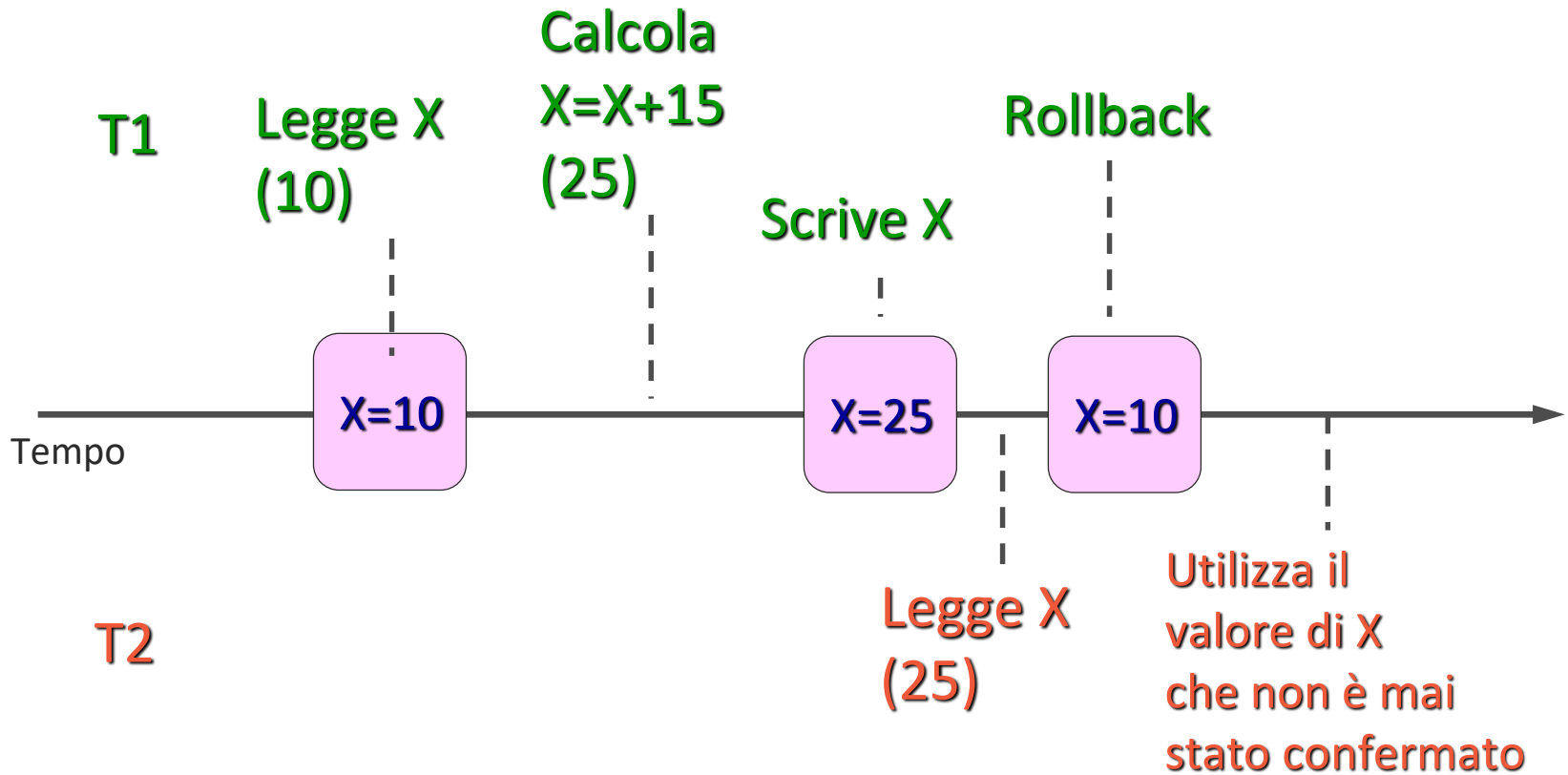
- Aggiornamento non sincronizzato



Lecture inconsistenti



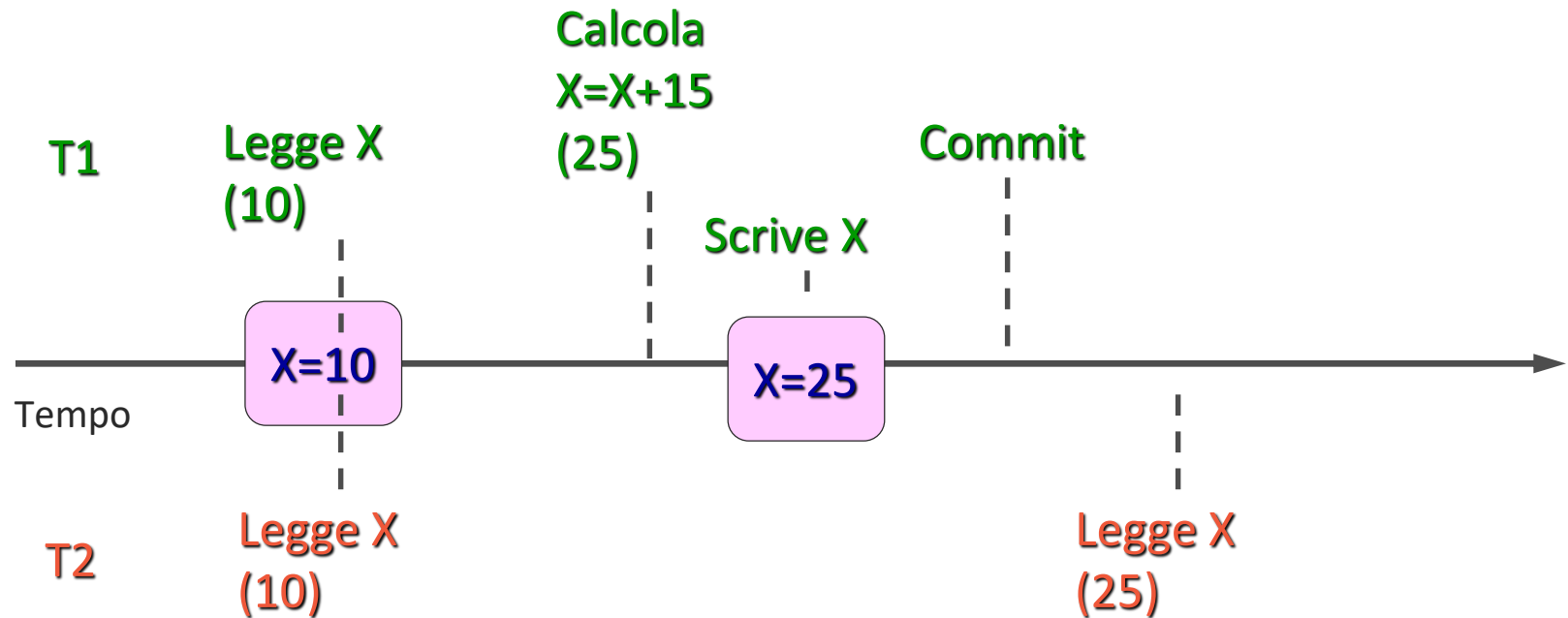
- T2 vede le modifiche non confermate di T1



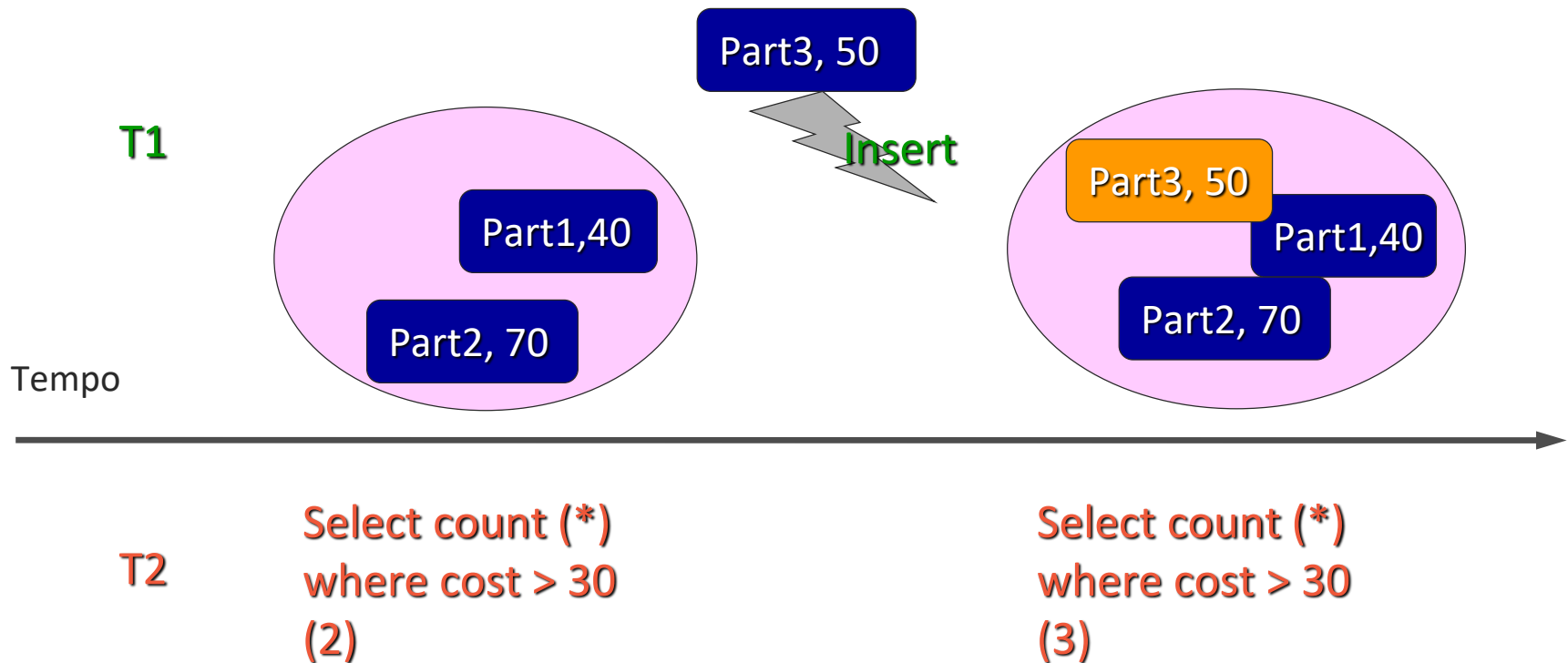
Lecture non ripetibili



- Lettura successiva inconsistente



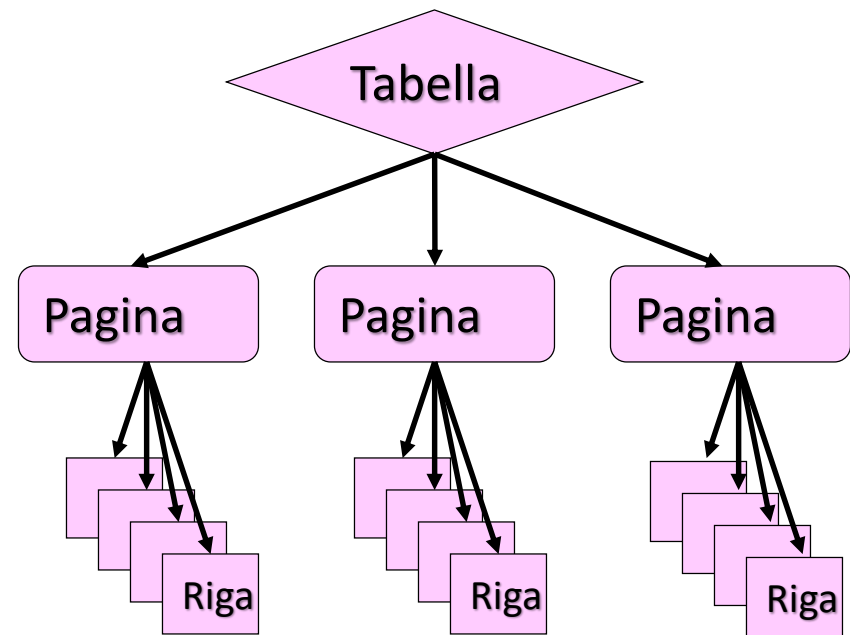
- Lettura intervallo successiva inconsistente





- 4 livelli supportati per standard ANSI
 - «Read Uncommitted», «Read Committed», «Repeatable Read», «Serializable»
- Predefinito «Read Committed» in due modalità
 - pessimistico, usa protocollo di «Locking»
 - ottimistico, usa «Row Versioning»
- Aggiunta di un ulteriore livello «Snapshot»
 - simile a «Serializable» ma ottimistico in scrittura
 - genera errore alla «Commit» in caso di conflitto

- Tipologie di lock
 - Condivisi (S)
 - Esclusivi (X)
 - Aggiornamento (U)
- Intenzione (I)
- Granularità
 - Riga/chiave
 - Intervallo tra chiavi
 - Pagina
 - Tabella
 - «Extent»
 - Database



- Sottoinsieme della matrice di compatibilità

Lock già rilasciato								
Lock richiesto	IS	S	U	IX	X	SchS	SchM	BU
IS	Si	Si	Si	Si	No	Si	No	No
S	Si	Si	Si	No	No	Si	No	No
U	Si	Si	No	No	No	Si	No	No
IX	Si	No	No	Si	No	Si	No	No
X	No	No	No	No	No	Si	No	No
SchS	Si	Si	Si	Si	Si	Si	No	Si
SchM	No	No	No	No	No	No	No	No
BU	No	No	No	No	No	Si	No	Si

«Read Committed» (pessimistico)



```
CREATE TABLE t1 ( c1 int unique, c2 int)
INSERT INTO t1 VALUES (1, 5)
```

Transazione 1

```
BEGIN TRAN
```

```
UPDATE t1
```

```
SET      c2 = 9
```

```
WHERE    c1 = 1
```

```
COMMIT TRAN
```

Transaction 2 (Read Committed)

```
BEGIN TRAN
```

```
SELECT c2 FROM t1
```

```
WHERE  c1 = 1
```

```
-- SQL Server risponde 9
```

```
COMMIT TRAN
```

Bloccato!



Tempo

«Read Committed Snapshot» (ottimistico)



```
CREATE TABLE t1 ( c1 int unique, c2 int)
INSERT INTO t1 VALUES (1, 5)
```

Transazione 1

```
BEGIN TRAN
```

```
UPDATE t1
```

```
SET      c2 = 9
```

```
WHERE    c1 = 1
```

```
COMMIT TRAN
```

Transaction 2

(Read Committed Snapshot)

```
BEGIN TRAN
```

```
SELECT c2 FROM t1
```

```
WHERE  c1 = 1
```

-- SQL Server risponde 5

```
SELECT c2 FROM t1
```

```
WHERE  c1 = 1
```

-- SQL Server risponde 9

```
COMMIT TRAN
```



Tempo

«Snapshot» (in lettura)



```
CREATE TABLE t1 (c1 int unique, c2 int)
INSERT INTO t1 VALUES (1, 5)
```

Transazione 1

```
BEGIN TRAN
UPDATE t1
SET      c2 = 9
WHERE    c1 = 1

COMMIT TRAN
```

Transazione 2 (Snapshot Isolation)

```
SET TRANSACTION ISOLATION LEVEL
SNAPSHOT

BEGIN TRAN

SELECT c2 FROM t1 WHERE c1 = 1
      -- SQL Server risponde 5

SELECT c2 FROM t1 WHERE c1 = 1
      -- SQL Server risponde 5

COMMIT TRAN

SELECT c2 FROM t1 WHERE c1 = 1
      -- SQL Server risponde 9
```



«Snapshot» (in aggiornamento)



```
CREATE TABLE t1 ( c1 int unique, c2 int)
INSERT INTO t1 VALUES (1,5)
```

Transazione 2

```
BEGIN TRAN
UPDATE t1
SET      c2 = 9
WHERE    c1 = 1
COMMIT TRAN
```

transazione 1 (Snapshot Isolation)

```
SET TRANSACTION ISOLATION LEVEL
SNAPSHOT
```

```
BEGIN TRAN
```

```
SELECT c2 FROM t1 WHERE c1 = 1
```

-- SQL Server risponde 5

```
UPDATE t1
SET      c2 = 15
WHERE    c1 = 1
```

Bloccato!

Rollback perché conflitto tra aggiornamenti

Tempo

Fenomeni permessi

Livelli di isolamento	Dirty Read	Non-Repeatable Read	Phantoms	Update Conflict	Modello concorrenza
Read Uncommitted	Si	Si	Si	No	
Read Committed 1 Locking 2 Snapshot	No No	Si Si	Si Si	No No	Pessimistico Ottimistico
Repeatable Read	No	No	Si	No	Pessimistico
Snapshot	No	No	No	Si	Ottimistico
Serializable	No	No	No	No	Pessimistico



- Mediazione
 - livello alto: più consistenza, meno prestazioni
 - livello basso: meno consistenza, più prestazioni
- Modello pessimistico o ottimistico?
 - dipende... quasi sempre va bene ottimistico
 - è bene censire gli altri casi
- Scelta in base alle necessità applicative!
 - non alle prestazioni!



- «Shared Lock» rilasciati durante «scan»
- Righe possono muoversi fisicamente
 - Es. aggiornamento indice «Clustered»
 - Fenomeni problematici
 - Rilettura stessa riga
 - Mancata lettura di una riga

«Deadlock» (blocco critico)

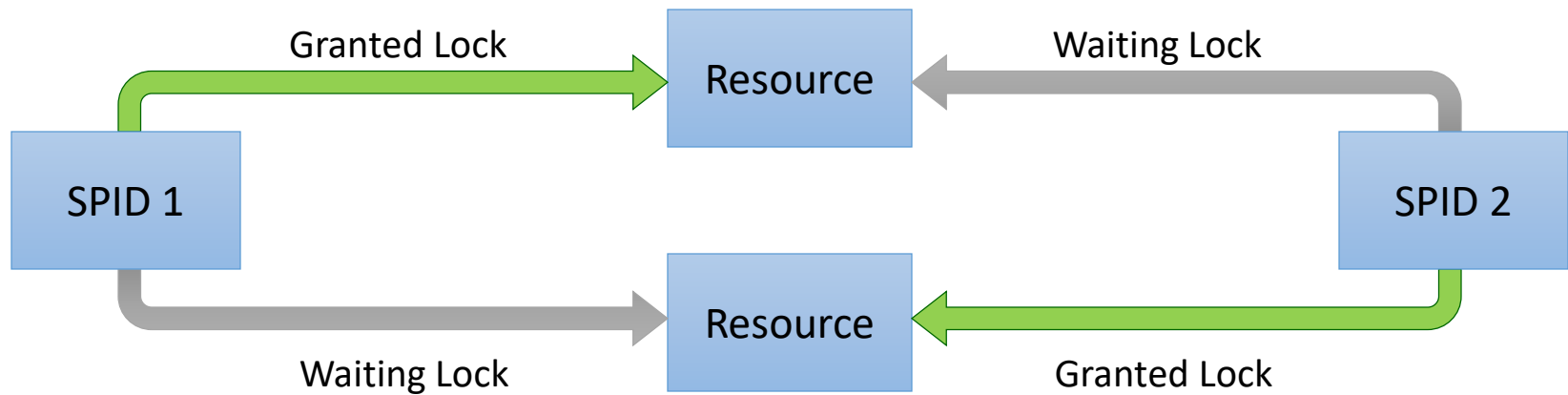


- Evento normale
 - tecniche per mitigarne la frequenza
 - non sempre si possono eliminare completamente
- Risoluzione automatica
 - controllo cicli ad intervalli regolari
 - transazione meno costosa per annullamento viene scelta come vittima («rollback»)
 - controllo più frequente quando si verificano «Deadlock»
- Applicazioni
 - ricevono errore 1205
 - devono gestire il problema, es. riprovando

«Cycle Deadlock»



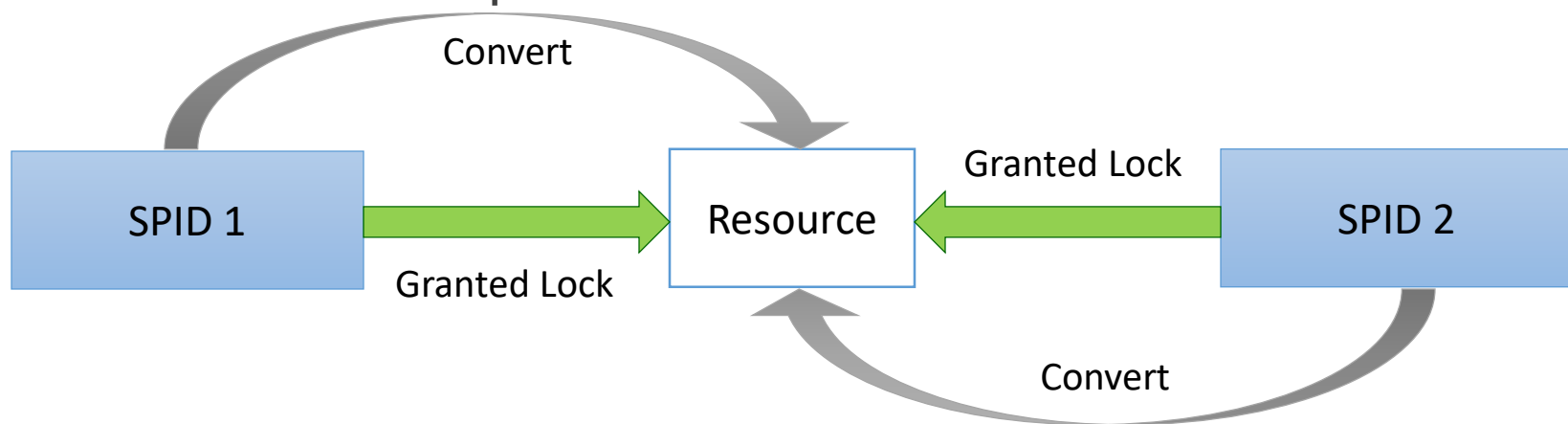
- Coinvolge più risorse
- Processi si bloccano a vicenda in modo ciclico



«Conversion Deadlock»



- Coinvolge una risorsa
- Entrambi i processi hanno un «Lock» sulla risorsa
 - tipicamente «Shared»
 - con livello di isolamento più alto di «Read Committed»
- Entrambi cercano di convertire il «Lock»
 - con uno incompatibile





- Usare livello di isolamento appropriato
- Verificare possibilità di usare livelli «Snapshot»
- Controllare gli indici
- Controllare normalizzazione della base dati
- Le transazioni dovrebbero
 - durare poco
 - non richiedere interazione utenti (ormai tutto stateless)
 - accedere alle risorse nello stesso ordine

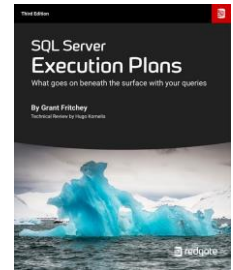


- Fondamentale!
 - Per avere una baseline
 - Per capire dove spendono tempo le query
- Analisi uso risorse
 - Numero di pagine lette (da disco e da cache)
 - Memoria utilizzata (se query massiva)
- Analisi tempi di esecuzione
 - Tempo di generazione piano esecuzione (compile time)
 - Tempo di processamento effettivo (CPU time)
 - Tempo di attesa risorse (Wait Stats)



- Fondamentale!
 - Per migliorare le prestazioni
 - Per determinare se si sono verificate regressioni
- SQL Server Management Studio
 - Visualizzazione grafica/Live
 - Confronto tra piani
- Strumenti Free di terze parti
 - SentryOne Plan Explorer
 - <https://www.sentryone.com/plan-explorer>
 - ApexSQL Plan
 - https://www.apexsql.com/sql_tools_plan.aspx

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





- Tempi di esecuzione query
 - «CPU time» e «Elapsed time»
 - SQL Server 2012 SP4, 2016 SP1, 2017, 2014 (TBD?)
- Top 10 «Wait Stats» a livello di query
 - Attenzione all'accuratezza
 - <https://www.brentozar.com/archive/2017/07/sql-2016-sp1-shows-wait-stats-execution-plans>
 - https://blogs.msdn.microsoft.com/sql_server_team/making-parallelism-waits-actionable
 - SQL Server 2016 SP1, 2017 (compatibilità 140!)
- «Trace Flags» attive
 - Include il livello: global, session o query
 - SQL Server 2012 SP4, 2016 SP1, 2017

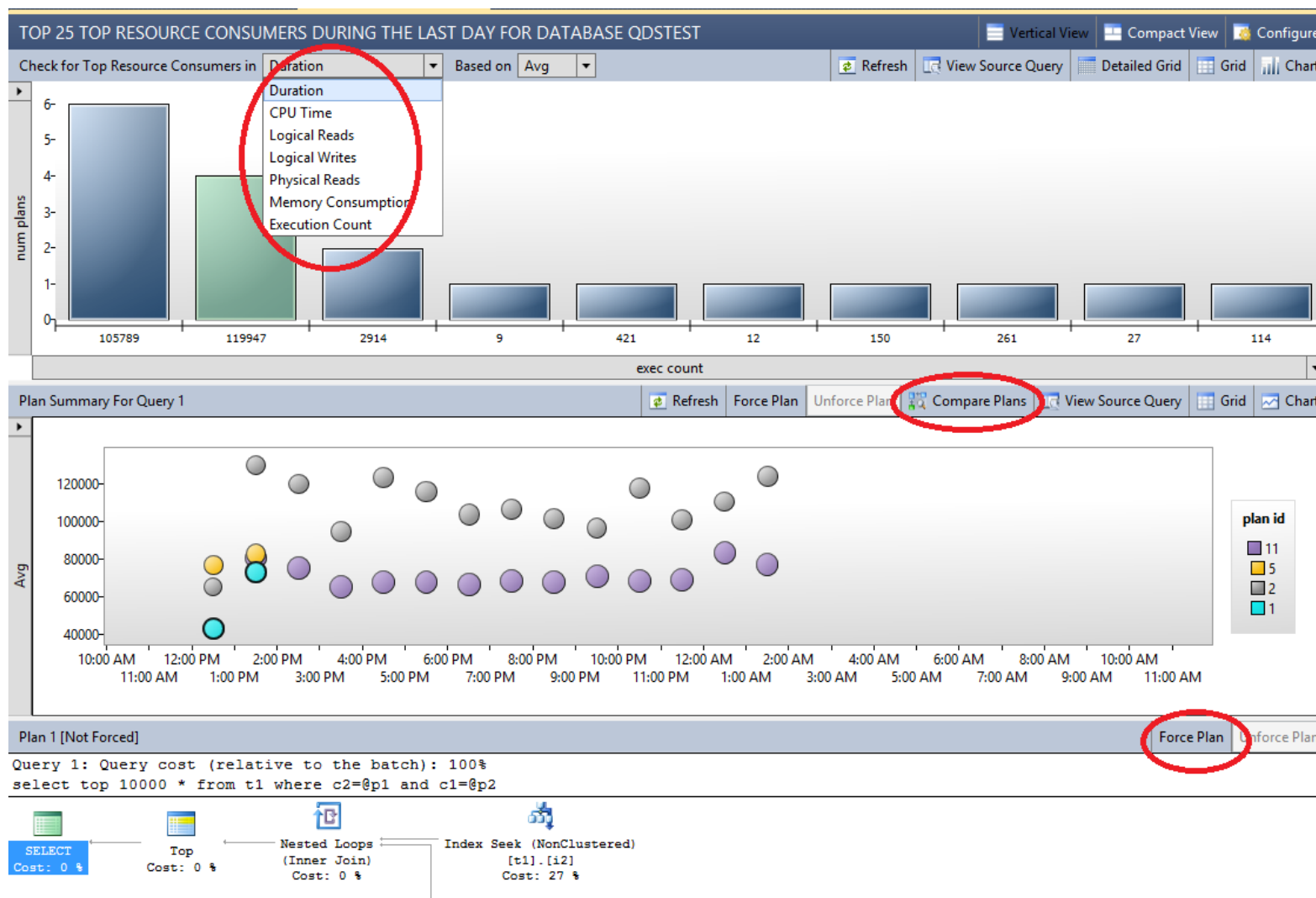


- `sys.dm_exec_query_statistics_xml(session)`
 - Piano di esecuzione con statistiche transitorie «in-flight» (es. n° righe, CPU)
 - Funziona con profilazione
 - Standard (eseguire SET STATISTICS XML ON prima di eseguire la query)
 - «lightweight statistics» (attivare Trace Flag 7412)
 - Disponibile in SQL Server 2016 SP1, 2017

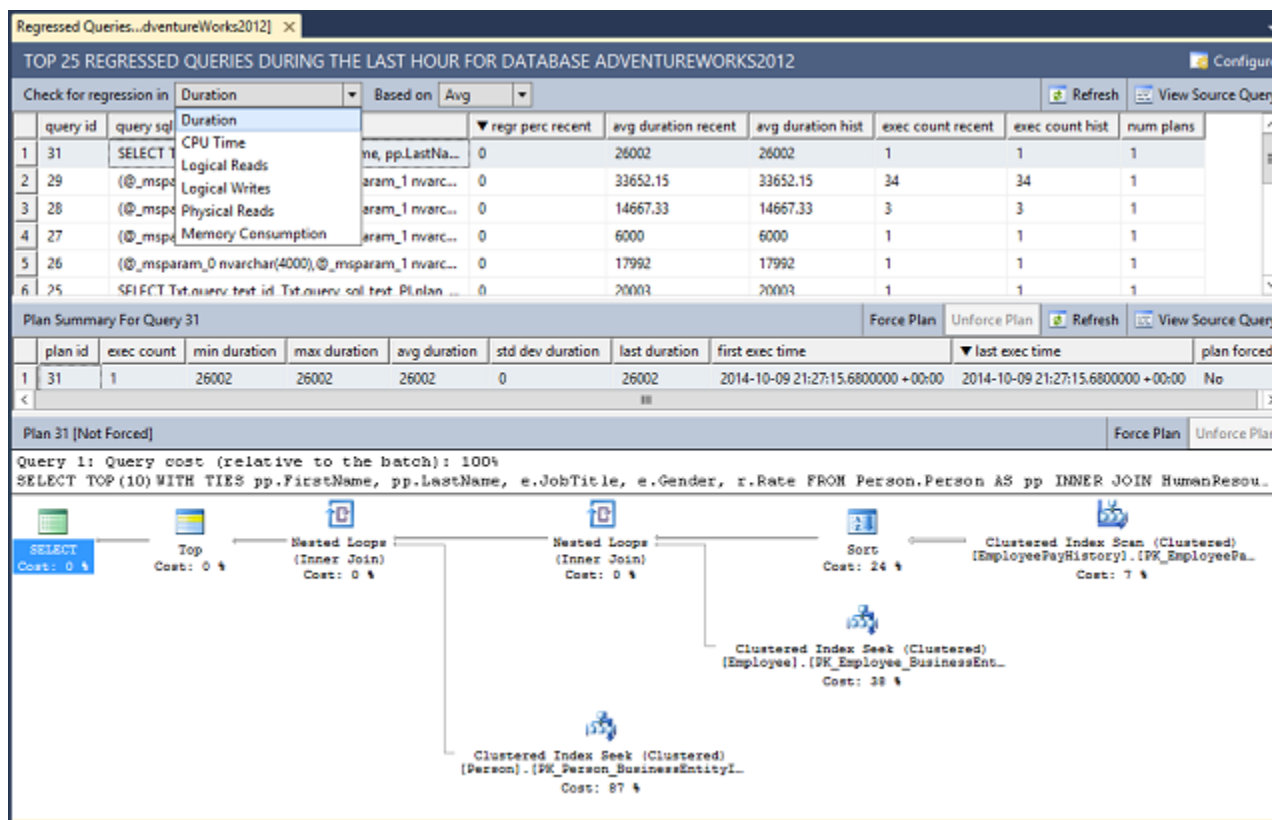


- Registra piani di esecuzione e metriche prestazioni
 - Per database
 - In memoria consolidando su disco in modalità asincrona
 - Integrato con SSMS e disponibile via DMV
 - Da SQL Server 2017: anche Wait Stats!!
- Riduce drasticamente complessità «troubleshooting»
 - Identificare «Top Query» per tempo/memoria/CPU/IO
 - Analizzare utilizzo risorse per un determinato database
 - Audit storia piani di esecuzione di una determinata query
 - Identificare regressione piani di esecuzione
 - Forzatura piano specifico da storico

QUERY STORE TOP CONSUMER



QUERY STORE TOP «REGRESSED QUERIES»



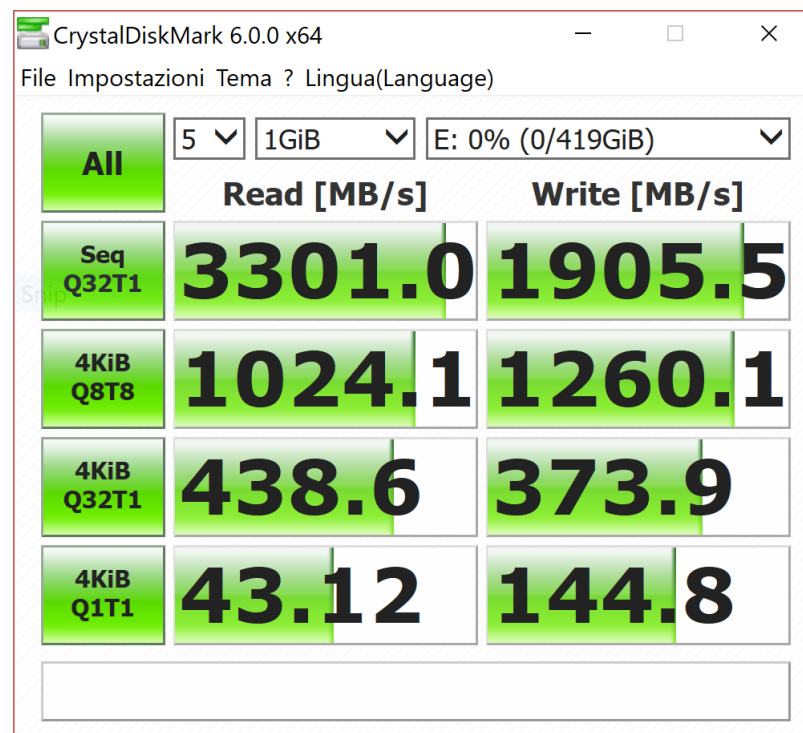
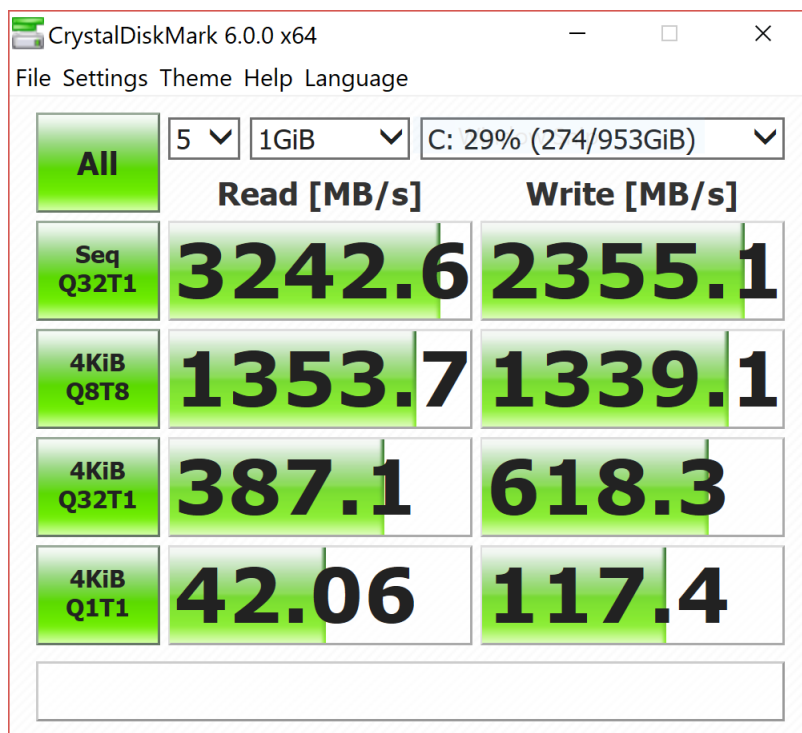


- «Wait Stats» integrate con Query Store!
 - Nuova DMV **sys.query_store_wait_stats**
 - Organizzate in categorie
 - **ALTER DATABASE ...
SET QUERY_STORE
(WAIT_STATS_CAPTURE_MODE = ON)**

Attenzione a contestualizzare sempre!



- «Come fa la query a essere lenta in produzione?»
- «Sul mio Portatile la Query è velocissima!»





- Essenziale per generare piani esecuzione efficienti, predice
 - Numero righe risultati intermedi (es. join, filtri, aggregazioni)
 - Numero righe finale
- Fino a SQL 2012 in gran parte basato su SQL 7.0
 - Abilitazione Fix QFE con Trace Flag per evitare regressioni
 - Alcuni problemi necessitavano di un ridisegno rilevante
- Introdotto in SQL Server 2014 può causare regressioni
 - Impatto su scenari specifici (es. data ordine, join)
 - Maggior parte delle query dovrebbe comunque beneficiarne

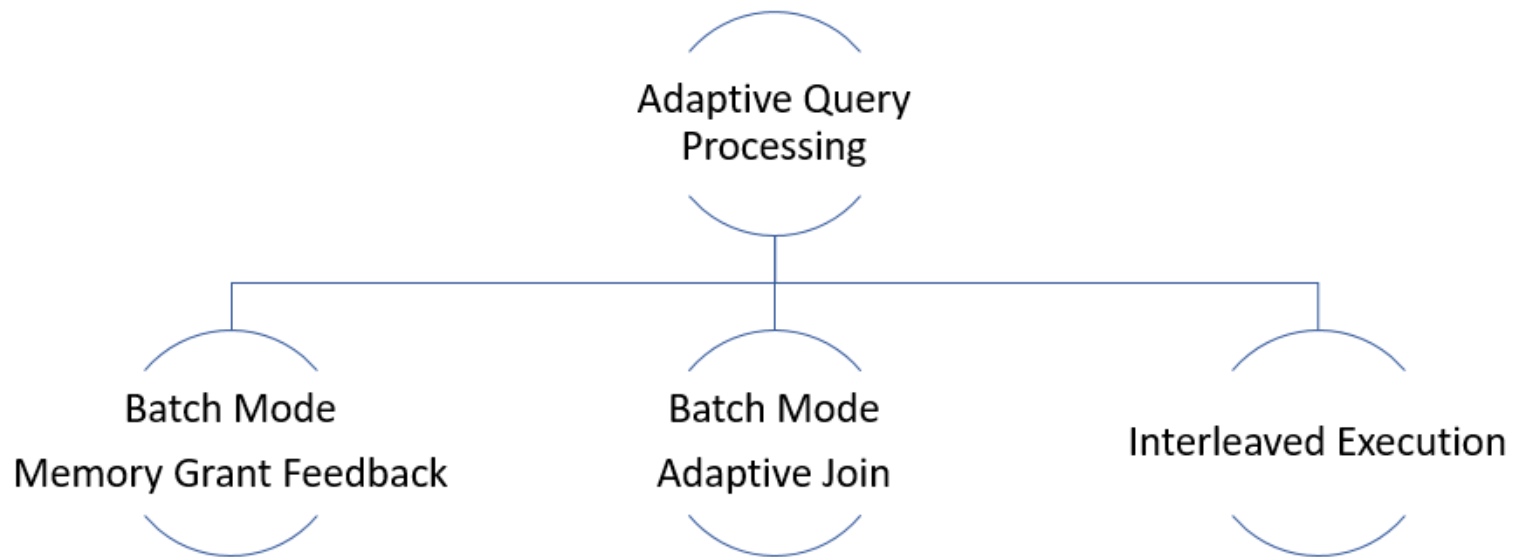


- Nessuna opzione indipendente in SQL Server 2014 ☹
 - Attivo con livello di compatibilità database a 120
 - Trace Flag specifici per singole query (troubleshooting)
- A partire da SQL Server 2016
 - ALTER DATABASE SCOPED CONFIGURATION
 - LEGACY_CARDINALITY_ESTIMATION
 - ma anche QUERY_OPTIMIZER_HOTFIXES
- A partire da SQL Server 2017
 - USE HINT('QUERY_OPTIMIZER_COMPATIBILITY_LEVEL_n')
 - <https://support.microsoft.com/en-us/help/4342424/update-to-query-optimizer-compatibility-level-n-in-use-hint-option>
- Impostare o non impostare...
 - Cambiamento con più potenzialità di regressioni da anni
 - ...ma era ora 😊 fare quanti più test possibili



- Aggiornamento database a 2016
 - Mantenendo «Database Compatibility Level» originale
- Abilitare «Query Store» per il database
 - Tutti i piani vengono congelati al livello originale
- Aspettare raccolta carico di lavoro rappresentativo
- Cambiare il livello di compatibilità a 130+
- Valutare problemi specifici di regressione con SSMS
 - Forzare piani pre-aggiornamento registrati in «Query Store»
 - Usare HINT specifico da 2017 CU10+ per singole query

«Adaptive Query Processing» in SQL 2017



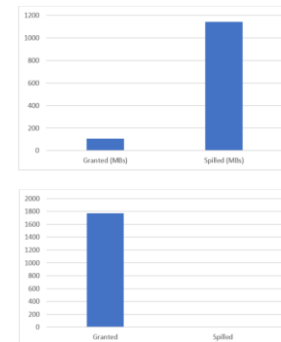


- «Excessive Grant»
 - Troppa memoria allocata rispetto a quanto utilizzata
 - Impatto: «blocking», «out-of-memory», concorrenza ridotta
- «Poor Grant»
 - Memoria allocata insufficiente con conseguente «spill» nel tempdb
 - Impatto: query lenta, eccessivo utilizzo del disco (tempdb)
- «Grant increase»
 - Incremento memoria allocata dinamicamente troppo grande
 - Impatto: instabilità del server, performance non predicibili

Batch Mode Memory Grant Feedback



- Valutazione post-esecuzione
 - Aggiorna il valore di allocazione dei piani in cache
 - Es. più memoria se «Poor», meno se «Excessive»
- Scenario sensibili a valori parametri
 - Alcune query piani con allocazione diversa
 - «memory grant feedback» si disabilita quando non stabile
 - memory_grant_feedback_loop_disabled extended event
- Memorizzazione nella cache dei piani
 - Non persistente (non salvata nel Query Store)
 - OPTION(RECOMPILE) impedisce memorizzazione e «memory grant feedback»

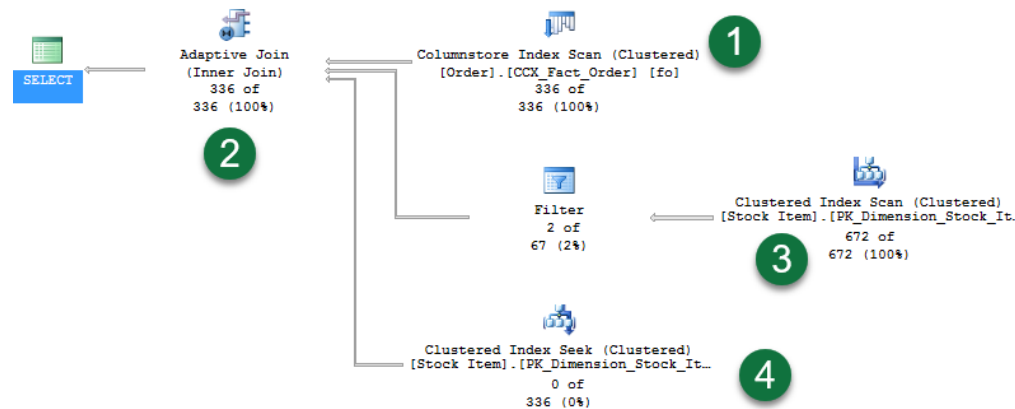


«Batch mode adaptive joins»



- Scenari join
 - Algoritmo «nested loop» migliore con piccole tabelle candidate come «build»
 - Algoritmo «hash» migliore con grandi tabelle candidate come «build»
- «Adaptive joins» rinvia la scelta dopo scansione primo

input



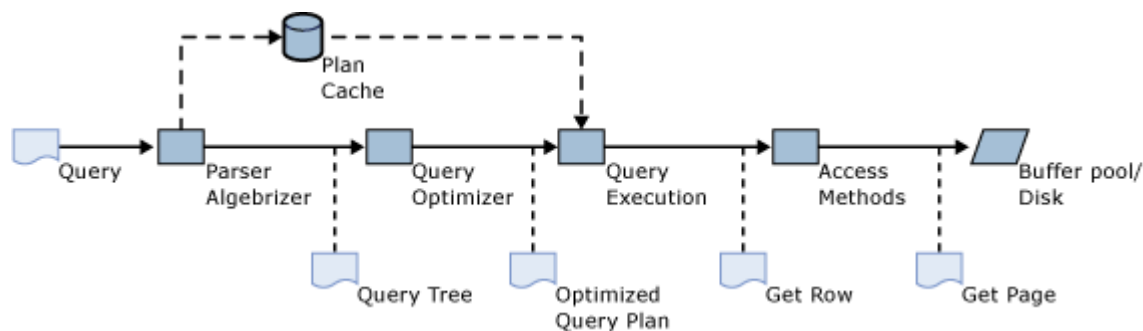


- Problema con «multi-statement table valued functions» (MSTVF)
 - SQL Server \leq 2012 ottimizza con cardinalità fissa = 1
 - SQL Server 2014 & 2016 ottimizza con cardinalità fissa = 100
- SQL Server \geq 2017
 - Ottimizzazione parte
 - Ottimizzazione viene in pausa per eseguire MSTVF (se candidata)
 - Ottimizzazione riprende con cardinalità corretta



- Altro argomento intrattabile in poco tempo
 - Vero ginepraio di dettagli implementativi ☹
 - Strettamente legato anche a implementazione schema
- In generale
 - Mai dimenticarsi che è un linguaggio dichiarativo
 - si specifica il «cosa» al posto del «come»...
 - ...lasciando spazio a continui miglioramenti del «come»!
 - Soluzioni «set based» al posto di «cursor based»
 - Oltre una certa complessità: «divide et impera»!

- Interpretazione richieste complesse
 - equivalenze logiche, individuazione contraddizioni, ...
- Ottimizzazione piano di esecuzione
 - algoritmi, parallelismo, utilizzo di indici, ...
- Riutilizzo dei piani di esecuzione
 - anche in modalità adattiva



- *"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."*
 - Martin Fowler



```
SELECT      Person.Contact.Title, Person.Contact.FirstName, Person.Contact.LastName, Sales.SalesOrderHeader.OrderDate, :
            Sales.SalesOrderHeader.DueDate, Sales.SalesOrderHeader.ShipDate, Sales.SalesOrderHeader.SalesOrd
            Sales.SalesOrderHeader.PurchaseOrderNumber, Sales.SalesOrderHeader.TotalDue, Sales.SalesOrderHea
            Production.Product.ProductNumber, Production.Product.Name, Production.ProductInventory.Quantity
FROM        Production.Product INNER JOIN
            Production.ProductInventory ON Production.Product.ProductID = Production.ProductInventory.Produc
            Sales.SalesOrderDetail ON Production.Product.ProductID = Sales.SalesOrderDetail.ProductID INNER
            Sales.SalesOrderHeader ON Sales.SalesOrderDetail.SalesOrderID = Sales.SalesOrderHeader.SalesOrde
            Sales.SalesOrderDetail.SalesOrderID = Sales.SalesOrderHeader.SalesOrderID AND
            Sales.SalesOrderDetail.SalesOrderID = Sales.SalesOrderHeader.SalesOrderID AND
            Sales.SalesOrderDetail.SalesOrderID = Sales.SalesOrderHeader.SalesOrderID INNER JOIN
            Person.Contact ON Sales.SalesOrderHeader.ContactID = Person.Contact.ContactID AND
            Sales.SalesOrderHeader.ContactID = Person.Contact.ContactID AND Sales.SalesOrderHeader.ContactID
            Sales.SalesOrderHeader.ContactID = Person.Contact.ContactID INNER JOIN
            Sales.SalesTerritory ON Sales.SalesOrderHeader.TerritoryID = Sales.SalesTerritory.TerritoryID
WHERE       (Sales.SalesTerritory.Name = N'Northwest') AND (Sales.SalesTerritory.[Group] = N'North America')
```

- Tutto sommato è abbastanza banale...
- ...ma solo a vederla scoraggia chi deve metterci mano!

Meglio. Molto meglio 😊



```
SELECT [pc].Title,
       [pc].FirstName,
       [pc].LastName,
       [soh].OrderDate,
       [soh].Status,
       [soh].DueDate,
       [soh].ShipDate,
       [soh].SalesOrderNumber,
       [soh].PurchaseOrderNumber,
       [soh].TotalDue,
       [soh].Comment,
       [pp].ProductNumber,
       [pp].Name,
       [pi].Quantity
FROM
    Production.Product [pp]
INNER JOIN
    Production.ProductInventory [pi] ON [pp].ProductID = [pi].ProductID
INNER JOIN
    Sales.SalesOrderDetail [sod] ON [pp].ProductID = [sod].ProductID
INNER JOIN
    Sales.SalesOrderHeader [soh] ON [sod].SalesOrderID = [soh].SalesOrderID
                                   AND [sod].SalesOrderID = [soh].SalesOrderID
INNER JOIN
    Person.Contact [pc] ON [soh].ContactID = [pc].ContactID
                        AND [soh].ContactID = [pc].ContactID
INNER JOIN
    Sales.SalesTerritory [st] ON [soh].TerritoryID = [st].TerritoryID
WHERE
    [st].Name = N'Northwest'
AND
    [st].[Group] = N'North America'
```

- Stessa query di prima
 - Riformattata
 - Con qualche alias



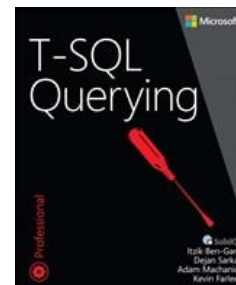
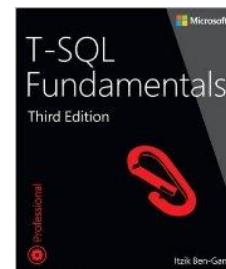
- Red Gate SQL Prompt
 - Gold Standard ma €325/utente
 - <https://www.red-gate.com/products/sql-development/sql-prompt/index>
- Apex SQL Refactor
 - Anche formattazione, gratuito
 - https://www.apexsql.com/sql_tools_complete.aspx
- Poor Man's T-SQL Formatter
 - Open Source (github)
 - <http://architectshack.com/PoorMansTSqlFormatter.ashx>

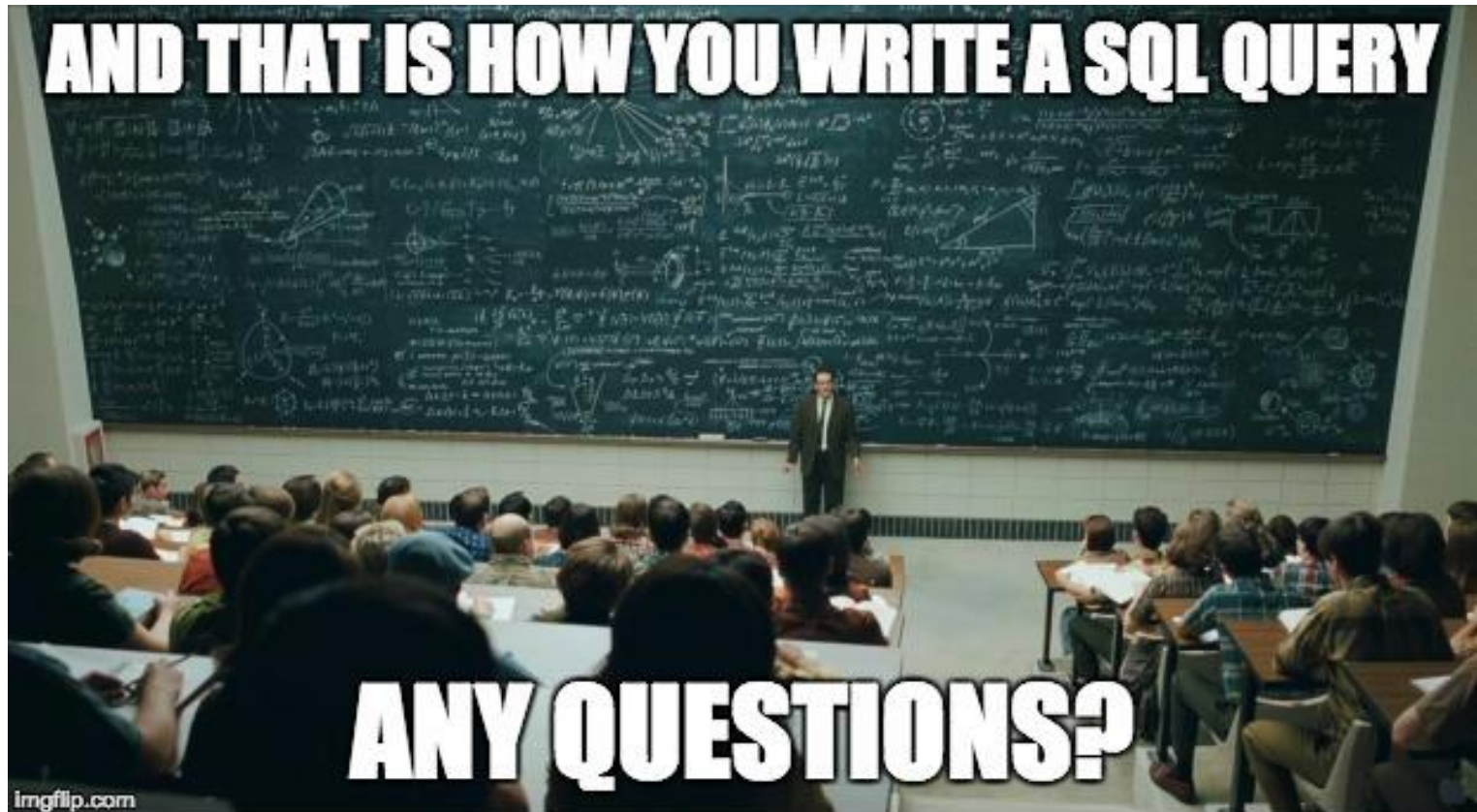
SELECT...



- Potenziali problemi applicativi con aggiunta colonne
- Rowset più grande del dovuto
- Può prevenire copertura query tramite indice
- Può favorire table scan rispetto a lookup
- In generale evitare di usarlo
 - Enumerare tutte le colonne non cambia nulla 😊
 - Eccezione: «Sparse Column», EXISTS()

- <https://sqlperformance.com>
 - Specialmente articoli di Paul White
- <http://sqlblog.com>
 - Moltissimi autori
- T-SQL Fundamentals Third Edition
 - <https://www.microsoftpressstore.com/store/t-sql-fundamentals-9781509302000>
- T-SQL Querying
 - <https://www.microsoftpressstore.com/store/t-sql-querying-9780735685048>





Gianluca Hotz



www.ugiss.org



ghotz@ugiss.org



[@glhotz](https://twitter.com/glhotz)