Microsoft

SQL Server’s Path Toward an Intelligent Database

SQL Intersection June 2019

Pedro Lopes

Intelligent Database

Anxiety-free upgrades

- Database compatibility level regression protection
- Automatic plan regression correction
- Query Tuning Assistant

Predictable performance

- Adaptive Query Processing
- Intelligent Query Processing
- Resource governor
- Guaranteed resources in SQL DB
- Auto-Soft NUMA

Management-by-default

- Automatic Indexing in SQL DB
- Flexible Scaling
- Integrity Checking
- Intelligent Insights
- Lightweight Query Profiling-enabled troubleshooting

Security in-depth 24x7

- Advanced Threat Detection
- Data Classification
- Vulnerability Assessment

Adapts to the constantly changing world of businesses and data

Intelligent Database

Anxiety-free upgrades

- Database compatibility level regression protection
- Automatic plan regression correction
- Query Tuning Assistant

Predictable performance

- Adaptive Query Processing
- Intelligent Query Processing
- Resource governor
- Guaranteed resources in SQL DB
- Auto-Soft NUMA

Management-by-default

- Automatic Indexing in SQL DB
- Flexible Scaling
- Integrity Checking
- Intelligent Insights
- Lightweight Query Profiling-enabled troubleshooting

Security in-depth 24x7

- Advanced Threat Detection
- Data Classification
- Vulnerability Assessment

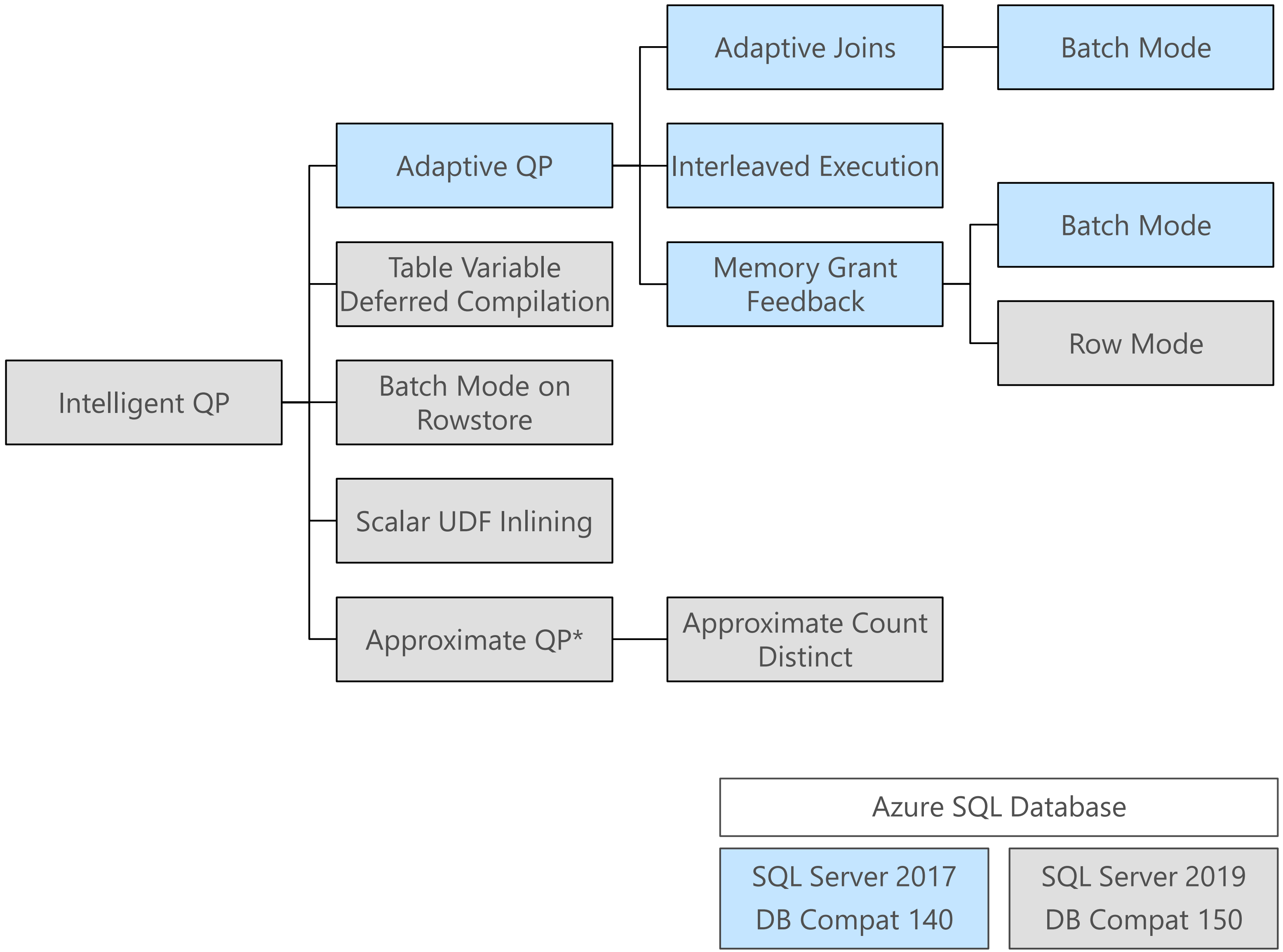
Adapts to the constantly changing world of businesses and data

The Intelligent Query Processing principles

- Available by default on the latest database compatibility level setting
- Delivering broad impact that improves the performance of existing workloads with minimal implementation effort
- Critical parallel workloads improve when running at scale, while remaining adaptive

Intelligent Query Processing

The **intelligent query processing** feature family includes features that automatically improve workload performance



Up to 130 database compatibility level

Interleaved Execution for MSTVFs

Multi-statement table-valued functions (MSTVFs) are treated as a black box by QP and SQL Server uses a fixed optimization guess.

Pre SQL 2017

Optimize → Execute

100 rows guessed for MSTVFs

Performance issues if # rows is much larger

140 database compatibility level

Interleaved Execution for MSTVFs

Multi-statement table-valued functions (MSTVFs) are treated as a black box by QP and SQL Server uses a fixed optimization guess.

Pre SQL 2017

Optimize → Execute

100 rows guessed for MSTVFs

Performance issues if # rows is much larger

Interleaved Execution will materialize and use row counts for MSTVFs.

Downstream operations will benefit from the corrected MSTVF cardinality estimate.

SQL 2017+

Optimize → Execute → Optimize → Execute

MSTVF identified

Execute MSTVF

500k rows assumed from 1st execution

Good performance!

140 and 150 database compatibility level

Memory Grant Feedback (MGF)

Queries may spill to disk or take too much memory based on poor cardinality estimates. Memory misestimations result in spills, and overestimations hurt concurrency

MGF will adjust memory grants based on execution feedback

First Execution

Adjusting

Stable

Disabled

Batch Mode in 140, Row Mode in 150

MGF will remove spills and improve concurrency for repeating queries

- Spills to disk → MGF corrects grant misestimations
- Excessive memory grant → MGF corrects wasted memory, improves concurrency

140 database compatibility level

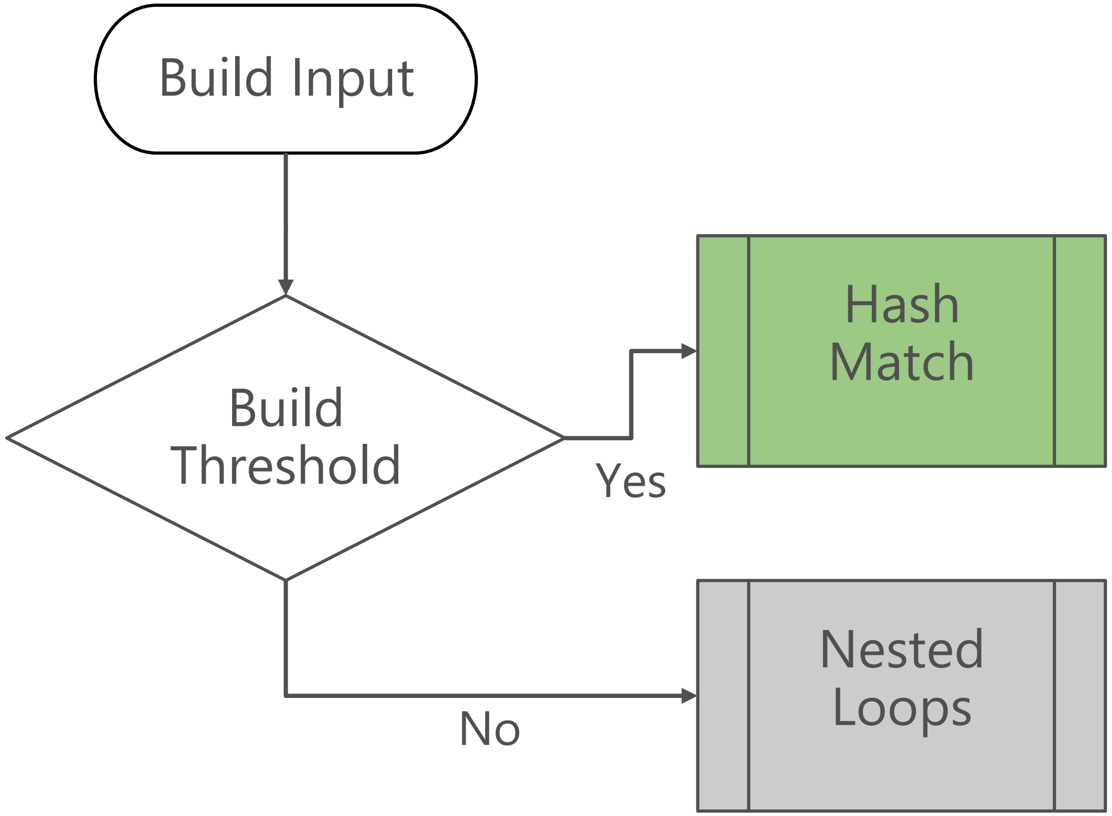
Batch Mode Adaptive Joins (AJ)

If cardinality estimates are skewed, we may choose an inappropriate join algorithm.

AJ will defer the choice of Hash Match or Nested Loops join until after the first join input has been scanned.

Adaptive Buffer is used up to the point where it’s needed as the Build Table for HJ, or Outer Table for NLJ – Threshold is dynamic

AJ uses Nested Loops for small inputs, Hash Match for large inputs.



Up to 140 database compatibility level

Table Variable Deferred Compilation

Legacy behavior

Area	Temporary Tables	Table Variables
Manual stats creation and update	Yes	No
Indexes	Yes	Only inline index definitions allowed.
Constraints	Yes	Only PK, uniqueness and check constraints.
Automatic stats creation	Yes	No
Creating and using a temporary object in a single batch	Compilation of a statement that references a temp table that doesn't exist is deferred until the first execution of the statement	A statement that references a table variable is compiled along with all other statements before any statement that populates the TV is executed, so compilation sees it as "1".

150 database compatibility level

Table Variable Deferred Compilation

Azure SQL Database and SQL Server 2019 behavior

Area	Temporary Tables	Table Variables
Manual stats creation / update	Yes	No
Indexes	Yes	Only inline index definitions allowed.
Constraints	Yes	Only PK, uniqueness and check constraints.
Automatic stats creation	Yes	No
Creating and using a temporary object in a single batch	Compilation of a statement that references a temp table that doesn't exist is deferred until the first execution of the statement	Compilation of a statement that references a table variable that doesn't exist is deferred until the first execution of the statement

APPROX_COUNT_DISTINCT
When approximate is good enough…

Provides approximate COUNT DISTINCT for big data scenarios with the benefit of high performance and a **(very) low memory** footprint.

Dashboard scenarios and trend analysis against big data sets with many distinct values (for example, distinct orders counts over a time period) – and many concurrent users where exact values are not necessary.

Data science big data set exploration. Need to understand data distributions quickly and exact values are not paramount.

Not banking applications or anywhere an exact value is required!

Batch Mode and Columnstore

Since SQL Server 2012 we've bound these two features together

Columnstore indexes

I/O

- Access only the data in columns that you need
- Effective compression over traditional rowstore

Batch Mode

CPU

- Allows query operators to process data more efficiently by working on a batch of rows at a time
- Built for analytical workload scale

150 database compatibility level

Batch Mode on Rowstore

Sometimes Columnstore isn't an option:

- OLTP-sensitive workloads
- Vendor support
- Columnstore interoperability limitations

Now get analytical processing CPU-benefits **without Columnstore indexes.**

Batch mode on rowstore supports:

- On-disk heaps and B-tree indexes and existing batch-capable operators (**new scan operator** can evaluate batch mode bitmap filters)
- Existing batch mode operators

Batch Mode on Rowstore candidate workloads

A significant part of the workload consists of analytical queries **AND**

The workload is CPU bound **AND**

- Creating a columnstore index adds too much overhead to the transactional part of your workload **OR**
- Creating a columnstore index is not feasible because your application depends on a feature that is not yet supported with columnstore indexes **OR**
- You depend on a feature not supported with columnstore (for example, triggers)

T-SQL Scalar User-Defined Functions (UDFs)

User-Defined Functions that are implemented in Transact-SQL and return a single data value are referred to as **T-SQL Scalar User-Defined Functions**

T-SQL UDFs are an elegant way to achieve code reuse and modularity across SQL queries

Some computations (such as complex business rules) are easier to express in imperative UDF form

UDFs help in building up complex logic without requiring expertise in writing complex SQL queries

T-SQL Scalar UDF performance issues!

Iterative invocation: Invoked once per qualifying row. Repeated context switching – and even worse for UDFs that have T-SQL queries that access data

Lack of costing: Scalar operators are not costed (realistically)

Interpreted execution: Each statement itself is compiled, and the compiled plan is cached. Although this caching strategy saves some time as it avoids recompilations, each statement executes in isolation. No cross-statement optimizations are carried out.

Serial execution: SQL Server does not allow intra-query parallelism in queries that invoke Scalar UDFs. In other words, Scalar UDFs are parallelism inhibitors.

150 database compatibility level

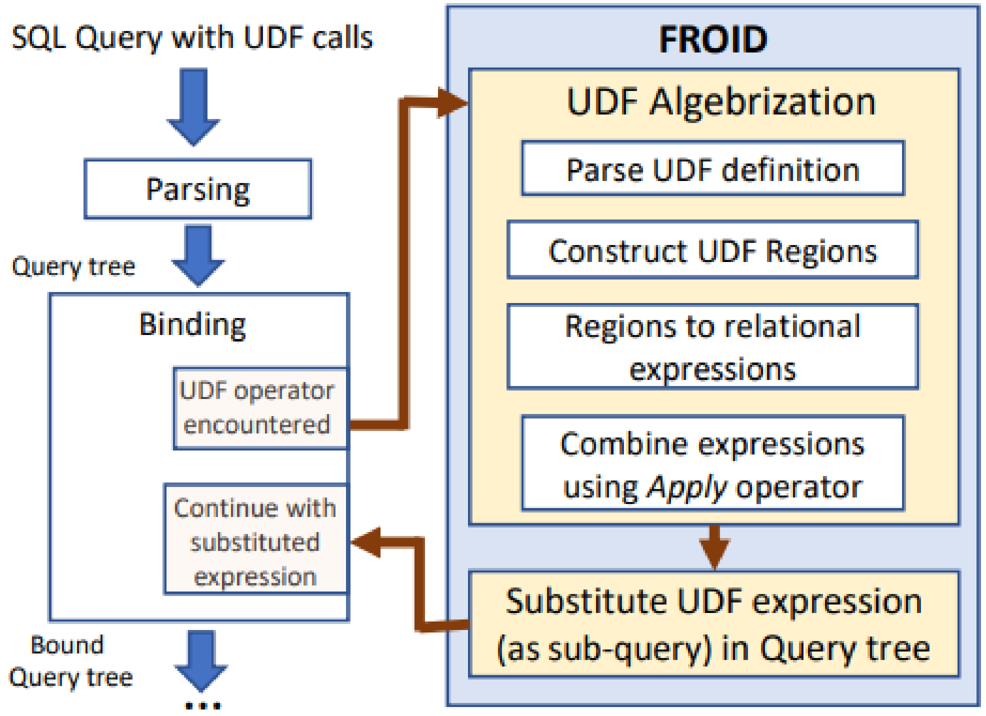
T-SQL Scalar UDF Inlining

Enable the benefits of UDFs without the performance penalty!

- Goal of the Scalar UDF Inlining feature is to improve performance for queries that invoke scalar UDFs where UDF execution is the main bottleneck

Before SQL 2019/DB Compat 150:

- Using query rewriting techniques, UDFs are transformed into equivalent relational expressions that are "inlined" into the calling query



Source: "Froid: Optimization of Imperative Programs in a Relational Database"

T-SQL Scalar UDF Inlining

Table 1: Relational algebraic expressions for imperative statements (using standard T-SQL notation from [33])

Imperative Statement (T-SQL)	**Relational expression (T-SQL)**
DECLARE {@*var* *data_type* [= *expr*]}[, . . . *n*];	SELECT {*expr*|*null* AS *var*}[, . . . *n*];
SET {@*var* = *expr*}[, . . . *n*];	SELECT {*expr* AS *var*}[, . . . *n*];
SELECT {@*var*1 = *prj_expr*1}[, . . . *n*] FROM *sql_expr*;	{SELECT *prj_expr*1 AS *var*1 FROM *sql_expr*}; [, . . . *n*]
IF (*pred_expr*) {*t_stmt*; [. . . *n*]} ELSE {*f_stmt*; [, . . . *n*]}	SELECT CASE WHEN *pred_expr* THEN 1 ELSE 0 END AS *pred_val*; {SELECT CASE WHEN *pred_val* = 1 THEN *t_stmt* ELSE *f_stmt*; }[. . . *n*]
RETURN *expr*;	SELECT *expr* AS *returnVal*;

Source: “Froid: Optimization of Imperative Programs in a Relational Database”

Scalar UDF Inlining non-starters

Non-inlineable constructs:

- Invoking any intrinsic function that is either time-dependent (such as GETDATE()) or has side effects (such as NEWSEQUENTIALID())
- Referencing table variables or table-valued parameters
- Referencing scalar UDF call in its GROUP BY clause
- Natively compiled (interop is supported)
- Used in a computed column or a check constraint definition – we’ve received lots of feedback on this scenario for this one
- References user-defined types
- Used in a partition function

What's next?

The features we saw today are in public preview – and we want your feedback!

We continue working on intelligent query processing features as we speak – share your scenarios with us!

Please email IntelligentQP@microsoft.com

Learn more

Download and try SQL Server 2019	https://aka.ms/ss19
Check out these great data-related demos	https://aka.ms/DataDemos https://aka.ms/IQPDemos
Continue learning with our new book	https://aka.ms/LearnTSQLQuerying
One shortcut to rule them all!	https://aka.ms/SQLShortcuts