Query Compilation and Execution

Introduction

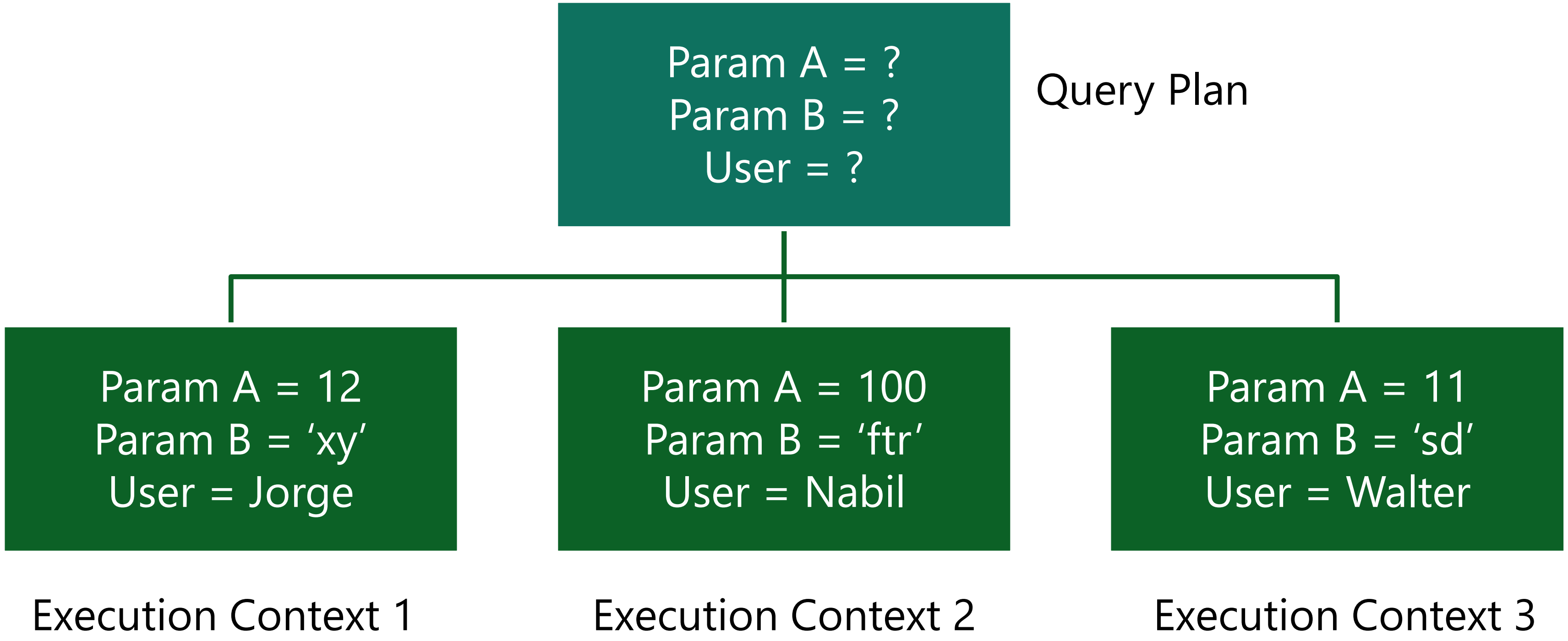
Query Compilation

- Query compilation is the process of choosing a good enough execution plan that the Optimizer has to act in the short amount of time
  - Parse a query into a tree representation
  - Normalize and validate the query
  - Evaluate possible query plans
  - Pick a good enough plan, based on cost

Query Execution

- Query execution is the process of executing the plan that is created during query compilation and optimization
  - Not necessarily performed directly after query compilation
  - May trigger a query recompile
  - Compilation versus recompilation
  - Query recompiles may occur because of correctness-related reasons or plan optimality related reasons

Query Plans and Execution Contexts



Param A = ?
Param B = ?
User = ?

Query Plan

Param A = 12
Param B = ‘xy’
User = Jorge

Execution Context 1

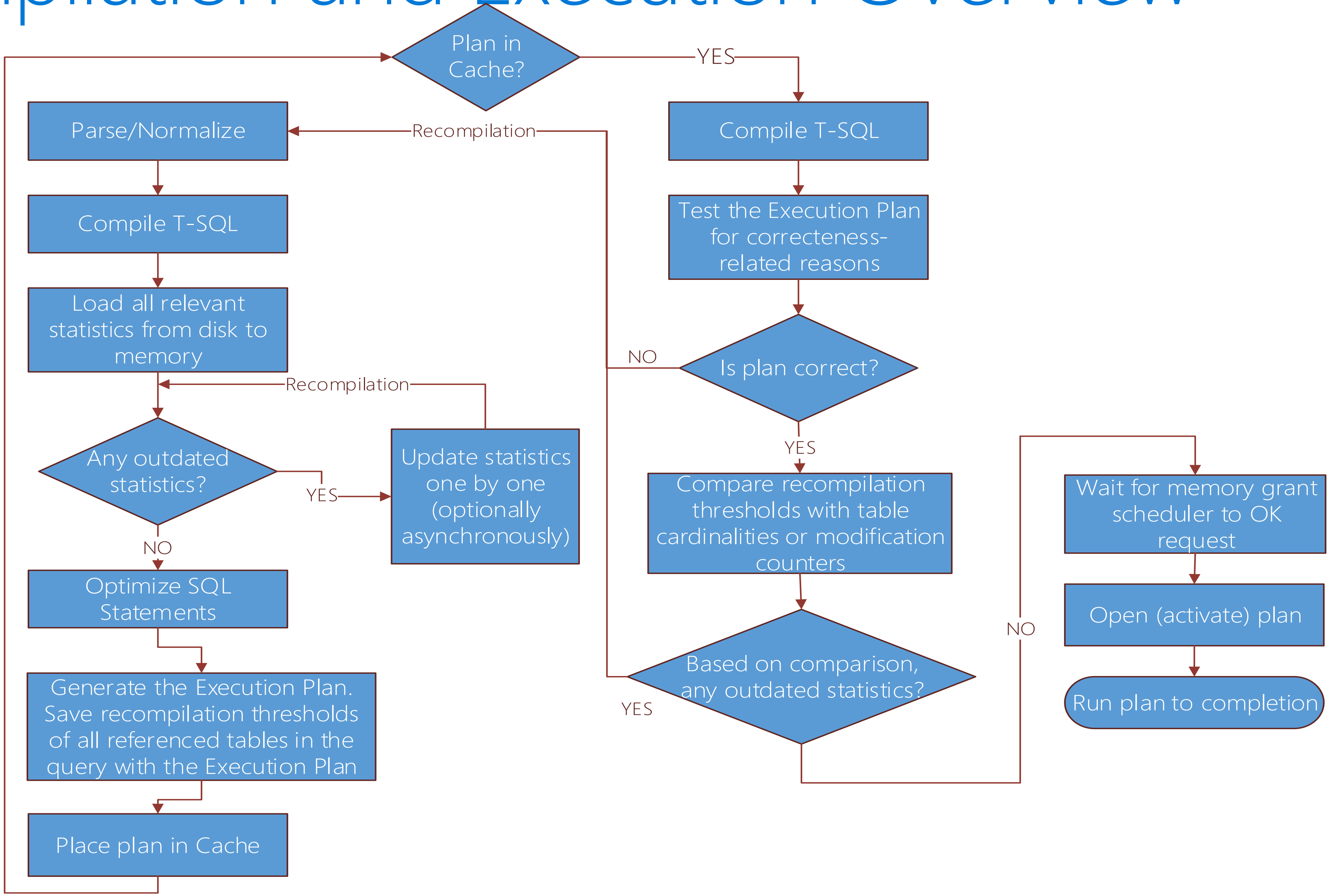
Param A = 100
Param B = ‘ftr’
User = Nabil

Execution Context 2

Param A = 11
Param B = ‘sd’
User = Walter

Execution Context 3

Compilation and Execution Overview



Plan in Cache?

Parse/Normalize

Compile T-SQL

Load all relevant statistics from disk to memory

Any outdated statistics?

Update statistics one by one (optionally asynchronously)

Optimize SQL Statements

Generate the Execution Plan. Save recompilation thresholds of all referenced tables in the query with the Execution Plan

Place plan in Cache

Compile T-SQL

Test the Execution Plan for correcteness-related reasons

Is plan correct?

Compare recompilation thresholds with table cardinalities or modification counters

Based on comparison, any outdated statistics?

Wait for memory grant scheduler to OK request

Open (activate) plan

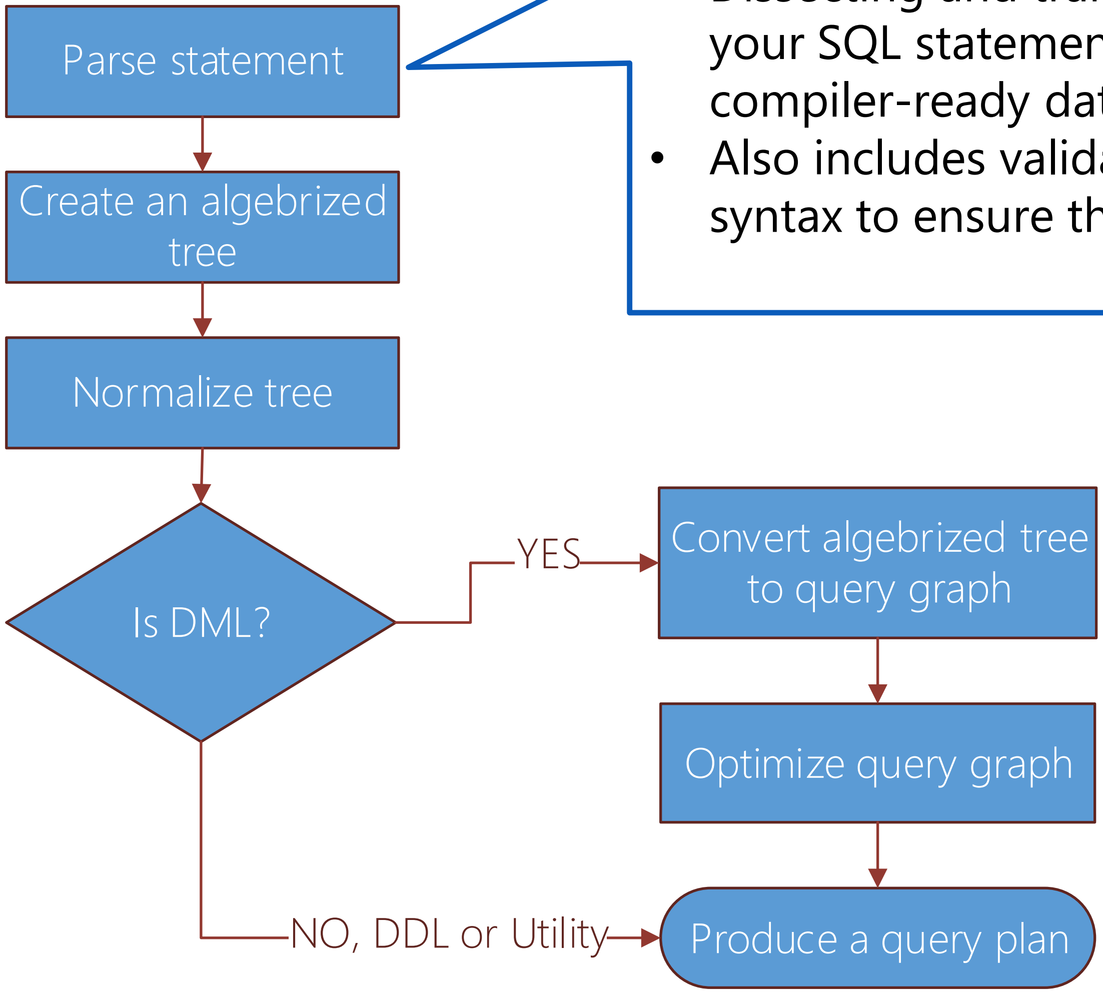
Run plan to completion

Query Processing

Query Plan Generation

First Stage - Compilation

- Dissecting and transforming your SQL statements into compiler-ready data structures
- Also includes validation of the syntax to ensure that it is legal



Parse statement

Create an algebrized tree

Normalize tree

Is DML?

YES

Convert algebrized tree to query graph

Optimize query graph

NO, DDL or Utility

Produce a query plan

- Object binding, which includes verifying that the tables and columns exist, and expanding the views
- Loading the metadata information
- Syntax based optimizations

Second Stage – Optimization

Stage 1 - **Trivial Plan**

- Non-cost-based optimizer
  - Statistics are loaded and validated at this stage
  - This step generates plans for which there are no alternatives that require a cost-based decision
- Example: INSERT statement with a VALUE has only one possible plan

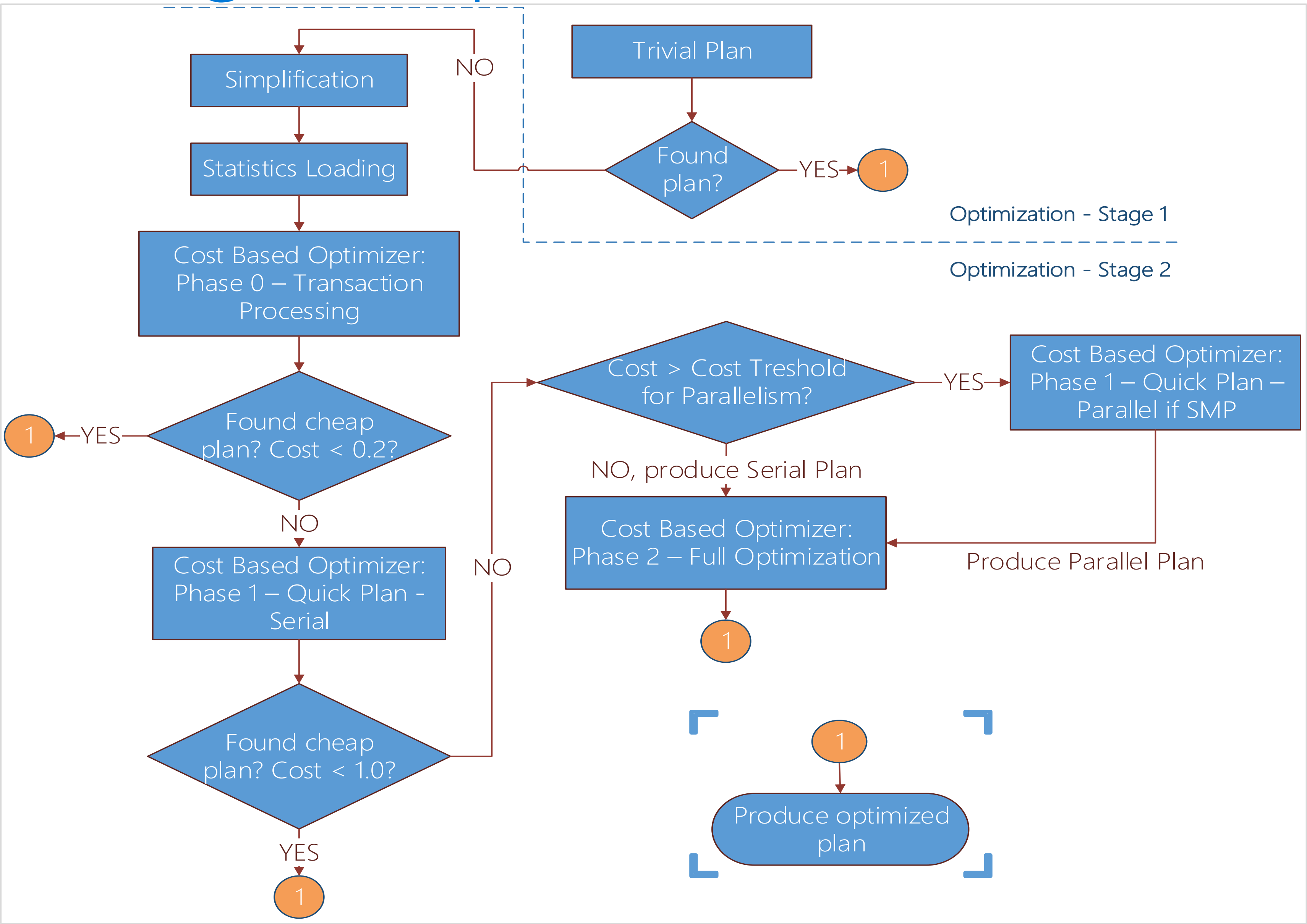
Stage 2 - **Simplification**

- Cost-based optimizer if previously unsuccessful
- Has three phases:
  - Phase 0 - Transaction Processing
  - Phase 1 - Quick Plan
  - Phase 2 - Full Optimization

Second Stage – Optimization (Continued)

- Query Optimizer may use various inputs (for example, stats, parameterized values) to reason about density/selectivity and cardinality
- Evaluates the cost of various plan alternatives and gives you the best one based on the provided information
- If it gets it wrong – you get what is perceived as *inefficient plan*
- Sources of inefficiency:
  - Bad cardinality estimation?
    - Look at plan
  - Parameter-sensitive plans?
    - Dynamic un-parameterized SQL Server?
  - Bad physical database design?
    - Missing indexes?

Second Stage – Optimization Overview


Trivial Plan
Found plan?
YES
1
NO
Simplification
Statistics Loading
Optimization - Stage 1
Optimization - Stage 2
Cost Based Optimizer: Phase 0 – Transaction Processing
Found cheap plan? Cost < 0.2?
YES
1
NO
Cost Based Optimizer: Phase 1 – Quick Plan - Serial
Found cheap plan? Cost < 1.0?
YES
1
NO
Cost > Cost Treshold for Parallelism?
YES
Cost Based Optimizer: Phase 1 – Quick Plan – Parallel if SMP
NO, produce Serial Plan
Cost Based Optimizer: Phase 2 – Full Optimization
Produce Parallel Plan
1
1
Produce optimized plan

Query Processing

Statistics

Recompilation Threshold (RT)

The RT is a mechanism used by SQL Server to determine if a table has changed enough to force a recompile of a query plan to determine if a more efficient plan is available for the current data distribution

The *threshold crossing test* is performed to decide whether to recompile a query plan:

- `| colmodctr(current) – colmodctr(snapshot) | >= RT`

If there are no statistics, or nothing is *interesting,* then table cardinality is used:

- `| cardinality(current) – cardinality(snapshot) | >= RT`

Recompilation Threshold Calculation

Permanent table

- If n <= 500, RT = 500.
- If n > 500, RT = 500 + 0.20 * n

Temporary table

- If n < 6, RT = 6.
- If 6 <= n <= 500, RT = 500
- If n > 500, RT = 500 + 0.20 * n

Table variable

- RT does not exist

With TF2371:

- RT when colmodctr > SQRT(table cardinality*1000)

n = table rows (cardinality) or colmodctr of the leading column of the statistics object

Query Processing

Optimizations

What QP Searches and Considers When Optimizing?

- Join reordering
- Outer joins
- Sub-queries
- Aggregation
- Stars and snowflakes
- Join elimination
- Materialized views
- Index plans
- Update plans

- Halloween protection
- Empty tab simplification (Integrity constraints)
- Partitioned tables
- Parallelism
- Remote queries
- Correlation elimination
- Sub-query elimination

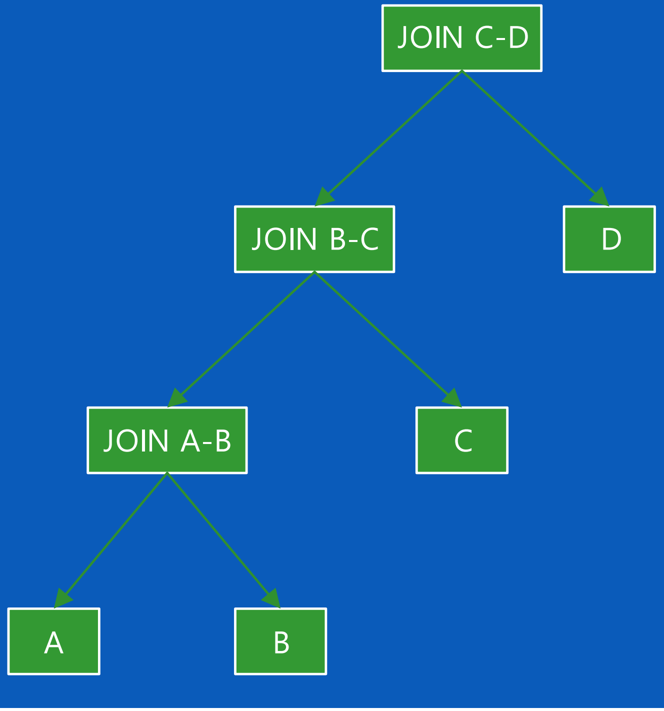
Join Reordering

- SQL Server join-paths between tables may differ from the actual written Transact-SQL form, known as **join reordering**
- The goal is to reduce the row-count of each join as early as possible

Join Reordering (Continued)

- JOIN ordering uses:
  - Heuristics
  - Statistics
  - Indexing
- Queries can be represented as trees, in accordance with the JOINed tables:
  - Linear trees

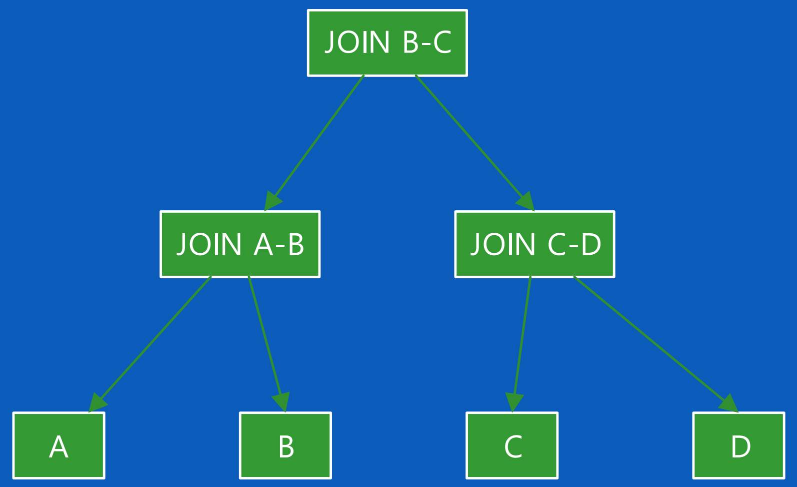
```
SELECT *
FROM tA JOIN tB ON tA.c1 = tB.c1
JOIN tC ON tB.c1 = tC.c1
JOIN tD ON tC.c1 = tD.c1
```



Join Reordering (Continued)

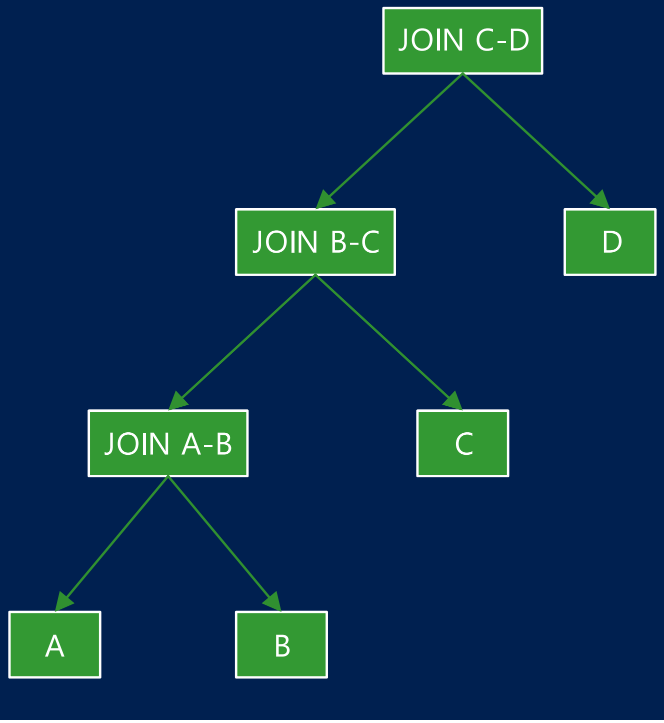
- JOIN ordering uses:
    - Heuristics
    - Statistics
    - Indexing
- Queries can be represented as trees, in accordance with the JOINed tables:
    - Linear trees
    - Bushy trees

```
SELECT *
FROM (tA JOIN tB ON tA.c1 = tB.c1)
JOIN (tC JOIN tD ON tC.c1 = tD.c1)
ON tB.c1 = tC.c1
```

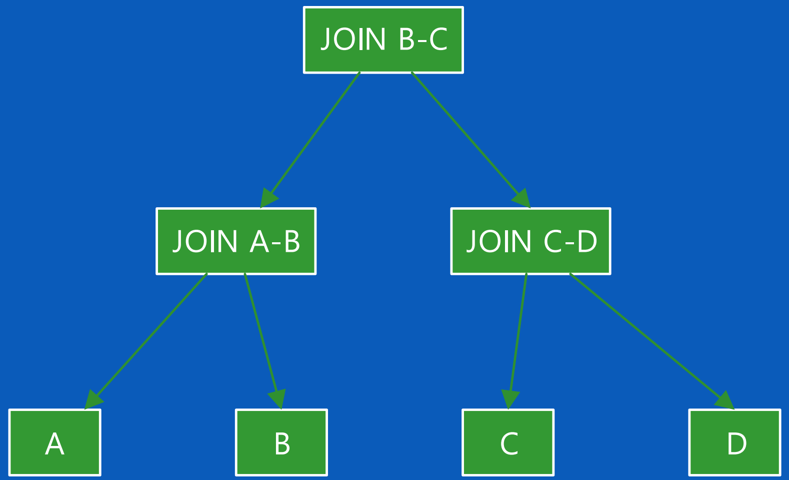


Join Reordering (Continued)

```
SELECT *
FROM tA JOIN tB ON tA.c1 = tB.c1
JOIN tC ON tB.c1 = tC.c1
JOIN tD ON tC.c1 = tD.c1
```

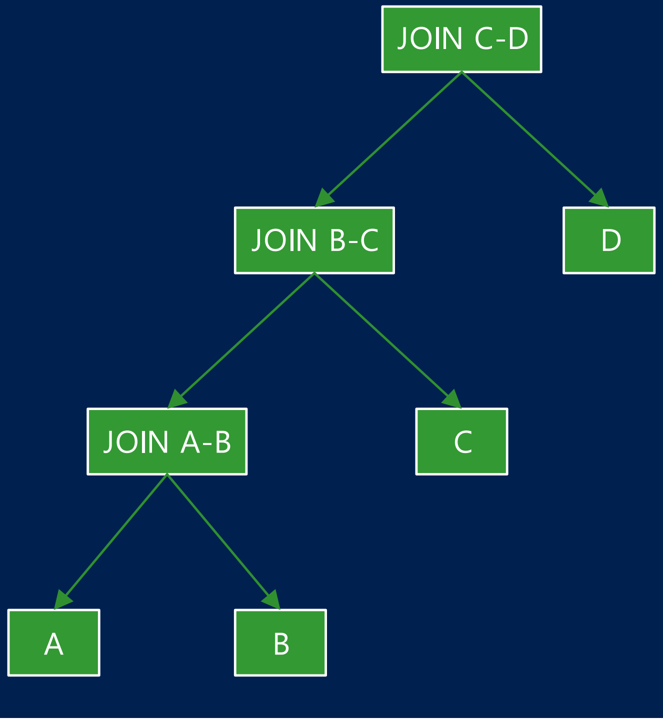


```
SELECT *
FROM (tA JOIN tB ON tA.c1 = tB.c1)
JOIN (tC JOIN tD ON tC.c1 = tD.c1)
ON tB.c1 = tC.c1
```



Join Reordering – Permutations

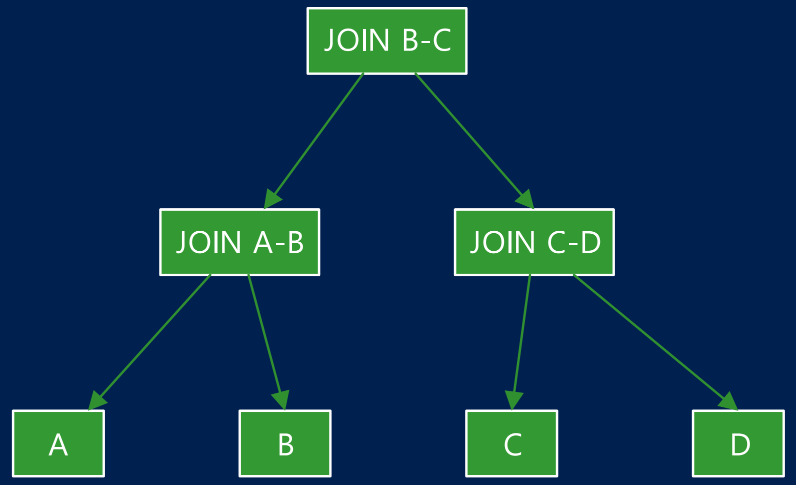
```
SELECT *
FROM tA JOIN tB ON tA.c1 = tB.c1
JOIN tC ON tB.c1 = tC.c1
JOIN tD ON tC.c1 = tD.c1
```



Nr. Tables	Linear Tree
1	1
2	2
3	6
4	**24**
5	120
6	720
7	5,040
8	40,320

Join Reordering – Permutations (Continued)

```
SELECT *
FROM (tA JOIN tB ON tA.c1 = tB.c1)
JOIN (tC JOIN tD ON tC.c1 = tD.c1)
ON tB.c1 = tC.c1
```



Nr. Tables	Bushy Tree
1	1
2	2
3	12
4	**120**
5	1,680
6	30,240
7	665,280
8	17,297,280

Query Parallelism

- Used by SQL Server to reduce the run-time of a query
- CPU cost is generally higher than a serial plan
- Queries are parallelized by horizontally partitioning data and assigning a thread to each partition
- The degree of parallelism (DOP) is determined at the time of execution based on resource availability and Resource Governor settings