Ladies and gentlemen… The Query Optimizer

by Alessandro Mortola
Technical review by Sergio Govoni

Sponsor & Org

UNIVERSITÀ DEGLI STUDI DI PARMA

daniela malvisi COMMUNICATION

engage LABS

DATA SKILLS
UNDERSTANDING THE WORLD

Bi Factory
DATA KNOWLEDGE ADVISOR

iSolutions
SOFTWARE ENGINEERING

ETLOG
SOFTWARE SOLUTIONS LOGISTICS

CREDEM | CREDEMTEL

packt

LOBRA
A CULTURE OF INNOVATION EXCELLENCE

QUANTUMDATIS

Alessandro Mortola

@AlexMortola

https://www.linkedin.com/in/alessandromortola

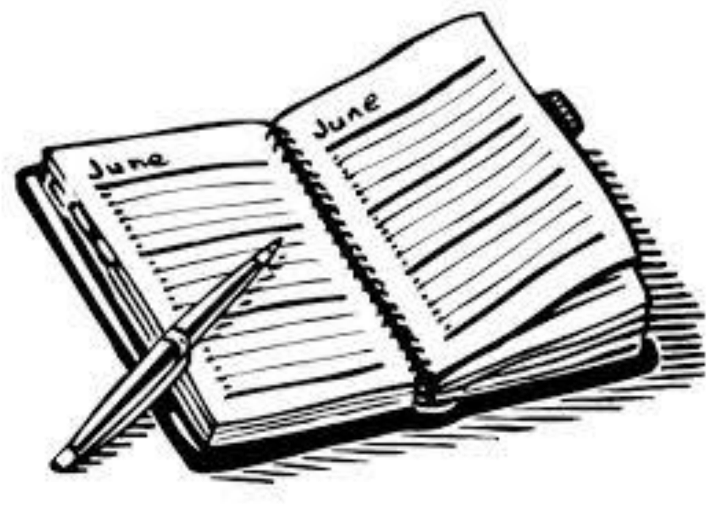
alessandro.mortola@gmail.com

- I have worked with all versions from Sql Server 7.0 to Sql Server 2022
- I am certified Sql Server 2016 Dev
- I am an author for sqlservercentral.com
- I spoke several times at Sql Saturday and SqlStart editions
- I currently work in Zucchetti S.p.A. - Asset Management Division as
  - Sql Server DBA
  - Analyst
  - Dev (.Net & Java)
  - …

Something magic happens!

	ProductID	Name	ProductNumber	MakeFlag
1	403	External Lock Washer 3	LE-1000	0
2	404	External Lock Washer 4	LE-1200	0
3	405	External Lock Washer 9	LE-1201	0
4	406	External Lock Washer 5	LE-1400	0

Agenda



- Introduction to Logical Trees
- (The knowledge of Trace Flags is taken for granted)
- A difficult job for the Query Optimizer
- Inside the Sql Server Query Optimizer’s optimization process

Logical Trees

"A logical query tree is a tree consisting of relational operators and relations. It specifies what operations to apply and the order in which to apply them. A logical query tree does ***not*** *select a particular algorithm to implement each relational operator."*

[Dr. Ramon Lawrence University of British Columbia Okanagan]

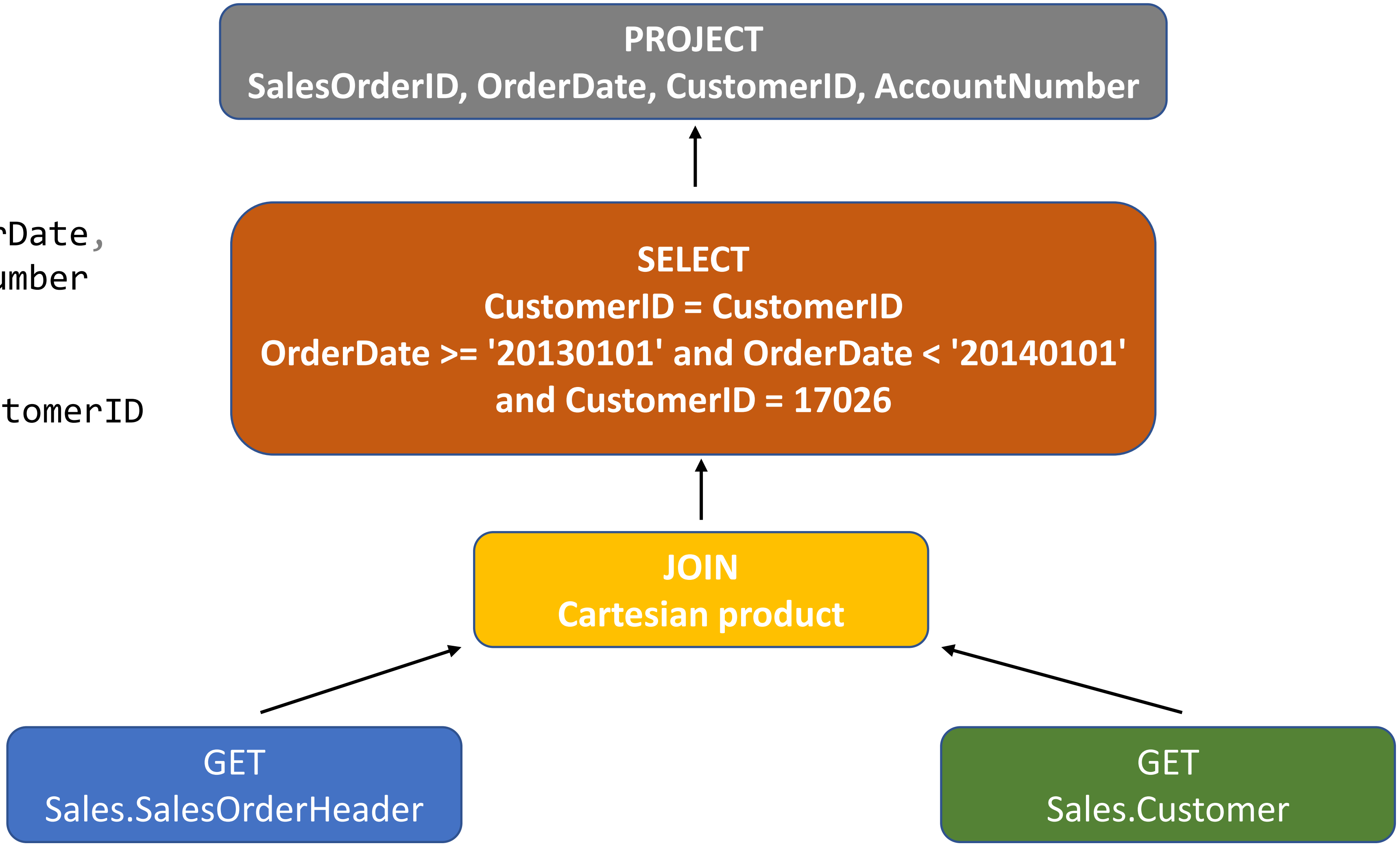
Logical Trees

The terminology is different from the one used with the Physical trees. For instance:

Operator	Description
GET	It reads the entire table
JOIN	It is a cartesian product that logically joins every row of one table with every row of the other
SELECT	It means which rows to return, i.e. “filtering”. It nearly corresponds to the SQL WHERE clause
PROJECT	It means defining which columns to return. It corresponds to the SQL SELECT clause

A sample query and one of its Logical Tree

```sql
select oh.SalesOrderID, oh.OrderDate,
       c.CustomerID, c.AccountNumber
from Sales.SalesOrderHeader oh
inner join Sales.Customer c
        on oh.CustomerID = c.CustomerID
where oh.OrderDate >= '20130101'
   and oh.OrderDate < '20140101'
   and oh.CustomerID = 17026;
```



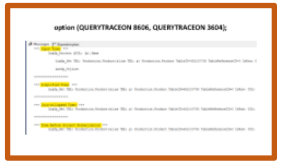
Trace Flags

They are used to set specific server characteristics or to alter a particular behaviour.

There are three scopes where trace flags can work: global, session and query.

At the query level they are set using the query hint QUERYTRACEON:

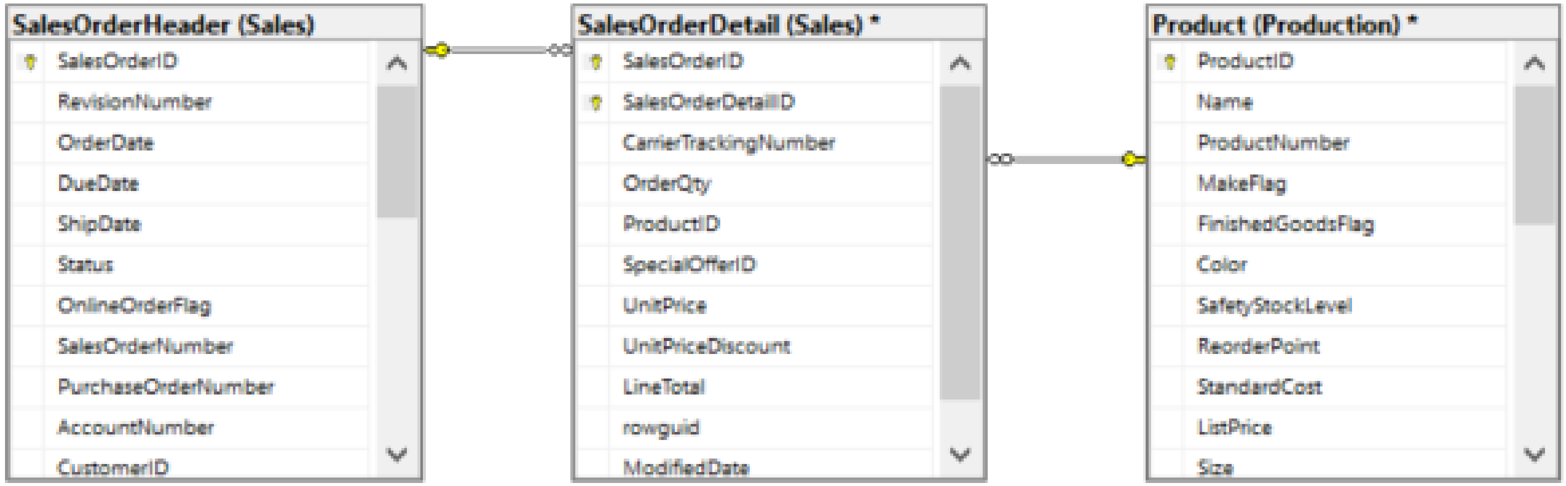
```sql
select p.Name
from Production.Product p
option (QUERYTRACEON 3604, QUERYTRACEON 8606);
```



A difficult job for the Query Optimizer

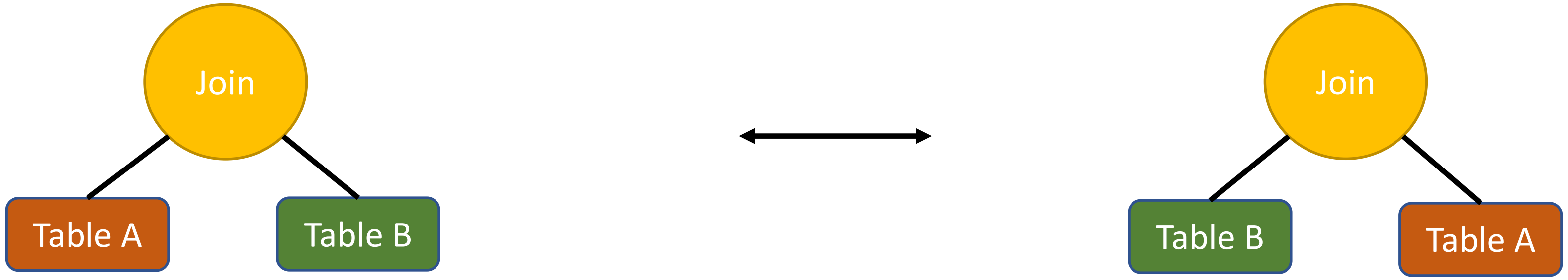
Which kind of job?

```
select oh.SalesOrderID, oh.OrderDate, p.Name as ProductName
from Sales.SalesOrderHeader oh
inner join Sales.SalesOrderDetail od on oh.SalesOrderID = od.SalesOrderID
inner join Production.Product p on od.ProductID = p.ProductID
where od.SpecialOfferID = 2 and p.ProductSubcategoryID = 23;
```

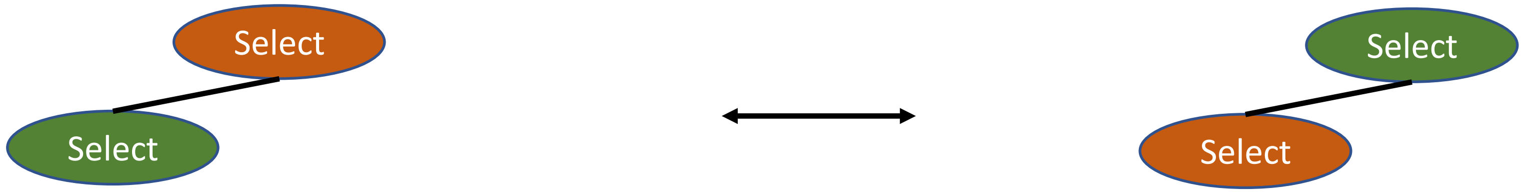


Equivalence rules

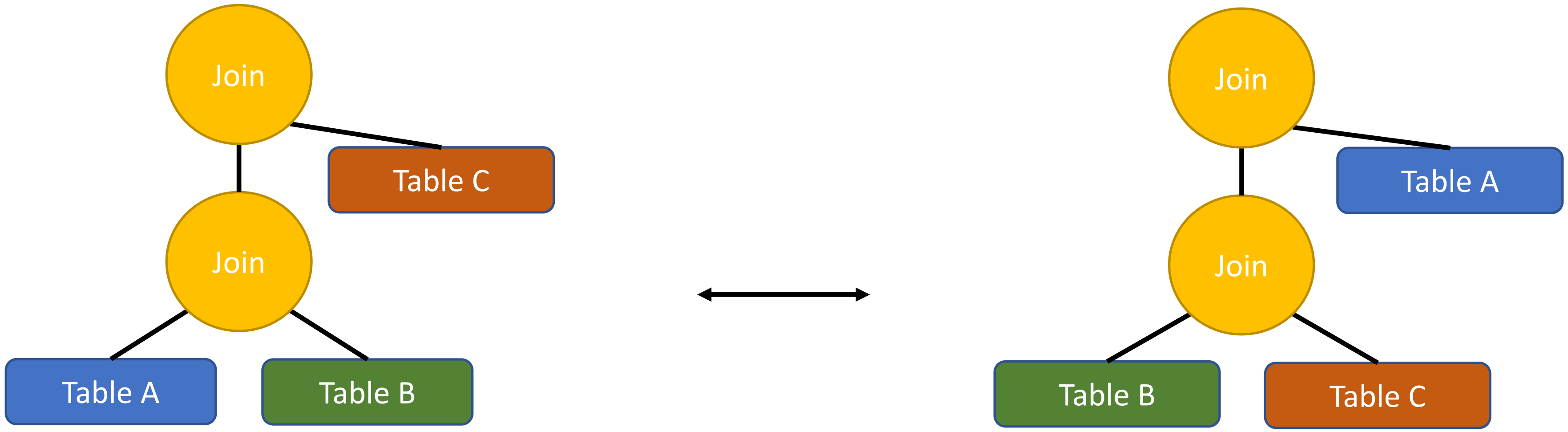
Joins commute



Selections commute

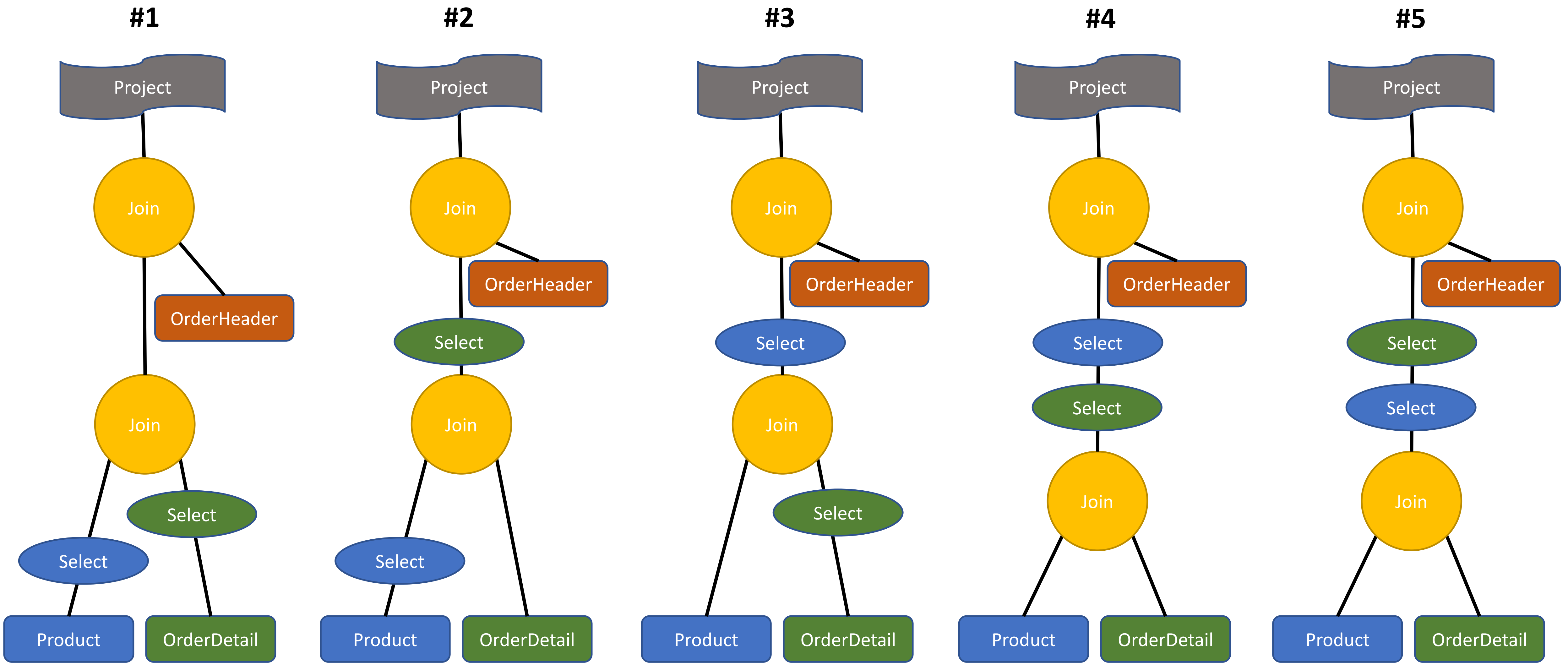


Joins are associative



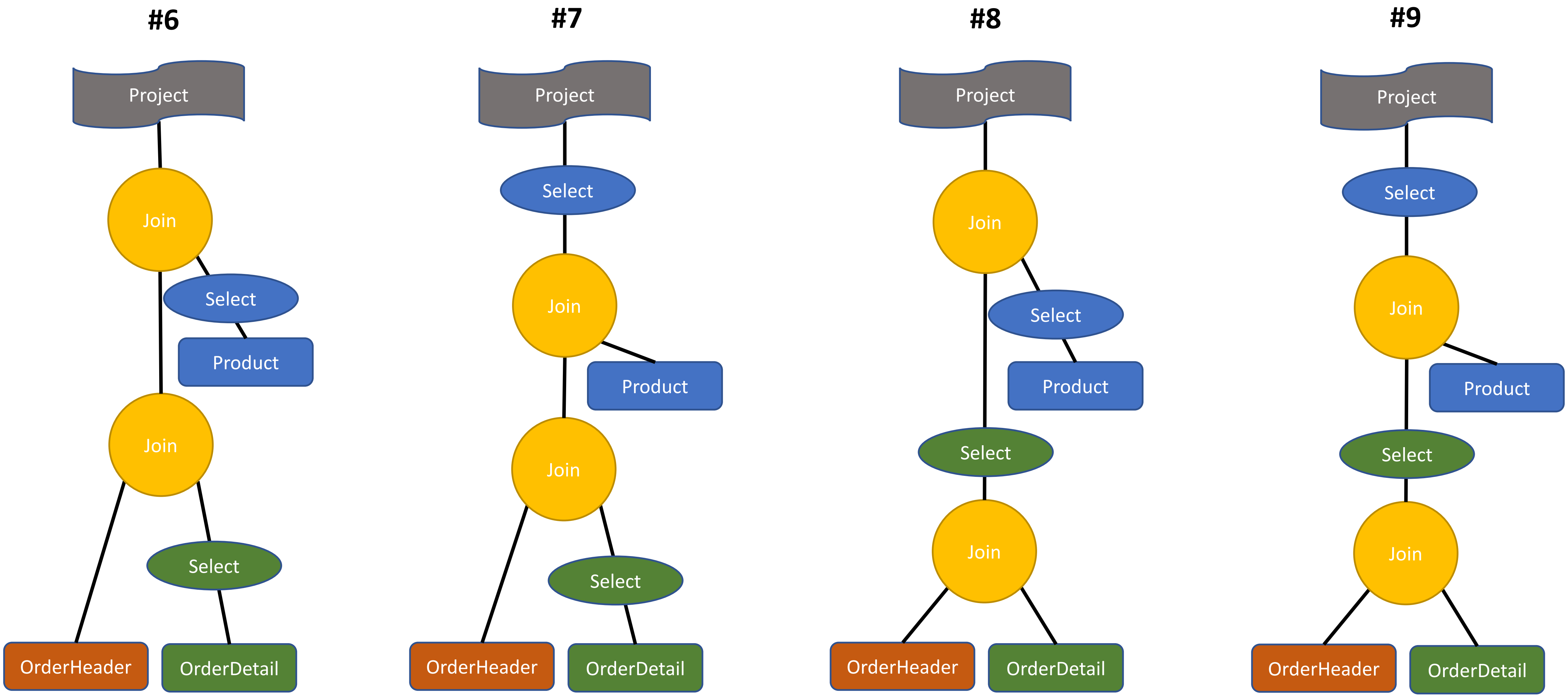
A sample query

Each of them, considering the “joins commute” rule, stands for 4 logical plans



A sample query

Each of them, considering the “joins commute” rule, stands for 4 logical plans



Some Maths

Total Logic plans: 9 * 4 → 36

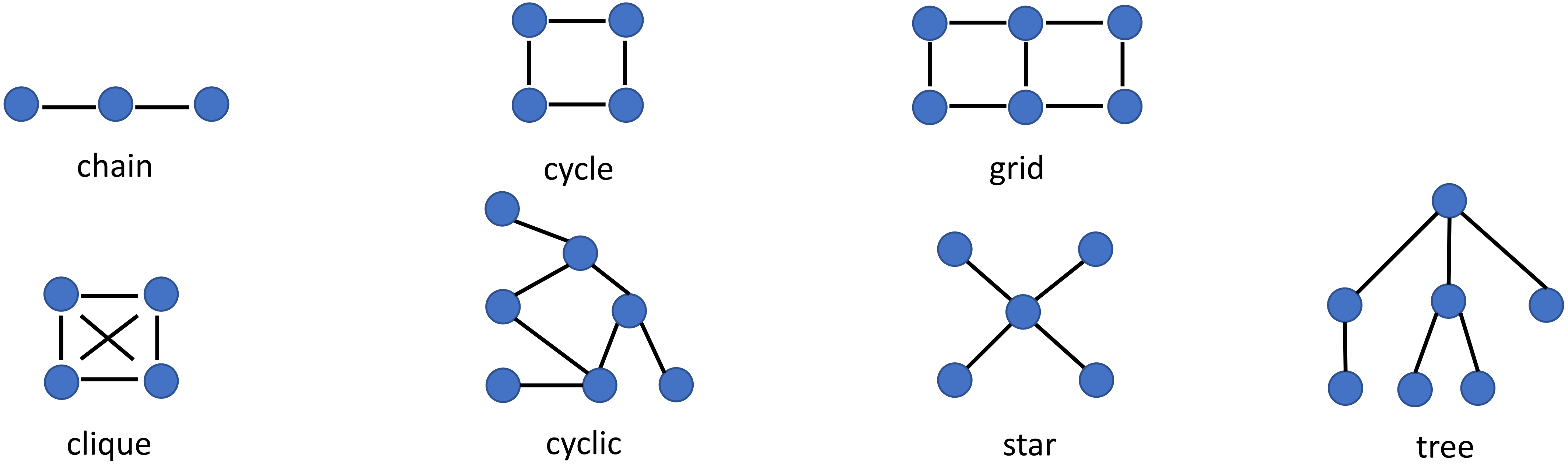
Assuming that the optimizer has three join strategies (Nested Loop, Hash Join, Sort-Merge Join), in a query with tree tables and two joins, considering just one plan, we have $3^{\wedge(\text{numberOfTables} - 1)}$ (→ 9) possible physical plans.

Considering all logical plans we have: 36 * 9 (→ 324) physical plans for such simple query!!!

Let’s generalize: Query Graphs

The query graph is an undirected graph with R1, …, Rn as nodes where Ri represents a relation.
A predicate of the form a1 = a2 where a1 is an attribute of Ri and a2 is an attribute of Rj, forms an edge between Ri and Rj labelled with the predicate.
<u>Real world queries are somewhere in-between.</u>

Shapes of Query Graphs:



Let’s generalize: Join Trees

A Join Tree is a binary tree with:

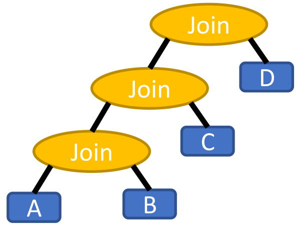
- Join operators as inner nodes
- Relations as leaf nodes

Commonly used classes of join trees

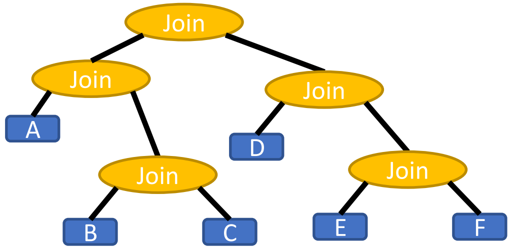
- Left-deep – the right hand side is always a base table
- Right-deep – the left hand side is always a base table
- Zig-zag
- Bushy

The first three are summarized as linear trees

Left-deep



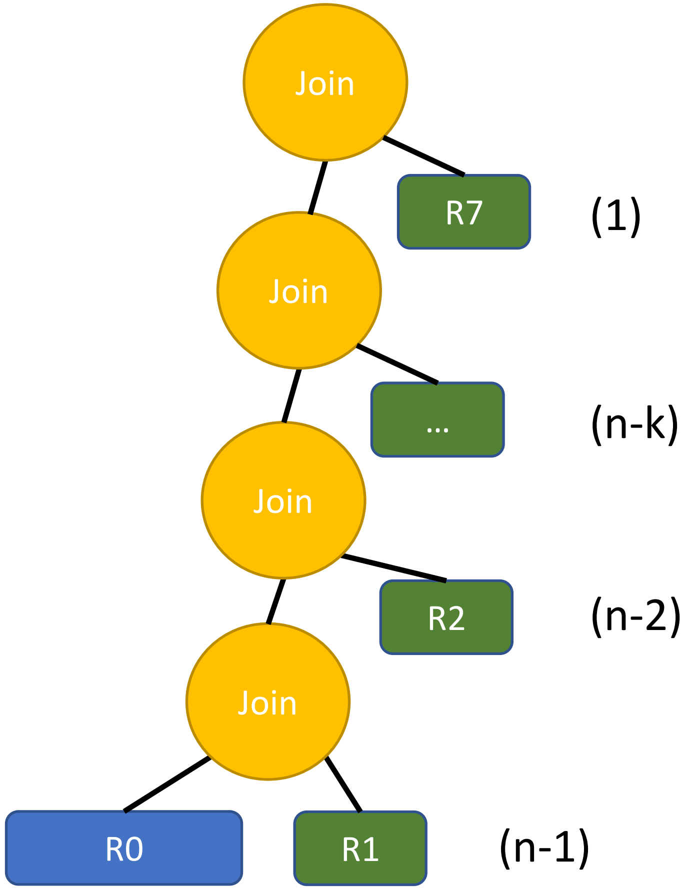
Bushy



Number of logical plans depending on Query Graphs and Join Trees

	Chain Queries		Star Queries	
Tables	**Left-Deep**	**Bushy**	**Left-Deep**	**Bushy**
2	2	2	2	2
3	4	8	4	8
4	8	40	12	48
5	16	224	48	384
6	32	1344	240	3840
7	64	8448	1440	46080
8	**128**	**54912**	**10080**	**645120**
9	256	366080	80640	10321920
10	512	2489344	725760	18579450

Star query with eight tables in join using a left-deep tree



Nr of possible combinations: 2 * (n – 1)!

If n = 8 → 2 * 7! → 10080

In this scenario, for a Star Query with 8 tables, considering only one access method per table, we have 10080 * $3^{\wedge 7}$ possible physical plans. This number is… **22.044.960**

That means that it is impossible for the QO to evaluate all of them!

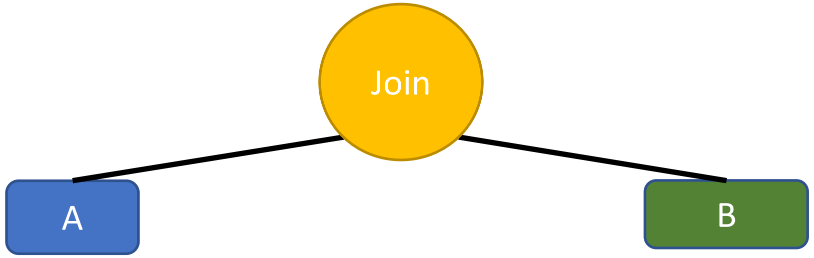
How to deal with it? Left-deep trees and Dynamic Programming

Usually DB systems, in order to reduce the search space, choose Left-Deep plans and not Bushy plans because they are generally less.

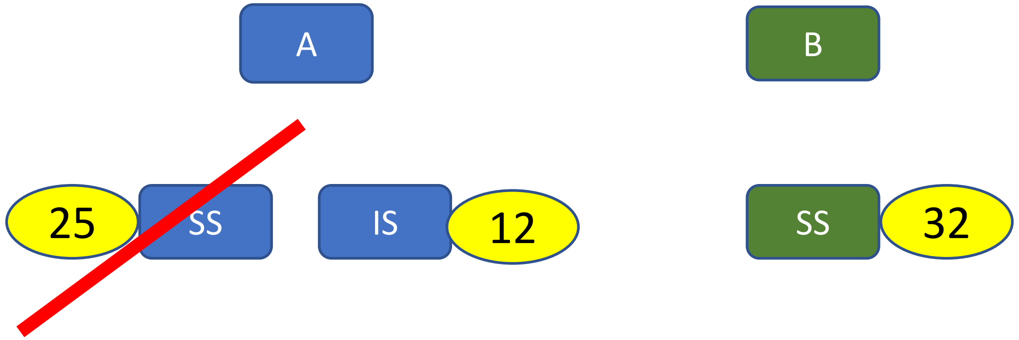
Most query optimizers use the Dynamic Programming algorithm that:

- is used to select a join order
- is performed in N steps (if N relations are joined)
- at each step, the algorithm remembers the best (based on cost) subplans for later use pruning useless subtrees

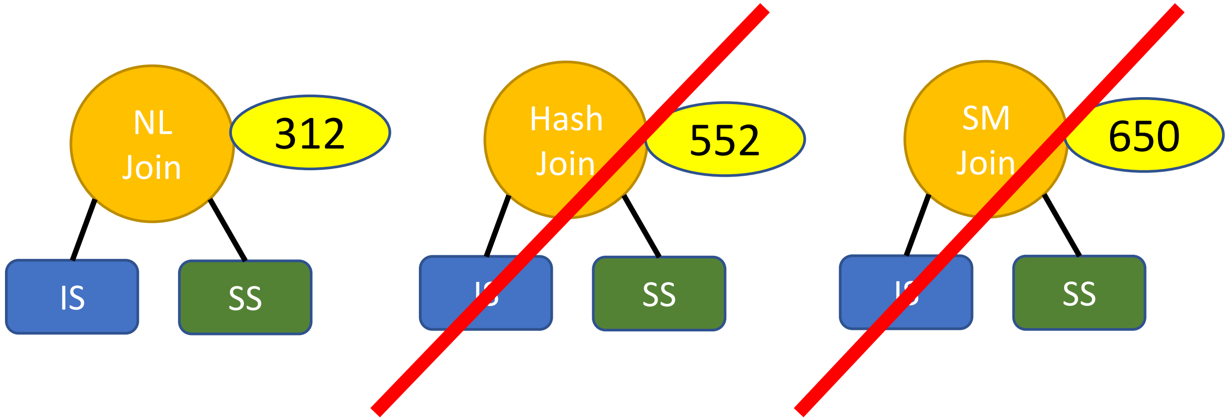
Dynamic Programming – A simple example



Step # 1 - Best plan for each relation:



Step # 2 – Best plan with two relations:



SS → Sequential Scan
IS → Index Scan / Seek

Inside the Sql Server Query Optimizer’s Optimization process

Inside the Sql Server QO – The query-processing process

First Step

- Parser
- Binder
- Simplification
- Derive Cardinality

Logical Tree Types

Converted
Input (aka Bound)
Simplified
Join Collapsed
Before Project Normalization
After Project Normalization

Qualify Trivial Plan?

No

Second Step - Exploration
Cost-based

- Search 0
- Search 1
- Search 2

MEMO

Yes

Output Tree (physical tree)

Post Optimization rewrites

Execution Engine

Parser & Binder | Simplification | Derive Cardinality | Trivial Plan

First step – Parser & Binder (Language Processing)

Parser	Binder (aka Algebrizer or Normalizer)
It validates the syntax	It performs the metadata discovery and the name resolution
It identifies the constants	It checks the user permission
It translates the T-SQL query into an algebra tree representation of logical operators, called Parse Tree	It performs the Data Type resolution
	It generates the query hash based on the query text and checks if the plan exists into the plan cache
	It expands the views
	It builds the Converted Tree

Parser & Binder → Simplification → Derive Cardinality → Trivial Plan

First step - Simplification

- It is the first optimization step; the goal is to reduce the query tree into a simpler form 
- When possible, it converts subqueries to joins
- It performs the “Join simplification”
  - It removes unnecessary joins 
  - It converts outer to inner joins, when appropriate 
- It performs the “Predicate pushdown”
- It performs the “Contradiction detection” 
- It performs the “Constant folding” 
- It performs the “Domain simplification” 

Parser & Binder | Simplification | **Derive Cardinality** | Trivial Plan

First step – Derive Cardinality

The cardinality estimation is computed for base tables and the statistics can provide additional information.

Statistics for relevant columns are created or updated as required.

Based on this, a cardinality is computed for each node of the plan and an initial join order is set.

Parser & Binder → Simplification → Derive Cardinality → Trivial Plan

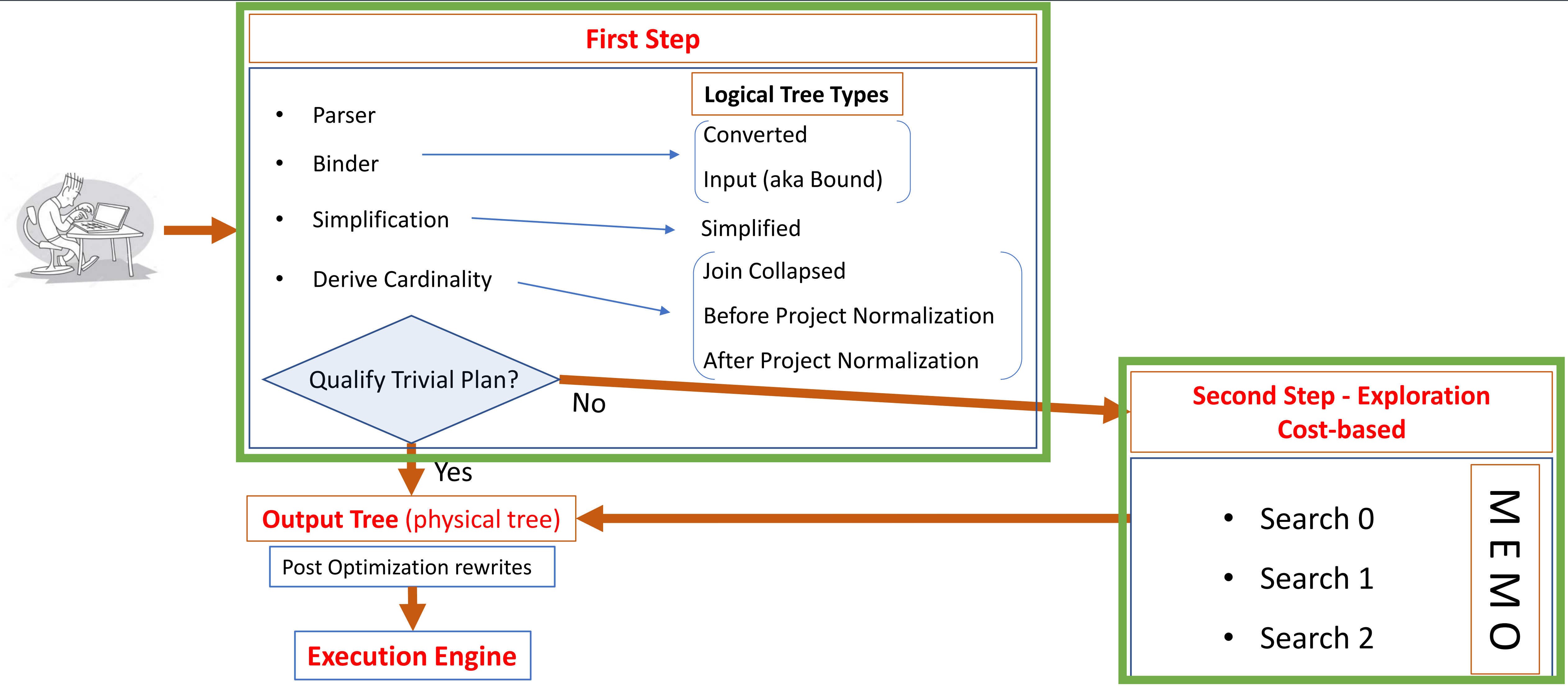
First step - Trivial plan - Characteristics

- The query is considered “simple”
- There is only one way to solve the query or one obviously best way; the QO does not invest any more time for a simple query
- The plan is not cost-based and not cached
- The “Optimization Level” entry in the Properties window of the graphical plan shows TRIVIAL
- Joins, inequality conditions, aggregations, subqueries and non-covering index presence usually prevent this optimization
- Nothing is registered in the missing index DMVs

First step - Trivial plan - Notes

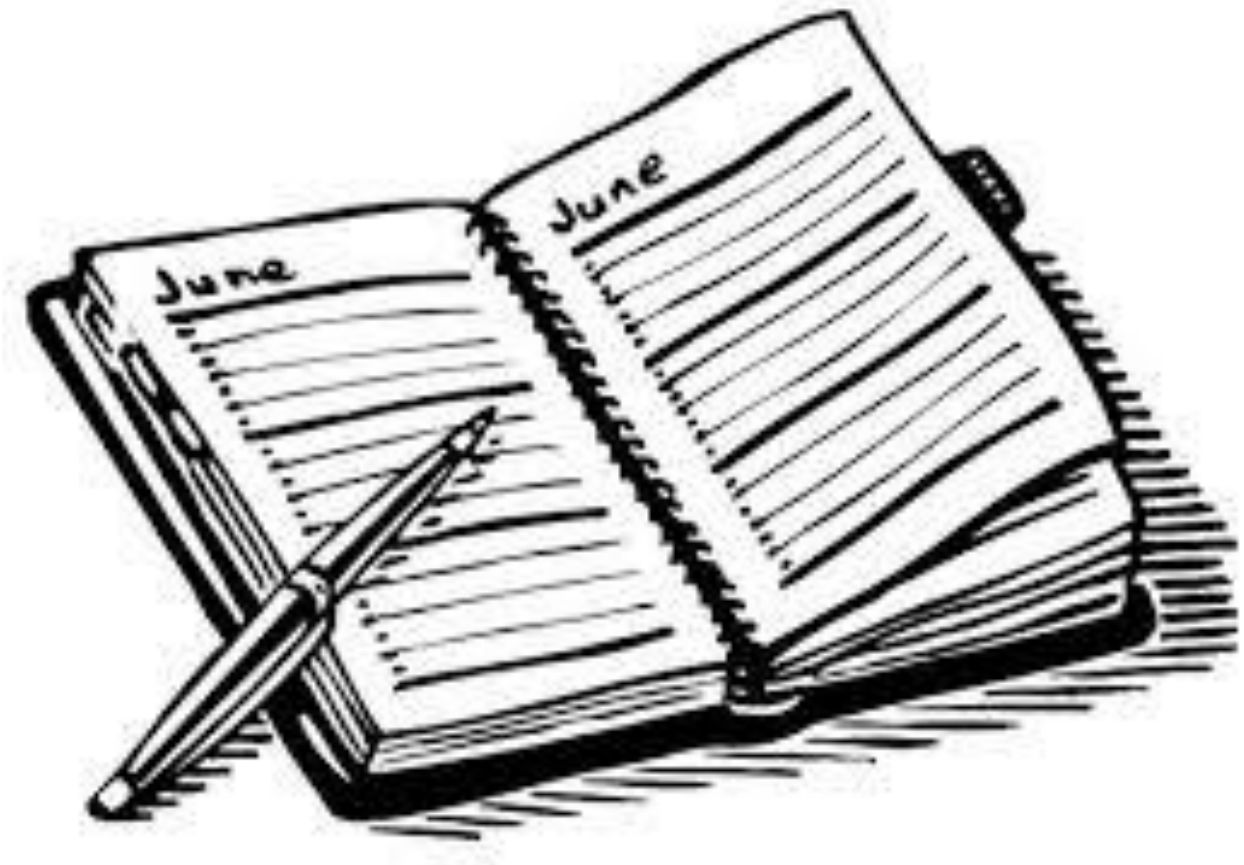
- The Trivial plan is not applied when the estimated cost exceeds the configured “cost threshold for parallelism” because *any trivial plan that would qualify for a parallel plan suddenly presents a choice, and a choice means the query is no longer ‘trivial’ [Paul White]*
- The consequence here is that a plan produced at this stage will always be serial. And if you put the Cost Threshold to 0 no query will be marked as trivial

Inside the Sql Server QO – The query-processing process



The Second Step Agenda

- Introductory concepts
    - Phases and costing
    - Rules
    - The Memo
- Search 0
- Search 1
- Search 2



Phases and Costing

- The Second Step consists of three Searches
  - Search 0 (aka Transaction Processing Phase)
  - Search 1 (aka Quick Plan)
  - Search 2 (aka Full Optimization)
- Costing - Elements like available memory, CPU, DOP, expected I/O (sequential, random), partitioning, average row size and SET options are also considered for “costing”

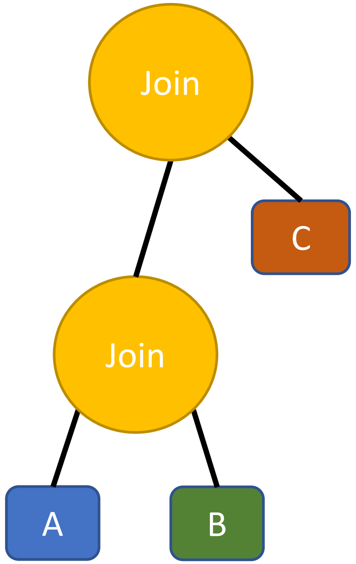
Rules

- They *are based on relational algebra, taking a relational operator tree and generating equivalent alternatives, in the form of equivalent relational operator trees [Benjamin Navarez]*
- We can find both Logical Transformation Rules (Exploration) and physical alternatives (Implementation)
- Have a look at the DMV sys.dm_exec_query_transformation_stats to see all the rules available for the current SQL Server version (405 in Sql Server 2017, 421 in Sql Server 2019 and 439 in Sql Server 2022). For instance:
  - JoinCommute
  - JNtoHS
  - GbAggBeforeJoin

The Memo comes into the game

- The QO does not materialize every single alternative
- All the logical or physical alternatives found during the optimization process, are stored in the Memo structure and organized in groups
- The alternatives that belong to the same group share the same logical properties
- Each group is independently optimized and the resulting plan is the combination of the best group options
- The optimization process stops when there are no more valuable optimization steps to decrease the cost

The Memo – Encoding of alternatives



Group 6	Join 3 & 4	Join 5 & 1	Nested Loop 5 & 1	Hash Join 5 & 1
Group 5		Join 2 & 4	Nested Loop 2 & 4	Merge Join 4 & 2
Group 4	Scan C		Clustered Index Scan	
Group 3	Join 1 & 2		Nested Loop 1 & 2	
Group 2	Scan B		Clustered Index Scan	
Group 1	Scan A		Clustered Index Scan	

https://www.red-gate.com/library/inside-the-sql-server-query-optimizer

Search 0 (aka Transaction Processing Phase)

- It is designed to process small queries as you can find in an OLTP workload
- It is used with queries with at least three tables
- Only basic heuristics are considered
- The only join orders considered in the Search 0 phase are those generated in the initial set of join orders. *The process usually starts joining the smallest tables or the tables that achieve the largest filtering based on their selectivity [Benjamin Navarez]*

Search 0 (aka Transaction Processing Phase)

- *This phase primarily considers nested-loop joins, though hash match may be used when a loop join implementation is not possible [Paul White]*
- Parallel plans are not considered in this phase
- The cost of the best plan evaluated is compared to an internal threshold. If the cost of the plan overtakes the threshold, the optimization moves on to the next level

Search 1 (aka Quick Plan)

- It enables more transformation rules compared to the Phase 0
- It enables a limited join reordering
- *The cost of the best plan evaluated is compared to an internal threshold. If the cost is below the threshold, the plan is selected, otherwise the optimization moves on to the parallel queries (if possible). The serial and the parallel are compared, the most convenient is used in the next level of optimization, the Search 2 phase [Sergio Govoni]*

Search 2 (aka Full Optimization)

- It considers complex and very complex queries
- Advanced optimization strategies are used
- It considers the parallelism, spills and spools to tempdb
- All transformation rules are unlocked
- A plan is returned anyway, even if it still seems to the optimizer not good enough

Entry and Termination conditions

- *Each phase has entry conditions, a set of enabled rules and termination conditions*
- *Termination: if the current lowest plan cost boundary drops below a configured value, the search will terminate early with a "Good Enough Plan Found" result*
- *The optimizer also <u>sets a budget at the start of a phase</u> for the number of optimization "moves" it considers sufficient to find a good plan (remember the optimizer's goal is to find a good enough plan quickly). If the process of exploring and implementing alternatives exceeds this 'budget' during a phase, the phase terminates with a TimeOut message* *[Paul White]*

Inside the Sql Server QO – The query-processing process

First Step

- Parser
- Binder
- Simplification
- Derive Cardinality

Logical Tree Types

Converted

Input (aka Bound)

Simplified

Join Collapsed

Before Project Normalization

After Project Normalization

Qualify Trivial Plan?

No

Yes

Second Step - Exploration
Cost-based

- Search 0
- Search 1
- Search 2

MEMO

Output Tree (physical tree)

Post Optimization rewrites

Execution Engine

Output tree and Post-optimization rewrite

- *The Output tree is the first physical tree where the logical operators are replaced with physical operators*
- At this stage, some other beneficial modifications, not considered as a "cost", can be applied to the "plan"
- For instance:
  - a suitable predicate can be pushed into the seek or scan as a residual predicate
  - a Parallelism (Repartition Streams) operator can be used to swaps rows between different streams in order to optimize the query for what's ahead

Resources

Trace Flags

Trace Flag	Description
TF 3604	It enables output in the messages pane
TF 8605	It will output the Converted tree
TF 8606	It enables the output of the parse tree in the different phases of optimization
TF 8607	It shows the optimization output tree (before Post Optimization Rewrite)
TF 8619	It shows applied transformation rules
TF 8608	It shows the initial memo structure, input tree for cost based optimization
TF 8615	It shows the final memo structure
TF 8675	It shows the optimization stages and times
TF 7352	It shows the final query tree (after Post Optimization Rewrite)
TF 2373	It displays memory utilization and used rules during the optimization process
TF 8780	It gives more time to the QO for searching the optimal plan

Credits

Paul White – Query Optimizer Deep Dive
https://www.sql.kiwi/2012/04/query-optimizer-deep-dive-part-1.html

Paul Holmes – Logical Trees
http://www.paulholmes.net/2020/07/logical-plans-part-1-introduction.html

Conor Cunningham – Inside the Sql Server Query Optimizer
https://sqlbits.com/Sessions/Event6/Inside_the_SQL_Server_Query_Optimizer

Credits

Benjamin Navarez – Dive into the Query Optimizer-Undocumented Insight
https://sqlbits.com/Sessions/Event12/Dive_into_the_Query_Optimizer-Undocumented_Insight

David DeWitt (Brent Ozar Unlimited) - SQL Query Optimization. Why is it so hard to get right?
https://www.youtube.com/watch?v=RQfJkNqmHB4

Sergio Govoni – Sql Saturday 777 – SQL Server Query Optimizer end-to-end
https://vimeo.com/304150423

Resources

Dmitry Piliugin
https://www.sqlshack.com/query-plan-on-a-busy-server/

Sql Server Query Tree Viewer
https://www.tf3604.com/tools/ssqtv/

sys.dm_exec_query_optimizer_info
https://github.com/MicrosoftDocs/sql-docs/blob/live/docs/relational-databases/performance/use-dmvs-determine-usage-performance-views.md

Trace Flags
https://docs.microsoft.com/en-us/sql/t-sql/database-console-commands/dbcc-traceon-trace-flags-transact-sql?view=sql-server-ver16

Are there any questions?

DATA
SATURDAYS

Thank you!

engage
LABS

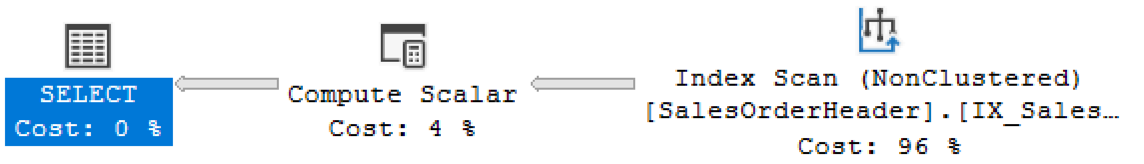
Simplification – First example

```
select SalesOrderID, COUNT(*)
from Sales.SalesOrderHeader
group by SalesOrderID
option (recompile, querytraceon 3604, querytraceon 8606);
```

The Simplified tree

```
*** Input Tree: ***
        LogOp_Project QCOL: [AdventureWorks].[Sales].[SalesOrderHeader].SalesOrderID COL: Expr1002

            LogOp_GbAgg OUT(QCOL: [AdventureWorks].[Sales].[SalesOrderHeader].SalesOrderID,COL: Expr1002 ,)
                                      BY(QCOL: [AdventureWorks].[Sales].[SalesOrderHeader].SalesOrderID,)

[...]

*******************

*** Simplified Tree: ***
        LogOp_Project

            LogOp_Get TBL: Sales.SalesOrderHeader Sales.SalesOrderHeader TableID=1922105888 TableReferenceID=0

            AncOp_PrjList

                AncOp_PrjEl COL: Expr1002

                    ScaOp_IIF int,Null,ML=4

                        ScaOp_Logical x_lopIsNull

                            ScaOp_Const TI(int,ML=4) XVAR(int,Not Owned,Value=0)

                        ScaOp_Const TI(int,ML=4) XVAR(int,Not Owned,Value=0)

                        ScaOp_Const TI(int,ML=4) XVAR(int,Not Owned,Value=1)

*******************
```

The Query plan



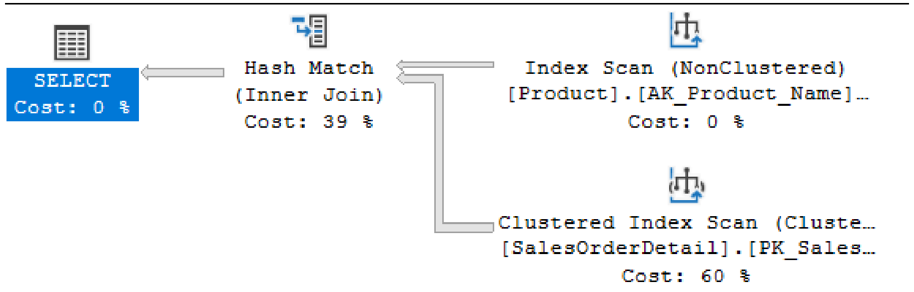
Join Simplification - 1

```sql
select od.SalesOrderDetailID, od.OrderQty, p.Name as ProductName
from Sales.SalesOrderDetail od
inner join Production.Product p on od.ProductID = p.ProductID
left join Production.ProductModel pm on p.ProductModelID = pm.ProductModelID
OPTION (RECOMPILE, QUERYTRACEON 3604, QUERYTRACEON 8606);
```

The Simplified tree

```
*** Input Tree: ***
[...]
    LogOp_Get TBL: Sales.SalesOrderDetail(alias TBL: od) Sales.SalesOrderDetail TableID=
[...]
    LogOp_Get TBL: Production.Product(alias TBL: p) Production.Product TableID=48210075
[...]
    LogOp_Get TBL: Production.ProductModel(alias TBL: pm) Production.ProductModel Table

*******************

*** Simplified Tree: ***
        LogOp_Join
            LogOp_Get TBL: Sales.SalesOrderDetail(alias TBL: od) Sales.SalesOrderDetail
            LogOp_Get TBL: Production.Product(alias TBL: p) Production.Product TableID=
[...]
*******************
```

The Query plan



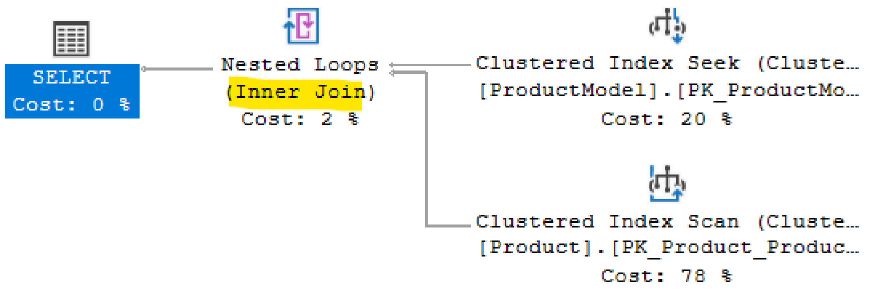
Join Simplification - 2

```sql
select p.ProductID, p.Name
from Production.Product p
left join Production.ProductModel pm on p.ProductModelID = pm.ProductModelID
where pm.ProductModelID = 9
option (recompile, QUERYTRACEON 8606, QUERYTRACEON 3604);
```

The Simplified tree

```
*** Input Tree: ***
[...]
    LogOp_LeftOuterJoin
        LogOp_Get TBL: Production.Product(alias TBL: p) Production.Product TableID=482100758 Ta
        LogOp_Get TBL: Production.ProductModel(alias TBL: pm) Production.ProductModel TableID=
[...]
*******************

*** Simplified Tree: ***
        LogOp_Join
            LogOp_Select
                LogOp_Get TBL: Production.Product(alias TBL: p) Production.Product TableID=482

                ScaOp_Comp x_cmpEq
                    ScaOp_Identifier QCOL: [p].ProductModelID
                    ScaOp_Const TI(int,ML=4) XVAR(int,Not Owned,Value=9)

            LogOp_Select
                LogOp_Get TBL: Production.ProductModel(alias TBL: pm) Production.ProductModel

[...]
*******************
```

The Query plan



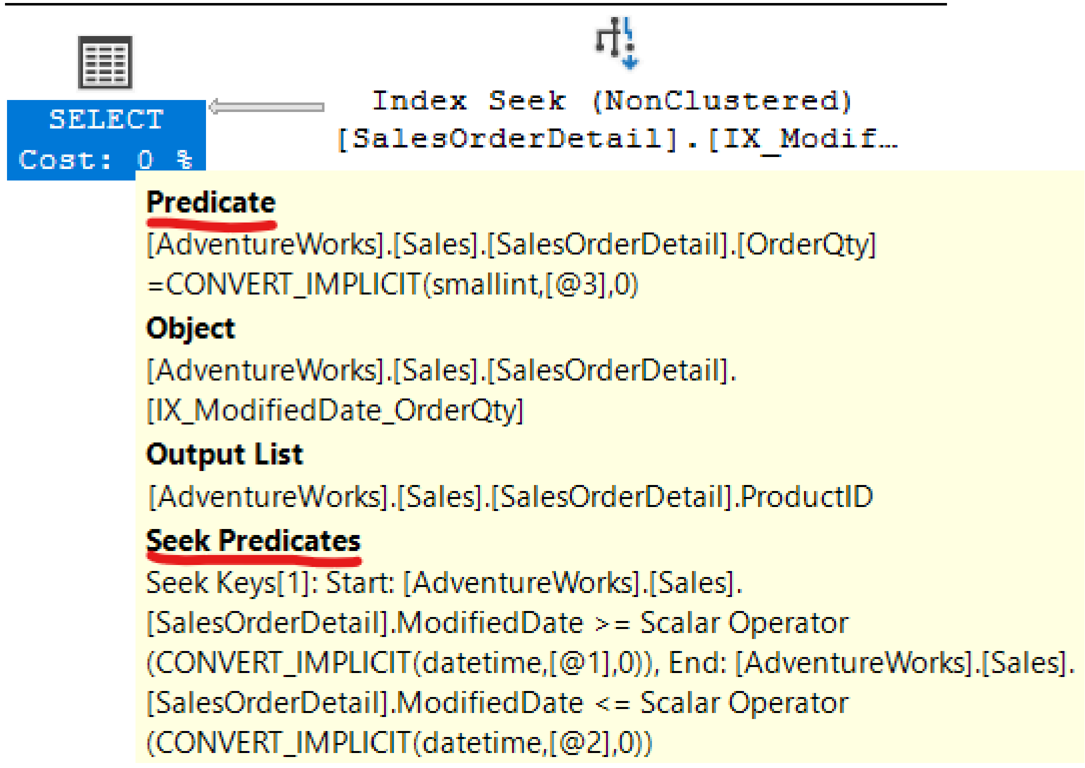
Predicate pushdown

```sql
SELECT ProductID
FROM Sales.SalesOrderDetail
WHERE ModifiedDate BETWEEN '2011-01-01' AND '2012-01-01' AND OrderQty = 2
```

A new index has been created

```sql
CREATE NONCLUSTERED INDEX
IX_ModifiedDate_OrderQty ON Sales.SalesOrderDetail

(ModifiedDate)

INCLUDE (ProductID, OrderQty);
```

The Query plan



Contradiction detection

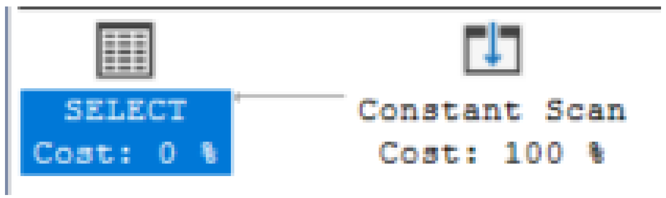
```sql
select *
from Sales.SalesOrderHeader
where Freight = -1
option (recompile, QUERYTRACEON 8606, QUERYTRACEON 3604);
```

(General)	
Expression	([Freight]>=(0.00))
Identity	
(Name)	CK_SalesOrderHeader_Freight
Description	Check constraint [Freight] >= (0.00)

The Simplified tree

```
*** Simplified Tree: ***
        LogOp_ConstTableGet (0) COL: IsBaseRow1000  QCOL: [A

********************
```

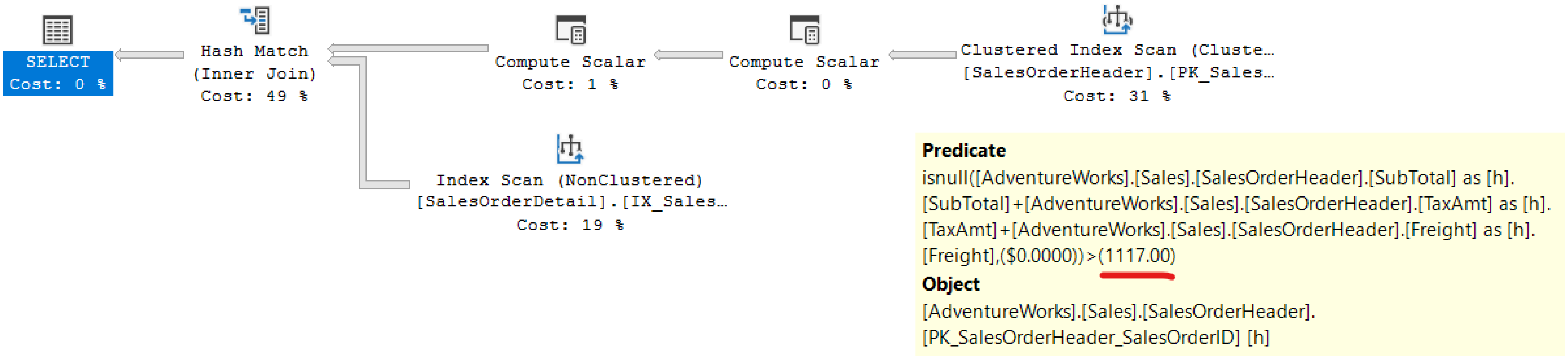
The Query plan



Constant folding

```sql
select h.SalesOrderID, h.OrderDate, h.TotalDue, d.ProductID
from Sales.SalesOrderHeader AS h
inner join Sales.SalesOrderDetail AS d ON h.SalesOrderID = d.SalesOrderID
where h.TotalDue > 117.00 + 1000.00;
```

The Query plan



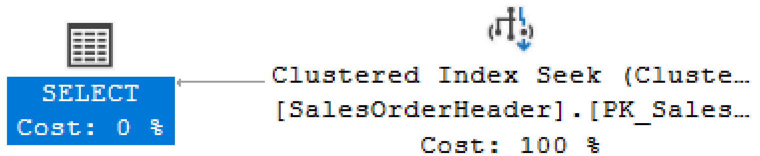
Domain simplification

```sql
select SalesOrderID, OrderDate, ShipDate
from Sales.SalesOrderHeader
where SalesOrderID between 40000 and 50000 and SalesOrderID between 50000 and 60000
option (recompile, QUERYTRACEON 8606, QUERYTRACEON 3604);
```

The Simplified tree

```
*** Simplified Tree: ***
        LogOp_Select
            LogOp_Get TBL: Sales.SalesOrderHeader Sales.SalesOrderHeader TableID=1922105888 Tabl
            ScaOp_Comp x_cmpEq
                ScaOp_Identifier QCOL: [AdventureWorks].[Sales].[SalesOrderHeader].SalesOrderID
                ScaOp_Const TI(int,ML=4) XVAR(int,Not Owned,Value=50000)
*******************
```

The Query plan



Seek Predicates
Seek Keys[1]: Prefix: [AdventureWorks].[Sales].[SalesOrderHeader].SalesOrderID = Scalar Operator((50000))

option (QUERYTRACEON 8606, QUERYTRACEON 3604);

Messages | Execution plan

```
*** Input Tree: ***
        LogOp_Project QCOL: [p].Name

            LogOp_Get TBL: Production.Product(alias TBL: p) Production.Product TableID=482100758 TableReferenceID=0 IsRow: C

            AncOp_PrjList

*******************

*** Simplified Tree: ***
        LogOp_Get TBL: Production.Product(alias TBL: p) Production.Product TableID=482100758 TableReferenceID=0 IsRow: COL:

*******************

*** Join-collapsed Tree: ***
        LogOp_Get TBL: Production.Product(alias TBL: p) Production.Product TableID=482100758 TableReferenceID=0 IsRow: COL:

*******************

*** Tree Before Project Normalization ***
        LogOp_Get TBL: Production.Product(alias TBL: p) Production.Product TableID=482100758 TableReferenceID=0 IsRow: COL:
```