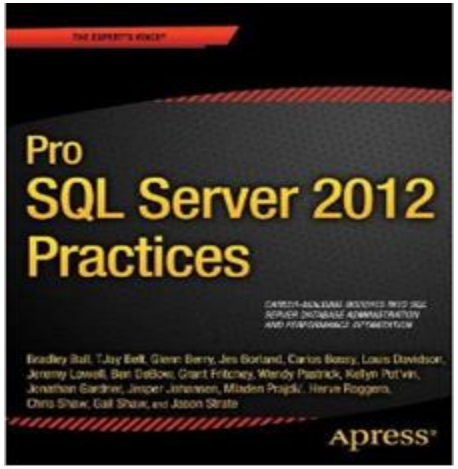
Inside the Query Optimizer

Speaker Introduction
Bradley Ball

- Almost 15 Years IT Experience
- Previous experience DBA, for the U.S. Army, The Executive Office of the President, Sr. SQL DBA Staff Specialist at Publix
- Currently the Data Platform Management Lead for [Pragmatic Works]
- Microsoft VTSP for the Greater North East
- MCITP SQL 2005 DBA & SQL 2008 DBA
- Blog: http://www.SQLBalls.com
- Twitter: @SQLBalls @BradleyBall_PW
- Email: bball@PragmaticWorks.com
- Pro SQL Server 2012 Practices Author
-   Chapter 14 PAGE & ROW COMPRESSION!


THE EXPERT'S VOICE
Pro
SQL Server 2012
Practices
Bradley Ball, TJay Belt, Glenn Berry, Jes Borland, Carlos Bossy, Louis Davidson, Jeremy Lowell, Ben DeBow, Grant Fritchey, Wendy Pastrick, Kellyn Pot'vin, Jonathan Gardner, Jesper Johansen, Mladen Prajdić, Herve Roggero, Chris Shaw, Gail Shaw, and Jason Strate
Apress

Agenda

- Input Trees
- Optimize
- Joins
- Reasons to Join
- Hints, Plan Guides

FUN!

No Seriously FUN!

It won’t be Lame I Promise

Why Optimize?

- You Want to go FAST!
- Everybody's Doing it!
- My Boss Told Me To

How T-SQL is Processed

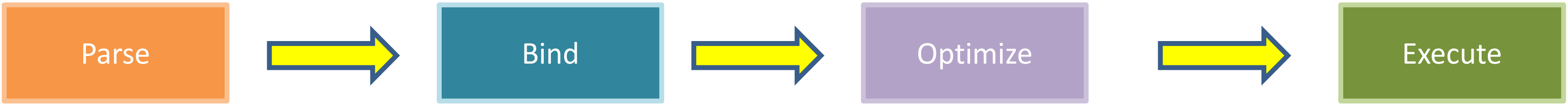
How We Write SQL

SELECT
TOP
DISTINCT
FROM
JOIN
ON
WHERE
GROUP BY
CUBE / ROLLUP
ORDER BY

SQL Reads What We Write

FROM
ON
JOIN
WHERE
GROUP BY
CUBE /ROLLUP
HAVING
SELECT
DISTINCT
ORDER BY
TOP

Relational Engine Query Process



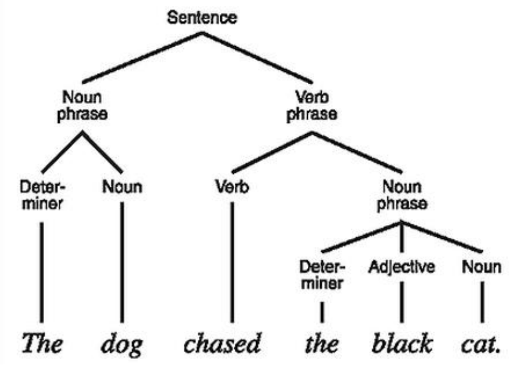
Relational Engine Query Process

Parse | Bind | Optimize | Execute

Parse

Validation of the syntax

No Executing a table



Relational Engine Query Process

Parse | Bind | Optimize | Execute

Bind

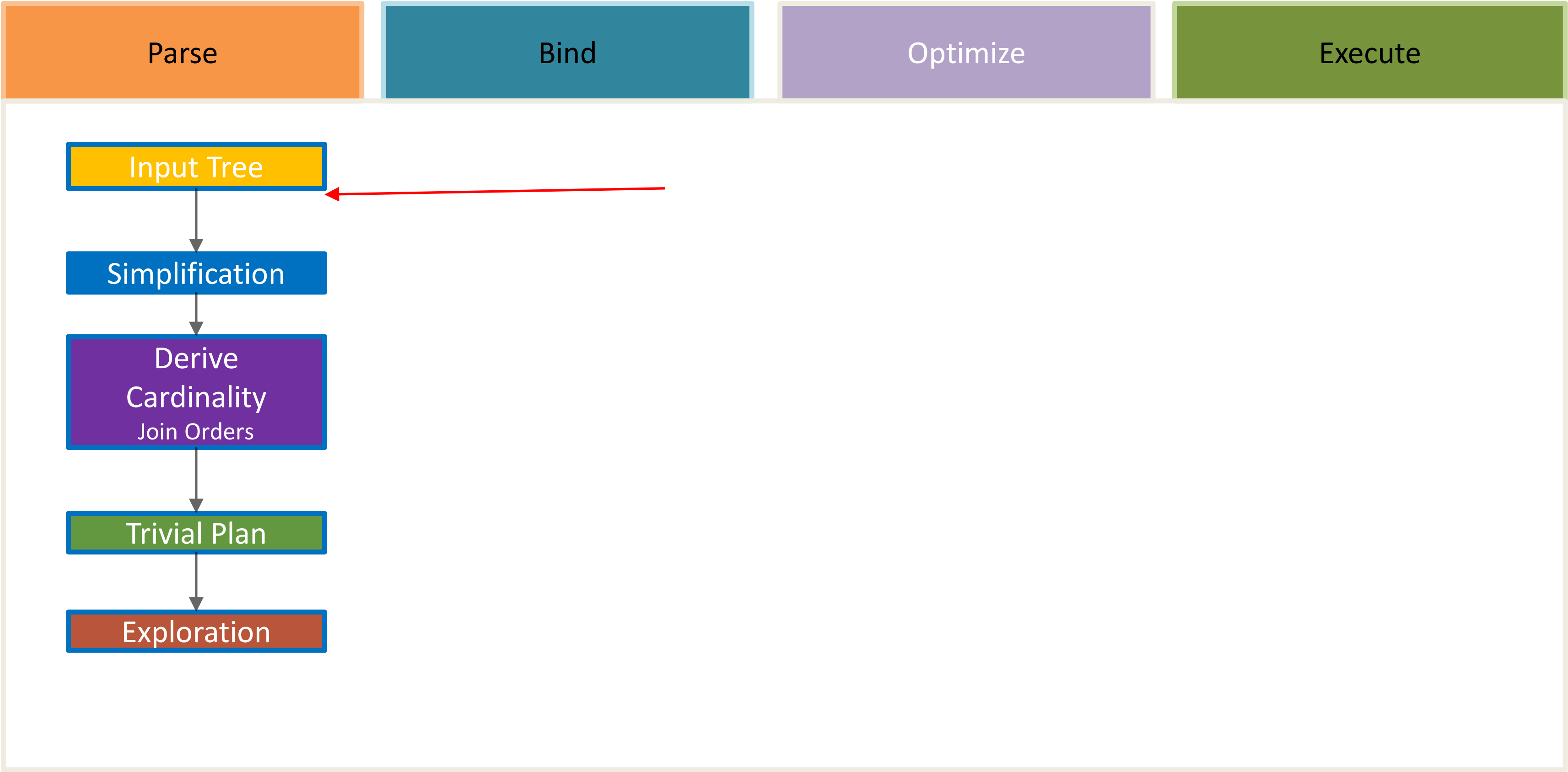
Name Resolution

Type Derivation

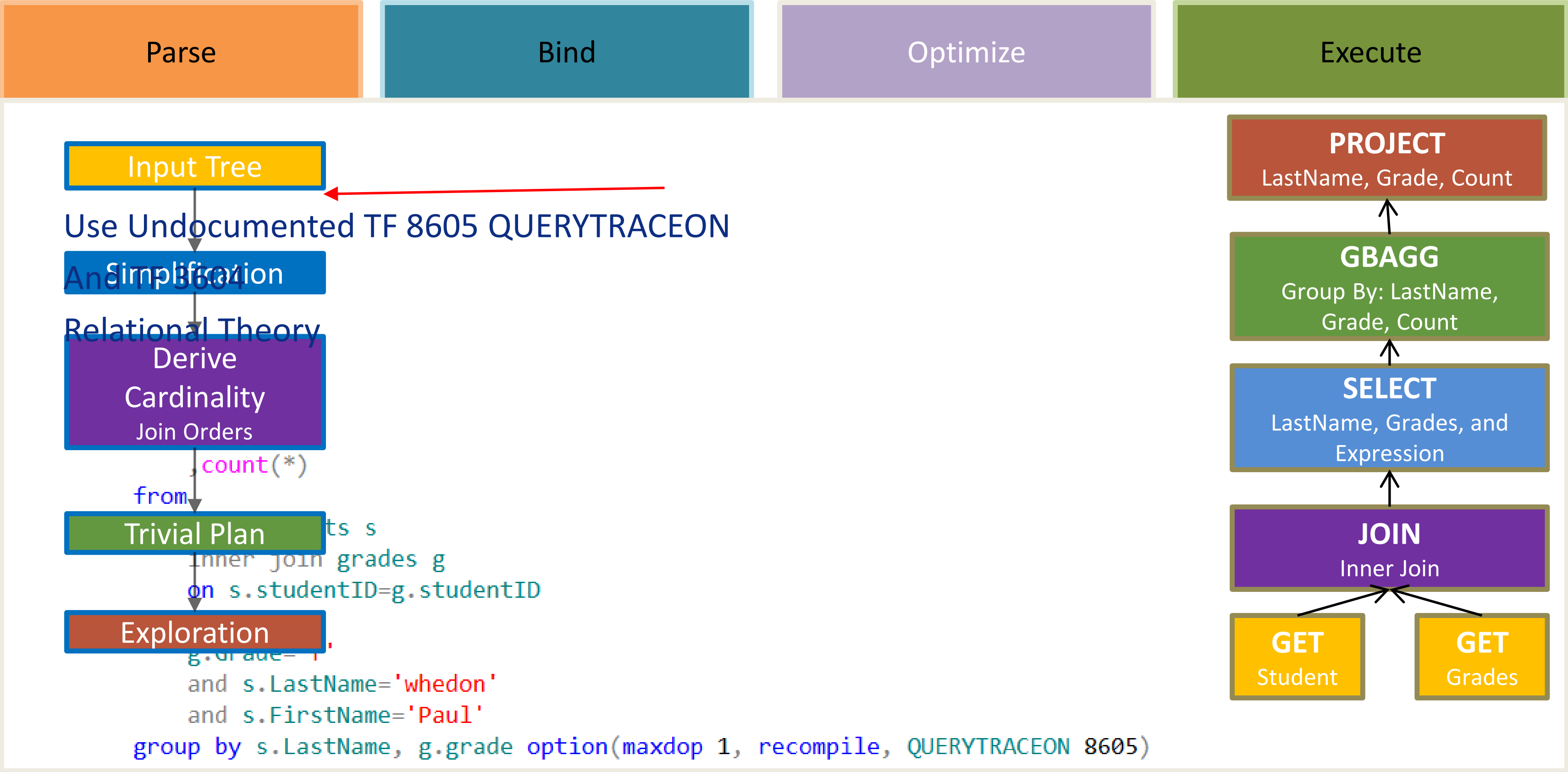
Aggregate Binding

Group Binding

Relational Engine Query Process



Relational Engine Query Process



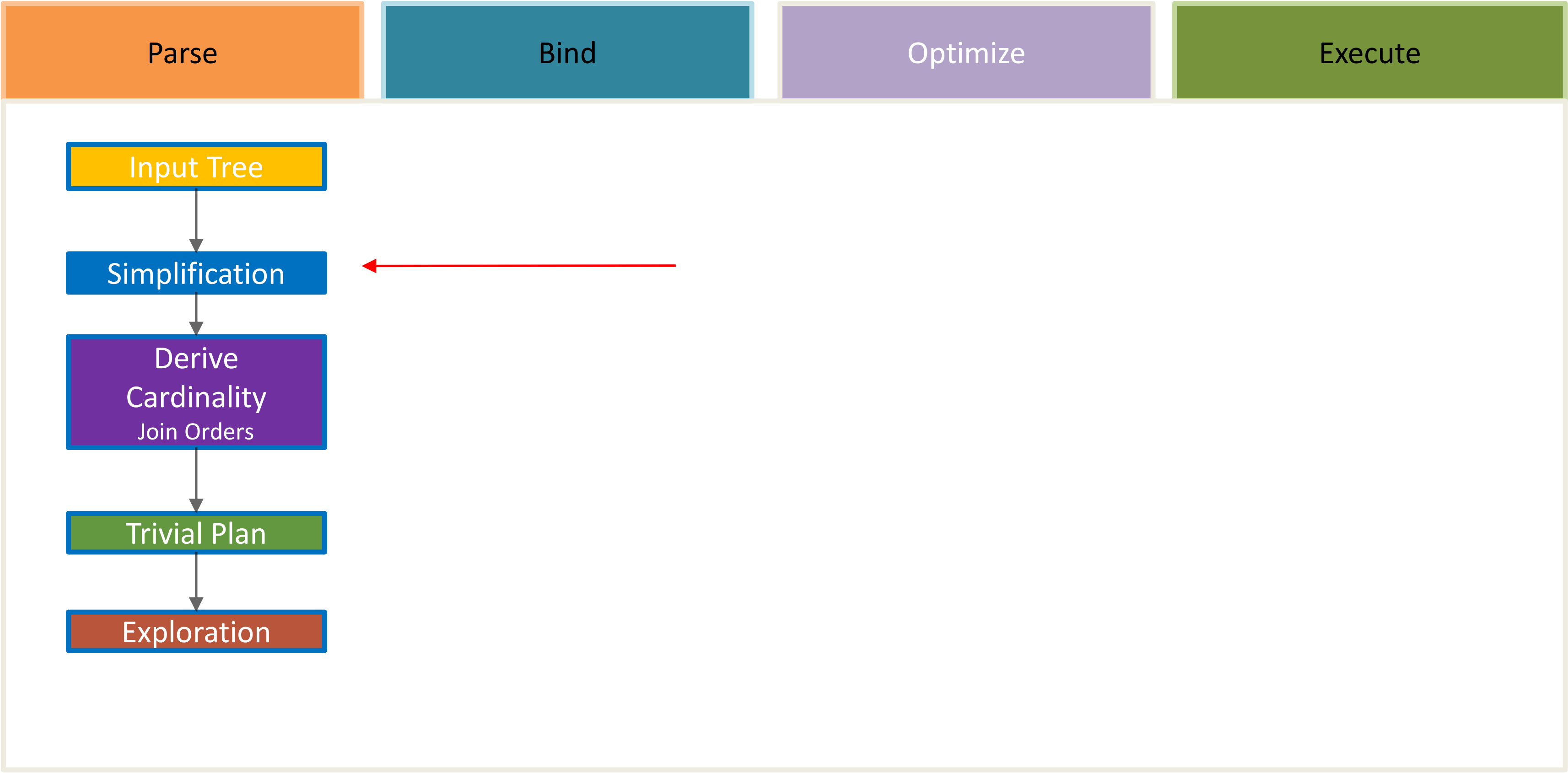
Use Undocumented TF 8605 QUERYTRACEON

And TF 8604

Relational Theory

```
        ,count(*)
    from
                     ts s
        inner join grades g
        on s.studentID=g.studentID
        g.grade='?'
        and s.LastName='whedon'
        and s.FirstName='Paul'
    group by s.LastName, g.grade option(maxdop 1, recompile, QUERYTRACEON 8605)
```

Relational Engine Query Process



Parse

Bind

Optimize

Execute

Input Tree

Simplification

Derive Cardinality

Join Orders

Trivial Plan

Exploration

Relational Engine Query Process

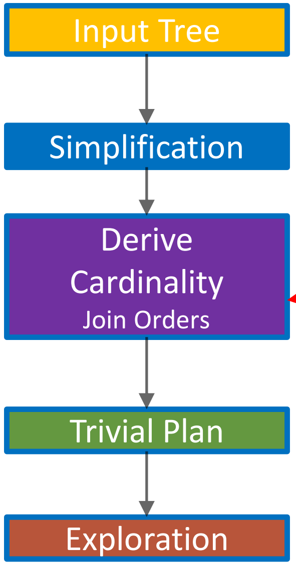
Parse | Bind | Optimize | Execute

Simplification

- Constant Folding
  - Evaluating an Expression during optimization instead of having to constantly do it during Execution
- Domain Simplification
  - Allows the Optimizer to “Reason” the Valid Values within a Range
- Predicate push-down
  - Read Rob Farley, eliminate rows up front
- Join Simplification
  - Removes Unnecessary Joins (Powerful option)
- Contradiction Detection
  - Prevents I/O Reads when predicates contradict one another

Relational Engine Query Process

Parse | Bind | Optimize | Execute



Input Tree

Simplification

Derive Cardinality
Join Orders

Trivial Plan

Exploration

Uses STATISTICS

Get's the number of rows in Tables to be able to "guess" the number of rows per table for a logical Tree

Used in order to determine Join Types

The more complex a query or it's Expressions the more difficult it is to provide a good Cardinality Estimate.

E.G. Scalar Functions, Views, Table Variables, CTE's

Relational Engine Query Process

Parse | Bind | Optimize | Execute

Derive Cardinality
Join Orders

Statistics

- Auto Update Statistics
- 20% of table plus 500 rows
- Not bad for small tables, but Really Bad for VLDB’s
- Filtered Indexes Require the Same Base Table Updates as Other Non-Clustered Indexes

Relational Engine Query Process

Parse | Bind | Optimize | Execute

Derive Cardinality
Join Orders

Statistics on VLDB's

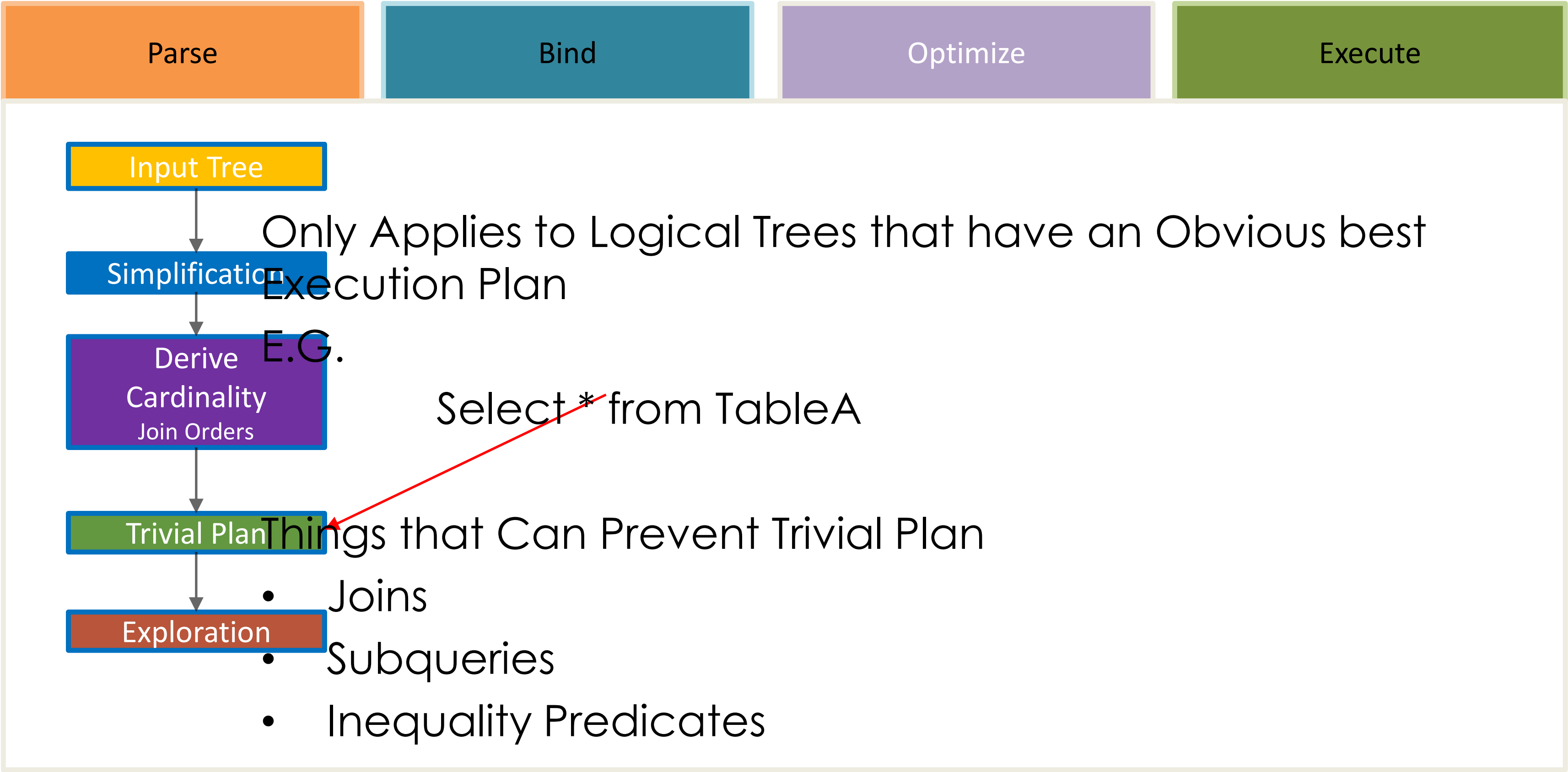
Trace Flag 2371

Table must have more than 25,000 Rows

Lowers the Threshold for Updates

100,000 = 10%
1,000,000 = 3.2%
10,000,000=1%
50,000,000=0.5%

Relational Engine Query Process



Only Applies to Logical Trees that have an Obvious best Execution Plan

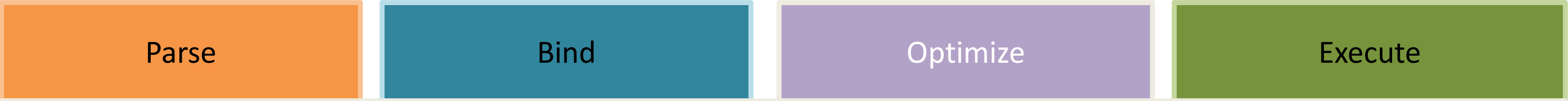
E.G.

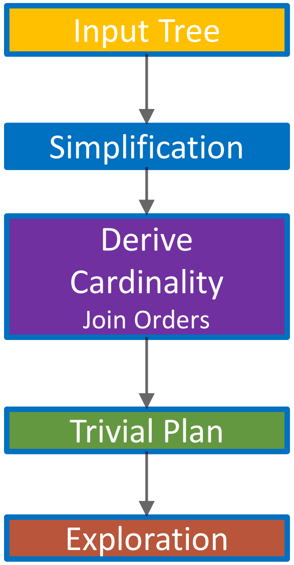
Select * from TableA

Things that Can Prevent Trivial Plan

- Joins
- Subqueries
- Inequality Predicates

Relational Engine Query Process





Search 0 – Transaction Processing

Simple Joins, not a lot of rows to return, has a cost of <0.2

Search 1 – Quick Plan (also known as Complex Query I)

Explore more rules and alternative joining orders. IF best plan cost less than 1.0 Use.

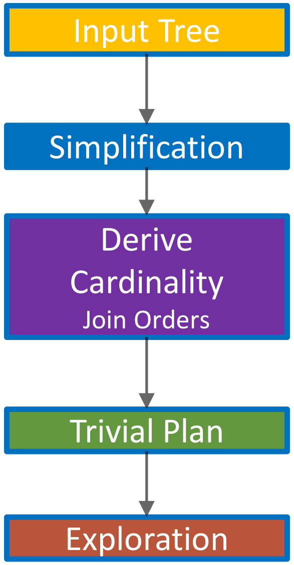
IF best cost plan > than 1.0 & MAXDOP > 1 & Serial Plan is > Cost Threshold for Parallelism then Generate a parallel plan compare to serial go with least cost.

Search 2 – Full Optimization

Explore all options, go with the cheapest plan after a limited number of operations

Relational Engine Query Process

Parse | Bind | Optimize | Execute



Input Tree

Simplification

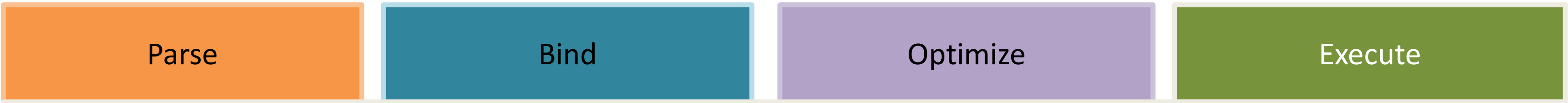
Derive Cardinality
Join Orders

Trivial Plan

Exploration

- Parse
- Algebrize
- Expand Views
- NNF convert
- Simplify
- Auto stats create /update
- Cardinality derivation
- Heuristic join reordering
- Project normalization
- Trivial Plan
- Exploration
- Copy Out
- Convert to executable Tree

Relational Engine Query Process



Parse

Bind

Optimize

Execute

Execution

- Physical Plan is Created
- The Memo is populated
- Plan is placed in the Plan Cache
- Interaction with the Storage Engine

Demo

What is Parallelism

A Process split across multiple CPU's/Cores

CAN speed up a plan

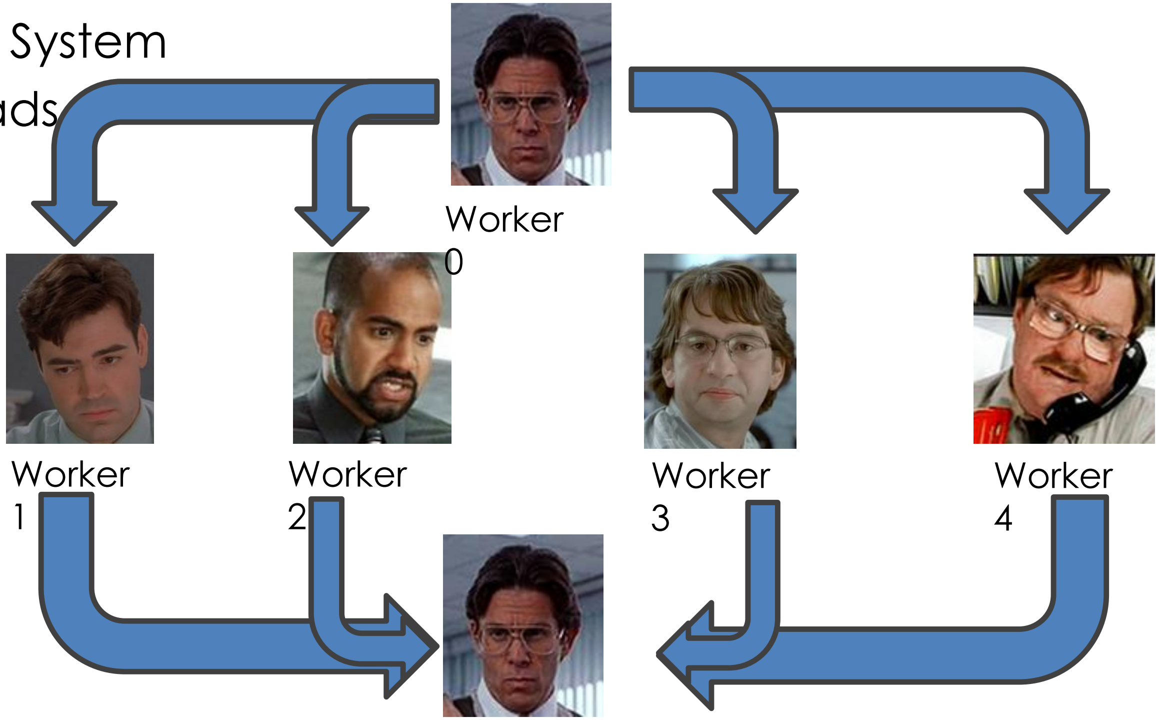
Optimizer determines Parallelism use based on

IF(Cost of Parallel Plan/Max DOP + Cost of Exchange Operators) < Cost of Serial Plan

What is Parallelism

4 Core System

5 Threads



Worker 0

Worker 1

Worker 2

Worker 3

Worker 4

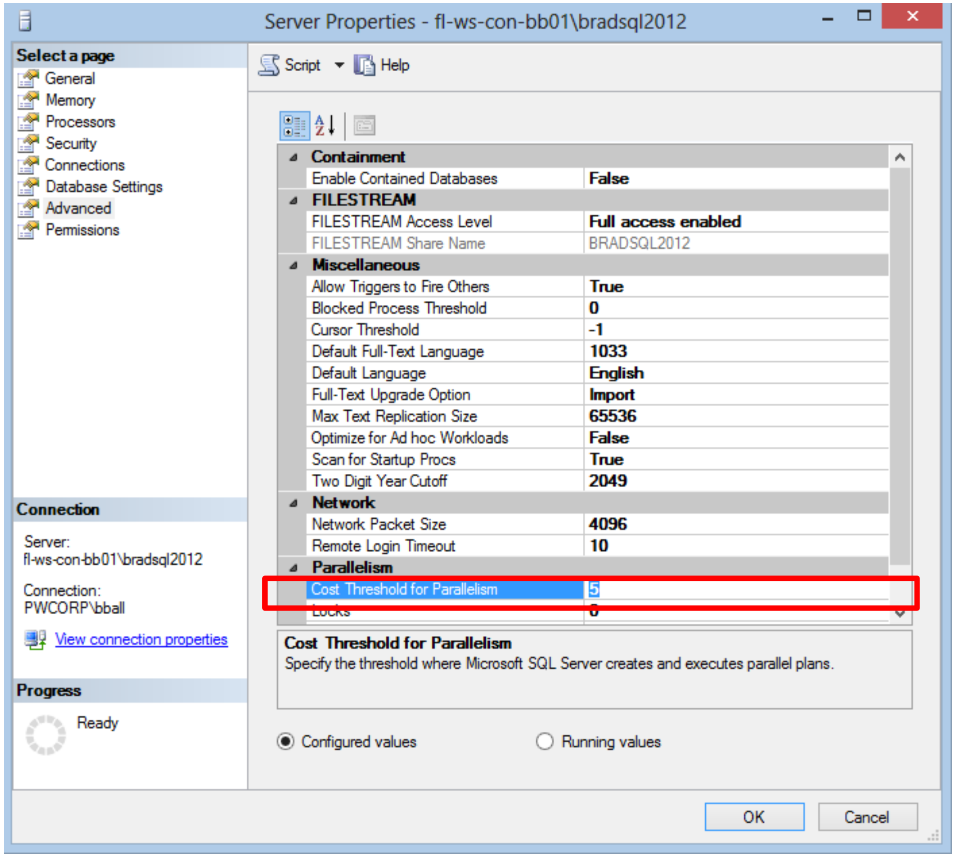
Worker 0

Cost Threshold for Parallelism

Default is 5

Has been 5 since 1997

Was the benchmark for someone on the Optimizer team at Microsoft.



Server Properties - fl-ws-con-bb01\bradsql2012

Containment	
Enable Contained Databases	False
FILESTREAM	
FILESTREAM Access Level	Full access enabled
FILESTREAM Share Name	BRADSQL2012
Miscellaneous	
Allow Triggers to Fire Others	True
Blocked Process Threshold	0
Cursor Threshold	-1
Default Full-Text Language	1033
Default Language	English
Full-Text Upgrade Option	Import
Max Text Replication Size	65536
Optimize for Ad hoc Workloads	False
Scan for Startup Procs	True
Two Digit Year Cutoff	2049
Network	
Network Packet Size	4096
Remote Login Timeout	10
Parallelism	
Cost Threshold for Parallelism	5
Locks	0

Cost Threshold for Parallelism
Specify the threshold where Microsoft SQL Server creates and executes parallel plans.

Connection

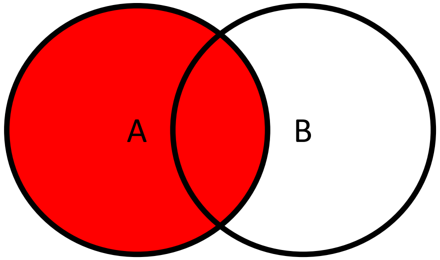
Server:
fl-ws-con-bb01\bradsql2012

Connection:
PWCORP\bball

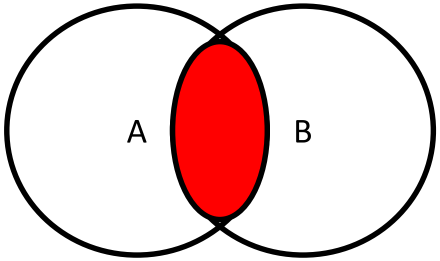
● Configured values ○ Running values

Demo

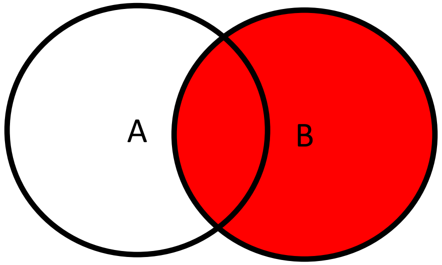
Syntactical Joins



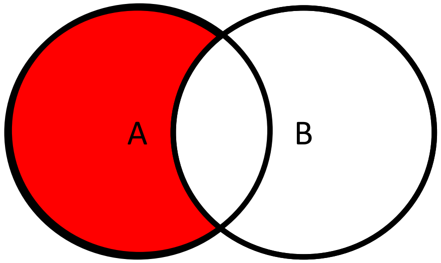
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
On A.Key=B.Key
```



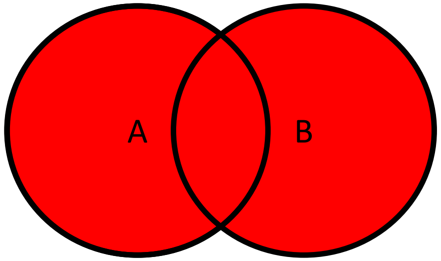
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
On A.Key=B.Key
```



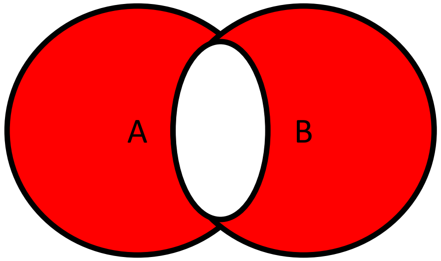
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
On A.Key=B.Key
```



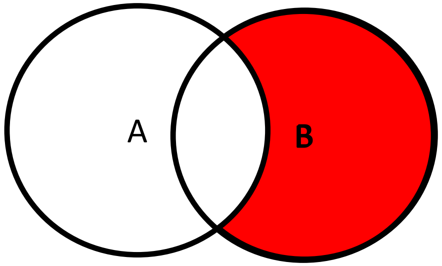
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
On A.Key=B.Key
WHERE B.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
On A.Key=B.Key
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
On A.Key=B.Key
Where A.Key IS NULL OR B.Key IS
NULL
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
On A.Key=B.Key
WHERE A.KEY IS NULL
```

Physical Operators

Joins

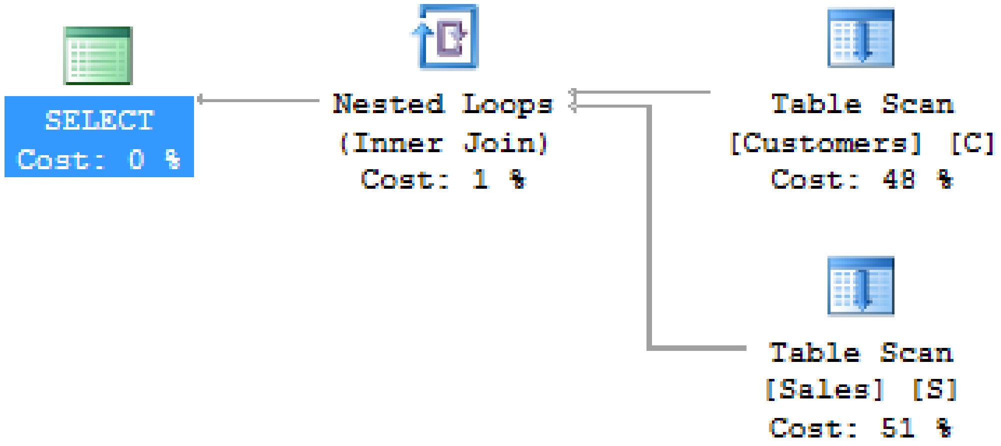
- Syntactical Joins
  - Inner Join
  - Outer Join
  - Cross Join
  - Cross Apply
  - Outer Apply
  - Semi-Join
  - Anti-Semi Join
- Physical Joins
  - Nested Loop Join
  - Merge Join
  - Hash Join

***If no Syntactical Join Type is specified Inner Join is the default**

Nested Loop Join

Compares Each Row from the Outer Table To Each Row in the Inner Table

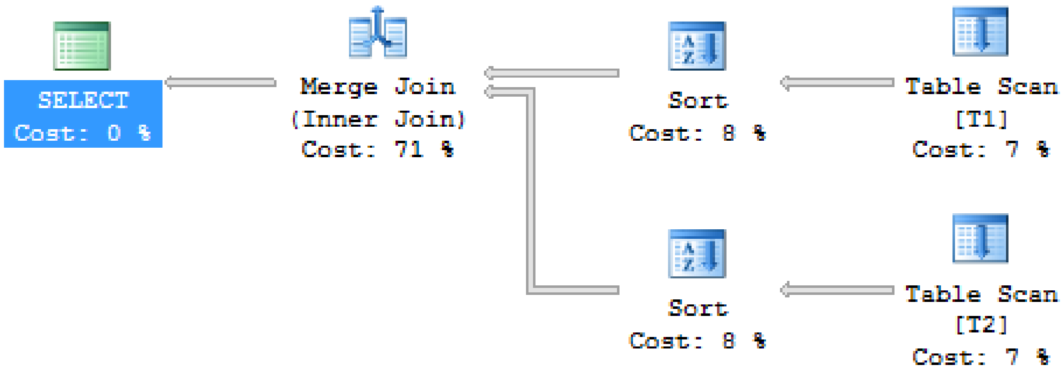
```
for each row R1 in the outer table
    begin
        for each row R2 in the inner table
            if R1 joins with R2
                return (R1, R2)
            if R1 did not join
                return (R1, NULL)

    end
```



Sorted Merge Join

Simultaneously Reads and Compares two Sorted Inputs One Row at a Time

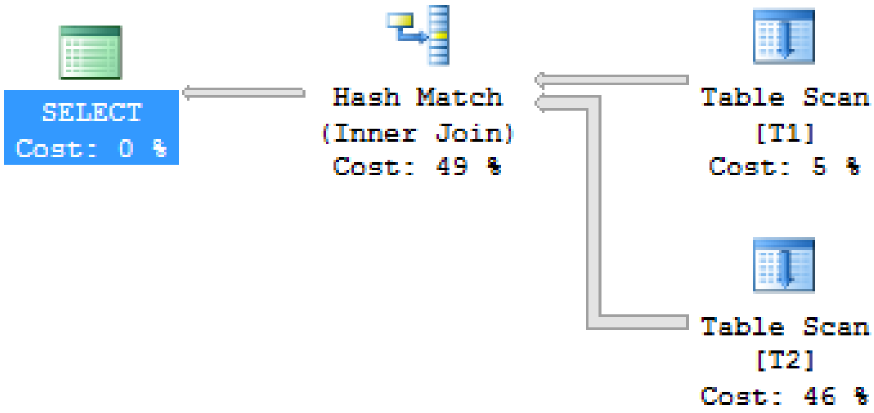
```
get first row R1 from input 1
get first row R2 from input 2
while not at the end of either input
   begin
      if R1 joins with R2
         begin
            return (R1, R2)
            get next row R2 from input 2
         end
      else if R1 < R2
         get next row R1 from input 1
      else
         get next row R2 from input 2
   end
```



Hash Join

Join Heavy Lifter. Built in Two Phases: Build & Probe.

- Build
  - Reads Rows from 1st Input
  - Hashes on Equijoin Keys
  - Creates In-Memory Hash Table



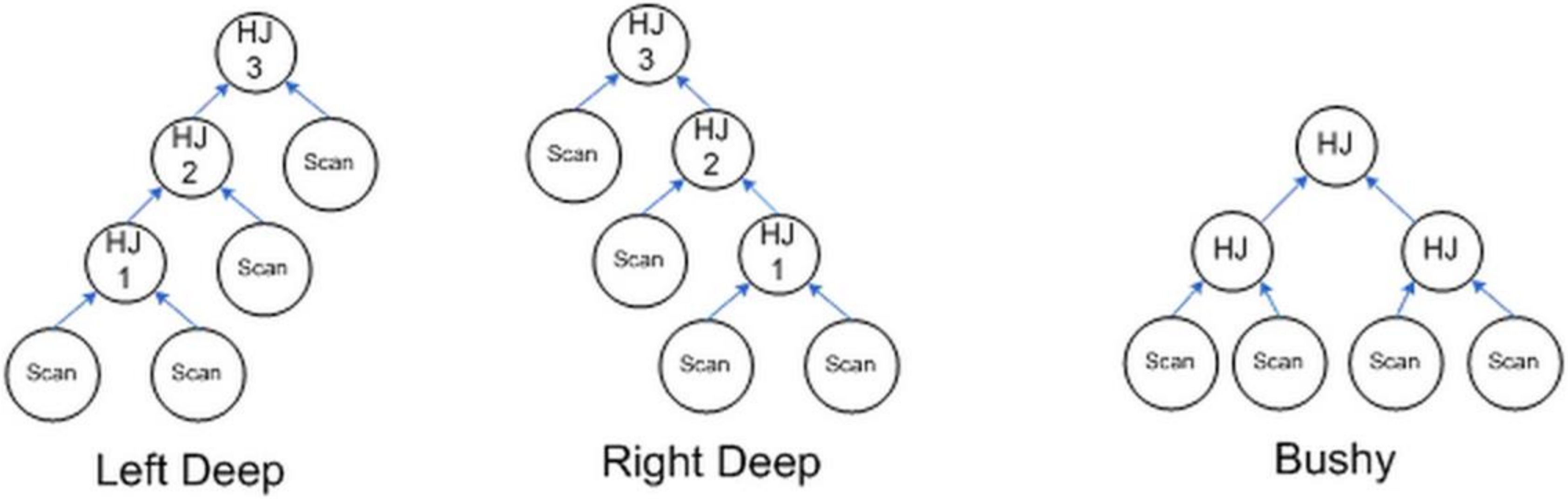
- Probe

  Reads All Rows from 2nd Input
  Hashes on Same Equijoin Keys
  Probes for Matching Rows in Hash Table

```
for each row R1 in the build table
  begin
    calculate hash value on R1 join key(s)
    insert R1 into the appropriate hash bucket
  end
for each row R2 in the probe table
  begin
    calculate hash value on R2 join key(s)
    for each row R1 in the corresponding hash bucket
      if R1 joins with R2
        return (R1, R2)
  end
```

Hash Join

Has 3 Types of Tree Structures



Left Deep

Right Deep

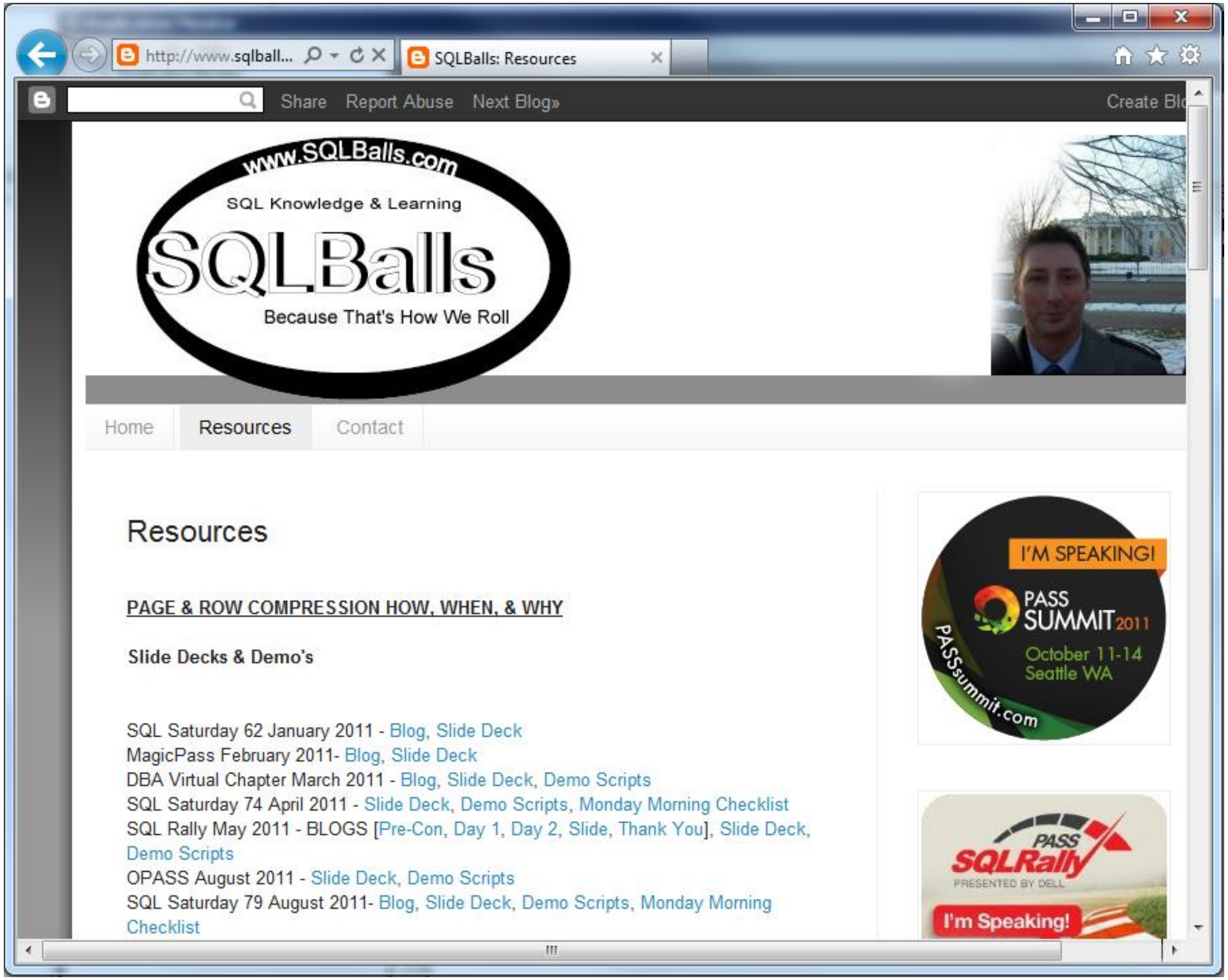
Bushy

Reasons to Join

- Extra Columns
- Eliminating Rows
- Duplicating Rows
- Introducing Nulls

Demo

Resources



Resources

PAGE & ROW COMPRESSION HOW, WHEN, & WHY

Slide Decks & Demo's

SQL Saturday 62 January 2011 - Blog, Slide Deck
MagicPass February 2011- Blog, Slide Deck
DBA Virtual Chapter March 2011 - Blog, Slide Deck, Demo Scripts
SQL Saturday 74 April 2011 - Slide Deck, Demo Scripts, Monday Morning Checklist
SQL Rally May 2011 - BLOGS [Pre-Con, Day 1, Day 2, Slide, Thank You], Slide Deck, Demo Scripts
OPASS August 2011 - Slide Deck, Demo Scripts
SQL Saturday 79 August 2011- Blog, Slide Deck, Demo Scripts, Monday Morning Checklist

Questions?