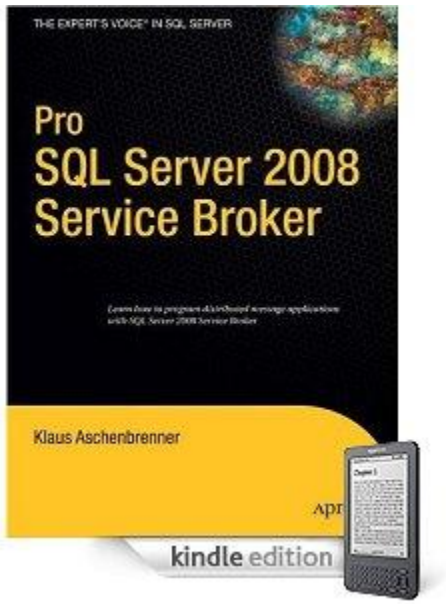
SQL Server Query Tuning

Klaus Aschenbrenner
Independent SQL Server Consultant
SQLpassion.at
Twitter: @Aschenbrenner

About me

- Independent SQL Server Consultant
- International Speaker, Author
- „Pro SQL Server 2008 Service Broker“
- SQLpassion.at
- Twitter: @Aschenbrenner



SQL Server Performance & Troubleshooting Workshop

- 19. – 21. Juni in Bern
  - Gurten
- Agenda
  - SQL Server Indexing
  - SQL Server Locking & Blocking
  - SQL Server Performance Monitoring & Troubleshooting
- Weitere Informationen
  - http://www.SQLpassion.at/events.html

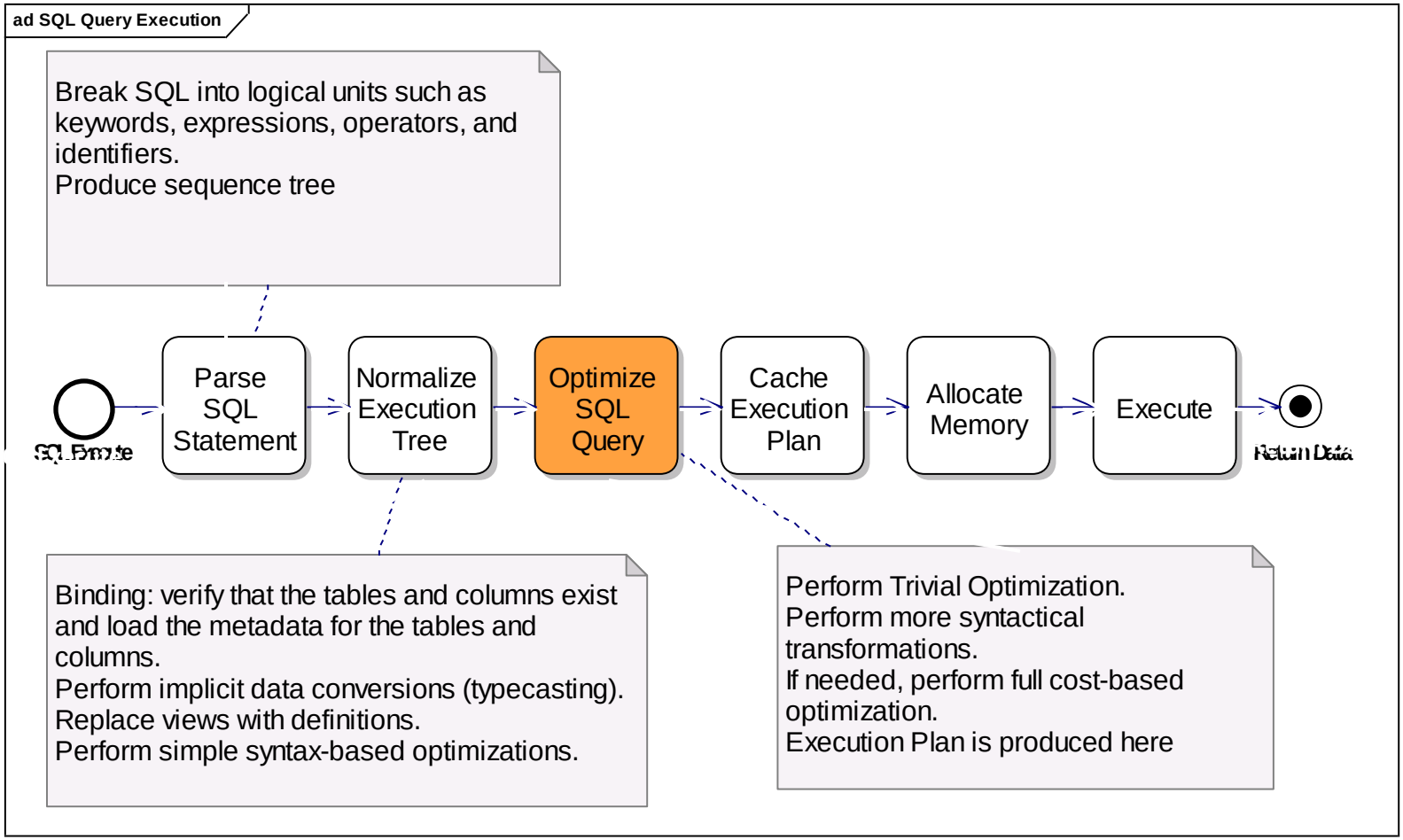
Agenda

- Basics of Query Execution
- Execution Plan Overview
- Index Tuning

Agenda

- **Basics of Query Execution**
- Execution Plan Overview
- Index Tuning

Execution of a query



Query Optimization

- Execution Plans and cost-based optimizations
- Optimization phases
- Indexes and distribution statistics
- Join Selection

Execution Plan

- Strategy determined by the Optimizer to access/manipulate data
  - Can be influenced by the developer – query hints
- Key decisions are made
  - Which indexes to use?
  - How to perform JOIN operations?
  - How to order and group data?
  - In what order tables should be processed?
  - Can be cached plans reused?

Agenda

- Basics of Query Execution
- **Execution Plan Overview**
- Plan Cache
- Plan Caching
- Parameter Sniffing
- Recompilations

Why understanding Execution Plans?

- Insight into query execution/processing strategy
- Tune performance problems at the source
  - Hardware is not every time the problem
  - High CPU/IO consumption -> poorly tuned Execution Plans
- Understanding Execution Plans is a prerequisite to performance tuning!

Execution Plan Types 1/2

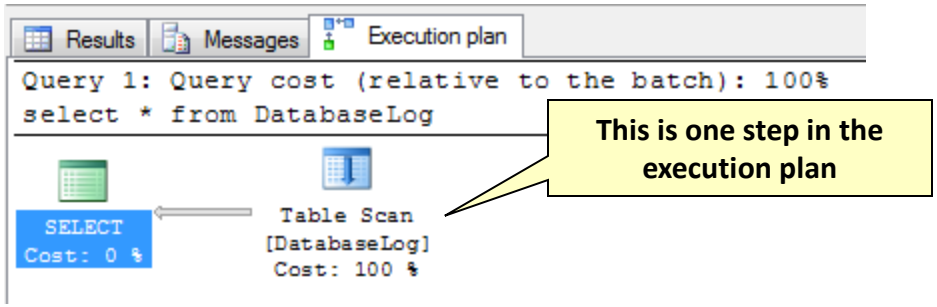
- Estimated Execution Plan
  - Created without ever running the query
  - Uses statistics for estimation
  - Good for long running query tuning
- Actual Execution Plan
  - Created when the actual query runs
  - Uses the real data
- They can be different
  - Statistics out of date
  - Estimated Execution Plan not valid any more

Execution Plan Types 2/2

- Textual Execution Plan
  - Depricated in further versions of SQL Server
- XML based Execution Plan
  - Very good for further analysis
  - Can be queried
- Graphic Execution Plan
  - Uses internally the XML based Execution Plan

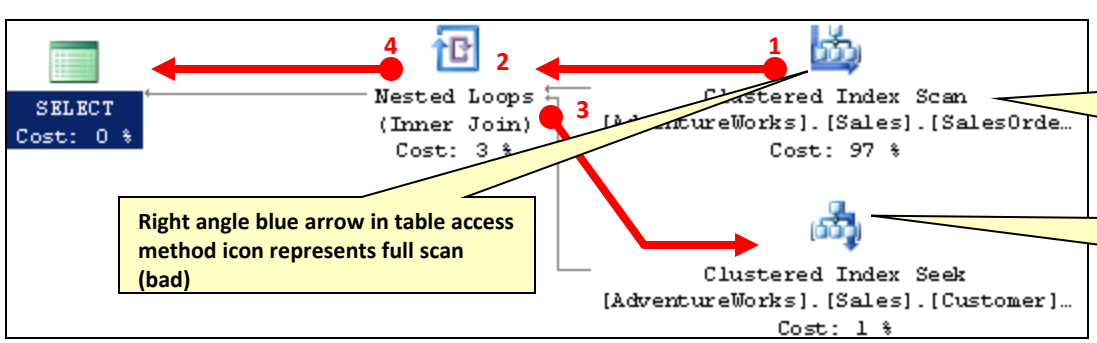
Reading Execution Plans

- SHOWPLAN permission needed
- Query execution is serial
  - A series of sequential steps/operators
  - Executed one after one
- Execution Plan displays these steps/ operators



Execution Plan Sample

```
select c.CustomerID
     , soh.SalesOrderID
     , soh.OrderDate
from Sales.Customer c
join Sales.SalesOrderHeader soh on c.CustomerID = soh.CustomerID
where soh.OrderDate > '20050101'
```



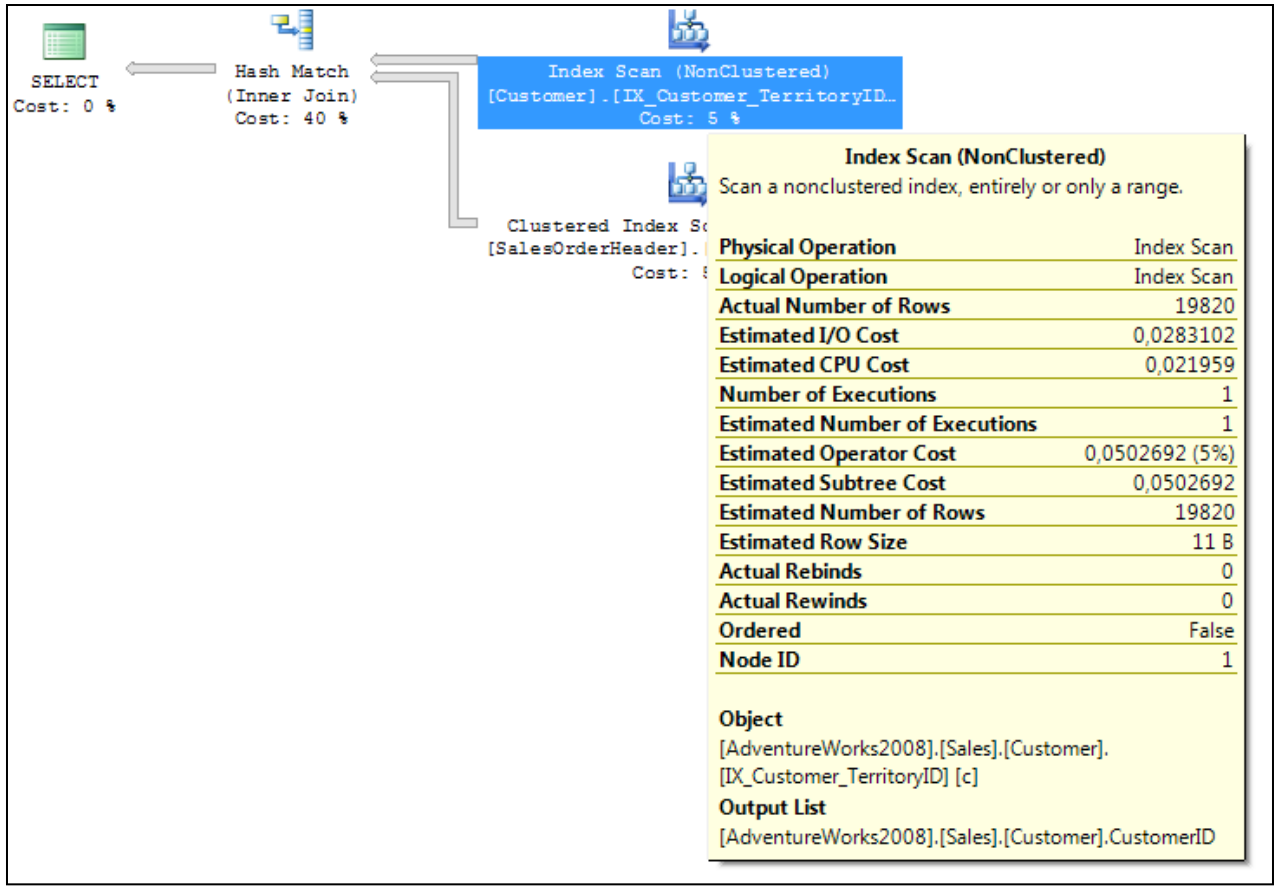
Note: plan starts at [SalesOrderHeader] even though [Customer] is actually named first in query expression

Right angle blue arrow in table access method icon represents full scan (bad)

Stepped blue arrow in table access method icon represents index "seek", but could be either a single row or a range of rows

Operator Properties

- Each operator has several properties with additional information



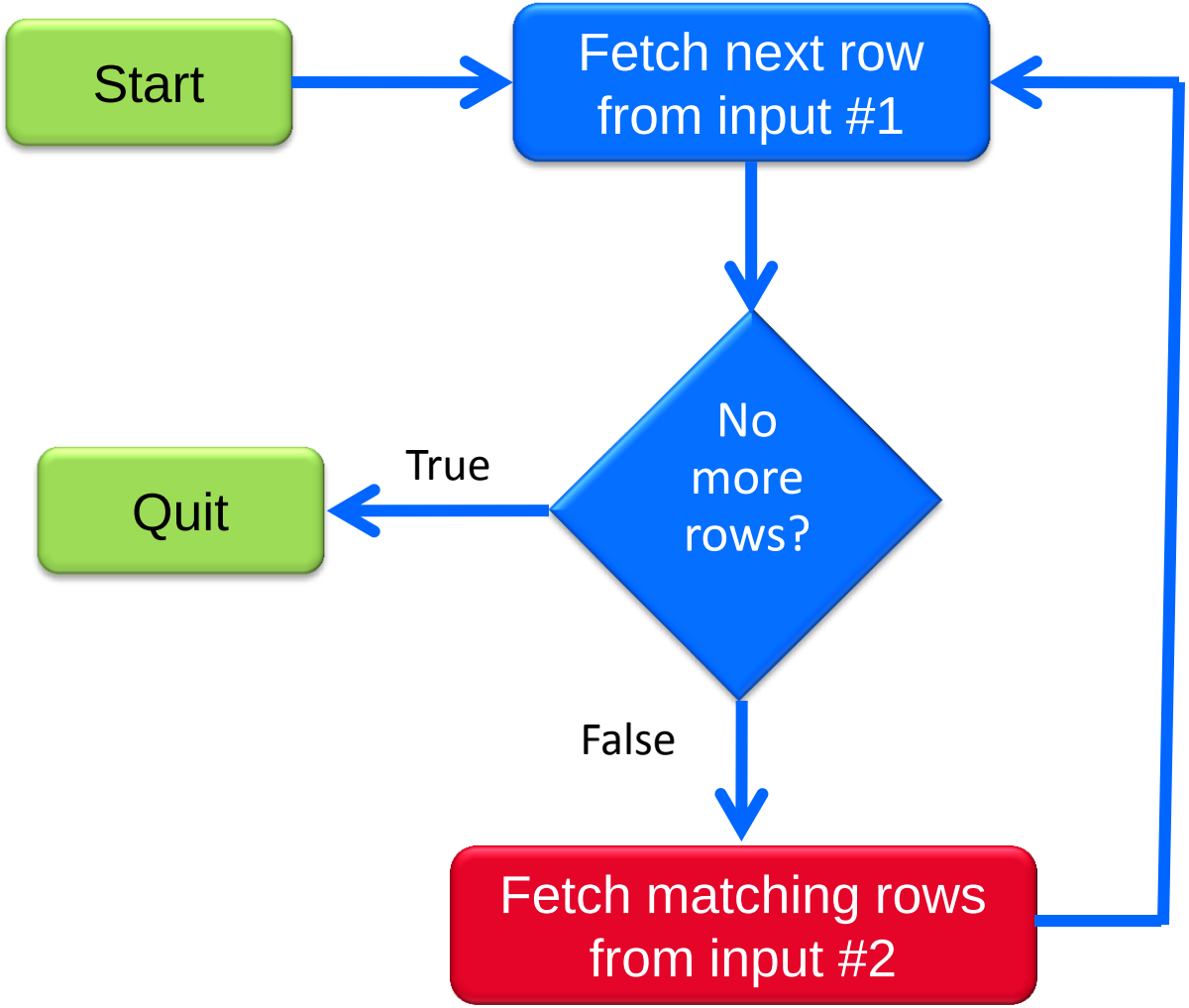
Common Properties

- Physical Operation
- Logical Operation
- Estimated I/O Cost
- Estimated CPU Cost
- Estimated Operator Cost
- Estimated Subtree Cost
- Estimated Number of Rows
- Estimated Row Size

Common Operators

- Data Retrieval Operators
  - Table Scan (reads a whole table)
  - Index Scan (reads a whole index)
  - Index Seek (seeks into a index)
- Join Operators
  - Nested Loop (outer loop, inner loop)
  - Merge Join (needs sorted input)
  - Hash Match (uses a hash table internally)
- Aggregation Operators
  - Stream Aggregate
  - Hash Aggregate

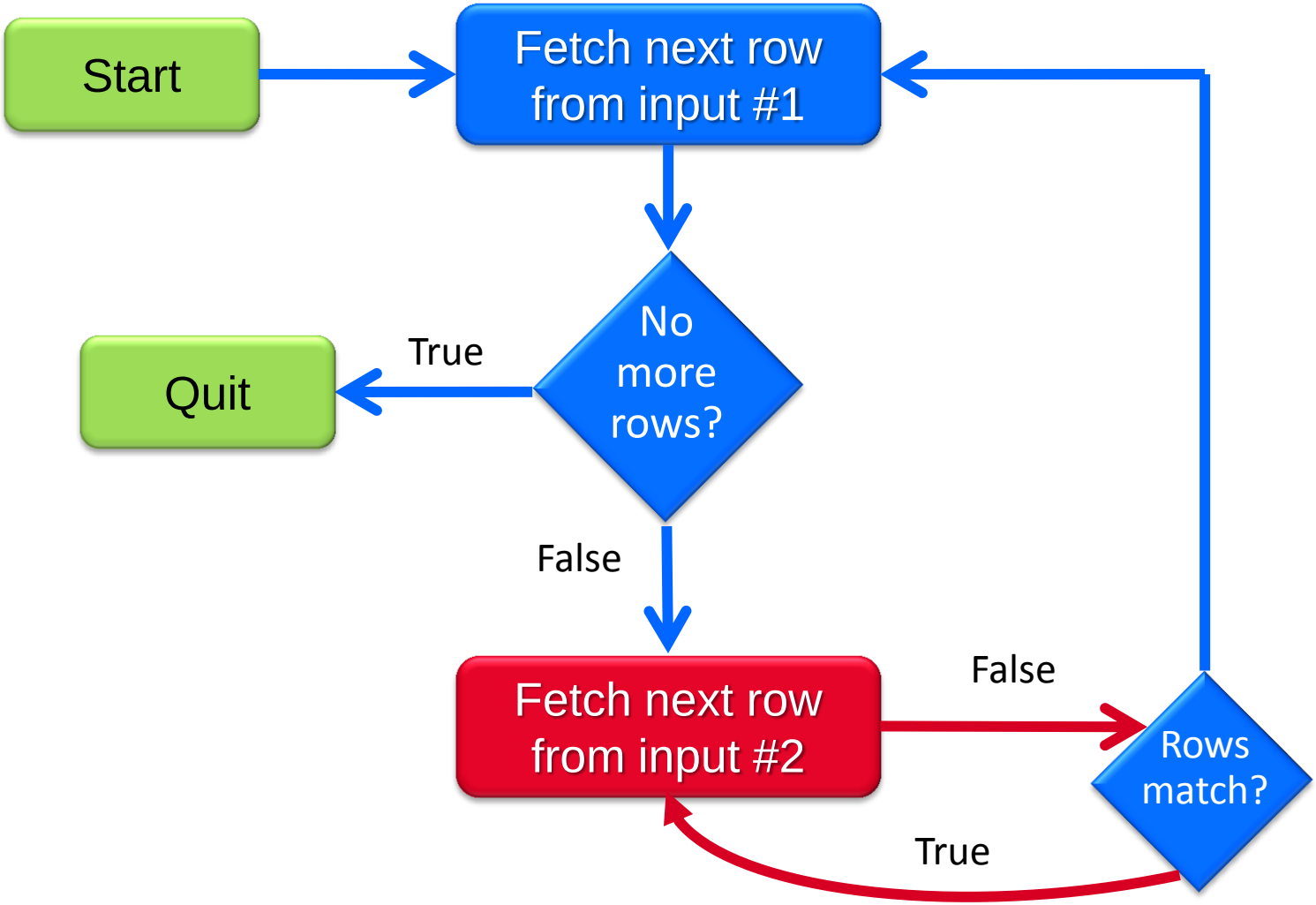
Nested Loop Join



Nested Loop

- „For each Row“ operator
- Takes output from one step and executes another operation „for each“ output row
- Outer Loop, Inner Loop
- Only join type that supports inequality predicates

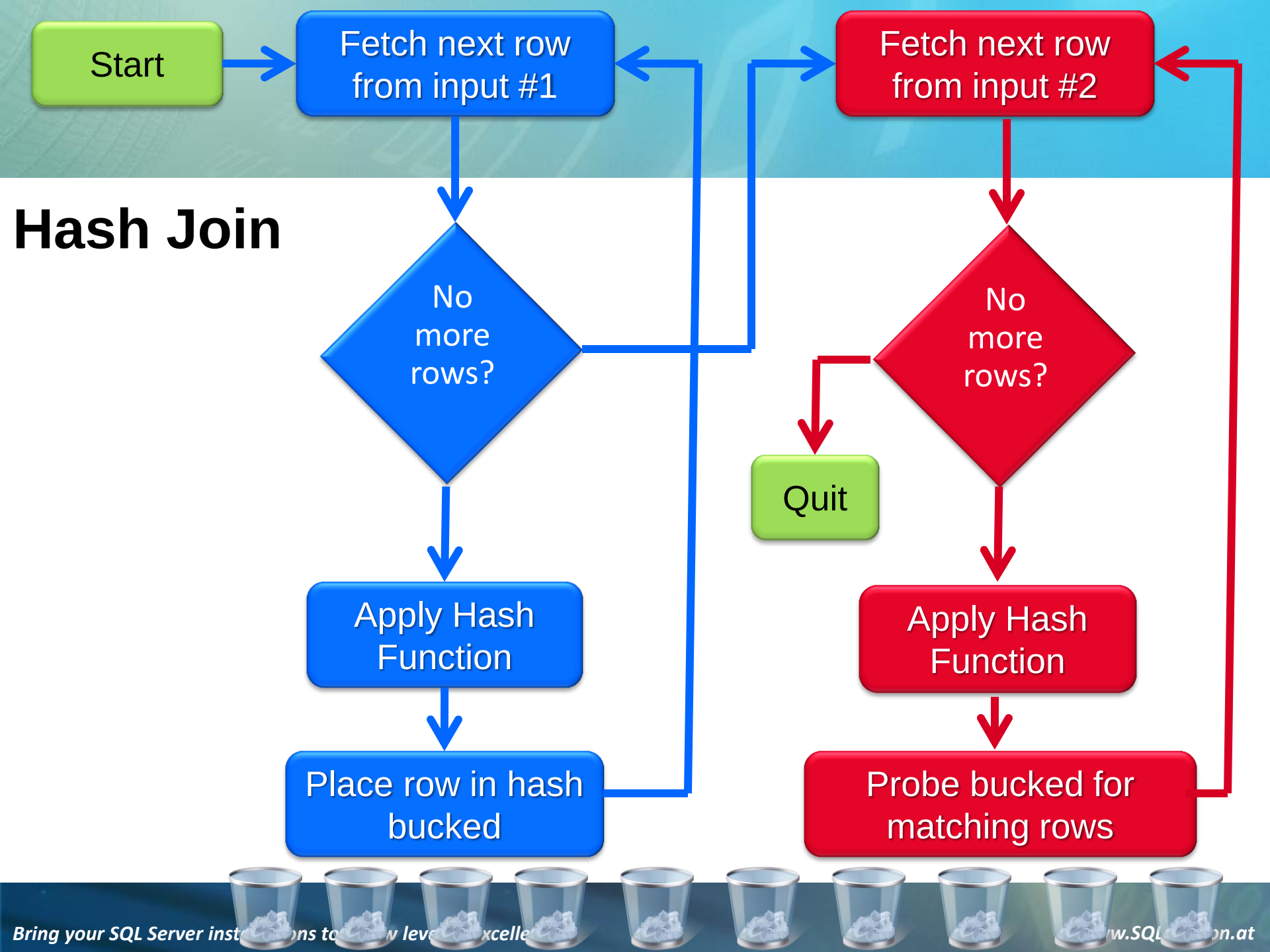
Merge Join


Start
Fetch next row from input #1
No more rows?
True
Quit
False
Fetch next row from input #2
False
Rows match?
True

Merge Join

- Needs at least one equijoin predicate
- Used when joined columns are indexed (sorted)
- Otherwise (expensive) sorting is needed
  - Plan may include explicit sort

Hash Join


Start
Fetch next row from input #1
Fetch next row from input #2
No more rows?
No more rows?
Quit
Apply Hash Function
Apply Hash Function
Place row in hash bucked
Probe bucked for matching rows

Hash Join

- Needs at least one equijoin predicate
- Hashes values of join columns from one side (smaller table)
  - Based on Statistics
- Probes them with the join columns of the other side (larger table)
- Uses hash buckets
- Stop and Go for the Probe Phase
- Needs memory grants to build the hash table
  - If they exceed, Hash Join is spilled to TempDb
  - Performance decreases!

Stream Aggreate

- Data must be sorted on the aggregation columns
- Processes groups one at a time
- Does not block or use memory
- Efficient if sort order is provided by index
  - Plan may include explicit sort

Hash Aggregate

- Data need not be sorted
- Builds a hash table of all groups
- Stop and Go
- Same as Hash Join
  - Needs memory grants to build the hash table
  - If they exceed, Hash Aggregate is spilled to TempDb
  - Performance decreases!
- General better for larger input sets

Capturing Execution Plans

- SQL Server Management Studio
  - Development
- SQL Profiler
  - Production
  - Event: „Performance/Showplan XML“
  - Dump Execution Plan to file
- sys.dm_exec_cached_plans
  - Production

Demo

Working with Execution Plans

Agenda

- Basics of Query Execution
- Execution Plan Overview
- **Index Tuning**
  - Search Arguments
  - Covering Index
  - Tipping Point
  - Index Intersection
  - Index Joins
  - Filtered Indexes
  - Indexed Views

General

- SET STATISTICS IO ON/OFF
  - Returns the IO page count
- SET STATISTICS TIME ON/OFF
  - Returns the elapsed time
- Index improves read performance
- Index degrades write performance
  - Index maintenance
  - But it can also improve performance, because the record to be updated/deleted must be found in the first step (SELECT)

Search Arguments

- Can be part of
  - WHERE clause
  - JOIN clause
- Has major impact on Index usage
  - Breaks performance
- Wrong usage can lead to SCANs instead of SEEKs
- Recommendations
  - No functions/expressions on Search Arguments

Demo

Search Arguments

Bookmark Lookup

- Occurs when query is not satisfied by a Covering Index
  - Base table must be touched
  - Index Page + Data Page access occurs
- 2 kinds of Bookmark Lookup
  - Clustered Key Lookup (Clustered Table)
  - RID Lookup (Heap Table)
- Deadlocks are possible
  - Writer: X(CI) => X(NCI)
  - Reader: S(NCI) => S(CI)

Covering Index

- Non-Clustered Index that satisfies a complete query
  - Without touching the base table
  - No access to the data page containing the data
- The following referred columns must be present in the Non-Clustered index
  - SELECT
  - WHERE
- Otherwise
  - Traditional „Bookmark Lookup“

INCLUDE Property

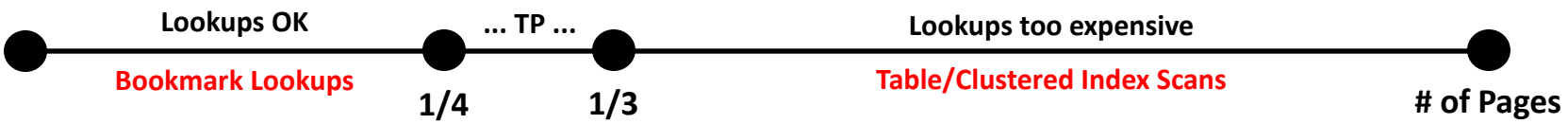
- Includes columns in the leaf-pages of the index
- Columns are not included in the navigation hierarchy
- Navigation hierarchy is very small
- Benefits
  - More elegant covering index
  - More than 900 bytes per index possible
  - More than 16 columns per index possible

Demo

Covering Index

Tipping Point

- Defines whether to do a Bookmark Lookup or a Table/Clustered Index Scan
- Bookmark Lookups are only done when the Non-Clustered Index is selective enough
- Covering Index will not have a Tipping Point



Tipping Point Sample

- 80.000 records per 4.000 bytes
  - 12,5% - 16,7% records
- 80.000 records per 400 bytes
  - 1,25% - 1,67% records
- 80.000 records per 40 bytes
  - 0,125% - 0,167% records
- Doesn‘t depend on the number of records
- Depends on the size of the records!!!

Demo

Tipping Point

Index Intersection

- SQL Server can use multiple indexes to execute a query
  - Selecting small subsets of data
  - Finally performing an intersection between those subsets
- Why
  - No chance to modify existing indexes
  - Already too much columns/bytes in the Non-Clustered Index
- Create multiple narrow indexes, instead of wide indexes
  - Reuse through index intersection is possible!

Disadvantages

- Not the fastest option for performance
  - More than 1 index must be accessed
  - Join(s) necessary
  - Hash Join needs (maybe) space in TempDb
- Not recommended for high priority queries

Demo

Index Intersection

Filtered Index

- Specialized version of Non-Clustered Index
  - WHERE clause
  - Creates a highly selective set of keys
  - Reduces the size (page count) of the Non-Clustered Index and therefore the query performance
  - Decreased maintenance cost, because index is smaller
- E.g.
  - Column with large amount of NULL values

Filtered Indexes - Prerequisites

- ANSI settings ON
  - ANSI_NULLS
  - ANSI_PADDING
  - ANSI_WARNING
  - ARITHABORT
  - CONCAT_NULL_YIELDS_NULL
  - QUOTED_IDENTIFIER
- ANSI settings OFF
  - NUMERIC_ROUNDABORT

Demo

Filtered Index

Indexed Views

- Views
  - Stores only the SELECT statement (virtual table)
  - Contains no data
  - Data is retrieved by every call again and again
  - Poor performance if you have to do a lot of calculations (like aggregations)
- Indexed Views
  - Unique Clustered Index on a view
  - The data in the view in materialized into the index
  - Index is stored physically in the database file

Indexed Views - Restrictions

- First index must be a unique Clustered Index
- Non-Clustered Index after Clustered Index
- View definition must be deterministic
- No reference to other views
- Float columns can‘t be the Clustered Key
- View must be schema-bound

Indexed Views - Prerequisites

- ANSI settings ON
  - ARITHABORT
  - CONCAT_NULL_YIELDS_NULL
  - QUOTED_IDENTIFIER
  - ANSI_NULLS
  - ANSI_PADDING
  - ANSI_WARNING
- ANSI settings OFF
  - NUMERIC_ROUNDABORT

Demo

Indexed Views

Indexing for OR

- OR returns individual sets
  - Ensures that rows that appears in multiple sets are only returned once
- IN is a simplified version of OR conditions
- Each column must be indexed
  - Condition must be selective enough (regarding Tipping Point)

Demo

Indexing for OR

Indexing for Aggregates

- 2 Types of Aggregates
- Stream Aggregate
  - Needs sorted input on the aggregated columns
- Hash Aggregate
  - Done otherwise, when input is not sorted
  - Can be spilled out to TempDb
  - Can lead to TempDb overhead/problems
- Aggregation on multiple columns
  - All columns must be indexed through ONE index

Demo

Indexing for Aggregates

Summary

- Basics of Query Execution
- Execution Plan Overview
- Index Tuning

SQL Server Performance & Troubleshooting Workshop

- 19. – 21. Juni in Bern
  - Gurten
- Agenda
  - SQL Server Indexing
  - SQL Server Locking & Blocking
  - SQL Server Performance Monitoring & Troubleshooting
- Weitere Informationen
  - http://www.SQLpassion.at/events.html