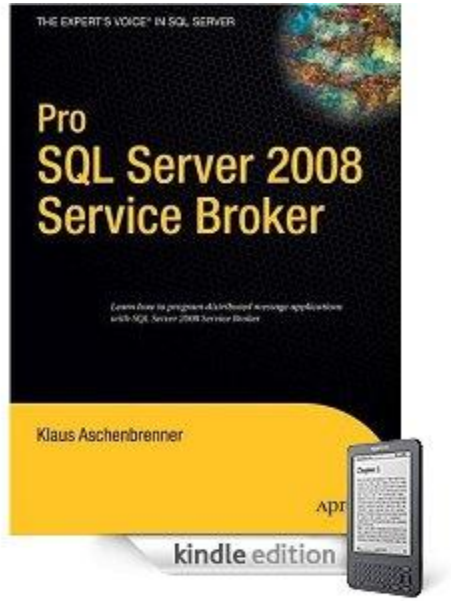
Understanding SQL Server Execution Plans

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

Bring your SQL Server installations to a new level of excellence!

About me

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



Flightdeck Breitenlee

- Based on Microsoft Flight Simulator X
- 6 PCs in a network
- Around 2km cables
- Projection
  - Fully 180 degree curved project surface
  - 6 x 2m Display
  - 3 Beamers
  - 3072 x 768 Pixel
- Get your boarding pass here
  - http://www.flightdeck-breitenlee.at

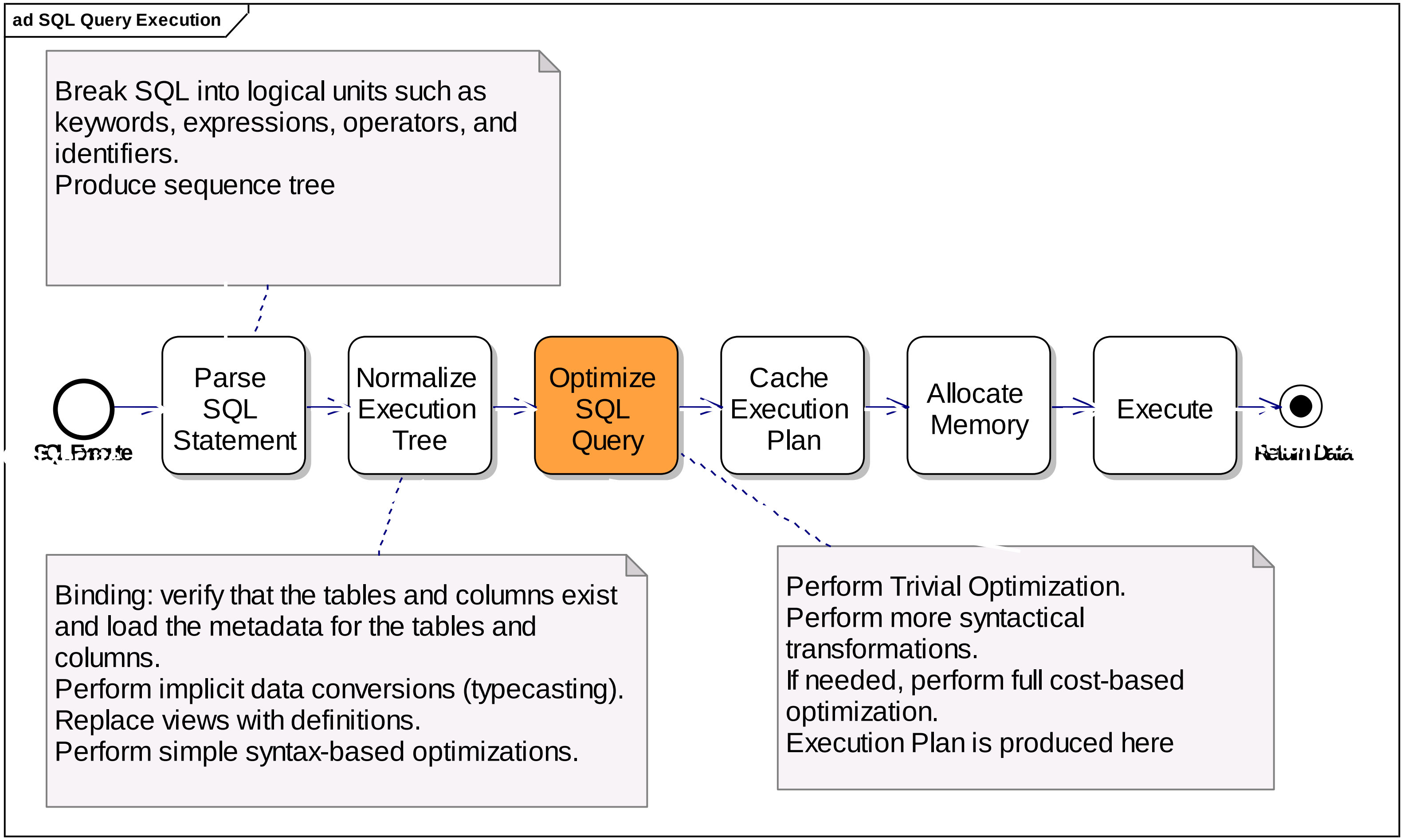
Agenda

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

Agenda

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

Execution of a query



ad SQL Query Execution

Break SQL into logical units such as keywords, expressions, operators, and identifiers.
Produce sequence tree

SQL Execute

Parse SQL Statement

Normalize Execution Tree

Optimize SQL Query

Cache Execution Plan

Allocate Memory

Execute

Return Data

Binding: verify that the tables and columns exist and load the metadata for the tables and columns.
Perform implicit data conversions (typecasting).
Replace views with definitions.
Perform simple syntax-based optimizations.

Perform Trivial Optimization.
Perform more syntactical transformations.
If needed, perform full cost-based optimization.
Execution Plan is produced here

Stored Procedure vs. Raw SQL

- Stored Procedure
  - Execution Plan is cached
  - Better security
  - No way to inject SQL
  - Needs to be recompiled from time to time
- Raw SQL
  - Compiled EVERY time
  - No security isolation
  - SQL injection sometimes possible (depends on the developer...)

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

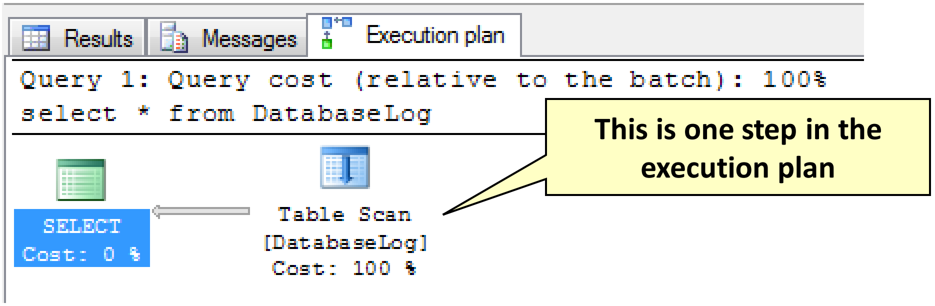
- Estimated Execution Plan
  - Created without ever running the query
  - Uses statistics for estimation
  - Good for long running query tuning
- Actual Execution Plan
  - Created when the actual query runs
  - Uses the real data
- They can be different
  - Statistics out of date
  - Estimated Execution 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

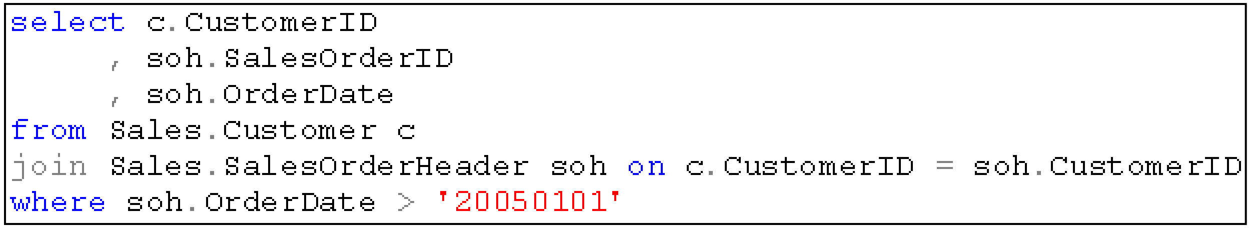
- SHOWPLAN permission needed
- Query execution is serial
  - A series of sequential steps/operators
  - Executed one after one
- Execution Plan displays these steps/ operators



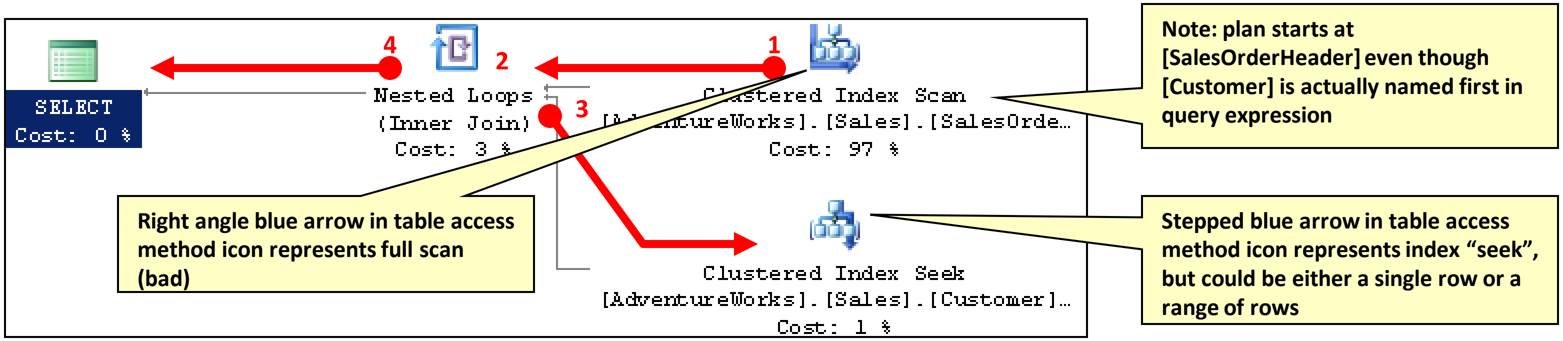
```
Query 1: Query cost (relative to the batch): 100%
select * from DatabaseLog
```

This is one step in the execution plan

Execution Plan Sample



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



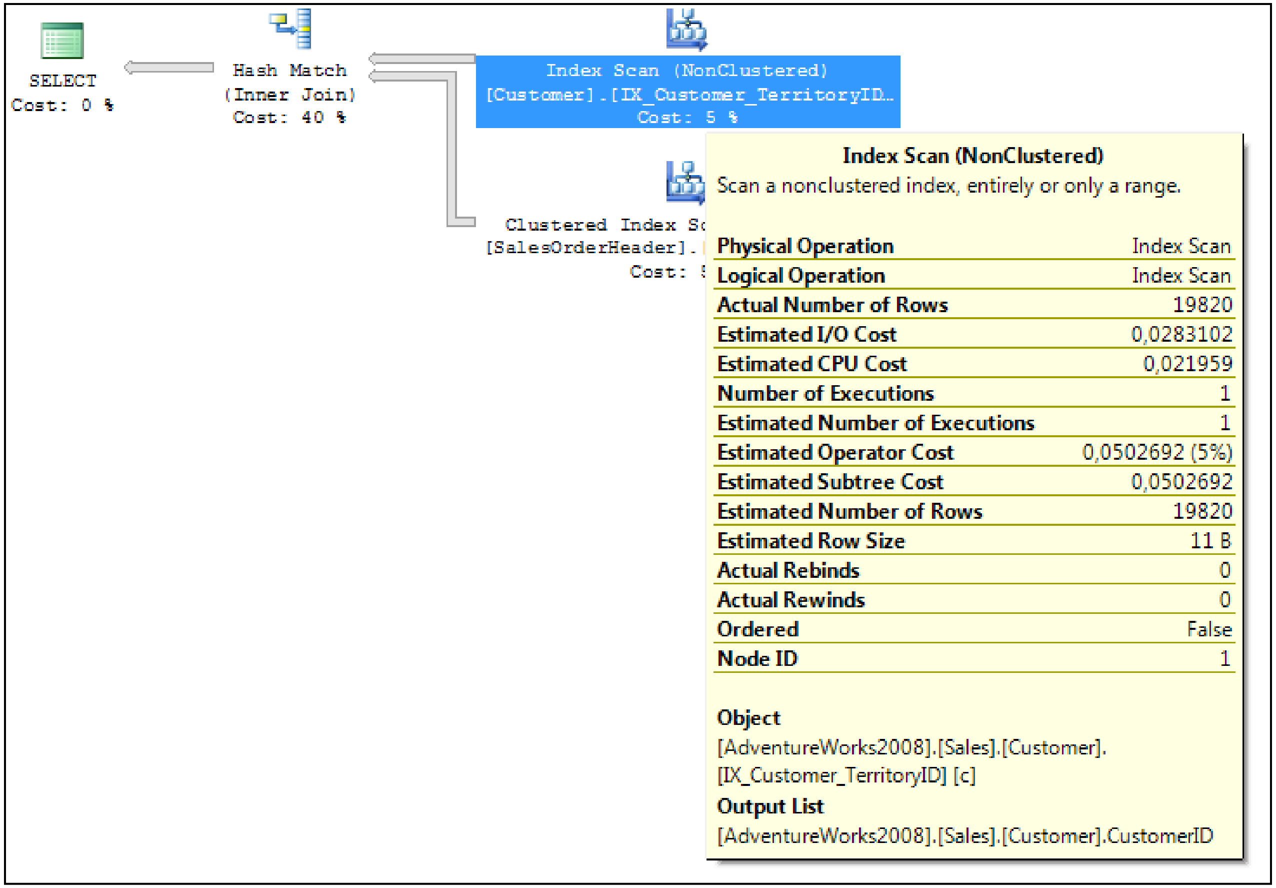
Note: plan starts at [SalesOrderHeader] even though [Customer] is actually named first in query expression

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



Index Scan (NonClustered)

Scan a nonclustered index, entirely or only a range.

Physical Operation	Index Scan
Logical Operation	Index Scan
Actual Number of Rows	19820
Estimated I/O Cost	0,0283102
Estimated CPU Cost	0,021959
Number of Executions	1
Estimated Number of Executions	1
Estimated Operator Cost	0,0502692 (5%)
Estimated Subtree Cost	0,0502692
Estimated Number of Rows	19820
Estimated Row Size	11 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	1

Object

[AdventureWorks2008].[Sales].[Customer].
[IX_Customer_TerritoryID] [c]

Output List

[AdventureWorks2008].[Sales].[Customer].CustomerID

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

- „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

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

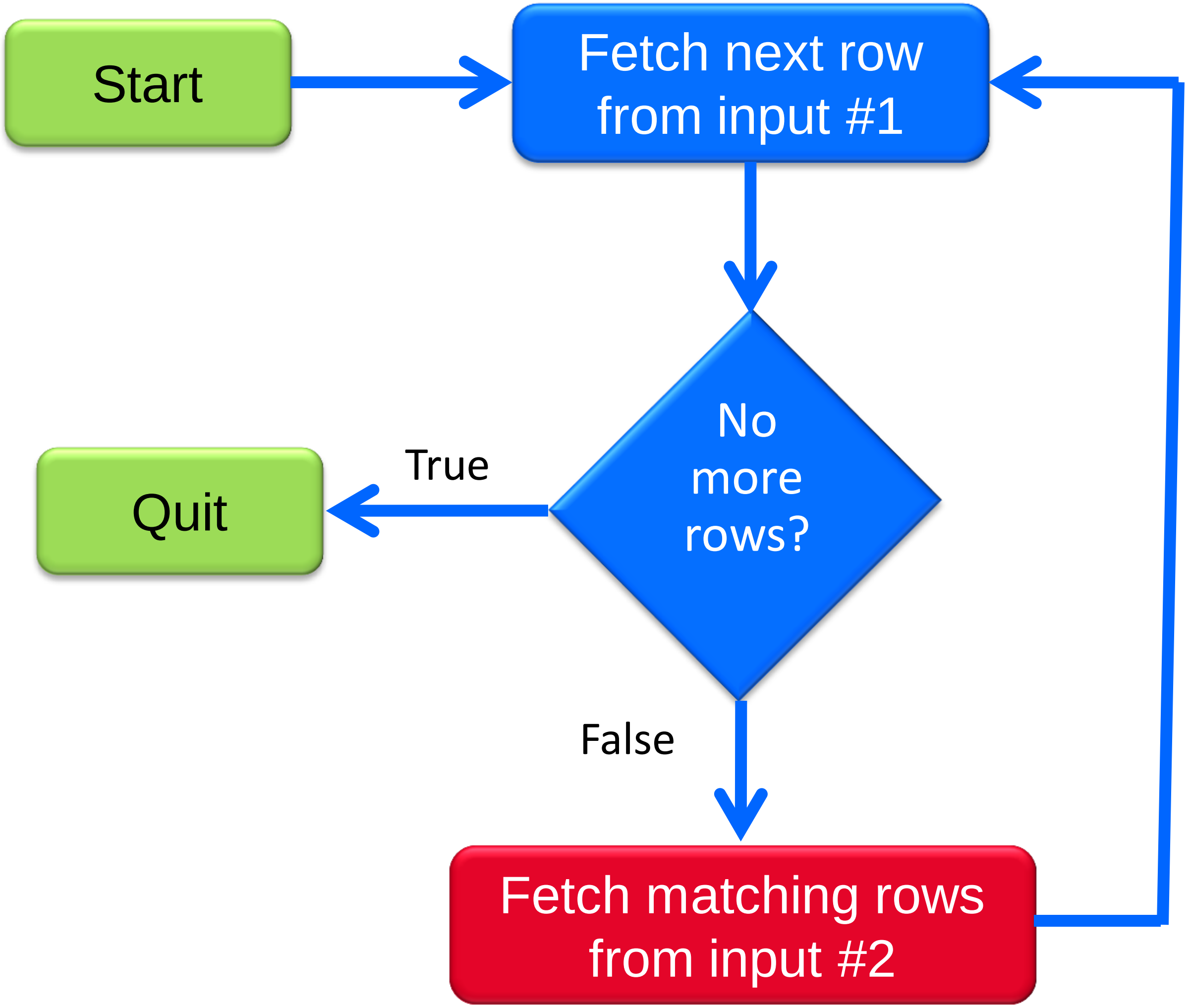
Hash Join

- 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!

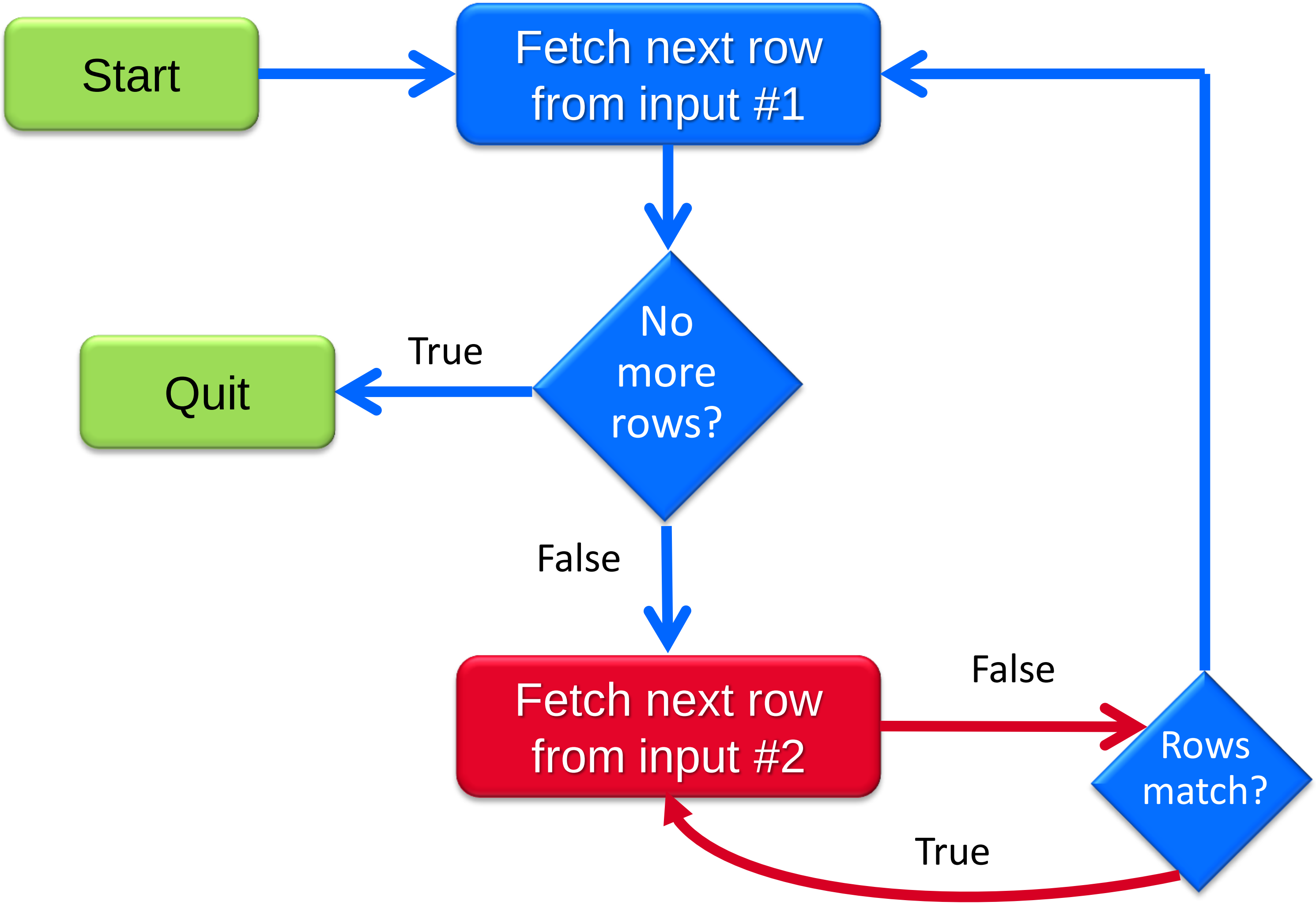
Hash Join Types

- In-Memory Hash
  - Builds hash table in memory
  - Memory grant needed
- Grace Hash
  - Used when Hash Join not fits into memory
  - Hash buckets are stored in temporary work tables (inside TempDb)
- Hybrid Hash
  - Used when Hash Join is slightly larger than available memory
  - Combination of In-Memory and Grace Hash
- Recursive Hash
  - Grace Hash is still to large
  - Must be further re-partitioned

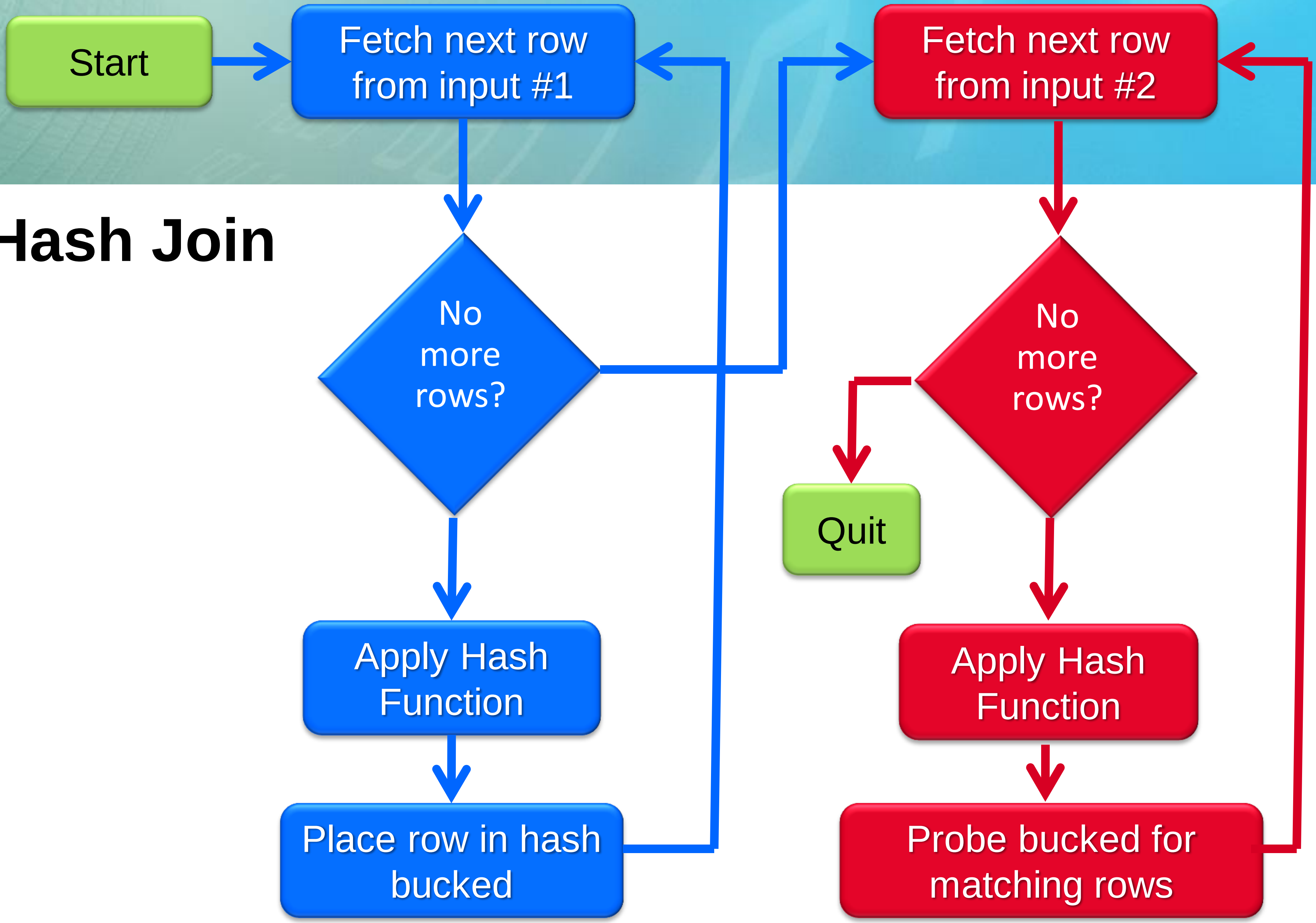
Nested Loop Join


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

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

Hash Join



Start

Fetch next row from input #1

No more rows?

Apply Hash Function

Place row in hash bucked

Fetch next row from input #2

No more rows?

Quit

Apply Hash Function

Probe bucked for matching rows

Stream Aggregate

- 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

Demo

Working with Execution Plans

Capturing Execution Plans

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

Demo

Capturing Execution Plans

Agenda

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

Execution Plan Cache

- Each unique query gets an Execution Plan
  - Performed by the Query Optimizer
  - Identical queries gets the same Execution Plan
  - Problematic when used with dynamic T-SQL
- Creation takes some time
- Stored in the Plan Cache
  - Internal memory inside SQL Server
  - Accessible through sys.dm_exec_cached_plans

sys.dm_exec_cached_plans

- refcounts
  - Number of objects referecing this plan
- usecounts
  - Usage count
- size_in_bytes
- cacheobjtype
  - Compiled Plan: a completed execution plan
  - Parse Tree: a plan stored for accessing a view
  - Ad Hoc Workload: used for ad hoc queries
- objtype
  - Proc
  - Ad hoc
  - View
- plan_handle
  - Identifier for this plan in memory

sys.dm_exec_sql_text

- Returns the SQL statement for each Execution Plan in the cache
- Can be cross joined with sys.dm_exec_cached_plans
- Needs the plan handle as an input

Clearing the Plan Cache

- DBCC FreeSystemCache
  - Plan Cache Stores can be cleaned up individually
    - „Object Plans“
    - „SQL Plans“
    - „Bound Trees“
    - „Extended Stored Procedures“
  - Clearing by
    - plan_handle
    - sql_handle
    - pool_name
- DBCC FreeProcCache
  - Useful for testing, but NOT in production!

Execution Plan Cache Aging

- Plan is saved with
  - Age
  - Cost
- Each time a plan is referenced, the age field is incremented by the compiliation cost factor
- Plan Cache is cleaned periodically
  - Age field is decremented by 1 for each cached plan
- Execution Plan is removed when
  - Memory Manager requires more memory AND
  - All available memory is currently in use AND
  - Age field is 0 for an Execution Plan AND
  - Not currently referenced by a connection AND

Execution Plan Cache Size

- SQL Server 2005 RTM & SP1
  - 0 – 8 GB: 75%
  - 8 – 64 GB: 50%
  - > 64 GB: 25%
  - E.g. 32 GB RAM
    - 6 GB + 12 GB = 18 GB Plan Cache!
- SQL Server 2005 SP2/SQL Server 2008
  - 0 – 4 GB: 75%
  - 4 – 64 GB: 10%
  - > 64 GB: 5%
  - E.g. 32 GB RAM
    - 3 GB + 2,8 GB = 5,6 GB Plan Cache!

Demo

Plan Cache

Agenda

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

Plan Caching

- Adhoc Query Caching
- Auto Parametrization
- Forced Parametrization
- Optimize for Adhoc Workload
- Query Hash Analysis

Adhoc Query Caching

- Each unique query gets cached
  - Only reused for the identical query
  - Exact text match necessary
- sys.dm_exec_cached_plans
  - cacheobjtype „Compiled Plan“
  - objtype „Adhoc“

Auto Parametrization

- Safe Plans can be reused
- SQL Server parametrizes them automatically
- Statistics are used to determine if a plan is safe
- Each invidiual query gets also cached
  - „Shell Query“
  - Cached to make it easier to find the parametrized version
- sys.dm_exec_cached_plans
  - cacheobjtype „Compiled Plan“
  - objtype „Prepared“

Restrictions

- JOIN
- IN
- BULK INSERT
- UNION
- INTO
- DISTINCT
- TOP
- GROUP BY, HAVING, COMPUTE
- Sub Queries
- ...

Forced Parametrization

- Database Option
  - ALTER DATABASE <db_name> SET PARAMETERIZATION FORCED
- Forces Auto Parametrization
  - Constants are treated as parameters
  - Plans are considered as safe... are they?
- Only a few exceptions
  - INSERT ... EXECUTE
  - Prepared Statements
  - RECOMPILE
  - COMPUTE
  - ...

Optimize for Adhoc Workloads

- Available on SQL Server 2008 and higher
- Server Option
- Adhoc Query Plans are not cached on the first use
  - Stub is put into the Plan Cache (~ 344 bytes)
  - On subsequent reuse the whole Execution Plan is cached
- Better Memory Management
- 2nd Recompile necessary!
- sys.dm_exec_cached_plans
  - cacheobjtype „Compiled Plan Stub“
  - objtype „Adhoc“

Query Hash Analysis

- Exposed through sys.dm_exec_query_stats
  - query_hash
  - query_plan_hash
- Can be used to determine if Forced Parametrization should be enabled, or not
  - Each query without constants gets a hash value
  - Each generated Execution Plan gets a hash value
- Goal
  - Each query_hash (without constants) should have the SAME query_plan_hash
  - Consistent, safe plan across different input parameters

Demo

Plan Caching

Agenda

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

Parameter Sniffing

- SQL Server compiles Stored Procedures
  - On their first use
  - Considers input parameters for generating an Execution Plans
  - Execution Plans are cached for further usage
- Different Values
  - Compiled Value
  - Runtime Value

Problems

- Sub-optimal Execution Plans
- High I/O
  - Because of inappropriate Index Usage
  - Leads to bad performance
- Resolutions
  - Query Hints
  - Don‘t consider input parameters
  - „Override“ SQL Servers behaviour

Query Hints

- RECOMPILE
  - Stored Procedure Level
  - Statement Level
    - Performs better
    - Leads to better Execution Plans
- OPTIMIZE query hint
  - For specific values
  - FOR UNKOWN
    - SQL Server assumes that 30% of the data is returned...!
  - Only SQL Server 2008 onwards
- Plan Guides can be attached to existing queries
  - We don‘t have to change/rewrite our applications

Other Resolutions

- Use local variables instead of parameters
  - SQL Server can‘t „sniff“ their values...
- Trace Flag 4136
  - Disables parameter sniffing completely
  - KB980653
  - Do we really want this...?

Demo

Detecting & Resolving Parameter Sniffing

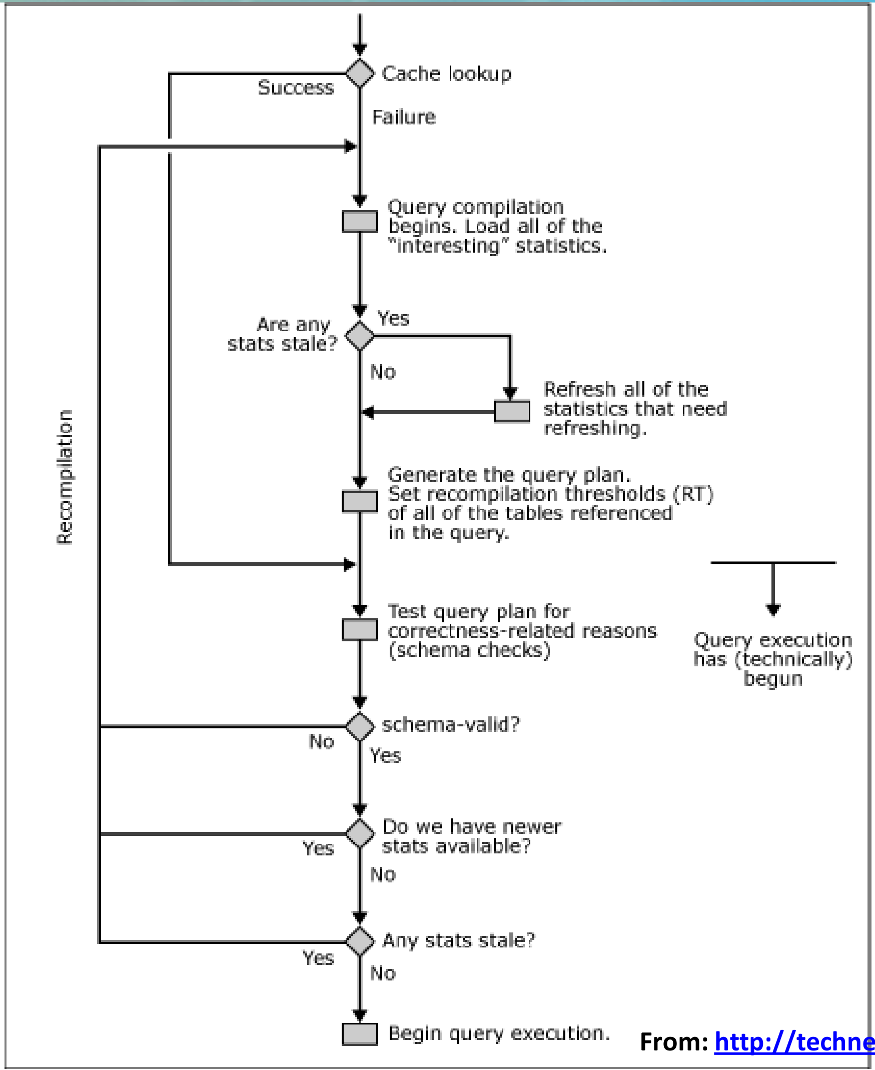
Agenda

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

Recompilations

- Query is recompiled before it is executed again
  - Plan is already in the Plan Cache
  - Recompiled Plan is replaced in the Plan Cache
  - Statement-Level Recompile on SQL Server 2005 and higher
- 2 Types
  - Correctness-Based Recompilation
  - Optimality-Based Recompilation

Recompilations


Cache lookup
Success
Failure
Query compilation begins. Load all of the "interesting" statistics.
Are any stats stale?
Yes
No
Refresh all of the statistics that need refreshing.
Generate the query plan. Set recompilation thresholds (RT) of all of the tables referenced in the query.
Recompilation
Test query plan for correctness-related reasons (schema checks)
Query execution has (technically) begun
schema-valid?
No
Yes
Do we have newer stats available?
Yes
No
Any stats stale?
Yes
No
Begin query execution.

From: http://technet.microsoft.com/en-us/library/cc966425.aspx

Correctness-Based Recompilations

- Occurs when plan is not correct anymore, because of
  - Schema Changes
  - Environment Changes

Schema Changes

- Adding/Dropping
  - Columns
  - Indexes
  - Triggers
- Dropping Statistics
- Changing Data Types
- Running sp_recompile
  - Does NOT the actual recompile
  - Marks the object for Recompilation

Environment Changes

- Changing SET Options
  - sys.dm_exec_plan_attributes

Optimality-Based Recompilations

- Updated Statistics
  - Automatically
  - Manually
- Stale Statistics
  - A table with no rows gets a row
  - A table has fewer than 500 rows and is increased by 500 or more rows
  - A table has more than 500 rows and is increased by 500 rows + 20% of the number of rows

Demo

Recompliations

Summary

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