

Data-driven systems

Query optimization



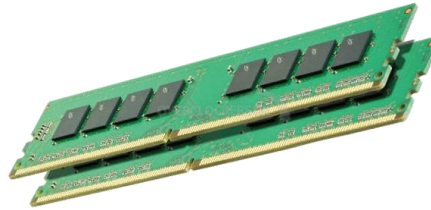
Automatizálási és
Alkalmazott
Informatikai Tanszék

Contents

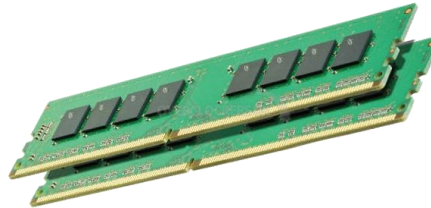
- Purpose of query optimization
- Microsoft SQL Server
 - > Execution plans
 - > Join and table access options
 - > General recommendation
- MongoDB
 - > Indices and execution plans

What is the purpose of query optimization?

What is the purpose of query optimization?



What is the purpose of query optimization?



Response time is affected by

- I/O cost
 - > Most prominent in data bases
 - > Does not improve according to Moore's law
 - > Needs special tricks
- CPU usage
 - > Complex queries
 - > Complex computations
- Memory usage
 - > Cache effect

Microsoft SQL Server

General concepts

Basics of the optimization

- Evaluates based on statistics
 - > Cost = response time (CPU + I/O time)
- Trivial plan
 - > Unambiguous for simple queries
 - > Rule-based
- When no trivial plan is available
 - > Complex queries
 - > Three phase optimization

Three phase optimization

- No trivial plan
- 0. Phase
 - > Simple optimizations
 - > Preferred hash join
 - > If cost $< X \rightarrow$ execute
- 1. Phase
 - > Complex optimizations
 - > If cost $< Y \rightarrow$ execute
- 2. Phase
 - > Parallel execution

Process of executing a query

- Analyzer
 - > Compiler
 - > Logical plan
- Optimization
 - > Physical execution plan
 - > Read tables
 - > Joining tables
- Row executor
 - > Mapping physical plan to I/O operations
- Executor
 - > Executes the operations

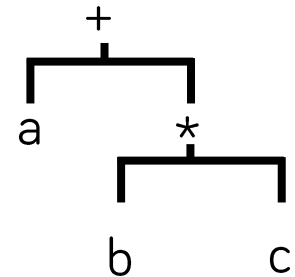
Microsoft SQL Server

Logical execution plan

Elements of the logical execution plan

- Parser tree
 - > Relations (leaf)
 - > Operations (node)
 - > Data flow from bottom to top
- Relational algebra operations
 - > Cartesian-join ($R \times S$)
 - > Projection ($\pi_L(R)$)
 - > Selection ($\sigma_F(R)$)
 - > Join ($R \bowtie S$)
 - > Filtering duplicates ($\delta(R)$)
 - > Grouping ($\gamma_L(R)$)
 - > Sorting ($\tau_L(R)$)

a + b * c

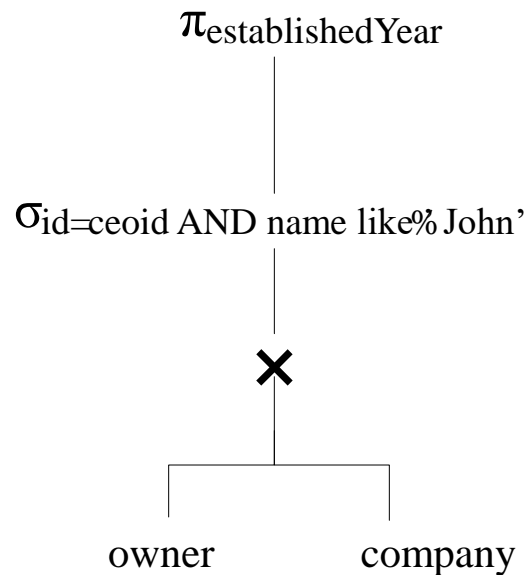


Parser tree

select establishedYear

from owner o, company c

where o.id = c.ceoid AND name like '%John'



Refactoring the parser tree - 1

- Create an optimal logical execution plan
- Reduce possible physical execution options
- Basic concept
 - > Move selection (where) down in the tree
 - > Using joins
 - Cartesian join only when explicitly specified
 - > One side of a join should be a table

Refactoring the parser tree - 2

- Selection

- > Can be re-ordered: $\sigma_{F_1}(\sigma_{F_2}(R)) = \sigma_{F_2}(\sigma_{F_1}(R))$

- > Can be re-written:

- $\sigma_{F \text{ and } G}(R) = \sigma_F(\sigma_G(R))$

- $\sigma_{F \text{ or } G}(R) = \sigma_F(R) \text{ UNION } \sigma_G(R)$

- Join

- > $R \bowtie_F S = \sigma_F(R \times S)$

- > $R \bowtie S = S \bowtie R$

- > $(R \bowtie S) \bowtie U = R \bowtie (S \bowtie U)$

Refactoring the parser tree - 3

- Joining and selection

- > $\sigma_F(R \bowtie S) = \sigma_F(R) \bowtie S$

- If R has all attributes of F

- > $\sigma_F(R \bowtie S) = R \bowtie \sigma_F(S)$

- If S has all attributes of F

- > $\sigma_F(R \bowtie S) = \sigma_F(R) \bowtie \sigma_F(S)$

- If both R and S has attributes of F

- Duplicates

- > $\delta(\gamma_L(R)) = \gamma_L(R)$

- > $\delta(R \times S) = \delta(R) \times \delta(S)$ (similarly for joins)

Microsoft SQL Server

Physical execution plan

Physical plan

- Elements of the physical plan
 - > Operators for reading the table
 - Logical plan leaf reading operations
 - > Executing relational algebra operations
- Creating plans
 - > Rule-based
 - > Cost-based
 - Table seek methods
 - Implementation of joins
 - Order of joins

Nested loop join

- Two embedded for cycles
- I/O cost
 - > $O(\text{num_block_1} * \text{num_block_2})$
- Works in all cases
 - > In case of large tables: keep only partitions of the tables in memory

Hash join

- First pass
 - > Read the smaller table
 - > Build a hash table in memory
 - Key is the column used for joining
- Second pass
 - > Read the larger table
 - > Search for matching records in the hash table
- I/O cost
 - > $O(\text{num_block}_1 + \text{num_block}_2)$

Sort Merge Join

- Reads both tables into memory
- Sorts based on the joined columns
- Merge the two sorted lists
 - > While "walking" the two sorted lists
- For small tables
- Index due to sorting
- I/O cost
 - > $O(\text{num_block_1} + \text{num_block_2})$

Table scan methods - 1

- Generally there are two ways
 - > Full scan
 - For small tables
 - When most rows are needed
 - > Index scan
 - When using filtering
 - If there is an index covering the filter criteria
 - Sorting

Table scan methods - 2

- Table scan
 - > No index
 - > Evaluates the filtering condition
- Clustered index scan
 - > Clustered reading
 - > Data blocks ordered by index
 - > Clustered index created along primary key
 - > Preferred over table scan

Table scan methods - 3

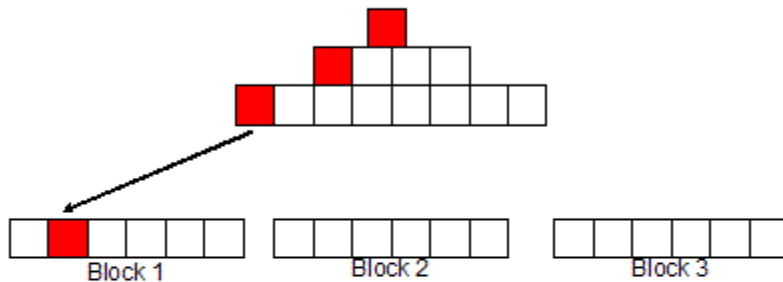
- Nonclustered index scan
 - > Similar to clustered index scan
 - > Mostly for evaluating =
- Clustered/Nonclustered index seek
 - > Similar to index scan
 - > Walks the index from a starting point
 - >, between, < operators

MS SQL Server indexes

- B* tree
 - > Simple
 - > Compound indexes
 - Hierarchical
 - > Clustered
 - Data blocks ordered by index
 - One for each table
 - Automatically created for the primary key

MS SQL Server indexes

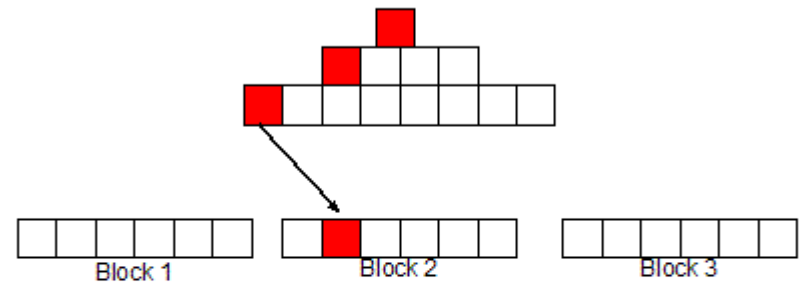
- Clustered / non-clustered



Select order_nbr, item_name from ordor natural join item;

Clustered table rows

Clustering_factor ~= blocks



Select order_nbr, item_name from ordor natural join item;

Un-Clustered table rows

Clustering_factor ~= num_rows

Source: http://www.dba-oracle.com/t_table_row_resequencing.htm

Indexes - 1

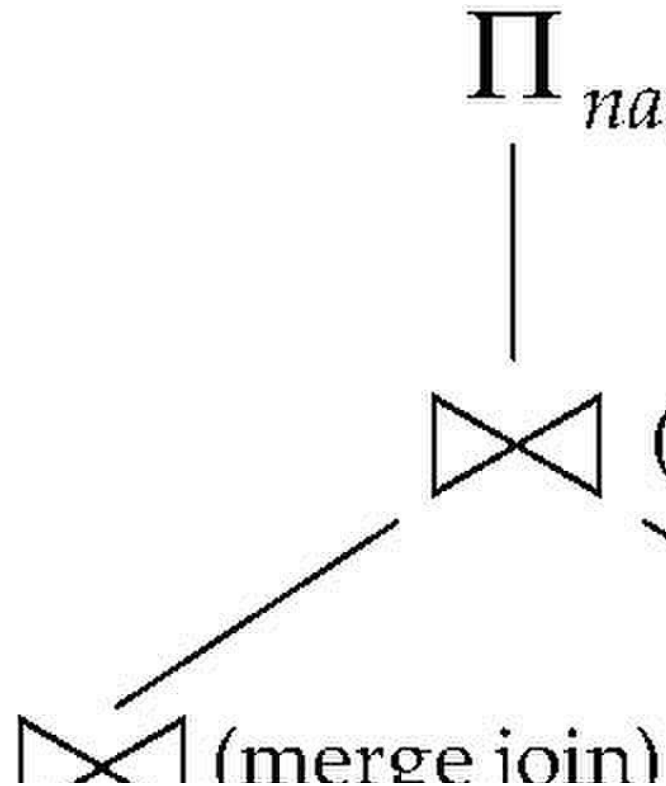
- Cover index (included column)
 - > Adding further data into the B* tree leaf nodes
 - > The row data does not need to be accessed
- Using clustered and non clustered indices together
 - > Nonclustered index leaf
 - Does not contain a physical address
 - Points to the clustered index
 - > Double index read

Indexes - 2

- Indexed views
 - > The view result is stored
 - > Index can be defined, works as for tables

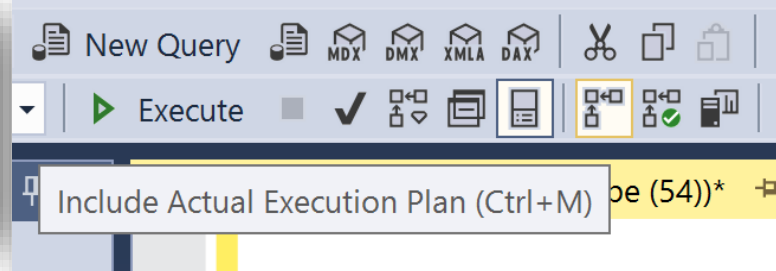
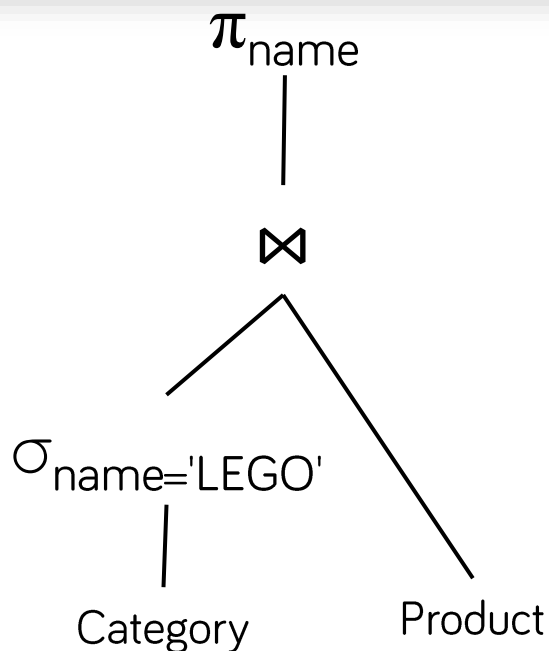
Execution plan

- It defines exactly what action is performed on each node



Végrehajtási terv megnézése

```
select p.name from Product p
join Category c on p.CategoryID = c.ID
where c.Name = 'LEGO'
```



Query 1: Query cost (relative to the best query plan available) for the query: select p.name from Product p join Category c on p.CategoryID = c.ID where c.Name = 'LEGO'

SELECT	Nested Loops (Inner Join)	Clustered Index... [Product].[PK_...]
Cost: 0 %	Cost: 1 %	Cost: 41 %
	0.000s	0.000s
	1 of	10 of
	1 (100%)	10 (100%)
		Clustered Index... [Category].[PK...]
		Cost: 58 %
		0.000s
		1 of
		10 (10%)

Parallel	False
Physical Operation	Clustered Index Seek
Predicate	[M22XDS].[dbo].[Category].[Name] as [c].[Name]=N'LEGO'
Scan Direction	FORWARD

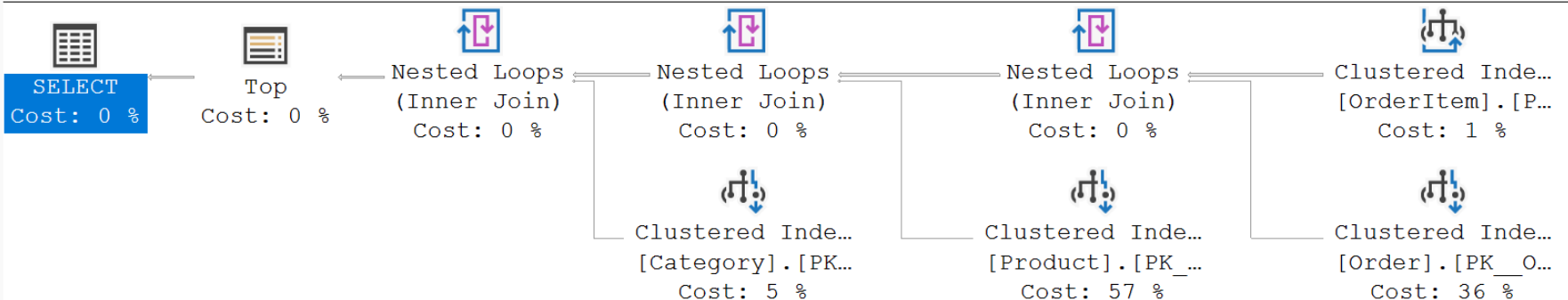
Query plan in SQL Server

- Press CTRL+L after executing the query

```
select top 100 O.ID, p.name, c.name, oi.price
from
  OrderItem oi join [Order] o on oi.OrderID = o.ID
  join Product p on oi.ProductID = p.ID
  join Category c on p.CategoryID = c.ID
```

Query 1: Query cost (relative to the batch): 100%

select top 100 O.ID, p.name, c.name, oi.price from OrderItem oi join [Order] o on oi.Ord

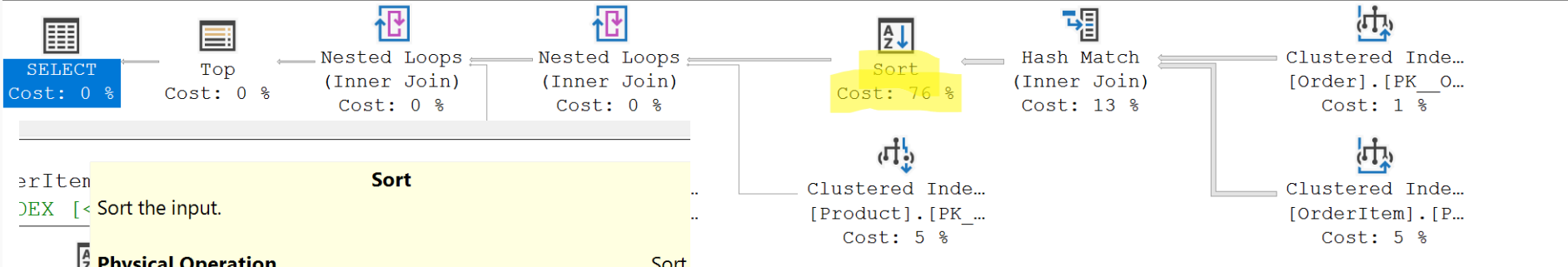


Index hints

```

select top 100 O.ID, p.name, c.name, oi.price
from
    OrderItem oi join [Order] o on oi.OrderID = o.ID
    join Product p on oi.ProductID = p.ID
    join Category c on p.CategoryID = c.ID
order by o.Deadline desc
    
```

Query 1: Query cost (relative to the batch): 100%
 select top 100 O.ID, p.name, c.name, oi.price from OrderItem oi join [Order] o on oi.OrderID = o.ID joi...
 Missing Index (Impact 17.5168): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo]...

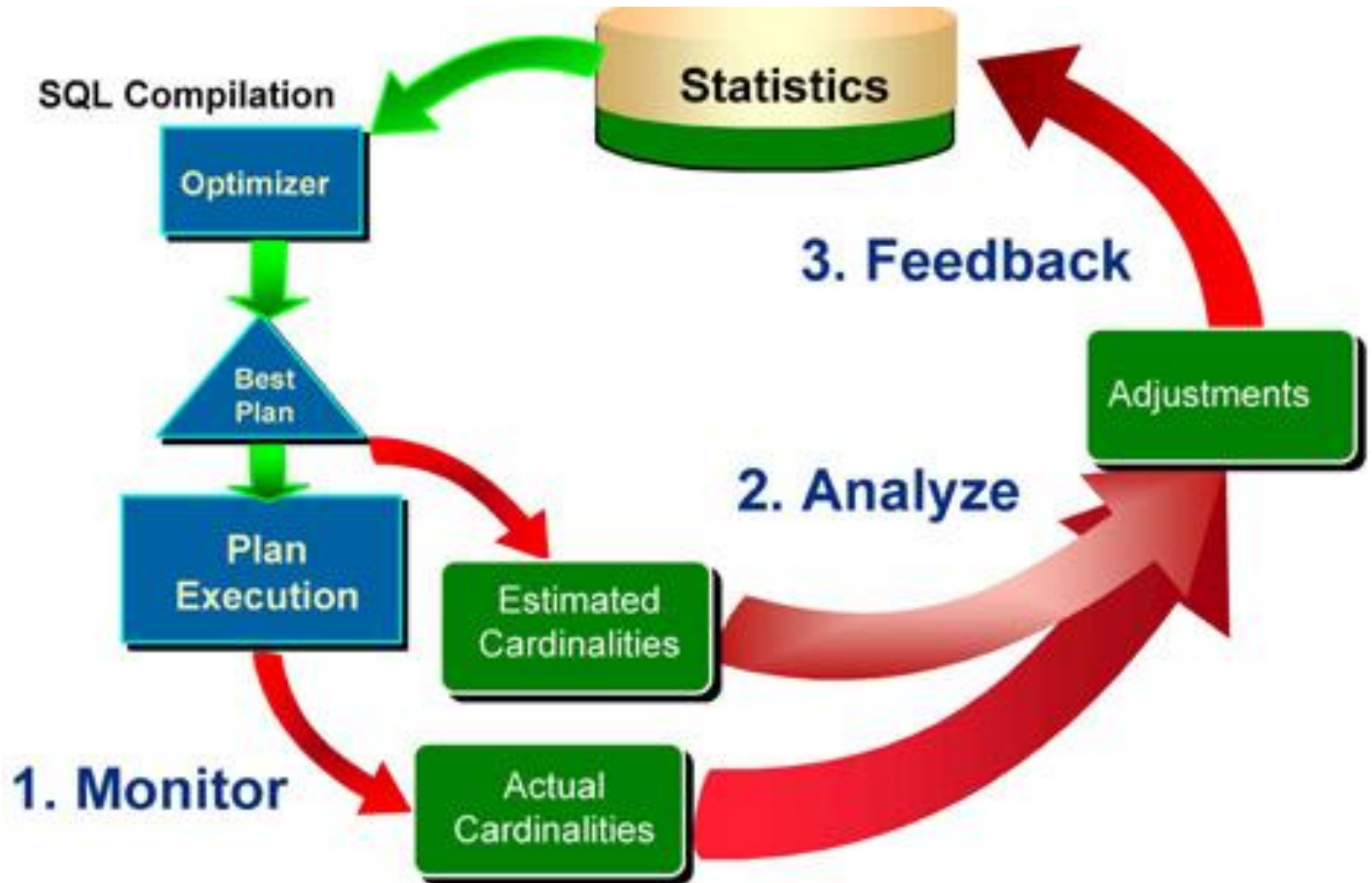


Sort	
Physical Operation	Sort
Logical Operation	Sort
Estimated Execution Mode	Row
Estimated I/O Cost	0,0112613
Estimated Operator Cost	3,40483 (76%)
Estimated CPU Cost	3,39357
Estimated Subtree Cost	4,24715
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	100
Estimated Number of Rows Per Execution	100
Estimated Row Size	31 B
Node ID	4
Output List	
[M22XDS].[dbo].[OrderItem].Price; [M22XDS].[dbo].	
[OrderItem].ProductID; [M22XDS].[dbo].[Order].ID; [M22XDS].	
[dbo].[Order].Deadline	
Order By	
[M22XDS].[dbo].[Order].Deadline Descending	

Execution plan alternatives

- There can be several plan alternatives, which one is optimal?
 - > Huge differences: seconds or days
 - > The cost depends on how many lines are the result for each phase
 - > The system estimates this based on statistics
 - -> Self-tuning database, redesigns if necessary, during execution
- Plan cache
 - > Execution plan cache
 - > Used when for queries with the same structure
 - > Statistics have not changed
 - You may need to update the cache manually!
 - The statistics maintenance must be setup!

Self-tuning



Microsoft SQL Server

General recommendation

Best practices - 1

- Keep statistics up to date
 - > Deprecated statistics → suboptimal execution plan
 - > (This is the default unless turned off)
- Structure of the query
 - > SQL is declarative
 - Keep procedural execution in mind
 - > Same result can be obtained multiple ways
 - > Make it simple
 - > Avoid select *
 - > Good structure is advantageous
 - Use hints as a last resort

Best practices - 2

- Prefer join over
 - > In / Not in
 - > Exists / Not exists
- Prefer In over Exists
- Views
 - > Avoid if possible
 - > Do not join them
- Avoid Or clauses → Union all
- Union all if possible

Best practices - 3

```
select *  
from Invoice i  
where not exists  
(  
    select 1  
    from Invoiceltem ii  
    where i.Id=ii.InvoiceID  
)
```

```
select i.*  
from Invoice i  
where i.id not in  
(  
    select InvoiceID  
    from Invoiceltem  
)
```

```
select i.*  
from Invoice i left outer join Invoiceltem ii  
on i.Id=ii.InvoiceID  
where ii.id is null
```

Does not matter in simple cases

Best practices - 4

- Using indexes
 - > Usually one can be used for one table in a query
 - Join may use it up
 - > Compound indexes
 - Hierarchy matters
 - > If the key is used in an expression, the optimizer cannot use it
 - E.g., `key+0` (← Optimizer may handle this though)

Best practices - 5

- Using functions
 - > No problem in a select
 - Does not affect the execution plan
 - > Avoid in a where clause
 - Has to be evaluated for each record
 - Hard to move in the query tree
 - No statistics are available for its output → hard to optimize

Further reading material

- Grant Fritchey: SQL Server Execution Plans, Simple Talk Publishing, 2012
 - > pdf: <http://www.red-gate.com/community/books/sql-server-execution-plans-ed-2>

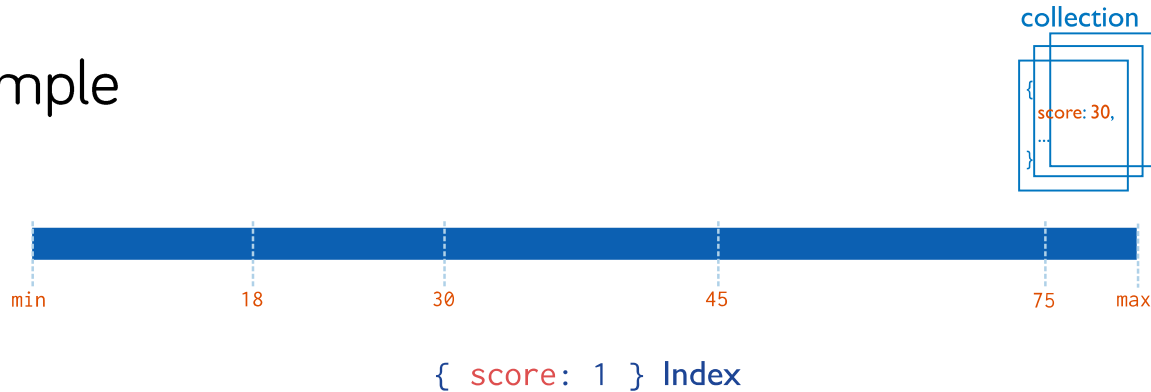
MongoDB

Indices in MongoDB

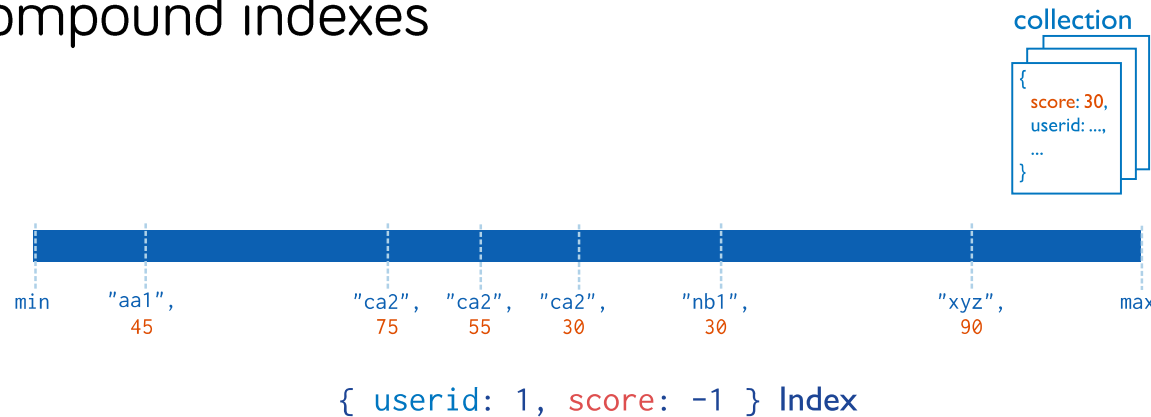
- Index “only” for lookup
 - > (As there is no join operation)
- Types of indices
 - > Simple and compound
 - > Unique index
 - Can be used to ensure a primary key-like attribute
 - > Indexes content of arrays too
 - > Indexes nested objects too
 - > TTL, Geospatial, full text
- Index must be defined
 - > Except: `_id` unique

Types of indices

Simple



Compound indexes



Images source: <https://docs.mongodb.com/manual/indexes/>

Basics of the optimization

- Does not use statistics
- Choice between multiple possible plans
 - > Starts execution all, whichever yields the first 101 results, is the best
- How can there be multiple plans?
 - > There are multiple indices covering the query

Optimization steps

- Move filtering ahead of other steps
 - > Before projection, and if needed, split the filtering into two
 - > Before sorting
- Move skip and limit ahead
 - > Projection
- Merge
 - > Limit + limit, skip + skip

Plan cache

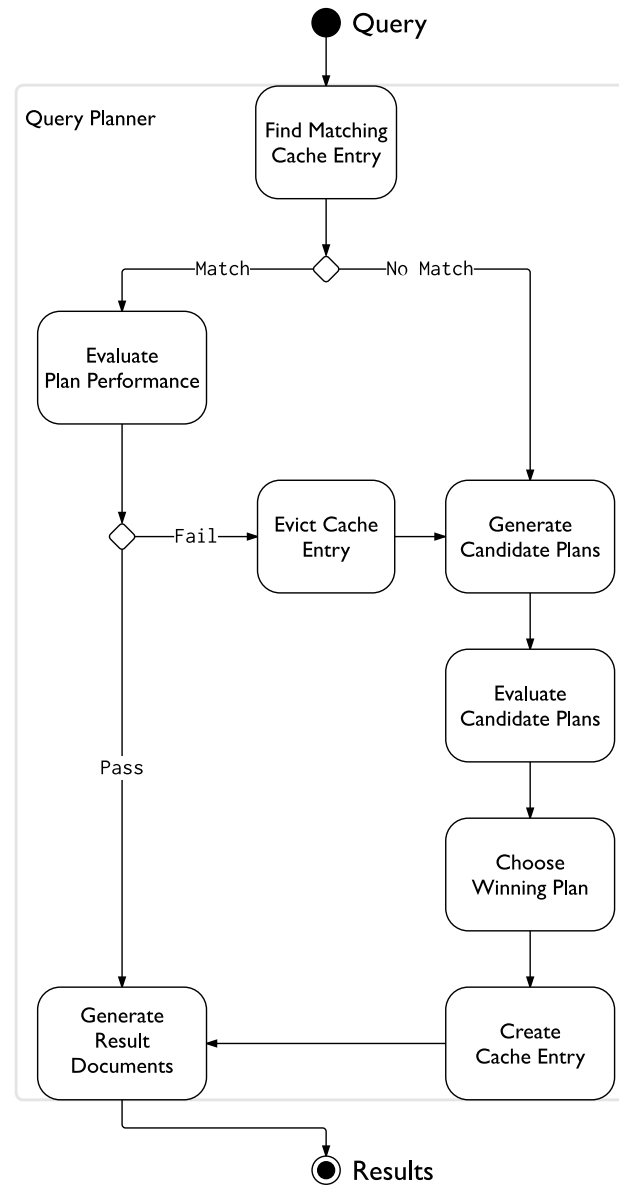


Image source:
<https://docs.mongodb.com/manual/core/query-plans/>

Plan cache

- Plan cache
 - > Structurally similar plans
 - > Pass/fail evaluation
- *Query shape*
 - > Filters, sorting, etc. used
 - > No values
 - E.g. for filtering only the filtered field name is present

Explain

- `query.explain()`

```
"winningPlan" : {  
  "stage" : <STAGE1>,  
  ...  
  "inputStage" : {  
    "stage" : <STAGE2>,  
    ...  
    "inputStage" : {  
      "stage" : <STAGE3>,  
      ...  
    }  
  }  
},  
"rejectedPlans" : [  
  <candidate plan 1>,  
  ...  
]
```

Stage-ek

- *COLLSCAN*
- *IXSCAN*
- *FETCH*
- ...