Elektrotehnički fakultet
Univerzitet u Beogradu
Katedra za računarsku tehniku i informatiku

Baze podataka 2

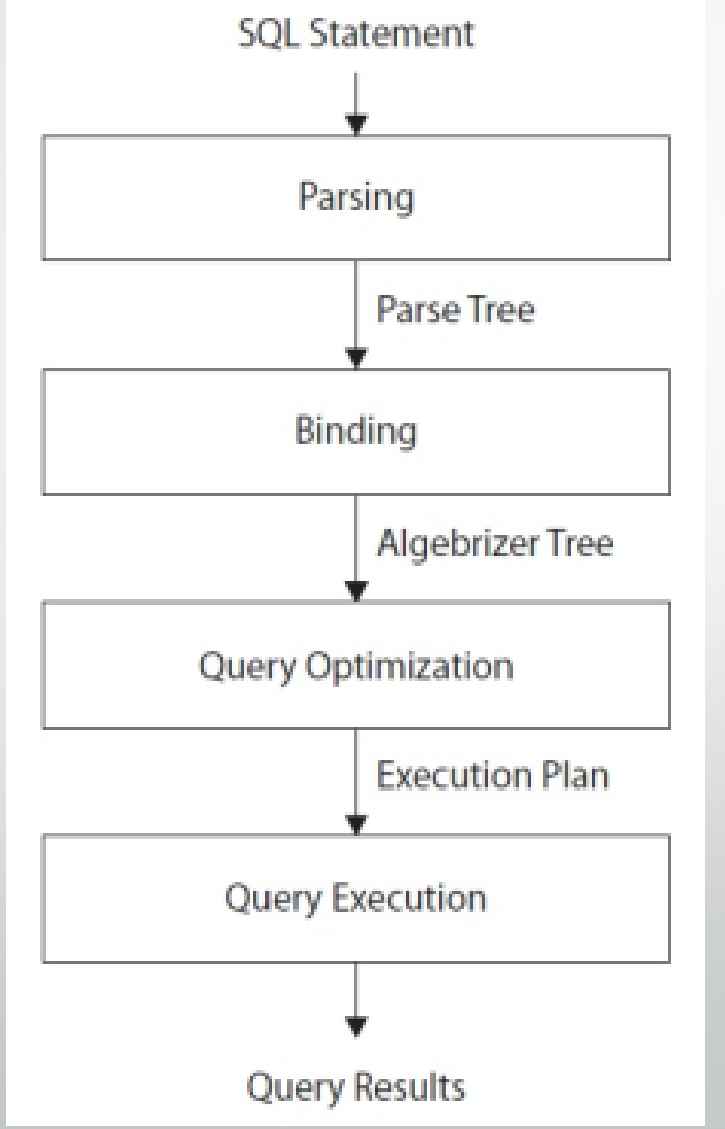
Izvršavanje upita na SQL Server-u

2023/2024.

Sadržaj

- Uvod u procesiranje upita
- The Query optimizer
- The Execution Engine
- Indexes
- Analiza statistike

Faze procesiranja upita



Parsing and binding

- Provera sintaksne ispravnosti upita
- Provera korektnosti tipova argumenata, funkcija, korelacija, podupita, ...
- Prevodenje upita na interni zapis
  - Rezultat parsiranja je **Parse Tree**
  - Rezultat bindinga je **Algebrized Tree**
- Rezultat ove faze je logičko stablo, pri čemu svaki čvor u stablu predstavlja logičku operaciju koju upit mora izvršiti, kao što je čitanje određene tabele ili izvođenje unutrašnjeg spajanja.

Query optimization

- Optimizacija upita se sastoji iz dva koraka:
  1. Generisanje mogućih planova izvršenja
  2. Procena cene svakog plana

- Rezultat optimizacije upita je **Execution plan**

Query optimization

- Generisanje mogućih planova izvršenja:
  - Koristeći generisano logičko stablo, optimizator upita osmišljava nekoliko mogućih načina za izvršenje upita (tj. nekoliko mogućih planova izvršenja).
  - Plan izvršenja je skup fizičkih operacija (kao što je traženje indeksa ili spajanje ugnežđene petlje) koje se mogu izvesti da bi se proizveo traženi rezultat, koji je opisan logičkim stablom.

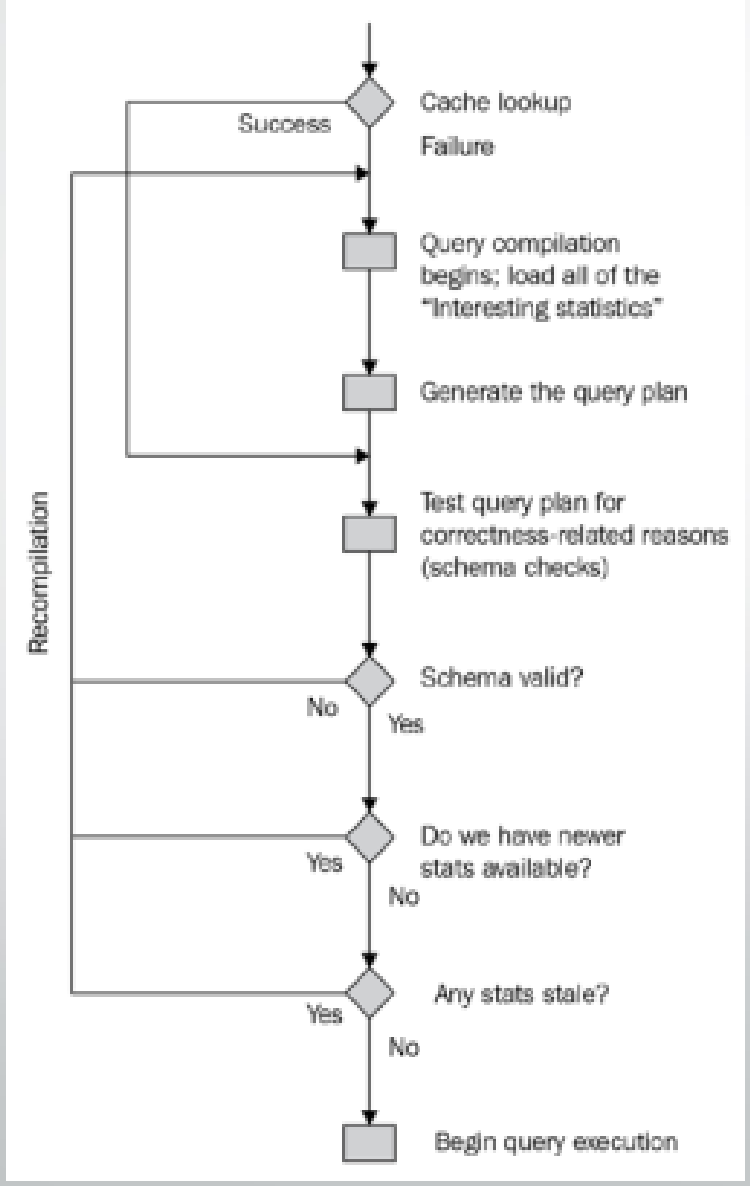
Query optimization

- Procena cene svakog plana
  - Optimizator upita ne generiše svaki mogući plan izvršenja.
  - Optimizator procenjuje troškove resursa i vremena svakog generisanog plana (svake razmatrane operacije). Bira se plan za koji optimizator upita smatra da ima najnižu cenu, a zatim se prosleđuje mašini za izvršavanje.

Query execution and plan caching

- Izvršenje upita i keširanje plana
- Upit se izvršava od strane mašine za izvršenje prema izabranom planu
- Plan može biti uskladišten u memoriji u kešu plana.

Compilation and recompilation process



Korišćena baza podataka

- Primeri će biti radjeni na bazi AdvantureWorks2019
- Link:

  https://learn.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver16&tabs=ssms

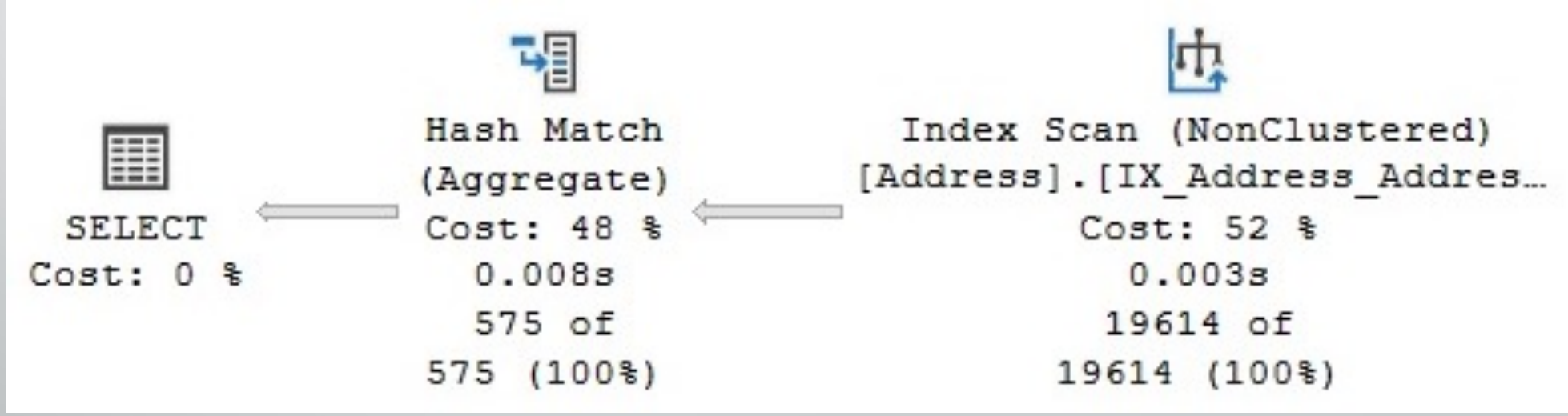
Plan izvršavanja

- Moguće je analizirati stvarni i procenjeni plan izvršavanja.
- Oba plana izvršavanja se mogu prikazati na tri načina:
  1. Grafički plan izvršavanja
  2. XML plan izvršavanja
  3. Tekstualni plan izvršavanja
- Sva tri načina prikazuju isti plan izvršenja, jedina razlika je načinu prikaza i nivou detalja koji sadrže.

Grafički plan izvršavanja

- Prikazuju se izborom opcija **Display Estimated Execution Plan** i **Include Actual Execution Plan** u okviru **SQL Editor** toolbar.
- Primer:

```
SELECT DISTINCT(City) FROM Person.Address
```



XML plan izvršavanja

- Prikazuju se izborom opcija **XML Execution Plan** u okviru Pop up menija u **Execution Plan** prozoru ili pomoću komande SHOWPLAN_XML.
- Primer:

```
SET SHOWPLAN_XML ON
GO
SELECT DISTINCT(City) FROM Person.Address
GO
SET SHOWPLAN_XML OFF
```

	Microsoft SQL Server 2005 XML Showplan
1	<ShowPlanXML xmlns="http://schemas.microsoft.com...

Tekstualni plan izvršavanja

- Prikazuju se pomoću komande SHOWPLAN_TEXT i SHOWPLAN_ALL.
- Primer:

```
SET SHOWPLAN_ALL ON
GO
SELECT DISTINCT(City) FROM Person.Address
GO
SET SHOWPLAN_ALL OFF
```

	StmtText	StmtId	NodeId	Parent	PhysicalOp	LogicalOp
1	SELECT DISTINCT(City) FROM Person.Address	1	1	0	NULL	NULL
2	|--Hash Match(Aggregate, HASH:([AdventureWorks2...	1	2	1	Hash Match	Aggregate
3	|--Index Scan(OBJECT:([AdventureWorks2019].[P...	1	3	2	Index Scan	Index Scan

Tekstualni plan izvršavanja

- Komandom `STATISTICS PROFILER` možemo dobiti dodatne statistike koje mogu pomoći pri uočavanju problema sa procenom kardinalnosti.
- Primer:

```
SET STATISTICS PROFILE ON
GO
SELECT * FROM Sales.SalesOrderDetail
WHERE OrderQty * UnitPrice > 25000
GO
SET STATISTICS PROFILE OFF
```

	Rows	EstimateRows	Executes	StmtText
1	5	36395.1	1	SELECT * FROM [Sales].[SalesOrderDetail] WHERE [OrderQty]*[UnitPrice]>@1
2	5	36395.1	1	|--Filter(WHERE:([Expr1003]>($25000.0000)))
3	0	121317	0	|--Compute Scalar(DEFINE:([AdventureWorks2019].[Sales].[SalesOrderDetail].[LineTotal]=[A
4	0	121317	0	|--Compute Scalar(DEFINE:([AdventureWorks2019].[Sales].[SalesOrderDetail].[LineTotal]
5	121317	121317	1	|--Clustered Index Scan(OBJECT:([AdventureWorks2019].[Sales].[SalesOrderDetail].[P

STATISTICS TIME and IO

- Komandom `STATISTICS TIME` i `STATISTICS IO` možemo dobiti informacije o trajanju i broju obradjenih podataka.
- Primer:

```
SET STATISTICS TIME ON
GO
SELECT DISTINCT(CustomerID)
FROM Sales.SalesOrderHeader
GO
SET STATISTICS TIME OFF
```

```
 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 140 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
```

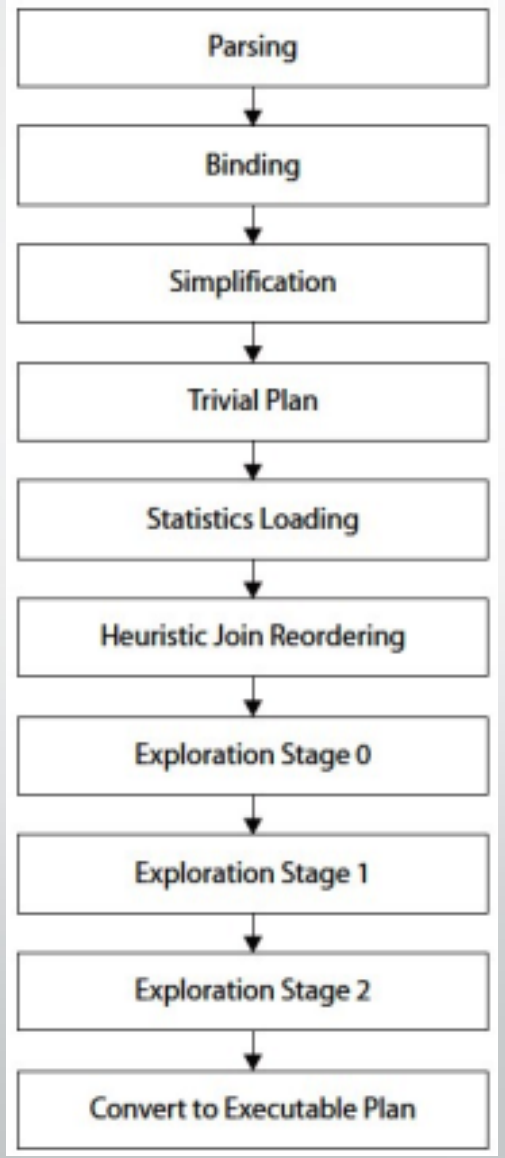
DMVs i DMFs

- **Dynamic management views** (DMVs) and **dynamic management functions** (DMFs) se mogu koristiti da se otkrije broj resursa servera koji se koriste za upite i za pronalaženje najskupljih upita u vašoj instanci SQL Server-u.
- To su:
  - `sys.dm_exec_requests`
  - `sys.dm_exec_sessions`
  - `sys.dm_exec_query_state`
  - `sys.dm_exec_query_optimizer_info`
  - ….

The query optimizer

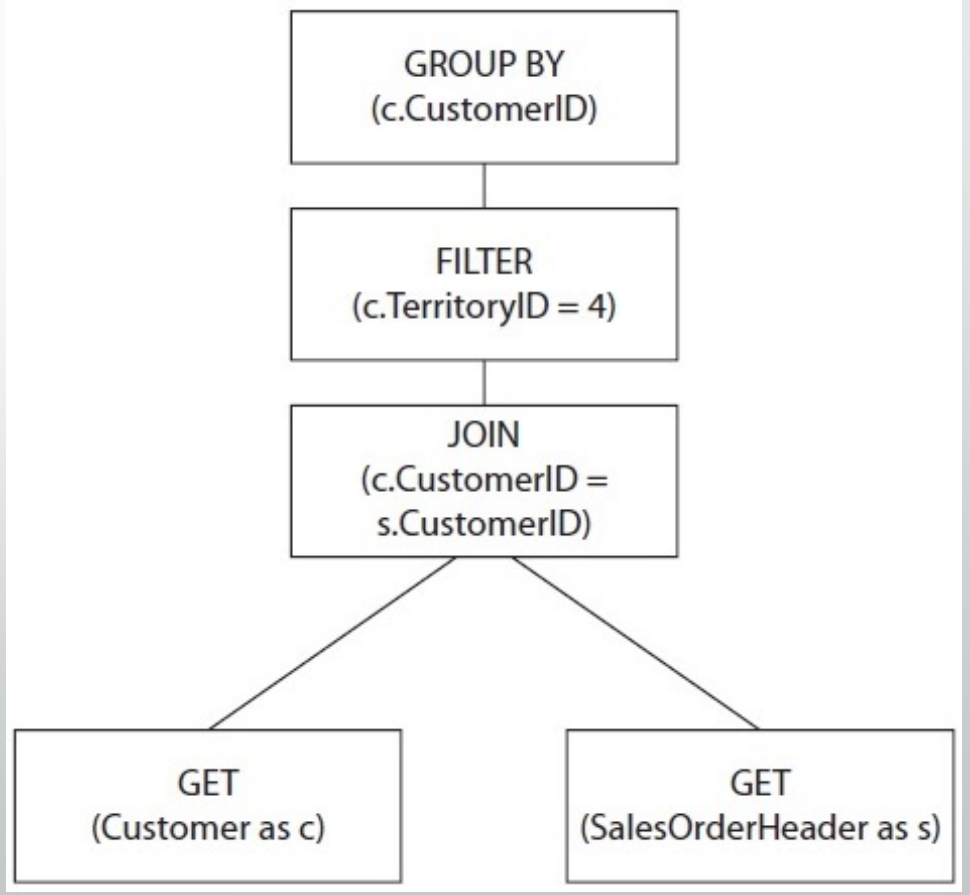
- Faze parsiranja upita
- Parsing and binding
- Simplification
- Trivial plan
- Join reordering
- Transformation rules
- The memo
- Full optimization

Faze procesiranja upita (detaljnije)



Parsing and binding

```
SELECT c.CustomerID, COUNT(*)
FROM Sales.Customer c JOIN Sales.SalesOrderHeader s
ON c.CustomerID = s.CustomerID
WHERE c.TerritoryID = 4
GROUP BY c.CustomerID
```



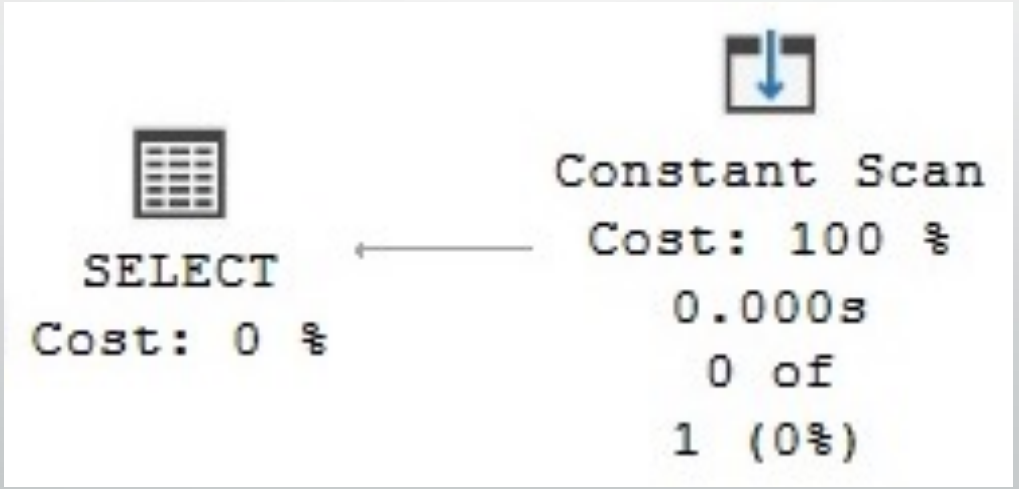
Simplification

- Podupite konvertuje u spajanje tabela. Po potrebi dodaje i grupisanje.
- Uklanja redudantna spajanja
  - Često se javljaju spajanja po principu stranog i primarnog ključa, gde podaci iz osnovne tabele nisu potrebni.
- WHERE iskaze spušta niže, kako bi se ranije izvršili
- Proverava da li ima kontradiktornih iskaza i uklanja ih

Simplification

- Cardinality detection
  - Kod upita koji dohvataju vrednosti koje su izvan ograničenem dozvoljenog opsega.

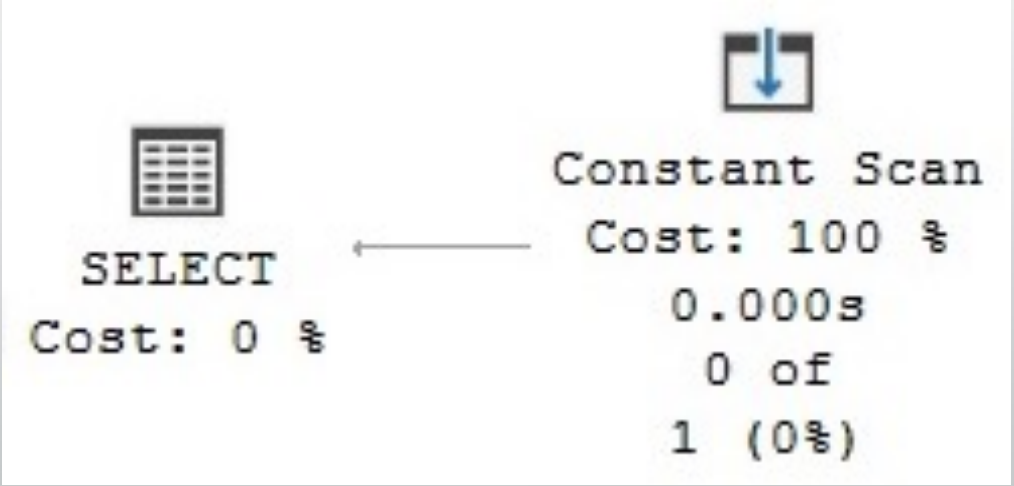
```
SELECT *
FROM HumanResources.Employee
WHERE VacationHours > 300
```



Simplification

- Cardinality detection
  - Kod upita gde su uslovi kontradiktorni.

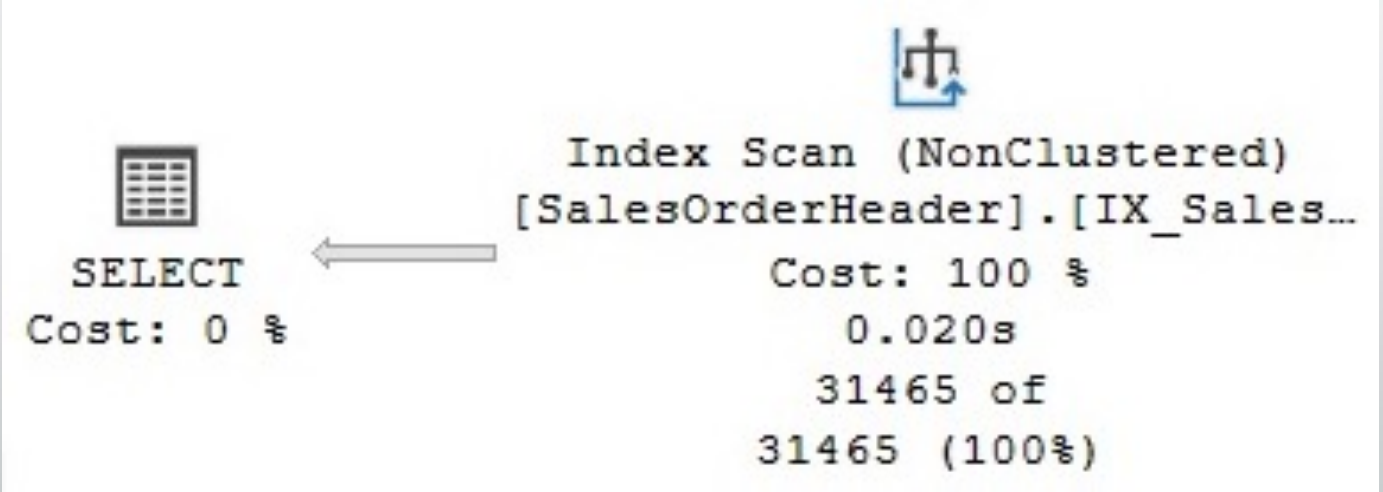
```
SELECT *
FROM HumanResources.Employee
WHERE VacationHours > 10 AND VacationHours < 5
```



Simplification

- Foreign Key Join elimination

```
SELECT soh.SalesOrderID --, c.AccountNumber
FROM Sales.SalesOrderHeader soh
JOIN Sales.Customer c ON soh.CustomerID = c.CustomerID
```



Trivial plan

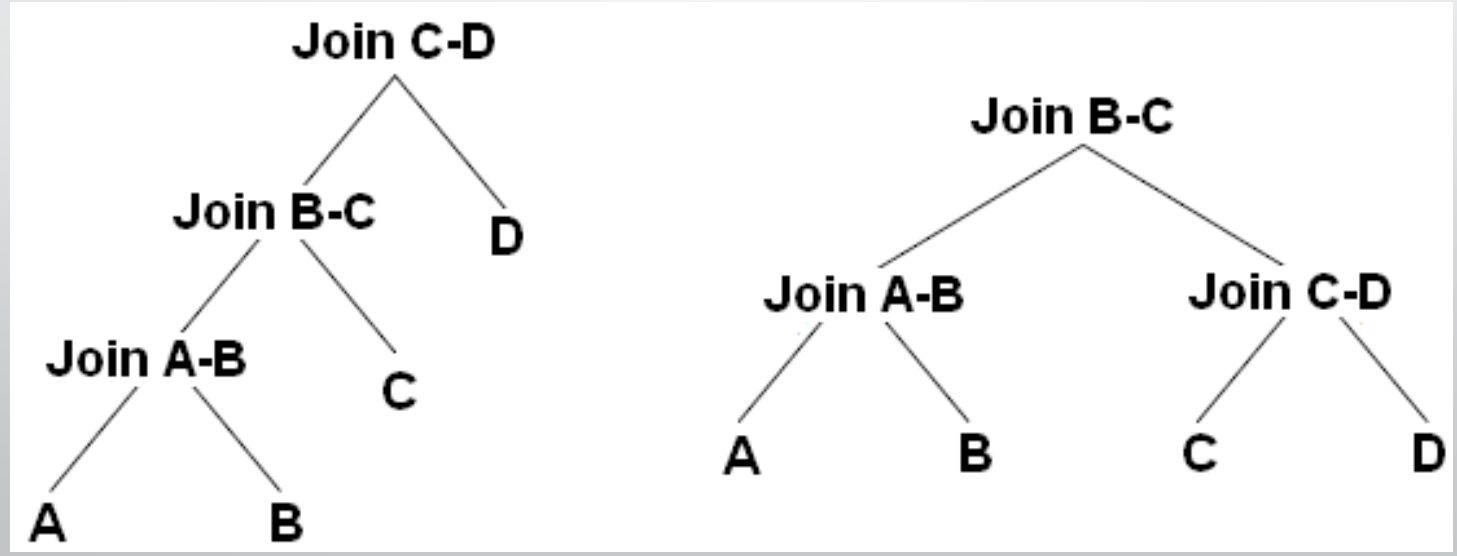
- Proces optimizacije može biti skup za inicijalizaciju i pokretanje za jednostavne upite koji ne zahtevaju nikakvu procenu troškova.
- Da bi izbegao ovu skupu operaciju za jednostavne upite, SQL Server koristi trivijalnu optimizaciju plana.

Join reordering

- Redosled spajanja je jedan od najsloženijih problema u optimizaciji upita.
- To je proces izračunavanja optimalnog redosleda spajanja (odnosno redosleda u kome su potrebne tabele spojene) prilikom izvršavanja upita.
- Redosled spajanja ključni faktor u kontroli količine podataka koji teku između svakog operatera u planu izvršenja, to je faktor na koji optimizator upita treba da obrati veliku pažnju.
- Redosled pridruživanja je direktno povezan sa veličinom prostora za pretragu jer broj mogućih planova za upit eksonencialno raste u odnosu na broj spojenih tabela.

Join reordering

- Neka stabala imaju svoje nazive:
  - left-deep
  - right-deep
  - bushy tree



Join reordering

- Zbog velikog broja mogućih opcija se ne analizijaju sve opcije, već samo neke.
- Na osnovu heuristike se biraju neke od opcija i pronalazi se dovoljno dobro rešenje.

Tables	Left-Deep Trees	Bushy Trees
2	2	2
3	6	12
4	24	120
5	120	1,680
6	720	30,240
7	5,040	665,280
8	40,320	17,297,280
9	362,880	518,918,400
10	3,628,800	17,643,225,600
11	39,916,800	670,442,572,800
12	479,001,600	28,158,588,057,600

Transformation rules

- Pravila transformaciji
  - Simplification rules
    - U okviru faze za pojednostavnjihanje upita se rade ove transformacije.
  - Exploration rules (logical transformation rules)
    - Pronalazi alternativne planove izvršavanja na logičkom nivou
  - Implementation rules (phisical transformation rules)
    - Pronalazi alternativna planove izvršavanja na fizičkom nivou
- Na primer, logički izraz može biti definicija logičkog spajanja, dok fizički izraz može biti stvarna implementacija, kao što je Merge join ili Hash join.

Transformation rules

- Optimizator upita koristi skupove pravila transformacije (komutativnost, asociativnost, ...) da generiše i ispita moguće alternativne planove izvršenja.
- Primena pravila ne smanjuje nužno cenu generisanih alternativa, a komponenta obračuna troškova i dalje treba da proceni njihove troškove.
- Iako se i logičke i fizičke alternative čuvaju u Memo strukturi, samo se fizičkim alternativama utvrđuju troškovi.
- Iako alternative mogu biti ekvivalentne i dati iste rezultate, njihove fizičke implementacije mogu imati veoma različite troškove.
- Konačni izbor, biće najjeftinija fizička alternativa sačuvana u Memo strukturi.

Transformation rules

- `sys.dm_exec_query_transformation_stats` DMV prikazuje informacije o 439 transformacije koje se primenjuju
- `sys.dm_exec_query_optimizer_info` DMV prikazuje informacije o izvršenim transformacijama

```
SELECT * FROM sys.dm_exec_query_transformation_stats
```

Name	promise_total	promise_avg	promised	built_substitute	Succeeded
JNtoNL	203352155	92.96051768	2187511	350792	285,532
LOJNtoNL	35938188	449.0701754	80028	80028	79,504
LSJNtoNL	40614706	450.6936171	90116	90116	90,116
LASJNtoNL	4029039	451.6353548	8921	8921	8,921
JNtoSM	366499276	418.7495941	875223	814754	441,413
FOJNtoSM	6356	454	14	14	4
LOJNtoSM	11180944	443.3891422	25217	24996	17,098
ROJNtoSM	11179128	443.3874589	25213	24992	17,094
LSJNtoSM	4263232	443.2094812	9619	9453	1,495
RSJNtoSM	4263232	443.2094812	9619	9453	6,922

The memo

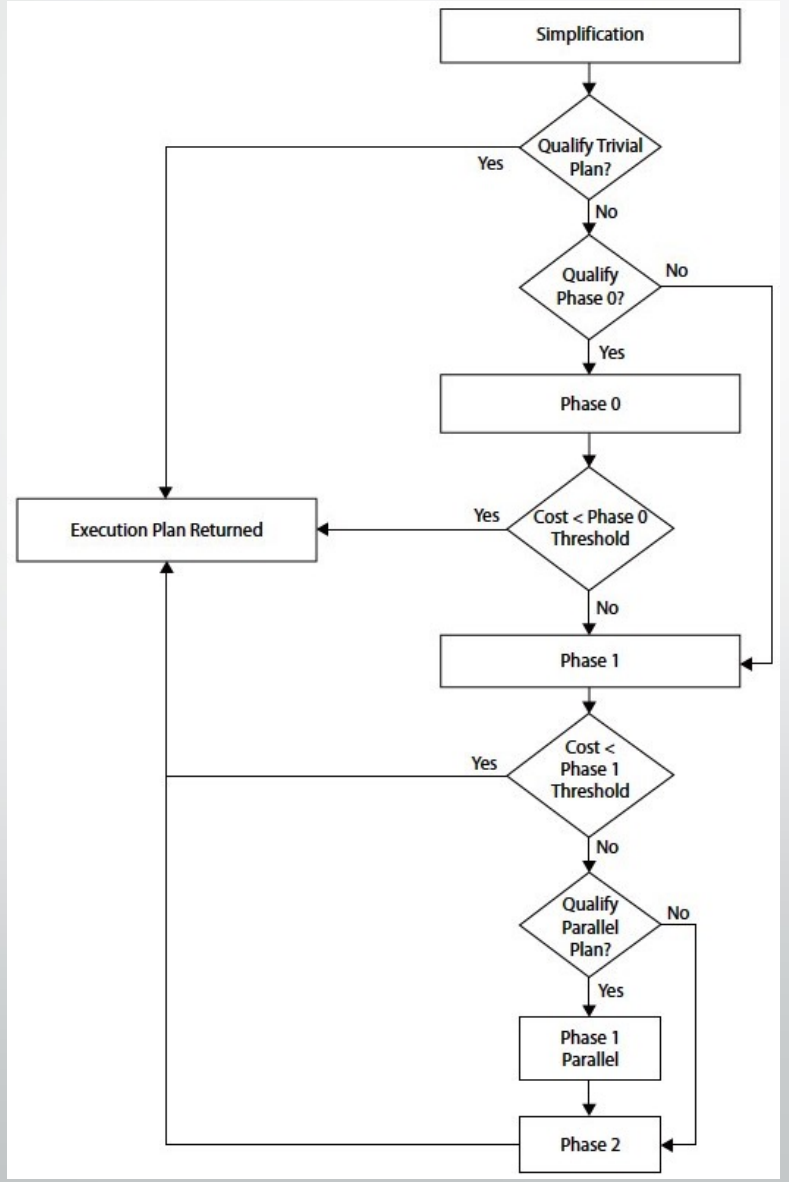
- Memo je struktura podataka koja se koristi za čuvanje alternativa koje generiše i analizira Optimizator upita.
- Ove alternative mogu biti logički ili fizički operatori i organizovane su u grupe ekvivalentnih alternativa, tako da svaka alternativa u istoj grupi daje iste rezultate. Alternative u istoj grupi takođe dele ista logička svojstva.
- Nova Memo struktura se kreira za svaku optimizaciju. Optimizator upita prvo kopira logičke izraze originalnog stabla upita u Memo strukturu, postavljajući svaki operator iz stabla upita u sopstvenu grupu, a zatim pokreće ceo proces optimizacije.
- Tokom ovog procesa primenjuju se pravila transformacije za generisanje svih alternativa. Alternative se dodaju u odgovarajuću grupu ili se kreiraju nove grupe, ukoliko novi izraz nije ekvivalentan nijednoj postojećoj grupi.

Broj ovih alternativa i grupa u Memo strukturi može biti ogroman.

Full optimization

- Ako se upit ne kvalifikuje za trivijalni plan, SQL Server će pokrenuti proces optimizacije zasnovan na troškovima, koji koristi pravila transformacije za generisanje alternativnih planova, čuva ove alternative u strukturi Memo i koristi procenu troškova za odabir najboljeg plana.
- Ovaj proces optimizacije se može izvršiti u do tri faze, sa različitim pravilima transformacije koja se primenjuju u svakoj fazi.
- Pored primene pravila transformacije, Optimizator upita koristi nekoliko heuristika da kontroliše strategiju pretrage i da ograniči broj generisanih alternativa za brzo pronalaženje dobrog plana. Optimizator upita treba da uravnoteži vreme optimizacije i kvalitet izabranog plana.
- Na kraju bilo koje od tri faze optimizacije, proces se završava ako se pronađe dovoljno dobar plan. Međutim, ako je na kraju bilo koje faze najbolji plan i dalje veoma skup, optimizator upita će pokrenuti sledeću fazu, koja će pokrenuti dodatni skup pravila transformacije.

Full optimization



Full optimization

- Search 0 - Transaction processing phase
  - Cilj ove faze je da pronađe plan što je brže moguće bez naprednih transformacija.
  - Idealna je za male upite koji se obično nalaze u sistemima za obradu transakcija i koristi se za upite sa najmanje tri tabele.
  - Pre nego što se pokrene potpuni proces optimizacije, Optimizator upita generiše inicijalni skup redosleda join operacija na osnovu heuristika. Ove heuristike prvo spajaju tabele sa najmanje podataka ili tabele sa velikim stepenom filtriranja. U ovom fazi se ne razmatraju drugi redosledi spajanja.
  - Na kraju ove faze, optimizator upita upoređuje cenu najbolje generisanog plana sa internim threshold-om. Ako je najbolji plan dovoljno jeftin, on se bira.

Full optimization

- Search 1 – Quick plan
  - Koristi dodatna pravila transformacije, i ograničeno reorganizovanje JOIN operatora, i prikladno je za složenije upite.
  - Na kraju ove faze, SQL Server upoređuje cenu najjeftinijeg plana sa threshold-om. Ako je najbolji plan dovoljno jeftin, on se bira.
  - Ako je upit i dalje skup i sistem može da pokreće paralelne upite, ova faza se ponovo izvršava da bi se pronašao dobar paralelni plan.
  - Na kraju ove faze upoređuju se troškovi najboljeg serijskog i paralelnog plana, a najjeftiniji se koristi u sledećoj fazi.

Full optimization

- Search 2 – Full optimization
  - Koristi se za složene upite.
  - U ovoj fazi se razmatra veći skup potencijalnih pravila transformacije, paralelni operatori i druge napredne strategije optimizacije. Pošto je ovo poslednja faza, ovde se mora naći plan izvršenja.
  - Proces optimizacije takođe uključuje koncept budžeta troškova optimizacije. Kada se ovaj budžet premaši, potraga za optimalnim planom se prekida, a optimizator upita će prikazati vremensko ograničenje optimizacije. Ovo vremensko ograničenje nije fiksno vreme, već se umesto toga izračunava na osnovu broja primenjenih transformacija zajedno sa proteklim vremenom.
  - Kada se premaši vremensko ograničenje, proces optimizacije se zaustavlja i vraća se najjeftiniji plan koji je do tada pronadjen. Taj plan može biti iz ove faze ili iz neke prethodne faze.

The Execution Engine

- Data access operators
- Aggregations
- Joins
- Parallelism
- Updates

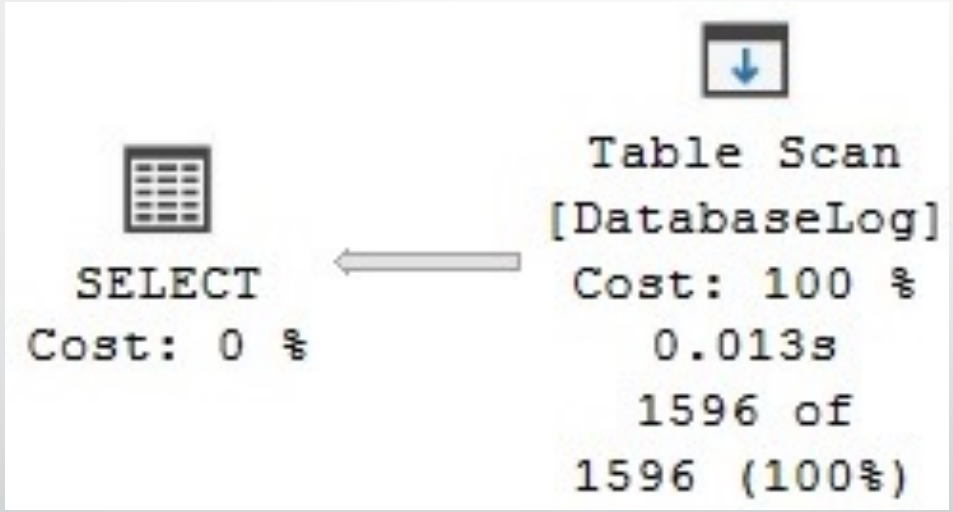
Data access operators

Structure	Scan	Seek
Heap	Table scan	
Clustered index	Clusted Index scan	Clusted Index Seek
Nonclustered index	Index Scan	Index Seek

Data access operators

- Table Scan:

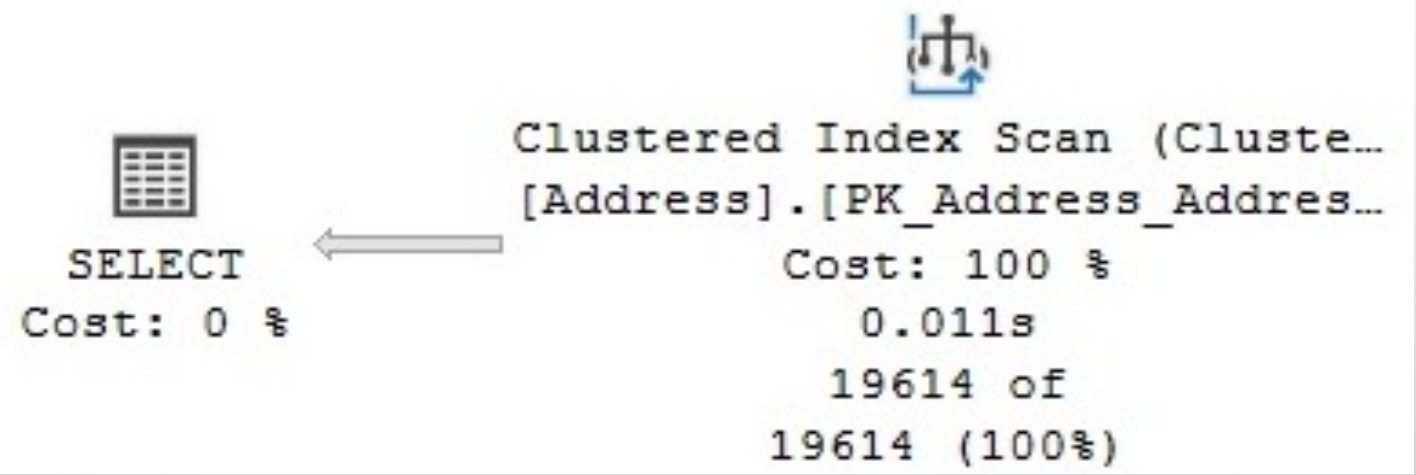
```
SELECT * FROM DatabaseLog
```



Data access operators

- Clustered Index scan:

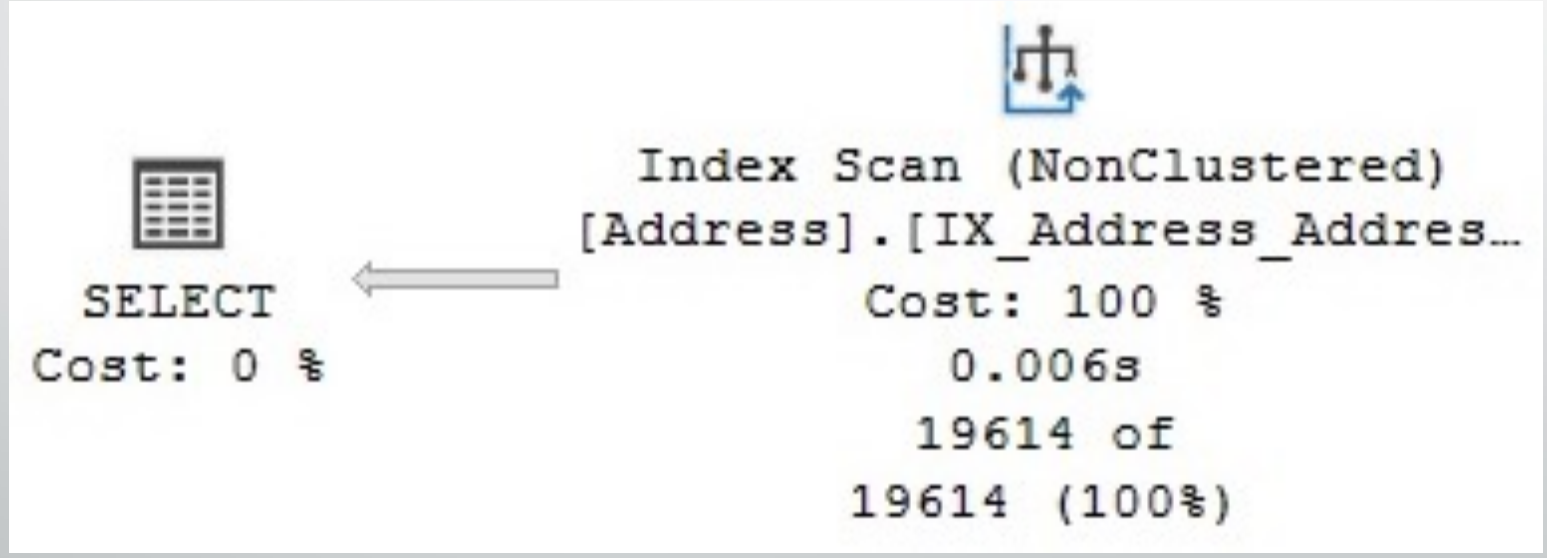
```
SELECT * FROM Person.Address
```



Data access operators

- Index scan:

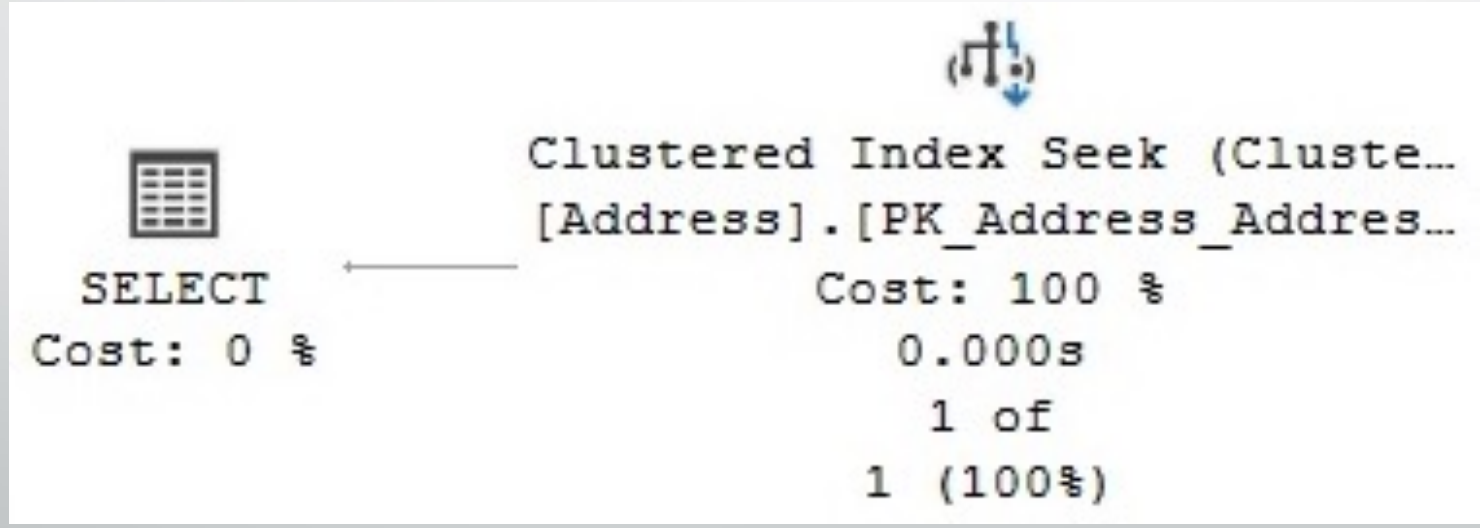
```
SELECT AddressID, City, StateProvinceID
FROM Person.Address
```



Data access operators

- Clustered Index Seek:

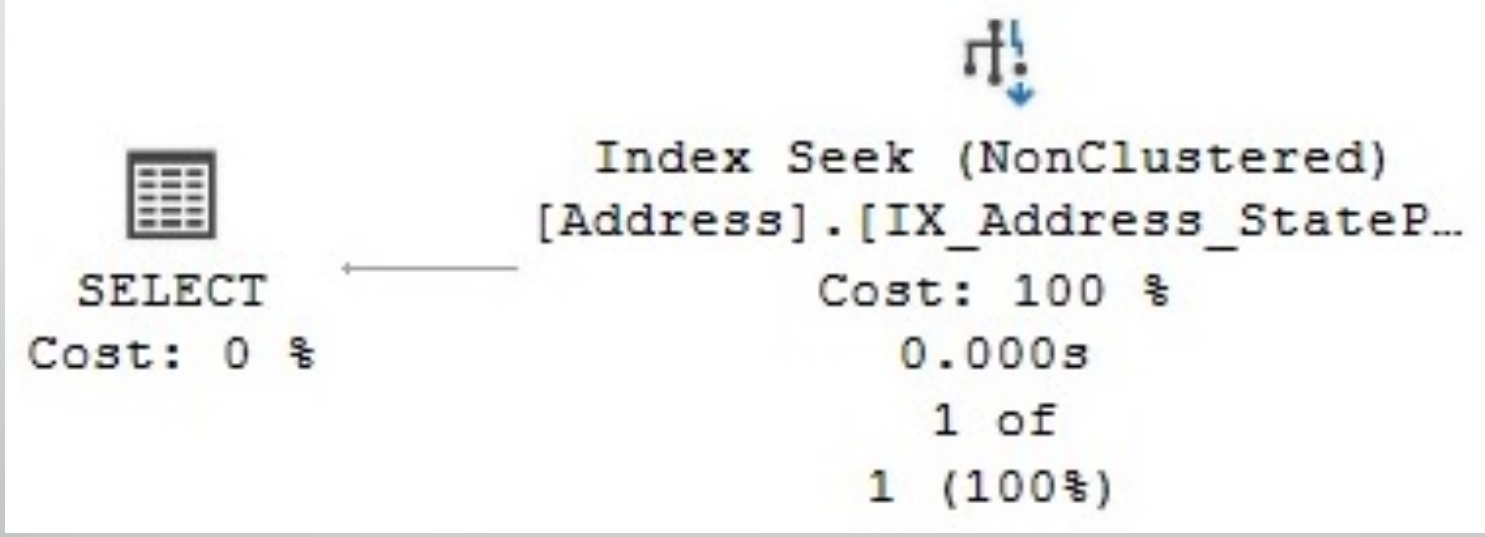
```
SELECT AddressID, City, StateProvinceID
FROM Person.Address
WHERE AddressID = 12037
```



Data access operators

- Index Seek:

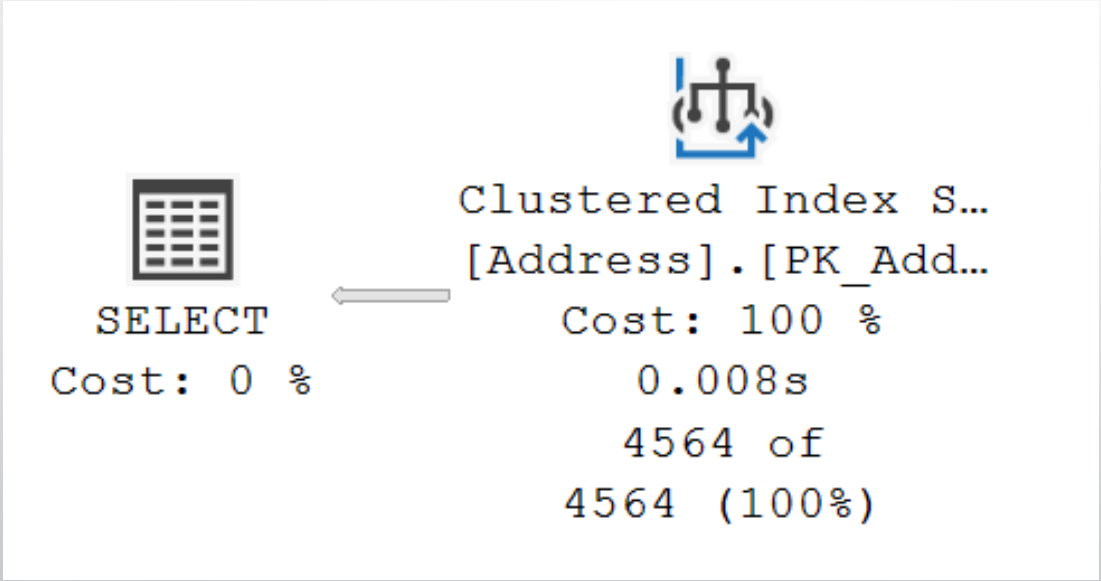
```
SELECT AddressID, StateProvinceID
FROM Person.Address
WHERE StateProvinceID = 32
```



Data access operators

- Primer gde indeksa nedostaje:

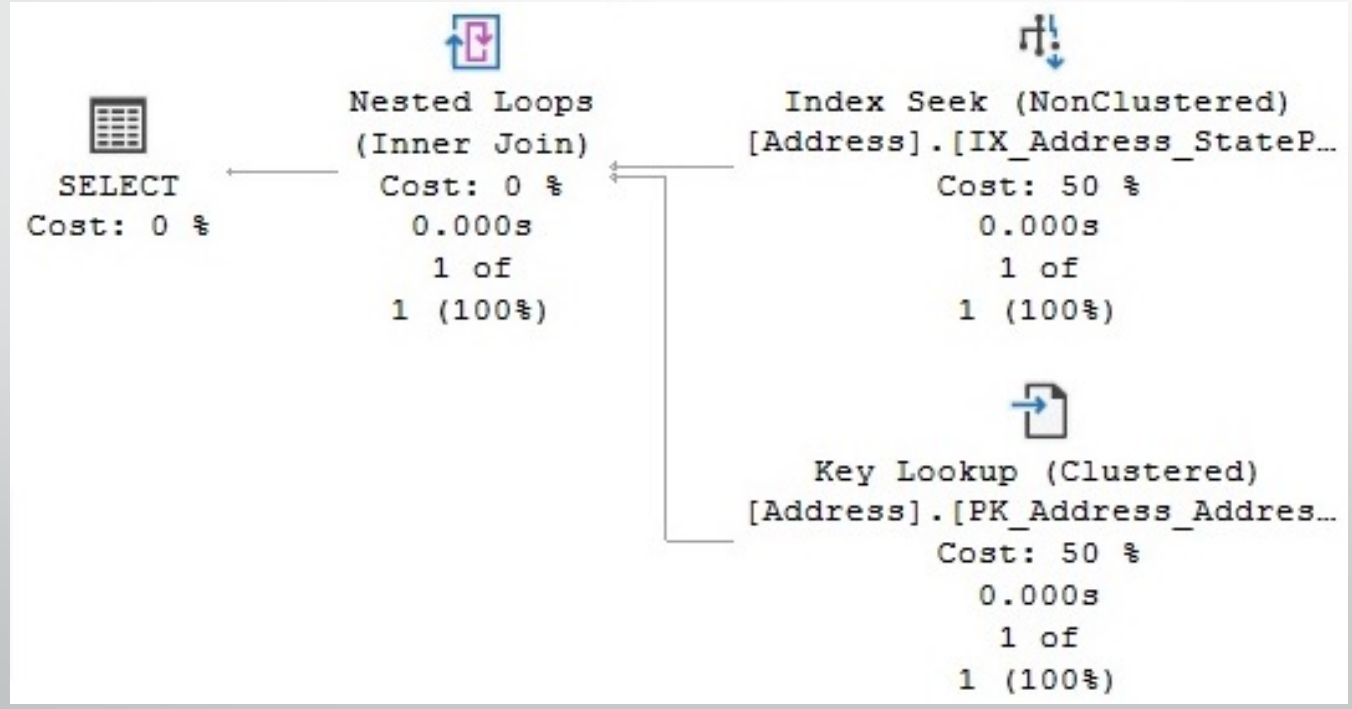
```
SELECT *
FROM Person.Address
WHERE StateProvinceID = 9
```



Data access operators

- Bookmark lookup

```
SELECT AddressID, City, StateProvinceID, ModifiedDate
FROM Person.Address
WHERE StateProvinceID = 32
```



Aggregations

- Stream Aggregate
- Hash Aggregate
- Distinct sort

Aggregations

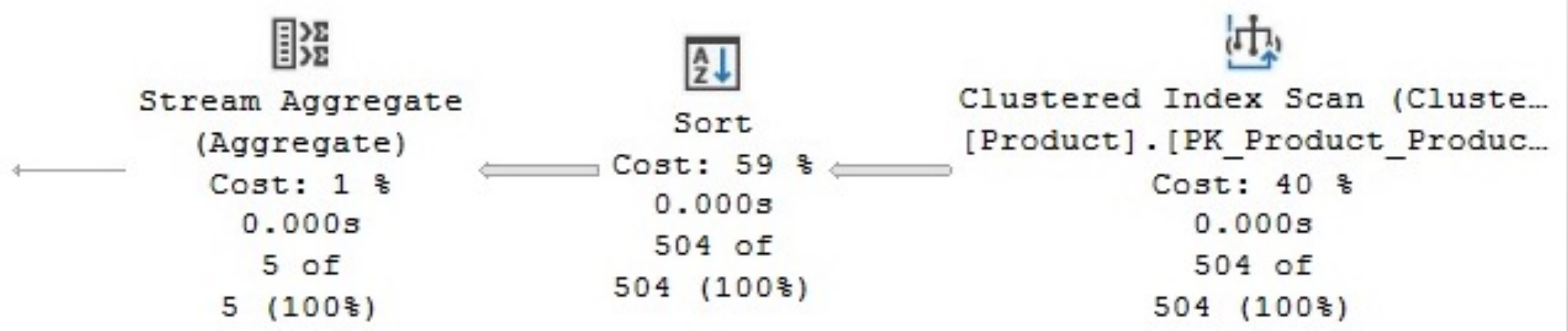
- Stream Aggregate

```
SELECT AVG(ListPrice)
FROM Production.Product
```

Aggregations

- Stream Aggregate

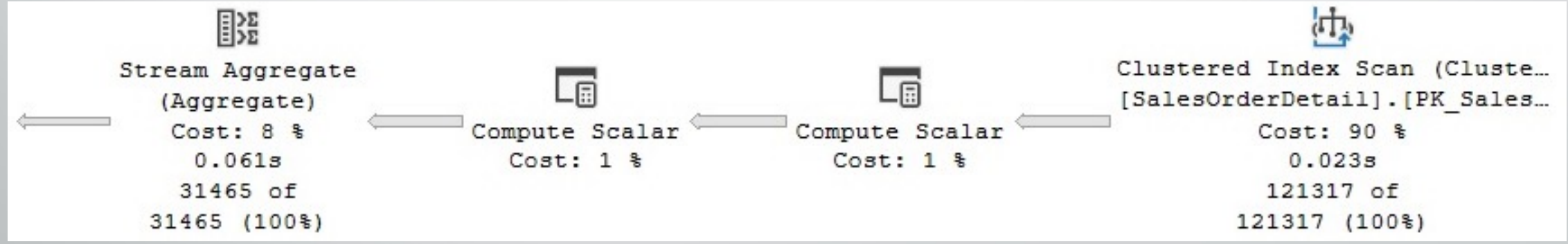
```
SELECT ProductLine, COUNT(*)
FROM Production.Product
GROUP BY ProductLine
```



Aggregations

- Stream Aggregate

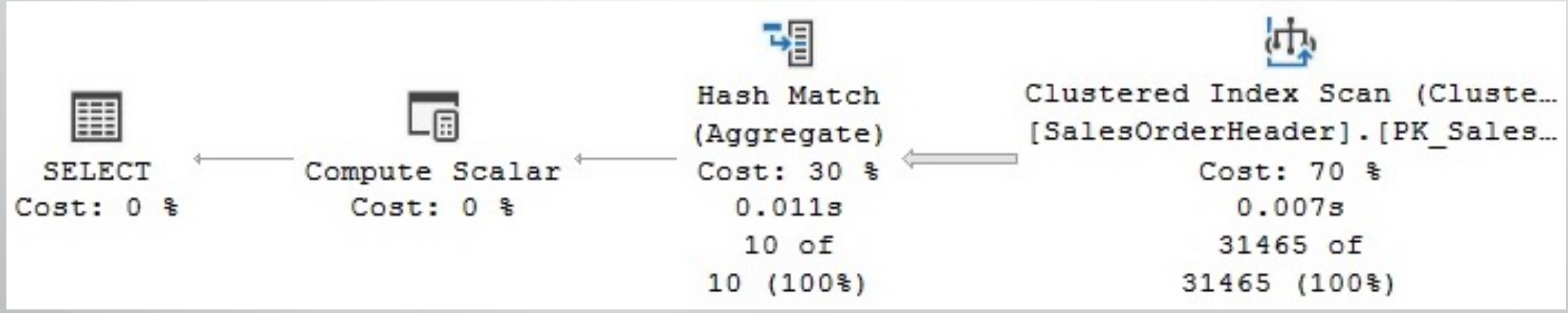
```
SELECT SalesOrderID, SUM(LineTotal)
FROM Sales.SalesOrderDetail
GROUP BY SalesOrderID
```



Aggregations

- Hash Aggregate

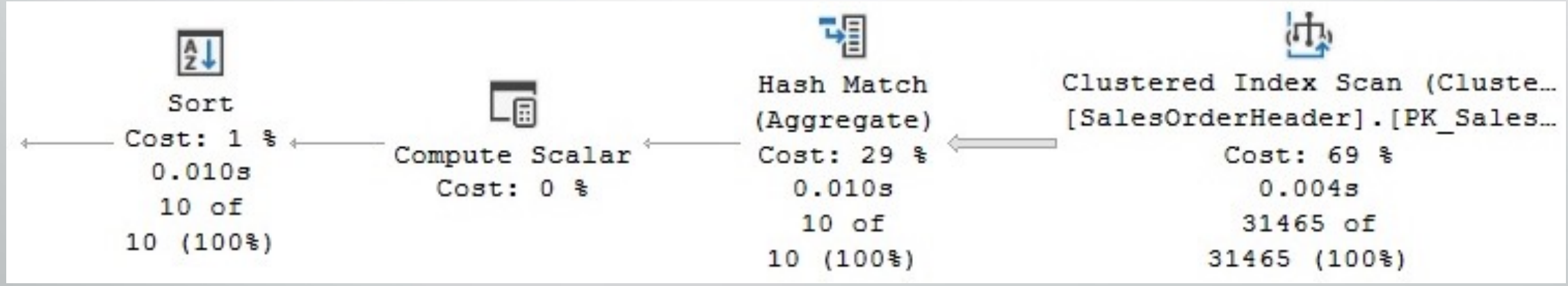
```
SELECT TerritoryID, COUNT(*)
FROM Sales.SalesOrderHeader
GROUP BY TerritoryID
```



Aggregations

- Hash Aggregate vs Stream Aggregate

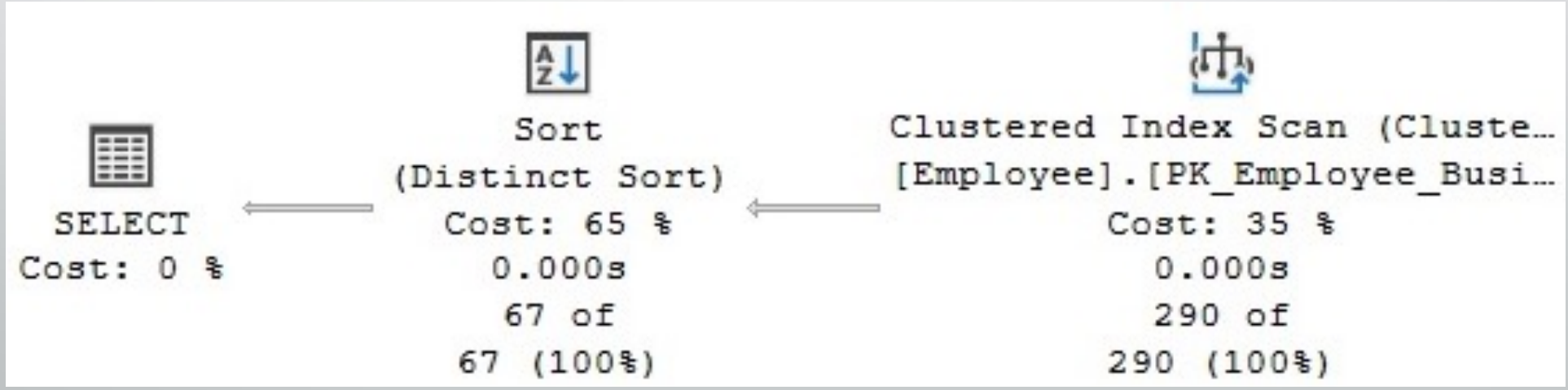
```
SELECT TerritoryID, COUNT(*)
FROM Sales.SalesOrderHeader
GROUP BY TerritoryID
ORDER BY TerritoryID
```



Aggregations

- Distinct sort
  - Da li postoji razlika izmedju group by i distinct?

```
SELECT DISTINCT(JobTitle)
FROM HumanResources.Employee
```



Aggregations

- Distinct sort vs Hash Aggregate
  - Koja je razlika kad postoji indeks i kad ne postoji?

```
CREATE INDEX IX_JobTitle ON HumanResources.Employee(JobTitle)

SELECT DISTINCT(JobTitle)
FROM HumanResources.Employee

DROP INDEX HumanResources.Employee.IX_JobTitle
```

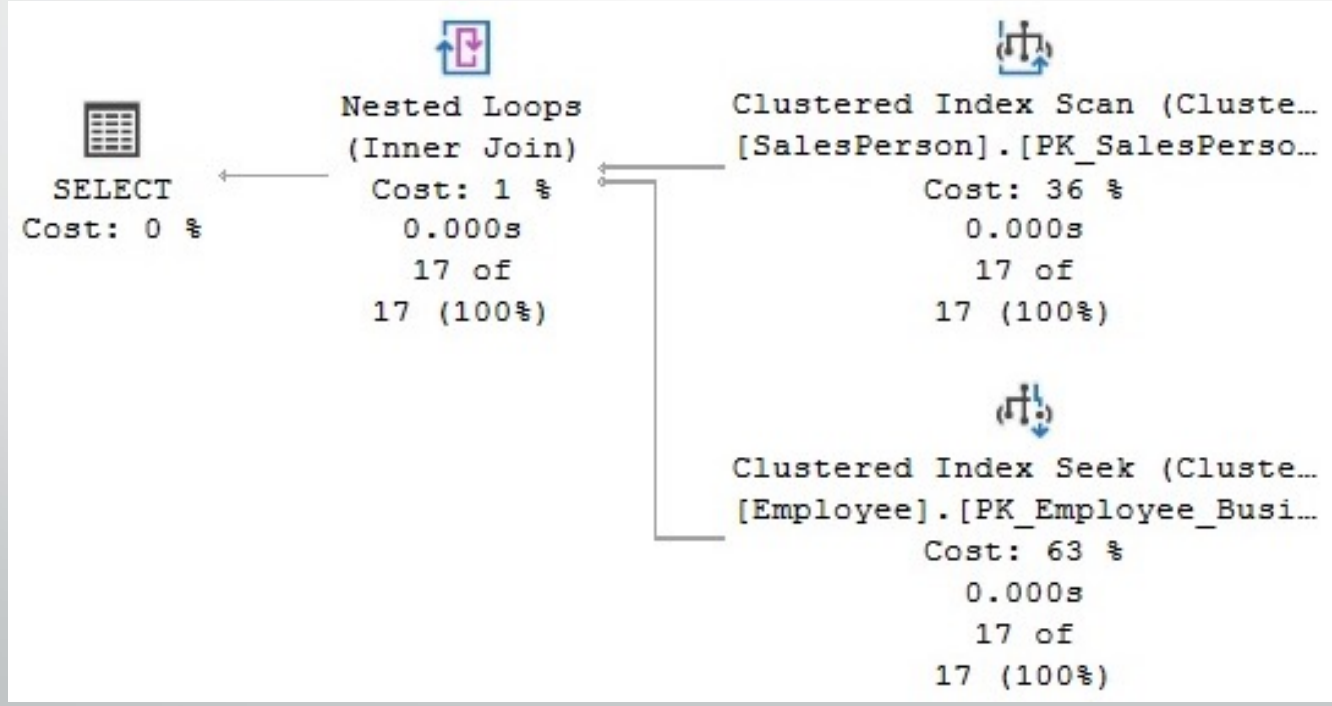
Joins

- Nested Loops Join
- Merge Join
- Hash Join

Joins

- Nested Loops Join

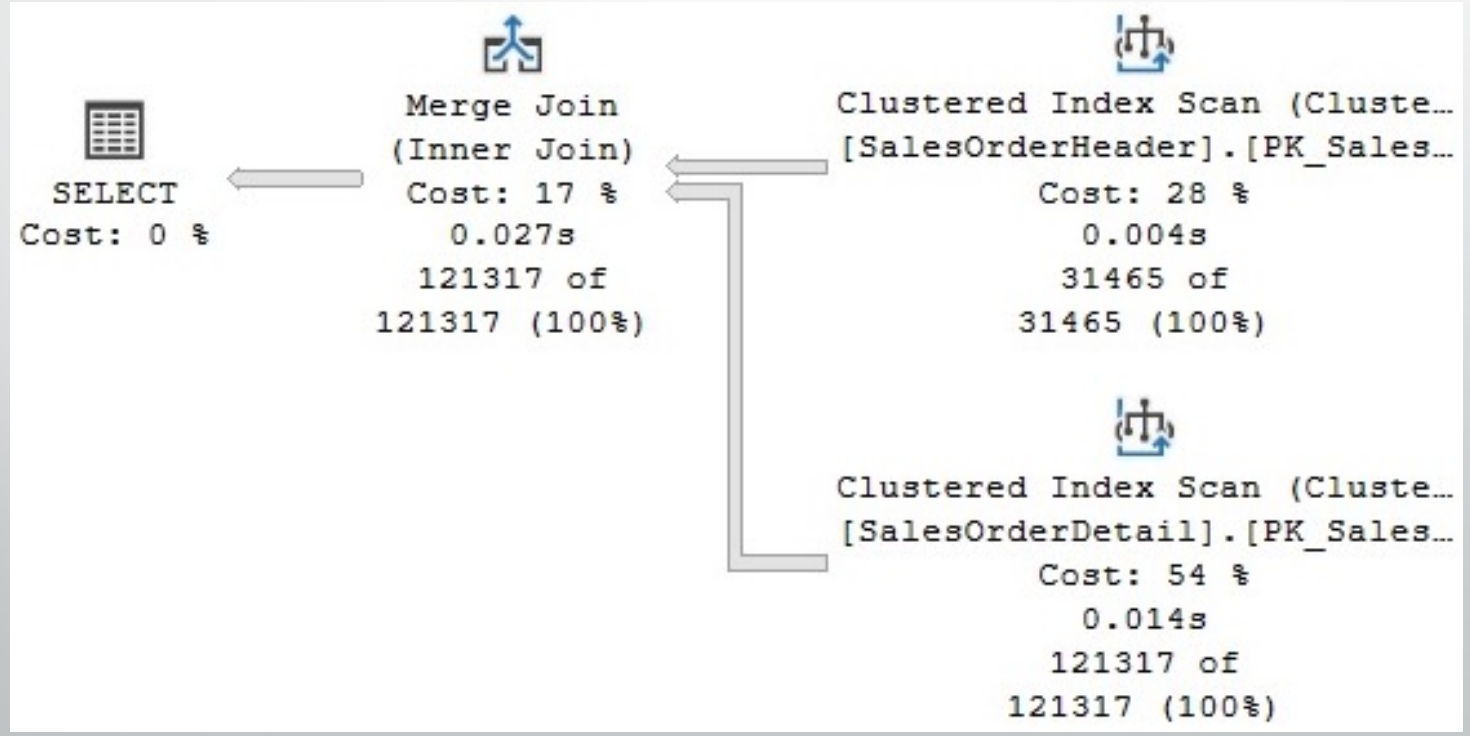
```
SELECT e.BusinessEntityID, TerritoryID
FROM HumanResources.Employee AS e JOIN Sales.SalesPerson AS s
     ON e.BusinessEntityID = s.BusinessEntityID
```



Joins

- Merge Join

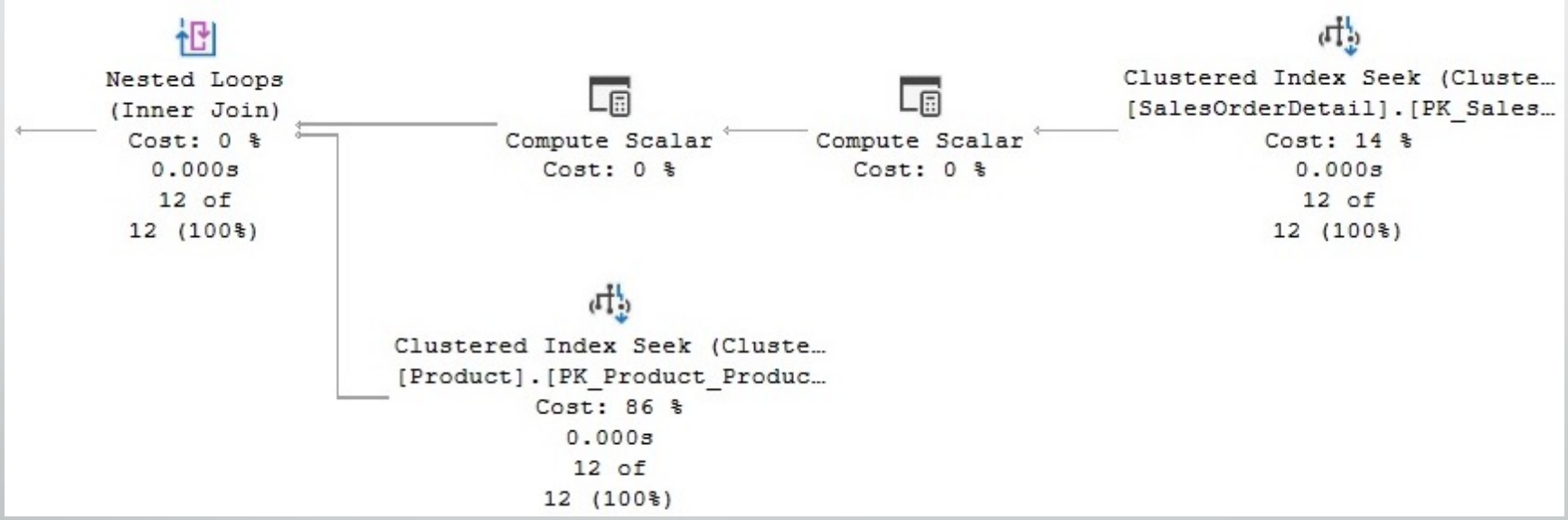
```
SELECT h.SalesOrderID, s.SalesOrderDetailID, OrderDate
FROM Sales.SalesOrderHeader h JOIN Sales.SalesOrderDetail s
    ON h.SalesOrderID = s.SalesOrderID
```



Joins

- Nested Loops Join vs Merge Join

```
SELECT *
FROM Sales.SalesOrderDetail s JOIN Production.Product p
     ON s.ProductID = p.ProductID
WHERE SalesOrderID = 43659
```

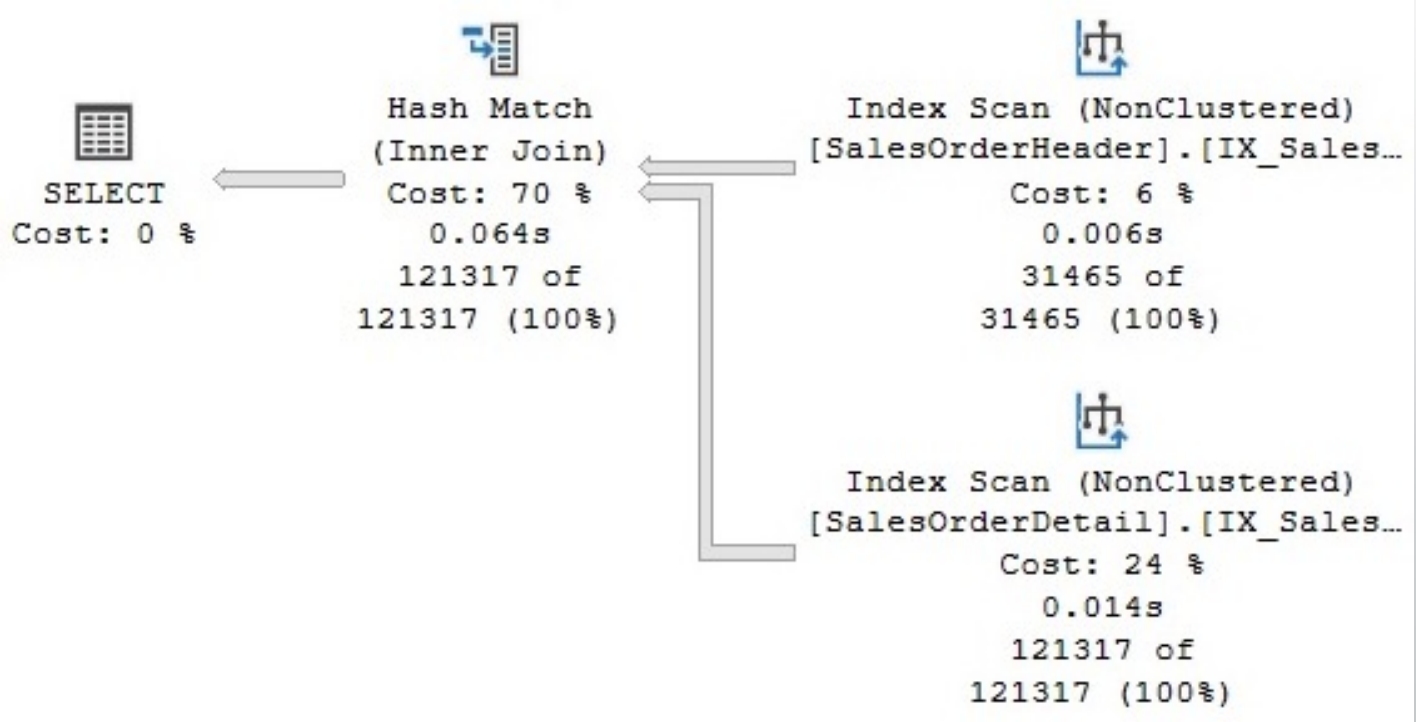


- Kakav bi bio rezultat da je kada bi proširili upit sa `OPTION (MERGE JOIN)`?

Joins

- Hash Join

```
SELECT h.SalesOrderID, s.SalesOrderDetailID
FROM Sales.SalesOrderHeader h JOIN Sales.SalesOrderDetail s
     ON h.SalesOrderID = s.SalesOrderID
```



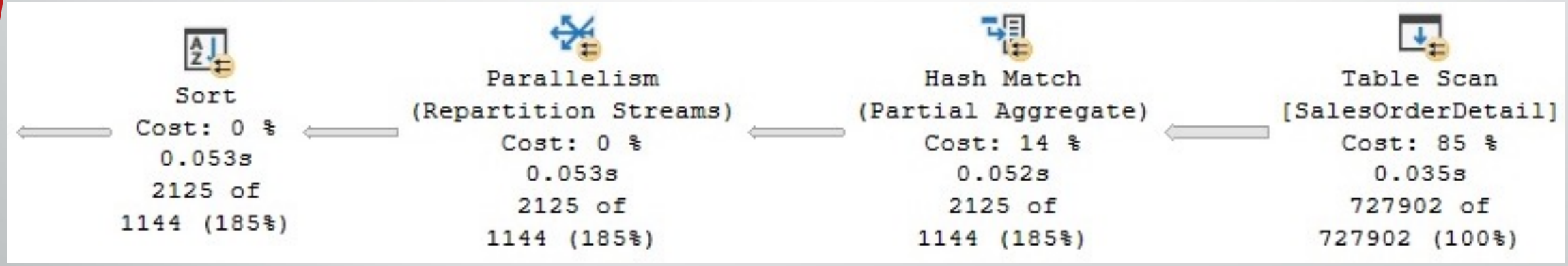
Parallelism

- Za prikaz paralelizacije, potrebno je izgenerisati dodatne podatke:

```
SELECT *
INTO #temp
FROM Sales.SalesOrderDetail
UNION ALL SELECT * FROM Sales.SalesOrderDetail
UNION ALL SELECT * FROM Sales.SalesOrderDetail
UNION ALL SELECT * FROM Sales.SalesOrderDetail
UNION ALL SELECT * FROM Sales.SalesOrderDetail
UNION ALL SELECT * FROM Sales.SalesOrderDetail

SELECT IDENTITY(int, 1, 1) AS ID, CarrierTrackingNumber,
OrderQty, ProductID,
UnitPrice, LineTotal, rowguid, ModifiedDate
INTO dbo.SalesOrderDetail FROM #temp

SELECT IDENTITY(int, 1, 1) AS ID, CarrierTrackingNumber,
OrderQty, ProductID,
UnitPrice, LineTotal, rowguid, ModifiedDate
INTO dbo.SalesOrderDetail2 FROM #temp

DROP TABLE #temp
```

Parallelism

```
SELECT ProductID, COUNT(*)
FROM dbo.SalesOrderDetail
GROUP BY ProductID
```



- Kakav bi rezultat bio da je kada bi proširili upit sa `OPTION (MAXDOP 1)`?

Parallelism

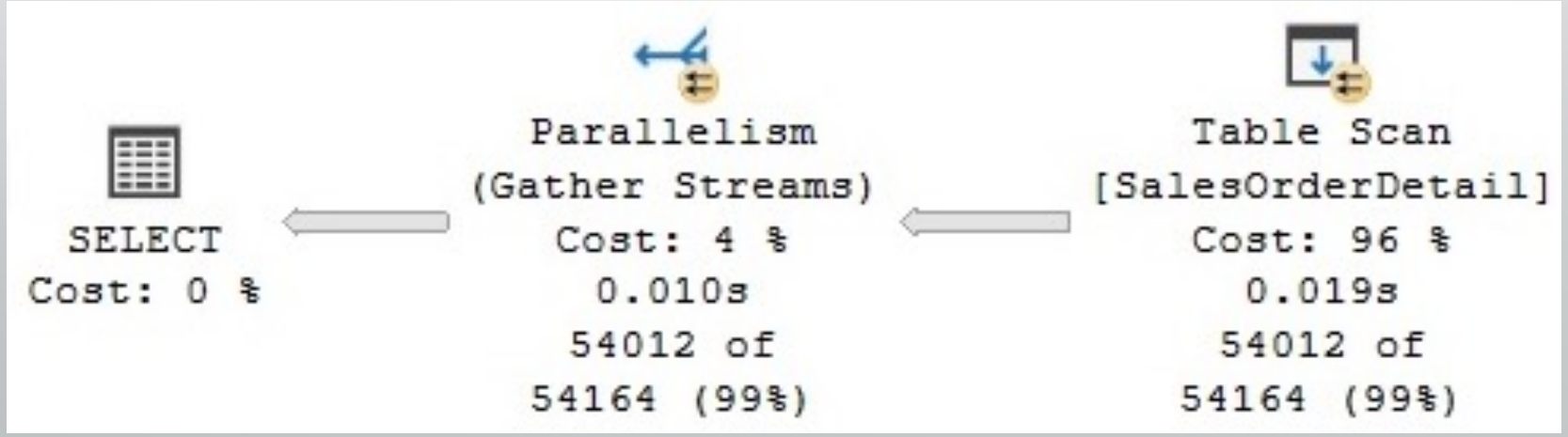
- Moguće je podesiti minimalnu cenu upita za koju će se pokušati paralelizacija upita.
  - Podrazumevana vrednost cene za koju je se radi paralelizacija je 5

```
EXEC sp_configure 'cost threshold for parallelism', 10
GO
RECONFIGURE
GO
```

Parallelism

- Exchange operators
  - Gather Streams exchange operator
  - Distribute Streams exchange operator
  - Repartition Stream exchange operators

```
SELECT * FROM dbo.SalesOrderDetail
WHERE LineTotal > 3234
```



Parallelism

- Exchange operators
  - Exchange operatoros prosledjuju podatke ostalim operatorima, jer ostali operatori ni ne znaju da se trenutno izvršavaju u paraleli
  - Partition types:
    - Hash
    - Round robin
    - Broadcast
    - Demand
    - Range

Parallelism

- Exchange operators
  - Merging ili order-preserving exchange - Exchange operatoros koji imaju svojsto da očuvaju redosled podataka
  - Non-merging ili non-order-preserving exchanges - Exchange operatoros koji ne čuvaju redosled podataka

```
SELECT ProductID, COUNT(*)
FROM dbo.SalesOrderDetail
GROUP BY ProductID
```

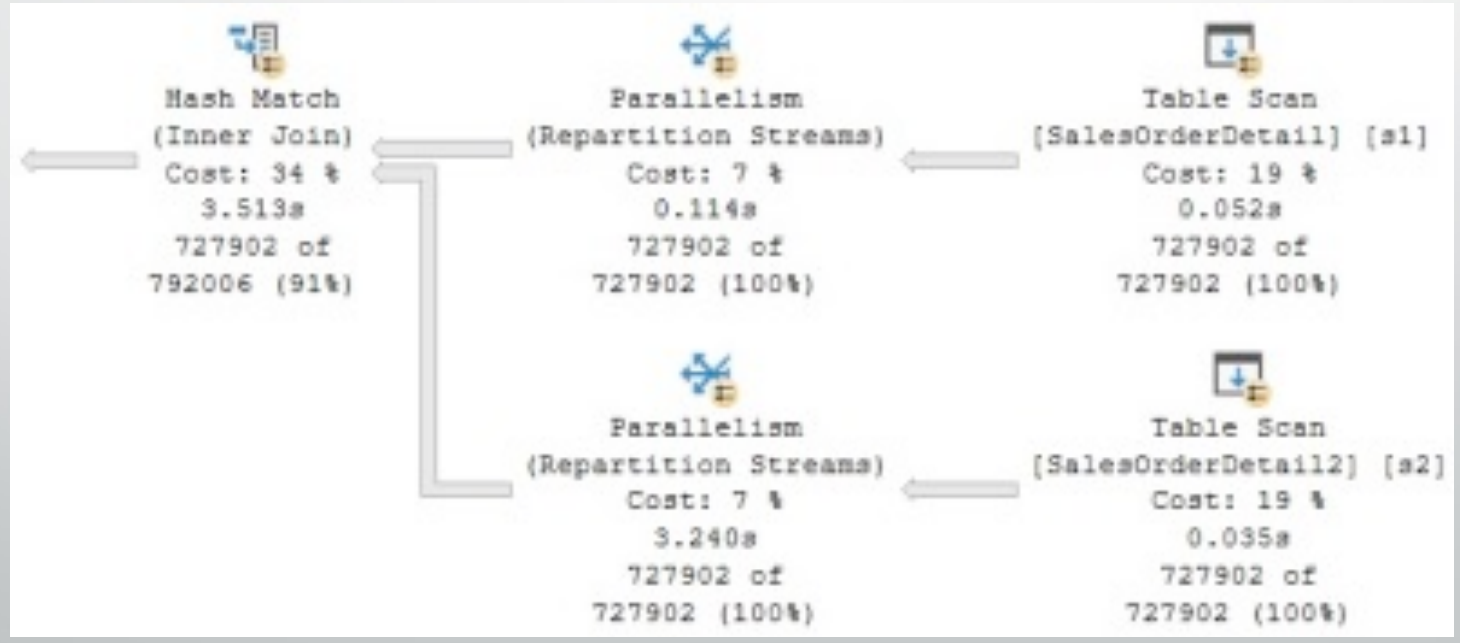
VS

```
SELECT ProductID, COUNT(*)
FROM dbo.SalesOrderDetail
GROUP BY ProductID
ORDER BY ProductID
```

Parallelism

- Hash partitioning

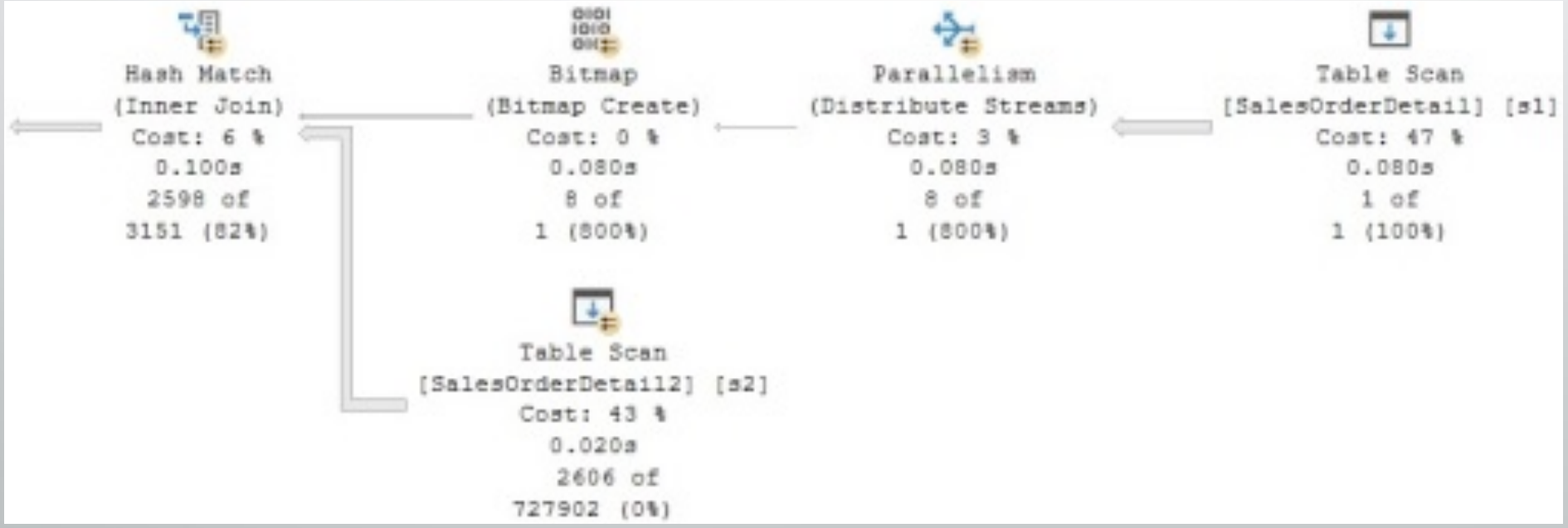
```
SELECT * FROM dbo.SalesOrderDetail s1
JOIN dbo.SalesOrderDetail2 s2
ON s1.id = s2.id
```



Parallelism

- Broadcast partitioning

```
SELECT * FROM dbo.SalesOrderDetail s1
JOIN dbo.SalesOrderDetail2 s2 ON s1.ProductID = s2.ProductID
WHERE s1.id = 123
```



Parallelism

- Nije uvek moguće paralelizovati upit.
- Sekvencialno se izvršavaju upiti koji sadrže:
  - Scalar-valued user-defined functions
  - CLR user-defined functions with data access
  - Miscellaneous built-in functions such as `OBJECT_ID()`, `ERROR_NUMBER()` and `@@TRANCOUNT`
  - Dynamic cursors
- U paralelnim upitima se sekvencialno izvršavaju delovi koji sadrže:
  - Multistatement, table-valued, and user-defined functions
  - TOP clauses
  - Global scalar aggregates
  - Sequence functions
  - Multiconsumer spools
  - Backward scans
  - System table scans
  - Recursive queries

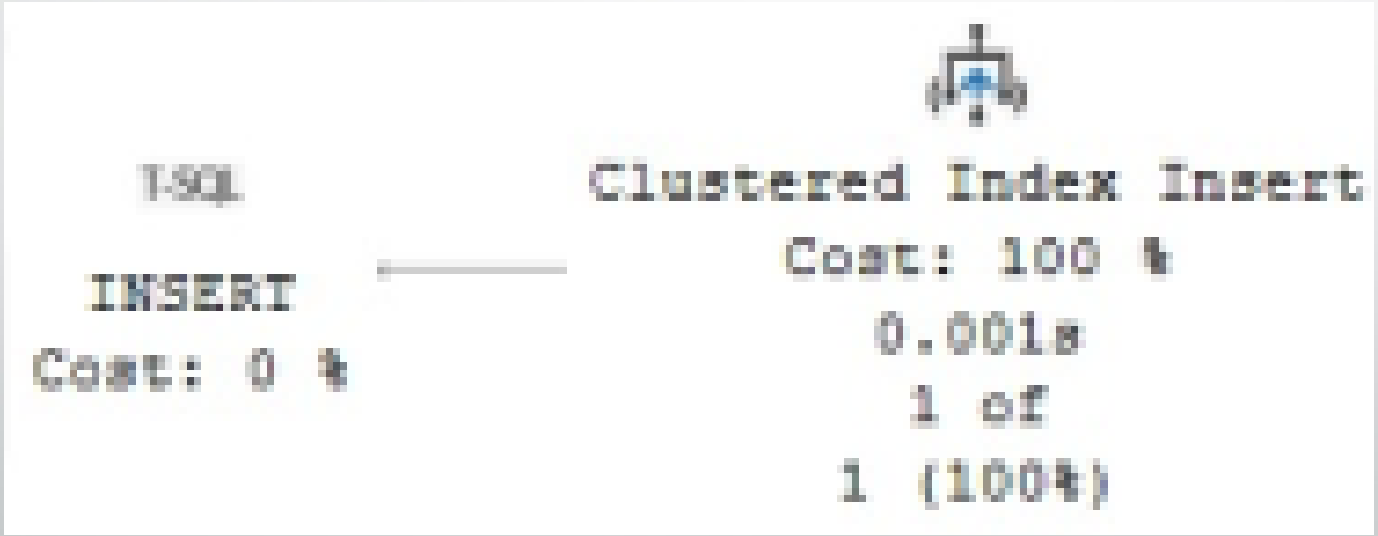
Updates

- Update operacija se izvršava u dva koraka, korak za čitanje i korak za ažuriranje podataka.
- Prvi korak prikuplja detalje o promenama koje treba primeniti i koji redovi će biti ažurirani.
  - Kod INSERT operacije prikupljaju se vrednosti koje treba dodati.
  - Kod DELETE operacije prikupljaju se ključevi redova koji se brišu.
  - Kod UPDATE operacije se primenjuje kombinacija prva dva.
- U drugom koraku se izvršavaju operacije ažuriranja, uključujući ažuriranje indeksa, validaciju ograničenja i izvršavanje okidača.
- Ako se prekrši bilo koje definisano ograničenje, kompletna operacija ažuriranja neće biti realizovana i vratiće se u početno stanje.

Updates

- INSERT

```
INSERT INTO Person.CountryRegion (CountryRegionCode, Name)
VALUES ('ZZ', 'New Country')
```



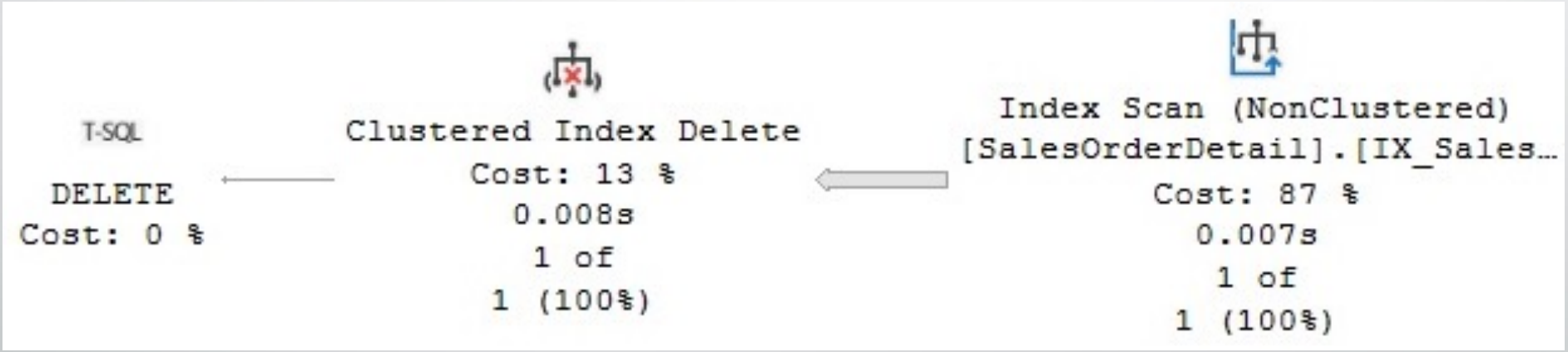
Updates

- Per-row and per-index plans
  - U per-row planu, ažuriranja osnovne tabele i postojećih indeksa vrši jedan operater, red po red.
  - U per-index planu, osnovna tabela i svaki unclustered index se ažuriraju u zasebnim operacijama.
  - Osim u nekim slučajevima kada su planovi po indeksu obavezni, optimizator upita bira koji od ova dva plana da iskoristi. Bira se na osnovu broja redova koje je potrebno ažurirati.

Updates

- Per-row plan

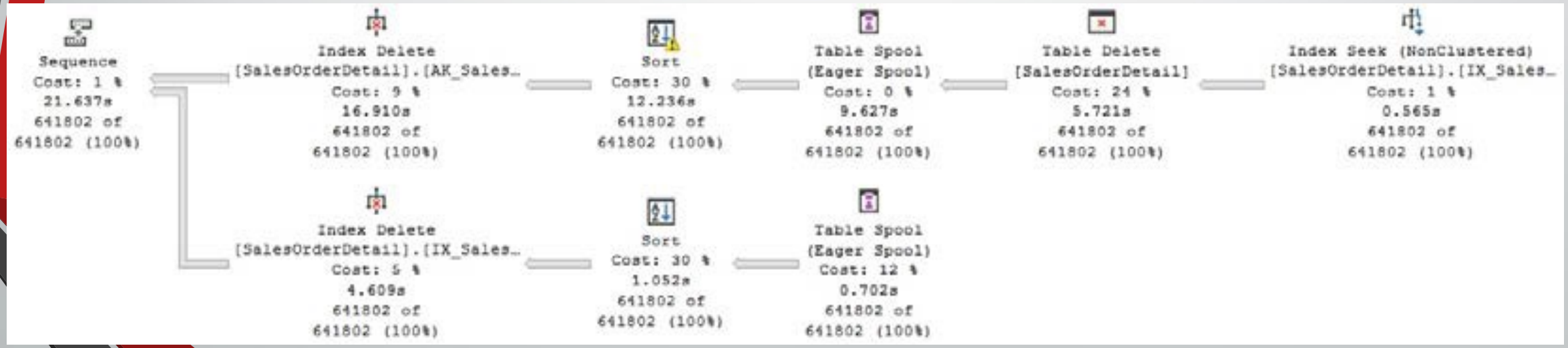
```
DELETE FROM Sales.SalesOrderDetail
WHERE SalesOrderDetailID = 61130
```



Updates

- Per-index plan

```
CREATE NONCLUSTERED INDEX AK_SalesOrderDetail_rowguid
ON dbo.SalesOrderDetail (rowguid)
CREATE NONCLUSTERED INDEX IX_SalesOrderDetail_ProductID
ON dbo.SalesOrderDetail (ProductID)

DELETE FROM dbo.SalesOrderDetail WHERE ProductID < 953
```



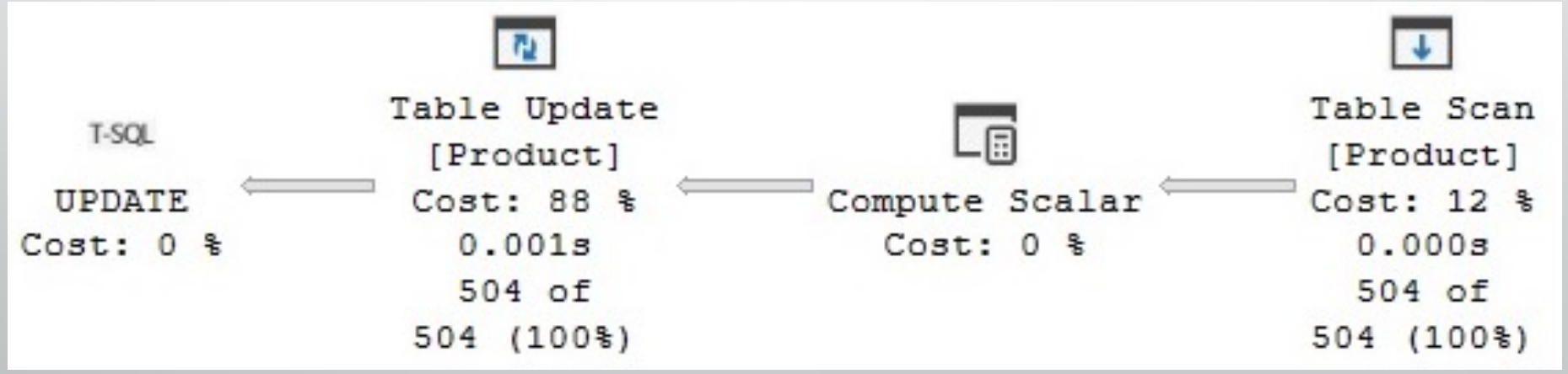
Updates

- Halloween protection
  - Sprečavanje ažuriranje već ažuriranih redova
  - Ovo je obezbedjuje, tako što korak u kome se čutaju podaci mora da se završi pre početka koraka u kome se ažurirajupodaci.

Updates

- Halloween protection
  - Ovde Halloween protection nije potrebna

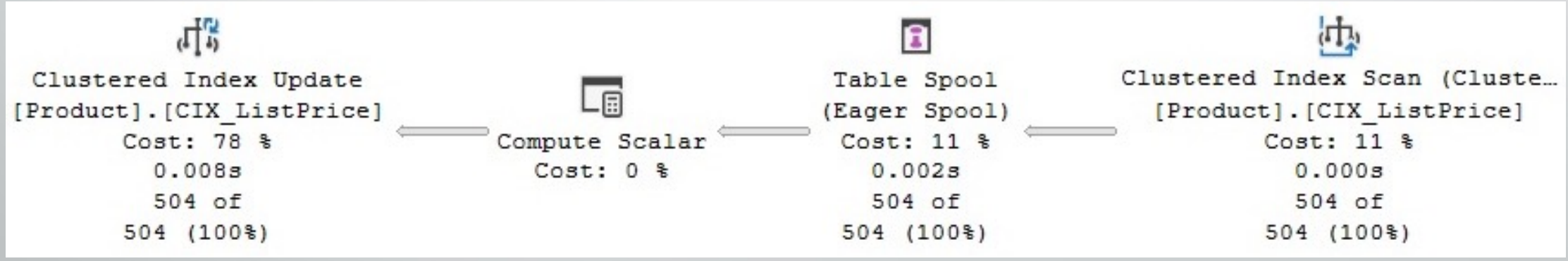
```
SELECT * INTO dbo.Product
FROM Production.Product
UPDATE dbo.Product
SET ListPrice = ListPrice * 1.2
```



Updates

- Halloween protection
  - Ovde je Halloween protection potrebna

```
CREATE CLUSTERED INDEX CIX_ListPrice ON dbo.Product(ListPrice)
UPDATE dbo.Product
SET ListPrice = ListPrice * 1.2
```



Indexes

- Indexes
- The Database Engine Tuning Advisor
- Missing indexes
- Index fragmentation
- Unused indexes

Indexes

- Indeksi se koriste kod scan i seek operacija
- Ubrzavaju izvršavanje upita

Indexes

- Heap
  - Struktura podataka u kojoj se redovi čuvaju bez konkretnog redosleda.
  - Tabela bez clustered index-a.
- Clustered index
  - Cela tabela se čuva logički sortirana prema određenom ključu.
  - Podaci se čuvaju u B^+ stablu. U listovima stabla se nalaze redovi tabele koji su povezani u dvostruko ulančanu listu.
- Nonclustered index
  - Sadrži ključ indeksa i pokazivač na red u osnovnoj tabeli.
  - Podaci se čuvaju u B^+ stablu.
  - Mogu da se kreiraju nad heap-om i nad clustered index-om.
  - Nad jednom tabelom, može da ih ima najviše 999, ali se preporučuje da ih ima što manje.
  - Opciono može da sadrži i druge kolone (klauzula INCLUDE), što dobrinosi boljoj pokrivenosti upita.

Indexes

- Unique index
  - Ne dozvoljava da dva reda imaju istu vrednost ključa.
  - Podrazumevano se kreira kao nonclustered index.
- Primary key
  - Ključ koji jedinstvreno identifikuje svaki red u tabeli.
  - Porazumevano se kreira kao unique clustered index.
  - Kolone koje čine primarni ključ ne smeju da sadrže NULL vrednosti.
  - Može da postoji najviše jedan primarni ključ.

Indexes

- Creating indexes

```
CREATE [UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX index_name
ON <object> ( column [ ASC | DESC ] [ ,...n ] )
[ INCLUDE ( column_name [ ,...n ] ) ]
[ WHERE <filter_predicate> ]
[ WITH ( <relational_index_option> [ ,...n ] ) ]
```

Indexes

- Heap

```
SELECT * INTO dbo.SalesOrderDetail
FROM Sales.SalesOrderDetail

SELECT * FROM sys.indexes
WHERE object_id = OBJECT_ID('dbo.SalesOrderDetail')
```

object_id	Name	index_id	type	type_desc	is_unique
1287675635	NULL	0	0	HEAP	0

Indexes

- Create nonclustered and clustered index

```
CREATE INDEX IX_ProductID ON dbo.SalesOrderDetail(ProductID)
```

object_id	Name	index_id	type	type_desc	is_unique
1287675635	NULL	0	0	HEAP	0
1287675635	IX_ProductID	2	2	NONCLUSTERED	0

```
CREATE CLUSTERED INDEX IX_SalesOrderID_SalesOrderDetailID
ON dbo.SalesOrderDetail(SalesOrderID, SalesOrderDetailID)
```

object_id	Name	index_id	type	type_desc	is_unique
1287675635	IX_SalesOrderID_ SalesOrderDetailID	1	1	CLUSTERED	0
1287675635	IX_ProductID	2	2	NONCLUSTERED	0

Indexes

- Drop clustered index and nonclustered index

```
DROP INDEX dbo.SalesOrderDetail.IX_ProductID
```

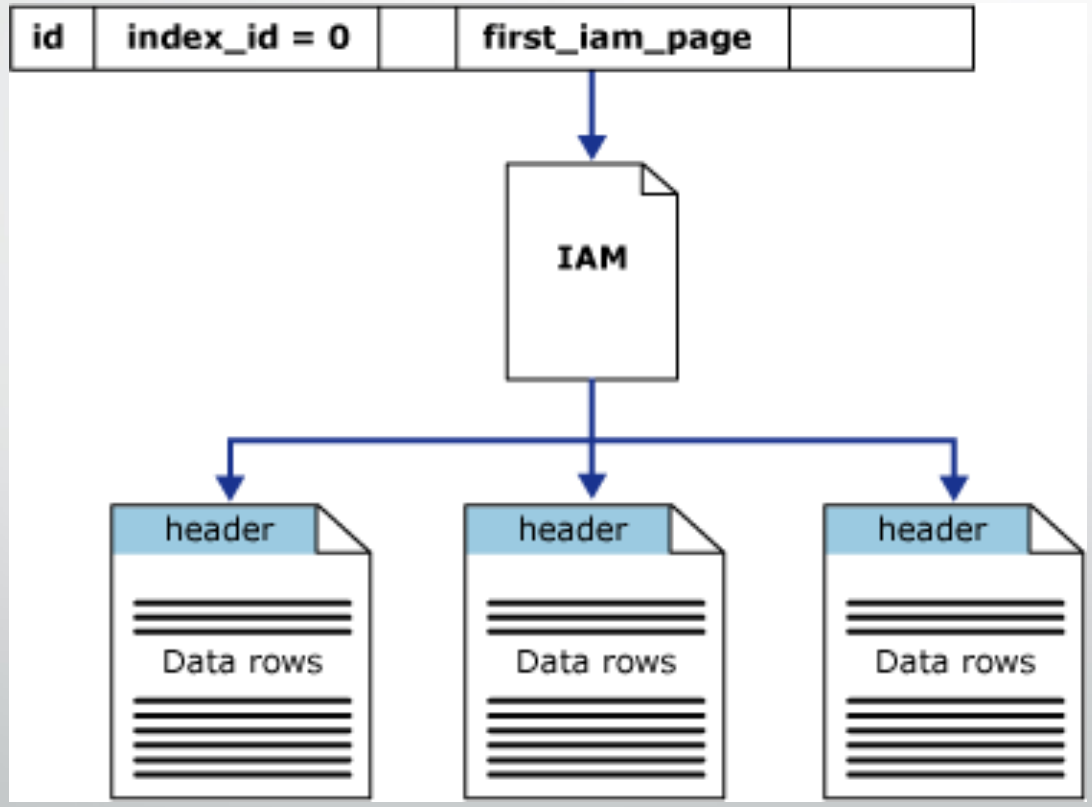
object_id	Name	index_id	type	type_desc	is_unique
1287675635	IX_SalesOrderID_ SalesOrderDetailID	1	1	CLUSTERED	0

```
DROP INDEX dbo.SalesOrderDetail.IX_SalesOrderID_SalesOrderDetailID
```

object_id	name	index_id	type	type_desc	is_unique
1287675635	NULL	0	0	HEAP	0

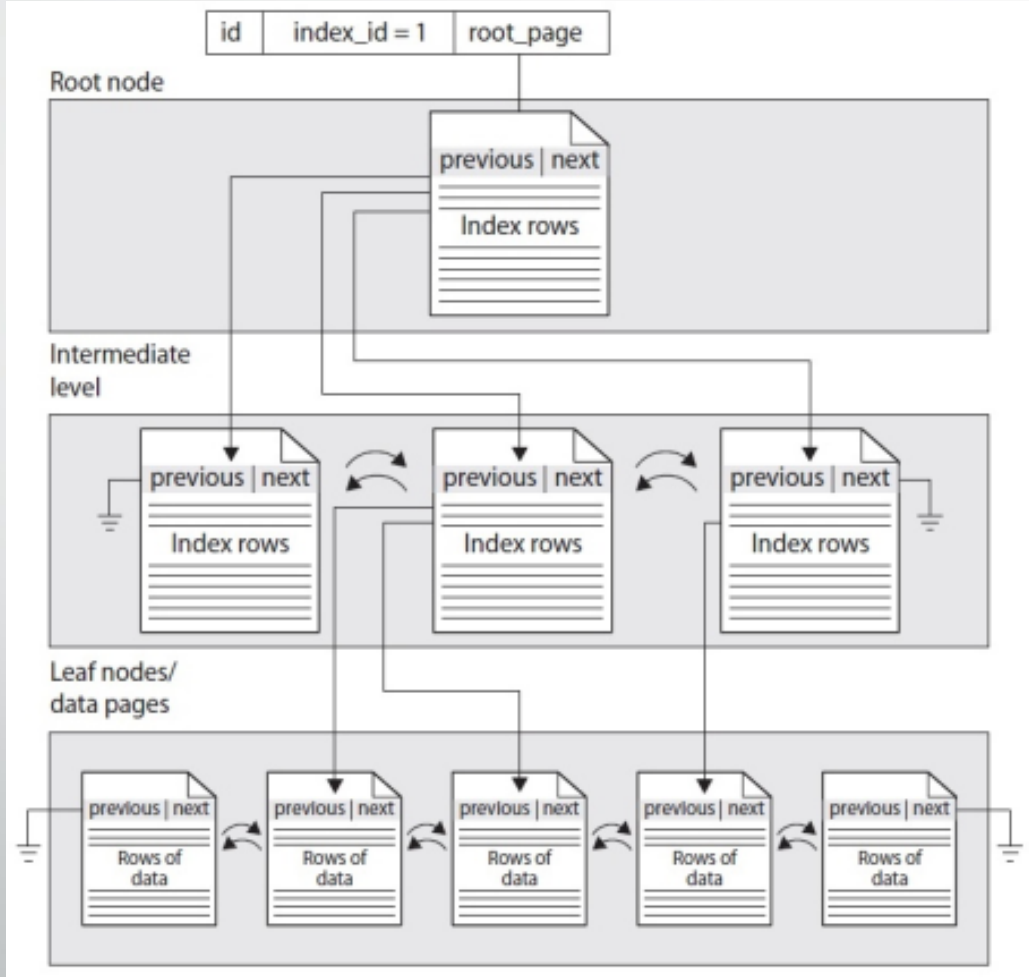
Indexes

- Structure of heap



Indexes

- Structure of clustered index



Indexes

- Clustered index vs Heap
  - Najbolje rešenje može zavisiti od definicije vaše tabele i radnog opterećenja.
  - Obično se preporučuje da za svaku tabelu bude definisan clustered index.

Indexes

- Clustered index vs Heap
  - Situacije u kojima je dobro da podaci u tabeli budu sačuvani u heap-u:
    - Kada je tabela veoma mala obe opcije su dobre.
    - Kada je RID manji od kandidata za ključ u clustered index-u. Redovi u heap-u se identifikuju pomoću RID, koji je lokator redova koji uključuje informacije kao što su datoteka baze podataka, stranica i brojevi slotova kako bi se omogućilo da se određeni zapis lako pronađe.
  - Situacije u kojima se preporučuje korišćenje clustered index-a:
    - Kada se često dohvataju podaci iz tabele u sortiranom poretku ili se dohvataju podaci iz nekog opsega.
    - Kada se često radi grupisanje podataka.

Indexes

- Clustered index vs Heap
  - U okviru artikla *SQL Server Best Practices Article* je prikazana serija testova koja prikazuje razlike u performansama kod heap-a i clustered index-a.
  - U testovima su poredjene dve opcije:
    - Clustered index sa tri kolone bez drugih nonclustered index-a
    - Heap sa nonclustered index definisan nad iste tri kolone
  - Testirano je sa 6 različitih testova.
  - Link ka artiklu:

    http://technet.microsoft.com/en-us/library/cc917672.aspx

Indexes

- Clustered index vs Heap
  - Rezultati testova:
    - INSERT operacija je 3% efikasnija kod clustered index-a.
    - UPDATE operacija je 8.2% efikasnija kod clustered index-a.
    - DELETE operacija je 18.25% brža kod clustered index-a.
    - SELECT operacija koja dohvata jedan red je 13.8% brža kod clustered index-a.
    - SELECT operacija koja dohvata opseg redova je 29.1% brža kod clustered index-a.
    - Na disk utilization test-u INSERT operacija daje slične rezultat, dok DELETE operacija clustered index daje drastično bolje rezultate.
    - Kod konkurentnih INSERT operacija, test je pokazao da kako se povećava broj procesa koji istovremeno umetanje podatke, tako se povećava i količina vremena koje je potrebno za umeranje. Ovo povećanje je značajnije kod clustered index-a.
    - Prostor na disku koji clustered index je 35 posto manji.

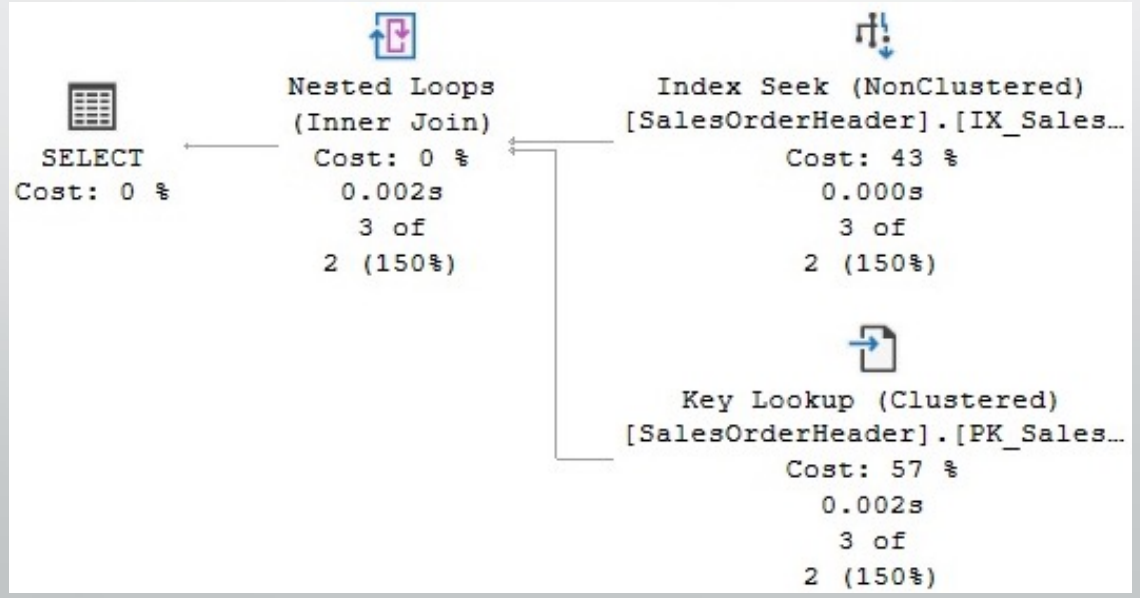
Indexes

- Clustered index key
  - Unique
    - Ako clustered index nije UNIQUE, SQL Server će dodati 4-bajtni identifikator svakom zapisu, povećavajući veličinu clustered index key.
  - Narrow
    - Manja veličina ključa će zauzimati manje prostora.
  - Static ili nonvolative
    - Menjanje vredonsti cluster index key će izazvati promene u non clustered index-ima.
  - Even-increasing
    - Umetanje novih redova na osnovu nasumičnih ulaznih podataka stvara podele stranice i stogu fragmentaciju.

Indexes

- Covering indexes
  - Ukoliko nemamo sve potrebne kolone u non clustered index-u, moramo da pristupamo i clustered index-u.

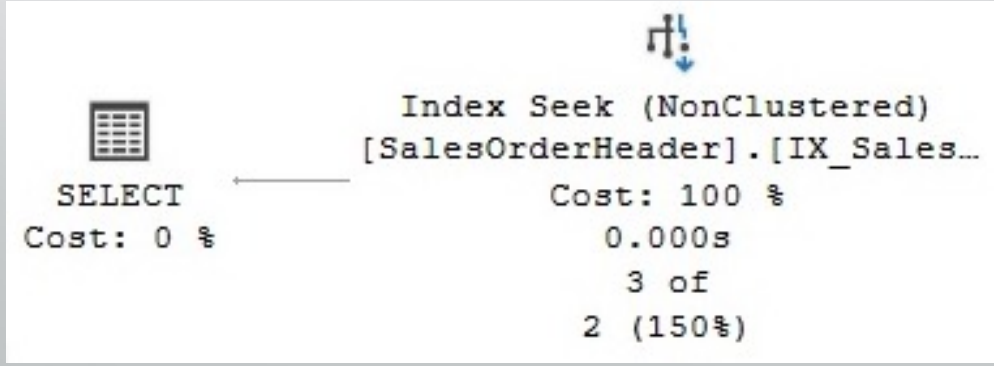
```
SELECT SalesOrderID, CustomerID, SalesPersonID
FROM Sales.SalesOrderHeader
WHERE CustomerID = 16448
```



Indexes

- Covering indexes

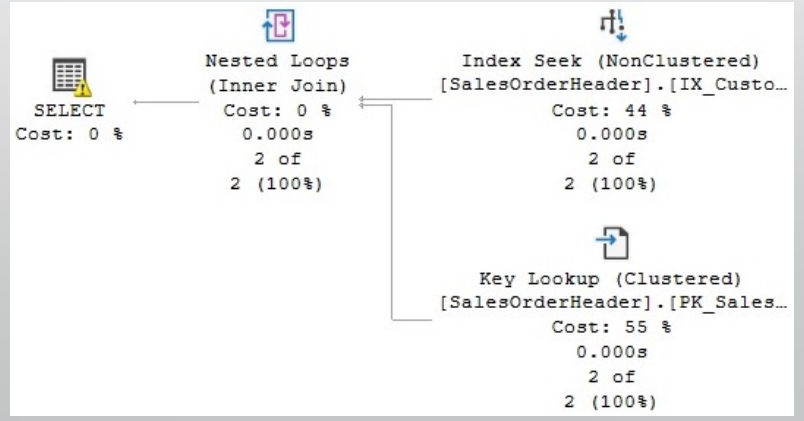
```
CREATE INDEX IX_SalesOrderHeader_CustomerID_SalesPersonID
ON Sales.SalesOrderHeader(CustomerID)
INCLUDE (SalesPersonID)

SELECT SalesOrderID, CustomerID, SalesPersonID
FROM Sales.SalesOrderHeader
WHERE CustomerID = 16448

DROP INDEX Sales.SalesOrderHeader.
    IX_SalesOrderHeader_CustomerID_SalesPersonID
```



Indexes

- Filtered indexes
  - Na kolonom koja ima veliki broj NULL vrednosti ili za često dohvatanje koknretne vrednosti

```
CREATE INDEX IX_CustomerID ON Sales.SalesOrderHeader(CustomerID)
WHERE TerritoryID = 4

SELECT CustomerID, OrderDate, AccountNumber
FROM Sales.SalesOrderHeader
WHERE CustomerID = 13917 AND TerritoryID = 4
```



Indexes

- Filtered indexes
  - Kada vrednost nije poznata (variabla ili parametar) ne koristi se filtered index.

```
DECLARE @territory int
SET @territory = 4
SELECT CustomerID, OrderDate, AccountNumber
FROM Sales.SalesOrderHeader
WHERE CustomerID = 13917 AND TerritoryID = @territory

DROP INDEX Sales.SalesOrderHeader.IX_CustomerID
```

Indexes

- Index seek je moguće primeniti na sledećim primerima:
  - `ProductID = 771`
  - `UnitPrice < 3.975`
  - `LastName = 'Allen'`
  - `LastName LIKE 'Brown%'`
- Index seek nije moguće primeniti na sledećim primerima:
  - `ABS(ProductID) = 771`
  - `UnitPrice + 1 < 3.975`
  - `LastName LIKE '%Allen'`
  - `UPPER(LastName) = 'Allen'`

Indexes

- U slučaju kad imamo index nad više kolona, moguće je raditi seek nad obe kolone samo ako nad prvom kolonom imamo uslov za jednakost. Na primer:
  - `ProductID = 771 AND SalesOrderID > 34000`
  - `LastName = 'Smith' AND FirstName = 'Ian'`
- U slučaju kad imamo index nad više kolona, nije moguće je raditi seek ako nad prvom kolonom imamo uslov koji nije jednakost. Na primer:
  - `ProductID < 771 AND SalesOrderID = 34000`
  - `LastName > 'Smith' AND FirstName = 'Ian'`
  - `ProductID = 771 AND ABS(SalesOrderID) = 34000`
- U situacijama kada nemamo mogućnost da pretražujemo po prvoj koloni, uopšte se ne koristi index nad više kolona. Na primer:
  - `ABS(ProductID) = 771 AND SalesOrderID = 34000`
  - `LastName LIKE '%Smith' AND FirstName = 'Ian'`

The Database Engine Tuning Advisor

- DTA je alat koji omogućava pronalaženje index-a koji fale i optimizuje upite.
- DTA session se pokreće opcijom *Database Engine Tuning Advisor* iz *Tools* menija.
- DTA tuning session ne kreira prave index-e već hypothetical indexes. Ovi index-i se mogu kreirati i korišćenjem opcije `WITH STATISTICS_ONLY`.
- Odluke o korišćenju indeksa zavise on nekih metadata podataka i statističih podataka nad kolonama odgovarajućih index-a.

The Database Engine Tuning Advisor

- Primer DTA analize nad fajlom
  - Potrebni podaci:

```
SELECT * INTO dbo.SalesOrderDetail
FROM Sales.SalesOrderDetail
```

  - U fajlu je potrebno sačuvati upit:

```
SELECT * FROM dbo.SalesOrderDetail
WHERE ProductID = 897
```

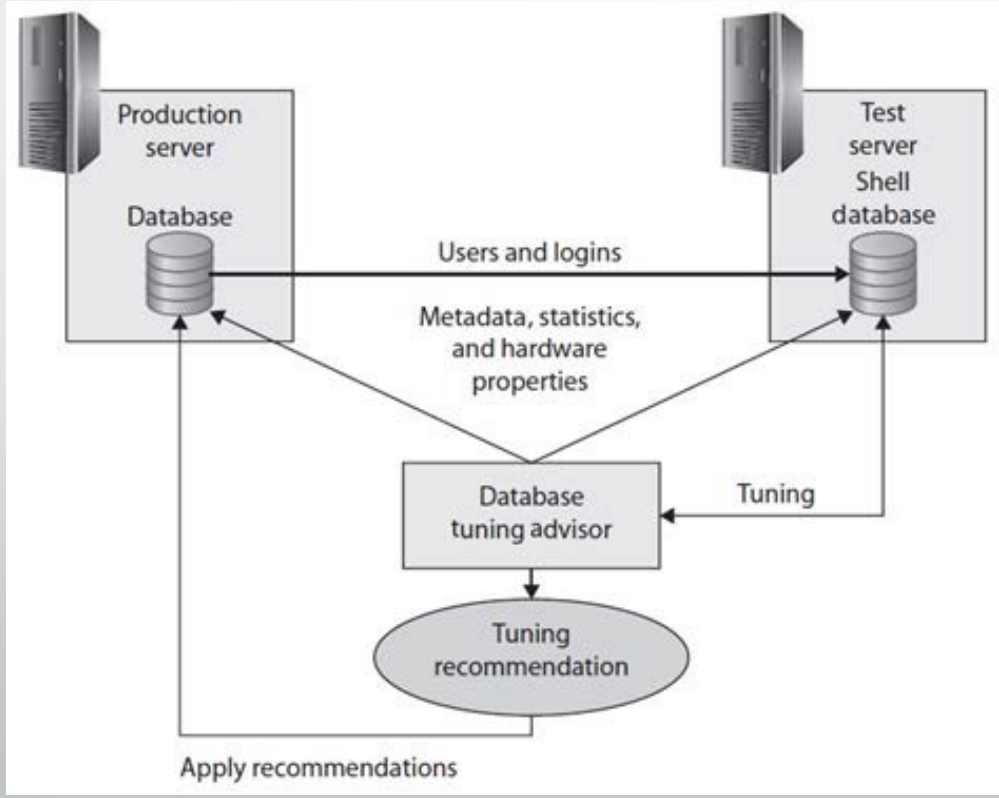
  - Nakon pokretanja DTA analize, rezultati se mogu videti:

```
SELECT * FROM msdb..DTA_reports_query
```

StatementString	CurrentCost	RecommendedCost
SELECT * FROM dbo.SalesOrderDetail WHERE ProductID = 897	1.2471	0.00333395

The Database Engine Tuning Advisor

- DTA može da koristi metadata podatke i statistike sa produkcionog servera, da kreira novu sličnu bazu bez podataka na drugom serveru (*shell database*) i da nam njom izvršava analize.



Missing indexes

- Omogućava pronalaženje index-a koji fale
- Nije toliko efikasno kao DTA
- Informacije o indeksima koji fale se mogu pronaći u `sys.dm_db_missing_index DMV.`
- Prilikom izvršavanja upita, ukoliko fali neki index, u Execution planu se može videti predlog za index koji fali za konkretni upit.

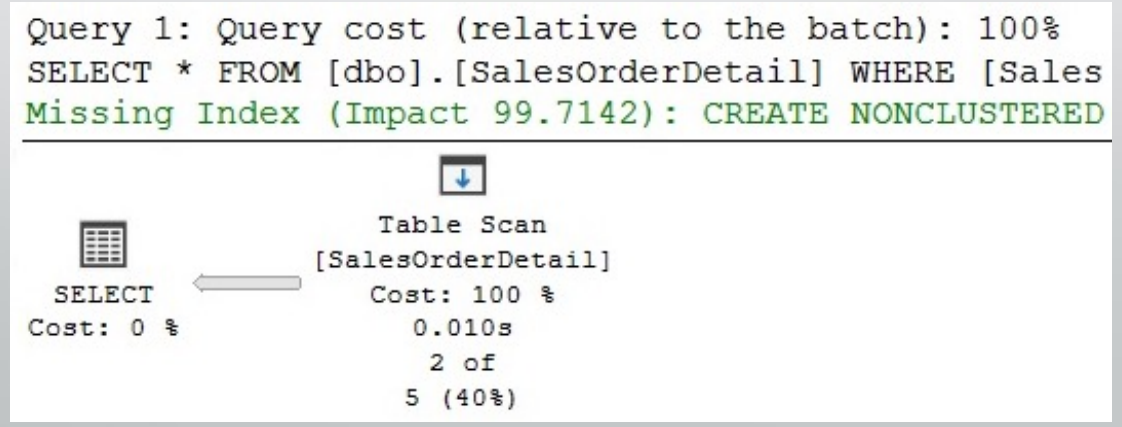
Missing indexes

- Missing index neće pokušati da pronadje index koji fali ako se izvršava trivijalni plan

```
SELECT * INTO dbo.SalesOrderDetail
FROM Sales.SalesOrderDetail

SELECT * FROM dbo.SalesOrderDetail
WHERE SalesOrderID = 43670 AND SalesOrderDetailID > 112
```

- Dodavanje neadekvatnog indeksa, pronaći će i indeks koji fali.

```
CREATE INDEX IX_ProductID ON dbo.SalesOrderDetail(ProductID)
```



Index fragmentation

- Veliki fragmentation level može da uspori izvršavanje scan operacija.
- Informacije o fragmentation level-u se mogu dobiti pomoću `sys.dm_db_index_physical_stats` DMV.
- Nad index-ima se mogu pozivati operacije `REBUILD` i `REORGANIZE`.

Index fragmentation

```
SELECT a.index_id, name, avg_fragmentation_in_percent,
 fragment_count, avg_fragment_size_in_pages
FROM sys.dm_db_index_physical_stats (DB_ID('AdventureWorks2019'),
 OBJECT_ID('Sales.SalesOrderDetail'), NULL, NULL, NULL) AS a
JOIN sys.indexes AS b
 ON a.object_id = b.object_id AND a.index_id = b.index_id
```

index_id	name	avg_fragmentation_in_percent
1	PK_SalesOrderDetail_SalesOrderID_ SalesOrderDetailID	36.13581245
2	AK_SalesOrderDetail_rowguid	2.643171806
3	IX_SalesOrderDetail_ProductID	25.83892617

Index fragmentation

```
ALTER INDEX ALL ON Sales.SalesOrderDetail REBUILD

SELECT a.index_id, name, avg_fragmentation_in_percent,
 fragment_count, avg_fragment_size_in_pages
FROM sys.dm_db_index_physical_stats (DB_ID('AdventureWorks2019'),
 OBJECT_ID('Sales.SalesOrderDetail'), NULL, NULL, NULL) AS a
JOIN sys.indexes AS b
 ON a.object_id = b.object_id AND a.index_id = b.index_id
```

index_id	Name	avg_fragmentation_in_percent
1	PK_SalesOrderDetail_SalesOrderID_SalesOrderDetailID	0.24291498
2	AK_SalesOrderDetail_rowguid	0
3	IX_SalesOrderDetail_ProductID	0

Unused indexes

- Index koji se ne koristi samo opterećuje bazu i zauzima prostor, a višak je.
- Index koji se ranije korstio, a više se ne koristi bi trebalo obrisati.
- Informacije o korišćenju index-a se mogu dobiti pomoću `sys.dm_db_index_usage_stats` DMV.

Unused indexes

```
SELECT * INTO dbo.SalesOrderDetail
FROM Sales.SalesOrderDetail

CREATE NONCLUSTERED INDEX IX_ProductID ON
 dbo.SalesOrderDetail(ProductID)

SELECT DB_NAME(database_id) AS database_name,
 OBJECT_NAME(s.object_id) AS object_name, i.name, s.*
FROM sys.dm_db_index_usage_stats s JOIN sys.indexes i
 ON s.object_id = i.object_id AND s.index_id = i.index_id
 AND OBJECT_ID('dbo.SalesOrderDetail') = s.object_id
…
```

name	index_id	user_seeks	user_scans	user_lookups	user_updates
NULL	0	0	3	4	1
IX_ProductID	2	7	0	0	1

Analiza statistike

- Prikupljanje statistike
- Prikaz statistike
- Density vector
- Histograms
- Cardinality estimator
- Cardinality estimation errors
- Incremental statistics
- Statistics on computed columns
- Filtered statistics
- Statistics on ascending keys
- Cost estimation

Prikupljanje statistike

- SQL Server kreira i održava statistiku da bi omogućio optimizatoru upita da izračuna procenu kardinalnosti.
- Procena kardinalnosti (cardinality estimation) je procenjeni broj redova koji će biti vraćeni upitom ili specifičnom operacijom upita, kao što je spajanje ili filter.
- Selectivity je koncept sličan proceni kardinalnosti, koji se može opisati kao deo redova u skupu koji zadovoljava predikat. Vrednost je uvek između 0 i 1. Visok selectivity vraća mali broj redova.

Prikupljanje statistike

- Statistika će se automatski kreirati i ažurirati ako je opcija `AUTO_CREATE_STATISTICS` aktivirana.
- Manuelno se mogu kreirati i ažurirati instrukcije pomoću operacija `CREATE STATISTICS` i `UPDATE STATISTICS`.
- Podrazumevano će statistika biti radjena nad nekim manjim delom tabele. Korišćenjem opcije `WITH FULLSCAN` statistika će biti odradjena nad celom tabelom.

Prikaz statistike

- `sys.stats catalog view`

```
SELECT * FROM sys.stats
WHERE object_id = OBJECT_ID('Sales.SalesOrderDetail')
```

object_id	name	stats_id
1154103152	PK_SalesOrderDetail_SalesOrderID_SalesOrderDetailID	1
1154103152	AK_SalesOrderDetail_rowguid	2
1154103152	IX_SalesOrderDetail_ProductID	3

Prikaz statistike

- `SHOW_STATISTICS`
  - Prikazuje:
    - Osnovne informacije
    - The density vector
    - Histograms

Prikaz statistike

- SHOW_STATISTICS

```
SELECT * FROM Sales.SalesOrderDetail
WHERE UnitPrice = 35
DBCC SHOW_STATISTICS ('Sales.SalesOrderDetail', UnitPrice)
```

```
Name                        Updated Rows          Rows      Rows Sampled Steps
--------------------------- --------------------- --------- ------------ -----
_WA_Sys_00000007_44CA3770 Jun 3 2022 11:55PM    121317    110388       200
All density   Average Length Columns
------------- -------------- ------------------
0.003205128   8              UnitPrice
RANGE_HI_KEY      RANGE_ROWS     EQ_ROWS        DISTINCT_RANGE_ROWS  AVG_RANGE_ROWS
----------------- -------------- -------------- -------------------- --------------
1.3282            0              1              0                    1
1.374             35.19722       142.3062       0                    370.2226
2.29              35.19722       2747.751       0                    370.2226
2.994             417.5509       341.9168       3                    123.5255
3.975             35.19722       1              0                    370.2226
3.99              35.19722       2061.052       0                    370.2226
```

Prikaz statistike

- sys.dm_db_stats_properties i sys.dm_db_stats_histogram

```
SELECT * FROM sys.dm_db_stats_properties(
 OBJECT_ID('Sales.SalesOrderDetail'), 4)
SELECT * FROM sys.dm_db_stats_histogram(
 OBJECT_ID('Sales.SalesOrderDetail'), 4)
```

Density vector

- All density ima vrednost 1/broj_različitih kombinacija

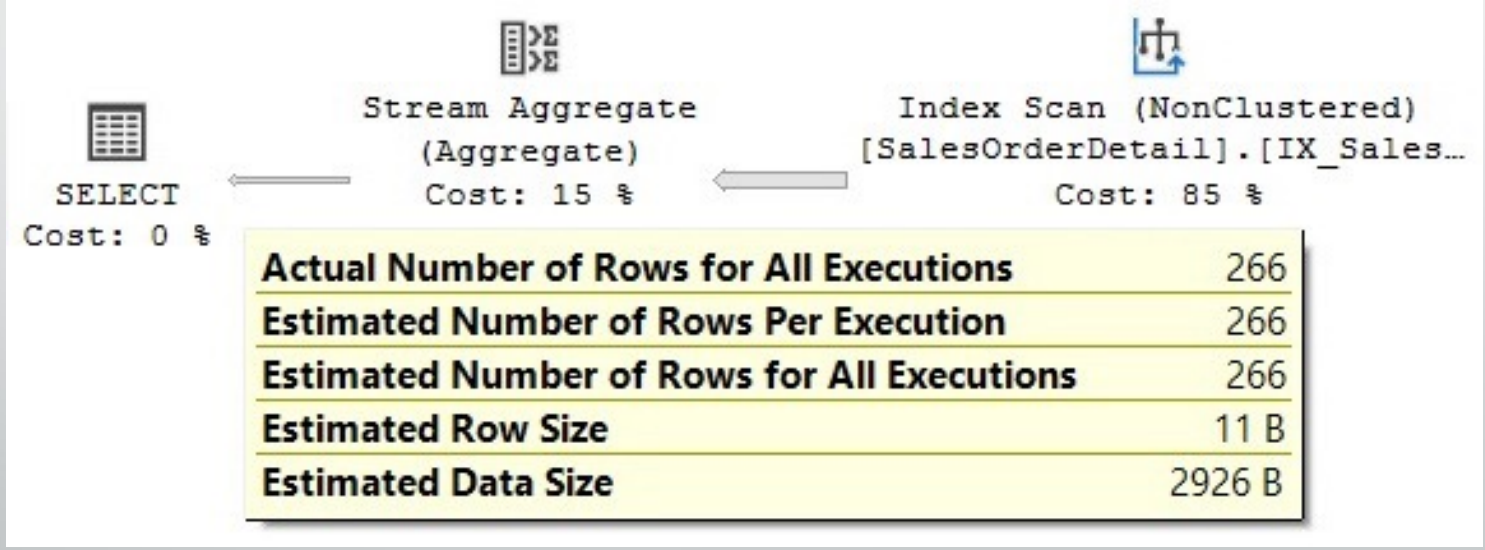
```
DBCC SHOW_STATISTICS ('Sales.SalesOrderDetail',
    IX_SalesOrderDetail_ProductID)
```

All density	Average Length	Columns
0.003759399	4	ProductID
8.242868E-06	8	ProductID, SalesOrderID
8.242868E-06	12	ProductID, SalesOrderID, SalesOrderDetailID

Density vector

- Koriste se za procenu broja grupa kod GROUP BY operacije:

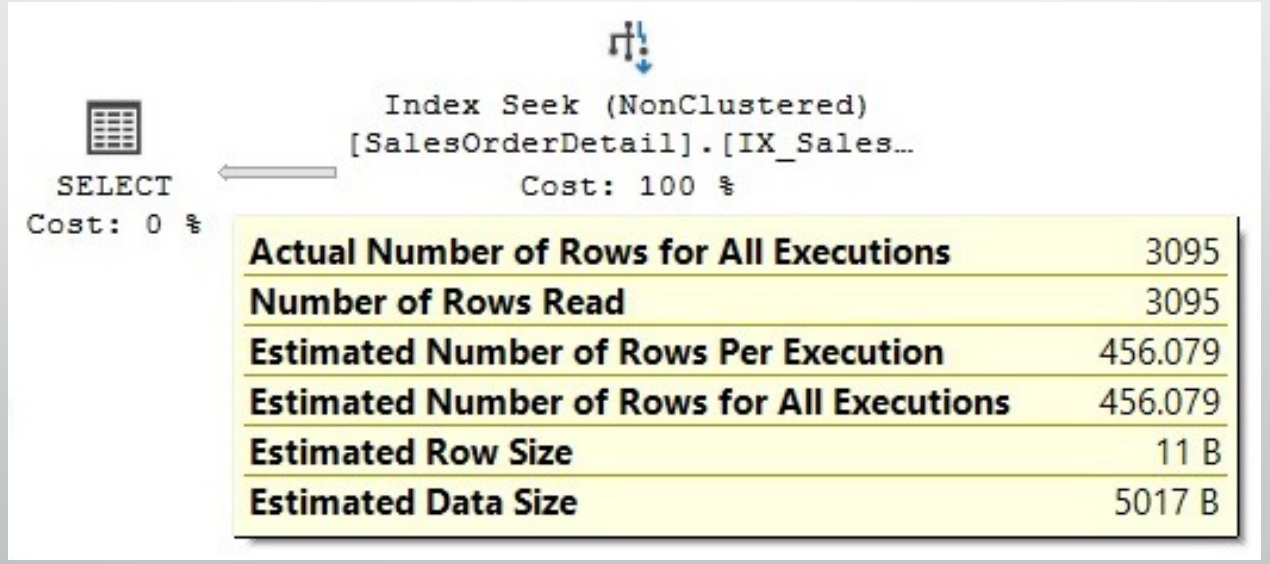
```
SELECT ProductID
FROM Sales.SalesOrderDetail
GROUP BY ProductID
```



Density vector

- Koriste se za procenu kardinalnosti upita sa jednakošću koji sadrži promenljivu.

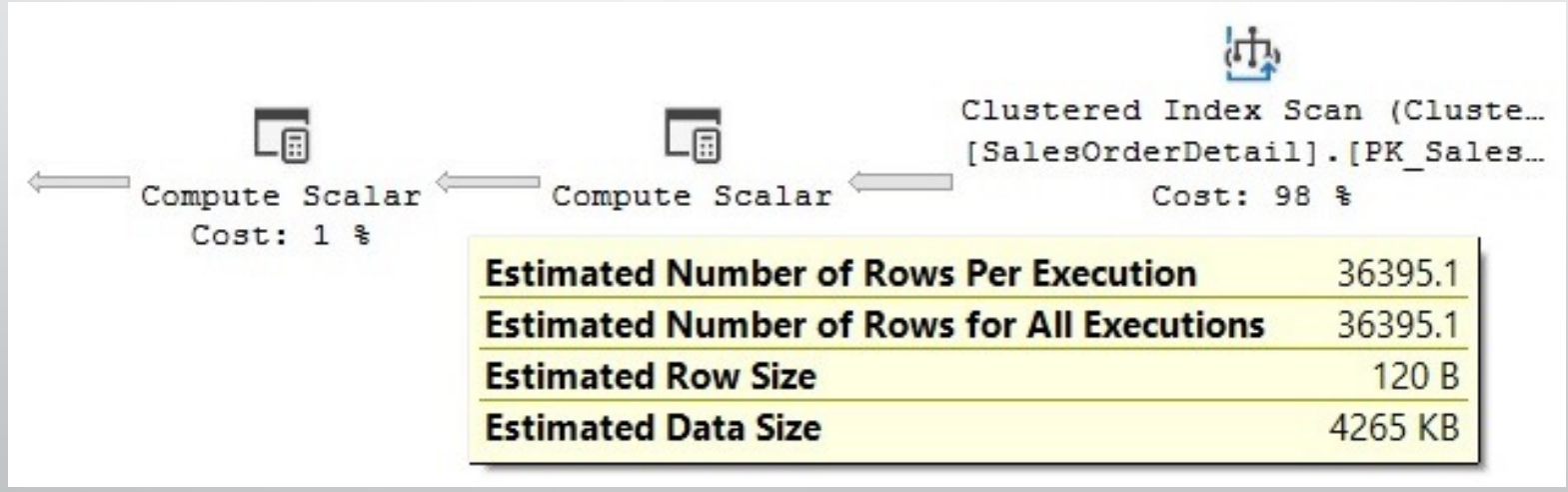
```
DECLARE @ProductID int
SET @ProductID = 921
SELECT ProductID FROM Sales.SalesOrderDetail
WHERE ProductID = @ProductID
```



Density vector

- Ne koriste se za procenu kardinalnosti upita koji nema jednakost i koji sadrži promenljivu. Ovde se uzima standardna procena od 30% od ukupnog broja redova.

```
DECLARE @pid int = 897
SELECT * FROM Sales.SalesOrderDetail
WHERE ProductID < @pid
```



Histograms

- Maksimalan broj koraka u histogramu je 200, ali čak i ako tabela sadrži 200 ili više jedinstvenih vrednosti, histogram može i dalje imati manje od 200 koraka.
- SQL Server pronalazi jedinstvene vrednosti u koloni i bira one koji se najčešće pojavljuju koristeći varijaciju *maxdiff* algoritma, tako da statistički najznačajniji informacije budu sačuvane.
- *Maxdiff* je jedan od dostupnih histograma čija je svrha da preciznopredstavljaju distribuciju vrednosti podataka u relacionim bazama podataka.

Histograms

- RANGE_HI_KEY - gornja granica koraka.
- EQ_ROWS – procena broj redova koji imaju vrednost kolone RANGE_HI_KEY
- RANGE_ROW – procena broja redova koji imaju vrednost kolone izmedju predhodnog i ovog RANGE_HI_KEY
- DISTINCT_RANGE_ROW – procena broja različitih vrednosti kolone izmedju predhodnog i ovog RANGE_HI_KEY
- AVG_RANGE_ROW = RANGE_ROW /DISTINCT_RANGE_ROW

Histograms

```
DBCC SHOW_STATISTICS ('Sales.SalesOrderDetail',
    IX_SalesOrderDetail_ProductID)
```

RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
826	0	305	0	1
831	110	198	3	36.66667
832	0	256	0	1

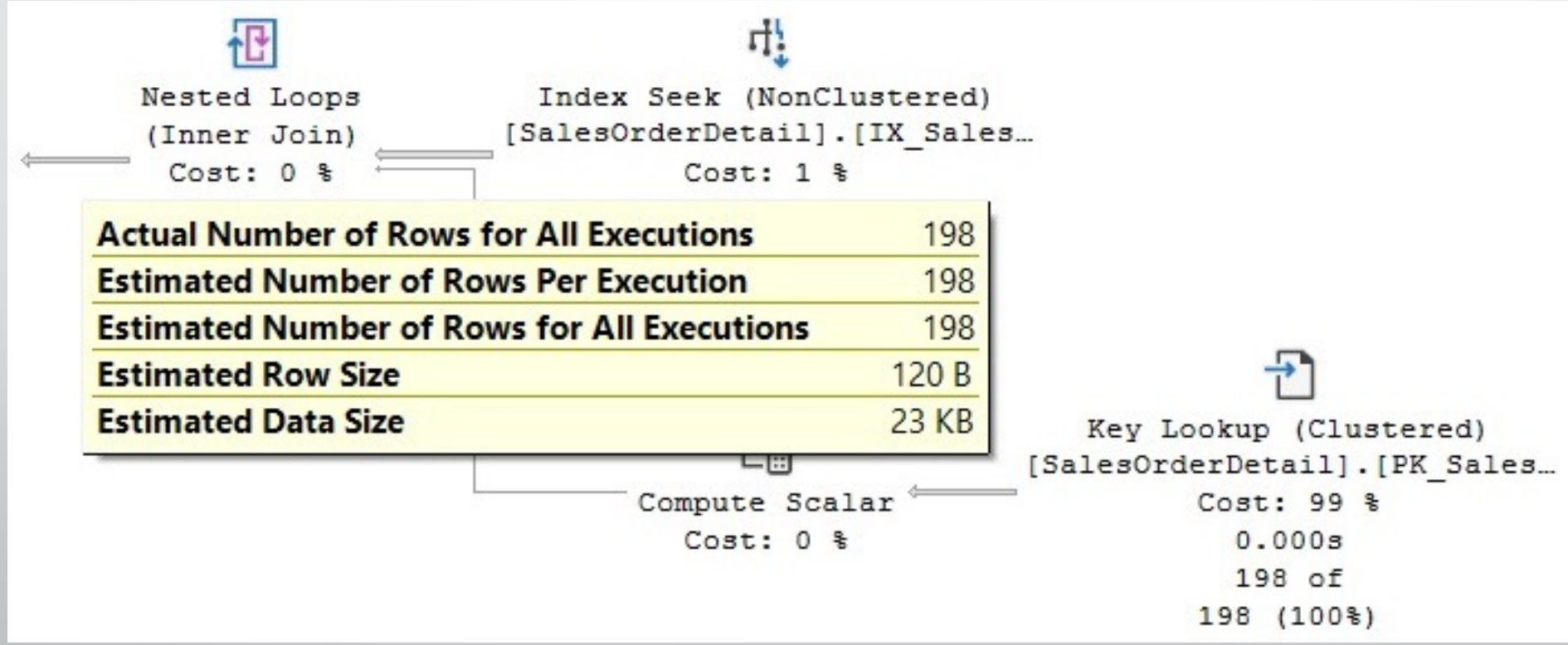
Histograms

```
SELECT ProductID, COUNT(*) AS Total
FROM Sales.SalesOrderDetail
WHERE ProductID BETWEEN 827 AND 831
GROUP BY ProductID
```

ProductID	Total
827	31
828	46
830	33
831	198

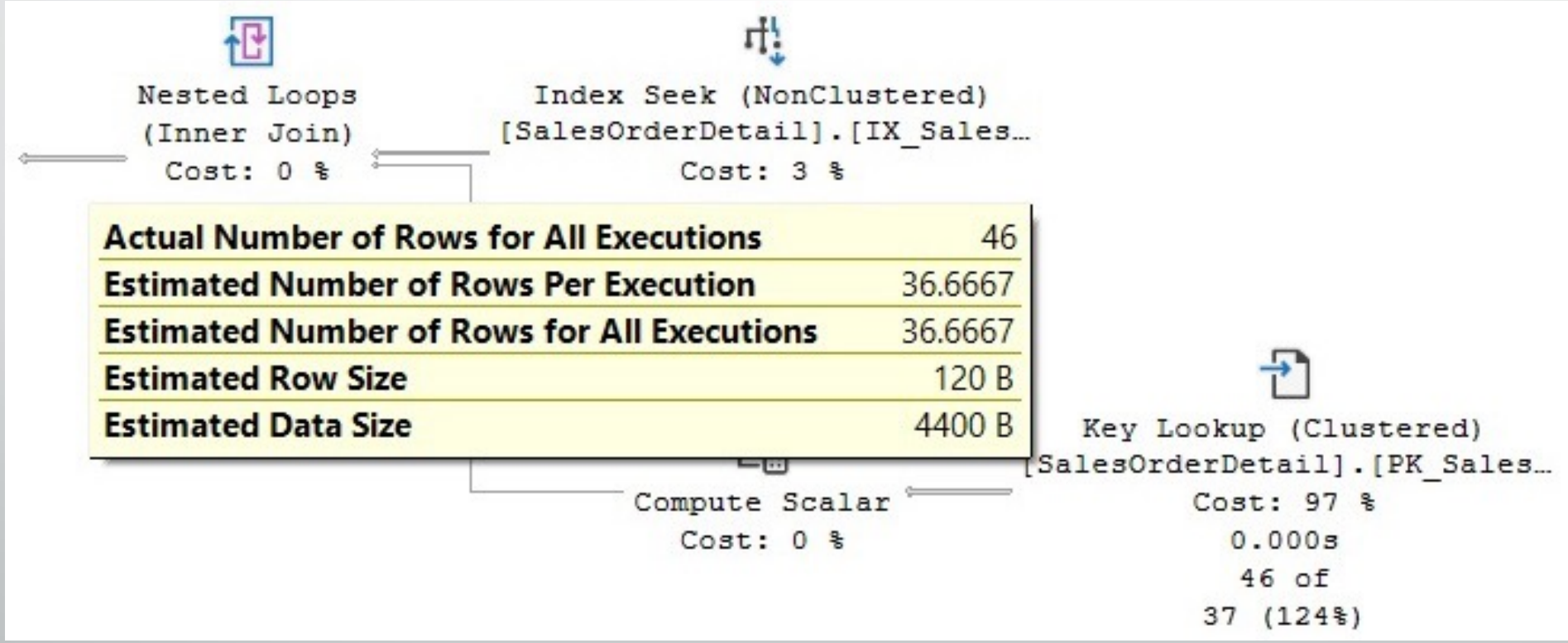
Histograms

```
SELECT * FROM Sales.SalesOrderDetail
WHERE ProductID = 831
```



Histograms

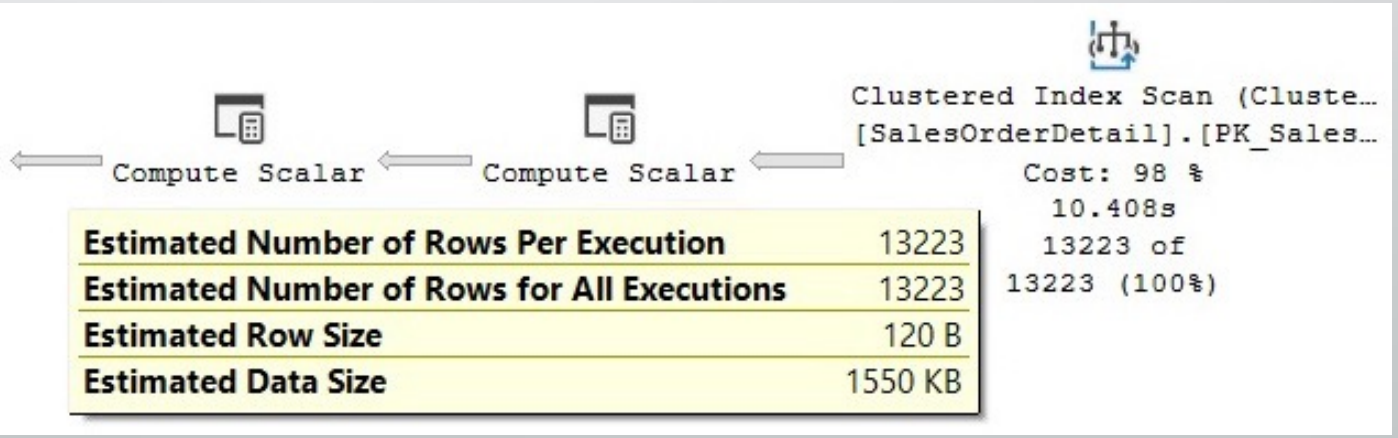
```
SELECT * FROM Sales.SalesOrderDetail
WHERE ProductID = 829
```



Histograms

```
SELECT * FROM Sales.SalesOrderDetail
WHERE ProductID < 714
```

RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
707	0	3083	0	1
708	0	3007	0	1
709	0	188	0	1
710	0	44	0	1
711	0	3090	0	1
712	0	3382	0	1
713	0	429	0	1
714	0	1218	0	1
715	0	1635	0	1



Cardinality estimator

- Cardinality estimator je komponenta čiji je zadatak da prilikom procesiranja upita proceni broj redova koje vraćaju relacione operacije u upitu.
- Na osnovu rezultata Cardinality estimator-a, zajedno sa nekim ostalim podacima, optimizator upita bira koji Execution plan će se izvršiti.
- Ovo je matematički model koji se oslanja na statističke informacije, pa ne daje uvek tačne podatke.
- Oslanja se na sledeće pretpostavke:
  - Uniformity
  - Indepemdence
  - Containment
  - Inclusion

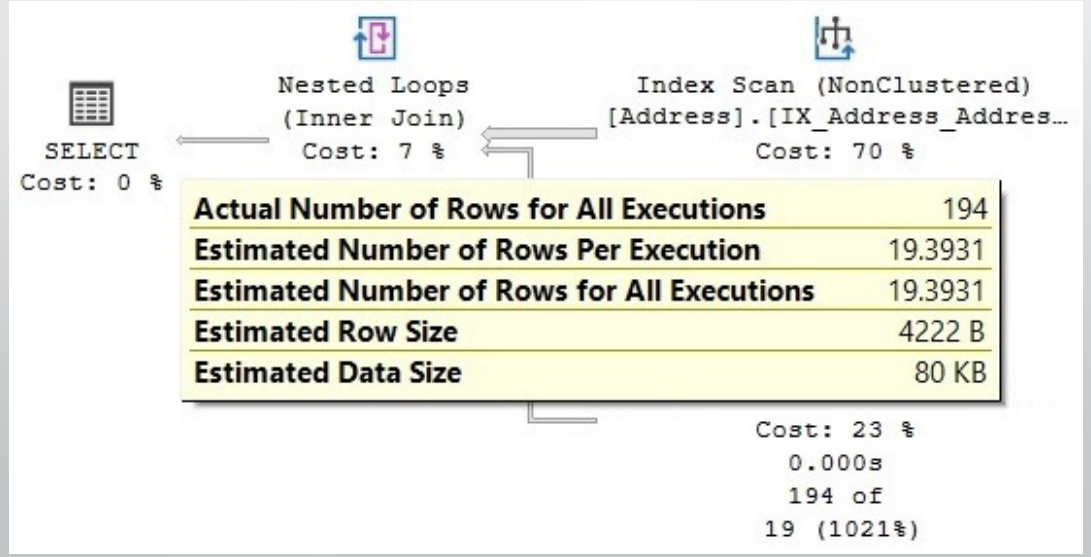
Cardinality estimator

- Korišćena formula kad upit sadrži AND operator:

*selectivity of most selective filter * SQRT(selectivity of next most selective filter)*

(194/19614) * SQRT(196/19614) * 19614

```
SELECT *
FROM Person.Address
WHERE City = 'Burbank' AND PostalCode = '91502'
```

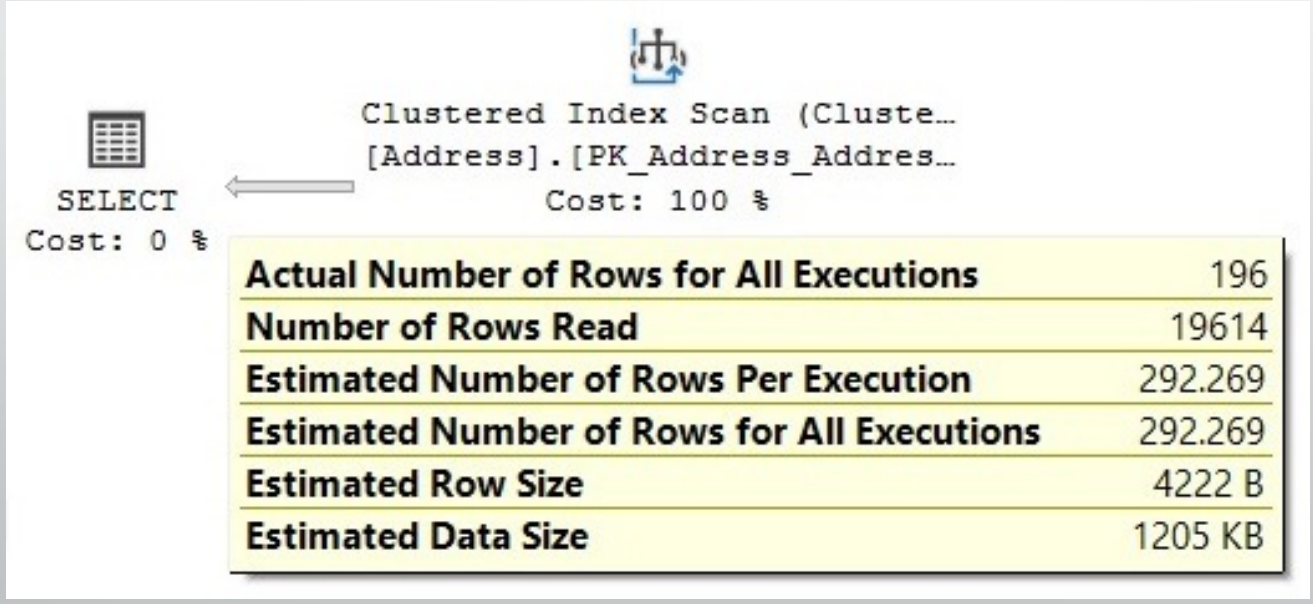


Cardinality estimator

- Korišćena formula kad upit sadrži OR operator:

(1-(1-(196/19614)) * SQRT(1-(194/19614))) * 19614

```
SELECT *
FROM Person.Address
WHERE City = 'Burbank' OR PostalCode = '91502'
```



Cardinality estimation errors

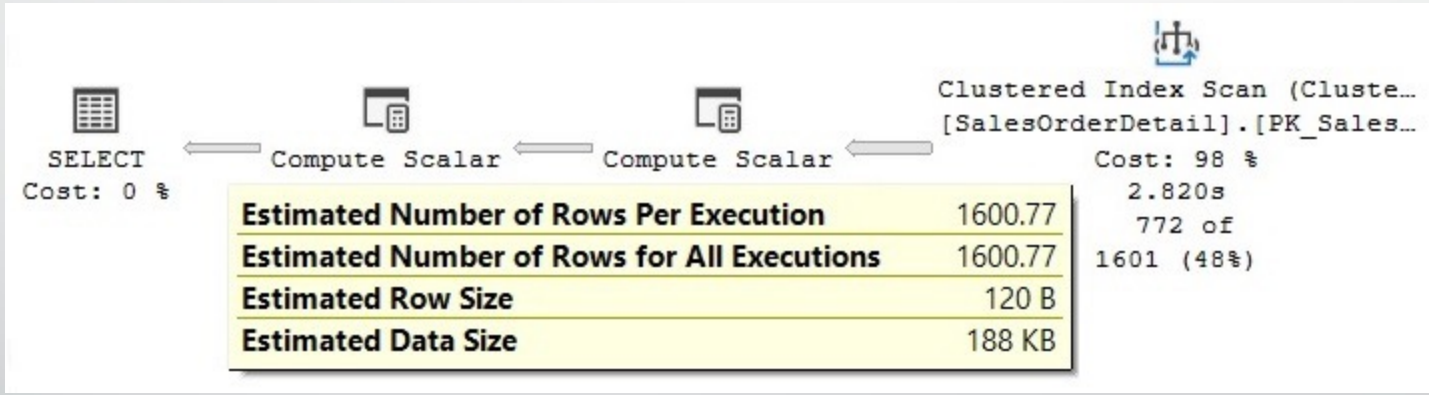
```
SET STATISTICS PROFILE ON
GO
SELECT * FROM Sales.SalesOrderDetail
WHERE OrderQty * UnitPrice > 10000
GO
SET STATISTICS PROFILE OFF
GO
```

Rows	EstimateRows	StmtText
772	36395.1	SELECT * FROM [Sales].[SalesOrderDetail] WHERE [OrderQty]
772	36395.1	|--Filter(WHERE:([Expr1003]>($10000.0000)))
0	121317	|--Compute Scalar(DEFINE:([AdventureWorks2019].[Sales]
0	121317	|--Compute Scalar(DEFINE:([AdventureWorks2019].[S
121317	121317	|--Clustered Index Scan(OBJECT:([AdventureWo

Statistics on computed columns

- Kopletna statistika postoji i nad izvedenim kolonama

```
ALTER TABLE Sales.SalesOrderDetail ADD cc AS OrderQty * UnitPrice
SELECT * FROM Sales.SalesOrderDetail
WHERE OrderQty * UnitPrice > 10000
ALTER TABLE Sales.SalesOrderDetail DROP COLUMN cc
```



- Za `UnitPrice * OrderQty` nećemo dobiti dobru procenu broja redova. Biće standardna procena od 30%.

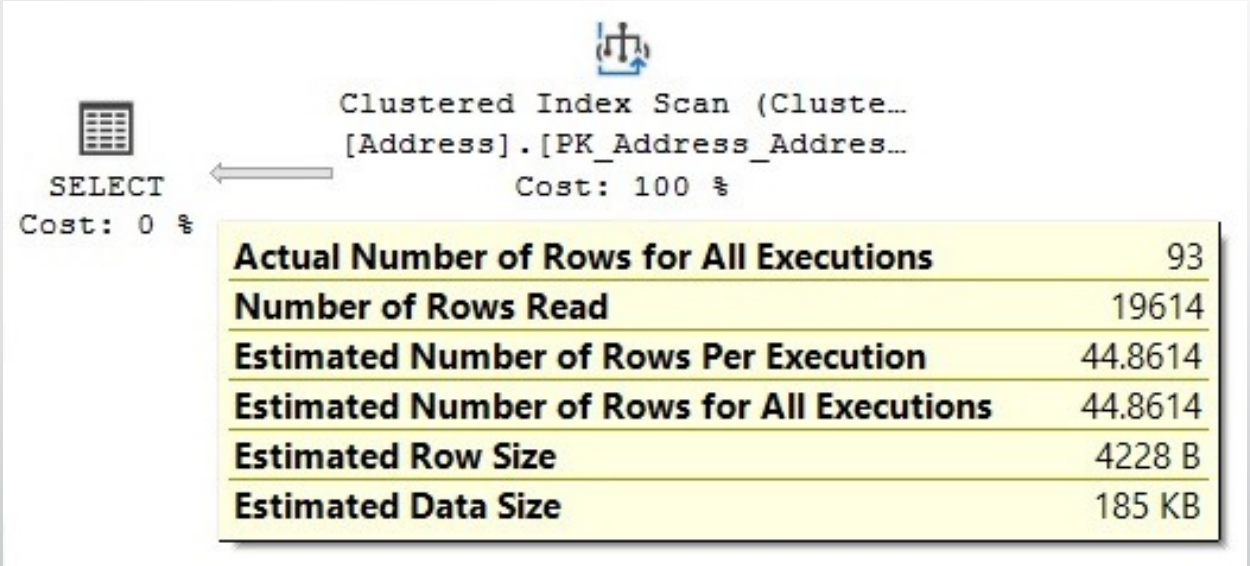
Filtered statistics

- Kreira se automatski kada se kreira filtered index.
- Može se kreirati i manuelno definisanjem WHERE klauzule u okviru CREATE STATISTICS statementa.

```
CREATE STATISTICS california
ON Person.Address(City)
WHERE StateProvinceID = 9
```

Filtered statistics

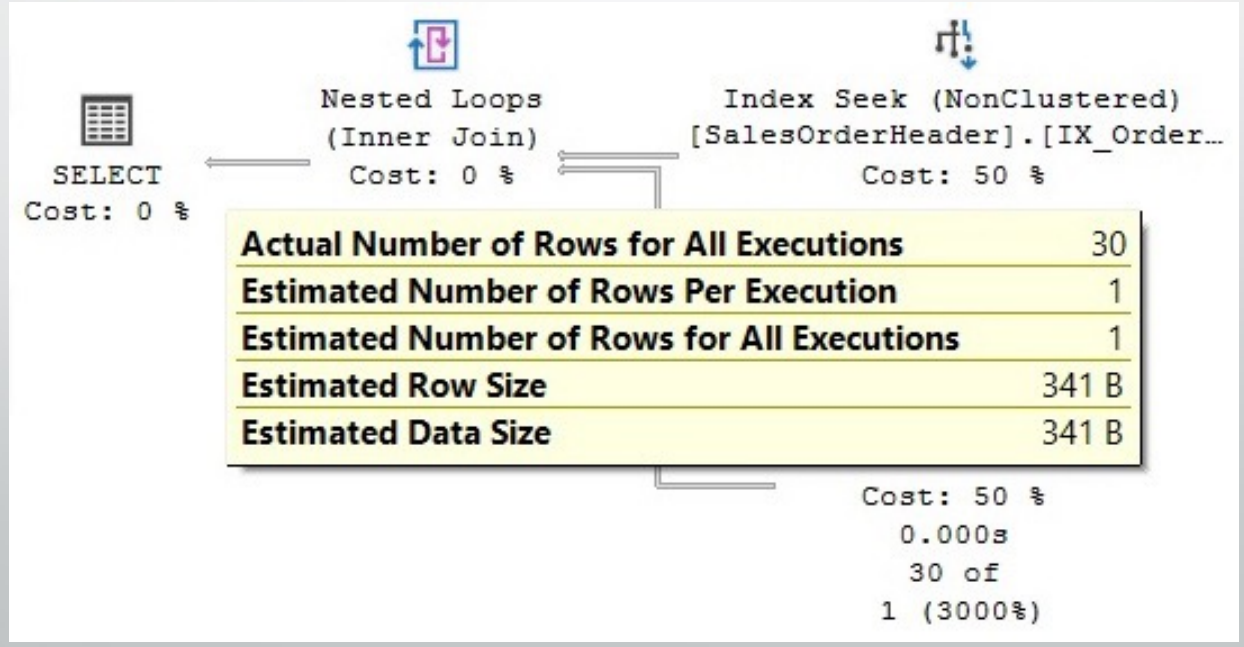
```
DBCC FREEPROCCACHE
GO
SELECT * FROM Person.Address
WHERE City = 'Los Angeles' AND StateProvinceID = 9
```



- Procena broja redova pre kreiranja filtrirane statistike je bila 44.861.

Statistics on ascending keys

- Može da daje pogrešne rezultate, ako je dodat neki broj podataka koji nije dovoljan da se automatski izvrši ažuriranje statistike.
- Za dohvatanje koji dohvata nove redove biće procenjeno da postoji jedan red, jer trenutno statistika ne zna da uopšte postoje podaci iz tog opsega vrednosti.



Cost estimation

- Tokom optimizacije upita, Query optimizer proučava veći broj planova i procenjuje njihovu cenu, a zatim bira najefikasniji.
- Troškovi se procenjuju za bilo koji delimični ili potpuni plan.

- Obračun troškova se vrši po operatoru, a ukupni trošak plana je zbir troškova svih operatora u tom planu. Cena svakog operatera zavisi od njegovog algoritma i procenjenog broja zapisa koje vraća.
- Neki operatori, kao što su Sort i Hash Join, takođe uzimaju u obzir dostupnu memoriju u sistemu.

Cost estimation

- Primer:

```
SELECT * FROM Sales.SalesOrderDetail
WHERE LineTotal = 35
```

  - Za clustered index Scan operator CPU cost:
    0.0001581 + 0.0000011 * (121317 – 1)
    - 0.0001581 je procena vrema koje je potrebno da se pronadje prvi red
    - 0.0000011 je procena vremea koje je potrebno da se dohvati svaki naredni red
    - Procenjeno je da će biti dohvaćeno 121317 redova