DBMS

Optimizacija SQL upita na Microsoft SQL Server-u

Studenti:

Filip Stamenković, 342

Aleksa Trajković, 370

Apstrakt

Ovaj rad se bavi analizom procesa optimizacije jednog kompleksnog SQL upita, kod Microsoft SQL Servera. Analizirani upit sadrži preko 10 spojeva, sa tabelama koje sadrže nekoliko stotina hiljada slogova.

Motivacija za optimizaciju upita je proistekla iz potrebe da se podrži do 10 miliona slogova, a već kod par stotina hiljada podataka, inicijalni SQL upit je pokazivao slabe performanse.

Upit planiramo da optimizujemo pomoću tehnika koje su dostupne kod većine relacionih DBMS-ova, kao što su: prepisivanje upita na efikasniji način, dodavanje indeksa, dodavanje pogleda (materijalizovanih i nematerijalizovanih). Pored ovih „standardnih“ tehnika, dalju analizu upita planiramo da obavimo pomoću alata koji su dostupni u *SQL Management Studio*-u. Korišćeni alati su: pregled plana (predviđeni i realni) izvršenja, *Database Tuning Advisor*, predlozi *Query Optimizer*-a.

Za svaku izvršenu optimizaciju biće dat pregled performansi, na osnovu kojih se može zaključiti da je cena spoja između dve tabele veoma visoka, tako da se, za veliki broj podataka i spojeva, upit ne može izvršavati dovoljno brzo kako bi zadovoljio korisnikova očekivanja.

Sadržaj

1 Uvod

Najkorišćeniji upiti su oni koji pribavljaju podatke preko SELECT naredbe. Povećanjem količine podataka postaje bitno kako je upit napisan. Pored poznavanja SQL sintakse, programer mora poznavati specifičnosti, prednosti i mane DBMS-a nad kojim se izvršava upit. Pored prepravljanja upita, na raspolaganju su tehnike dodavanje indeksa, pogleda, statistika itd.

SQL jezik je upitni jezik visokog nivoa, orijentisan ka korisnicima. Preko njega, korisnik zahteva podatke od baze. Da bi baza odgovorila na zahtev (pa čak i jednostavan), podacima može pristupiti na različite načine, preko različitih struktura podataka i korišćenjem različitih algoritama. Svaki od različitih pristupa zahteva različito vreme obrade, koje može da varira od dela sekunde, do nekoliko minuta, pa čak i sati. Zadatak optimizacije, kao automatizovanog procesa, je da nađe način na koji se traženi podaci mogu pribaviti za najmanje vremena.

Optimizacija upita je jedna od glavnih komponenti relacionih sistema za upravljanje bazama podataka. Zadatak optimizatora upita je da odredi najefikasniji plan izvršenja, razmatranjem više mogućih planova. Obično, optimizator ne izabere najefikasniji plan (osim u trivijalnim upitima), već zadovoljavajuće efikasan, jer nalaženje najoptimalnije strategije obično zahteva previše vremena.

Svaki relacioni DBMS ima više opštih algoritama za pristup podacima, koji implementiraju operacije relacione algebre, kao što su selekcija, projekcija, spoj i njihove kombinacije. Sam proces izvršenja upita i kreće prevođenjem SQL upita u odgovarajući upit relacione algebre, a zatim se biraju algoritmi koji implementiraju ove logičke operacije.

Jedan od prvih problema na koje DBMS nailazi je da originalni SQL upit može da se prevede u više ekvivalentnih izraza relacione algebre. Ovakav upit relacione algebre, predstavljen strukturom stabla, sa varijantama algoritama za svaki operator, se naziva plan izvršenja.

Optimizator počinje proces optimizacije dekompozicijom kompleksih upita (sadrže spojeve, ugnježdene upite...) na više manjih blokova upita, nakon čega vrši optimizaciju svakog bloka pojedinačno. Ovakvi blokovi imaju samo jednu SELECT, FROM i najviše jednu WHERE, GROUP BY i HAVING klauzulu, i prevode se u izraz relacione algebre.

Još jedan od problema sa kojima se relacioni DBMS susreće je nalaženje podskupa planova koji će biti razmatrani (prostor svih planova je za složenije upite ogroman), a zatim i procenu cene izvršenja svakog od odabranih planova.

Procena cene celokupnog plana se vrši procenom cene svakog od operatora u stablu plana, i zavisi od veličina ulaznih skupova podataka, kao i od izabranih algoritama koji implementiraju operaciju relacione algebre. Za procenu veličine skupa podataka (kardinalnost) se koriste informacije o ulaznim relacijama, kao što su broj torki, stranica, postojanje i tip indeksa. Ove informacije DBMS čuva u katalogu i ažurira ih periodično.

2 SQL Server Query Optimizer

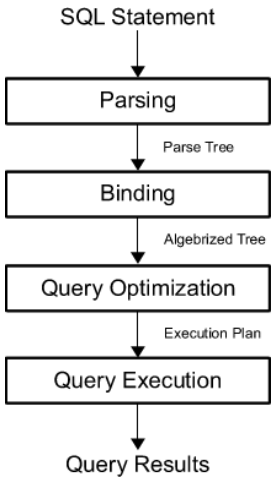
Microsoft SQL Server je relacioni DBMS, koji sadrži optimizator upita za određivanje najpovoljnijeg plana.

SQL Server Query Optimizer je optimizator na bazi predviđene cene (*cost-based*). Optimizator analizira više mogućih planova izvršenja upita (kandidati), procenjuje cenu izvršenja svakog od plana i na kraju bira plan sa najnižom cenom među razmatranim kandidatima. Optimizator ne može da razmotri svaki mogući plan izvršenja upita, jer mora da vodi računa i o ceni same analize planova, pored cene izvršenja.

2.1 Kako optimizator upita radi

U jezgru SQL Server DBMS-a su dve glavne komponente, ***Storage Engine*** i ***Query Processor***. *Storage Engine* je odgovoran za razmenu podataka između diska i memorije, tako da maksimizuje konkurentnost, dok održava integritet podataka. *Query Processor* prihvata upite koji su prosleđeni SQL Serveru na izvršenje, osmišlja plan „optimalnog“ izvršenja, izvršava ga i vraća rezultate. Upiti se SQL Serveru prosleđuju korišćenjem SQL jezika (ili T-SQL jezika, Microsoft-ovog procedularnog proširenja SQL jezika). Pošto je SQL deklarativni jezik visokog nivoa, upitom definišemo samo koje podatke želimo, ne i neophodne korake za pribavljanje tih podataka, niti algoritme kojima se ovaj zahtev obrađuje. Stoga, za svaki upit koji primi, prvi zadatak *Query Processor*-a je da formira plan, što brže, koji opisuje najbolji način za izvršenje zadatog upita (ili barem efikasan način). Drugi zadatak je izvršenje upita prema formiranom planu.

Za svaki od ovih zadataka, zadužena je posebna komponenta *Query Processor*-a. Optimizator upita (***Query Optimizer***) formira plan i prosleđuje ga ***Execution Engine***-u, koji zapravo izršava plan i vraća rezultat iz baze, kao što je prikazano na *Slici 1*.



Slika 1 Proces izvršenja upita

Parsiranje proverava da li (T-)SQL ima validnu sintaksu, i prevodi SQL upit u stablo logičkih operacija visokog nivoa, koje su bliske operacijama iz originalnog SQL upita.

Binding se bavi proverom imena. Tokom ove operacije, SQL Server proverava da li svi objekti postoje, i vezuje svako ime tabele i kolone, iz stabla dobijenog parsiranjem, za odgovarajući objekat iz sistemskog kataloga.

Sledeći korak je generisanje određenog broja planova izvršenja, kao kandidata i odabir „najboljeg" procenom cene izvršenja. Takođe, u ovom koraku se logičke operacije iz stabla mapiraju u fizičke operacije, koje *Execution Engine* ume da izvrši. Neke logičke operacije je moguće mapirati u više fizičkih, pa je posao optimizatora da izabere najefikasniju.

2.2 Generisanje i procena planova izvršenja (kandidata)

Generisani i odabrani planovi kandidati se čuvaju u memoriji, u komponenti koja se naziva *Memo*. Za procenu cene plana, optimizator procenjuje cenu svakog fizičkog operatora iz toga plana, pomoću formula koje uzimaju u obzir korišćenje I/O, procesora i memorije. Ova cena najviše zavisi od algoritma koji fizički operator koristi, kao i od procenjenog broja slogova koje treba obraditi. Ova procena broja slogova se još naziva i **procena kardinalnosti**.

Za procenu kardinalnosti SQL Server koristi i održava **statistiku optimizatora**, koja sadrži statističke informacije koje opisuju raspodelu vrednosti u okviru neke kolone. Kada se izračuna cena svakog od operatora, njihovim sumiranjem se dobija cena celokupnog plana.

2.3 Keširanje test plana

Generisani plan izvršenja može biti sačuvan u memoriji, u kešu planova, za slučaj ponovnog iskorišćenja, ako se zahteva izvršenje istog upita. Ako je validan plan dostupan u kešu planova, onda se proces optimizacije i procene cena može preskočiti.

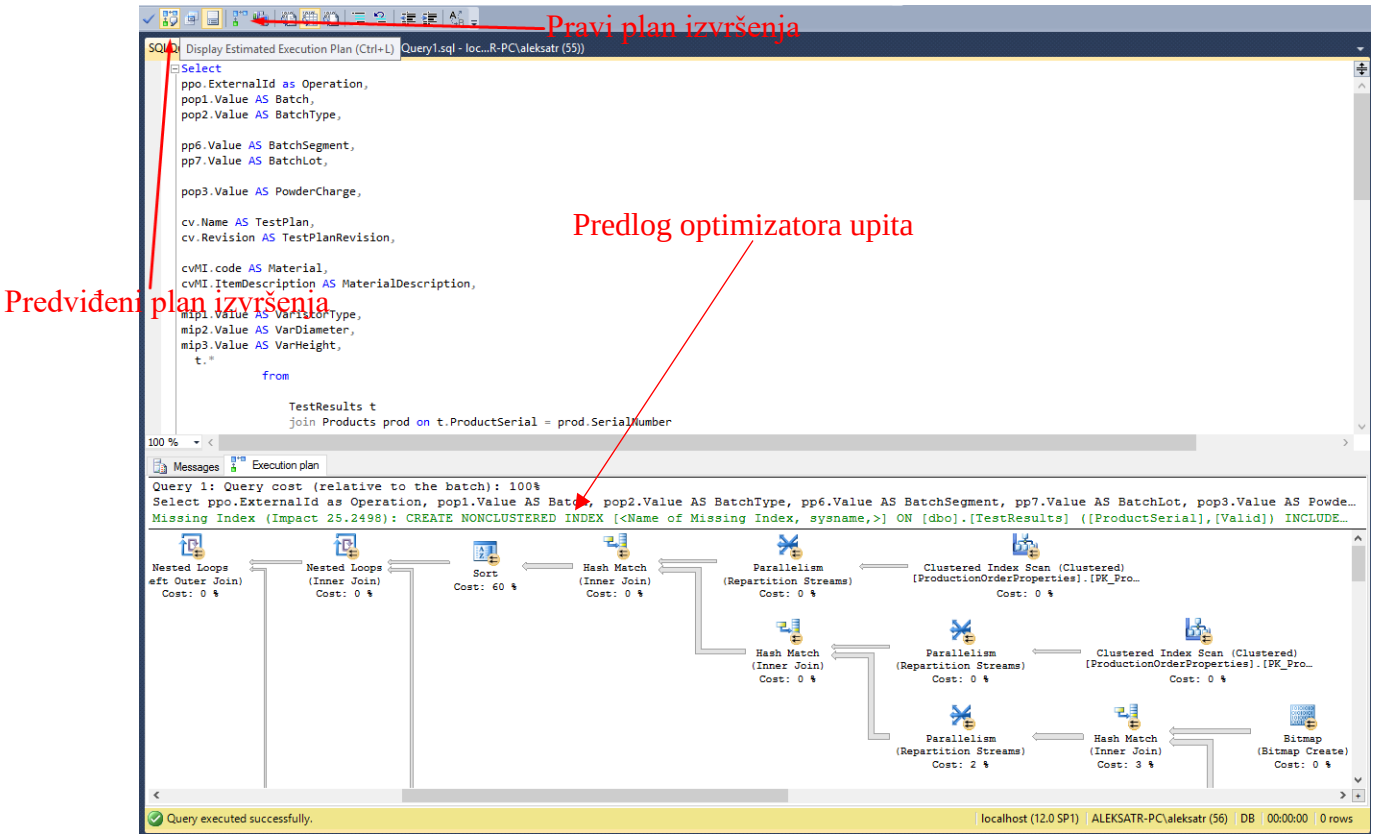
Ponovno iskorišćenje postojećeg plana ne mora uvek biti najbolja opcija. U zavisnosti od raspodele u okviru neke tabele, optimalnost plana može u velikoj meri zavisiti od parametara koji se prosleđuju upitu. Pored ovoga, neki metapodaci mogu biti promenjeni, brisanjem/dodavanjem indeksa, ograničenja, ili čak velikom promenom u sadržaju baze (menja se statistička raspodela vrednosti), čime keširani plan izvršenja postaje suboptimalan. Ovakve test planove SQL Server briše iz keša, kada proceni da postoji mogućnost da više nisu optimalni.

2.4 Hints

Mogu postojati slučajevi kada izabrani plan izvršenja nije dovoljno efikasan. U ovim slučajevima mogu se upotrebiti *hint*-ovi (*hints*). Pošto *hint*-ovi *override*-uju operacije optimizatora, moraju se koristiti oprezno, i samo kao zadnja opcija. Možemo nagovestiti optimizatoru da iskoristi određeni indeks, ili određeni algoritam spoja. Takođe, možemo proslediti kompletan plan izvršenja optimizatoru u vidu XML-a (za slučaj kada imamo plan koji je generisan na nekoj drugoj verziji SQL Servera, a ima bolje performanse).

2.5 Plan izvršenja

Možemo zahtevati pravi plan izvršenja ili procenjeni, koji mogu biti prikazani grafički (u okviru *Microsoft SQL Server Management Studio* alata), kao što je prikazano na *Slici* 2, ili u XML formatu, *Slika* 3. Kada se zahteva pravi plan izvršenja, potrevno je izvršiti upit, dok za procenjeni plan nije potrevno izvršavati upit i on predstavlja plan koji bi SQL Server najverovatnije odabrao, da ga je potrebno izvršiti.



Slika 2 Plan izvršenja (Grafički)

Slika 3 Plan izvršenja (XML)

2.6 Redosled spojeva

Određivanje optimalnog redosleda spojeva je jedan od najkompleksnijih koraka u optimizaciji upita. Skup mogućih planova raste veoma brzo sa porastom broja spojeva tabela. Spoj kombinuje slogove iz dve tabele na osnovu predikata spoja. Spoj *n* tabela zahteva *n-1* spoj, jer je spoj operacija koja se izvršava nad dve tabele. Redosled spojeva utiče na količinu podataka koja se razmenjuje između operatora u planu izvršenja. Optimizator mora da napravi dve važne odluke u vezi spojeva:

1. Izbor redosleda spojeva
2. Izbor algoritma spoja

Kao posledica komutativne i asocijativne osobine operacije spoja, čak i jednostavni upiti nude puno različitih redosleda spojeva, i ovaj broj raste eksponencijalno sa porastom broja tabela koje se spajaju.

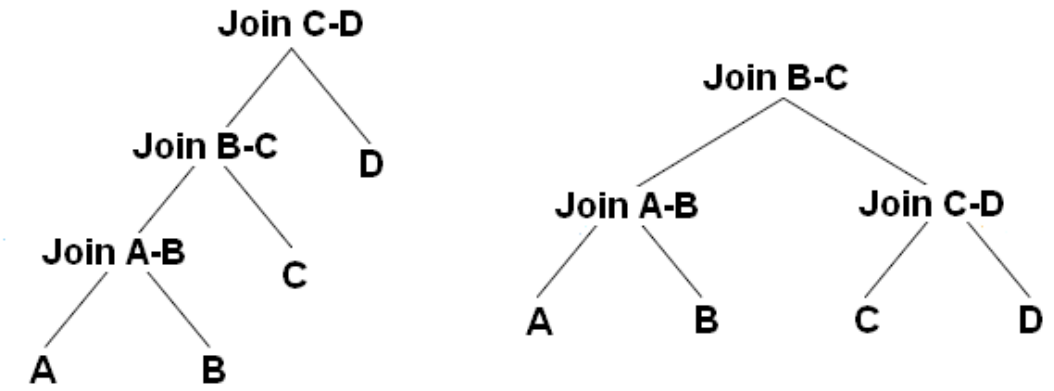
Ako pogledamo plan izvršenja loše struktuiranog upita (prilog A), možemo videti da se redosled spojeva razlikuje od specificiranog u upitu, što znači da je optimizator našao efikasniji redosled.

Možemo naterati optimizator upita da prati redosled spojeva koji je specificiran u upitu. Ovo se obavlja korišćenjem *hint*-ova:

```
OPTION (FORCE ORDER)
```

Proverom plana izvršenja možemo se uveriti da redosled spojeva prati redosled spojeva u upitu. Pošto opitimizator ne može da razmotri sve moguće redoslede spojeva, već samo određeni podskup, *hint*-ovima možemo nekada postići bolje performanse.

Pošto optimizator ne može da razmotri svaki redosled spojeva, on koristi razne heuristike, kao što je oblik stabla upita, da suzi skup redosleda. Neka od ovih stabala imaju imena, kao što su *left-deep*, *right-deep* i *bushy* stabla (*Slika* 4).



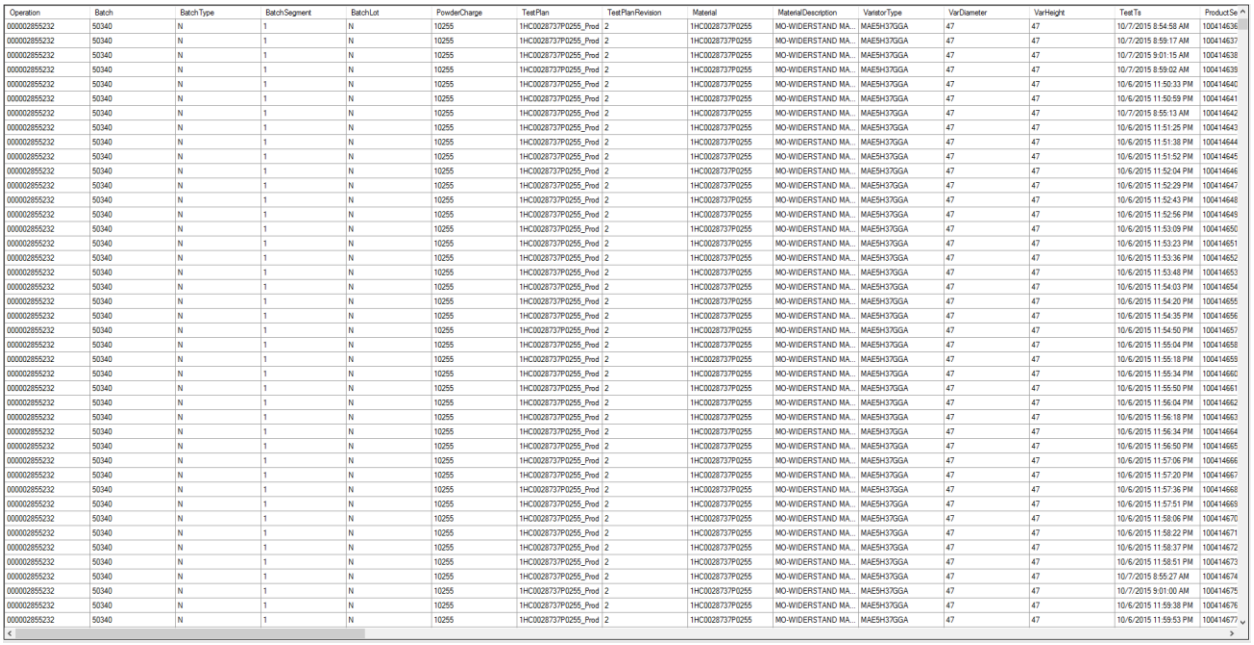
Slika 4 Left-deep i bushy stabla

Broj tabela u spoju	Left-Deep Stabla	Bushy Stabla
1	1	1
2	2	2
3	6	12
4	24	120
5	120	1,680
6	720	30,240
7	5040	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

3 Opis problema

Ovaj rad se bavi analizom procesa optimizacije jednog kompleksnog SQL upita, koji sadrži preko 10 spojeva, sa tabelama koje sadrže nekoliko stotina hiljada slogova. Ovaj upit se koristi u informacionom sistemu koji pruža podršku radnicima u proizvodnji, kao i menadžerima koji prate kako proizvodnja napreduje. Upit pribavlja rezultate raznih električnih testova nad proizvodima, kao i neke osnovne karakteristike samih proizvoda.

Rezultate ovih testiranja je potrebno predstaviti tabelarno, tako da menadžeri mogu pregledati podatke i obaviti određene statističke analize, kao što je prikazano na *Slici* 5.



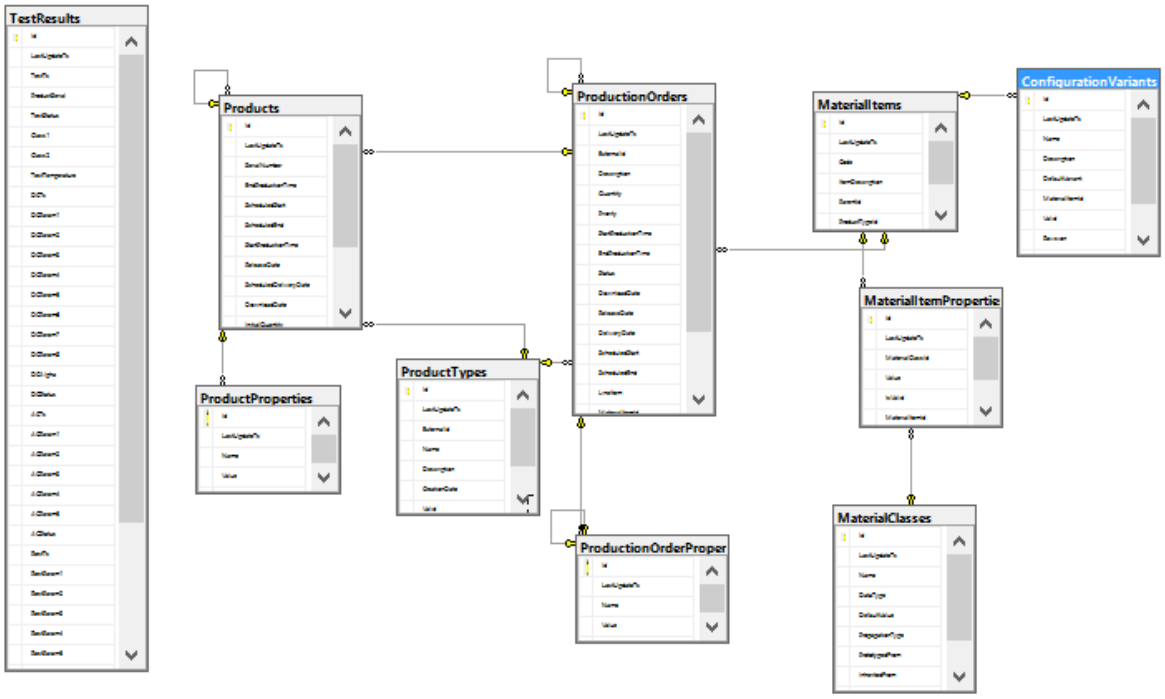
Operation	Batch	BatchType	BatchSegment	BatchLot	PowderCharge	TestPlan	TestPlanRevision	Material	MaterialDescription	VaristorType	VarDiameter	VarHeight	TestTs	ProductSe
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/7/2015 8:54:58 AM	100414636
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/7/2015 8:59:17 AM	100414637
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/7/2015 9:01:15 AM	100414638
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/7/2015 8:59:02 AM	100414639
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:50:33 PM	100414640
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:50:59 PM	100414641
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/7/2015 8:55:13 AM	100414642
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:51:25 PM	100414643
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:51:38 PM	100414644
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:51:52 PM	100414645
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:52:04 PM	100414646
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:52:29 PM	100414647
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:52:43 PM	100414648
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:52:56 PM	100414649
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:53:09 PM	100414650
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:53:23 PM	100414651
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:53:36 PM	100414652
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:53:48 PM	100414653
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:54:03 PM	100414654
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:54:20 PM	100414655
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:54:35 PM	100414656
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:54:50 PM	100414657
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:55:04 PM	100414658
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:55:18 PM	100414659
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:55:34 PM	100414660
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:55:50 PM	100414661
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:56:04 PM	100414662
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:56:18 PM	100414663
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:56:34 PM	100414664
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:56:50 PM	100414665
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:57:06 PM	100414666
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:57:20 PM	100414667
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:57:36 PM	100414668
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:57:51 PM	100414669
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:58:06 PM	100414670
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:58:22 PM	100414671
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:58:37 PM	100414672
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:58:51 PM	100414673
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/7/2015 8:55:27 AM	100414674
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/7/2015 9:01:00 AM	100414675
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:59:38 PM	100414676
00000285232	50340	N	1	N	10255	1HC0028737P0255_Prod	2	1HC0028737P0255	MO-WIDERSTAND MA...	MAE5H37GGA	47	47	10/6/2015 11:59:53 PM	100414677

Slika 5 Prikaz rezultata u DataGridView-u

Pošto je broj rezultata, proizvoda, kao i pratećih podataka bio izuzetno velik, pribavljanje svih podataka u glavnu memoriju, za njihov tabelarni prikaz, nije bilo moguće u razumnom vremenskom intervalu (plus *OutOfMemoryException*). Stoga, iskorišćena je tehnika straničenja, kojom se funkcionisanje tabele (*GridView*-a) omogućilo.

Korisničko iskustvo nije bilo na zadovoljavajućem nivou, pa je upit morao biti ubrzan. Pored ovog ubrzanja, zahtevi su prošireni na proizvoljno filtriranje i sortiranje test rezultata, čime je upit dodatno zakomplikovan i usporen.

Proces optimizacije je prošao kroz više faza (naredna poglavlja), uz proširene korisničke zahteve, zbog kojih je bilo potrebno podržati do 10 miliona test rezultata. ER dijagram baze je dat na *Slici* 6.



Slika 6 ER dijagram baze

Iako ne postoji strani ključ između tabela *TestResults* i *Products*, kao i između tabela *ConfigurationVariantItems* i *ProductionOrderProperties*, postoji spoj ovih tabela koja se koristi u analiziranom SQL upitu.

Tabele *TestResults* i *Products* su spojene preko kolone *SerialNumber* koja je tekstualnog tipa, bez indeksa, što oduzima dosta vremena, na šta nas i sam optimizator upozorava.

Jedna od zahtevnijih operacija je i spoj između tabela *Products* i *ProductProperties*, zbog velikog broja *ProductProperty*-ja, nad čijim stranim ključem ne postoji indeks.

Važno je napomenuti da modifikacija šeme baze nije bila moguća, pa je samu optimizaciju trebalo obaviti dodavanjem objekata koji ne remete rad ostatka sistema.

Baza sadrži 10 tabela:

TestResults – 712.718 podataka o rezultatima testiranja

Products – 697.625 podataka o proizvodima

ProductProperties – 3.486.173 podatka o osobinama proizvoda

ProductionOrders – 642 podatka

ProductionOrderProperties – 11.151 podatka

ProductTypes – 6 podataka

MaterialItems – 105 podataka

MaterialItemProperties – 337 podataka

MaterialClasses – 5 podataka

ConfigurationVariants – 219 podataka

Projekat koji prati ovaj rad se nalazi na adresi: https://github.com/FilipStamenkovic/DBMS

3.1 Inicijalno stanje i modifikacija SQL upita

Inicijalni SQL upit je imao podršku za *paging*, ali zbog slabih performansi i nemogućnosti filtriranja i sortiranja, morao je biti prepravljen.

U prilogu A je dat prikaz inicijalnog SQL upita.

Inicijalni SQL upit je prepravljen tako da podrži filtriranje i sortiranje. Umesto 4 ugnježdena SELECT upita, modifikovani SQL upit ima samo jedan SELECT.

U prilogu B je dat prikaz modifikovanog SQL upita.

4 Dodatna optimizacija upita sa *paging*-om

Kod *paging*-a se može iskoristiti tehnika optimizacija upita, gde se u prvom delu upita pribavljaju primarni ključevi željenih tabela, dok se u drugom delu upita pribavljaju konkretni podaci na osnovu izvučenih primarnih ključeva.

Ova tehnika najviše povećava performance upita kada se upit koristi prilikom *paging*-a i kada upit selektuje mnogo kolona.

Ova tehnika se može primeniti i kada se podaci prikupljaju preko *view*-a, mada su ubrzanja manja nego kod klasičnog SQL upita, ubrzanja i dalje postoje.

Tehnika je detaljnije opisana na sajtu:

https://sqlperformance.com/2015/01/t-sql-queries/pagination-with-offset-fetch

U prilogu C se nalazi SQL upit koji koristi ovu tehniku, dok se u prilogu D nalazi upit koji koristi *view*.

U poglavlju „Grafički prikaz rezultata testiranja“ se može videti odnos vremena izvršenja korišćenjem ove tehnike.

5 Views

Pored klasičnih *view*-eva, SQL server podržava jos 3 različite vrste *view*-a:

- Indexed Views
- Partitioned Views
- System Views

System view se koristi za meta podatke SQL servera, a *partitioned view* se koristi kod distribuiranih baza, gde se različiti delovi podataka nalaze na različitim SQL serverima. Stoga, ova dva tipa *view*-a se ne mogu koristiti za dati upit.

5.1 Standardni View

Pribavljanje podataka pomoću klasičnog SELECT upita nad *view*-om ima slabije performance nego izvršavanje modifikovanog SQL upita. Kod modifikovanog SQL upita imamo razne tehnike ubrzanja koje nisu iskorišćene kod View-a (npr. pribavljanje prvo primarnih ključeva pa kasniji spoj za pribavljanje podataka, *force*-ovanje rasporeda *Query Optimizer*-u itd.).

Kada se ove tehnike primene i nad *view*-om, tada su razlike u performansama zanemarljive.

To je zato što standardni *view* koristi SQL upit u pozadini (kojim je i generisan) prilikom pribavljanja podataka, a taj upit se ne može dalje ubrzati bez indeksa.

5.2 Indexed View

Indexed View se kreira tako što se standardnom *view*-u dodaje jedan jedinstveni klasterovani indeks. Takođe, *view* mora biti kreiran pomoću „WITH SCHEMABINDING“ opcije.

Indexed View ima velika ograničenja prilikom kreiranja, neka od tih ograničenja su da SELECT upit ne sme sadržati:

- COUNT, DISTINCT, ROWSET, MIN, MAX, ORDER BY, TOP, OFFSET
- *Join*-ovanje istih tabela više puta (*self-joins*)
- OUTER *join*-ove (LEFT, RIGHT, FULL)
- *Common table expressions*
- Itd.

Detaljnije informacije o tome kako se kreira *Indexed View* kod SQL Servera se mogu naći na sajtu:

https://msdn.microsoft.com/en-us/library/ms191432.aspx

6 Indeksi

Loše projektovani indeksi, ili nedostatak istih, je obično glavni uzrok loših performansi. U sledećoj tabeli (preuzetoj sa sajta https://msdn.microsoft.com/en-us/library/ms175049.aspx), dat je pregled nekih od podržanih tipova indeksa u SQL Serveru:

Tip indeksa	***Opis***
Hash	Podacima se pristupa preko heš tabele, koja se čuva u memoriji.
Clustered	Klasterovani indeksi čuvaju slogove podataka (tabele ili *view*-a) u redosledu koji odgovara redosledu ključeva indeksa. Ovi indeksi se implementiraju kao B-stabla. Može postojati samo jedan klasterovani indeks nad jednom tabelom.
Nonclustered	Neklasterovani indeksi mogu biti definisani nad tabelom ili *view*-om sa klasterovanim indeksom. Redosled u kojem su slogovi podataka sačuvani ne mora odgovarati redosledu ključeva indeksa.
Unique	Osigurava da ključ indeksa neće sadržati duplikate. Može biti osobina klasterovanih, kao i neklasterovanih indeksa.
Index with included columns	Neklasterovani indeks koji je proširen da uključi i kolone koje nisu ključne.
Filtered	Optimizovani neklasterovani indeks, naročito pogodan za upite koji selektuju podskup podataka na osnovu filter predikata.
Spatial	Prostorni indeks povećava efikasnost izvršenja prostornih upita nad geometrijskim objektima. (SQL Server od verzije 2008 poseduje prostorno proširenje)
XML	Omogućava efikasne operacije nad kolonama koje čuvaju XML BLOB-ove.

Indeks je struktura na disku, koja je vezana za neku tabelu ili *view* i ubrzava pribavljanje podataka iz te tabele. Indeks sadrži ključeve koji predstavljaju vrednosti iz jedne ili više kolona u tabeli ili *view*-u. Ovi ključevi se čuvaju u strukturi podataka B-stablo.

Odabir pravog indeksa za bazu, i upite na koje treba da odgovori, predstavlja složeni kompromis između brzine izvršenja upita i cene *update* operacije. Generalno, treba pratiti sledeće savete:

- Veliki broj indeksa nad nekom tabelom utiče na performanse INSERT, UPDATE, DELETE i MERGE operacija, jer svi indeksi moraju biti prilagođeni novonastalom stanju u tabeli.
- Kreiranje indeksa nad malom tabelom ne mora biti optimalno, jer može biti potrebno više vremena da se obiđe indeks prilikom traženja podatka, nego prostim skeniranjem.
- Indeksi nad *view*-evima mogu značajno povećati performanse kada *view* sadrži agregacije, spojeve ili njihovu kombinaciju. *View* ne mora biti eksplicitno referenciran u upitu, da bi ga optimizator iskoristio.
- Dobri kandidati za neklasterovane indekse su kolone koje učestvuju u spojevima i predikatima.
- Treba iskoristi **Database Engine Tuning Advisor**, koji daje predlog indeksa koje treba kreirati.

Pošto smo ograničeni na jedan klasterovani indeks po tabeli, a svaka tabela već ima kreiran klasterovani indeks nad primarnim ključem, odlučili smo da dodamo neklasterovane indekse nad kolonama koje učestvuju u spojevima problematičnog upita.

```
CREATE INDEX IX_Product_SerialNumber ON Products (SerialNumber);

CREATE INDEX IX_TestResult_ProductSerial on TestResults (ProductSerial);

CREATE INDEX IX_ProductProperty_Product ON ProductProperties (ProductId);

CREATE INDEX IX_ProductionOrderProperty_ProductionOrder ON ProductionOrderProperties
(ProductionOrderId);
```

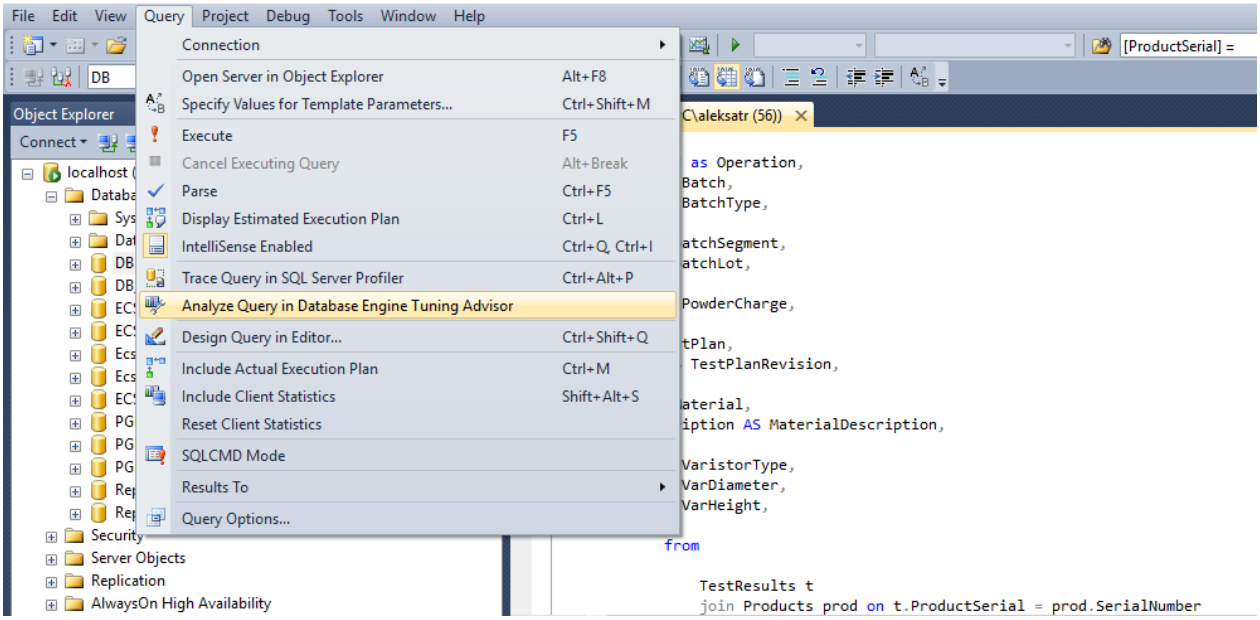
U poglavlju „Grafički prikaz rezultata testiranja“ se može videti povećanje performansa koje korišćenje indeksa donosi.

Predlog optimizatora upita, koji se može videti na formi za prikaz plana izvršenja:

```
CREATE NONCLUSTERED INDEX TestResultsIndex
ON [dbo].[TestResults] ([ProductSerial])
INCLUDE ([Id], [Valid])
```

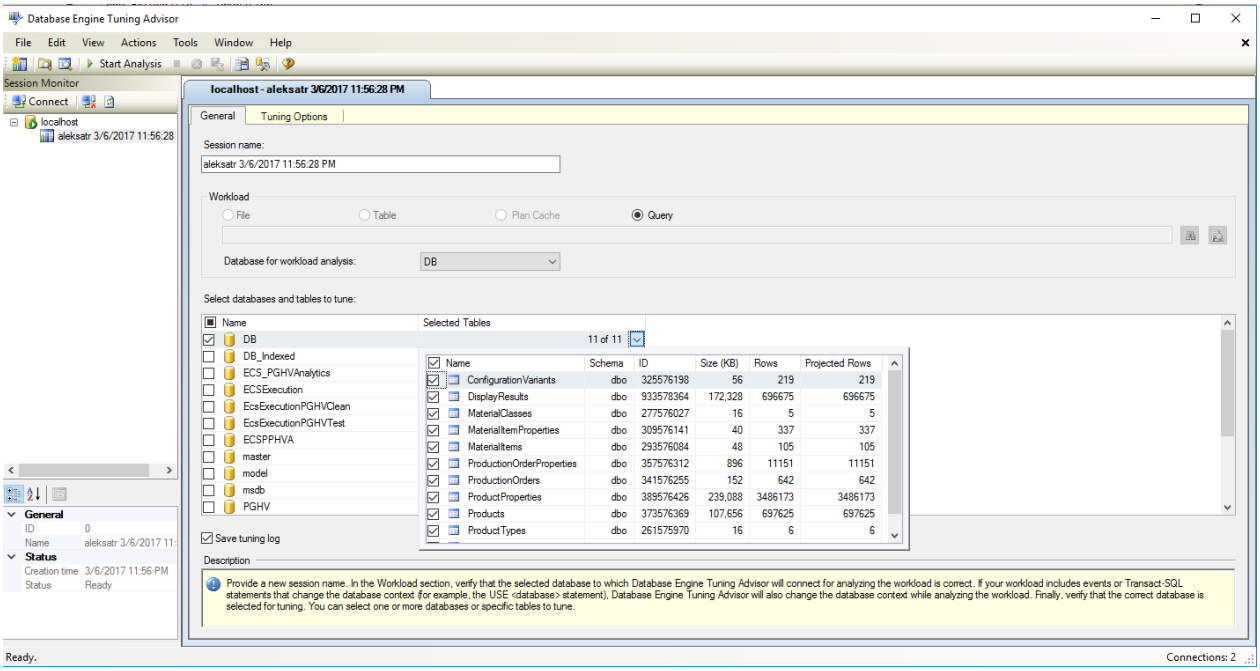
6.1 Predlog Tuning Advisor-a

Predlog *Tuning Advisor*-a možemo zahtevati prečicom koja je prikazana na *Slici* 7.

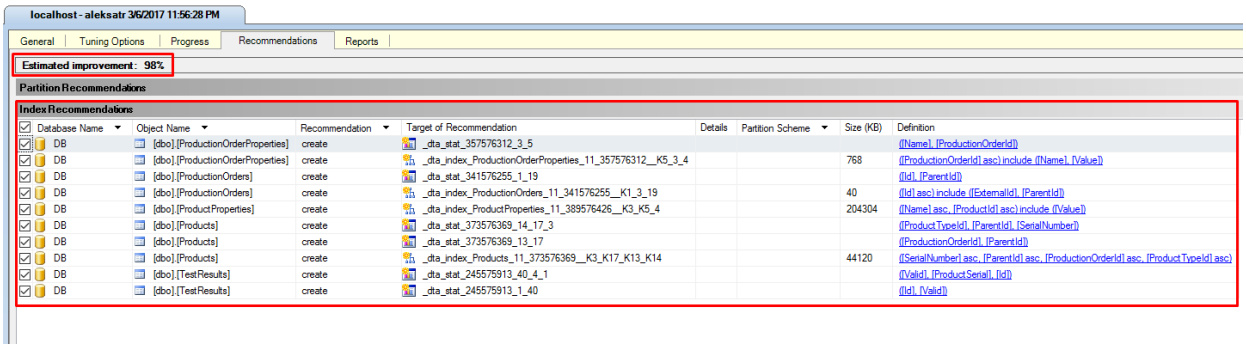


Slika 7 Prečica do Tuning Advisor-a

Potrebno je odabrati bazu nad kojom će se vršiti analiza, kao što je prikazano na *Slici* 8.

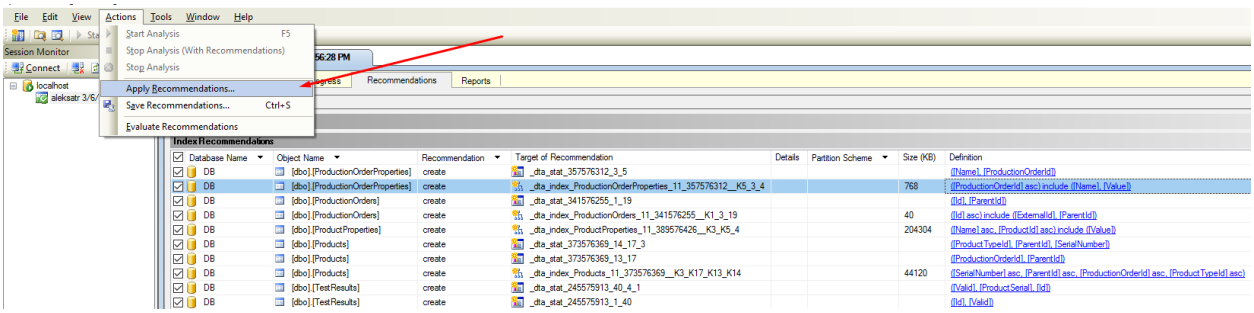


Slika 8 Odabir baze i tabela za analizu



Slika 9 Predlog Tuning Advisor-a

Tuning advisor je predložio više indeksa i statistika za kreiranje (*Slika* 9), koje možemo jednostavno primeniti (*Slika* 10).



Slika 10 Primena predloga

Primer kompozitnog neklasterovanog indeksa koji *Tuning Advisor* predlaže uključuje dodatnu neključnu kolonu (*Value*):

```
CREATE NONCLUSTERED INDEX [_dta_index_ProductProperties_11_389576426__K3_K5_4] ON
[dbo].[ProductProperties]
(
	[Name] ASC,
	[ProductId] ASC
)
INCLUDE ([Value]) WITH (SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON
[PRIMARY]
```

6.2 Dodatna zapažanja

Očigledno je da u našem slučaju optimizator upita ne uzima sve redoslede spojeva u razmatranje, pa mora da nađe balans između vremena za optimizaciju i kvaliteta plana izvršenja. U nekim slučajevima za analizirani SQL upit, dodavanjem opcije FORCE ORDER za pribavljanje prve stranice podataka dobija se ubrzanje do 15%.

Povećanje prioriteta procesa SQL Servera povećava performanse za 10%...

6.3 Grafički prikaz rezultata testiranja

Testiranje je vršeno pomoću SQL upita datih u prilozima A, B, C, D i pomoću Entity Framework-a (prilog E). Testirana su vremena koliko je upitu potrebno za pribavljanje 1. stranice (*offset* = 0), 26. stranice (*offset* = 10.000) i 1251. stranice (*offset* = 500.000).

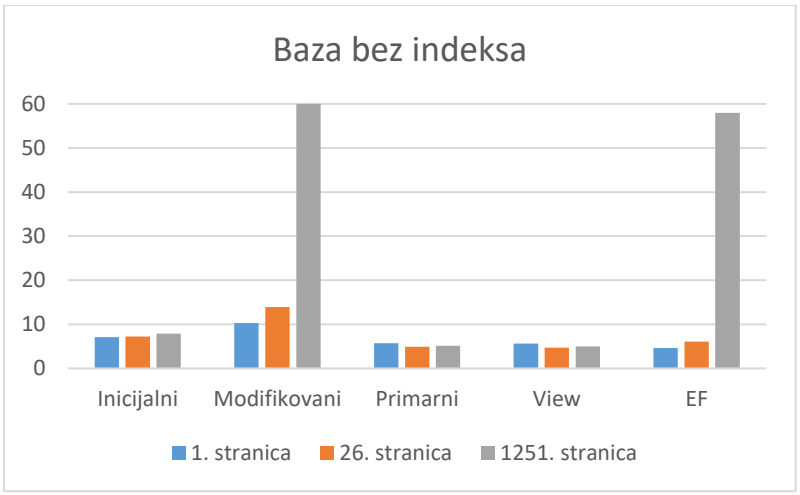
Podešena veličina stranice je 400.

Za testiranje korišćene su 3 baze: baza koja ne sadrži indekse, baza koja sadrži indekse i *Indexed View*, kao i baza koja sadrži indekse koji su kreirani pomoću *SQL Server Tuning Advisor*-a.

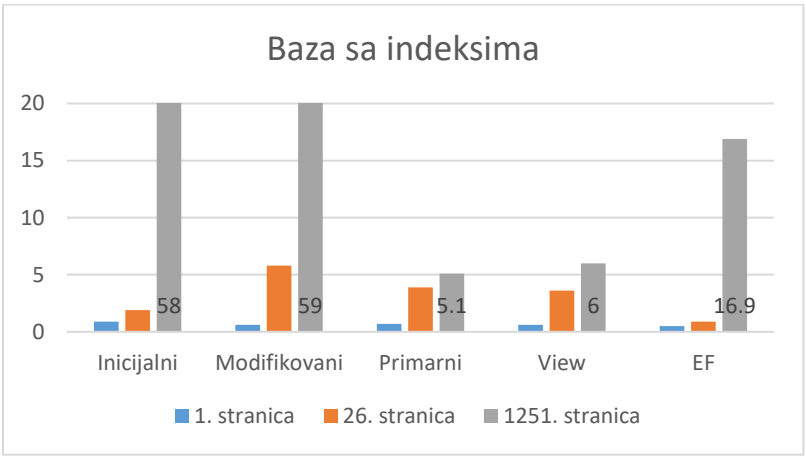
Na histogramima postoje 4 različite grupe:

- Inicijalni – Vreme izvršenja inicijalnog SQL upita (prilog A).
- Modifikovani – Vreme izvršenja modifikovanog upita za filtriranje i sortiranje (prilog B).
- Primarni – Vreme izvršenja upita iz priloga C.
- View – Vreme izvršenja upita pomoću *View*-a (prilog D).
- EF – Vreme izvršenja upita pomoću *Entity Framework*-a (prilog E).

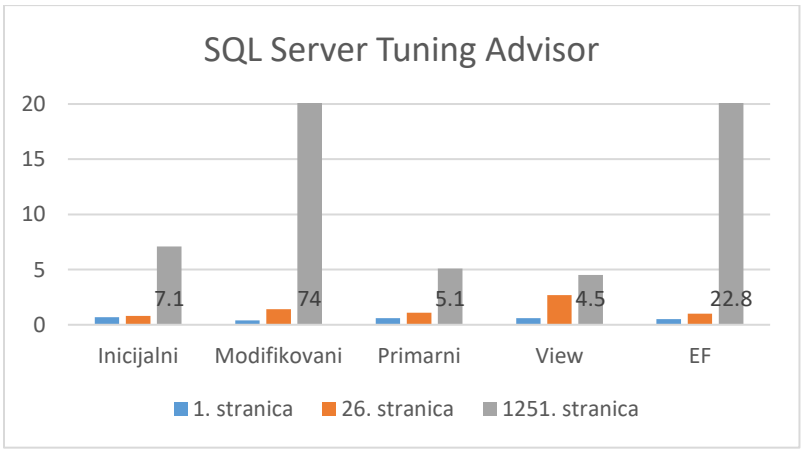
Na *Slici* 11 je prikazano poređenje performansi primenjenih tehnika, nad bazom bez indeksa. Na *Slikama* 12 i 13 je histogramima prikazano poređenje performansi istih tehnika, nad bazom sa indeksima i bazom sa primenjenim preporukama *Tuning Advisor*-a, respektivno.



Slika 11 Poređenje performansi bez indeksa



Slika 12 Poređenje performansi sa indeksima



Slika 13 Poređenje performansi nakon primene Tuning Advisor-a

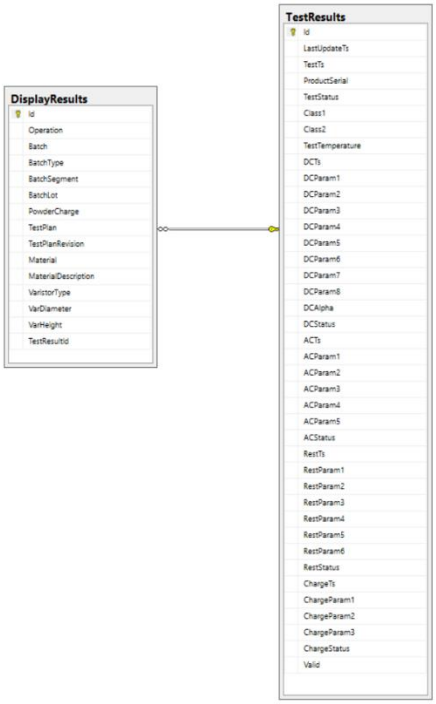
7 Zaključak

Na osnovu rezultata prikazanih u prethodnog poglavlju vidi se da se najbolje performanse dobijaju pomoću SQL Server *Tuning Advisor*-a. Time se može zaključiti da SQL Server ima odlične alate koji se mogu iskoristiti kako bi se određeni upit optimizovao. Ti alati za kratko vreme (nekoliko minuta) mogu predložiti i kreirati više indeksa i statistika koje značajano ubrzavaju vreme izvršenja, dok iskusni database administrator bi morao utrošiti znatno više vremena.

Na osnovu rezultata se može videti da se kreiranjem indeksa pribavljanje 1. stranice podataka značajno ubrzava (ubrzanje veće od 10x), ali se primećuje kako redni broj stranice raste (veći *offset*) tako je ubrzanje sve manje. U nekim slučajevima (za veliki *offset*) se primećuje da su performanse niže nego kada indeksi nisu uopšte ni kreirani. To se dešava kod upita koji moraju obraditi mnogo podataka i koji ne koriste tehniku pribavljanja primarnih ključeva pre ostalih kolona. U ovim slučajevima tehnika *full scan* je brža, zato što čitanje index bloka prouzrokuje još jedan I/O poziv, što dosta degradira performanse.

I pored svih optimizacija korisničko iskustvo nije bilo zadovoljavajuće, jer prilikom pribavljanja stranice sa *offset*-om 500.000+ vreme izvršenja upita nije dovoljno kratko. Kako baza treba podržati 10 miliona slogova, uočava se da bi vreme izvršenja bilo sve veće.

Krajnje rešenje koje je implementirano je kreiranje nove tabele (*DisplayResults*) u kojoj će se nalaziti svi podaci koji nisu dostupni u tabeli *TestResults*, a dobijaju se izvršenjem analiziranog upita. Umesto više različitih spojeva, koji se koriste za pribavljanje karakteristika proizvoda, kreirana tabela agregira te podatke i ima strani ključ ka tabeli *TestResults* (*Slika* 14), nad kojim je kreiran neklasterovani indeks.



Kako za statističku analizu podataka nije od najvišeg prioriteta da podaci budu potpuno ažurni, tabela *DisplayResults* se osvežava noću, kada nije aktivan proces proizvodnje. Time se postižu bolje performanse kada se kreiraju i testiraju novi proizvodi (INSERT naredbe), tokom operativnog dela dana, a i pribavljanje stranica za tabelarni prikaz podataka u *grid*-u je neuporedivo brže (*Slika* 15).

SQL naredba koja kreira novu tabelu i indeks je data u prilogu F, kao i upit koji se izvršava nad novom tabelom.

Slika 14 Tabela DisplayResults

000003146622	50634	N	1	3	10446	MAA6H42GGAP	6	1HC0127492P0001	MO-WIDERSTAND MA...	MAA6H42GGAP	62	42.7	2/10/2017 10:02:16 AM	101232549
000003146622	50634	N	1	3	10446	MAA6H42GGAP	6	1HC0127492P0001	MO-WIDERSTAND MA...	MAA6H42GGAP	62	42.7	2/10/2017 10:02:42 AM	101232550
000003146622	50634	N	1	3	10446	MAA6H42GGAP	6	1HC0127492P0001	MO-WIDERSTAND MA...	MAA6H42GGAP	62	42.7	2/10/2017 10:03:09 AM	101232551
000003146622	50634	N	1	3	10446	MAA6H42GGAP	6	1HC0127492P0001	MO-WIDERSTAND MA...	MAA6H42GGAP	62	42.7	2/10/2017 10:03:37 AM	101232552
000003146622	50634	N	1	3	10446	MAA6H42GGAP	6	1HC0127492P0001	MO-WIDERSTAND MA...	MAA6H42GGAP	62	42.7	2/10/2017 10:04:01 AM	101232553 ...

Slika 15 Vreme izvršenja za offset > 600000

8 Prilog A

```
SELECT distinct

ppo.ExternalId as Operation,

pop1.Value AS Batch,
pop2.Value AS BatchType,

pp6.Value AS BatchSegment,
pp7.Value AS BatchLot,

pop3.Value AS PowderCharge,

cv.Name AS TestPlan,
cv.Revision AS TestPlanRevision,

cvMI.code AS Material,
cvMI.ItemDescription AS MaterialDescription,

mip1.Value AS VaristorType,
mip2.Value AS VarDiameter,
mip3.Value AS VarHeight,
TestResult.*

FROM
        (
    select *
    FROM
                (
                    SELECT ROW_NUMBER() OVER (order by temp.ProductSerial) AS rn, temp.*
                    FROM
                        (
                        Select distinct top 100 percent TestResults.*
                    from
                            Products
                join ProductTypes on Products.ProductTypeId = ProductTypes.Id and ProductTypes.Name = 'Varistor'
                            join TestResults on Products.SerialNumber = TestResults.ProductSerial and
TestResults.Valid = 1,
                ProductionOrders,
                            ProductionOrderProperties
                    WHERE
                            ProductionOrderProperties.Name = 'MOBatch'
                            and
                        Products.ProductionOrderId = ProductionOrders.Id
                            and
                            ProductionOrders.Id = ProductionOrderProperties.ProductionOrderId
                        order by TestResults.ProductSerial asc
                    ) as temp
        ) as temp2
        where temp2.rn between {0} and {1}
    ) TestResult

        left join Products p on TestResult.ProductSerial = p.SerialNumber

        left join ProductionOrders po on po.Id = p.ProductionOrderId
    left join ProductionOrders ppo on ppo.Id = po.ParentId

--      GET TESTPLAN ID
        left join ProductionOrderProperties popTestPlan on po.Id = popTestPlan.ProductionOrderId  and
popTestPlan.Name = 'TestPlanId'

--      GET DATA FROM PRODUCTION ORDER PROPERTY TABLE
        left join ProductionOrderProperties pop1 on po.ParentId = pop1.ProductionOrderId  and pop1.Name =
'MOBatch'
        left join ProductionOrderProperties pop2 on po.ParentId = pop2.ProductionOrderId  and pop2.Name =
'ProductionVersion'
    left join ProductionOrderProperties pop3 on po.ParentId = pop3.ProductionOrderId  and pop3.Name =
'PowderCharge'
```

```
--      GET DATA FROM PRODUCT PROPERTY TABLE
        left join ProductProperties pp6 on p.ParentId = pp6.ProductId and pp6.Name = 'SegmentName'
        left join ProductProperties pp7 on p.ParentId = pp7.ProductId and pp7.Name = 'LayerName'

--      GET CONFIGURATION VARIANT BASED ON FETCHED TESTPLAN ID
        left join ConfigurationVariants cv on CAST (popTestPlan.Value as bigint) = cv.Id
        left join MaterialItems cvMI on cv.MaterialItemId = cvMI.id

--      GET DATA FROM MATERIAL ITEM PROPERTY TABLE
        left join MaterialItemProperties mip1 on cvMI.id = mip1.MaterialItemId and mip1.MaterialClassId in
(select Id from MaterialClasses where Name = 'Var_Typ')
        left join MaterialItemProperties mip2 on cvMI.id = mip2.MaterialItemId and mip2.MaterialClassId in
(select Id from MaterialClasses where Name = 'Diameter')
        left join MaterialItemProperties mip3 on cvMI.id = mip3.MaterialItemId and mip3.MaterialClassId in
(select Id from MaterialClasses where Name = 'Height')
```

9 Prilog B

```
Select
ppo.ExternalId as Operation,
pop1.Value AS Batch,
pop2.Value AS BatchType,
pp6.Value AS BatchSegment,
pp7.Value AS BatchLot,
pop3.Value AS PowderCharge,
cv.Name AS TestPlan,
cv.Revision AS TestPlanRevision,
cvMI.code AS Material,
cvMI.ItemDescription AS MaterialDescription,
mip1.Value AS VaristorType,
mip2.Value AS VarDiameter,
mip3.Value AS VarHeight,  t.*
from
TestResults t
join Products prod on t.ProductSerial = prod.SerialNumber
join ProductTypes pt on pt.Id = prod.ProductTypeId and pt.Name = 'Varistor'
join ProductionOrders prodOrder on prodOrder.Id = prod.ProductionOrderId
join ProductionOrders ppo on ppo.Id = prodOrder.ParentId
join ProductionOrderProperties pop1 on pop1.ProductionOrderId = prodOrder.Id and pop1.Name = 'MOBatch'
join ProductionOrderProperties pop2 on pop2.ProductionOrderId = prodOrder.Id and pop2.Name = 'ProductionVersion'
join ProductionOrderProperties pop3 on pop3.ProductionOrderId = prodOrder.Id and pop3.Name = 'PowderCharge'
left join ProductProperties pp6 on (pp6.ProductId =  prod.ParentId and pp6.Name = 'SegmentName')
left join ProductProperties pp7 on (pp7.ProductId =  prod.ParentId and pp7.Name = 'LayerName')
left join ProductionOrderProperties TestPlan on prodOrder.Id = TestPlan.ProductionOrderId  and TestPlan.Name =
'TestPlanId'
left join ConfigurationVariants cv on CAST (TestPlan.Value as bigint) = cv.Id
left join MaterialItems cvMI on cv.MaterialItemId = cvMI.id
left join MaterialItemPropertIes mip1 on cvMI.id = mip1.MaterialItemId and mip1.MaterialClassId in (select Id
from MaterialClasses where Name = 'Var_Typ')
left join MaterialItemPropertIes mip2 on cvMI.id = mip2.MaterialItemId and mip2.MaterialClassId in (select Id
from MaterialClasses where Name = 'Diameter')
left join MaterialItemProperties mip3 on cvMI.id = mip3.MaterialItemId and mip3.MaterialClassId in (select Id
from MaterialClasses where Name = 'Height')
      WHERE
         t.valid = 1
       {2}
         order by {3} t.Id
OFFSET {0} ROWS
FETCH NEXT {1} ROWS ONLY;
```

10 Prilog C

```
with TestIds as (Select
t.Id as TestResultId,
ppo.Id as OperationId,
pop1.Id AS BatchId,
pop2.Id AS BatchTypeId,

pp6.Id AS BatchSegmentId,
pp7.Id AS BatchLotId,

pop3.Id AS PowderChargeId,

cv.Id AS TestPlanId,

cvMI.Id AS MaterialId,

mip1.Id AS VaristorTypeId,
mip2.Id AS VarDiameterId,
mip3.Id AS VarHeightId
from
TestResults t
join Products prod on t.ProductSerial = prod.SerialNumber

--       join ProductTypes pt on pt.Id = prod.ProductTypeId and pt.Name = 'Varistor'

join ProductionOrders prodOrder on prodOrder.Id = prod.ProductionOrderId
join ProductionOrders ppo on ppo.Id = prodOrder.ParentId

left join ProductionOrderProperties TestPlan on prodOrder.Id = TestPlan.ProductionOrderId  and TestPlan.Name =
'TestPlanId'
--       GET CONFIGURATION VARIANT BASED ON FETCHED TESTPLAN ID
left join ConfigurationVariants cv on CAST (TestPlan.Value as bigint) = cv.Id
left join MaterialItems cvMI on cv.MaterialItemId = cvMI.id

--       GET DATA FROM MATERIAL ITEM PROPERTY TABLE
left join MaterialItemPropertIes mip1 on cvMI.id = mip1.MaterialItemId and mip1.MaterialClassId in (select Id
from MaterialClasses where Name = 'Var_Typ')
left join MaterialItemPropertIes mip2 on cvMI.id = mip2.MaterialItemId and mip2.MaterialClassId in (select Id
from MaterialClasses where Name = 'Diameter')
left join MaterialItemProperties mip3 on cvMI.id = mip3.MaterialItemId and mip3.MaterialClassId in (select Id
from MaterialClasses where Name = 'Height')

join ProductionOrderProperties pop1 on pop1.ProductionOrderId = prodOrder.Id and pop1.Name = 'MOBatch'
join ProductionOrderProperties pop2 on pop2.ProductionOrderId = prodOrder.Id and pop2.Name = 'ProductionVersion'
join ProductionOrderProperties pop3 on pop3.ProductionOrderId = prodOrder.Id and pop3.Name = 'PowderCharge'
left join ProductProperties pp6 on (pp6.ProductId =  prod.ParentId and pp6.Name = 'SegmentName')
left join ProductProperties pp7 on (pp7.ProductId =  prod.ParentId and pp7.Name = 'LayerName')
WHERE
t.valid = 1
{2}
order by {3} t.Id
OFFSET {0} ROWS
FETCH NEXT {1} ROWS ONLY)

Select
ppo.ExternalId as Operation,
pop1.Value AS Batch,
pop2.Value AS BatchType,

pp6.Value AS BatchSegment,
pp7.Value AS BatchLot,

pop3.Value AS PowderCharge,

cv.Name AS TestPlan,
cv.Revision AS TestPlanRevision,

cvMI.code AS Material,
cvMI.ItemDescription AS MaterialDescription,
```

```
mip1.Value AS VaristorType,
mip2.Value AS VarDiameter,
mip3.Value AS VarHeight,
t.*
from
        TestIds ids
        join TestResults t on t.Id = ids.TestResultId
        join ProductionOrders ppo on ppo.Id = ids.OperationId
        join ProductionOrderProperties pop1 on pop1.Id = ids.BatchId
        join ProductionOrderProperties pop2 on pop2.Id = ids.BatchTypeId
        join ProductionOrderProperties pop3 on pop3.Id = ids.PowderChargeId
        left join ProductProperties pp6 on (pp6.Id =  ids.BatchSegmentId)
        left join ProductProperties pp7 on (pp7.Id =  ids.BatchLotId )

        left join ConfigurationVariants cv on cv.Id = ids.TestPlanId
        left join MaterialItems cvMI on cvMI.Id = ids.MaterialId

        left join MaterialItemPropertIes mip1 on mip1.Id = ids.VaristorTypeId
        left join MaterialItemPropertIes mip2 on mip2.Id = ids.VarDiameterId
        left join MaterialItemProperties mip3 on mip3.Id = ids.VarHeightId

order {3} by t.Id

OPTION (FORCE ORDER)
```

11 Prilog D

```
with pg as (SELECT
    p.id
  FROM PaggingView p
    {2}
         order by {3} p.id
  offset {0} rows fetch next {1} rows only)

  select
p.*
from PaggingView p
  inner join pg on pg.id = p.id
  order by {3} p.Id;
```

12 Prilog E

12.1 Entity Framework

Isti upit korišćenjem ORM-a i LINQ to SQL sintakse:

```
var validTestResults = dbModel.TestResults.Where(tr => tr.Valid);
var varistorProducts = dbModel.Products.Where(p => p.ProductType.Name == "Varistor");

var q1 = varistorProducts.Join(validTestResults, p => p.SerialNumber, tr => tr.ProductSerial
    , (p, tr) => new { Product = p, TestResult = tr, TestPlanId =
p.ProductionOrder.ProductionOrderProperties.FirstOrDefault(pop => pop.Name == "TestPlanId") });

var q2 = q1.Select(a => new {
    Product = a.Product
    , TestResult = a.TestResult
    , TestPlanId = a.TestPlanId
    , ConfigurationVariant = (a.TestPlanId == null) ? null : dbModel.ConfigurationVariants.FirstOrDefault(c =>
c.Id.ToString() == a.TestPlanId.Value)
    , Batch = a.Product.ProductionOrder.ProductionOrderProperties.FirstOrDefault(pop => pop.Name ==
"MOBatch").Value
    , BatchType = a.Product.ProductionOrder.ProductionOrderProperties.FirstOrDefault(pop => pop.Name ==
"ProductionVersion").Value
    , BatchSegment = a.Product.Parent.ProductProperties.FirstOrDefault(pp => pp.Name == "SegmentName").Value
    , BatchLot = a.Product.Parent.ProductProperties.FirstOrDefault(pp => pp.Name == "LayerName").Value
    , PowderCharge = a.Product.ProductionOrder.ProductionOrderProperties.FirstOrDefault(pop => pop.Name ==
"PowderCharge").Value
});

var q3 = q2.Select(a => new {
    Product = a.Product
    , TestResult = a.TestResult
    , TestPlanId = a.TestPlanId
    , ConfigurationVariant = a.ConfigurationVariant
    , Batch = a.Batch
    , BatchType = a.BatchType
    , BatchSegment = a.BatchSegment
    , BatchLot = a.BatchLot
    , PowderCharge = a.PowderCharge
    , mip1 = (a.ConfigurationVariant == null || a.ConfigurationVariant.MaterialItem == null) ? null :
a.ConfigurationVariant.MaterialItem.MaterialItemProperties.FirstOrDefault(p => p.MaterialClass.Name == "Var_Typ")
    , mip2 = (a.ConfigurationVariant == null || a.ConfigurationVariant.MaterialItem == null) ? null :
a.ConfigurationVariant.MaterialItem.MaterialItemProperties.FirstOrDefault(p => p.MaterialClass.Name ==
"Diameter")
    , mip3 = (a.ConfigurationVariant == null || a.ConfigurationVariant.MaterialItem == null) ? null :
a.ConfigurationVariant.MaterialItem.MaterialItemProperties.FirstOrDefault(p => p.MaterialClass.Name == "Height")
});

query = q3.OrderBy(a => a.Product.Id).Select(a => new
{
    Operation = a.Product.ProductionOrder.Parent.ExternalId
    , Batch = a.Batch
    , BatchType = a.BatchType
    , BatchSegment = a.BatchSegment
    , BatchLot = a.BatchLot
    , PowderCharge = a.PowderCharge
    , TestPlan = a.ConfigurationVariant == null ? null : a.ConfigurationVariant.Name
    , TestPlanRevision = a.ConfigurationVariant == null ? null : a.ConfigurationVariant.Revision
    , Material = a.ConfigurationVariant == null ? null : a.ConfigurationVariant.MaterialItem.Code
    , MaterialDescription = a.ConfigurationVariant == null ? null :
a.ConfigurationVariant.MaterialItem.ItemDescription
    , VaristorType = a.mip1 == null ? null : a.mip1.Value
    , VarDiameter = a.mip2 == null ? null : a.mip2.Value
    , VarHeight = a.mip3 == null ? null : a.mip2.Value
    , TestResult = a.TestResult
});

var page = query.Skip(requestedPageNumber * pageSize).Take(pageSize).ToArray();
```

13 Prilog F

```
select ROW_NUMBER() OVER(ORDER BY pv.Id ASC) as Id, pv.Operation, pv.Batch, pv.BatchType,
 pv.BatchSegment, pv.BatchLot, pv.PowderCharge, pv.TestPlan   , pv.TestPlanRevision,
  pv.Material, pv.MaterialDescription, pv.VaristorType, pv.VarDiameter, pv.VarHeight, pv.Id as TestResultId
into
DisplayResults
from
PaggingView pv;

go

--index
CREATE INDEX IX_DisplayResult_TestResult ON DisplayResults (TestResultId);


--query
with pg as (SELECT
    p.Id as pid, r.Id as rid
  from DisplayResults p inner join TestResults r on p.TestResultId = r.Id
    {2}
        order by {3} p.Id
  offset {0} rows fetch next {1} rows only)

  select p.*,r.*
from DisplayResults p,  TestResults r, pg
where
pg.pid = p.Id
and
pg.rid = r.Id
  order by {3} pg.pid
```

14 Literatura

- https://msdn.microsoft.com/en-us/library/ff650689.aspx
- https://msdn.microsoft.com/en-us/library/ms190174.aspx
- https://msdn.microsoft.com/en-us/library/ms191432.aspx
- https://technet.microsoft.com/en-us/library/dd171921(v=sql.100).aspx
- https://sqlperformance.com/2015/01/t-sql-queries/pagination-with-offset-fetch
- https://msdn.microsoft.com/en-us/library/ms181714.aspx
- https://msdn.microsoft.com/en-us/library/ms188709.aspx
- https://technet.microsoft.com/en-us/library/ms176005(v=sql.105).aspx
- http://www.sql-server-performance.com/2013/sql-server-query-optimization-part-1/
- https://blog.sqlauthority.com/2013/08/28/sql-server-tips-for-sql-query-optimization-by-analyzing-query-plan/
- https://www.simple-talk.com/sql/sql-training/the-sql-server-query-optimizer/
- https://technet.microsoft.com/en-us/library/jj835095(v=sql.110).aspx