UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matevž Grbec

Optimizacija zmogljivosti sistemov OLAP, zgrajenih z orodjem MS Analysis Services 2005

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: izr. prof. dr. Marko Bajec

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01652/2010

Datum: 05.04.2010

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATEVŽ GRBEC**

Naslov: **OPTIMIZACIJA ZMOGLJIVOSTI SISTEMOV OLAP, ZGRAJENIH Z ORODJEM MS ANALYSIS SERVICES 2005**

OPTIMISATION OF OLAP APPLICATIONS BASED ON MS ANALYSIS SERVICES 2005

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

V diplomskem delu preučite področje optimizacije sistemov OLAP. Za potrebe identifikacije konkretnih pristopov k optimizaciji izberite enega od odprtokodnih OLAP sistemov ali enega od komercialnih sistemov, če so vam na voljo. Za izbran sistem testirajte in ocenite primernost identificiranih načinov optimizacije.

Mentor:

prof. dr. Marko Bajec

Fakulteta za računalništvo in informatiko
UNIVERZA V LJUBLJANI
2

Dekan:

prof. dr. Nikolaj Zimic

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01652/2010
Datum: 05.04.2010

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATEVŽ GRBEC**

Naslov: **OPTIMIZACIJA ZMOGLJIVOSTI SISTEMOV OLAP, ZGRAJENIH Z ORODJEM MS ANALYSIS SERVICES 2005**

OPTIMISATION OF OLAP APPLICATIONS BASED ON MS ANALYSIS SERVICES 2005

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

V diplomskem delu preučite področje optimizacije sistemov OLAP. Za potrebe identifikacije konkretnih pristopov k optimizaciji izberite enega od odprtokodnih OLAP sistemov ali enega od komercialnih sistemov, če so vam na voljo. Za izbran sistem testirajte in ocenite primernost identificiranih načinov optimizacije.

Mentor:

prof. dr. Marko Bajec

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani Matevž Grbec,

z vpisno številko 63040043,

sem avtor diplomskega dela z naslovom:

Optimizacija zmogljivosti sistemov OLAP, zgrajenih z orodjem MS Analysis Services 2005

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Marko Bajec
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 13. 10. 2010 Podpis avtorja: _____

Zahvala

Kratko in iskreno zahvalo želim izreči vsem, ki so mi omogočili ali na kakršen koli način pozitivno vplivali na moj študij in življenje od vpisa na fakulteto do izdelave diplomske naloge.

Kazalo

Povzetek 1
Abstract 3
1 Uvod 5
2 Poslovna inteligenca in OLAP 7
2.1 Odločanje v organizaciji 7
2.1.1 Kdo sprejema odločitve? 7
2.1.2 Kaj je učinkovita odločitev? 7
2.2 Poslovna inteligenca - BI 8
2.2.1 Načini zagotavljanja ustreznih informacij 8
2.3 Podatkovno skladišče kot vir podatkov za BI 9
2.3.1 Transakcijski podatki 9
2.3.2 Podatkovno skladišče 9
2.4 Sistemi OLAP 9
2.4.1 Mera 10
2.4.2 Dimenzija 10
2.4.3 Kocka 10
2.4.4 Prednosti sistemov OLAP 10
2.5 Jezik MDX 10
2.5.1 Osnovna zgradba poizvedbe MDX 11
2.6 Microsoft SQL Server Analysis Services 2005 11
3 Arhitektura poizvedovanja in procesiranja v SSAS 13
3.1 Arhitektura poizvedovanja 13
3.1.1 Upravljanje s sejami 13
3.1.2 Izvajanje poizvedb MDX 14
3.1.3 Pridobivanje podatkov o dimenzijah 14
3.1.4 Pridobivanje podatkov o merah 16
3.2 Arhitektura procesiranja 18
3.2.1 Posli pri procesiranju dimenzij 18
3.2.2 Ukazi za procesiranje dimenzij 20
3.2.3 Posli pri procesiranju particij 20
3.2.4 Ukazi za procesiranje particij 21

3.2.5 Izvajanje poslov procesiranja ... 21
4 Načini optimizacije sistemov OLAP ... 23
4.1 Podatkovno skladišče ... 23
4.1.1 Stolpci in vrstice ... 23
4.1.2 Ključi ... 24
4.1.3 Indeksi ... 24
4.1.4 Tabele in podatkovna baza ... 26
4.2 Zasnova podatkovne kocke ... 28
4.2.1 Dimenzije ... 28
4.2.2 Kocka ... 31
4.3 Zmogljivost poizvedb ... 32
4.3.1 Način shranjevanja mer in agregacij ... 32
4.3.2 Uporaba particij ... 32
4.3.3 Uporaba agregacij ... 33
4.3.4 Poizvedbe MDX ... 37
4.3.5 Uporaba predpomnilnika ... 43
4.4 Zmogljivost procesiranja ... 44
4.4.1 Začasna mapa ... 44
4.4.2 Uporabi OLE DB Provider ... 44
4.4.3 Dimenzije ... 44
4.4.4 Kocka ... 45
4.5 Posebne zasnove ... 46
4.5.1 Posebna agregacijska funkcija – DistinctCount ... 46
4.5.2 Semiaditivne agregacijske funkcije ... 47
4.5.3 Referenčna razmerja ... 47
4.5.4 Razmerja mnogo proti mnogo ... 47
4.6 Nastavitve strežnika in strojna oprema ... 51
4.6.1 Učinkovita raba pomnilnika ... 52
4.6.2 Učinkovita raba procesorja ... 54
4.6.3 Učinkovita raba diska ... 55
4.6.4 Strojna oprema ... 55
5 Primer optimizacije sistema OLAP ... 57
5.1 Opis testnega okolja ... 57

5.1.1 Strojna oprema ... 57
5.1.2 Programska oprema ... 57
5.2 Metodologija testiranja ... 57
5.3 Predstavitev testnih podatkov ... 58
5.4 Predstavitev testnih poizvedb ... 60
5.5 Rezultati testiranja ... 62
5.5.1 Začetno stanje ... 62
5.5.2 Optimizacija podatkovnega skladišča ... 63
5.5.3 Optimizacija zasnove podatkovne kocke ... 65
5.5.4 Optimizacija zmogljivosti poizvedb ... 67
5.5.5 Optimizacija zmogljivosti procesiranja ... 69
5.5.6 Optimizacija posebnih zasnov ... 70
5.5.7 Optimizacija nastavitev strežnika in strojne opreme ... 70
5.6 Analiza rezultatov ... 71
5.6.1 Velikost podatkovnega skladišča ... 72
5.6.2 Velikost večdimenzijske podatkovne baze SSAS ... 72
5.6.3 Čas procesiranja ... 73
5.6.4 Čas izvajanja poizvedb ... 74
5.6.5 Dodatna opažanja ... 75
6 Zaključek ... 77
Dodatek A ... 79
Dodatek B ... 85
Dodatek C ... 87
Dodatek Č ... 91
Seznam slik ... 93
Seznam tabel ... 95
Literatura ... 97

Seznam uporabljenih kratic in simbolov

BI	Business intelligence
EIS	Executive information system
DSS	Decision support system
OLTP	Online Transaction Processing
OLAP	Online Analytical Processing
MDX	MultiDimensional eXpressions
SSAS	SQL Server Analysis Services
SSIS	SQL Server Integration Services
SSRS	SQL Server Reporting Services
MOLAP	Multidimensional Online Analytical Processing
BIDS	Business Intelligence Development Studio
HOLAP	Hybrid Online Analytical Processing
ROLAP	Relational Online Analytical Processing
SSMS	SQL Server Management Studio

Povzetek

V diplomski nalogi obravnavam področje optimizacije sistemov OLAP. Najprej predstavim pojme poslovne inteligence in sistemov OLAP, ki v organizacijah podpirajo sprejemanje učinkovitih odločitev.

V osrednjem delu opišem možne načine za krajšanje časa poizvedovanja in časa procesiranja v sistemih OLAP, zgrajenih s pomočjo produkta SQL Server Analysis Services 2005. Možnih načinov optimizacije je veliko in sem jih razdelil na naslednja področja: podatkovno skladišče, zasnova podatkovne kocke, zmogljivost poizvedb, zmogljivost procesiranja, posebne zasnove, nastavitve strežnika in strojna oprema.

Namen diplomske naloge je testirati in oceniti posamezne načine optimizacije. Zanima nas, kolikšno izboljšanje dosežemo z različnimi načini in predvsem katere postopke je najbolj priporočljivo uporabiti. Testiranje sem izvedel na pravih podatkih slovenskega podjetja, in sicer tako, da sem po vsakem optimizacijskem koraku izmeril čas procesiranja in čas izvajanja testnih poizvedb. Po testiranju sem dobljene rezultate analiziral in v zaključku podal strnjene ugotovitve in priporočila.

Ključne besede:

optimizacija, sistem OLAP, Microsoft SQL Analysis Services 2005, poizvedba MDX

Abstract

This work covers optimization of OLAP systems. In the beginning we define terms of business intelligence and OLAP systems which purpose is to support making effective decisions in organizations.

The core of this work is a description of possible methods for improvement of processing and querying times in OLAP systems built with the product SQL Server Analysis Services 2005. There are many different optimization methods and we arranged them into the following fields: data warehouse, design of the data cube, querying performance, processing performance, special designs, server settings and hardware.

The goal of this work is to test and assess individual optimization methods. For each method we are interested in the amount of possible performance improvement. Finally we would like to find out which methods are the most effective. We tested our optimization methods on the real data from a Slovenian company. After each optimization step we measured processing time and time required to get results from the test queries. At the end we analyze the results and give final findings and recommendations.

Key words:

optimization, OLAP system, Microsoft SQL Analysis Services 2005, MDX query

1 Uvod

Organizacije iz leta v leto ustvarjajo in shranjujejo vedno več podatkov. Vse te podatke želijo analizirati in izsledke uporabiti kot podlago za sprejemanje poslovnih odločitev na različnih ravneh odločanja. V ta namen so se razvili sistemi poslovne inteligence, ki so realizirani s pomočjo tehnologije OLAP. Sistemi OLAP so prilagojeni potrebam analitike in se zato v marsičem razlikujejo od transakcijskih sistemov.

Po raziskavah vodilnega analitičnega podjetja Gartner [3, 4, 5] je poslovna inteligenca (BI) že nekaj let na seznamu deset najpomembnejših tehnoloških prioritet podjetij po svetu. Razlog za to je gotovo dostopnost orodij za poslovno inteligenco ter strateška prednost, ki jo le-ta prinaša podjetjem.

Vendar se tekom projektov implementacije poslovne inteligence v podjetjih pogosto izkaže, da je treba premostiti določene probleme, če želimo, da je projekt uspešen. Najpogosteje omenjan tehnični problem OLAP produktov so prepočasni odgovori na poizvedbe [19]. Neodvisna raziskava na Nizozemskem je pokazala, da so poslovni uporabniki pripravljeni čakati na rezultat poizvedbe največ 30 sekund, nato pa so prepričani, da je prišlo do napake v sistemu in želijo proces prekiniti. Zato je eden od glavnih izzivov BI projektov zagotoviti čim boljšo zmogljivost sistemov OLAP.

Slovar slovenskega knjižnega jezika definira zmogljivost na več načinov. Za nas je najprimernejša naslednja definicija [8]: »sposobnost česa, določena z največjo količino izdelkov, največjim številom dejanj, ki jih zmore v določenem času«. V računalniški stroki se zmogljivost običajno meri s hitrostjo delovanja in skalabilnostjo. Še konkretnejše: sistem OLAP je dovolj zmogljiv, če na večino uporabniških zahtev odgovori v roku 5 sekund, pri čemer na enostavnejše poizvedbe odgovori v največ 1 sekundi, zelo redko pa je potrebno na odgovor čakati več kot 20 sekund [19].

Zgodi se, da sistem v začetni fazi (ali kasneje) ne ustreza zgornjim specifikacijam zmogljivosti. Težave so pogostejše pri obsežnejših sistemih (zapleten podatkovni model ali velika količina podatkov). V tem primeru poskusimo z različnimi metodami optimizacije doseči potrebne izboljšave.

Glavni cilj tega diplomskega dela je raziskati možne načine optimizacije z uporabo orodja Microsoft SQL Server 2005 in njegove komponente SQL Server Analysis Services (krajše: SSAS). Podjetje Microsoft je že nekaj časa med vodilnimi proizvajalci BI platform [21] in podjetja se pogosto odločajo za njihove rešitve.

Na začetku diplomskega dela podrobneje predstavim področje poslovne inteligence in sisteme OLAP, s poudarkom na podpori procesom odločanja. Sledi opis arhitekture procesiranja podatkov in izvajanja poizvedb v SSAS.

V nadaljevanju opredelim možne načine optimizacije sistemov OLAP, ki so realizirani z orodjem SSAS. Ker gre za kompleksne sisteme, je optimizacija možna na več nivojih. V

grobem sem jo razdelil na 6 nivojev, ki se med seboj delno prekrivajo, tako kot njihovi načini izboljšav:

- Podatkovno skladišče
- Zasnova podatkovne kocke
- Zmogljivost poizvedb
- Zmogljivost procesiranja
- Posebne zasnove
- Nastavitve strežnika

Namen pričujočega dela ni predstavitev orodja SSAS ter način gradnje kock, dimenzij in mer. Osredotočim se zgolj na načine, ki so nam na voljo po tem, ko imamo osnovno rešitev že postavljeno.

Večino ugotovljenih metod optimizacije sem preizkusil tudi v praksi. Zaradi želje zagotoviti čim bolj realne življenjske pogoje, sem testiranje opravil na dejanskih podatkih slovenskega podjetja iz živilske industrije. Postavil sem popolnoma neoptimiziran sistem in ga nato postopoma optimiziral s posameznimi koraki, opredeljenimi v teoretičnem delu diplomskega dela. V tem delu so predstavljene opravljene meritve in njihovi rezultati. Namen je ugotoviti, s katerimi metodami pridobimo največ in koliko se sploh da pridobiti.

V zaključku podam strnjene ugotovitve in dodatne ideje, ki so se mi porodile tekom izdelave tega dela.

2 Poslovna inteligenca in OLAP

2.1 Odločanje v organizaciji

Življenje je polno odločitev. Na dnevni ravni se srečujemo z odločitvami kot so: »Ali naj za kosilo jem makarone ali dunajski zrezek?«, »Ali naj prihranke uporabim za počitnice ali naj denar investiram v investicijske sklade?« in podobno. Izbire, za katere se odločimo, imajo lahko ogromne posledice na naše življenje [10].

Sprejemanje dobrih odločitev je, tako kot v našem vsakdanjem življenju, pomembno tudi za organizacije. V organizacijah vsak dan sprejmejo veliko odločitev, ki določajo njihovo usmeritev in učinkovitost. Odločitve sprejemajo v proizvodnji, marketingu, kadrovski službi, itd. Ključ do uspeha organizacije je sprejemanje dobrih odločitev.

2.1.1 Kdo sprejema odločitve?

Odločitve sprejemajo vsi člani organizacije na vseh nivojih organizacijske hierarhije [10]. Zmotno je misliti, da je uspeh organizacije odvisen samo od odločitev direktorjev, predsednikov in ostalih ljudi, ki so po položaju na vrhu hierarhije. Izkušnje kažejo, da je veliko odličnih strateških planov propadlo zaradi slabo sprejetih odločitev tistih, ki so bili odgovorni za implementacijo teh odličnih planov. Uspeh organizacije je zato zagotovljen šele takrat, ko učinkovite odločitve sprejemajo tudi vodje posameznih oddelkov in projektnih skupin ter vsi ostali člani nižje po organizacijski hierarhiji.

2.1.2 Kaj je učinkovita odločitev?

Učinkovito odločitev definiramo kot: Učinkovite odločitve so odločitve, ki organizacijo pravočasno pomikajo bližje dogovorjeni skupini ciljev [10].

Iz tega lahko izluščimo tri glavne ključe za učinkovito odločanje (slika1, vir [10]):

1.) Obstajati mora skupina ciljev, katerim se želimo približevati.
2.) Obstajati mora način, s katerim merimo ali izbrana smer vodi v smeri dogovorjenih ciljev ali stran od njih.
3.) Nosilec odločanja mora pravočasno prejeti informacije, ki temeljijo na izbranih merah.

Cilje je potrebno definirati zelo natančno. V nasprotnem primeru nimamo od njih nobenih koristi. Cilj »veliko zadovoljstvo strank« se sliši izredno lepo, vendar ne pove nič o tem, kako vedeti kdaj je cilj dosežen oziroma ali se mu približujemo ali oddaljujemo.

Ko imamo cilje in načine za njihovo merjenje natančno definirane, moramo poskrbeti, da bodo meritve (informacije o odločitvah) pravočasno dostavljene nosilcu odločanja. Ločimo dva tipa informacij:

- **Temeljne informacije** so informacije, ki služijo kot osnova za sprejemanje določene odločitve med samim procesom odločanja.
- **Povratne informacije** so informacije, ki služijo za ovrednotenje učinkovitosti določene odločitve po tem, ko je bila odločitev že sprejeta.

Slika 1. Trije ključi za učinkovito odločanje.

2.2 Poslovna inteligenca - BI

[10] definira poslovno inteligenco kot: Poslovna inteligenca (BI) je zagotavljanje natančnih in uporabnih informacij ustreznim nosilcem odločanja v potrebnem časovnem okviru za podporo učinkovitemu odločanju.

BI je torej pojem, ki pokriva drugi in tretji ključ za učinkovito odločanje. Koncept, ki ga BI pokriva, se je začel pojavljati že v sedemdesetih letih 20. stoletja pod imenom EIS (executive information system – izvršilni informacijski sistem) in kasneje kot DSS (decision support system – sistem za podporo odločanju) [9]. Pojem BI (business intelligence – poslovna inteligenca) pa je postal popularen v poznih devetdesetih letih in se loči od prejšnjih predvsem v dveh lastnostih:

- BI zagotavlja informacije nosilcem odločanja na vseh ravneh odločanja, ne samo na najvišjih.
- Proces ustvarjanja poročil ni vezan na eno samo osebo, ampak si poročila lahko pripravi vsak udeleženec sam, oziroma je njihova priprava samodejna.

2.2.1 Načini zagotavljanja ustreznih informacij

Način, kako BI dostavi ustrezne informacije, je odvisen od tipa informacije, ki ga potrebujemo [10]. Včasih zelo natančno vemo, kje najdemo odgovor na naše vprašanje, včasih vemo samo približno, v določenih situacijah pa tega sploh ne vemo.

Kadar ne vemo, kateri podatek točno iščemo ter kako ga najti, nam BI omogoča uporabo podatkovnega rudarjenja. Podatkovno rudarjenje z uporabo zapletenih matematičnih algoritmov preuči podatke in v njih poišče vzorce, korelacije in povezave.

Kadar vemo, katera informacija nas zanima, uporabimo različne tipe poročil, kot so kontrolne plošče (angl. dashboards), produkcijska poročila (angl. production reports) in analitična poročila (angl. analytical reports) [9].

2.3 Podatkovno skladišče kot vir podatkov za BI

2.3.1 Transakcijski podatki

BI dobi podatke za analizo, oblikovanje ustreznih informacij in kazalcev iz transakcijskih podatkov organizacije. Transakcijski podatki so informacije, ki jih organizacije shranjujejo z namenom sledenja interakcijam oziroma poslovnim transakcijam, ki se izvršujejo znotraj organizacije [10]. Primeri transakcijskih podatkov: proizvedeni produkti, obdelana naročila, prejeta plačila, ...

Transakcijske podatke običajno shranjujemo v transakcijske podatkovne baze oziroma sisteme OLTP. Sistemi OLTP so namenjeni podpori vsakodnevnemu poslovanju in zato tudi optimizirani za tip operacij, ki se razlikuje od narave analitičnih operacij, ki se izvajajo v BI. Zaradi omenjenih slabosti uvedemo podatkovno skladišče kot vmesni člen med transakcijskimi podatki in BI.

2.3.2 Podatkovno skladišče

Podatkovno skladišče je predmetno usmerjena, integrirana, nespremenljiva in časovno različna zbirka podatkov, ki je namenjena podpori učinkovitemu odločanju [7].

Razlaga posameznih komponent definicije:

- **Predmetno usmerjena**: V središču podatkovnega skladišča so posamezni objekti (npr. stranke) in ne posamezne operacije.
- **Integrirana**: Podatki so konsistentno definirani. Ista entiteta je lahko v različnih transakcijskih sistemih različno definirana. V podatkovnem skladišču so entitete enotno definirane.
- **Nespremenljiva**: Podatki se v podatkovno skladišče shranjujejo v nekem rednem določenem časovnem intervalu (npr. vsako noč, vsak vikend, ...). Glede na konstantno shranjevanje in spreminjanje podatkov v sistemih OLTP je to velika razlika.
- **Časovno različna**: Podatkovno skladišče vsebuje podatke za daljše časovno obdobje, na primer za več preteklih let.

Podatkovno skladišče v določenih časovnih intervalih črpa podatke iz transakcijskih sistemov. Med prenosom podatke po potrebi preoblikuje in filtrira ter tako doseže stanje kot ga zahteva definicija podatkovnega skladišča. Podatkovno skladišče je visoko denormalizirana relacijska podatkovna baza, zato so določeni podatki v njem redundantni. S tem dosežemo, da je za poizvedbe nad podatki potrebno relativno malo število stikov in posledično so odzivni časi poizvedb kratki.

2.4 Sistemi OLAP

Definicija tehnologije OLAP [19]: Hitra analiza večdimenzijskih informacij v skupni rabi (angl. Fast Analysis of Shared Multidimensional Information; kratko: FASMI).

Sistemi OLAP omogočajo uporabnikom hitro in enostavno pridobivanje informacij iz podatkov, ki so običajno shranjeni v podatkovnem skladišču [10]. Sistemi OLAP predstavijo podatke z uporabo mer, dimenzij in kock.

2.4.1 Mera

V okviru poslovnih procesov se dogajajo posamezni dogodki [18]. Dogodki imajo pogosto številčne vrednosti, ki določajo njihovo obsežnost, na primer prodajna vrednost, prodajna količina. Te vrednosti imenujemo dejstva oziroma mere. S pomočjo mer vrednotimo odločanje in dosežke organizacije [10].

2.4.2 Dimenzija

Dimenzija je kategorizacija, ki jo uporabimo za razbitje mere na njene sestavne dele [10].

Dimenzije opisujejo poslovne objekte, ki sodelujejo pri poslovnih procesih [18]. Primeri dimenzij: stranka, produkt, poslovalnica. Objekt lahko obstaja tudi, če nikoli ni vključen v noben poslovni dogodek. Na primer: Obstaja lahko produkt, ki ga nikoli ne prodamo. Po drugi strani pa se poslovni proces ne more zgoditi brez dimenzij, ki so povezane z njim.

2.4.3 Kocka

Kocka je struktura, ki vsebuje vrednosti mer za vsako unikatno kombinacijo članov vseh njenih dimenzij [10]. Kocka vsebuje tudi seštete vrednosti, dobljene s hierarhijo dimenzij.

2.4.4 Prednosti sistemov OLAP

Sistemi OLAP temeljijo na uporabi večdimenzijskih podatkovnih baz [10]. Večdimenzijske podatkovne baze so najbolj naraven način shranjevanja podatkov, ki jih uporabljamo za BI, kjer analiziramo mere po dimenzijah.

Večdimenzijske podatkovne baze omogočajo tudi shranjevanje vnaprej seštetih določenih podatkov. Običajno želijo uporabniki videti vrednosti določene mere za izbrano množico članov neke dimenzije. Izračun te vrednosti je lahko časovno zelo potraten, zato sistemi OLAP določen delež vrednosti mer glede na različne kombinacije članov iz dimenzij izračunajo vnaprej in shranijo v kocko. Tako uporabniku ni potrebno čakati na izračun, ampak rezultat dobi zelo hitro.

Prednost sistemov OLAP pred sistemi OLTP je tudi v močni uporabi metapodatkov. Sistemi OLTP običajno uporabljajo imena polj in tabel, ki so prijazna do razvijalcev, ne pa do poslovnih uporabnikov. Sistemi OLAP po drugi strani nudijo veliko možnosti za definiranje imen, struktur ter izračunov, ki ustrezajo realni strukturi in izračunom organizacije.

2.5 Jezik MDX

MDX je jezik, s katerim izražamo izbire, izračunane vrednosti in določene metapodatke iz večdimenzijske podatkovne baze [22]. Omogoča tudi možnosti, s katerimi specificiramo način predstavitve rezultata poizvedbe. Rezultate poizvedbe MDX dobimo v podatkovni strukturi, ki jo je potrebno še dodatno obdelati, da bo izgledala kot preglednica, graf ali

kakšen drug način predstavitve podatkov. To zelo spominja na delovanje jezika SQL, ki se uporablja za poizvedovanje v relacijskih podatkovnih bazah.

2.5.1 Osnovna zgradba poizvedbe MDX

Primer enostavne poizvedbe MDX:

Imamo zelo preprosto kocko z imenom Prodaja, ki vsebuje meri prodajna vrednost in prodajna količina ter dimenziji datum in mesto. Dobiti želimo tabelo vrednosti, ki bo prikazovala prodajo v Ljubljani za avgust 2009 in avgust 2010.

Naslednja poizvedba MDX bi nam vrnila želen rezultat:

```
SELECT
{ [Measures].[Prodajna vrednost], [Measures].[Prodajna količina] }
on columns,
{ [Datum].[08-2009], [Datum].[08-2010] }
on rows
FROM [Prodaja]
WHERE { [Mesto].[Ljubljana] }
```

Slika 2 prikazuje rezultat poizvedbe MDX v obliki tabele.

	Prodajna vrednost	Prodajna količina
08-2009	96.949,10	3.866
08-2010	104.510,20	4.125

Slika 2. Rezultat enostavne poizvedbe MDX.

Kljub temu, da zgradba in ključne besede poizvedbe MDX močno spominjajo na jezik SQL, sta pomen in semantika različna.

2.6 Microsoft SQL Server Analysis Services 2005

Analysis Services 2005 je sestavni del Microsoftovega produkta SQL Server 2005. SQL Server 2005 poleg Analysis Services 2005 (kratko: SSAS) vsebuje tudi druge sestavne dele, kot so Integration Services (kratko: SSIS), Reporting Services (kratko: SSRS) in Notifications Services. SSAS, SSIS in SSRS tvorijo jedro platforme BI s strežnikom SQL Server v ozadju.

SSAS omogoča uporabo tehnologije OLAP (izdelava dimenzij, mer in kock) in vseh njenih dobrih lastnosti. Uporabniške prednosti produkta SSAS [18]:

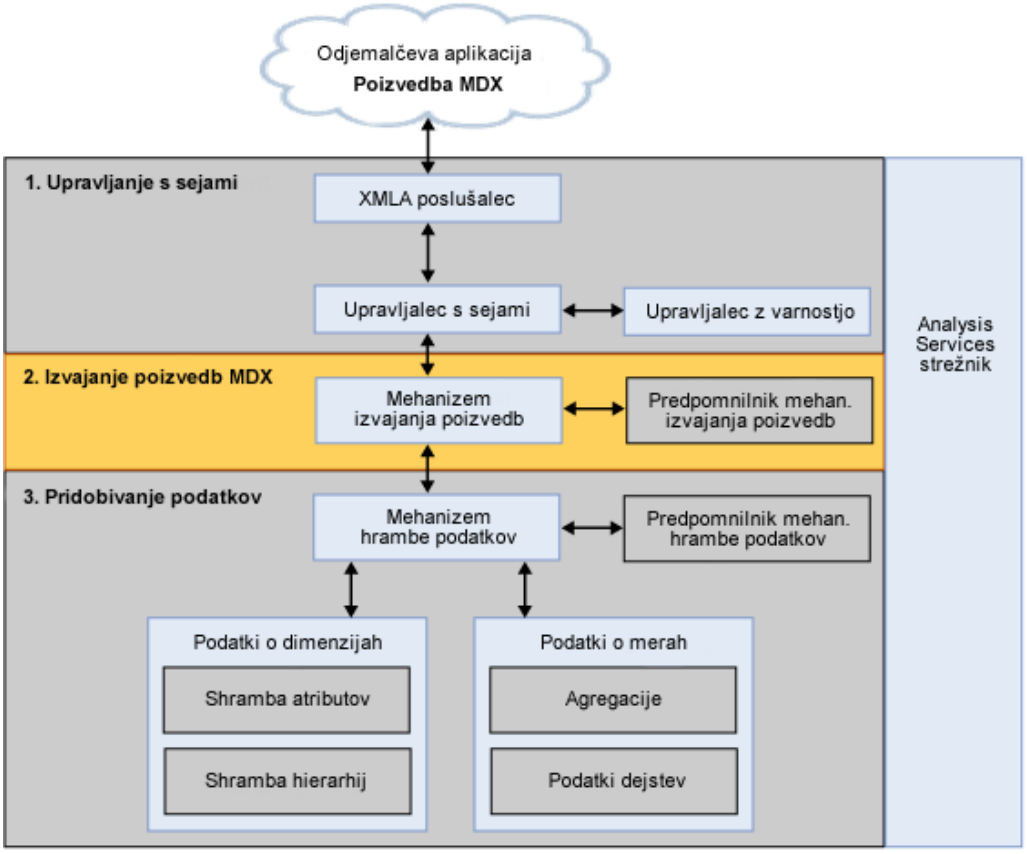
- **Upravljanje z agregiranimi podatki** (angl. aggregations): Kjerkoli je mogoče SSAS za odgovor na poizvedbe uporablja predhodno shranjene seštevke podatkov.
- **Formule in izračuni**: SSAS omogoča centralno definiranje formul nad podatki. S tem se izognemo problemom, ki lahko nastanejo, če vsak uporabnik sam definira formule. Če jih vsak uporabnik definira sam, se lahko zgodi, da jih vsi ne definirajo enako in pride do nesporazumov.

- **Logika za agregiranje mer**: Nekatere mere je potrebno agregirati po določenem pravilu. Na primer stanja zalog za določen mesec ne dobimo tako, da skupaj seštejemo stanja vseh dni v mesecu. SSAS nudi definiranje takih mer in njihovega načina agregiranja.

3 Arhitektura poizvedovanja in procesiranja v SSAS

3.1 Arhitektura poizvedovanja

Poizvedovanje je operacija, s katero SSAS aplikacijam posreduje želene podatke glede na zahteve podane v poizvedbi MDX [23]. Arhitektura poizvedovanja vsebuje različne komponente, ki med seboj učinkovito sodelujejo. Slika 3 (vir [23]) prikazuje glavne operacije in njihove sestavne dele, ki se izvedejo med poizvedovanjem.



Slika 3. Arhitektura poizvedovanja v SSAS.

3.1.1 Upravljanje s sejami

Odjemalci s SSAS komunicirajo preko protokola TCP IP ali HTTP z uporabo jezika XML for Analysis (XMLA[1]). Za vso komunikacijo na strani SSAS poskrbi *XMLA poslušalec* (angl. XMLA Listener).

Komponenta *upravljalec s sejami* (angl. Session Manager) nadzira način povezovanja odjemalcev na določeno instanco SSAS. Uporabnikom, ki imajo potrebne pravice, dovoli dostop do SSAS. Ko se uporabnik poveže, mu *upravljalec z varnostjo* (angl. Security Manager) glede na definirane vloge SSAS dodeli pravice, ki mu pripadajo.

[1] Več o XMLA: http://en.wikipedia.org/wiki/XML_for_Analysis

Običajno (odvisno od arhitekture odjemalca) ob zagonu aplikacije odjemalec ustvari sejo, ki jo nato uporablja za vse uporabnikove poizvedbe. Seja poskrbi za ustrezno okolje, v katerem *mehanizem izvajanja poizvedb* izvede poizvedbe. Seja obstaja dokler je odjemalec ali strežnik ne končata.

3.1.2 Izvajanje poizvedb MDX

Za izvajanje poizvedb MDX je zadolžen *mehanizem izvajanja poizvedb* (angl. Query Execution Engine). Tekom procesa izvajanja se izvede več korakov, vendar sta za potrebe optimizacije pomembna predvsem dva: pridobivanje podatkov in produciranje rezultatov.

1. **Pridobivanje podatkov:** *Mehanizem izvajanja poizvedb* razstavi vsako poizvedbo MDX na več zahtevkov za podatke. Te zahtevke nato pretvori v zahtevke za podkocke, ki jih razume *mehanizem hrambe podatkov*. Podkocke (angl. subcube) predstavljajo logične enote poizvedovanja, predpomnjenja in pridobivanja podatkov. Od vsebine in kompleksnosti poizvedbe je odvisno ali bo razčlenjena v en sam zahtevek za podkocko ali več. Smotrno je opozoriti, da je izraz podkocke, uporabljen na tem mestu, generičen pojem, ki ne pomeni iste podkocke, ki jo lahko ustvarimo z izrazom MDX CREATE SUBCUBE.
2. **Produciranje rezultatov:** *Mehanizem izvajanja poizvedb* uporablja dva možna načina izračuna rezultatov iz podatkov, ki jih je pridobil iz *mehanizma hrambe podatkov*. Izračuna lahko celotno podkocko naenkrat ali pa posebej izračuna vsako individualno celico. V splošnem je prvi način učinkovitejši, vendar se *mehanizem izvajanja poizvedb* vsakič sproti, glede na kompleksnost poizvedbe MDX, odloči za enega od načinov. Še več, za vsak posamezen del poizvedbe se neodvisno odloči kateri način izračuna je primernejši. Primer: če poizvedba zahteva poslovalnice, ki so letošnjo prodajo povečale za več kot 20% glede na lansko leto, potem lahko *mehanizem izvajanja poizvedb* izbere en način izračuna rezultatov za izračun povečanja prodaje letošnjega leta glede na lansko ter drug način za pridobitev tistih poslovalnic, ki imajo povečanje večje od 20%.
   Ko mehanizem izvajanja poizvedb izračuna rezultat, ga shrani v *predpomnilnik mehanizma izvajanja poizvedb* (angl. Query Execution Engine Cache) in v *predpomnilnik mehanizma hrambe podatkov* (Storage Engine Cache). Poglavitna prednost predpomnilnika je optimizacija računanja rezultatov ter podpora ponovni uporabi izračunanih rezultatov.

3.1.3 Pridobivanje podatkov o dimenzijah

V fazi pridobivanja podatkov mora *mehanizem hrambe podatkov* izbrati čim boljši način za izpolnitev zahtev po podatkih o dimenzijah in merah.

Podatke o dimenzijah dobi iz shramb atributov (angl. Attribute Store) in hierarhij (angl. Hierarchy Store). Pri tem uporablja dinamično predpomnjenje. To pomeni, da v pomnilnik ne shranjuje vseh članov dimenzij, ampak samo tiste, ki jih potrebuje. Podatkovne strukture, ki hranijo te podatke, so lahko shranjene na disku, v pomnilniku Analysis Services ali v predpomnilniku operacijskega sistema Windows. Dejanska lokacija je odvisna od zasedenosti sistemskega pomnilnika.

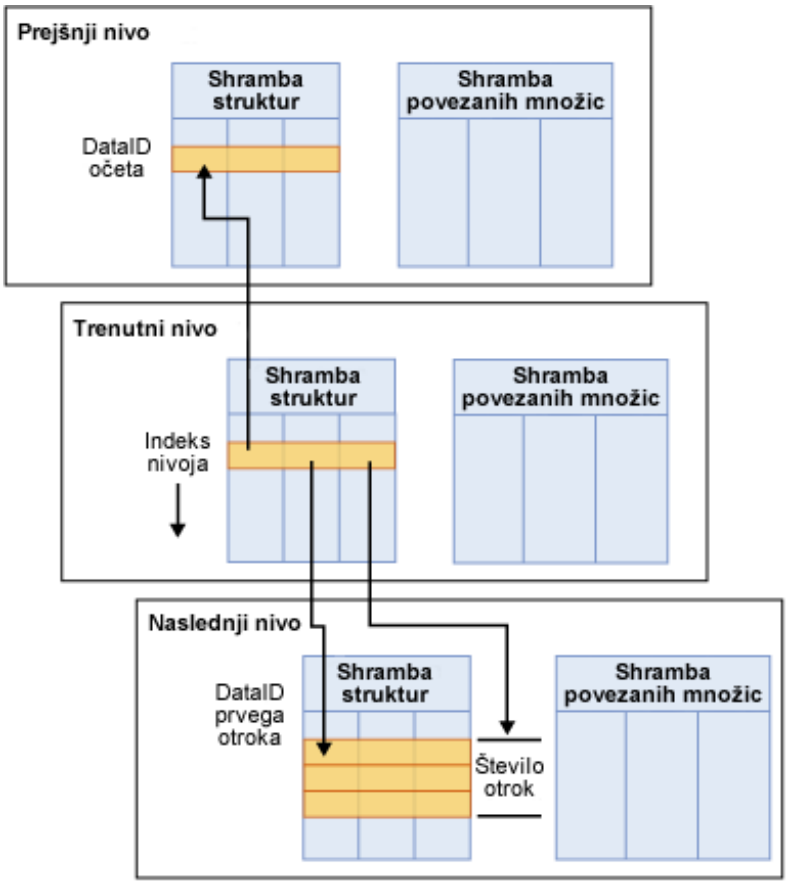
Shramba atributov dimenzij vsebuje vse informacije o atributih dimenzij. Njene komponente so prikazane na sliki 4 (vir [23]).

Slika 4. Shramba atributov dimenzij.

Shramba atributov dimenzij za vsak atribut v dimenziji hrani naslednje komponente:

- **Shramba ključev:** Vsebuje vrednosti ključev za vsakega člana atributa in njihove interne DataID-je. DataID je enolični identifikator, ki ga SSAS določi vsakemu članu atributa in se uporablja v vseh shrambah.
- **Shramba lastnosti:** Vsebuje različne lastnosti atributov, kot so na primer nazivi in prevodi, ki so povezane s posameznim DataID-jem. Zapisi v shrambi lastnosti so fizično urejeni po DataID-jih, s čimer je zagotovljen hiter naključni dostop do njihove vsebine.
- **Razpršene tabele:** Tekom procesiranja podatkov se ustvarita dve razpršeni tabeli (angl. hash table), ki ostaneta shranjeni na disku in omogočata lažje iskanje atributov med procesiranjem in poizvedovanjem. Zgoščena tabela ključev indeksira člane po njihovih enoličnih ključih. Zgoščena tabela nazivov indeksira člane po nazivih.
- **Shramba povezav:** Vsebuje povezave atributa z drugimi atributi. Shranjuje se DataID atributa ter DataID-je vseh povezanih atributov. Če nek atribut ne vsebuje povezave na noben drug atribut, potem se tega atributa ne shrani v shrambo povezav. Zapisi v shrambi povezav so podobno kot v shrambi lastnosti fizično urejeni po DataID-jih.
- **Bitni indeksi**: Tekom procesiranja podatkov se ustvarijo bitni indeksi, ki omogočajo lažje lociranje podatkov o povezavah med atributi. V primeru, da nek atribut vsebuje veliko število DataID-jev, lahko procesiranje bitnih indeksov traja relativno dolgo časa, vendar pohitritve pri poizvedovanju v večini primerov odtehtajo daljši čas procesiranja.

Poleg shrambe atributov se uporablja še shrambo hierarhij, ki atribute uredi v navigacijske poti namenjene končnim uporabnikom. Slika 5 (vir [23]) prikazuje shemo shrambe hierarhij.



Slika 5. Shramba hierarhij.

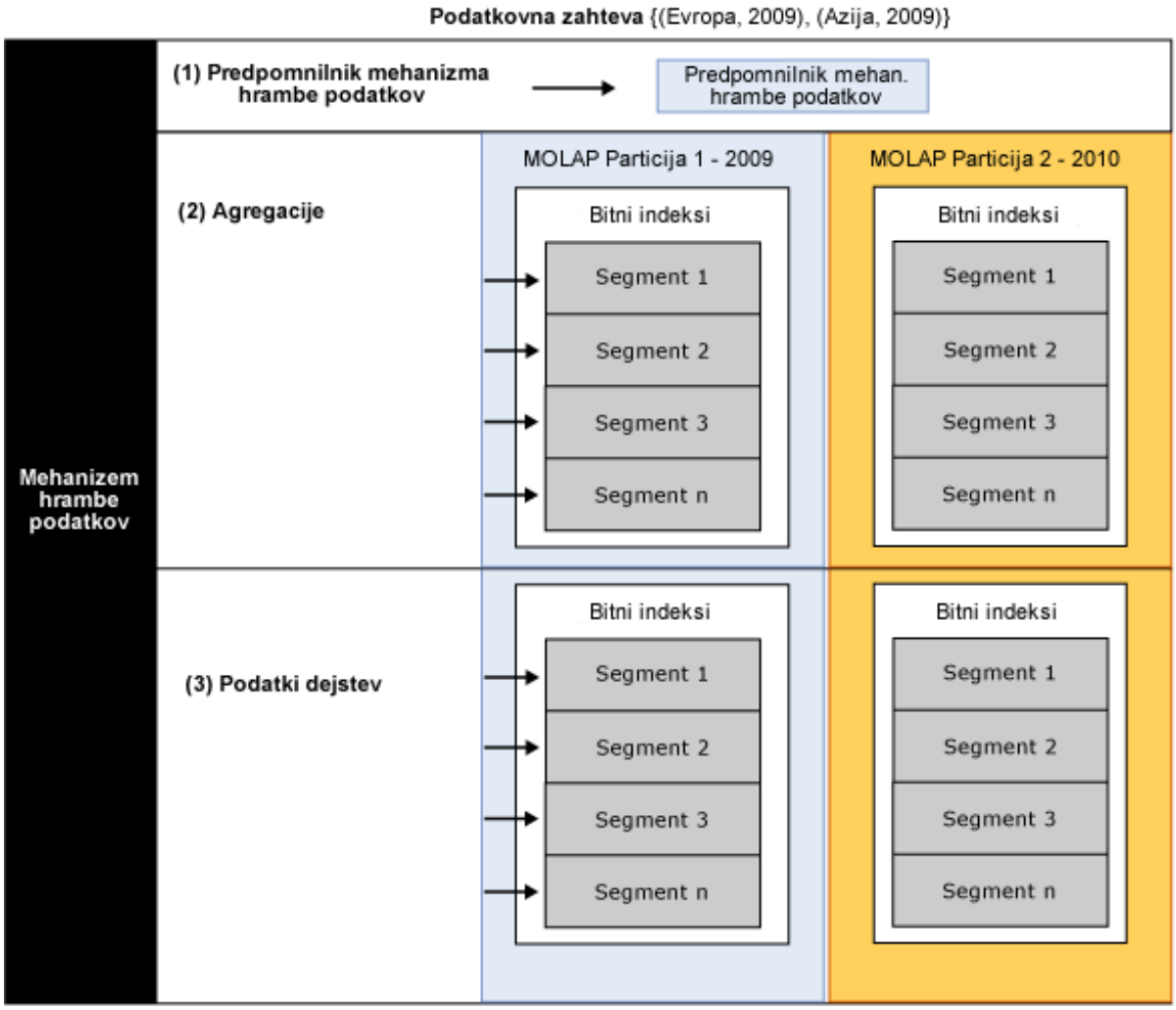
Shrambo hierarhij sestavljata dve glavni komponenti:

- **Shramba povezanih množic:** Za vsakega člana vsebuje iz DataID-jev sestavljeno pot od prvega do trenutnega nivoja.
  Primer: Pot All, Slovenija, Ljubljana, Poslovalnica 1100 bi bila shranjena kot 1,2,5,6, kjer predstavlja 1 DataID za All, 2 je DataID za Slovenija, 5 je DataID za Ljubljana in 6 je DataID za Poslovalnica 1100.
- **Shramba struktur:** Za vsakega člana nivoja vsebuje DataID očeta, DataID prvega otroka in število vseh otrok. Zapisi v shrambi struktur so urejeni po indeksu nivoja, ki ga vsebuje vsak član. Indeks nivoja nekega člana predstavlja njegovo pozicijo znotraj nivoja, kot je določena v nastavitvah dimenzije.
  Primer: Če Slovenija vsebuje 4 otroke, bi bil zapis za Slovenijo v shrambi struktur shranjen kot 1,5,4. 1 je DataID očeta (All), 5 je DataID Ljubljane, 4 je število vseh otrok Slovenije.

3.1.4 Pridobivanje podatkov o merah

Podatki o merah, ki jih mehanizem hrambe podatkov potrebuje, so fizično shranjeni v particijah (angl. partitions), ki vsebujejo podatke dejstev in agregacije. Vsaka particija lahko podatke hrani na drugačen način, ki ga specificiramo z načinom hrambe podatkov (angl. storage mode). S stališča hitrosti poizvedb je v večini primerov najboljša izbira način MOLAP.

Podatkovne strukture za hranjene podatkov dejstev in agregacij so v načinu MOLAP identične. Oboji so razdeljeni na segmente. Segment vsebuje fiksno število zapisov (64 KB), ki so razdeljeni v 256 strani po 256 zapisov. Vsak zapis hrani vse mere iz posamezne skupine mer (angl. measure group) in množico DataID-jev, ki se nanašajo na granulacijski atribut vsake dimenzije. V particiji so shranjeni samo zapisi, ki obstajajo v relacijski tabeli dejstev (angl. fact table), kar omogoča visoko kompresijo podatkov.



Slika 6. Pridobivanje podatkov o merah.

Slika 6 (vir [23]) prikazuje podatkovno zahtevo za {(Evropa, 2009), (Azija, 2009)}. Mehanizem hrambe podatkov ima za izpolnitev te zahteve na voljo naslednje pristope:

1. **Predpomnilnik mehanizma hrambe podatkov**: *Mehanizem hrambe podatkov* vedno poskusi zadostiti podatkovni zahtevi z uporabo predpomnilnika. Ta način pridobivanja podatkov je najhitrejši, ker je predpomnilnik vedno shranjen v hitrem pomnilniku.
2. **Agregacije** (angl. aggregations): Če se potrebni podatki ne nahajajo v predpomnilniku, mehanizem hrambe podatkov preveri, če jih je možno dobiti z uporabo vnaprej izračunanih agregiranih vrednosti. V idealnih primerih določena agregacija natančno ustreza podatku, ki ga potrebujemo. V nasprotnem primeru lahko želeno vrednost dobimo z vsoto več agregacij.

3. **Podatki dejstev** (angl. fact data): Če ustrezne agregacije ne obstajajo, mora mehanizem hrambe podatkov pridobiti podatke iz particije na disku oziroma iz predpomnilnika operacijskega sistema Windows. Pri tej operaciji se uporablja več optimizacijskih metod, na primer indeksiranje in razvrščanje povezanih zapisov v skupine.

Uporaba particij je pomembna optimizacijska metoda. Podatkovna zahteva na sliki 6 potrebuje samo podatke za leto 2009. Zato je mehanizmu hrambe podatkov potrebno preiskati samo particijo s podatki za leto 2009, ki je na sliki obarvana modro.

3.2 Arhitektura procesiranja

V splošnem je naloga procesiranja črpanje podatkov iz enega ali več podatkovnih virov v enega ali več objektov SSAS.

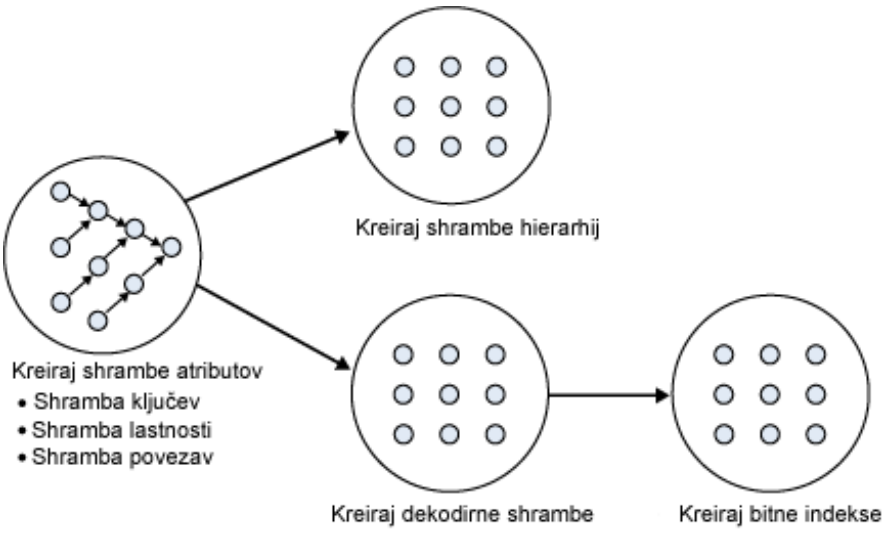
SSAS izvaja operacije procesiranja s pomočjo centralno nadzorovanih poslov. Posel procesiranja je generična enota dela, ustvarjena ob zahtevku za procesiranje. Razdelimo jih lahko na posle, ki so očetje in na posle, ki so otroci. Število in tip poslov sta odvisna od:

1) objekta, ki ga procesiramo (dimenzija, kocka, skupina mer, particija)
2) operacije, ki je bila zahtevana (ProcessFull, ProcessUpdate, ProcessIndexes)

Na primer, če je za neko skupino mer zahtevan ProcessFull, potem se za to skupino mer ustvari posel, ki je oče ter otroci za vsako particijo. Poleg tega med posli obstajajo tudi odvisnosti. Posli za procesiranje kocke so na primer odvisni od poslov za procesiranje dimenzij.

3.2.1 Posli pri procesiranju dimenzij

Med procesiranjem dimenzij MOLAP *mehanizem hrambe podatkov* ustvari različne shrambe, pri čemer uporablja množico poslov, ki so prikazani na sliki 7 (vir [23]).

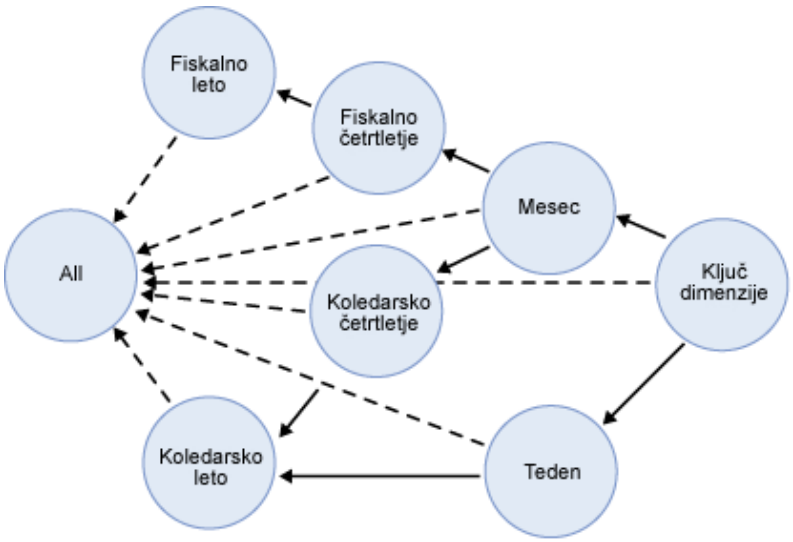


Slika 7. Posli pri procesiranju dimenzij.

Kreiranje shrambe atributov

Za vsak atribut v dimenziji se ustvari posel, ki izvleče in shrani člane atributa v shrambo atributov. Kot smo omenili že pri arhitekturi poizvedovanja, je shramba atributov sestavljena iz shrambe ključev, shrambe lastnosti in shrambe povezav. Čeprav ima SSAS možnost vzporednega procesiranja več atributov, mora poskrbeti za pravilni vrstni red operacij. Vrstni red določajo povezave med atributi, ki so shranjene v shrambi povezav. Če želimo pravilno kreirati shrambo povezav za določen atribut, morajo biti vsi odvisni atributi pred tem že sprocesirani. Na podlagi tega pravila mehanizem hrambe podatkov analizira povezave med atributi dimenzije, najde odvisnosti med njimi in zgradi izvajalno drevo. Izvajalno drevo označuje v kakšnem vrstnem redu se sme procesirati atribute ter katere atribute lahko procesiramo vzporedno.

Slika 8 (vir [23]) prikazuje izvajalno drevo za primer časovne dimenzije. Neprekinjene puščice predstavljajo definirane povezave med atributi. Črtkane puščice predstavljajo implicitne povezave vseh atributov z atributom All.



Slika 8. Primer izvajalnega drevesa.

V primeru iz slike 8 se bo najprej procesiral atribut All, ker ni odvisen od nobenega drugega atributa. Sledila bosta atributa Fiskalno leto in Koledarsko leto, ki se ju lahko procesira vzporedno. Glede na odvisnosti si nato sledijo še vsi ostali atributi. Atribut, ki predstavlja ključ dimenzije, se vedno procesira zadnji.

Čas, potreben za procesiranje posameznega atributa, je odvisen od števila članov tega atributa in števila povezav tega atributa.

Kreiranje dekodirne shrambe

Mehanizem hrambe podatkov uporablja dekodirne shrambe za pridobivanje podatkov pri poizvedovanju in za gradnjo bitnih indeksov pri procesiranju.

Kreiranje shrambe hierarhij

Za vsako naravno hierarhijo v dimenziji se ustvari posel za kreiranje shrambe hierarhij.

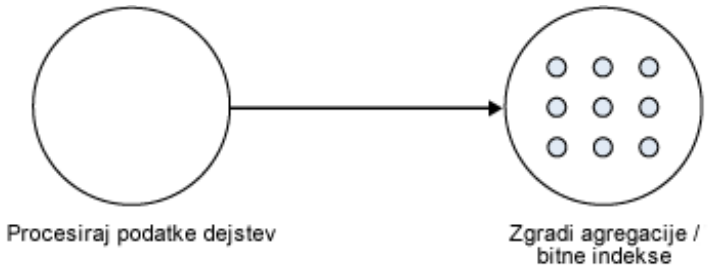
Kreiranje bitnih indeksov
Mehanizem hrambe podatkov med procesiranjem ustvari bitne indekse, ki se kasneje uporabljajo za hitrejše poizvedovanje. Pri atributih z velikim številom DataID-jev se lahko zgodi, da procesiranje bitnih indeksov traja relativno dolgo časa.

3.2.2 Ukazi za procesiranje dimenzij
Kadar želimo procesirati dimenzijo, uporabimo ukaze za procesiranje dimenzij. Vsak ukaz ustvari enega ali več poslov.

Najpomembnejši ukazi za procesiranje dimenzij so:

- **ProcessFull**: Ukaz zavrže vso shranjeno vsebino dimenzije in jo ponovno izgradi. Poleg tega zavrže tudi shranjeno vsebino vseh odvisnih particij, ki jih je zato tudi potrebno ponovno procesirati.
- **ProcessData**: Ukaz zavrže vso shranjeno vsebino dimenzije in ponovno izgradi samo shrambe atributov in shrambe hierarhij. Zavrže tudi vsebino odvisnih particij.
- **ProcessIndexes**: Ukaz ponovno izgradi bitne indekse. Pogoj za izvedbo so že zgrajene shrambe atributov in hierarhij.
- **ProcessUpdate**: Ukaz ne zavrže shranjene vsebine dimenzije, ampak jo samo inteligentno osveži, zato da se ohrani vsebina odvisnih particij. Morebitne agregacije zavrže.
- **ProcessAdd**: Ukaz ne izbriše ali osveži obstoječih podatkov, temveč samo doda nove člane dimenzije. Agregacij ne zavrže.

3.2.3 Posli pri procesiranju particij
Procesiranje particij pridobi podatke iz podatkovnih virov in jih shrani na disk. Pri tem se uporablja različne posle, ki so prikazani na sliki 9 (vir [23]).



Slika 9. Posli pri procesiranju particij.

Procesiranje podatkov dejstev
Podatki dejstev se procesirajo z uporabo treh sočasnih niti, ki izvedejo naslednje naloge:

- Podatkovnim virom pošljejo poizvedbe SQL, s katerim pridobijo podatke.
- V shrambah dimenzij poiščejo ključe dimenzij in napolnijo procesirni medpomnilnik (angl. processing buffer).
- Ko je medpomnilnik poln, zapišejo njegovo vsebino na disk.

Kreiranje agregacij in bitnih indeksov

Med procesiranjem se v pomnilniku zgradi agregacije. (Pre)veliko število agregacij lahko močno poveča čas procesiranja.

Na podlagi podatkov dejstev in agregacij se zgradi bitne indekse. Nato se jih po segmentih zapiše na disk.

3.2.4 Ukazi za procesiranje particij

Kadar želimo procesirati particijo, uporabimo ukaze za procesiranje particij. Vsak ukaz ustvari enega ali več poslov.

Najpomembnejši ukazi za procesiranje particij so:

- **ProcessFull**: Ukaz zavrže shranjeno vsebino particij in jo ponovno izgradi.
- **ProcessData**: Ukaz zavrže shranjeno vsebino particij in ponovno izgradi samo podatke dejstev.
- **ProcessIndexes**: Ukaz ustvari manjkajoče agregacije in bitne indekse, obstoječe pa obdrži. Pogoj za izvedbo so že procesirani podatki dejstev.
- **ProcessAdd**: Ukaz interno ustvari začasno particijo, jo sprocesira s ciljnimi podatki dejstev in združi z obstoječo particijo. V Microsoftovih orodjih je ta ukaz na voljo pod imenom ProcessIncremental.

3.2.5 Izvajanje poslov procesiranja

Glede na odvisnosti med posli SSAS iz njih sestavi urnik procesiranja. Na primer dimenzije morajo biti vedno procesirane pred particijami, ki so odvisne od njih. Posli brez odvisnosti se lahko izvajajo vzporedno. Na primer, vzporedno se lahko procesira več dimenzij, več skupin mer in več particij. Glede na količino virov, ki jih ima na voljo in glede na nastavitve strežnika poskuša SSAS vzporedno izvajati čim več operacij.

4 Načini optimizacije sistemov OLAP

Že v 1. poglavju sem omenil, da se zaradi kompleksnosti sistemov OLAP, njihove optimizacije lotimo na več nivojih. Količini, ki ju želimo optimizirati, sta čas izvajanja poizvedb MDX in čas procesiranja podatkov. Obe količini želimo čim bolj zmanjšati. V praksi to pomeni, da rezultat poizvedbe MDX dobimo v čim krajšem času in da se podatki iz podatkovnega skladišča prenesejo in sprocesirajo v čim krajšem času.

Medtem ko se podatki procesirajo, je kocka odklopljena (angl. offline) in na njej ne moremo izvajati poizvedb. Poleg tega je strežnik takrat zelo obremenjen. Zato želimo čas procesiranja čim bolj skrajšati.

Čas, potreben za posredovanje rezultata poizvedbe MDX uporabniku, je še pomembnejši od časa procesiranja. Uporabnikom v organizacijah je čas običajno zelo dragocen, zato jim moramo potrebne podatke servirati čim hitreje. Ne samo, da prepočasne poizvedbe uporabnikom »kradejo« čas, ampak jih spravljajo tudi v slabo voljo in s tem dodatno zmanjšujejo njihovo delovno učinkovitost.

V tem poglavju si bomo ogledali možne načine optimizacije sistemov OLAP na posameznih nivojih. Pri tem nam bo glavni cilj zmanjšati čas izvajanja poizvedb, stranski pa zmanjšati čas procesiranja.

4.1 Podatkovno skladišče

4.1.1 Stolpci in vrstice

Števila shrani kot števila
Če za določen stolpec v tabeli podatkovnega skladišča zagotovo vemo, da bo vseboval številčne vrednosti, potem ga shranimo kot številčni namesto tekstovni podatkovni tip [12]. S tem poskrbimo za učinkovitejšo rabo prostora in učinkovitejše procesiranje.

Uporabljaj čim manjše podatkovne tipe
Pri gradnji logičnega modela podatkovnega skladišča vedno izberimo najmanjši podatkovni tip, ki ga podatki še dovoljujejo [18].

V primeru celih števil izberemo najprimernejšega izmed različnih tipov podatkovnega tipa INTEGER:

- Tinyint: vsebuje števila od 0 do 255 in zavzame 1 bajt prostora.
- Smallint: vsebuje števila od -2^{15} (-32768) do $2^{15}-1$ (32767) in zavzame 2 bajta prostora.
- Int: vsebuje števila od -2^{31} do $2^{31}-1$ in zavzame 4 bajte prostora.
- Bigint: vsebuje števila od -2^{63} do $2^{63}-1$ in zavzame 8 bajtov prostora.

Posebno pozornost velja posvetiti tekstovnim podatkovnim tipom. Marsikdo jim preprosto nastavi podatkovni tip VARCHAR ter dolžino 100 znakov, ker ve, da bo to zadostovalo vsem vnosom v bazi. Medtem ko v relacijski bazi to še ni pravi problem, se problem pojavi v

SSAS. SSAS poljem, ki imajo definirano dolžino 100 znakov, doda toliko presledkov, da zapolni prostor teh 100 znakov. Zato se moramo o podatkih, ki jih shranjujemo, poučiti in nastaviti dolžino tekstovnih polj na čim manjšo, vendar še vedno dovolj veliko, da bo zadostovala vsem trenutnim in prihodnjim vnosom.

Če ima nek stolpec dolžino manjšo od pet znakov, ga je bolje definirati kot podatkovni tip CHAR namesto VARCHAR. Isto velja za polja, katerih dolžina variira samo za en, dva ali tri znake.

Izogibati se moramo uporabi podatkovnih tipov TEXT, NCHAR in NVARCHAR. Zadnja dva sta identična tipom CHAR in VARCHAR, le da sta namenjena shranjevanju Unicode znakov. Vsem trem omenjenim podatkovnim tipom je skupno, da za shranjevanje uporabljajo veliko prostora.

Odstrani odvečne zapise
V podatkovnem skladišču se lahko nahajajo zapisi, ki na analiziranje podatkov na noben način ne vplivajo. Na primer v datumski dimenziji potrebujemo samo tista leta, za katera imamo podatke dejstev. Leta 1985 ali 2050 tako najverjetneje ne potrebujemo. Take vnose iz podatkovnega skladišča izbrišemo.

4.1.2 Ključi

Definiraj primarne in tuje ključe
Tabelam dejstev in dimenzijskim tabelam določi primarne ključe [2]. Glede na povezave med tabelami jim določi tudi tuje ključe. S tem pohitrimo poizvedbe nad relacijskimi podatki.

Ključi naj bodo tipa INTEGER
Vsi definirani primarni ključi naj bodo celoštevilski [11, 15, 18]. Glavne prednosti v primerjavi s tekstovnimi ključi:

- učinkovitejša poraba prostora
- učinkovitejša primerjava vrednosti med seboj
- učinkovitejši stiki med tabelami pri procesiranju

Če so vrednosti ključa neke tabele tekstovne, potem uporabimo nadomestni ključ (angl. surrogate key). To je stolpec tipa INTEGER, ki ga tabeli umetno dodamo in za vsak zapis vsebuje enolično vrednost, zato ga lahko uporabimo kot ključ te tabele.

4.1.3 Indeksi

Postavi čim več indeksov
Ker se v podatkovnem skladišču operaciji pisanja in spreminjanja izvajata zelo redko, lahko v njem brez strahu postavimo veliko število indeksov [11, 12, 18]. Z njihovo postavitvijo lahko vplivamo predvsem na hitrejše procesiranje podatkov v SSAS.

Predlog strategije začetne postavitve indeksov:

- <u>Dimenzijske tabele</u>:

V dimenzijskih tabelah postavimo indeks gruče (angl. clustered index) na ključ tabele. Če poleg primarnega ključa obstaja tudi stolpec, ki predstavlja naravni ključ tabele, postavimo indeks tudi na njem. Indekse postavimo na tuje ključe. V večjih dimenzijah lahko poskusimo s postavitvijo indeksov na stolpce, ki se uporabljajo v hierarhijah podatkov dimenzije.

- Tabele dejstev:
  V tabelah dejstev postavimo indeks gruče na stolpcu, ki vsebuje datum. Če imamo več datumov, izberemo tistega, kateri se največ uporablja oziroma tistega, po katerem podatke particioniramo. Indekse postavimo tudi na tuje ključe.

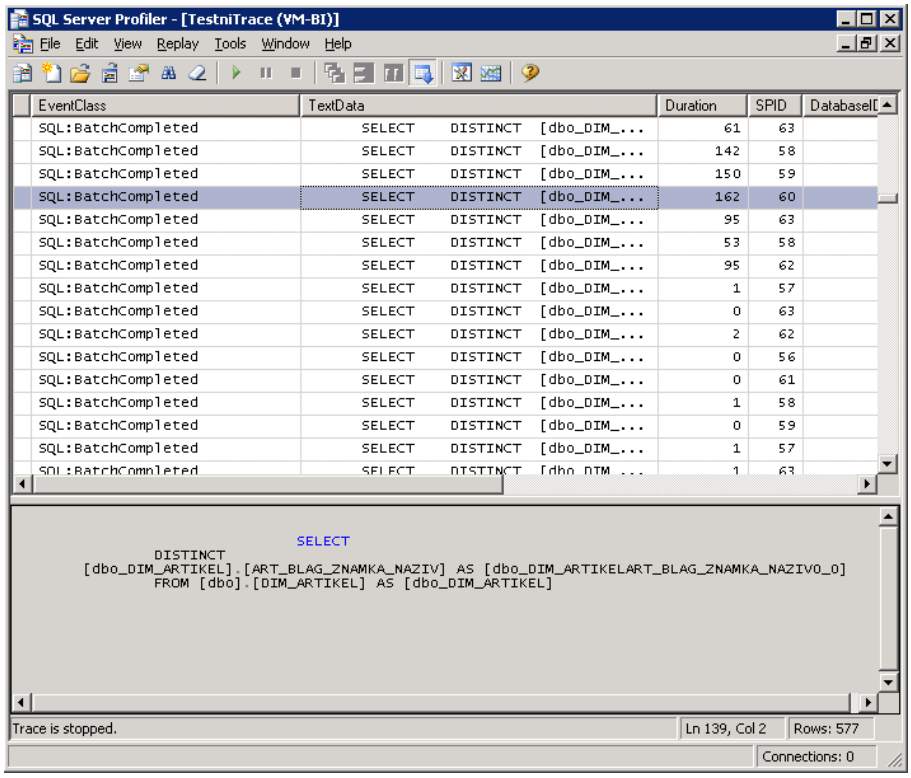
Nastavi nastavitev »Fill factor« na 100

Pri kreiranju indeksov naj bo nastavitev Fill factor nastavljena na 100 [12]. S tem zagotovimo, da bodo indeksne strani polne, kolikor se bo le dalo in pridobimo na zmogljivosti.

Uporabi namenska orodja za optimizacijo indeksov

S pomočjo orodij SQL Server Profiler[2] in Database Engine Tuning Advisor[3], ki sta del paketa SQL Server 2005, lahko uporabo indeksov dodatno optimiziramo [12].

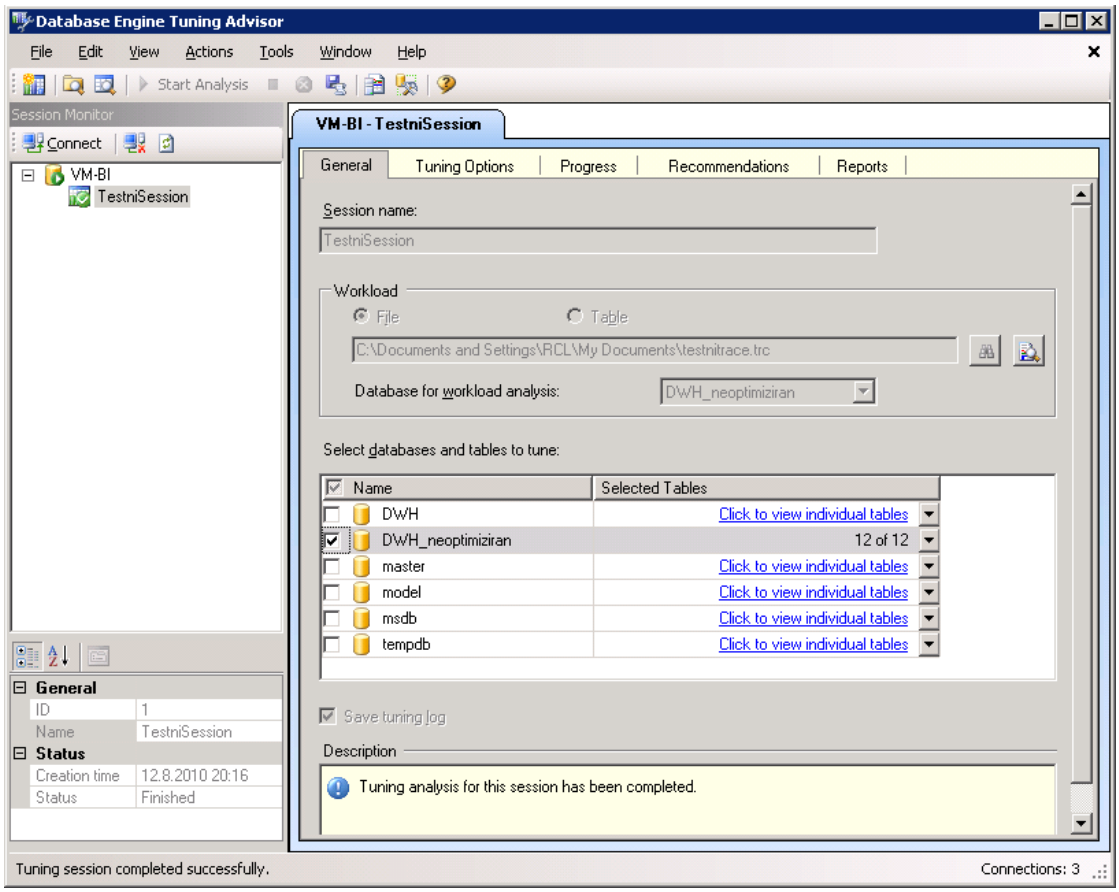
SQL Server Profiler uporabimo za izdelavo datoteke, v katero se shranijo vse poizvedbe, ki se izvedejo med procesiranjem. Slika 10 prikazuje zaslonski posnetek SQL Server Profiler-ja po tem, ko se je procesiranje že izvedlo in je zabeležil vse dogodke, ki so se medtem izvršili na podatkovnem skladišču.



Slika 10. SQL Server Profiler.

[2] Več o programu in njegovi uporabi: http://msdn.microsoft.com/en-us/library/ms181091.aspx
[3] Več o programu in njegovi uporabi: http://msdn.microsoft.com/en-us/library/ms166575.aspx

Database Engine Tuning Advisor nato analizira datoteko z dogodki, ki smo jo ustvarili s programom SQL Server Profiler. Kot rezultat analize nam predlaga postavitev dodatnih indeksov ali odstranitev odvečnih. Slika 11 prikazuje zaslonski posnetek prve strani Database Engine Tuning Advisor-ja, kjer nastavimo splošne nastavitve analize.



Slika 11. Database Engine Tuning Advisor.

V dodatku A je uporaba programov SQL Server Profiler in Database Engine Tuning Advisor obširneje predstavljena.

Indekse pred uvozom novih podatkov izbriši

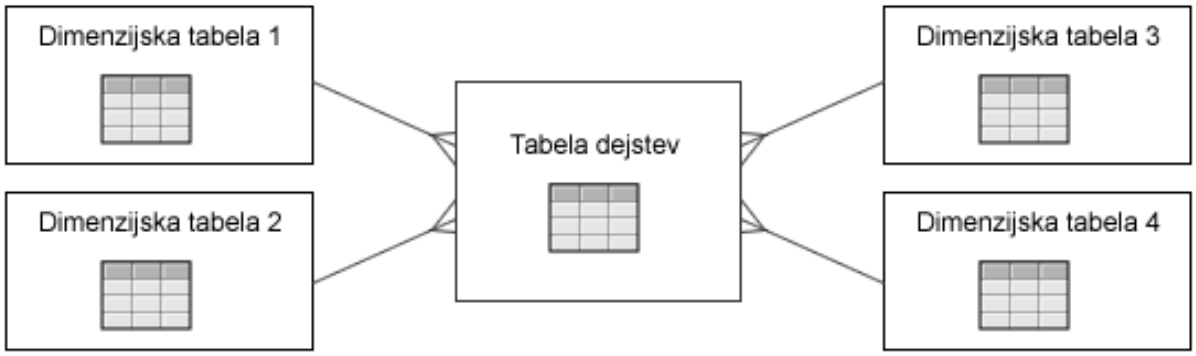
[12] Zaradi velikega števila indeksov je lahko vstavljanje novih podatkov v podatkovno skladišče zelo počasno. Splača se preizkusiti naslednji scenarij: najprej indekse iz podatkovnega skladišča izbrišeš, nato uvoziš podatke in nazadnje indekse ponovno kreiraš. Ta scenarij se lahko izkaže za hitrejšega od vstavljanja podatkov brez brisanja in kreiranja indeksov. Pridobljen čas je zelo odvisen od konkretne situacije, zato je najbolje testirati oba scenarija in se odločiti na podlagi rezultatov.

4.1.4 Tabele in podatkovna baza

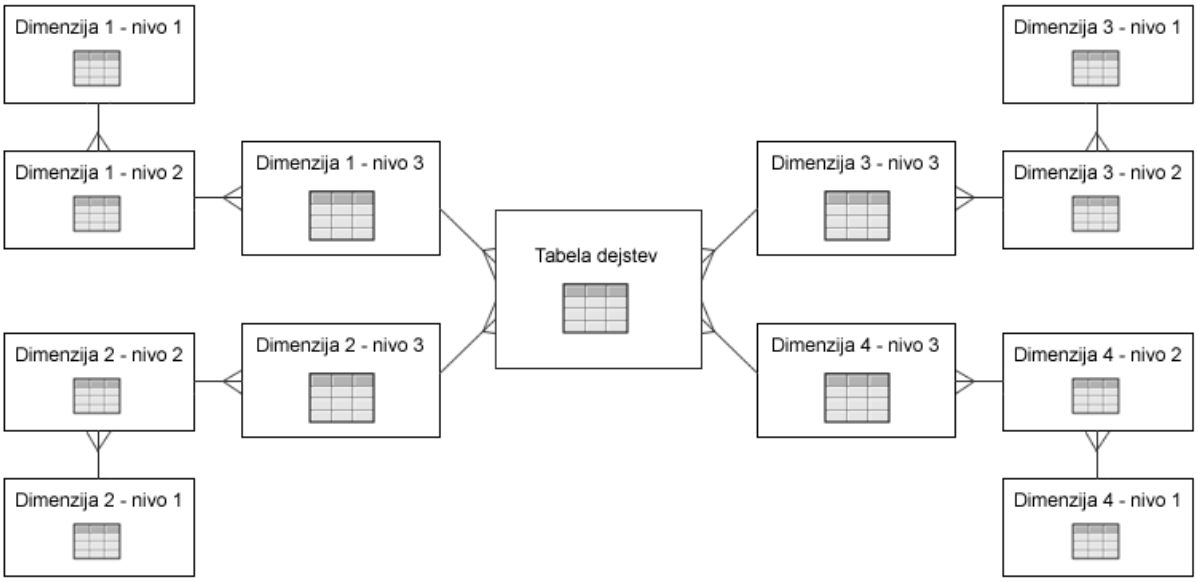
Zvezdasta in snežinkasta shema podatkovne baze

Podatkovno skladišče lahko shranjuje mere in dimenzije v dveh različnih shematičnih oblikah, in sicer snežinkasti (angl. snowflake) ali zvezdasti (angl. star) [10]. Shemi sta dobili imeni po obliki njunih relacijskih diagramov.

V obeh shemah je v središču tabela dejstev, ki se preko tujih ključev povezuje na dimenzijske tabele. Razlika med shemama je ravno v dimenzijskih tabelah. Pri zvezdasti shemi vsako dimenzijo tvori natanko ena (pogosto denormalizirana) tabela. Pri snežinkasti shemi pa so dimenzije, ki vsebujejo hierarhije, razbite na več tabel, tako da za vsak nivo hierarhije ustvarimo svojo tabelo. Sliki 12 in 13 prikazujeta shematičen prikaz relacijskega diagrama obeh shem.



Slika 12. Relacijski diagram zvezdaste sheme.



Slika 13. Relacijski diagram snežinkaste sheme.

Snežinkasta zasnova je s stališča relacijskih podatkovnih baz boljša od zvezdaste. V njej se podatki ne podvajajo in posledično jo je lažje vzdrževati. Po drugi strani pri agregiranju mer zahteva več stikov med tabelami kot zvezdasta zasnova, kar lahko vodi v slabše zmogljivosti. Teorija zato za potrebe podatkovnih skladišč priporoča uporabo zvezdaste sheme.

Nastavi možnost »read-only«

Če na podatkovnem skladišču nastavimo možnost Read-only na TRUE, na podatkovni bazi izklopimo zaklepanje, kar lahko pohitri predvsem fazo procesiranja kocke SSAS [12].

4.2 Zasnova podatkovne kocke

4.2.1 Dimenzije

Izbriši nepotrebne atribute
Čeprav čarovnik v orodju BIDS[4] privzeto ustvari atribut za vsak stolpec v dimenzijski tabeli, je iz stališča optimizacije bolje izbrisati atribute, ki jih ne potrebujemo [1]. Priporočeno je najprej ustvariti samo najnujnejše atribute in kasneje postopoma dodajati tiste, ki manjkajo.

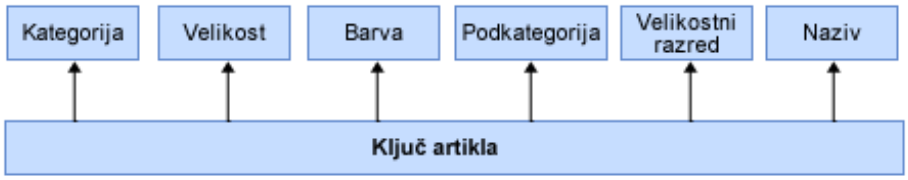
Uporabljaj numerične ključe
[23] Vsak atribut določata dve lastnosti: KeyColumn in NameColumn. KeyColumn predstavlja ključ atributa in določa stolpec iz podatkovnega skladišča, ki je enolični identifikator vsake instance tega atributa. NameColumn določa stolpec, ki bo prikazan končnemu uporabniku. Če NameColumn ni določen, se za prikaz uporabi kar KeyColumn.

Ključi atributov naj bodo numerični in enostavni (sestavljeni iz enega samega stolpca) namesto tekstovnih ali sestavljenih. Logika, ki stoji za tem, je podobna logiki pri uporabi numeričnih ali nadomestnih ključev v relacijski podatkovni bazi.

Z uporabo omenjenih lastnosti lahko združimo tekstovne in numerične atribute, ki določajo isto stvar. Numerični atribut uporabimo za ključ, tekstovnega pa za naziv atributa, ki bo končnemu uporabniku dobro razumljiv.

Določi razmerja med atributi
Tako kot v relacijskih podatkovnih bazah je tudi v SSAS pomembno poznati in definirati odvisnosti med atributi [23]. S tem izboljšamo izdelavo agregacij in učinkovitost hranjenja ter pridobivanja podatkov.

Odvisnosti med atributi se definira z uporabo razmerij (angl. attribute relationship). Razmerje je lahko tipa *ena proti ena* (angl. one-to-one) ali *ena proti mnogo* (one-to-many). Ko z orodjem BIDS dimenzijo ustvarimo, se samodejno kreirajo razmerja ena proti mnogo med atributom, ki je ključ dimenzije, in vsemi ostalimi atributi. Primer je prikazan na sliki 14, kjer so vsi atributi s ključem v razmerju ena proti mnogo.

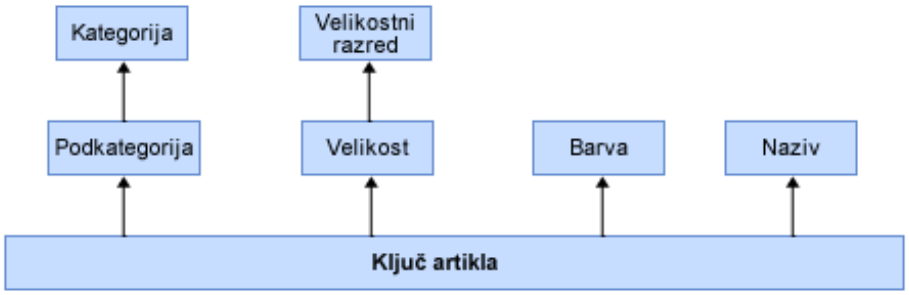


Slika 14. Privzeta razmerja med atributi.

Taka zasnova ni optimalna, zato je potrebno razumeti, kako so atributi povezani med seboj in to z uporabo razmerij tudi določiti. Primer, ko se pokaže slabost privzete zasnove: Recimo, da imamo že izračunane vsote posameznih podkategorij in želimo izvedeti še vsoto posameznih

[4] Več o programu BIDS: http://msdn.microsoft.com/en-us/library/ms173767.aspx

kategorij artiklov. Ker SSAS ne ve, da med kategorijami in podkategorijami obstaja odvisnost, mora sešteti vrednosti vseh posameznih artiklov znotraj vsake kategorije. Če bi imeli definirana razmerja kot na sliki 15, bi bilo potrebno za dosego istega rezultata sešteti samo vsote posameznih podkategorij artiklov, kar bi nedvomno trajalo manj časa.



Slika 15. Optimizirana razmerja med atributi.

Razmerjem določi tip

[23] Vsako razmerje je določenega tipa (angl. relationship type), in sicer *fleksibilno* (angl. flexible) ali *togo* (angl. rigid). Privzeto so vsa ustvarjena razmerja fleksibilna. Če želimo optimizirati procesiranje sprememb v podatkih, moramo razmerjem nastaviti pravilen tip.

Razmerjem med atributi, za katera pričakujemo, da se v prihodnosti ne bodo spreminjala (na primer mesec januar 2010 bo vedno pripadal letu 2010), nastavimo tog tip razmerja. Razmerjem, za katera obstaja možnost, da se bodo v prihodnosti spreminjala (na primer zakonski stan zaposlenih), nastavimo fleksibilen tip razmerja.

V primeru osvežitve ali izbrisa podatkov iz atributov, ki so fleksibilno povezani s ključem, ni potrebno ponovno procesirati celotne kocke. V primeru togih razmerij je to potrebno. Po drugi strani se agregacije, sestavljene iz atributov, ki so togo povezani s ključem, pri ukazu ProcessUpdate ohranijo, v primeru fleksibilnih razmerij pa ne.

Iz povezanih atributov sestavi naravne hierarhije

[23] V SSAS lahko končnemu uporabniku prikažemo atribute z uporabo dveh tipov hierarhij: *hierarhije atributov* (angl. attribute hierarchy) in *uporabniške hierarhije* (angl. user hierarchy). Hierarhije atributov so hierarhije, ki se privzeto ustvarijo za vsak atribut in so sestavljene iz dveh nivojev – iz nivoja All in nivoja samega atributa.

V uporabniških hierarhijah so atributi sestavljeni v vnaprej definirane večnivojske navigacijske poti, ki končnim uporabnikom olajšujejo analizo. Ločimo dva tipa uporabniških hierarhij: *naravne* in *nenaravne*. V naravni hierarhiji so vsi atributi, ki nastopajo na posameznih nivojih, od najnižjega do najvišjega, med seboj povezani z razmerji. Običajno naravne hierarhije sledijo verigi razmerij ena proti mnogo, ki »naravno« obstajajo v podatkih. V nenaravni hierarhiji je hierarhija sestavljena iz najmanj dveh nivojev, med katerima ne obstaja povezava v obliki razmerja med atributi.

Iz stališča zmogljivosti so naravne hierarhije veliko boljša izbira, zato jih je dobro kreirati čim več. Navigacijske poti, ki sestavljajo naravne hierarhije, so materializirane na disku v shrambi

hierarhij. Atributi, ki sestavljajo naravne hierarhije, so samodejno obravnavani kot kandidati za agregacije. Za nenaravne hierarhije nič od naštetega ne drži. Niso niti materializirane niti privzeto se jih ne obravnava kot kandidate za agregacije.

Odstrani ponavljajoča razmerja

Vrnimo se nazaj na primer iz Slika 15. Atribut Kategorija ne potrebuje več neposredne povezave s ključem artikla, ker je preko Podkategorije z njim še vedno povezan. Če bi dodatno ustvarili še omenjeno razmerje, bi šlo za ponavljajoče razmerje, ki ima negativen vpliv na zmogljivost. Na to nas s prikazom posebne ikone opozori že orodje BIDS. Zato se ponavljajočim razmerjem izogibamo oziroma jih odstranimo [23].

Izogibaj se diamantno oblikovanim razmerjem

Diamantno oblikovana razmerja se nanašajo na verigo razmerij, ki se loči in kasneje ponovno združi, vendar ne vsebuje ponavljajočih razmerij [1]. Primer bi bila razmerja Dan →Mesec → Leto in Dan → Četrtletje → Leto. Diamantno oblikovana razmerja na strežniku povzročajo dvoumnost, zato se jim raje izogibamo.

Uporaba nastavitve AttributeHierarchyVisible

Atribute, ki so vključeni v uporabniške hierarhije, pogosto ne potrebujemo prikazati tudi v njegovi lastni enonivojski hierarhiji, ker bi s tem končnega uporabnika po nepotrebnem zmedli [1]. Prikaz enonivojske hierarhije atributa onemogočimo tako, da atributu nastavitev AttributeHierarchyVisible nastavimo na False.

Uporaba nastavitve AttributeHierarchyEnabled

Informacije o dimenziji lahko poleg hierarhij prikažemo tudi v obliki *lastnosti članov* (angl. member property) [23]. Atribut se obnaša kot lastnost članov, če ima onemogočen prikaz v obliki hierarhije in je z nekim drugim atributom v razmerju. Prikaz v obliki hierarhije onemogočimo tako, da atributu nastavitev AttributeHierarchyEnabled nastavimo na False. Tak atribut ne bo več indeksiran in agregiran, kar lahko pohitri čas procesiranja in zmanjša velikost kocke. Po drugi strani se lahko podaljša čas poizvedovanja in izgubimo možnost rezanja in agregiranja na nivoju tega atributa. Zato moramo biti pri določanju atributov kot lastnost članov previdni. Najpogostejši kandidati za lastnost članov so atributi, ki so s ključem v razmerju ena proti ena (na primer telefonska številka in naslov).

Uporaba nastavitve AttributeHierarchyOptimizedState

Določenim atributom lahko nastavitev AttributeHierarchyOptimizedState nastavimo na *Not Optimized* in s tem onemogočimo kreiranje bitnih indeksov za ta atribut [23]. Posledično se čas procesiranja skrajša in čas poizvedovanja podaljša. Za ta korak se odločimo pri zelo redko uporabljanih atributih ali pri atributih z veliko močjo množice. Pri atributih z veliko močjo množice z uporabo bitnih indeksov skoraj nič ne pridobimo, ker mora mehanizem hrambe podatkov vseeno obdelati veliko količino različnih vrednosti.

Uporaba nastavitve InstanceSelection

Odjemalci se lahko s pomočjo nastavitve InstanceSelection odločijo, na kakšen način bodo končnemu uporabniku prikazali atribute pri izbiri članov [1]. Nastavimo jo glede na pričakovano število članov. Priporočene vrednosti so prikazane v tabeli 1.

Vrednost nastavitve	Opis	Število članov
None	Seznama ne prikaži. Uporabniki naj vrednost sami neposredno vpišejo.	
DropDown	Število članov je dovolj majhno, da jih lahko prikažemo v padajočem seznamu.	< 100 članov
List	Število članov je preveliko za prikaz v padajočem seznamu, vendar ne zahteva filtriranja.	< 500 članov
FilteredList	Število članov je dovolj veliko, da je potrebno prikaz filtrirati.	< 5000 članov
MandatoryFilter	Število članov je tako veliko, da mora biti prikaz vedno filtriran.	> 5000 članov

Tabela 1. Priporočila za nastavitev InstanceSelection.

4.2.2 Kocka

Izbriši nepotrebne mere

Nepotrebne mere povečujejo velikost kocke in upočasnjujejo delovanje, zato jih izbrišimo [13].

Za mere uporabi najmanjši možni numerični podatkovni tip

Mere naj bodo čim manjšega podatkovnega tipa, vendar še vedno dovolj velikega, da bo mogoče definirati največjo možno agregirano vrednost [1]. Na ta način največ pridobimo pri prostoru, potrebnem za shranjevanje.

Mere, ki jih poizvedueš skupaj, daj v isto skupino mer

Mere, ki jih največkrat poizvedueš skupaj, daj v isto skupino mer [16]. Poizvedba, ki mora pridobiti podatke iz več skupin mer, zahteva več operacij. Mere, ki jih ne poizvedueš skupaj, daj v ločene skupine mer, ker s tem dosežeš bolj optimizirano uporabo predpomnilnika.

Izogibaj se več skupinam mer z isto granularnostjo in istimi dimenzijami

Skupine mer, ki imajo isto granularnost in uporabljajo iste dimenzije, po nepotrebnem ustvarjajo kompleksnejši model končnemu uporabniku. Ker imajo poleg tega tudi negativen vpliv na zmogljivost, jih je bolje združiti v eno samo skupino mer [1].

Isto dimenzijo uporabi večkrat

Namesto izdelave več identičnih dimenzij, kjer vsaki pripada drugačna vloga, izdelamo samo eno dimenzijo in jo uporabimo večkrat. S tem dosežemo lažje vzdrževanje in manj potrebnega prostora za shranjevanje [1].

Izogibaj se nepovezanim skupinam mer v isti kocki

Večje število skupin mer upočasnjuje delovanje kocke. Če imamo veliko tabel dejstev, ki med seboj niso povezane, oziroma jih ne bomo uporabljali skupaj, je bolje narediti več manjših kock in z vsako pokriti eno področje [1].

4.3 Zmogljivost poizvedb

4.3.1 Način shranjevanja mer in agregacij

[17] SSAS v kocki shranjuje podatke o merah in agregacijah. Podatki so shranjeni na disku in se uporabljajo pri generiranju rezultatov poizvedb. V grobem poznamo tri različne načine shranjevanja (angl. Storage Mode): MOLAP, HOLAP in ROLAP. Vsak nudi različne prednosti.

Način MOLAP shranjuje podatke in agregacije v večdimenzijski obliki (angl. multidimensional format). To je najučinkovitejši način shranjevanja podatkov. Tudi s stališča hitrosti poizvedb je način MOLAP najučinkovitejši. Slabost tega načina se kaže v aktualnosti podatkov, ki ni realnočasna. Ko kocka prebere podatke iz podatkovnega skladišča in jih sprocesira, se povezava s podatkovnim skladiščem prekine. Kakršne koli spremembe podatkovnega skladišča v kocki ne bodo vidne, dokler ne bo ponovno sprocesirana. Vendar večini organizacij ustreza načrtovano dnevno ali tedensko procesiranje kocke, kar omenjeni slabosti zmanjša pomen.

Način ROLAP podatke o merah in agregacijah pusti v relacijski obliki v podatkovnem skladišču. Ta način ima ničelno latenco. Vedno so na voljo najaktualnejši podatki. Vendar se ta način pri hitrosti poizvedb odreže najslabše, ker ničesar ne shranjuje v učinkoviti večdimenzijski obliki.

Način HOLAP predstavlja hibriden način med načinoma MOLAP in ROLAP. Podatki o merah so shranjeni v relacijskem podatkovnem skladišču, agregacije pa v večdimenzijski obliki. To pomeni, da so podatki na voljo realnočasno, agregacije pa se izračunavajo sproti, ko pride do sprememb v podatkih.

4.3.2 Uporaba particij

Particije (angl. partitions) fizično ločijo podatke dejstev na več enot [23]. *Mehanizem hrambe podatkov* lahko podatke iz več particij bere vzporedno, in sicer samo iz tistih particij, ki vsebujejo potrebne podatke. To lahko močno pohitri čas izvajanja poizvedb.

Podatke razdelimo na particije na podlagi elementa, ki te podatke učinkovito razdeli na posamezne enote. Najpogostejša izbira za ta element je primerna časovna enota, na primer leto ali mesec. Ko definiramo particije, SSAS interno, s pomočjo DataID-jev, zabeleži kateri podatek spada v katero particijo in na ta način ustvari podkocke.

Splošne smernice pri izdelavi particij [23]:

- Število zapisov v posamezni particiji naj bo manjše od 20 milijonov. Poleg tega naj velikost particije ne presega 250 MB prostora. Če presegamo katero od navedenih meja, premislimo o drugačni delitvi, ki bo ustvarila manjše particije.
- Število zapisov v posamezni particiji naj bo večje od 2 milijonov, oziroma 50 MB. V nasprotnem primeru premislimo o združitvi več particij v eno.
- Particija naj ima vsaj 4096 zapisov. V nasprotnem primeru mehanizem hrambe podatkov za to particijo ne bo ustvaril agregacij in indeksov. Mejo, ki je nastavljena na

4096 zapisov, sicer lahko na strežniku spreminjamo z nastavitvijo IndexBuildTreshold.

- Ni nujno, da so vse particije razdeljene na podlagi istega elementa. Na primer, ustvarimo lahko 3 letne, 11 mesečnih, 3 tedenske ter 1 – 7 dnevnih particij. Pomembno je le, da posamezen zapis spada v eno samo particijo.
- Vsaki particiji posebej lahko določimo različen način shranjevanja mer in agregacij.

4.3.3 Uporaba agregacij

Agregacija je vnaprej izračunan povzetek podatkov, s katerim SSAS izboljša zmogljivost poizvedb [23]. Natančneje je agregacija vsota mer glede na neko kombinacijo atributov dimenzij.

Agregacije poizvedbo pohitrijo zato, ker poskrbijo, da je potrebnih manj branj z diska. Na prvi pogled bi morda mislili, da je pri tem najbolj kritično število podatkov dejstev, vendar so dimenzije tiste, ki igrajo najbolj kritično vlogo. Pohitritev s pomočjo agregacij bom pojasnil na primeru.

Dimenzija Artikel	**Dimenzija Stranka**	**Dimenzija Datum naročila**
Ključ artikla - 200 članov	Ključ stranke - 5000 članov	Ključ datuma - 1095 članov
Barva - 10 članov	Spol - 2 člana	Mesec - 36 članov
Podkategorija artiklov - 20 članov	Mesto - 30 članov	Četrtletje - 12 članov
Kategorija artiklov - 50 članov	Regija - 4 člani	Leto - 3 člani

Slika 16. Dimenzije Artikel, Stranka in Datum naročila.

Slika 16 (vir [23]) prikazuje tri dimenzije enostavne kocke, ki vsebuje podatke o prodaji. Vsaka dimenzija ima štiri atribute. Na najnižjem nivoju kocke obstaja 200 posameznih artiklov, 5000 posameznih strank in 1095 posameznih datumov. Največje možno število vrednosti v kocki dobimo s kartezijskim produktom teh števil: 200 * 5000 * 1095 = 1.095.000.000. V realnosti je število vrednosti nekaj velikostnih razredov manjše, kajti izračunano število pomeni, da bi vsaka stranka vsak dan v letu kupila vse artikle, ki jih prodajamo. Recimo, da imamo samo 1.095.000 dejanskih vrednosti in želimo s poizvedbo dobiti vrednost prodaje po mesecih in podkategorijah izdelkov. Če imamo izdelano agregacijo, ki vnaprej izračuna prodajo po mesecih in podkategorijah izdelkov, je potrebno pregledati samo 720 vrednosti (20 podkategorij izdelkov * 36 mesecev), kar je veliko manj kot računanje iz 1.095.000 vrednosti.

Ob pravkar prebranem se lahko vprašamo zakaj ne bi izdelali kar vseh možnih agregacij in s tem maksimalno pohitrili delovanja kocke. Odgovor se skriva v prevelikem številu vseh teoretično možnih agregacij. Večina atributov v dimenziji je sestavljena iz dveh nivojev – nivo All in dejanski atribut. Iz tega sledi, da je potencialno mogoče izdelati 2^(število vseh atributov) agregacij. V primeru, da imamo v kocki 100 atributov, to pomeni 2^{100}, kar je enako $1{,}268 * 10^{30}$.

SSAS s pomočjo posebnega algoritma določi, katere agregacije bo dejansko izdelal in katerih ne. Algoritem uporablja analizo stroškov in koristi. Stroške agregacije primarno merimo z velikostjo agregacije. Velikost se izračuna s pomočjo različnih statističnih podatkov, kot so število zapisov v tabeli, število članov, število dimenzij, atributov, mer, ipd. Koristi agregacije dobimo tako, da primerjamo koliko vrednosti je potrebno pregledati z agregacijo in koliko brez nje. Na primer, če lahko 1.000.000 vrednosti agregiramo v 50 vrednosti, nam ta agregacija prinaša veliko koristi.

SSAS ponuja dva čarovnika, s katerima lahko ustvarimo agregacije in uporabljata opisani algoritem:

- Čarovnik za oblikovanje agregacij (angl. Aggregation Design Wizard): Algoritem stroškov in koristi uporabi na podlagi zasnove kocke in distribucije podatkov.
- Čarovnik za optimizacijo na podlagi uporabe (angl. Usage-Based Optimization Wizard): Algoritem stroškov in koristi uporabi na podlagi poizvedb, ki jih izvajajo uporabniki.

Natančnejši opis uporabe obeh orodij je na voljo v dodatkih B in C.

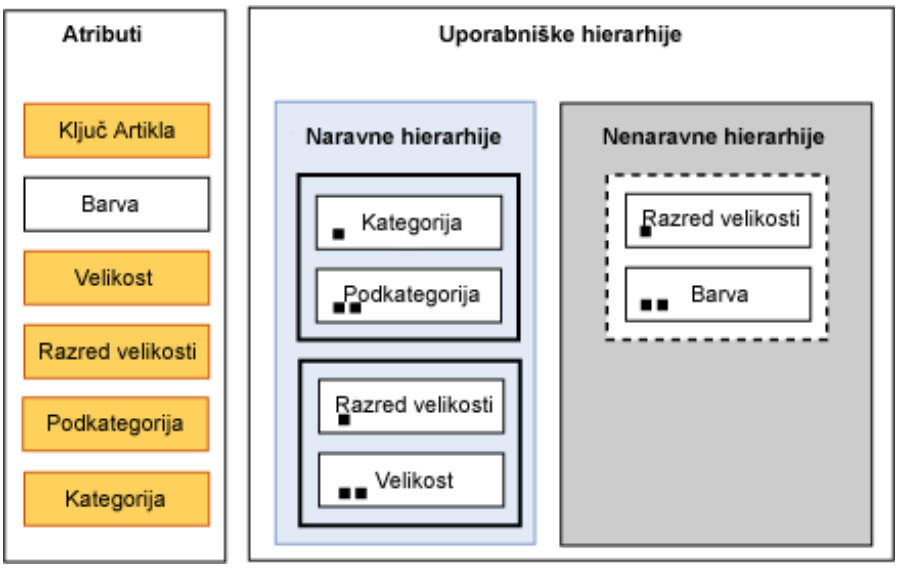
Algoritem izdelave agregacij v SSAS v prvem koraku določi atribute, ki so potencialni kandidati za agregacije. Kandidate določi z uporabo nastavitve AggregationUsage, ki ima lahko štiri različne vrednosti:

- Full: Vsaka agregacija v kocki mora vključevati tudi ta atribut.
- None: Nobena agregacija v kocki naj ne vključuje tega atributa.
- Unrestricted: Atribut je kandidat za agregacije. Potrebno je še oceniti ali je dober kandidat ali ne.
- Default: To je privzeta vrednost za vse atribute in pomeni, da se bo orodje samo odločilo na podlagi privzetega pravila (angl. default rule).

Privzeto pravilo deluje po naslednjih načelih:

1. Atributom, ki so ključ, in atributom All določi Unrestricted.
2. Atributom, ki nastopajo v posebnih zasnovah (npr. zasnova mnogo proti mnogo), določi None.
3. Atributom, ki so vključeni v naravne hierarhije, določi Unrestricted.
4. Vsem ostalim atributom določi None.

Slika 17 (vir [23]) prikazuje primer, kako bi lahko izgledala dimenzija Artikel po uporabi štirih privzetih pravil. Z rumeno barvo so označeni atributi, ki so določeni kot kandidati za agregacije.



Slika 17. Kandidati za agregacije v dimenziji Artikel po uporabi vseh privzetih pravil.

Atribut *Ključ Artikla* je določen kot kandidat na podlagi prvega pravila. Atributi *Velikost*, *Razred velikosti*, *Podkategorija* in *Kategorija* so določeni kot kandidati na podlagi tretjega pravila. Atribut *Barva* tudi po uporabi vseh štirih pravil ni kandidat za agregacije.

Ko so kandidati za agregacije določeni, se jih z analizo stroškov in koristi dejansko oceni. Pri oceni stroškov se upošteva ocenjeno število članov atributa in ocenjeno število članov v posamezni particiji. Te ocene lahko vpišemo sami na roke ali pa jih prešteje orodje namesto nas. Če so stroški agregacije previsoki, se jo takoj zavrže. Meja, ki določa previsoke stroške, je znana kot pravilo ene tretjine. Pravilo pravi, da SSAS nikoli ne ustvari agregacije, ki bi bila večja kot tretjina tabele dejstev. S tem se načeloma prepreči gradnjo agregacij, ki bi vsebovale atribute z veliko člani in bi prinašale minimalne koristi.

Za primer vzemimo kocko z naslednjo zasnovo:

- Dimenzija Stranka ima 5.000.000 posameznih strank organiziranih v 50.000 okrajev in 5.000 regij.
- Dimenzija Artikel ima 10.000 artiklov organiziranih v 1.000 podkategorij in 30 kategorij.
- Dimenzija Datum ima 1095 dni organiziranih v 36 mesecev in 3 leta.
- Tabela dejstev ima 12.000.000 zapisov o prodaji.

SSAS ne bo izdelal nobene agregacije, ki presega velikost 4.000.000 zapisov, kar je ena tretjina velikosti tabele dejstev. Zato atribut *Stranka* ne bo nastopal v nobeni agregaciji, ker že sam po sebi preseže dovoljeno mejo velikosti. Prav tako ne bo izdelana agregacija, ki bi vključevala *regijo stranke*, *kategorijo artiklov* in *mesece*. Njena potencialna velikost namreč prav tako presega dovoljeno mejo: 5000 * 30 * 36 = 5.400.000. Z uporabo particij bi lahko podatke razbili na manjše enote, ki bi ustrezale pravilu ene tretjine in s tem dovoljevale izdelavo določenih agregacij, ki brez uporabe particij to pravilo kršijo.

Poleg omenjenih čarovnikov, ki sta vgrajena v orodje BIDS, obstajajo še druga orodja, ki omogočajo ročno izdelavo agregacij. V njih se namesto algoritma sami odločimo, katere agregacije bomo ustvarili (kateri atributi jih bodo sestavljali). Najbolj znano in uporabniku prijazno orodje je Aggregation Manager[5].

Strategija kreiranja agregacij

Priporočljiva strategija kreiranja agregacij predlaga za začetek, ko smo končali z zasnovo kocke SSAS, kreiranje agregacij s pomočjo čarovnika za oblikovanje agregacij. Na koraku, kjer izberemo procentualno izboljšanje, ki ga želimo z agregacijami doseči, izberemo 20-30% ali manj [1]. Nato naj bo sistem OLAP nekaj časa v uporabi. Medtem beležimo poizvedbe, ki jih končni uporabniki izvajajo. Po izteku izbranega časovnega obdobja s čarovnikom za optimizacijo na podlagi uporabe ustvarimo dodatne agregacije. Pri tem naj bo izbran odstotek pričakovanega izboljšanja 70-80% [1]. Na koncu z dodatnimi orodji še ročno dodamo agregacije, za katere ocenimo, da manjkajo ter menimo, da bodo pohitrile določene poizvedbe, ki se izvajajo prepočasi.

Pogosto uporabljanim atributom nastavimo AggregationUsage na Unrestricted

Atribut, ki ni ključ dimenzije in ni vključen v nobeno naravno hierarhijo, ni obravnavan kot kandidat za agregacije. Če vemo, da je ta atribut zelo pogosto uporabljan in se uporablja v grupiranju in filtriranju, potem mu nastavimo nastavitev AggregationUsage na Unrestricted in ga s tem označimo kot kandidata za agregacije [1].

Pri velikih atributih se izogibaj nastavljeni AggregationUsage na Full

Atributom, ki imajo veliko število članov, ne nastavi nastavitve AggregationUsage na Full [1]. V nasprotnem primeru se lahko zgodi, da čarovnik ne bo ustvaril nobene agregacije. V vsaki agregaciji bi namreč moral biti vključen tudi omenjeni atribut. Posledično bi bili stroški vsake agregacije previsoki in se jih zato ne ustvari.

V agregaciji naj ne bo več atributov iz iste verige razmerij

Če imamo v agregacijo vključene atribute, ki pripadajo isti verigi razmerij, s tem podvajamo podatke, kajti vrednost enega atributa lahko izračunamo iz prvega. Zato morebitna podvajanja odstranimo [1]. Čarovnika za izdelavo agregacij to pravilo upoštevata sama po sebi.

Izogibaj se izdelavi fleksibilnih agregacij

Agregacija je fleksibilna kadar je v njej vsebovan vsaj en atribut, ki je neposredno ali posredno v fleksibilnem razmerju s ključem dimenzije [1]. Fleksibilne agregacije se pri procesiranju z ukazom ProcessUpdate izbrišejo, toge pa ne. Zato se v primeru, ko imamo na voljo obe možnosti, raje odločimo za izdelavo toge agregacije.

Čim bolj natančno nastavi MemberCount in RowCount

Ker se vrednosti MemberCount in RowCount uporabljata pri izračunu ocene stroškov agregacije, je pomembno, da ju čim bolj natančno določimo in s tem preprečimo, da bi ustvarili napačne agregacije, oziroma da bi pozabili na tiste, ki bi bile učinkovite [1].

---

[5] Več o programu in njegovi uporabi: http://bidshelper.codeplex.com/wikipage?title=Aggregation%20Manager

Naredi največ 500 agregacij
Izogibaj se pretiravanju s številom agregacij [1]. Če je agregacij preveč, se njihov učinek zmanjša. Splošno pravilo pravi, da naj agregacij ne bo več kot 500.

Podobne particije naj uporabljajo isto zasnovo agregacij
Če je več particij med seboj podobnih velikosti in se jih uporablja na podoben način, ni potrebno, da vsaki izmed njih posebej definiramo zasnove agregacij (angl. aggregation design) [1]. Dovolj je, da to storimo na eni particiji in zasnovo nato uporabimo še na drugih.

Za particije, ki so različnih velikosti in se jih ne uporablja na podoben način, ustvarimo svojo zasnovo agregacij. Splošno pravilo pravi, da so tri različne zasnove agregacij na eno skupino mer povsem dovolj.

Odstrani vse zasnove agregacij, ki se ne uporabljajo za nobeno particijo
Zasnove agregacij, ki jih ne uporablja nobena particija, po nepotrebnem upočasnjujejo operacije, ki delajo z metapodatki, zato jih odstranimo [1]. Čarovnika za kreiranje agregacij že sama po sebi odstranita take zasnove. Paziti moramo le, kadar agregacije ustvarjamo ročno.

4.3.4 Poizvedbe MDX

Kot sem omenil že v 1. Poglavju, se vse uporabniške zahteve po podatkih iz SSAS podajajo v jeziku MDX. Poizvedba je lahko enostavna in zahteva vse podatke iz določenega atributa ali pa je kompleksna in specificira želene podatke z uporabo različnih funkcij. Do sedaj sem opisal že veliko načinov, ki vplivajo na hitrost izvajanja poizvedb, vendar noben od opisanih načinov ne posega v samo poizvedbo. Tako kot pri drugih računalniških jezikih, velja tudi za jezik MDX, da v velikem številu primerov lahko dosežemo isti rezultat na več kot en način. Čeprav je končni rezultat isti, so določeni načini učinkovitejši od drugih. V nadaljevanju bom opisal možne načine optimizacije samih poizvedb MDX.

Postopek iskanja vira, ki povzroča počasnost poizvedbe, je sestavljen iz naslednjih korakov [20]:

1. Preveri zasnovo kocke SSAS in strojno opremo. Če se težave pojavljajo pri vseh poizvedbah, potem je morda problem nekje drugje in ne v samih poizvedbah. Lahko imamo slabo mrežno povezavo, prepočasen strežnik, ipd.
2. Preveri ali so težave s počasno poizvedbo stalne ali samo začasne. Če so težave samo začasne, potem problem zopet ni v sami poizvedbi, ampak verjetno v strežniku (veliko število sočasnih poizvedb, druge operacije na strežniku).
3. Ugotovi ali je ozko grlo *mehanizem izvajanja poizvedb* ali *mehanizem hrambe podatkov*. Pri tem si lahko pomagamo z različnimi orodji:
   - SQL Server Profiler: Omogoča nizkonivojski prikaz izvajanja poizvedbe. Za vsak del poizvedbe prikaže tudi informacijo o času izvajanja, zato lahko ugotovimo, v katerem delu izvajanja poizvedba porabi največ časa.
   - System Monitor: Omogoča prikaz različnih števcev, iz katerih lahko ugotovimo na primer, koliko branj iz diska je bilo potrebnih med izvajanjem poizvedbe in

podobno. Zaženemo ga s klikom na meni Start, nato kliknemo na Zaženi, vpišemo *perfmon.msc* in kliknemo na gumb V redu.

- Upravitelj opravil (angl. Task manager): Omogoča hitro ugotavljanje ozkega grla. *Mehanizem izvajanja poizvedb* uporablja eno samo nit, *mehanizem hrambe podatkov* pa več niti. Če med izvajanjem poizvedbe opazimo, da je večino časa obremenjeno samo eno jedro procesorja, je problem verjetno v *mehanizmu izvajanja poizvedb*. Če pa opazimo, da so obremenjena vsa jedra, je problem v *mehanizmu hrambe podatkov*. Upravitelja opravil zaženemo s pritiskom kombinacije tipk Ctrl+Alt+Del.
- Process Monitor[6]: Prikazuje aktivnosti datotečnega sistema. Tako lahko ugotovimo koliko časa je SSAS porabil za branje podatkov iz posamezne particije.
- MDX Script Performance Analyzer[7]: prikaže koliko časa posamezen izračun prispeva k skupnemu času poizvedbe.

Če ugotovimo, da je ozko grlo v *mehanizmu hrambe podatkov*, potem moramo uporabiti strategije za čim hitrejše branje iz diska oziroma čim bolj zmanjšati število branj iz diska. Te strategije vključujejo učinkovito kreiranje particij in agregacij ter učinkovito uporabo predpomnilnika.

Če ugotovimo, da je ozko grlo v *mehanizmu izvajanja poizvedb*, potem poskusimo poizvedbo MDX modificirati tako, da bo uporabljala blokovno usmerjeno vrednotenje (angl. block-oriented evaluation), da ne bo pregledovala praznih celic in da ne bo uporabljala počasnih funkcij. Poleg tega pri modificiranju poizvedbe MDX poskusimo upoštevati pravila, ki jih bom opisal v nadaljevanju.

Namesto CASE uporabi IIF

Če imamo poizvedbo z veliko pogojnimi vejitvami, je namesto funkcije CASE bolje uporabiti funkcijo IIF [20]. Funkcija IIF uporablja blokovno usmerjeno vrednotenje, funkcija CASE pa mnogo počasnejše vrednotenje posameznih celic (angl. cell-by-cell evaluation).

Namesto IIF uporabi SCOPE

Kadar želimo za različne člane dimenzij izračunati neko vrednost na različne načine, je bolje uporabiti stavek SCOPE kot funkcijo IIF [20]. Spodnja primera izračunata vrednost plana glede na državo. Za Slovenijo vrneta vrednost prodaje pomnoženo z 1,2, za ostale države pa vrednost prodaje pomnoženo z 1,7. Primera vrneta isti rezultat, vendar je drugi način učinkovitejši.

Primer 1 (uporaba IIF):

```
with member [Plan] as
iif ([Poslovalnica].[Drzava].CurrentMember is
[Poslovalnica].[Drzava].[Slovenija], 1.2 * [Measures].[Vrednost
prodaje], 1.7 * [Measures].[Vrednost prodaje])
```

[6] Več o programu: http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx
[7] Več o programu: http://mdxscriptperf.codeplex.com/

Primer 2 (uporaba SCOPE):

```
create member CURRENTCUBE.[Measures].[Plan] as
1.7 * [Measures].[Vrednost prodaje];

Scope ([Plan], [Poslovalnica].[Drzava].[Slovenija]);
This = 1.2 * [Measures].[Vrednost prodaje];
End scope;
```

Čim več uporabljaj ključno besedo NON EMPTY

Že pri agregacijah sem omenil, da kocke običajno ne vsebujejo podatkov za vse možne kombinacije atributov. Zato je pomembno kadarkoli se le da v poizvedbi uporabljati ključno besedo NON EMPTY [7]. Uporabimo jo pri specificiranju osi in s tem odstranimo prazne vrstice in stolpce iz rezultata. Recimo, da neka poizvedba brez uporabe NON EMPTY vrne rezultat, kot ga prikazuje tabela 2.

	Mera 1	**Mera 2**	**Mera 3**	**Mera 4**
Član 1	Null	Null	Null	Null
Član 2	Vrednost 1	Vrednost 2	Vrednost 3	Vrednost 4
Član 3	Null	Null	Null	Null
Član 4	Vrednost 5	Vrednost 6	Vrednost 7	Vrednost 8

Tabela 2. Rezultat poizvedbe brez uporabe NON EMPTY.

Če uporabimo ključno besedo NON EMPTY, bo ista poizvedba vrnila rezultat, kot ga prikazuje tabela 3.

	Mera 1	**Mera 2**	**Mera 3**	**Mera 4**
Član 2	Vrednost 1	Vrednost 2	Vrednost 3	Vrednost 4
Član 4	Vrednost 5	Vrednost 6	Vrednost 7	Vrednost 8

Tabela 3. Rezultat poizvedbe z uporabo NON EMPTY.

Čim več uporabljaj funkcijo NONEMPTY()

Medtem ko ključna beseda NON EMPTY deluje samo na oseh, lahko funkcijo NONEMPTY() uporabimo za odstranitev praznih vrednosti iz poljubne n-terice in nato, če želimo, dobljen rezultat uporabljamo v nadaljnjih funkcijah, kot sta na primer sortiranje in filtriranje [23].

Za izračunane člane uporabljaj ključno besedo NON_EMPTY_BEHAVIOR

Pri izračunanih članih (angl. calculated member) ključna beseda NON EMPTY nima učinka, kajti SSAS ne more vedeti ali je član brez vrednosti dokler ga ne izračuna. Zato uporabimo ključno besedo NON_EMPTY_BEHAVIOR [7]. Z uporabo te ključne besede vrednost izračunanega člana povežemo z vrednostjo neke mere iz tabele dejstev. S tem povemo SSAS, da če je vrednost izbrane mere prazna, potem je prazna tudi vrednost izračunanega člana.

Namesto FILTER uporabi EXISTS

Namesto filtriranja po lastnostih člana, raje uporabi funkcijo EXISTS [16].

Funkcijo FILTER uporabi pred ORDER
Če želimo dobiti urejene in filtrirane rezultate, jih najprej filtrirajmo s funkcijo FILTER in šele nato uredimo s funkcijo ORDER [22].

Funkcijo FILTER uporabi pred CROSSJOIN
Če želimo dobiti filtrirane rezultate in uporabiti funkcijo CROSSJOIN, potem najprej uporabimo funkcijo FILTER in šele nato CROSSJOIN [16].

Preoblikuj uporabo SUM in COUNT čez več dimenzij
Vsoto neke mere čez več dimenzij navadno izračunamo s formulo (1).

$$Sum(\{A\} * \{B\} * \{C\}, d) \qquad (1)$$

Vendar omenjeno formulo lahko preoblikujemo na način, ki omogoča učinkoviteje doseči isti rezultat. Formula (2) prikazuje preoblikovano formulo [22].

$$Sum(\{A\}, Sum(\{B\}, Sum(\{C\}, d\ ))) \qquad (2)$$

Podobno lahko storimo kadar čez več dimenzij uporabimo funkcijo COUNT. Formula (3) prikazuje kako izgleda preoblikovanje v tem primeru.

$$Count(\{A\} * \{B\} * \{C\}) = Sum(\{A\}, Sum(\{B\}, Count(\{C\}))) \qquad (3)$$

Namesto filtriranja s ključno besedo WHERE uporabi podpoizvedbo
Če želimo dobiti rezultate samo za določene člane, jih lahko v poizvedbi specificiramo s ključno besedo WHERE. Struktura poizvedbe izgleda takole:

```
SELECT
... on 0,
... on 1
FROM ime_kocke
WHERE ...
```

Namesto ključne besede WHERE je pogosto bolj učinkovito uporabiti podpoizvedbo [20]. Struktura preoblikovane poizvedbe potem izgleda takole:

```
SELECT
... on 0,
... on 1
FROM (SELECT
      ... on 0
      FROM ime_kocke)
```

Večji argument naj bo na levi strani formule
V formulah tipa *arg1 * arg2* naj bo levi argument tisti, ki v kocki zavzema večji prostor in ima več praznih (Null) vrednosti [23]. Na primer, bolje je uporabiti formulo *Prodajna vrednost * Menjalni tečaj* kot *Menjalni tečaj * Prodajna vrednost*. Podobno je bolje *Prodajna vrednost * 1,15* kot *1,15 * Prodajna vrednost*.

Mehanizem izvajanja poizvedb iterira prvi argument čez drugega, zato v predlaganih primerih izvede manj iteracij in posledično je čas izvajanja krajši.

Razvrščanje množic

Recimo, da želimo dobiti položaje prodajalcev glede na količino, ki so jo prodali. Želen rezultat dobimo z naslednjo poizvedbo:

```
WITH MEMBER [Measures].[Položaj] AS
RANK
([Prodajalec].[Prodajalec].CurrentMember, ORDER
([Prodajalec].[Prodajalec].[Prodajalec].MEMBERS,
[Measures].[Prodajna količina],
BDESC)
)
SELECT [Measures].[Položaj] ON 0,
[Prodajalec].[Prodajalec].[Prodajalec].MEMBERS on 1
FROM [ime_kocke]
```

Slabost opisane poizvedbe je v tem, da za vsakega prodajalca izvede sortiranje vseh prodajalcev in nato iz sortirane množice prebere položaj prodajalca. Poizvedbo lahko preoblikujemo na način, ki prodajalce sortira samo enkrat, nato pa za vsakega prodajalca prebere njegov položaj [20]. Preoblikovana poizvedba je sledeča:

```
WITH SET UrejeniProdajalci AS
ORDER
([Prodajalec].[Prodajalec].[Prodajalec].members,
[Measures].[Prodajna količina], BDESC)
MEMBER [Measures].[Položaj] AS
RANK
([Prodajalec].[Prodajalec].CurrentMember, UrejeniProdajalci)
SELECT [Measures].[Položaj] ON 0,
[Prodajalec].[Prodajalec].[Prodajalec].MEMBERS ON 1
FROM [ime_kocke]
```

Iz primera sledi pravilo, da je potrebno poizvedbo preoblikovati tako, da operacije na množici izvedemo samo enkrat. V ta namen množico definiramo izven mesta računanja s ključno besedo SET.

TopCount() in BottomCount() v povezavi s CROSSJOIN

Funkciji TopCount() in BottomCount() sta zelo potratni in se ju kot taki ne da pohitriti. Velike pohitritve se da doseči edino, če ju uporabljamo v povezavi s CROSSJOIN [22]. Poizvedba, napisana na počasen način, vse atribute s funkcijo CROSSJOIN najprej združi v ogromno množico in nato iz te množice vzame zgornjih oziroma spodnjih N elementov. Poizvedba, napisana na učinkovitejši način, najprej vzame zgornjih oziroma spodnjih N elementov vsakega atributa, jih združi v eno množico in na koncu iz te množice vzame zgornjih oziroma spodnjih N elementov. Rezultat je isti, vendar je drugi način hitrejši in manj potraten.

Kadar vrstni red ni pomemben, uporabi funkcijo UnOrder()
Poizvedba z uporabo funkcije UnOrder() lahko hitreje vrne rezultat kot poizvedba brez uporabe te funkcije [22].

Uporabljaj lastnost FORMAT_STRING
Lastnost FORMAT_STRING ponuja mehanizem za definiranje oblike vrednosti v celicah. Različne oblike lahko definiramo za pozitivne vrednosti, negativne vrednosti, ničle in vrednosti Null. Tak način določanja oblike vrednosti je hitrejši kot določanje s pomočjo različnih funkcij [23].

Namesto operatorja = uporabi operator IS
Če želiš preveriti vrednost člana, namesto formule (4) uporabi formulo (5). Formula (4) mora vsakega člana pretvoriti v niz znakov, kjer se izgublja dragocen čas [23].

$$IIF\ [Dimenzija].[Atribut].Name = "Naziv" \qquad (4)$$

$$IIF\ [Dimenzija].[Atribut] IS\ [Naziv] \qquad (5)$$

Izogibaj se izračunom na nivoju listov
Izračuni na nivoju listov so izračuni na članih, ki so v kocki na zelo nizkem nivoju [22]. Število članov se na nižjih nivojih močno povečuje, kar pomeni, da je v izračune na nivoju listov vključenih veliko število članov in so zato počasni. Tipičen primer bi bil izračun *Profit = Prihodki - Stroški*. Za dosego pravilnega rezultata moramo formulo izračunati na najnižjem nivoju kocke.

Rešitev je kreiranje nove mere v orodju BIDS. Kadarkoli imamo v izračunu vsoto ali razliko mer iz iste tabele dejstev, lahko namesto izračunanega člana uporabimo mero. Izračun profita iz primera, omenjenega v prejšnjem odstavku, bi tako premaknili v izračunan stolpec (angl. named calculation) v Data Source View in nato iz njega ustvarili novo mero. Tako bi se vrednosti izračunale že tekom procesiranja in ne šele pri uporabi izračunanega člana.

Izogibaj se počasnim funkcijam
Poizvedbe MDX poskusimo preoblikovati tako, da bodo čim manj uporabljale počasne funkcije [23]. Počasne funkcije so predvsem tiste, ki morajo nize znakov pretvarjati v člane dimenzij in njihove vrednosti. Primer takih funkcij so StrToSet, StrToMember, StrToValue.

Počasni funkciji sta tudi LookupCube in LinkMember ter vse funkcije, ki ne znajo uporabljati blokovno usmerjenega vrednotenja. Tabela 4 prikazuje seznam operatorjev in funkcij jezika MDX, ki podpirajo blokovno usmerjeno vrednotenje. Počasen izračun lahko v splošnem pohitrimo s preoblikovanjem na način, da uporabimo samo konstrukte iz tabele 4.

Aritmetični operatorji: *, /, +, -
Statične reference članov
Statične reference N-teric
Semiaditivne mere
Sum/Min/Max/Aggregate(Statično definirana množica)
Sum/Min/Max/Aggregate(množica, ...), kjer množica uporablja funkcije PeriodsToDate, YTD/QTD/MTD, Crossjoin, Cousin, CurrentMember, razpon (operator ":") in statično definirane

člane ali množice.
Measures.CurrentMember
IsEmpty
Relacijski operatorji: <, >, <=, itd.
NULL
Skalarji (konstante)

Tabela 4. Seznam konstruktov v SSAS 2005, ki podpirajo blokovno usmerjeno vrednotenje.

Namesto zapletenih izračunov uporabi semiaditivne funkcije

Pri uporabi semiaditivnih mer (angl. semiadditive measures) lahko uporabimo semiaditivne agregacijske funkcije, kot so FirstChild, LastChild, itd. Tak način je učinkovitejši od pisanja kompleksnih izračunov, s katerimi bi dosegli isti rezultat [23].

Numerične atribute spremeni v mere

Če želimo numerične atribute dimenzij med seboj seštevati, jih spremenimo v mere [23].

4.3.5 Uporaba predpomnilnika

Med izvajanjem poizvedb se rezultati shranjujejo v predpomnilnik in ostanejo na voljo bodočim poizvedbam, ki zahtevajo iste podatke. Branje podatkov iz predpomnilnika je zelo hitro, zato se da poizvedbe na ta način dobro pohitriti [1, 20].

Gretje predpomnilnika

Prednosti predpomnilnika lahko izkoristimo tako, da takoj po zagonu strežnika oziroma po procesiranju kocke, še preden uporabniki začnejo izvajati poizvedbe, čim več podatkov naložimo vanj. Ta postopek imenujemo *gretje predpomnilnika*. Največ pridobimo, če v predpomnilnik naložimo tiste podatke, ki se jih največ uporablja, oziroma tiste, za katere drugače porabimo veliko časa. Kateri so ti podatki, lahko ugotovimo s spremljanjem poizvedb, ki jih uporabniki izvajajo. Poizvedbe lahko spremljamo s funkcijo beleženja, ki je vgrajena v SSAS in je opisana v dodatku C.

Gretje predpomnilnika je v splošnem mogoče na dva načina. Z uporabo prvega načina namensko sprožimo izvajanje poizvedb, ki so čim bolj podobne tistim, ki jih želimo pohitriti. Med izvajanjem že sama logika v SSAS prenese potrebne podatke v predpomnilnik. Pri drugem načinu pa sami ročno napolnimo predpomnilnik z izbranimi kombinacijami atributov in poskusimo pokriti tiste kombinacije, ki jih želimo pohitriti.

Prvega načina se lahko lotimo z različnimi orodji:

- V SQL Server Reporting Services sestavimo poročila, ki vsebujejo poizvedbe, ki jih želimo pohitriti. Ta poročila nato zaženemo takoj po procesiranju kocke ali pa jim določimo urnik, po katerem naj se zaganjajo.
- V SQL Server Integration Services sestavimo paket, ki takoj po procesiranju kocke zažene še želene poizvedbe in s tem ogreje predpomnilnik.
- V ASCMD[8] sestavimo skripte z želenimi poizvedbami in jih nato zaženemo, kadar želimo.

[8] Več o programu: http://msdn.microsoft.com/en-us/library/ms365187(SQL.90).aspx

Drug način je mogoč z uporabo stavka CREATE CACHE[9] oziroma WITH CACHE. Omenjena stavka lahko uporabimo v poizvedbi MDX in z njuno pomočjo v predpomnilnik prenesemo podatke, ki jih specificiramo. Poizvedbo shranimo v ASCMD skripto.

Oba načina lahko avtomatiziramo, kar pomeni, da se predpomnilnik samodejno ogreje takrat, ko je to potrebno.

4.4 Zmogljivost procesiranja

Na čas procesiranja kocke in dimenzij vpliva marsikateri od že opisanih načinov optimizacije, predvsem tisti, ki optimizirajo podatkovno skladišče in zasnovo kocke. Nekateri načini na račun hitrejšega izvajanja poizvedb čas procesiranja celo podaljšajo. Drugi pa imajo pozitiven učinek tako na izvajanje poizvedb kot na procesiranje. V tem delu bom opisal nekaj dodatnih načinov optimizacije, ki vplivajo samo na čas, potreben za procesiranje.

4.4.1 Začasna mapa

Med procesiranjem SSAS uporablja začasni prostor na disku. Največjo učinkovitost dosežemo, če je ta prostor na voljo v diskovnem polju tipa RAID 0[10]. Pomembno je tudi, da je v diskovnem polju na voljo dovolj prostora. Nič nenavadnega namreč ni, če med procesiranjem velikost začasnega prostora preseže velikost kocke [13].

Privzeto je začasni prostor lociran na istem disku kot je nameščen SSAS. Lokacijo lahko spremenimo tako, da v nastavitvah strežnika SSAS spremenimo nastavitev TempDir na želeno lokacijo.

4.4.2 Uporabi OLE DB Provider

Ker je izvajalnik kode (angl. runtime) SSAS napisan v nativni kodi (angl. native code), je za povezovanje na vir podatkov najučinkovitejše uporabiti OLE DB Provider. Če uporabimo .NET Data Provider, je delovanje počasnejše [23].

4.4.3 Dimenzije

V določenih primerih nastavi Processing Group na nastavitev ByTable

Privzeto SSAS procesira dimenzije tako, da za vsak atribut v dimenziji izvrši ločen stavek SQL, ki vrne množico vseh različnih članov tega atributa [23]. Tak način delovanja je določen z nastavitvijo Processing Group, ki je privzeto nastavljena na ByAttribute. V večini primerov je ta nastavitev dobra, vendar obstajajo primeri, ko jo je bolje spremeniti na ByTable. Taki primeri običajno vsebujejo več atributov z velikim številom različnih članov. Če je Processing Group nastavljen na ByTable, potem SSAS izvrši en sam stavek SQL, ki vrne množico vseh različnih članov za vse atribute hkrati.

---

[9] Način, kako se najbolje lotiti gretja predpomnilnika z uporabo stavka CREATE CACHE, je zelo lepo opisal avtor Nicholas Dritsas v članku z naslovom How to warm up the Analysis Services data cache using Create Cache statement?, ki je dosegljiv na naslovu: http://blogs.msdn.com/b/sqlcat/archive/2007/04/26/how-to-warm-up-the-analysis-services-data-cache-using-create-cache-statement.aspx

[10] Več o RAID 0: http://en.wikipedia.org/wiki/RAID_0#RAID_0

Kadar uporabljamo nastavitev ByTable, se moramo zavedati, da strežnik uporabi drugačen algoritem procesiranja kot sicer. Pri tem algoritmu se lahko zgodi, da strežnik veliko podatkov shranjuje na disk, kar je zelo počasna operacija. Poleg tega lahko med procesiranjem dobimo opozorila o podvajanju podatkov, ki so v večini primerov lažna.

Uporabi ProcessAdd, če želiš samo dodati nove člane

Operacija ProcessAdd je izboljšana verzija ProcessUpdate, ki je namenjena samo vstavljanju novih članov. Ničesar ne briše ali osvežuje, zato je zelo učinkovita [11]. ProcessAdd ni na voljo v standardnem uporabniškem vmesniku, ampak samo preko XMLA skripte[11].

4.4.4 Kocka

Namesto Table binding uporabi Query binding

Posamezno particijo v kocki lahko vežemo na določeno tabelo (table binding) ali na določeno poizvedbo (query binding). Če uporabimo Query binding, imamo proste roke pri pisanju poizvedbe, ki iz podatkovnega skladišča bere podatke. Tako lahko napišemo učinkovitejšo poizvedbo [23].

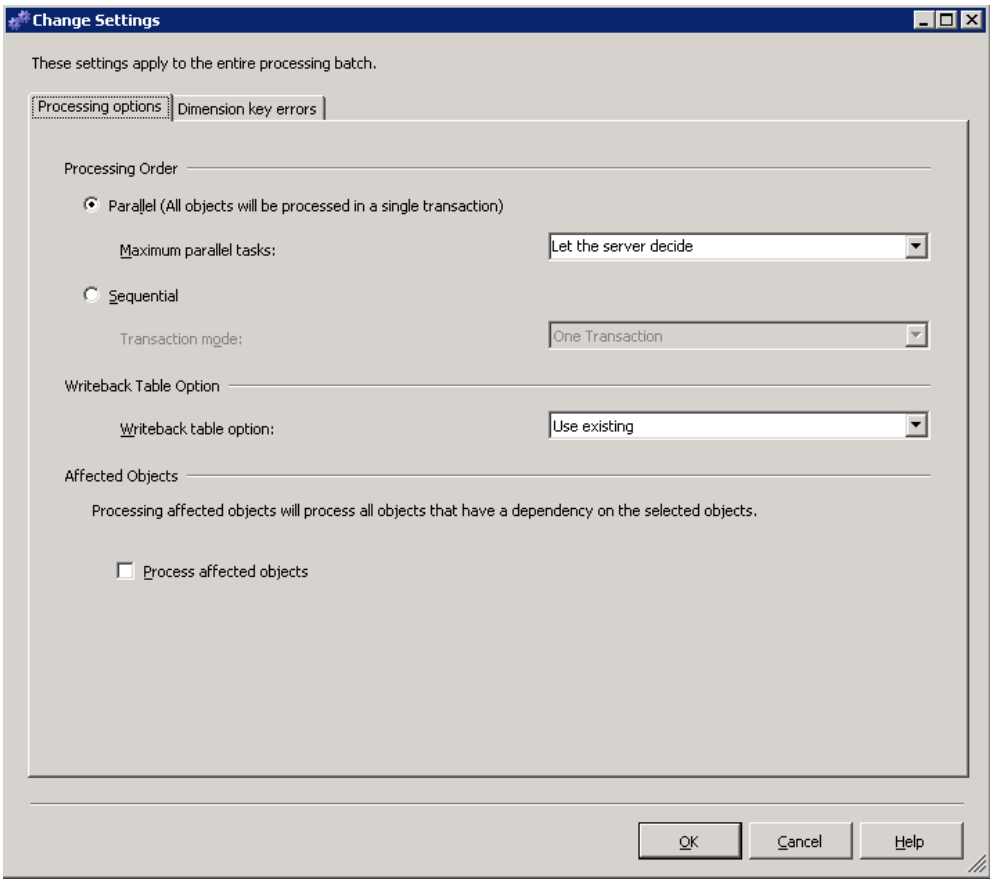
Namesto ProcessFull uporabi ProcessData in ProcessIndex

Operacija ProcessFull je sestavljena iz operacij ProcessData in ProcessIndex. V praksi se pogosto izkaže, da je ločena uporaba ProcessData in ProcessIndex učinkovitejša od uporabe ProcessFull. V splošnem je hitrejša in porabi manj strojnih virov [11].

Izkoristi vzporedno procesiranje

Uporabo particij sem omenil že pri optimizaciji poizvedb MDX. Tam smo z njihovo uporabo dosegli, da strežnik lahko pridobiva podatke iz več particij hkrati. Podobno pridobitev lahko izkoristimo tudi pri procesiranju. Če ustvarimo več particij, lahko strežnik procesira več particij vzporedno [11]. Privzeto se SSAS sam odloči koliko opravil se bo pri procesiranju izvajalo vzporedno. Če želimo to nastavitev spremeniti, moramo v oknu, kjer sprožimo procesiranje, klikniti na gumb Change Settings in v naslednjem oknu, ki se odpre, nastaviti polje Maximum parallel tasks na želeno vrednost. Priporočena vrednost je 1,5 – 2-kratnik števila procesorjev. Slika 18 prikazuje zaslonski posnetek okna, kjer lahko spreminjamo nastavitve procesiranja.

[11] Greg Galloway v članku ProcessAdd Examples, ki je dostopen na naslovu http://www.artisconsulting.com/blogs/greggalloway/Lists/Posts/Post.aspx?ID=4, lepo opiše izdelavo in uporabo XMLA skripte, namenjene operaciji ProcessAdd.



Slika 18. Nastavitve procesiranja.

4.5 Posebne zasnove

Struktura podatkov in njihov pomen včasih zahtevata realizacijo posebnih zasnov kocke, ki so drugačne od navadnih. Njihova drugačnost vpliva tudi na načine optimizacije. Do zdaj opisani načini imajo na njih pozitiven vpliv, vendar se jim moramo za dosego boljših rezultatov še dodatno posvetiti.

4.5.1 Posebna agregacijska funkcija – DistinctCount

DistinctCount je neaditivna agregacijska funkcija, ki prešteje vse enolične pojavitve entitete v skupini mer. Njena posebnost je v tem, da lahko močno poveča velikost agregacij.

DistinctCount mero daj v svojo skupino mer

Mero, katere agregacijska funkcija je DistinctCount, izoliraj stran od ostalih mer in jo daj v svojo skupino mer. Tako lažje nadzorujemo agregacije, v katerih nastopa [23].

Privzeto že samo orodje BIDS ustvari ločeno skupino mer za DistinctCount mere. Vendar, če kasneje spreminjamo agregacijske funkcije, moramo biti sami pozorni na to prakso.

DistinctCount mero particioniraj po atributu, ki ga štejemo

Pri DistinctCount merah particije najbolje izkoristimo, če jih ustvarimo glede na atribut, po katerem štejemo enolične pojavitve [23]. Na primer, če štejemo število različnih strank, potem

ustvarimo particije glede na enolični identifikator strank. V prvo particijo tako shranimo stranke z ID-jem 0 – 1000, v drugo stranke z ID-jem 1000-2000, ipd.

V tabeli dejstev ustvari indeks gruče

Procesiranje DistinctCount mer pogosto traja dolgo časa. Krivec za to je lahko poizvedba, ki bere podatke iz podatkovnega skladišča, kajti na koncu poizvedbe se samodejno doda stavek ORDER BY. Poizvedbo lahko pohitrimo s postavitvijo indeksa gruče na stolpec v izvorni tabeli, po katerem štejemo enolične pojavitve [23].

4.5.2 Semiaditivne agregacijske funkcije

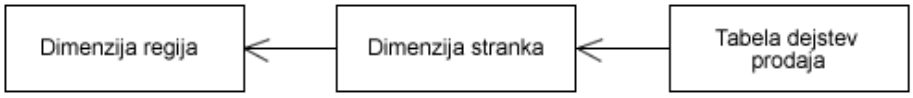
Semiaditivne agregacijske funkcije vključujejo funkcije ByAccount, AverageOfChildren, FirstChild, LastChild, FirstNonEmpty in LastNonEmpty. Te funkcije morajo vedno pridobiti podatke, ki vključujejo granulacijski atribut iz časovne dimenzije, drugače rezultati ne bi bili pravilni.

V agregacije semiaditivnih mer vključi granulacijski atribut časovne dimenzije

Če želimo imeti kaj koristi od agregacij pri semiaditivnih merah, moramo v agregacije obvezno vključiti granulacijski atribut časovne dimenzije [23]. Če vključimo atribute iz višjih nivojev časovne dimenzije, od njih ne bo nobenih koristi, ker se jih ne bo dalo uporabiti.

4.5.3 Referenčna razmerja

V navadnih razmerjih smo vajeni, da so vse dimenzijske tabele v neposrednem stiku s tabelo dejstev. Pri referenčnih razmerjih (angl. referenced relationship) to ne velja. Dimenzija, ki je na tabelo dejstev povezana z referenčnim razmerjem, uporablja referenčno dimenzijsko tabelo, preko katere je v stiku s tabelo dejstev. Slika 19 prikazuje primer dimenzije *regija*, ki je v referenčnem razmerju s tabelo dejstev *prodaja*.



Slika 19. Primer referenčnega razmerja.

Referenčna razmerja vedno materializiraj

Pri kreiranju referenčnih razmerij izberimo nastavitev, da naj se jih materializira [23]. S tem dosežemo, da se stik med tabelami izvede že med procesiranjem in ne šele med poizvedovanjem.

4.5.4 Razmerja mnogo proti mnogo

V večini scenarijev uporabljamo razmerja ena proti mnogo. To pomeni, da je vsak zapis v tabeli dejstev povezan z natanko enim zapisom v dimenzijski tabeli in vsak zapis v dimenzijski tabeli je lahko povezan z več zapisi v tabeli dejstev. Z uporabo takšne zasnove je podatke dimenzijskih atributov v poizvedbah enostavno agregirati.

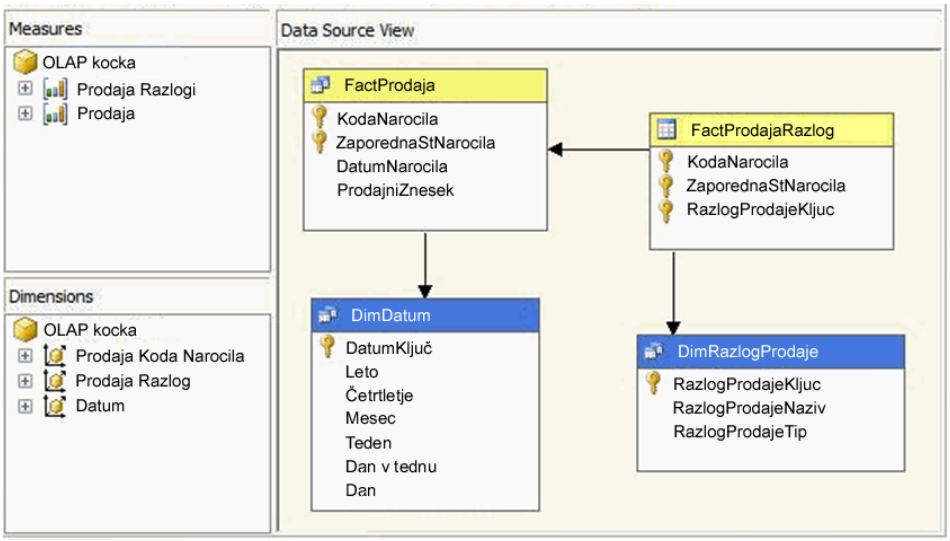
Razmerje mnogo proti mnogo pomeni, da je lahko posamezen zapis v tabeli dejstev povezan z več kot enim zapisom v dimenzijski tabeli. V takem primeru je podatke težje pravilno agregirati.

V SSAS modeliramo razmerje mnogo proti mnogo z uporabo vmesne skupine mer (angl. intermediate measure group) in vmesne dimenzije (angl. intermediate dimension), ki povezujeta tabelo dejstev in dimenzijsko tabelo.

Vsako razmerje mnogo proti mnogo (krajše M2M) je torej sestavljeno iz štirih komponent [6]:

- Podatkovna skupina mer: Ustvarjena je iz relacijske tabele dejstev s transakcijskimi podatki, ki jih želimo analizirati.
- Dimenzija M2M: Ustvarjena je iz dimenzijske tabele. Vsak zapis iz dimenzije M2M je lahko povezan z več zapisi iz podatkovne skupine mer in vsak zapis v podatkovni skupini mer je lahko povezan z več zapisi iz dimenzije M2M.
- Vmesna skupina mer: Ustvarjena je iz vmesne tabele dejstev, ki vsebuje podatke o povezavah med dimenzijo M2M in dimenzijsko tabelo, ki vsebuje stične ključe. Ta tabela dejstev ne vsebuje nobene mere. Ralph Kimball imenuje tak tip vmesne tabele dejstev *tabela dejstev brez dejstev* (angl. factless fact table)[12].
- Vmesna dimenzija: Običajno je ustvarjena iz glavne tabele dejstev. Vsebuje stične ključe, ki povezujejo zapise iz podatkovne skupine mer z zapisi iz vmesne skupine mer. Ralph Kimball imenuje tak tip dimenzije *degenerirana dimenzija* (angl. degenerate dimension)[13].

Primer na sliki 20 (vir [6]) prikazuje relacijski model razmerja M2M med dimenzijsko tabelo DimRazlogProdaje in tabelo dejstev FactProdaja preko vmesne tabele dejstev FactProdajaRazlog.



Slika 20. Model razmerja M2M.

[12] http://www.ralphkimball.com/html/designtipsPDF/DesignTips2003/KimballDT50FactlessFact.pdf
[13] http://www.kimballgroup.com/html/designtipsPDF/DesignTips2003/KimballDT46AnotherLook.pdf

Primer na sliki 20 vsebuje vse štiri komponente razmerja M2M:

- Podatkovna skupina mer se imenuje *Prodaja* in je ustvarjena iz glavne tabele dejstev *FactProdaja*.
- Vmesna skupina mer se imenuje *Prodaja Razlogi* in je ustvarjena iz vmesne tabele dejstev *FactProdajaRazlog*.
- Dimenzija M2M se imenuje *Prodaja Razlog* in je ustvarjena iz dimenzijske tabele *DimRazlogProdaje*.
- Vmesna dimenzija se imenuje *Prodaja Koda Narocila* in je ustvarjena iz glavne tabele dejstev *FactProdaja*.

Poizvedbe, ki jih izvajamo v zasnovi, ki vsebuje razmerja M2M, so lahko relativno počasne. Krivec je veliko število stikov, ki jih je potrebno izvesti med izvajanjem poizvedbe.

Definiraj agregacije

Z uporabo agregacij lahko zmanjšamo število stikov, ki jih je potrebno izvesti med izvajanjem poizvedbe [6]. Če želimo doseči zadovoljiv rezultat, moramo agregacije ustvariti tako na podatkovni kot vmesni skupini mer.

V podatkovni skupini mer ustvarimo agregacije, ki vključujejo granulacijski atribut vmesne dimenzije in granulacijske atribute ostalih dimenzij, ki se uporabljajo pri poizvedovanju. V primeru iz slike 20 bi ustvarili agregacijo, ki bi vsebovala atribut *KodaNarocila* iz dimenzije *Prodaja Koda Narocila* in atribut *DatumKljuc* iz dimenzije *DimDatum*.

V vmesni skupini mer ustvarimo agregacije, ki vključujejo granulacijski atribut vmesne dimenzije, atribut, po katerem želimo agregirati ter granulacijske atribute ostalih dimenzij, ki se uporabljajo pri poizvedovanju. V primeru iz slike 20 bi ustvarili agregacijo, ki bi vsebovala atribut *KodaNarocila* iz dimenzije *Prodaja Koda Narocila*, atribut *RazlogProdajeTip* iz dimenzije *DimRazlogProdaje* in atribut *DatumKljuc* iz dimenzije *DimDatum*.

Definiraj particije

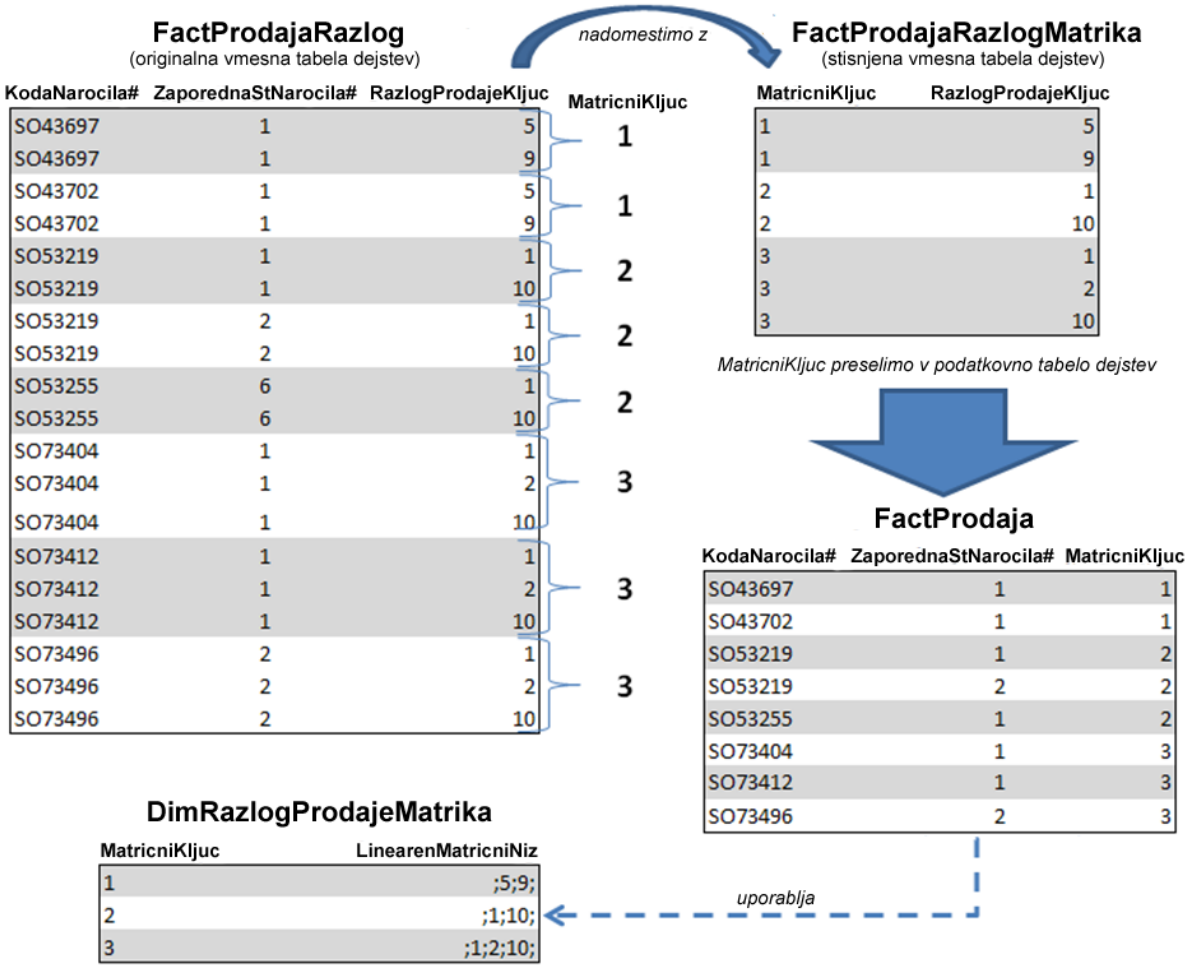
Z uporabo prednosti, ki jih prinašajo particije, lahko pohitrimo poizvedbe v zasnovah, ki vsebujejo razmerja M2M [6]. Če želimo doseči zadovoljiv rezultat, moramo particije definirati tako na podatkovni kot vmesni skupini mer. Particioniramo na podlagi atributa, ki se veliko uporablja v poizvedbah in s katerim podatke čim lepše razdelimo na več particij. Najpogostejša izbira je atribut iz datumske dimenzije. Ker želimo particionirati tudi vmesno skupino mer, jo moramo v kocki povezati z dimenzijo, po kateri particioniramo.

Uporaba matrike razmerij

Število potrebnih stikov, ki jih je potrebno izvesti med izvajanjem poizvedbe, lahko zmanjšamo tudi z optimizacijsko tehniko, ki se imenuje *matrika razmerij* (angl. Matrix Relationship Optimization Technique) [6]. Število stikov zmanjšamo s procesom stiskanja (angl. compression), s čimer odpravimo nepotrebne ponavljajoče podatke v vmesni tabeli dejstev. V procesu stiskanja najprej identificiramo ponavljajoče kombinacije članov dimenzije M2M, nato pa jih nadomestimo z nadomestnim ključem. Nadomestni ključ v tem primeru imenujemo *matrični ključ* (angl. matrix key). Z njegovo uporabo zmanjšamo velikost vmesne

tabele dejstev, kar se odraža v linearnem izboljšanju zmogljivosti poizvedb M2M. Stopnjo, do katere je velikost vmesne tabele dejstev možno zmanjšati, imenujemo *odstotek redukcije* (angl. reduction percentage). V dodatku Č je opisana poizvedba SQL, s katero lahko izračunamo pričakovan odstotek redukcije. Višji, kot je odstotek, več bomo s tehniko matrike razmerij pridobili.

Vsakemu zapisu v podatkovni tabeli dejstev pripišemo ustrezni matrični ključ. Ta ključ nato uporabimo kot stični ključ za povezovanje podatkovne tabele dejstev in dimenzije M2M preko vmesne tabele dejstev. Slika 21 (vir [6]) prikazuje uporabo opisanega koncepta na primeru iz slike 20.

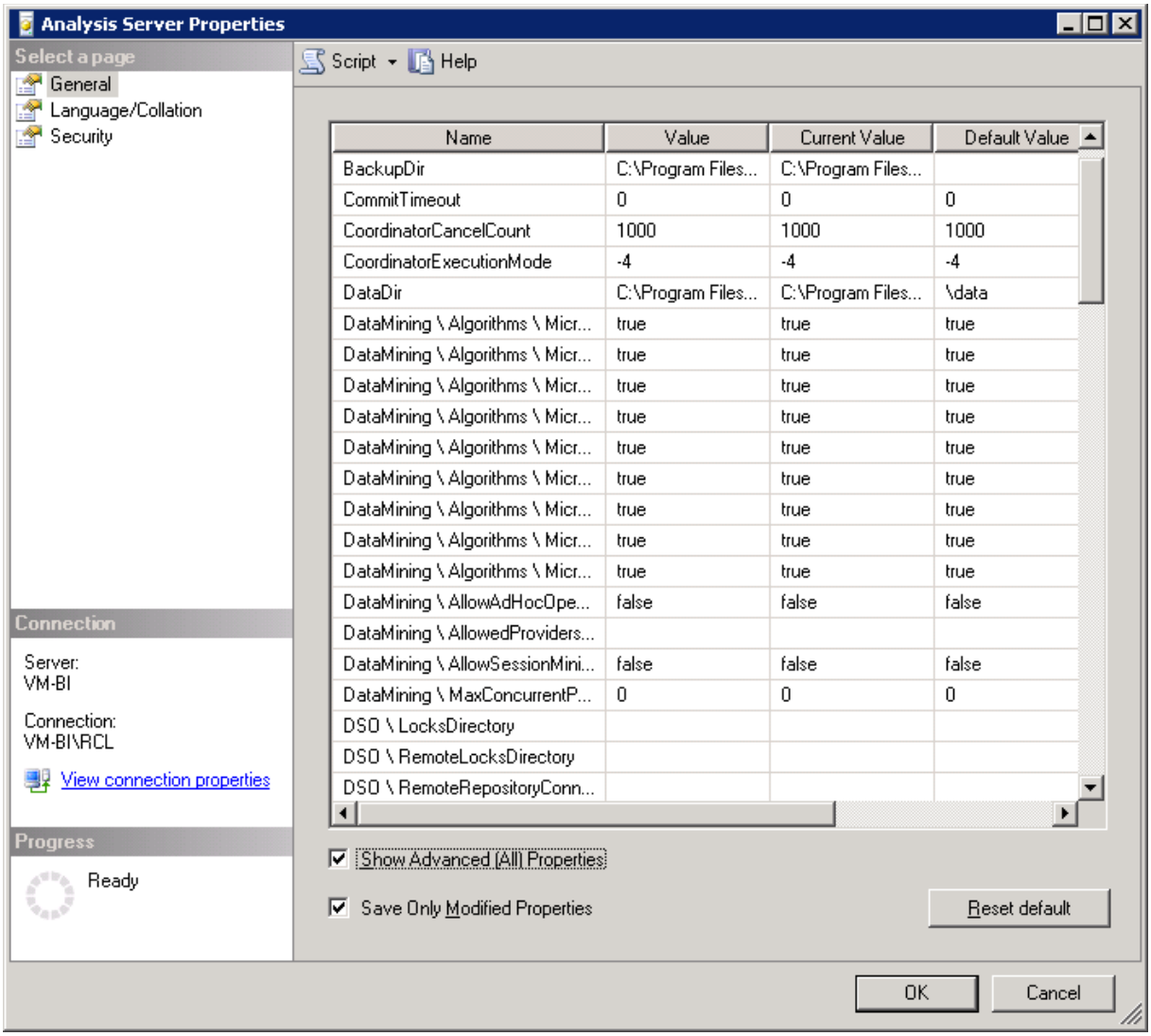


Slika 21. Stiskanje vmesne tabele dejstev v razmerju M2M.

V prvem koraku slike 21 smo v originalni vmesni tabeli dejstev identificirali tri ponavljajoče kombinacije razlogov prodaje. Tem enoličnim kombinacijam smo določili nadomestni matrični ključ. Vsako enolično kombinacijo smo združili v en niz in ga dodali v tabelo *DimRazlogProdajeMatrika*. Nato smo z uporabo matričnih ključev 19 zapisov iz originalne vmesne tabele dejstev stisnili na 7 zapisov v novi tabeli. Na koncu smo matrične ključe dodali še v podatkovno tabelo dejstev (tabela *FactProdaja*), tako da lahko za stike M2M namesto polj *KodaNarocila* in *ZaporednaStNarocila* uporabimo kar te ključe.

4.6 Nastavitve strežnika in strojna oprema

Dobri odzivni časi poizvedb in učinkovito procesiranje zahtevata učinkovito rabo računalniških virov: pomnilnika, procesorja in diska. SSAS 2005 omogoča nastavljanje različnih konfiguracijskih možnosti, s katerimi lahko optimiramo delovanje v različnih strežniških okoljih. Pri namestitvi orodja SSAS 2005 se nastavitve že same dinamično nastavijo na privzete vrednosti glede na količino fizičnega pomnilnika in število procesorjev. Kljub temu, da privzete nastavitve v večini okolij zadostujejo, jih lahko v določenih primerih spremenimo tako, da še povečamo učinkovitost rabe virov. Nastavitve strežnika lahko spreminjamo v oknu Analysis Services Properties, do katerega pridemo z orodjem SSMS[14]. V SSMS se povežemo na strežnik SSAS, nato nanj kliknemo z desnim miškinim gumbom in iz prikazanega menija izberemo Properties (Nastavitve). Slika 22 prikazuje zaslonski posnetek okna Analysis Services Properties.



Slika 22. Nastavitve strežnika SSAS.

[14] Več o programu: http://msdn.microsoft.com/en-us/library/ms174173.aspx

Poleg učinkovite rabe računalniških virov je pomembna tudi njihova količina in hitrost. V splošnem velja pravilo, ki pravi, da sta večja količina in večja hitrost boljši.

4.6.1 Učinkovita raba pomnilnika

Nastavitvi Memory\TotalMemoryLimit in Memory\LowMemoryLimit

SSAS za izvajanje operacij od operacijskega sistema Windows zahteva dodelitev pomnilnika. Ko dodeljenega pomnilnika ne potrebuje več, ga vrne operacijskemu sistemu. Nastavitvi TotalMemoryLimit in LowMemoryLimit nadzorujeta količino dodeljenega pomnilnika, in sicer na naslednji način [23]:

- *Memory\TotalMemoryLimit* predstavlja zgornjo mejo količine pomnilnika, ki ga strežnik uporablja za izvajanje operacij SSAS. Če je nastavljena vrednost med 0 in 100, se jo interpretira kot odstotek celotne količine fizičnega pomnilnika. Če je vrednost večja od 100, se jo razume kot absolutno količino pomnilnika izraženo v bajtih. Privzeto je vrednost nastavljena na 80. Nastavljena vrednost ne pomeni trdo postavljene meje, ampak opozorilo, da je uporabljena količina pomnilnika začela presegati dovoljeno in da jo je potrebno čim hitreje zmanjšati.
- *Memory\LowMemoryLimit* podobno predstavlja mejo količine pomnilnika, ki ga strežnik uporablja, vendar je postavljena nižje kot TotalMemoryLimit in služi kot blago opozorilo, da se je količina uporabljenega pomnilnika začela približevati dovoljeni meji. Vrednosti so izražene enako kot pri TotalMemoryLimit. Privzeto je vrednost nastavljena na 75.

Glede na količino uporabljanega pomnilnika in nastavljene mejne vrednosti se aktivira operacija čiščenja. Če je količina uporabljanega pomnilnika nižja od nastavitve LowMemoryLimit, potem čiščenje miruje. Če je količina uporabljanega pomnilnika med vrednostma LowMemoryLimit in TotalMemoryLimit, potem operacija čiščenja začne z uporabo posebnega algoritma (oceni, kaj se najbolj splača sprostiti) sproščati uporabljani pomnilnik. Če količina uporabljanega pomnilnika preseže vrednost TotalMemoryLimit, potem se aktivira agresivna operacija čiščenja, ki sprošča iz pomnilnika vse, kar se sprostiti da, dokler ponovno ne doseže vrednosti TotalMemoryLimit. Operacija čiščenja lahko počisti vrednosti iz *predpomnilnika mehanizma izvajanja poizvedb*, *predpomnilnika mehanizma hrambe podatkov*, shranjene podatke o dimenzijah ter pretečene seje.

Tekom delovanja SSAS moramo spremljati količino uporabljanega pomnilnika. Spreminjanje nastavitev TotalMemoryLimit in LowMemoryLimit se v različnih okoljih z različnimi podatki različno obnašata. Če se nam način uporabe pomnilnika ne zdi učinkovit, lahko testiramo različne vrednosti, v splošnem pa velja ohraniti razmik med TotalMemoryLimit in LowMemoryLimit vsaj 5%, raje več.

Nastavitev OLAP\Process\BufferMemoryLimit

[23] Med procesiranjem se pomnilnik uporablja za začasno shranjevanje podatkov o merah in agregacijah v medpomnilniku. V medpomnilniku se prebrane podatke iz relacijske podatkovne baze obdela in nato zapiše v particijo na disk. Nastavitev BufferMemoryLimit

določa količino pomnilnika, ki ga lahko uporablja posamezen posel pri procesiranju in s tem omejuje število zapisov, ki se jih v medpomnilniku procesira. Če je nastavljena vrednost manjša kot 100, se jo razume kot odstotek celotne količine fizičnega pomnilnika, če pa je večja od 100, pomeni absolutno količino izraženo v bajtih. Privzeto je nastavljena vrednost 60.

Privzeta vrednost v večini okolij ustreza, v določenih primerih pa se jo morda splača spremeniti. Če so uporabljene mere v kocki definirane na višjem nivoju kot podatki v podatkovnem skladišču (predstavljajo sumirane podatke iz relacijske podatkovne baze), se vrednost nastavitve BufferMemoryLimit morda splača povečati, zato da je med procesiranjem potrebno izvesti manjše število pisanj na disk. V nasprotnem primeru, ko je nivo mer enak relacijskim podatkov, se nastavitev morda splača zmanjšati, zato da se lahko med procesiranjem več poslov izvaja vzporedno.

Nastavitvi OLAP\Process\AggregationMemoryLimitMax in OLAP\Process\AggregationMemoryLimitMin

Med procesiranjem se za gradnjo agregacij uporablja medpomnilnik agregacij. Količina pomnilnika, ki se ga uporablja za medpomnilnik agregacij, je določena z naslednjima nastavitvama [23]:

- *OLAP\Process\AggregationMemoryLimitMax* predstavlja največjo dovoljeno količino pomnilnika, ki ga ima posamezen posel na voljo za gradnjo agregacij med procesiranjem. Če je vrednost manjša od 100, se jo razume kot odstotek celotne količine fizičnega pomnilnika, v nasprotnem primeru pa pomeni absolutno vrednost izraženo v bajtih. Privzeto je vrednost nastavljena na 80.
- *OLAP\Process\AggregationMemoryLimitMin* predstavlja najmanjšo dovoljeno količino pomnilnika, ki ga ima posamezen posel na voljo za gradnjo agregacij med procesiranjem. Če je vrednost manjša od 100, se jo razume kot odstotek celotne količine fizičnega pomnilnika, v nasprotnem primeru pa pomeni absolutno vrednost izraženo v bajtih. Privzeto je vrednost nastavljena na 10.

Med procesiranjem se v splošnem splača preveriti, če imajo vse agregacije dovolj prostora v pomnilniku. Če ga imajo na voljo premalo, se to odraža v rabi začasnega prostora na disku v mapi določeni z nastavitvijo TempDir. Pri spreminjanju nastavitev moramo biti pozorni na to, da ne presežemo meje, ki je določena z nastavitvijo *Memory\TotalMemoryLimit*. Rabo začasnega prostora lahko spremljamo z orodjem System Monitor.

Nastavitvi MinIdleSessionTimeout in MaxIdleSessionTimeout

Upravljanje s sejami odjemalcev se dogaja v pomnilniku. Vsaka seja uporablja različno količino pomnilnika, kar je odvisno od poizvedb in izračunov, ki se v seji izvajajo. Medtem ko aktivnih sej ne moremo uničiti, lahko iz pomnilnika pobrišemo pretečene seje. Kdaj seja postane pretečena določata naslednji nastavitvi [23]:

- *MinIdleSessionTimeout* predstavlja čas neaktivnosti seje, izražen v sekundah. Ko ta čas preteče, postane seja pretečena in operacija čiščenja jo lahko odstrani iz

pomnilnika, če presežemo dovoljeno količino uporabljanega pomnilnika. Privzeto je vrednost nastavljena na 2700 sekund (45 minut).

- *MaxIdleSessionTimeout* predstavlja čas neaktivnosti seje, izražen v sekundah. Ko ta čas preteče, strežnik uniči sejo ne glede na količino uporabljanega pomnilnika. Privzeto je vrednost nastavljena na 0 sekund, kar pomeni, da strežnik neaktivnih sej nikoli ne uniči sam od sebe.

V večini okolij privzete vrednosti zadoščajo, v določenih situacijah pa jih lahko spremenimo. Na primer, če želimo v pomnilniško bolj zasedenih obdobjih pretečene seje uničiti po 15 minutah, v nezasedenih pa po 45 minutah, bomo nastavili MinIdleSessionTimeout na 900 sekund (15 minut), MaxIdleSessionTimeout pa na 2700 sekund (45 minut). Pri tem moramo upoštevati tudi odjemalčevo aplikacijo. Nekatere aplikacije namreč že same vsebujejo mehanizem za upravljanje s sejami, ki je neodvisen od SSAS.

4.6.2 Učinkovita raba procesorja

Nastavitev CoordinatorExecutionMode

Procesorske vire najbolje izrabimo tako, da primerno število operacij izvajamo vzporedno. Kot sem omenil že v 2. poglavju, procesor operacije izvaja v obliki posameznih poslov. Nastavitev *CoordinatorExecutionMode* določa število poslov (vključuje tako poizvedovanje kot procesiranje), ki jih je dovoljeno izvajati vzporedno [23]. Negativno določena vrednost pomeni število poslov, ki se lahko vzporedno izvajajo na posameznem procesorju, pozitivno določena vrednost pa pomeni absolutno število poslov, ki se lahko vzporedno izvajajo na celotnem strežniku. Privzeto je vrednost nastavljena na -4.

Optimalno nastavitev vrednosti *CoordinatorExecutionMode* moramo iskati v kombinaciji z ostalimi nastavitvami, ki vplivajo na učinkovito rabo procesorja. Najti moramo najboljšo kombinacijo vseh nastavitev, kajti nastavitve vplivajo tudi ena na drugo.

Nastavitvi MaxThreads in MinThreads

Nastavitvi MaxThreads in MinThreads določamo ločeno za poizvedovanje in procesiranje. Namenjeni sta določanju največjega in najmanjšega števila niti, ki jih strežnik vzdržuje med izvajanjem poizvedb oziroma operacijami procesiranja. Privzete vrednosti so naslednje [23]:

- *ThreadPool\Query\MinThreads*: 1
- *ThreadPool\Query\MaxThreads*: 10
- *ThreadPool\Process\MinThreads*: 1
- *ThreadPool\Process\MaxThreads*: 64

Opisane vrednosti skupaj z nastavitvijo *CoordinatorExecutionMode* vplivajo na vzporednost izvajanja operacij. S pomočjo različnih sistemskih števcev, ki jih ponuja orodje System Monitor, lahko ugotavljamo kje je ozko grlo in katero nastavitev se splača spremeniti.

4.6.3 Učinkovita raba diska

Izklopi funkcijo beleženja

Funkcija *FlightRecorder* zagotavlja mehanizem hranjenja aktivnosti strežnika SSAS za kratko časovno obdobje [23]. Predvsem v situacijah, ko želimo razrešiti nek problem, nam s tem zelo olajša delo. Po drugi strani pa v določenih primerih prinaša nepotrebne stroške, ker podatke zapisuje na disk. Zato ga v produkcijskih okoljih, kjer te funkcije ne potrebujemo, izklopimo. To naredimo tako, da nastavitev *Log\FlightRecorder\Enabled* nastavimo na *false*. Privzeto je funkcija beleženja vklopljena.

4.6.4 Strojna oprema

OLAP in OLTP strežnika naj bosta ločena in povezana s hitro povezavo

Z ločitvijo transakcijskega in analitičnega strežnika lahko pridobimo na zmogljivosti [13]. Pri tem moramo poskrbeti, da med njima obstaja čim hitrejša mrežna povezava.

Uporabi čim več procesorjev in pomnilnika

Več kot imamo na voljo procesorskih virov (število procesorjev, število jeder) in pomnilnika, boljša bo zmogljivost sistema OLAP [7]. Števci orodja System Monitor so lahko v veliko pomoč pri odkrivanju težav zaradi premajhne količine pomnilnika. Eden izmed takih števcev je *Cache Hit Ratio*, ki pove kolikšen delež poizvedb je podatke dobilo v predpomnilniku. Zelo majhen delež je najverjetneje znak premajhne količine pomnilnika.

Uporabi RAID diskovno polje

Diskovni sistem strežnika SSAS naj bo sestavljen iz hitrega RAID polja [14]. Najbolj priporočljiva sta RAID 5[15] ali RAID 10[16]. Boljše zmogljivosti dobimo z uporabo več diskov z manjšo kapaciteto kot z uporabo manj diskov z večjo kapaciteto. Lokacijo podatkovnih datotek določa strežniška nastavitev DataDir.

TempDB prestavi na ločen hiter disk

Sistemsko podatkovno bazo TempDB se zaradi narave sistemov OLAP veliko uporablja. S stališča zmogljivosti se jo zato splača prestaviti na ločeno diskovno polje [14].

Pri ogromnem številu particij uporabi oddaljene particije

V kocki, ki vsebuje več tisoč particij, lahko hitrost izvajanja poizvedb povečamo z uporabo *oddaljenih particij* (angl. remote partitions) [7]. Oddaljene particije so fizično shranjene na ločenih strežnikih. Glavni strežnik zbira in agregira podatke, ki jih pridobi iz oddaljenih particij.

Uporaba gruče strežnikov

Pri velikem številu uporabnikov in poizvedb lahko ozko grlo sistema predstavlja procesor. V tem primeru zmogljivost poizvedb povečamo z uporabo gruče strežnikov SSAS, ki si med seboj porazdelijo breme zahtev [16]. Procesiranje podatkov prevzame eden od strežnikov,

---

[15] http://en.wikipedia.org/wiki/RAID_5#RAID_5
[16] http://en.wikipedia.org/wiki/RAID_10#RAID_10_.28RAID_1.2B0.29

ostali pa se z njim sinhronizirajo in služijo kot strežniki za poizvedovanje. Pridobitev zmogljivosti pri uporabi gruče je linearna.

Pozornost nameni tudi strojni opremi uporabnikov

Pri vsem načrtovanju in optimizaciji ne smemo pozabiti na strojno opremo končnih uporabnikov. Tudi ta lahko pomembno vpliva na zmogljivost poizvedb, zato naj bo čim boljša (večjedrni procesor, čim več pomnilnika, čim hitrejši disk) [14].

5 Primer optimizacije sistema OLAP

5.1 Opis testnega okolja

5.1.1 Strojna oprema

Vse teste v tem poglavju sem opravljal na strežniku z naslednjimi specifikacijami:

- Procesor: 2× Intel Xeon E5520 (2,27 GHz)
- RAM: 4 GB
- Disk: iSCSI diskovno polje, sestavljeno iz SATA2 diskov, povezanih v RAID 5

5.1.2 Programska oprema

Programska oprema testnega strežnika je bila naslednja:

- Operacijski sistem: Microsoft Windows Server 2003 R2, Standard x64 Edition, Service Pack 2
- SQL Strežnik: Microsoft SQL Server 2005 Service Pack 3, Enterprise Edition (64-bit)

5.2 Metodologija testiranja

Pri testiranju načinov optimizacije sistema OLAP sem se osredotočil na čas procesiranja in čas izvajanja testnih poizvedb. Čas procesiranja sem meril z orodjem BIDS, čas izvajanja poizvedb pa z orodjem SQL Server Profiler. Čas izvajanja poizvedb sem meril na 10 različnih poizvedbah, ki sem jih pripravil vnaprej. Vsako meritev sem opravil trikrat in kot končno vrednost upošteval aritmetično sredino teh treh meritev. Poleg časovnih količin sem spremljal tudi porabo prostora na disku – to pomeni velikost podatkovnega skladišča in večdimenzijske podatkovne baze.

Ker je bil cilj testirati čim več načinov, opisanih v teoretičnem delu diplomske naloge, sem se optimizacije lotil zelo postopno. V realnih okoljih bi verjetno veliko število korakov opravili istočasno. Jaz pa sem se namenoma lotil vsakega koraka posebej. Na ta način sem lahko izmeril morebitne pridobitve posameznih korakov in jih primerjal med seboj. Kljub temu sem na določenih mestih ocenil, da lahko nekatere sorodne korake združim in jih obravnavam skupaj.

Začetno stanje podatkovnega skladišča in kocke SSAS sem namenoma zasnoval popolnoma neoptimizirano. Čeprav takšnega začetnega stanja v realnosti ne bi pogosto srečali, je njegova zasnova povsem realna.

Pred testiranjem posameznega koraka sem izbrisal vse začasne podatke in podatke v predpomnilniku strežnika. S tem sem poskrbel za čim bolj enake pogoje testiranja vsakega od načinov.

5.3 Predstavitev testnih podatkov

Testiranje sem izvedel na pravih podatkih slovenskega podjetja iz živilske industrije. Podatkovno skladišče vsebuje podatke o prodaji, nabavi in zalogah od leta 2007 do 2010. Nivo granulacije v prodaji in nabavi je postavka na prodajnem ali nabavnem dokumentu. Nivo granulacije v zalogah je artikel na posamezen dan.

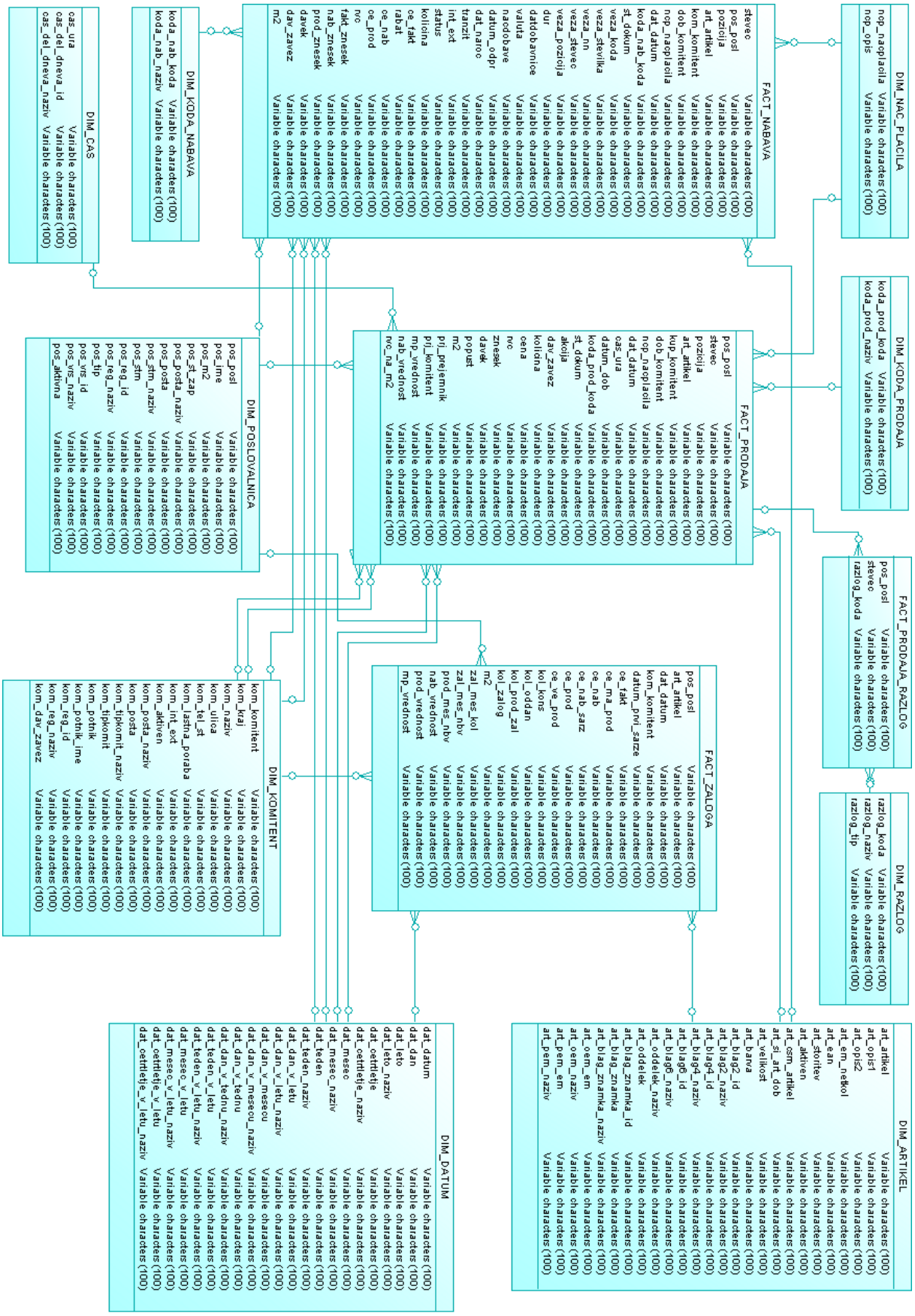
Podatki vsakega od omenjenih področij so shranjeni v svoji tabeli dejstev, in sicer:

- FACT_PRODAJA: Hrani podatke o prodaji (prodajna količina, prodajna vrednost, razlika v ceni, ipd.). Vsebuje 20.117.316 zapisov.
- FACT_NABAVA: Hrani podatke o nabavi (nabavna količina, nabavna vrednost, razlika v ceni, ipd.). Vsebuje 3.393.710 zapisov.
- FACT_ZALOGA: Hrani podatke o zalogah (količina, vrednost, ipd.). Vsebuje 4.883.336 zapisov.

Podatki o dimenzijah so shranjeni v naslednjih dimenzijskih tabelah:

- DIM_ARTIKEL: Hrani podatke o artiklih, blagovnih skupinah in blagovnih znamkah. Vsebuje 34.330 zapisov.
- DIM_CAS: Hrani podatke o urah v dnevu in delih dneva. Vsebuje 25 zapisov.
- DIM_DATUM: Hrani datumske podatke (leta, četrtletja, meseci, tedni, dnevi). Vsebuje 11.323 zapisov.
- DIM_KODA_NABAVA: Hrani podatke o tipih nabavnih dokumentov. Vsebuje 7 zapisov.
- DIM_KODA_PRODAJA: Hrani podatke o tipih prodajnih dokumentov. Vsebuje 4 zapise.
- DIM_KOMITENT: Hrani podatki o komitentih. Vsebuje 2.443 zapisov.
- DIM_NAC_PLACILA: Hrani podatke o načinih plačila. Vsebuje 18 zapisov.
- DIM_POSLOVALNICA: Hrani podatke o poslovalnicah, vrstah poslovalnic in regijah poslovalnic. Vsebuje 49 zapisov.
- DIM_RAZLOG: Hrani podatke o razlogih prodaje in tipih razlogov. Vsebuje 10 zapisov.

Poleg omenjenih tabel je v podatkovnem skladišču še tabela FACT_PRODAJA_RAZLOG. Gre za vmesno tabelo med tabelama FACT_PRODAJA in DIM_RAZLOG, ki sta v razmerju mnogo proti mnogo. Tabela vsebuje 5.782.794 zapisov. Slika 23 prikazuje entitetno-relacijski diagram podatkovnega skladišča, kakršnega smo uporabili za naše začetno stanje.



Slika 23. E-R diagram začetne zasnove podatkovnega skladišča.

5.4 Predstavitev testnih poizvedb

Čas izvajanja poizvedb sem meril na desetih različnih poizvedbah MDX.

Poizvedba 1:

```
with member [Measures].[Nova cena] as
case
when [ARTIKEL].[Blagovna skupina nivo 1 naziv].CurrentMember.Name = "MLEČNI
IZDELKI" then 1.1 * [Measures].[Prodajna cena prodaje]
when [ARTIKEL].[Blagovna skupina nivo 1 naziv].CurrentMember.Name =
"NEŽIVILA" then 1.2 * [Measures].[Prodajna cena prodaje]
when [ARTIKEL].[Blagovna skupina nivo 1 naziv].CurrentMember.Name =
"PIJAČA" then 1.3 * [Measures].[Prodajna cena prodaje]
else 0.9 * [Measures].[Prodajna cena prodaje]
end

select [DATUM].[Leto].[Leto 2010] on 0,
[ARTIKEL].[Blagovna skupina nivo 1 naziv].Children on 1
from [OLAP]
where [Measures].[Nova cena]
```

Opis: Z uporabo izračunanega člana *nova cena* za vsako blagovno skupino izračunamo novo prodajno ceno. Blagovni skupini MLEČNI IZDELKI bomo staro ceno dvignili za 10%, blagovni skupini NEŽIVILA za 20%, blagovni skupini PIJAČA za 30%, vsem ostalim blagovnim skupinam pa jo bomo znižali za 10%. Poizvedba nam vrne prihodke v letu 2010 po blagovnih skupinah, če bi se prodajali po novi ceni.

Poizvedba 2:

```
select [DATUM].[Leto].Children on 0, [POSLOVALNICA].[Naziv].Children on 1
from [OLAP]
where [Measures].[Prodajni RVC]
```

Opis: Poizvedba nam vrne razliko v ceni po letih in po poslovalnicah.

Poizvedba 3:

```
with member [Measures].[RVC nabave] as [Measures].[Prodajna vrednost
nabave] - [Measures].[Nabavna vrednost nabave]
select [Measures].[RVC nabave] on 0,
{[DATUM].[Dan].[1.7.2010], [DATUM].[Dan].[2.7.2010],
[DATUM].[Dan].[3.7.2010], [DATUM].[Dan].[4.7.2010],
[DATUM].[Dan].[5.7.2010], [DATUM].[Dan].[6.7.2010],
[DATUM].[Dan].[7.7.2010], [DATUM].[Dan].[8.7.2010],
[DATUM].[Dan].[9.7.2010], [DATUM].[Dan].[10.7.2010]} on 1
from [OLAP]
```

Opis: Za izbrane dneve izračunamo razliko v ceni iz področja nabave.

Poizvedba 4:

```
with member [Measures].[Povp cena] as
IIF([Measures].[Prodajna količina] = 0, null, ([Measures].[Prodajna
vrednost] + [Measures].[Popust]) / [Measures].[Prodajna količina])
select [Measures].[Povp cena] on 0,
```

```
FILTER(ORDER(CROSSJOIN([ARTIKEL].[Blagovna znamka naziv].Children,
[ARTIKEL].[Naziv].children), [Measures].[Povp cena], DESC),
[ARTIKEL].[Blagovna znamka naziv].CurrentMember IS [ARTIKEL].[Blagovna
znamka naziv].[PODJETJE D.D.]) on 1
from [OLAP]
where {[DATUM].[Mesec].[Maj 2010], [DATUM].[Mesec].[Junij 2010],
[DATUM].[Mesec].[Julij 2010]}
```

Opis: Poizvedba prikaže povprečno prodajno ceno artiklov blagovne znamke PODJETJE D.D. za mesece maj 2010, junij 2010 in julij 2010.

Poizvedba 5:

```
select {[DATUM].[Mesec].[Marec 2010], [DATUM].[Mesec].[April 2010],
[DATUM].[Mesec].[Maj 2010], [DATUM].[Mesec].[Junij 2010],
[DATUM].[Mesec].[Julij 2010]} on 0,
[ARTIKEL].[Blagovna skupina nivo 1 naziv].Children on 1
from [OLAP]
where [Measures].[Prodajna vrednost zaloge]
```

Opis: Poizvedba prikaže prodajno vrednost zalog po blagovnih skupinah za izbrane mesece.

Poizvedba 6:

```
select {[DATUM].[Četrtletje].[Č1, 2009], [DATUM].[Četrtletje].[Č1, 2010]}
on 0,
{CROSSJOIN([ARTIKEL].[Blagovna skupina nivo 1 naziv].Members,
[ARTIKEL].[Blagovna skupina nivo 2 naziv].Members)} on 1
from [OLAP]
where [Measures].[Prodajna vrednost]
```

Opis: Poizvedba vrne prodajno vrednost po prvem in drugem nivoju blagovnih skupin za prvo četrtletje letošnjega in lanskega leta.

Poizvedba 7:

```
select {[DATUM].[Mesec].[Avgust 2009],[DATUM].[Mesec].[September 2009],
[DATUM].[Mesec].[Oktober 2009], [DATUM].[Mesec].[November 2009],
[DATUM].[Mesec].[December 2009], [DATUM].[Mesec].[Januar 2010],
[DATUM].[Mesec].[Februar 2010], [DATUM].[Mesec].[Marec 2010],
[DATUM].[Mesec].[April 2010], [DATUM].[Mesec].[Maj 2010],
[DATUM].[Mesec].[Junij 2010], [DATUM].[Mesec].[Julij 2010]} on 0,
[KOMITENT].[Regija naziv].Members on 1
from [OLAP]
where [Measures].[Število različnih kupcev]
```

Opis: Poizvedba vrne število različnih kupcev po regijah za izbranih 12 mesecev.

Poizvedba 8:

```
SELECT [DATUM].[Leto].Members on 0,
[RAZLOG].[Tip razloga].[Tip razloga].Members on 1
from [OLAP]
where [Measures].[Prodajna vrednost]
```

Opis: Poizvedba vrne prodajno vrednost po tipih razlogov prodaje in letih.

Poizvedba 9:

```
SELECT {[DATUM].[Četrtletje].[Č3, 2009],[DATUM].[Četrtletje].[Č4, 2009],
[DATUM].[Četrtletje].[Č1, 2010],[DATUM].[Četrtletje].[Č2, 2010]} on 0,
[RAZLOG].[Tip razloga].[Tip razloga].Members on 1
from [OLAP]
where [Measures].[Prodajni RVC]
```

Opis: Poizvedba vrne razliko v ceni iz področja prodaje po tipih razlogov prodaje za izbrana četrtletja.

Poizvedba 10:

```
Select {[DATUM].[Teden].[Teden 19, 2010], [DATUM].[Teden].[Teden 20, 2010],
[DATUM].[Teden].[Teden 21, 2010], [DATUM].[Teden].[Teden 22, 2010],
[DATUM].[Teden].[Teden 23, 2010], [DATUM].[Teden].[Teden 24, 2010],
[DATUM].[Teden].[Teden 25, 2010], [DATUM].[Teden].[Teden 26, 2010],
[DATUM].[Teden].[Teden 27, 2010], [DATUM].[Teden].[Teden 28, 2010],
[DATUM].[Teden].[Teden 29, 2010], [DATUM].[Teden].[Teden 30, 2010]} on 0,
[ARTIKEL].[Blagovna znamka naziv].[Blagovna znamka naziv].Members on 1
from [OLAP]
where ([Measures].[RVC na M2])
```

Opis: Poizvedba vrne RVC na kvadraturo poslovalnice po blagovnih znamkah za izbranih 12 tednov.

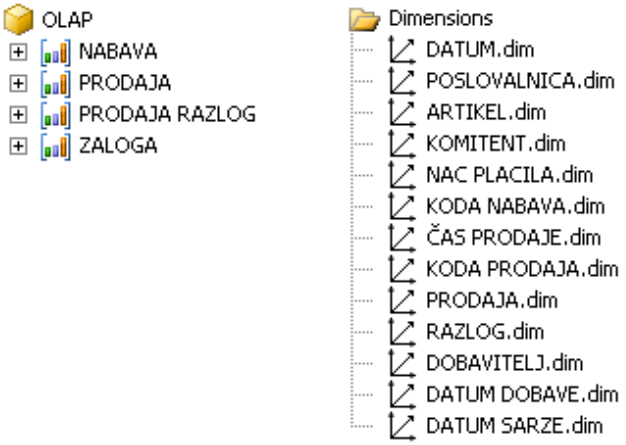
5.5 Rezultati testiranja

V tem podpoglavju bom predstavil rezultate testiranja načinov optimizacije. Pri vsakem načinu sem napisal kratek povzetek narejenega in rezultate prikazal v tabelah. V eni vrsti tabel so podatki o velikosti podatkovnega skladišča (stolpec DWH) in velikosti podatkovne baze SSAS (stolpec OLAP). V drugi vrsti tabel so podatki o času procesiranja (stolpec Proces.) in časih izvajanja posameznih testnih poizvedb (stolpci P1 do P10). Časi izvajanja poizvedb so podani v milisekundah (ms). Poleg meritev je v tabelah prikazano tudi procentualno izboljšanje glede na predhodni korak.

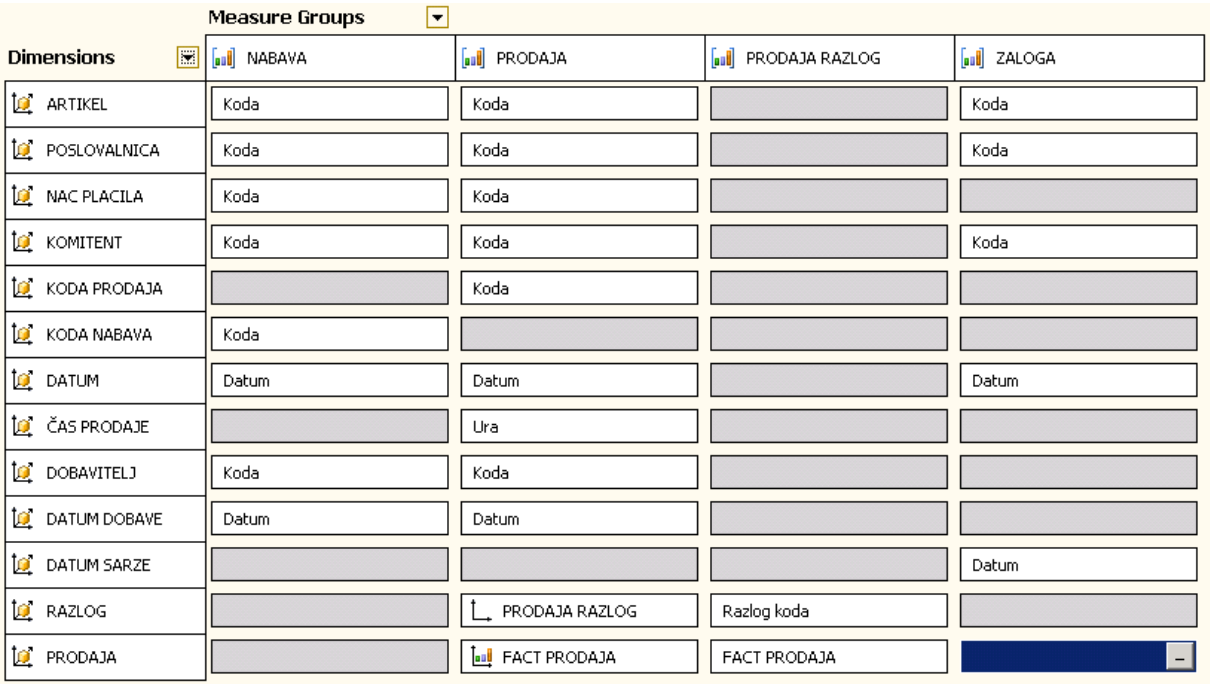
5.5.1 Začetno stanje

Začetno stanje podatkovnega skladišča sem predstavil že na sliki 23. Vidimo, da je uporabljena zvezdasta shema. Vsebuje 13 relacijskih tabel. Vsi atributi so tipa VARCHAR(100). Podatkovna baza ne vsebuje nobenih drugih objektov kot so ključi ali indeksi.

Začetno stanje kocke in dimenzij SSAS je določeno z uporabo čarovnikov v orodju BIDS. Z njim sem ustvaril vse potrebne objekte in jih pustil take, kot jih orodje privzeto ustvari. Ničesar nisem nastavljal ali spreminjal. Slilka 24 prikazuje seznam ustvarjenih skupin mer in dimenzij. Slika 25 prikazuje uporabo dimenzij v kocki.



Slika 24. Seznam skupin mer in dimenzij.



Dimensions	Measure Groups: NABAVA	PRODAJA	PRODAJA RAZLOG	ZALOGA
ARTIKEL	Koda	Koda		Koda
POSLOVALNICA	Koda	Koda		Koda
NAC PLACILA	Koda	Koda		
KOMITENT	Koda	Koda		Koda
KODA PRODAJA		Koda		
KODA NABAVA	Koda			
DATUM	Datum	Datum		Datum
ČAS PRODAJE		Ura		
DOBAVITELJ	Koda	Koda		
DATUM DOBAVE	Datum	Datum		
DATUM SARZE				Datum
RAZLOG		PRODAJA RAZLOG	Razlog koda	
PRODAJA		FACT PRODAJA	FACT PRODAJA	

Slika 25. Uporaba dimenzij v kocki.

Tabela 5 in tabela 6 prikazujeta rezultate testiranja začetne zasnove.

	DWH	OLAP
Meritev	5467 MB	1861 MB
Izboljšanje	-	-

Tabela 5. Rezultati testiranja začetne zasnove (velikost).

	Proces.	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Meritev	58min 12s	5587	17469	36	8576	1026	16912	19391	55252	17761	2650
Izboljšanje	-	-	-	-	-	-	-	-	-	-	-

Tabela 6. Rezultati testiranja začetne zasnove (čas).

5.5.2 Optimizacija podatkovnega skladišča

V prvem koraku sem vsem atributom v podatkovnem skladišču določil najmanjši možni podatkovni tip. To pomeni, da sem števila shranil kot števila in jim namesto podatkovnega

tipa VARCHAR(100) določil numerični tip. Tekstovnim atributom pa sem določil primerno dolžino. Rezultate testiranja prikazujeta tabela 7 in tabela 8.

	DWH	OLAP
Meritev	8677 MB	2186 MB
Izboljšanje	-37%	-15%

Tabela 7. Rezultati testiranja po določitvi podatkovnih tipov (velikost).

	Proces.	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Meritev	34min 55s	7061	22521	47	9191	765	21834	23761	66277	20589	3276
Izboljš.	67%	-21%	-22%	-22%	-7%	34%	-23%	-18%	-17%	-14%	-19%

Tabela 8. Rezultati testiranja po določitvi podatkovnih tipov (čas).

V naslednjem koraku sem iz vseh dimenzijskih tabel odstranil odvečne zapise (datume, komitente, …). Rezultate testiranja prikazujeta tabela 9 in tabela 10.

	DWH	OLAP
Meritev	8677 MB	2166 MB
Izboljšanje	0%	1%

Tabela 9. Rezultati testiranja po izbrisu odvečnih podatkov (velikost).

	Proces.	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Meritev	35min 22s	6983	22136	36	8951	771	20792	23302	65636	20355	3395
Izboljš.	-1%	1%	2%	28%	3%	-1%	5%	2%	1%	1%	-4%

Tabela 10. Rezultati testiranja po izbrisu odvečnih podatkov (čas).

Sledil je korak, v katerem sem ustvaril vse potrebne primarne in tuje ključe. V tabelah DIM_DATUM, DIM_KODA_NABAVA, DIM_KODA_PRODAJA, DIM_KOMITENT, FACT_PRODAJA, FACT_NABAVA, FACT_ZALOGA sem v ta namen ustvaril nove stolpce tipa INTEGER, ki služijo kot nadomestni ključi. Rezultate testiranja po tem koraku prikazujeta tabela 11 in tabela 12.

	DWH	OLAP
Meritev	10156 MB	3255 MB
Izboljšanje	-14%	-23%

Tabela 11. Rezultati testiranja po kreiranju ključev (velikost).

	Proces.	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Meritev	42min 32s	3983	12956	57	6572	776	13247	13330	56894	18205	2187
Izboljš.	-17%	75%	71%	-36%	36%	-1%	57%	75%	15%	12%	55%

Tabela 12. Rezultati testiranja po kreiranju ključev (čas).

Nato sem v podatkovnem skladišču postavil indekse. Držal sem se pravil podanih v teoretičnem delu naloge. Poleg ročnega kreiranja indeksov sem uporabil tudi orodja za optimizacijo postavitve indeksov. Rezultate testiranja prikazujeta tabela 13 in tabela 14.

	DWH	OLAP
Meritev	10156 MB	3239 MB
Izboljšanje	0%	0%

Tabela 13. Rezultati testiranja po kreiranju indeksov (velikost).

	Proces.	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Meritev	38min 7s	3910	12529	62	7118	760	13685	14648	58634	18787	2099
Izboljš.	12%	2%	3%	-8%	-8%	2%	-3%	-9%	-3%	-3%	4%

Tabela 14. Rezultati testiranja po kreiranju indeksov (čas).

Nazadnje sem v podatkovnem skladišču nastavil možnost *read-only* na TRUE. Na podatkovni bazi se tako lahko izvajajo samo operacije branja. Rezultate testiranja prikazujeta tabela 15 in tabela 16.

	DWH	OLAP
Meritev	10156 MB	3285 MB
Izboljšanje	0%	-1%

Tabela 15. Rezultati testiranja po uporabi nastavitve read-only (velikost).

	Proces.	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Meritev	39min 26s	4676	14419	88	7577	797	13705	14700	57794	20484	2140
Izboljš.	-3%	-16%	-13%	-30%	-6%	-5%	0%	0%	1%	-8%	-2%

Tabela 16. Rezultati testiranja po uporabi nastavitve read-only (čas).

5.5.3 Optimizacija zasnove podatkovne kocke

V tem sklopu sem se najprej lotil optimizacije dimenzij. V prvem koraku sem odstranil vse atribute, ki jih je orodje BIDS ustvarilo, vendar jih dejansko ne potrebujemo, zato so odvečni. Atributom sem tudi pravilno nastavil nastavitvi *KeyColumn* in *NameColumn*. Rezultate prikazujeta tabela 17 in tabela 18.

	DWH	OLAP
Meritev	10156 MB	2719 MB
Izboljšanje	0%	21%

Tabela 17. Rezultati testiranja po izbrisu odvečnih atributov (velikost).

	Proces.	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Meritev	32min 24s	4025	13450	67	7800	781	12836	13648	58055	18559	1968
Izboljš.	22%	16%	7%	31%	-3%	2%	7%	8%	0%	10%	9%

Tabela 18. Rezultati testiranja po izbrisu odvečnih atributov (čas).

V naslednjem koraku sem med atributi definiral vse povezave, ki obstajajo med njimi. Pri tem sem pazil, da nisem ustvaril nobene ponavljajoče ali diamantno oblikovane povezave. Vsaki povezavi sem določil tudi pravilen tip (toga ali fleksibilna). Rezultate testiranja prikazujeta tabela 19 in tabela 20.

	DWH	OLAP
Meritev	10156 MB	2719 MB
Izboljšanje	0%	0%

Tabela 19. Rezultati testiranja po določenih povezavah (velikost).

	Proces.	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Meritev	31min 4s	4280	12643	62	7055	776	12981	13299	55758	17683	1958
Izboljš.	4%	-6%	6%	9%	11%	1%	-1%	3%	4%	5%	1%

Tabela 20. Rezultati testiranja po določenih povezavah (čas).

Potem sem v vsaki dimenziji definiral smiselne naravne hierarhije atributov. Rezultate testiranja prikazujeta tabela 21 in tabela 22.

	DWH	OLAP
Meritev	10156 MB	2720 MB
Izboljšanje	0%	0%

Tabela 21. Rezultati testiranja po definiranih hierarhijah (velikost).

	Proces.	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Meritev	30min 34s	4346	13512	73	8383	1073	13429	22474	60849	27998	2114
Izboljš.	2%	-2%	-6%	-15%	-16%	-28%	-3%	-41%	-8%	-37%	-7%

Tabela 22. Rezultati testiranja po definiranih hierarhijah (čas).

Nato sem ustreznim atributom spremenil nastavitve *AttributeHierarchyVisible*, *AttributeHierarchyEnabled* in *AttributeHierarchyOptimizedState*. Rezultate testiranja prikazujeta tabela 23 in tabela 24.

	DWH	OLAP
Meritev	10156 MB	2692 MB
Izboljšanje	0%	1%

Tabela 23. Rezultati testiranja po spremenjenih nastavitvah atributov (velikost).

	Proces.	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Meritev	29min 24s	4759	13731	78	8207	1104	13919	22354	33279	27848	2166
Izboljš.	4%	-9%	-2%	-7%	2%	-3%	-4%	1%	83%	1%	-2%

Tabela 24. Rezultati testiranja po spremenjenih nastavitvah atributov (čas).

Sledila je optimizacija kocke. Odstranil sem vse odvečne mere, ki jih za potrebe analize ne potrebujemo. Rezultate testiranja prikazujeta tabela 25 in tabela 26.

	DWH	OLAP
Meritev	10156 MB	2535 MB
Izboljšanje	0%	6%

Tabela 25. Rezultati testiranja po odstranjenih odvečnih merah (velikost).

	Proces.	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Meritev	30min 37s	3317	10711	52	7180	615	10873	16038	28181	23484	1651
Izboljš.	-4%	43%	28%	50%	14%	80%	28%	39%	18%	19%	31%

Tabela 26. Rezultati testiranja po odstranjenih odvečnih merah (čas).

V zadnjem koraku tega sklopa sem odstranil podvojene dimenzije. Dimenzijski tabeli DIM_KOMITENT in DIM_DATUM se v kocki uporabljata večkrat. Na podlagi tabele DIM_KOMITENT smo ustvarili dimenziji *Komitent* in *Dobavitelj*. Na podlagi tabele DIM_DATUM smo ustvarili dimenzije *Datum*, *Datum dobave* in *Datum sarze*. Podvojene dimenzije sem izbrisal. Namesto njih sem dimenziji *Komitent* in *Datum* uporabil večkrat pod različnimi imeni. Rezultate testiranja prikazujeta tabela 27 in tabela 28.

	DWH	OLAP
Meritev	10156 MB	2534 MB
Izboljšanje	0%	0%

Tabela 27. Rezultati testiranja po izbrisu podvojenih dimenzij (velikost).

	Proces.	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Meritev	29min 30s	3562	10711	57	7301	630	11154	17376	28405	24348	1760
Izboljš.	4%	-7%	0%	-9%	-2%	-2%	-3%	-8%	-1%	-4%	-6%

Tabela 28. Rezultati testiranja po izbrisu podvojenih dimenzij (čas).

5.5.4 Optimizacija zmogljivosti poizvedb

V testnem okolju nimamo zahteve po realnočasnih podatkih, zato ni potrebe, da bi za način shranjevanja mer in agregacij izbrali ROLAP ali HOLAP. Kljub temu sem testiral tudi omenjena načina. Ker je način MOLAP najhitrejši v vračanju rezultatov poizvedb, smo ga obdržali tudi v nadaljevanju. Rezultate testiranja prikazujeta tabela 29 in tabela 30.

	DWH	OLAP
MOLAP	10156 MB	2534 MB
ROLAP	10156 MB	1484 MB
HOLAP	10156 MB	1484 MB

Tabela 29. Rezultati testiranja različnih načinov shranjevanja mer in agregacij (velikost).

	Proces.	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
MOLAP	29min 30s	3562	10711	57	7301	630	11154	17376	28405	24348	1760
ROLAP	8min 11s	54534	89510	7639	51971	29602	195313	198187	>30min	>30min	39351
HOLAP	8min 15s	48989	87793	6296	46677	30837	178570	209219	~20min	~16min	19027

Tabela 30. Rezultati testiranja različnih načinov shranjevanja mer in agregacij (čas).

Nato sem v kocki ustvaril več particij. Podatke sem razdelil na podlagi datuma. V skupinah mer NABAVA in ZALOGA sem za vsako leto ustvaril svojo particijo. V skupini PRODAJA sem za vsa pretekla leta ustvaril svojo particijo, za tekoče leto pa sem ustvaril več particij, in sicer za vsak mesec svojo. Rezultate testiranja prikazujeta tabela 31 in tabela 32.

	DWH	OLAP
Meritev	10156 MB	2523 MB
Izboljšanje	0%	0%

Tabela 31. Rezultati testiranja po particioniranju (velikost).

	Proces.	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Meritev	23min 20s	1703	7238	26	6462	599	6223	11638	26302	23677	823
Izboljš.	26%	109%	48%	119%	13%	5%	79%	49%	8%	3%	114%

Tabela 32. Rezultati testiranja po particioniranju (čas).

Sledila je uporaba agregacij. Držal sem se strategije opisane v teoretičnem delu naloge. Najprej sem preveril, če kakšen pogosto uporabljan atribut ni vključen v nobeno naravno hierarhijo. Če bi obstajal, bi mu nastavitev *AggregationUsage* nastavil na *Unrestricted*. Ker v našem primeru ni bilo nobenega takšnega atributa, sem nadaljeval z uporabo *čarovnika za oblikovanje agregacij*. V čarovniku sem želeno pridobitev z ustvarjenimi agregacijami nastavil na 30%. Nato sem uporabil še *čarovnika za optimizacijo na podlagi uporabe* in nastavil želeno pridobitev na 80%. Rezultate testiranja prikazujeta tabela 33 in tabela 34.

	DWH	OLAP
Meritev	10156 MB	2671 MB
Izboljšanje	0%	-6%

Tabela 33. Rezultati testiranja po uporabi agregacij (velikost).

	Proces.	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Meritev	45min 36s	21	16	16	7129	11	52	36	26828	23662	16
Izboljš.	-49%	8008%	45140%	63%	-9%	5516%	11867%	31932%	-2%	0%	5042%

Tabela 34. Rezultati testiranja po uporabi agregacij (čas).

Naslednji korak je bilo preoblikovanje poizvedb MDX. V tem koraku sem k seznamu testnih poizvedb dodal še eno dodatno poizvedbo. Z njo sem preizkusil eno od pravil, ki je opisano v teoretičnem delu naloge in ni vključeno v ostalih deset testnih poizvedb. Pri preoblikovanju sem uporabil naslednje načine:

- Poizvedba 1:
  - Uporabil sem ključno besedo NON EMPTY.
  - Namesto funkcije CASE sem uporabil SCOPE.
  - Spremenil sem vrstni red operandov v formuli za izračun nove cene.
- Poizvedba 2:
  - Uporabil sem ključno besedo NON EMPTY.
  - Uporabil sem funkcijo UnOrder().
- Poizvedba 3:
  - Izračun vrednosti *RVC nabave* sem premaknil v Data Source View.
- Poizvedba 4:
  - Uporabil sem lastnost FORMAT_STRING.
  - Uporabil sem lastnost NON_EMPTY_BEHAVIOR.
  - Spremenil sem vrstni red uporabe funkcij CROSSJOIN, ORDER in FILTER.
  - Uporabil sem ključno besedo NON EMPTY.
  - Uporabil sem funkcijo NonEmpty().
  - Namesto ključne besede WHERE sem uporabil podpoizvedbo.
- Poizvedba 5, poizvedba 6, poizvedba 7, poizvedba 8, poizvedba 9, poizvedba 10:
  - Uporabil sem ključno besedo NON EMPTY.

Za dodatno poizvedbo sem izbral izračun položaja vsakega komitenta glede na količino, ki smo mu jo prodali. Poizvedbo in njeno preoblikovano verzijo si lahko ogledate na strani 41 v podpoglavju Razvrščanje množic. Čas izvajanja osnovne (nepreoblikovane) verzije je bil 11977 ms.

Rezultate testiranja po izvedbi tega koraka prikazuje tabela 35.

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Dodatna
Meritev	21	16	16	338	11	52	36	26885	23496	16	94
Izboljš.	0%	0%	0%	2009%	0%	0%	0%	0%	1%	0%	12641%

Tabela 35. Rezultati testiranja po preoblikovanju poizvedb MDX (čas).

Zadnji način, ki sem ga testiral v tem sklopu, je gretje predpomnilnika. Ker s tem korakom pridobimo na času predvsem, če dovolj natančno poznamo poizvedbe, ki se izvajajo, sem ga v nadaljnjem testiranju izpustil. Težimo namreč k optimizaciji, ki izboljša zmogljivost večine poizvedb, ne samo ozko izbranih.

Za gretje predpomnilnika sem si izbral uporabo poročil SSRS. Za vsako poizvedbo sem izdelal poročilo, ki je prikazovalo iste podatke, kot jih je vračala poizvedba. Slika 26 prikazuje primer takega poročila. Takoj po procesiranju podatkov sem zagnal kreiranje vseh poročil. Nato sem testiral poizvedbe. Vseh deset testnih poizvedb se je izvedlo v času krajšem od 10 ms.

GretjePredpomnilnika

	Leto 2007	Leto 2008	Leto 2009	Leto 2010
Marketing	20.3471	9305.0530000005	261.568399993	5076.676499996
Ostalo	6570.1910999996	594150.32470119	795015.47910562	465463.83500459
Promocija	824.45119999999	60959.59110001	21601.824100001	5589.032899974

Slika 26. Primer poročila SSRS, ki služi gretju predpomnilnika.

5.5.5 Optimizacija zmogljivosti procesiranja

V tem sklopu sem testiral samo čas procesiranja. Ostalih vrednosti nisem meril, ker bi bile ves čas iste. Načini, ki sem jih testiral, na njih namreč ne vplivajo.

Za začetek sem za povezovanje na vir podatkov namesto *.NET Data Provider*-ja uporabil *SQL Native Provider*-ja. Rezultate testiranja prikazuje tabela 36.

	Proces.
Meritev	43min 42s
Izboljšanje	4%

Tabela 36. Rezultati testiranja po uporabi SQL Native Provider-ja (čas).

Nato sem testiral procesiranje kocke. Kocko lahko procesiramo z eno samo operacijo, in sicer *ProcessFull*, ali pa uporabimo dve operaciji zapored (*ProcessData* in *ProcessIndexes*). Tabela 37 prikazuje rezultate testiranja obeh možnosti.

	Proces.
ProcessFull	30min 13s
ProcessData, ProcessIndexes	6min 39s + 24min 12s = 30min 51s

Tabela 37. Rezultati testiranja načinov procesiranja kocke (čas).

V naslednjem koraku sem meril čase procesiranja glede na število poslov, ki se izvajajo vzporedno. Nastavitev *Maximum parallel tasks* sem testiral za vrednosti 2, 4, 6 in 8. Tabela 38 prikazuje rezultate testiranja. Ker smo najboljši čas dosegli pri vrednosti 4, sem v nadaljevanju uporabljal to vrednost.

	2	4	6	8
Meritev	42min 7s	36min 41s	41min 37s	37min 5s
Izboljšanje	4%	19%	5%	18%

Tabela 38. Rezultati testiranja pri različnih številih vzporednih poslov (čas).

5.5.6 Optimizacija posebnih zasnov

Pri optimizaciji posebnih zasnov sem se lotil testiranja izboljšav v primeru uporabe agregacijske funkcije DistinctCount in razmerja mnogo proti mnogo.

V testni zasnovi kocke je vsebovana mera *Število različnih kupcev*, ki uporablja agregacijsko funkcijo DistinctCount. Iz skupine mer PRODAJA sem jo prestavil v ločeno skupino mer. Tabela 39 in tabela 40 prikazujeta rezultate testiranja.

	DWH	OLAP
Meritev	10156 MB	2954 MB
Izboljšanje	0%	-10%

Tabela 39. Rezultati testiranja po optimizaciji DistinctCount mere (velikost).

	Proces.	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Meritev	53min 37s	16	5	16	276	11	16	36	9612	8675	5
Izboljš.	-32%	31%	200%	0%	22%	3%	225%	-1%	180%	171%	200%

Tabela 40. Rezultati testiranja po optimizaciji DistinctCount mere (čas).

V testni kocki je vsebovano tudi razmerje mnogo proti mnogo, ki povezuje tabelo FACT_PRODAJA in DIM_RAZLOG. Za izboljšanje zmogljivosti sem uporabil tehniko *matrike razmerij*, ki je opisana v teoretičnem delu naloge. Tabela 41 in tabela 42 prikazujeta rezultate testiranja.

	DWH	OLAP
Meritev	11001MB	3089 MB
Izboljšanje	-8%	-4%

Tabela 41. Rezultati testiranja po optimizaciji razmerja mnogo proti mnogo (velikost).

	Proces.	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Meritev	48min 41s	16	11	16	271	16	11	31	2073	1791	5
Izboljš.	10%	0%	-50%	2%	2%	-33%	50%	17%	364%	384%	0%

Tabela 42. Rezultati testiranja po optimizaciji razmerja mnogo proti mnogo (čas).

5.5.7 Optimizacija nastavitev strežnika in strojne opreme

V tem sklopu sem najprej poskusil optimizirati uporabo pomnilnika. Nastavitev *Memory\TotalMemoryLimit* sem iz privzete vrednosti 80 dvignil na 85. Nastavitev *Memory\LowMemoryLimit* sem iz privzete vrednosti 75 spustil na 70. Tabela 43 in tabela 44 prikazujeta rezultate testiranja. Ker so rezultati slabši od privzetih nastavitev, sem jih spremenil nazaj na privzete.

	DWH	OLAP
Meritev	11001MB	3089 MB
Izboljšanje	0%	0%

Tabela 43. Rezultati testiranja po spremembi ravnanja s pomnilnikom (velikost).

	Proces.	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Meritev	49min 17s	11	16	31	281	16	16	42	2135	1823	16
Izboljš.	-1%	50%	-31%	-48%	-4%	0%	-31%	-26%	-3%	-2%	-69%

Tabela 44. Rezultati testiranja po spremembi ravnanja s pomnilnikom (čas).

V naslednjem koraku sem povečal število vzporedno izvajanih poslov med procesiranjem in poizvedovanjem. Nastavitev *CoordinatorExecutionMode* sem iz privzete vrednosti -4 spremenil na 12. Nastavitev *ThreadPool\Process\MaxThreads* sem iz privzete vrednosti 10 spremenil na 12. Nastavitev *ThreadPool\Query\MaxThreads* sem iz privzete vrednosti 64 spremenil na 76. Tabela 45 in tabela 46 prikazujeta rezultate testiranja. Ker so rezultati slabši od privzetih nastavitev, sem jih spremenil nazaj na privzete.

	DWH	OLAP
Meritev	11001MB	3089 MB
Izboljšanje	0%	0%

Tabela 45. Rezultati testiranja po povečanju paralelnosti (velikost).

	Proces.	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Meritev	52min 6s	11	16	26	286	16	21	42	2118	1869	3
Izboljš.	-7%	50%	-31%	-38%	-5%	0%	-48%	-26%	-2%	-4%	88%

Tabela 46. Rezultati testiranja po povečanju paralelnosti (čas).

Nato sem se posvetil procesiranju. Z orodjem System Monitor sem preveril koliko se med procesiranjem uporablja začasen prostor na disku. Izkazalo se je, da je prostora v pomnilniku dovolj, zato se začasnega prostora ne uporablja. Nastavitev *OLAP\Process\AggregationMemoryLimitMax* in *OLAP\Process\AggregationMemoryLimitMin* zato nisem spreminjal. Spremenil sem le nastavitev *OLAP\Process\BufferMemoryLimit*. Privzeto vrednost 60 sem spremenil na 45. Tabela 47 prikazuje rezultate testiranja.

	Proces.
Meritev	45min 41s
Izboljš.	7%

Tabela 47. Rezultati testiranja po spremembi BufferMemoryLimit (čas).

5.6 Analiza rezultatov

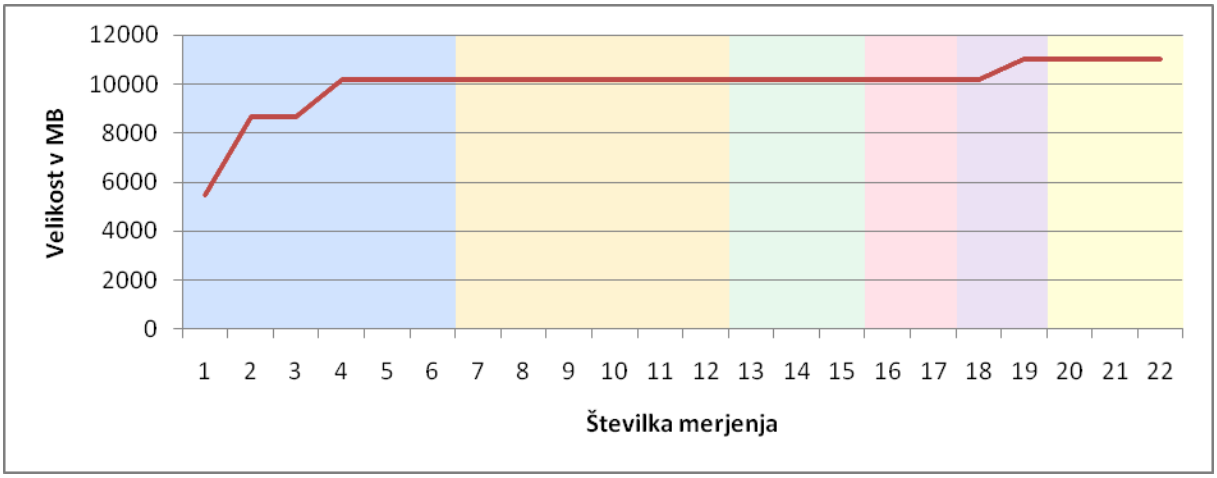
V tem podpoglavju bom podal strnjeno analizo rezultatov testiranja. Vrednosti, ki sem jih tekom testiranja meril, so prikazane v obliki grafikonov. V vsakem grafikonu so z različnimi barvami ozadja označeni posamezni sklopi testiranja. Pomen barv je razložen v tabeli 48.

Barva ozadja	Sklop testiranja
	Podatkovno skladišče
	Zasnova kocke
	Zmogljivost poizvedb
	Zmogljivost procesiranja
	Posebne zasnove
	Strežnik in strojna oprema

Tabela 48. Pomen barv na grafikonih.

5.6.1 Velikost podatkovnega skladišča

Slika 27 prikazuje gibanje velikosti podatkovnega skladišča tekom testiranja. Opazimo, da je v prvem sklopu velikost naraščala. Krivec za to so objekti podatkovne baze (ključi, indeksi), ki smo jih dodajali. Nato smo se posvetili optimizaciji drugih sklopov in velikost je ostala konstantna. Še eno povečanje velikosti smo povzročili na koraku 19. V tem koraku smo z matriko razmerij optimizirali razmerje mnogo proti mnogo. V ta namen smo morali določene tabele podatkovnega skladišča preoblikovati in velikost je narasla.



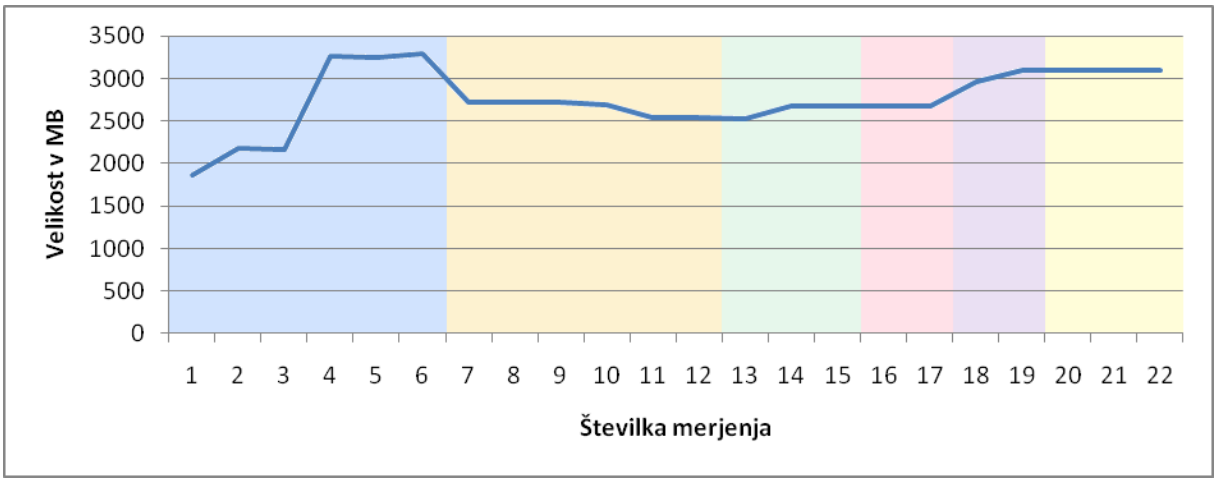
Slika 27. Gibanje velikosti podatkovnega skladišča tekom testiranja.

5.6.2 Velikost večdimenzijske podatkovne baze SSAS

Slika 28 prikazuje gibanje velikosti večdimenzijske podatkovne baze SSAS tekom testiranja. Vidi se veliko povezanost z velikostjo podatkovnega skladišča. Ko je naraščala velikost podatkovnega skladišča, je naraščala tudi velikost večdimenzijske podatkovne baze. Ta povezanost je povsem logična, kajti SSAS tekom procesiranja podatke prenese iz podatkovnega skladišča in jih shrani v večdimenzijsko podatkovno bazo.

Opazimo tudi, da je bil sklop korakov, ki je bil namenjen optimizaciji zasnove kocke, prostorsko učinkovit. Tekom tega sklopa se je velikost večdimenzijske podatkovne baze ves čas zmanjševala.

Do povečanja velikosti je prišlo tudi v koraku 14. V tem koraku smo v kocki ustvarili agregacije. Ker z njimi ustvarimo nove strukture in dodatne podatke, je povečanje velikosti pričakovano.



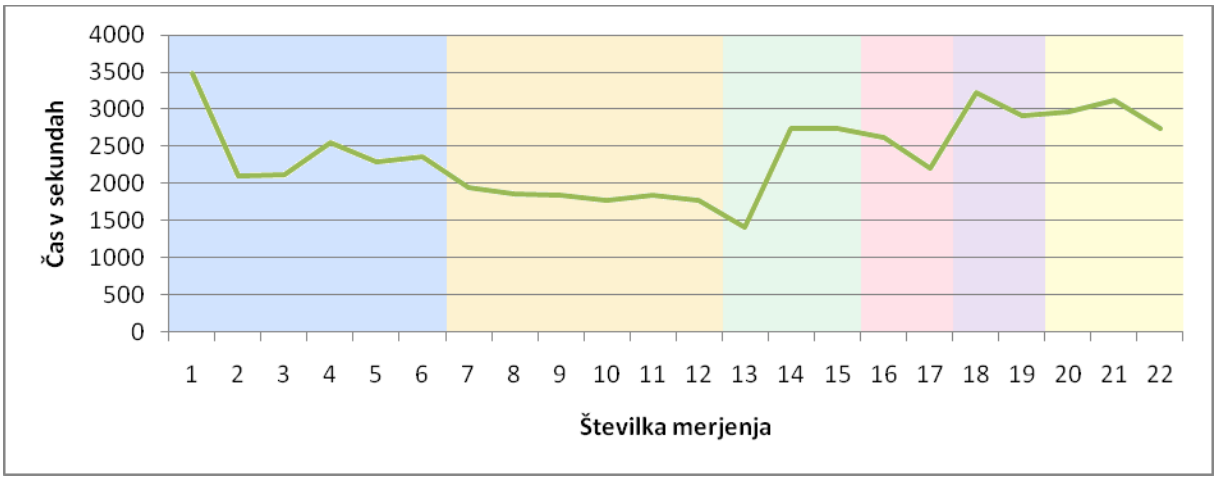
Slika 28. Gibanje velikosti večdimenzijske podatkovne baze SSAS tekom testiranja.

5.6.3 Čas procesiranja

Slika 29 prikazuje gibanje časa procesiranja tekom testiranja. Največja zmanjšanja smo dosegli z uporabo pravilnih podatkovnih tipov v podatkovnem skladišču (korak 2), z uporabo particij (korak 13), z ročno nastavljeno vrednostjo *Maximum parallel tasks* (korak 17) in z matriko razmerij (korak 19). Iz tega lahko sklepamo, da se čas procesiranja največ zmanjša takrat, kadar zmanjšamo količino podatkov v podatkovnem skladišču, pohitrimo branje podatkov iz podatkovnega skladišča ali poskrbimo za primerno vzporednost procesiranja.

Opazimo, da pravilna zasnova kocke tudi pomembno prispeva k zmanjšanju časa procesiranja. Tekom 2. sklopa se je čas procesiranja počasi krajšal.

Po drugi strani je uporaba agregacij izjemno podaljšala čas procesiranja (korak 14). To je razumljivo, kajti strežnik je moral poleg vseh podatkov sprocesirati še vse ustvarjene agregacije in shraniti agregirane podatke. Kot bomo videli pri času izvajanja poizvedb, je tam pohitritev zaradi agregacij tako velika, da odtehta slabost, ki jo predstavlja podaljšan čas procesiranja.



Slika 29. Gibanje časa procesiranja tekom testiranja.

5.6.4 Čas izvajanja poizvedb

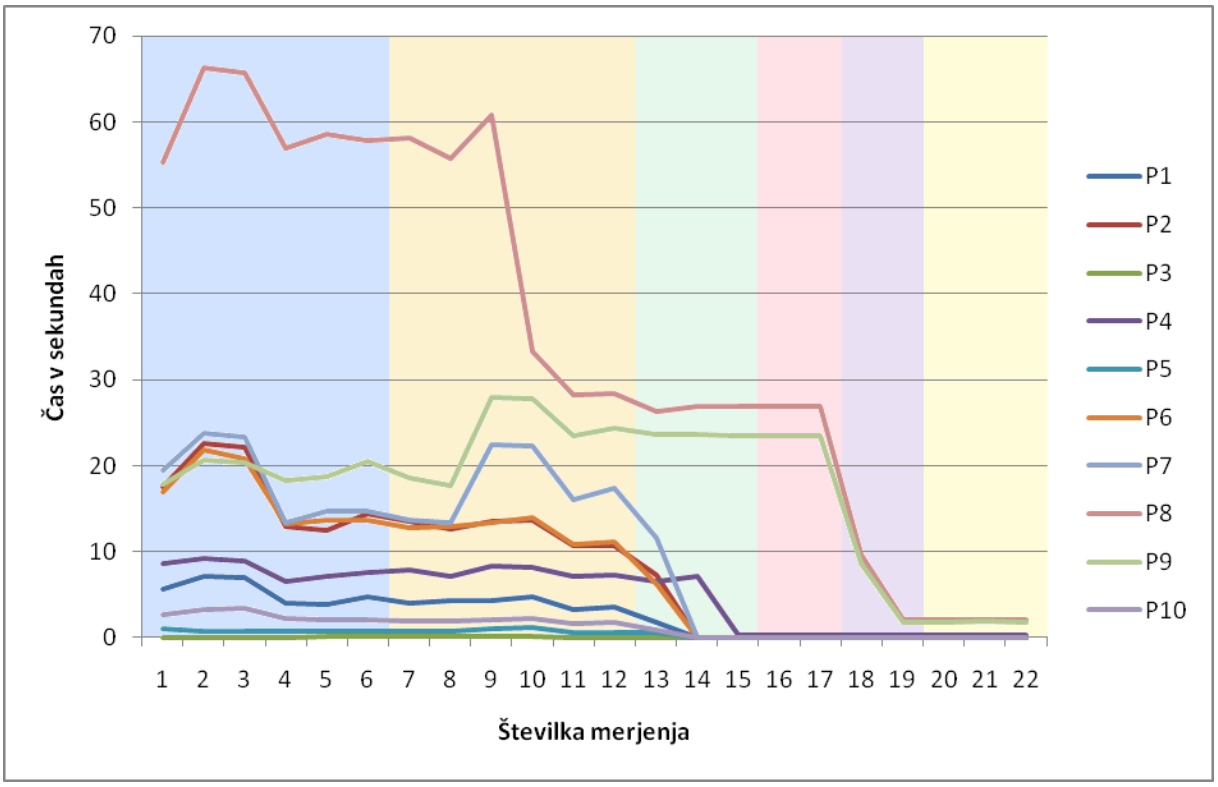
Slika 30 prikazuje gibanje časa izvajanja poizvedb tekom testiranja. Vidimo, da je uporaba particij in agregacij (koraka 13 in 14) najučinkovitejša metoda optimizacije poizvedb. Skoraj vsem poizvedbam se je čas izvajanja v teh dveh korakih močno skrajšal. Pri nekaterih poizvedbah se je čas izvajanja skrajšal na 1% prejšnjega časa, kar je izjemna pohitritev.

Poleg uporabe particij in agregacij sta bili zelo učinkoviti tudi uporaba ključev (korak 4) in odstranitev odvečnih mer (korak 11). Prav vse poizvedbe so v teh korakih zabeležile znatno izboljšanje.

Ločeno velja omeniti poizvedbe nad posebnimi zasnovami, predvsem poizvedbi 8 in 9, ki poizvedujeta podatke iz zasnove z razmerjem mnogo proti mnogo. Ti poizvedbi sta največ pridobili na korakih 18 in 19, torej v sklopu, ki je bil namenjen prav optimizaciji takega tipa poizvedb. Tehnika matrike razmerij (korak 19) se je izkazala za učinkovito. Presenetljivo smo dosegli zelo veliko izboljšanje tudi na koraku 18, kjer smo mero, ki uporablja agregacijsko funkcijo *DistinctCount*, premaknili v ločeno skupino mer. Najverjetneje je vzrok za to način, po katerem strežnik SSAS iz kocke pridobiva podatke o merah. Ko neka poizvedba zahteva podatke o določeni meri, strežnik običajno poleg izbrane mere prebere tudi vse ostale, ki se nahajajo v isti skupini mer. Mera tipa *DistinctCount* je torej močno zavirala izvajanje poizvedb 8 in 9, čeprav poizvedbi dejansko nista zahtevali podatkov iz te mere. Ko smo jo premaknili v ločeno skupino mer, na poizvedbi ni imela več nobenega vpliva.

Poizvedba 8 je zelo veliko (preko 25 sekund) pridobila tudi na koraku 10, kjer smo določenim atributom spremenili nastavitve shranjevanja in prikazovanja.

Opazno poslabšanje časa izvajanja poizvedb je vidno na koraku 9. V tem koraku smo v dimenzijah definirali hierarhije med atributi. Očitno je pridobivanje podatkov iz hierarhij počasnejše kot pridobivanje podatkov neposredno iz atributov. Menim pa, da je izgradnja hierarhij pomembna za kasnejše korake, torej se jo vseeno izplača vključiti v postopek optimizacije.



Slika 30. Gibanje časa izvajanja poizvedb tekom testiranja.

5.6.5 Dodatna opažanja

Testiranje je pokazalo, da je izbira MOLAP-a za način shranjevanja mer in agregacij optimalna odločitev. Izvajanje poizvedb v načinu ROLAP ali HOLAP lahko traja tudi do desetkrat dlje od načina MOLAP.

Med testiranjem smo preizkusili tudi korak gretja predpomnilnika, ki ga zaradi določenih specifik nismo vključili v celoten proces optimizacije. Kljub temu smo videli, da je to ena izmed najbolj učinkovitih metod optimizacije poizvedb. Vseh deset testnih poizvedb se je izvedlo v trenutku, natančneje v 10 ms ali manj.

V teoretičnem delu naloge smo omenili, da so privzete nastavitve strežnika SSAS običajno dobre in zadostujejo našim potrebam. V praktičnem delu se je to potrdilo. S spreminjanjem nastavitev nismo dosegli nobenega opaznega izboljšanja. Velikost podatkovnega skladišča, velikost večdimenzijske podatkovne baze SSAS in čas izvajanja poizvedb so ostali praktično nespremenjeni, le čas procesiranja se je nekoliko skrajšal.

6 Zaključek

V diplomski nalogi sem predstavil sisteme OLAP, razloge za njihovo uvedbo, glavne značilnosti in prednosti. Sistemi OLAP so namenjeni podpori sprejemanju odločitev na različnih nivojih organizacije. Nosilci odločanja v organizacijah sprejemajo veliko odločitev in mnoge od njih imajo lahko pomemben vpliv na uspešnost poslovanja. Zato morajo biti sistemi OLAP dovolj zmogljivi, da ne ovirajo procesa odločanja. Zmogljivost merimo predvsem v času, ki je potreben za procesiranje podatkov in času, ki je potreben, da dobimo rezultate poizvedb.

Produkt SQL Server 2005, ki ga je razvilo podjetje Microsoft, je sestavljen iz več komponent. Komponenta SQL Server Analysis Services omogoča gradnjo sistemov OLAP. Opisal sem njeno arhitekturo procesiranja in izvajanja poizvedb. Med procesiranjem se ustvari več različnih shramb in indeksov, med poizvedovanjem pa je potrebno pridobiti zahtevane podatke.

V teoretičnem delu naloge sem naštel in razložil možne načine optimizacije zmogljivosti sistemov OLAP na šest različnih nivojih. V praktičnem delu sem večino naštetih metod preizkusil na dejanskem sistemu, zgrajenem iz realnih podatkov neke organizacije. Nato sem dobljene rezultate tudi analiziral.

Izkazalo se je, da z optimizacijo lahko pridobimo zelo veliko. Čeprav sem med testiranjem med drugim sledil tudi gibanju velikosti podatkovnega skladišča in večdimenzijske podatkovne baze SSAS, tema vrednostma ne dajem posebnega poudarka. Trdi diski so relativno poceni in če na račun povečanja velikosti podatkovne baze pridobimo na času procesiranja ali poizvedovanja, se nedvomno izplača vložiti denar za nakup potrebne količine diskov.

Količini, ki sem ju meril v praktičnem delu diplomske naloge in jima dajem največji pomen, sta čas procesiranja in čas poizvedovanja, pri čemer ima čas poizvedovanja prednost pred časom procesiranja. Čas poizvedovanja se v organizacijah največkrat izkaže za nezadostnega. Če bi se moral odločiti za en sam način optimizacije, bi priporočil uporabo agregacij skupaj s particijami. Particije močno zmanjšajo čas procesiranja in čas poizvedovanja, agregacije pa na račun procesiranja še dodatno zelo skrajšajo čas poizvedovanja. Particije in agregacije bi zato imenoval kar temelj optimizacije sistemov OLAP. Tudi z nekaterimi drugimi načini optimizacije sem dosegel opazna izboljšanja. Čas izvajanja poizvedb sta pomembno izboljšali odstranitev odvečnih mer in dobro definirana zasnova kocke. Zasnova kocke je poleg definiranja ustreznih podatkovnih tipov v podatkovnem skladišču in definiranja števila poslov, ki se izvajajo vzporedno, pomembna tudi za izboljšanje časa procesiranja.

Pred pričetkom optimizacije je večina testnih poizvedb trajala več kot 5 sekund, nekatere tudi 20 sekund ali več. Po definiciji zmogljivosti, ki sem jo podal v uvodnem poglavju, ugotovimo, da sistem v začetku ni bil ravno zmogljiv. Z uporabo optimizacijskih postopkov sem dosegel, da je večina poizvedb trajala manj kot 1 sekundo, dve poizvedbi pa približno 2 sekundi. Taki časi poizvedb povsem ustrezajo definiciji zmogljivega sistema OLAP.

Dodatna izboljšanja zmogljivosti poizvedovanja dosežemo z gretjem predpomnilnika. Pomanjkljivost te metode je v tem, da moramo čim bolje poznati vsebino poizvedb, ki jih želimo optimizirati. V nasprotnem primeru ni nujno, da sploh dosežemo kakšno izboljšanje.

Zavedati se moramo, da v dejanskih organizacijah srečamo ogromno različnih situacij, struktur podatkov in zahtev. V vseh primerih zagotovo ne bi mogli doseči takšnega izboljšanja zmogljivosti poizvedb, kot sem ga dosegel sam. Glavne smernice podane v tej diplomski nalogi držijo ne glede na situacijo, vendar lahko izboljšanje na posameznem koraku optimizacije variira od primera do primera. Kadar nam struktura podatkov ne omogoča doseči dobrih rezultatov, je smiselno premisliti o preoblikovanju zbiranja in prikazovanja podatkov. V večini primerov lahko ustvarimo takšno zasnovo, ki bo prijazna do optimizacije.

V nadaljevanju bi lahko preizkusil še ostale metode optimizacije, ki sem jih omenil v teoretičnem delu diplomske naloge, vendar jih zaradi takšnih ali drugačnih razlogov nisem testiral. Nekaterih metod nisem mogel testirati, ker mi tega testni podatki niso omogočali, pri drugih pa sem bil omejen s strojnimi viri. Zanimivo bi se bilo dodatno posvetiti testiranju strojnih zahtev v okolju z večjo količino podatkov, kot sem jo uporabil v tej diplomski nalogi. Ugotavljal bi kolikšne pridobitve dosežemo s povečevanjem števila procesorjev, strežnikov ipd.

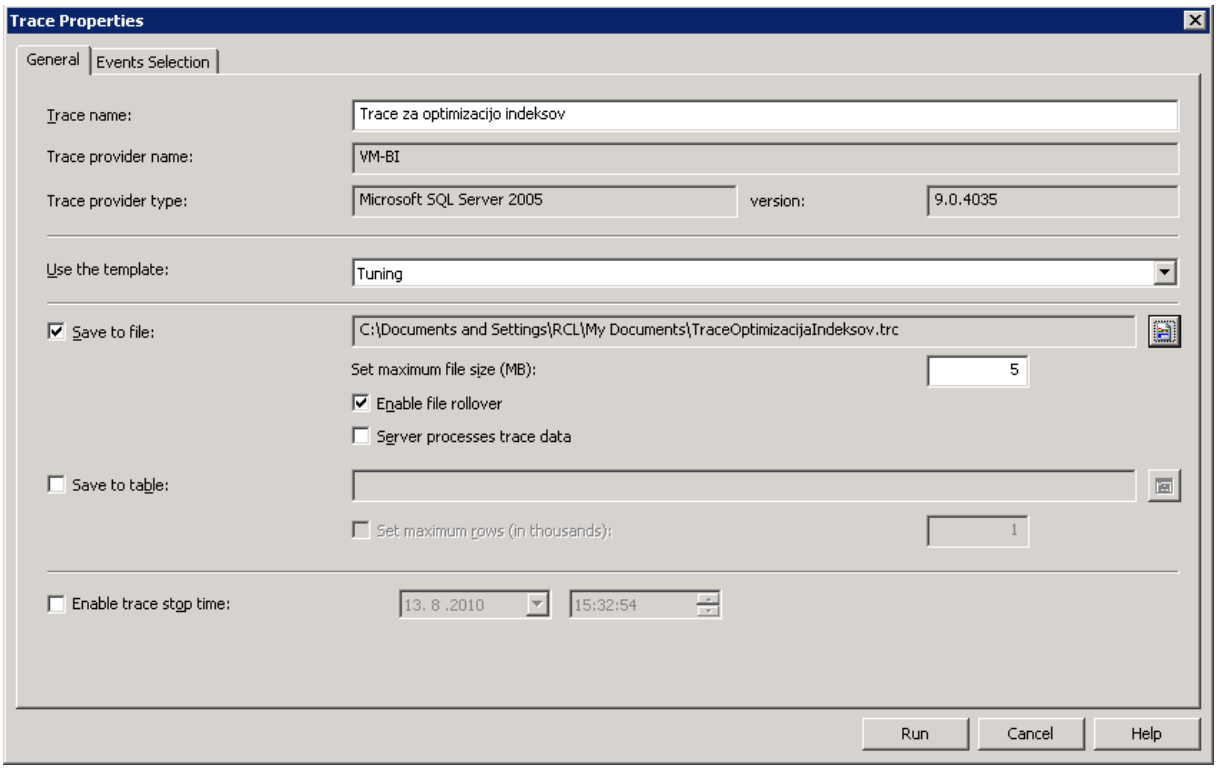
Na koncu velja omeniti, da je na tržišču že novejša verzija produkta SQL Server. V diplomski nalogi sem uporabljal verzijo 2005, medtem ko je trenutno najnovejša verzija 2008 R2. Bistvenih razlik med verzijama ni. Razlike so vidne predvsem v prijaznejšem uporabniškem vmesniku ter manjših izboljšavah nekaterih algoritmov, kar morda lahko vpliva tudi na boljši končni rezultat optimizacije. Vendar je verzija 2005 v Sloveniji še vedno močno prisotna.

Dodatek A

Uporaba programov SQL Server Profiler in Database Engine Tuning Advisor za optimizacijo postavitve indeksov v podatkovnem skladišču

Microsoft SQL Server Profiler je orodje z grafičnim vmesnikom, ki zna slediti dogodkom, ki se dogajajo na podatkovni bazi SQL. Zbrane podatke lahko shranimo v datoteko ali tabelo SQL in jih kasneje analiziramo.

Zaženemo ga s klikom na meni Start, nato izberemo Vsi programi, Microsoft SQL Server 2005, Performance Tools, SQL Server Profiler. Ko se program odpre, moramo ustvariti in definirati novo sledenje (angl. trace). Izberemo File, New Trace. Prikaže se okno z nazivom Connect to Server, kjer vpišemo potrebne podatke za povezovanje na strežnik, ki mu želimo slediti. Če se uspešno povežemo, se prikaže okno Trace Properties. Slika 31 prikazuje zaslonski posnetek Trace Properties okna.



Slika 31. Nastavitveno okno sledenja v SQL Server Profiler-ju.

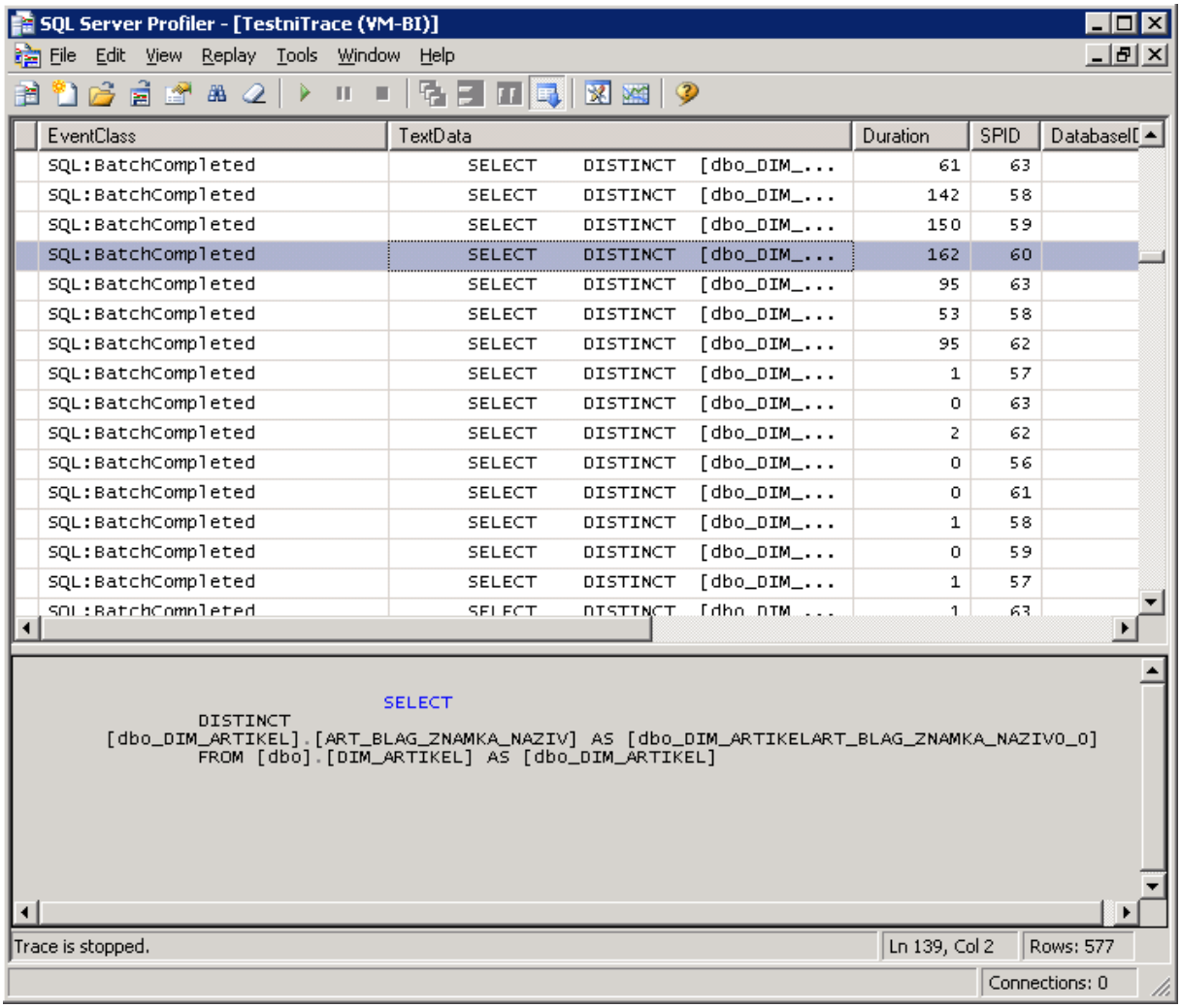
Okno Trace Properties je nastavitveno okno. Ker želimo spremljati dogodke, na podlagi katerih bomo optimizirali postavitev indeksov, izberemo sledeče nastavitve:

- Trace Name: V to polje vpišemo želeno ime sledenja.
- Use the template: Iz spustnega menija izberemo vnaprej pripravljeno predlogo z imenom *Tuning*. Ta predloga je že definirana tako, da sledi tistim dogodkom, ki so potrebni za optimizacijo indeksov.

- Save to file: Naredimo kljukico pri tej nastavitvi in v sosednjem polju izberemo pot in ime datoteke, kamor naj se shranijo rezultati sledenja.
- Ostale nastavitve lahko pustimo na privzetih vrednostih.

Ko smo vse nastavili, kliknemo gumb Run in s tem se začnejo vsi ustrezni dogodki shranjevati. Ker želimo podatkovno skladišče optimizirati za potrebe procesiranja, moramo sedaj izvesti procesiranje SSAS kocke. S tem se v zgoraj definirano datoteko shranijo dogodki, ki se zgodijo med procesiranjem.

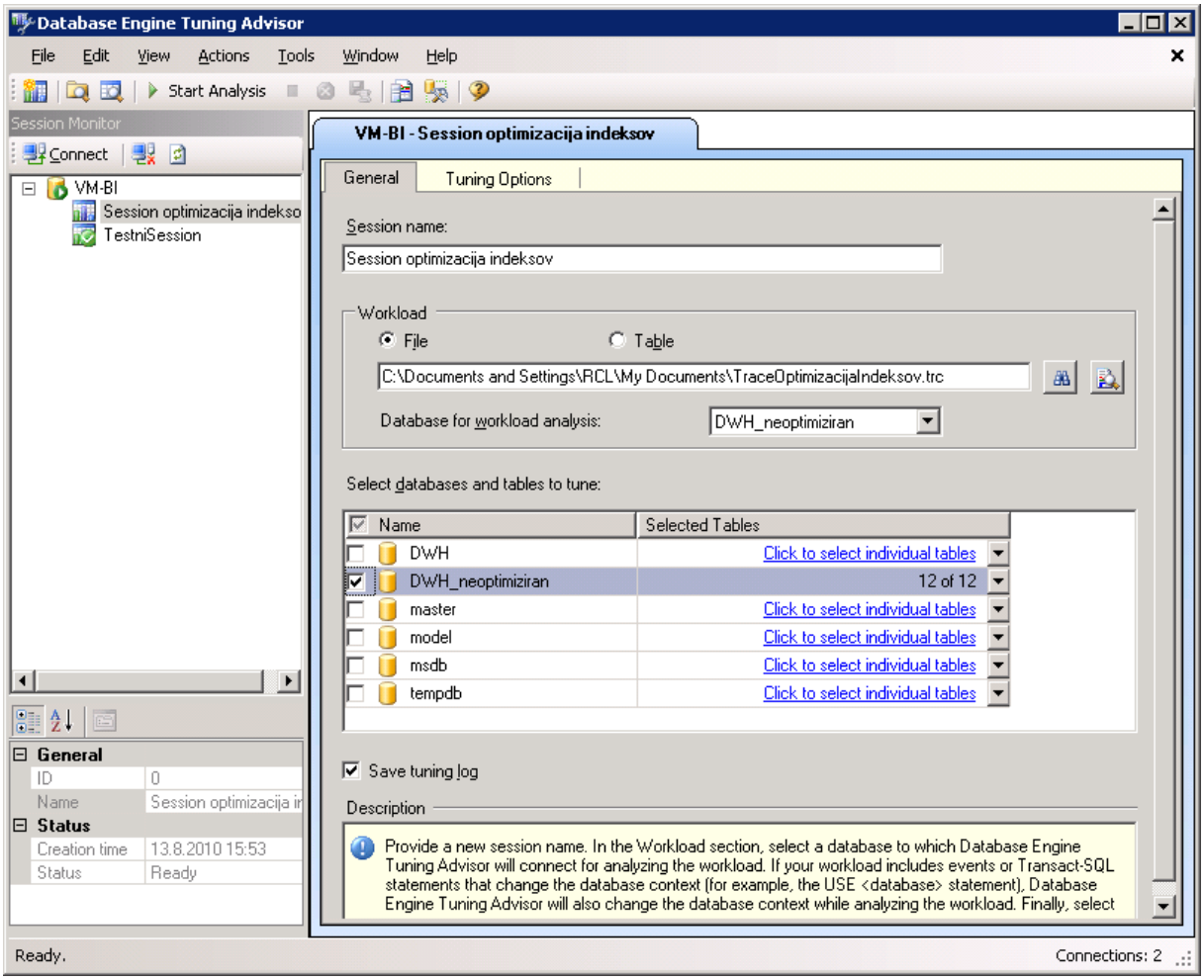
Ko se procesiranje kocke konča, v SQL Server Profilerju kliknemo na gumb Stop selected trace. Program neha slediti dogodkom na strežniku. Slika 32 prikazuje zaslonski posnetek SQL Server Profiler-ja po tem, ko smo sledenje ustavili. V glavnem oknu je seznam vseh dogodkov, ki so bili zabeleženi.



Slika 32. Glavno okno SQL Server Profiler-ja po končanem sledenju.

Database Engine Tuning Advisor je orodje, ki nam predlaga načine, s katerimi lahko skrajšamo čase poizvedb na podatkovni bazi. Program preuči izvajanje danih poizvedb in predlaga spremembe v povezavi z indeksi, indeksiranimi pogledi in particijami.

Zaženemo ga s klikom na meni Start, nato izberemo Vsi programi, Microsoft SQL Server 2005, Performance Tools, Database Engine Tuning Advisor. Ob zagonu nas program vpraša za strežnik, na katerega se želimo povezati. Izpolnimo potrebne podatke in se povežemo. Slika 33 prikazuje osnovno okno, ki se prikaže po uspešni povezavi na strežnik.

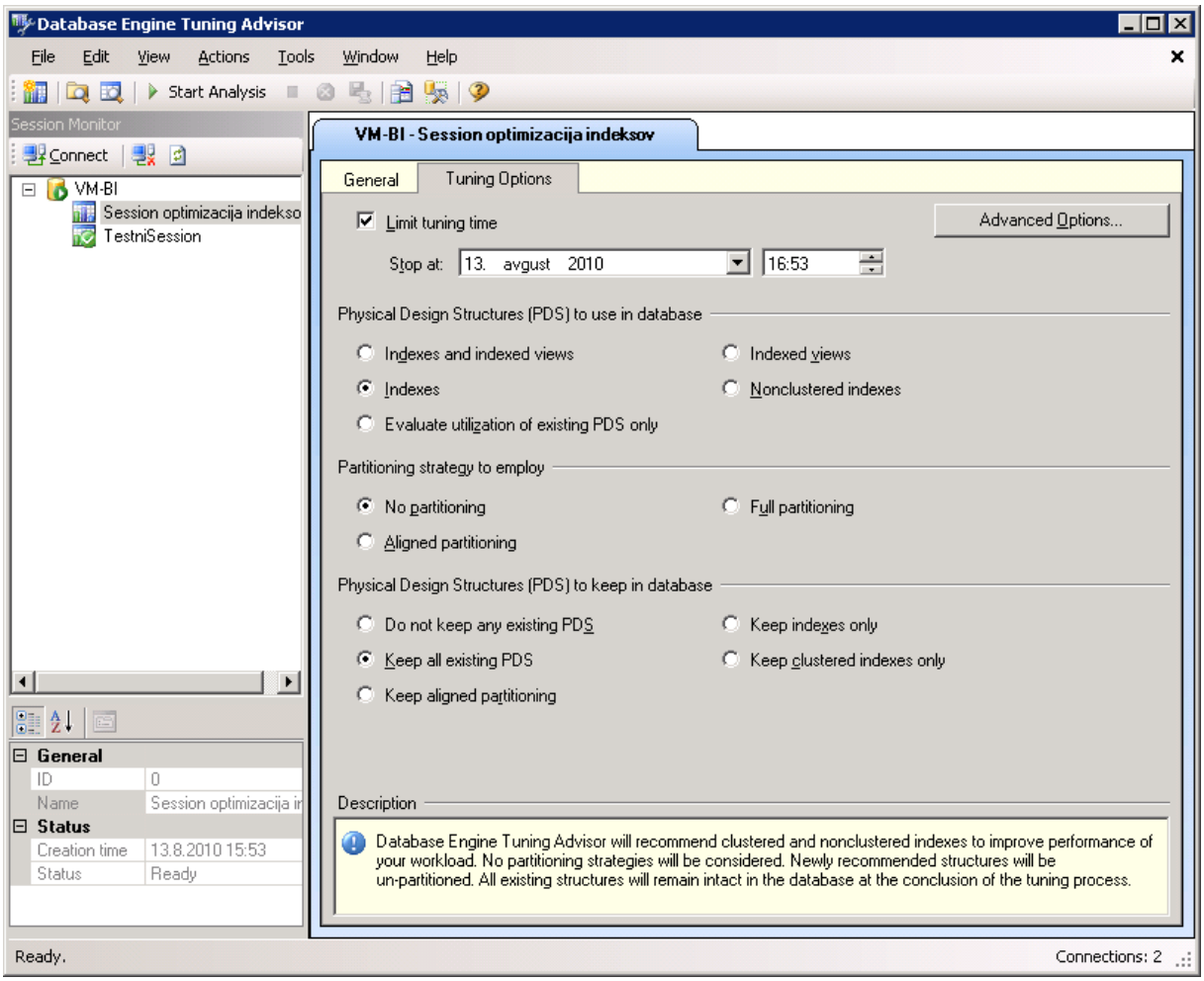


Slika 33. Glavno okno Database Engine Tuning Advisor-ja.

Na levi strani je razdelek Session Monitor, kjer je naveden seznam vseh dosedanjih sej, ki smo jih izvedli. Desna stran programa je sestavljena iz več zavihkov. Prvi zavihek se imenuje General in za naše potrebe ga izpolnimo sledeče:

- Session Name: Vpišemo želeno ime trenutne seje.
- Workload: V tem razdelku izberemo izbiro File ter v spodaj ležečem polju izberemo datoteko, v katero smo shranili dogodke v SQL Server Profiler-ju. V seznamu Database for workload analysis izberemo podatkovno bazo, ki predstavlja naše podatkovno skladišče.
- Select databases and tables to tune: Pred podatkovnim skladiščem naredimo kljukico.
- Ostale nastavitve pustimo privzete.

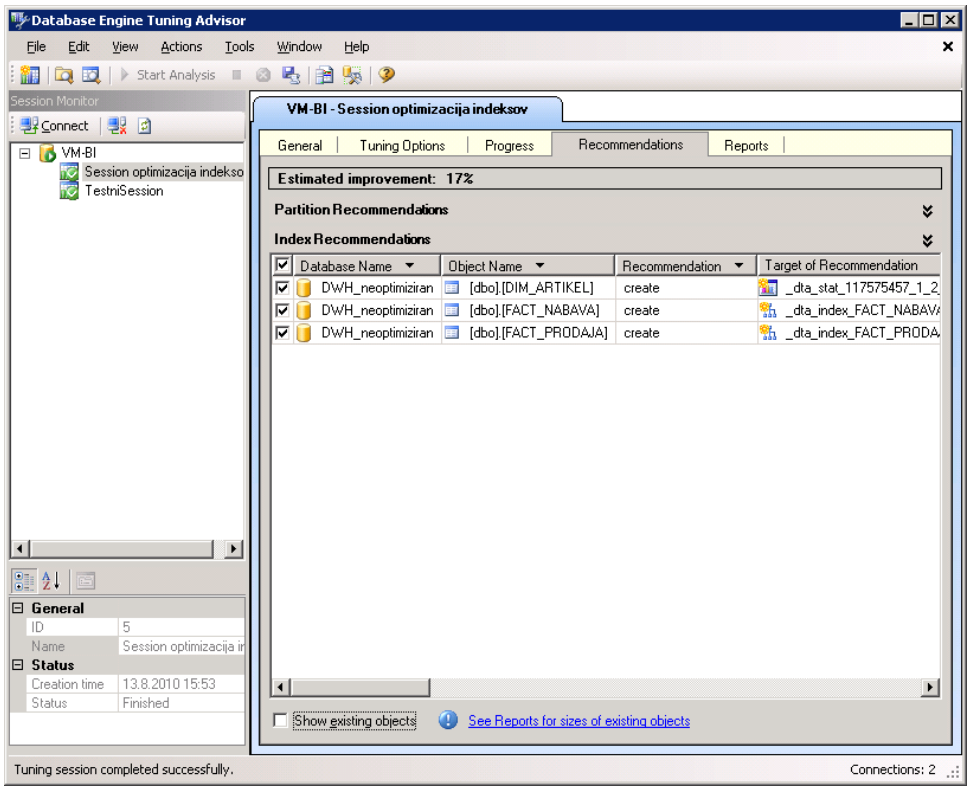
Naslednji zavihek se imenuje Tuning Options in je prikazan na sliki 34. Nastavitve na tem zavihku pustimo kar takšne, kot so privzeto nastavljene.



Slika 34. Zavihek Tuning Options.

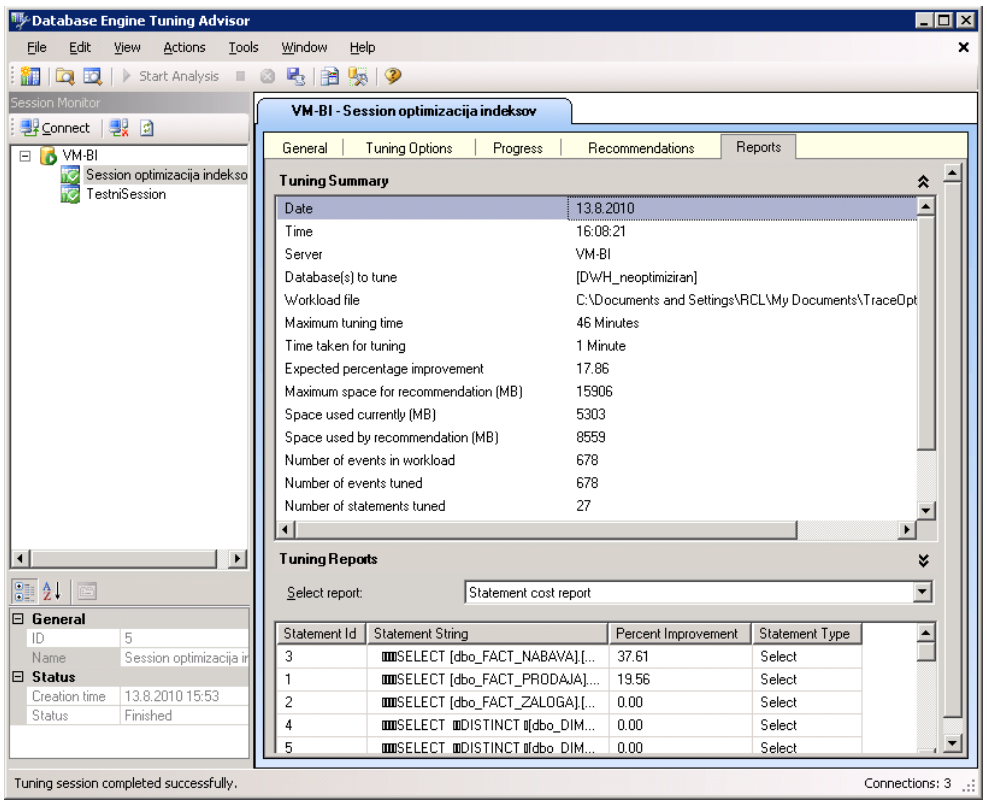
Nato kliknemo gumb Start Analysis in program začne z analizo. Med analiziranjem lahko spremljamo potek na zavihku Progress.

Ko program zaključi z analizo, se pojavita zavihka Recommendations in Reports. Zavihek Recommendations je prikazan na sliki 35 in vsebuje priporočila za kreiranje novih indeksov ter procentualno oceno izboljšanja. Če želimo uveljaviti predlagana priporočila, v glavnem meniju izberemo Actions, Apply Recommendations.



Slika 35. Zavihek Recommendations.

Zavihek Reports je prikazan na sliki 36 in vsebuje več poročil o trenutnem stanju podatkovne baze in o predlaganih izboljšavah.



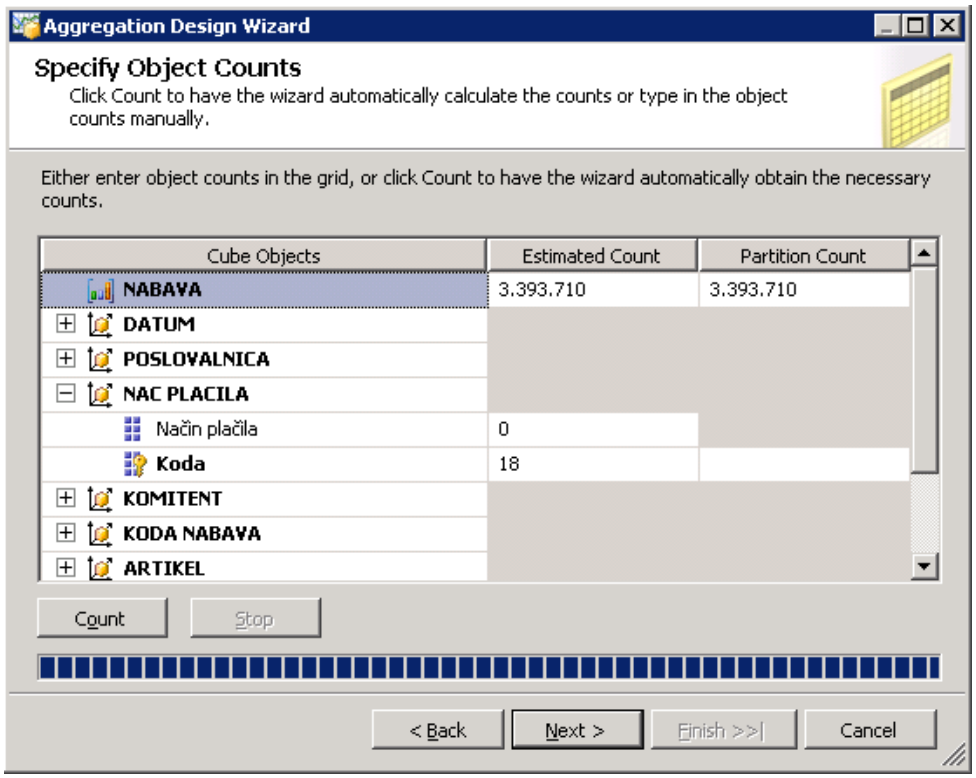
Slika 36. Zavihek Reports.

Dodatek B

Čarovnik za oblikovanje agregacij

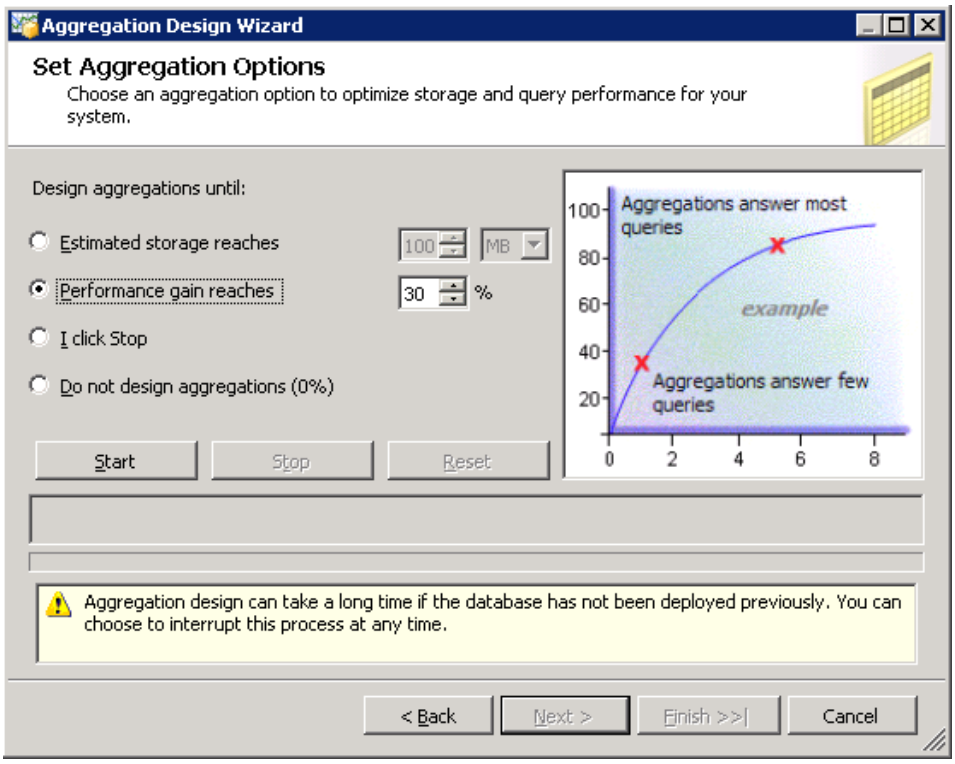
Čarovnik za oblikovanje agregacij (angl. Aggregation Design Wizard) je na voljo v orodju BIDS. Uporabimo ga, ko smo SSAS kocko že ustvarili, nastavili in postavili. Zaženemo ga tako, da v kocki kliknemo na zavihek Partitions, izberemo želeno particijo in kliknemo na povezavo Design Aggregations ... S klikom na to povezavo se čarovnik zažene.

Prvo okno čarovnika je samo pozdravno okno, zato kliknemo na gumb Next. Prikaže se okno za izbiro načina shranjevanja mer in agregacij. Načeloma smo način že predhodno določili, zato kliknemo na gumb Next. V naslednjem koraku specificiramo število članov posameznih objektov v kocki. Slika 37 prikazuje zaslonski posnetek tega koraka. Specificirati je potrebno število Estimated Count, ki predstavlja število vseh zapisov v posameznih tabelah in število Partition Count, ki predstavlja število zapisov v posameznih tabelah za trenutno izbrano particijo. Števila lahko vpišemo ročno ali pa kliknemo na gumb Count in jih čarovnik prešteje namesto nas. V primeru, da so števila predhodno že vpisana, gumb Count ne deluje. To pride prav v situacijah, ko kocko razvijamo v testnem okolju, nato pa jo prestavimo v produkcijsko okolje in ne želimo, da bi nam čarovnik povozil nastavljene vrednosti. V tem koraku vidimo tudi katere atribute je SSAS izbral kot kandidate za agregacije. Atributi, ki so izbrani, so napisani z odebeljeno pisavo.



Slika 37. Korak, kjer določimo števila objektov.

Ko je število objektov nastavljeno, kliknemo na gumb Next. Odpre se naslednji korak, kjer določimo nastavitve izdelave agregacij. Slika 38 prikazuje zaslonski posnetek tega koraka. Nastaviti moramo omejitve, po katerih bomo izdelali agregacije. Omejitve določajo količino ustvarjenih agregacij. Najpogosteje uporabimo prvi dve možnosti. Prva možnost deluje tako, da določimo maksimalno količino prostora, ki ga agregacije lahko zavzamejo. Druga možnost deluje tako, da v odstotkih določimo kolikšno izboljšanje zmogljivosti želimo pridobiti. Ko izberemo eno izmed možnosti, kliknemo na gumb Start in čarovnik začne ustvarjati zasnovo agregacij. Z ustvarjanjem konča, ko doseže nastavljeno omejitev. Na desni strani okna programa je narisan graf, ki na osi X prikazuje prostor, ki ga definirane agregacije zavzemajo, na osi Y pa oceno izboljšanja zmogljivosti v odstotkih.



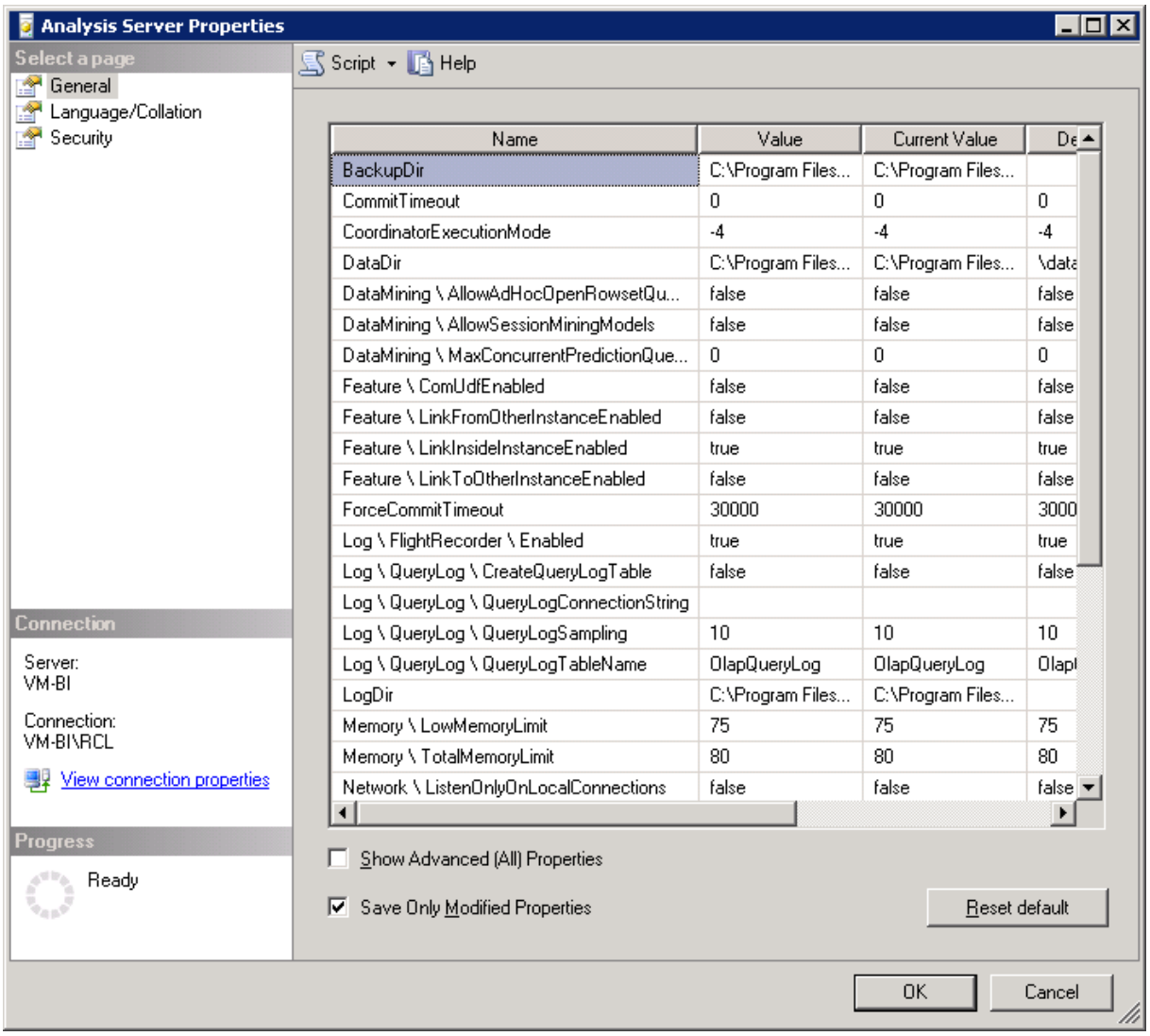
Slika 38. Korak, kjer določimo nastavitve izdelave agregacij.

Ko čarovnik konča s tem korakom, kliknemo na gumb Next. V zadnjem koraku se odločimo ali želimo ustvarjene agregacije trenutno samo shraniti ali jih želimo tudi takoj postaviti na strežnik. S klikom na gumb Finish zaključimo z uporabo čarovnika za oblikovanje agregacij.

Dodatek C

Čarovnik za optimizacijo na podlagi uporabe

Čarovnik za optimizacijo na podlagi uporabe (angl. Usage-Based Optimization Wizard) ustvari agregacije na podlagi uporabniških zahtev in s tem zagotovi, da bo zmogljivost teh poizvedb izboljšana. Agregacije naj bi bile zaradi tega zasnovane še učinkovitejše kot pri čarovniku za oblikovanje agregacij. Ker želimo analizirati poizvedbe uporabnikov, jih je potrebno nekam shranjevati. Shranjujemo jih v tabelo na strežniku SQL. Privzeto SSAS ne shranjuje poizvedb, zato moramo shranjevanje pred uporabo čarovnika omogočiti. To storimo z orodjem SSMS. Povežemo se na SSAS strežnik. Nato z desnim miškinim gumbom kliknemo na ime strežnika in izberemo Properties. Odpre se okno, ki je prikazano na sliki 39.



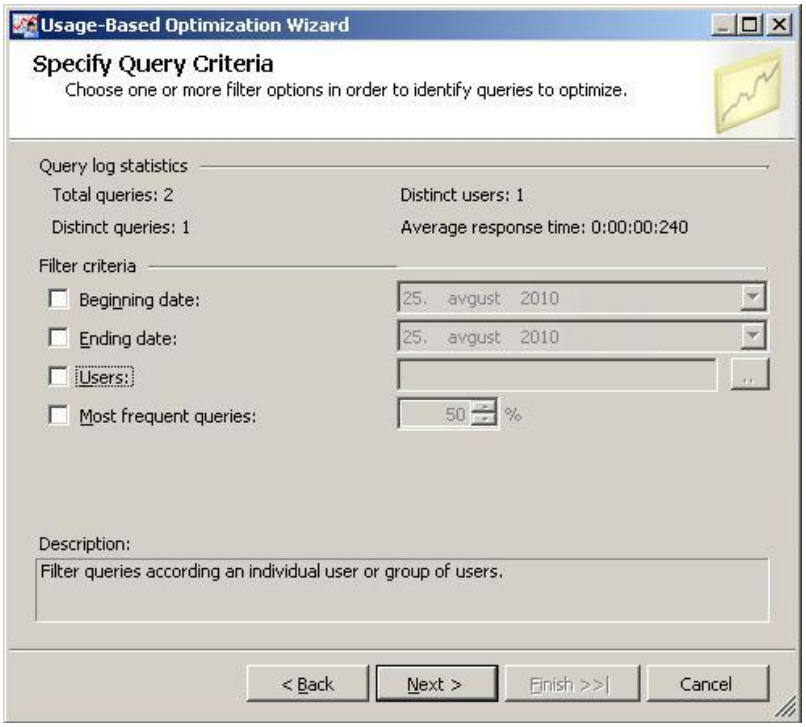
Slika 39. Nastavitve SSAS strežnika.

V tem oknu nastavimo nastavitev »Log \ QueryLog \ QueryLogConnectionString«, in sicer tako, da kaže na relacijsko podatkovno bazo, kjer bomo shranjevali poizvedbe uporabnikov. V vnosnem polju kliknemo na gumb in odpre se okno, kjer lahko te podatke vnesemo.

Če želimo, lahko spremenimo tudi nastavitev »Log \ QueryLog \ QueryLogSampling«. Privzeto je določena na 10, kar pomeni, da se bo zabeležila vsaka deseta poizvedba uporabnikov. Če želimo, da se zabeleži vsaka poizvedba, nastavitev spremenimo na 1.

Nato preverimo, da je nastavitev »Log \ QueryLog \ CreateQueryLogTable« nastavljena na true, potrdimo nastavitve strežnika s klikom na gumb OK in strežnik ponovno zaženemo. Po ponovnem zagonu se v podatkovni bazi, ki smo jo določili, ustvari tabela z imenom OLAPQueryLog. V njej se shranjujejo različne informacije od uporabniškega imena do časovnega žiga in podkock, ki so uporabljene v posamezni poizvedbi.

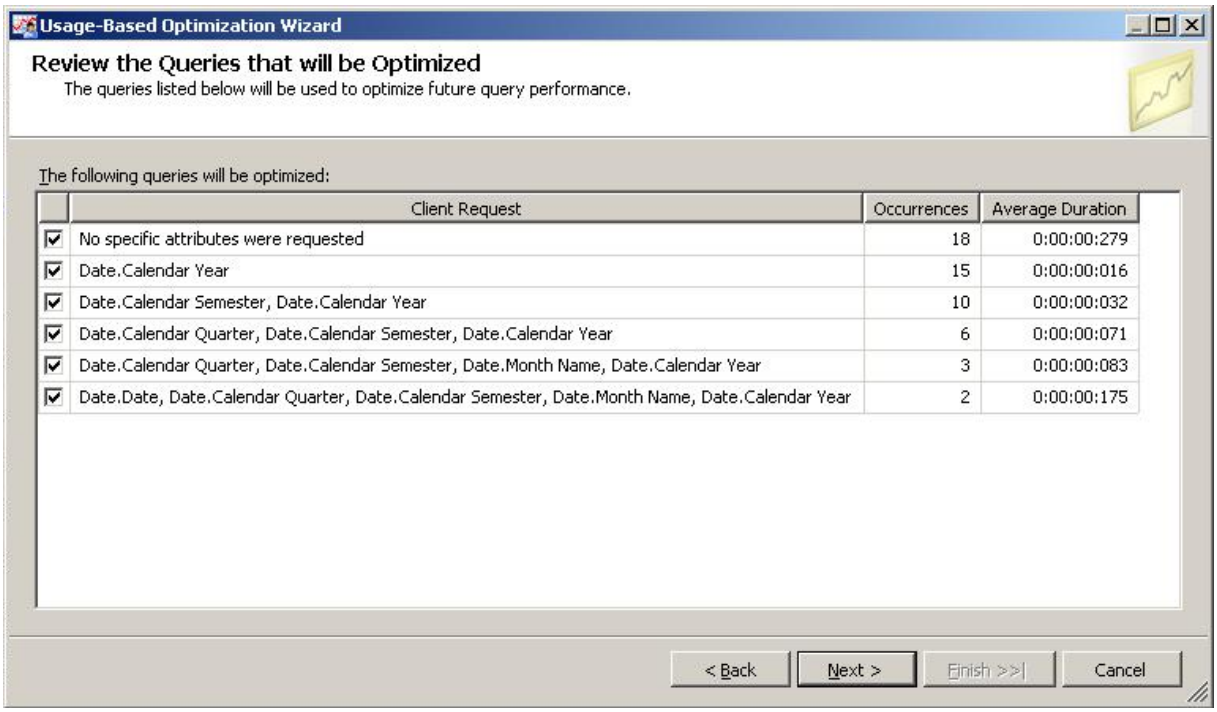
Potem pustimo kocko nekaj časa v uporabi. Medtem, ko uporabniki izvajajo poizvedbe, se informacije o poizvedbah beležijo v tabelo v podatkovni bazi. Ko menimo, da smo nabrali dovolj reprezentativnih poizvedb, je čas za uporabo čarovnika za optimizacijo na podlagi uporabe. Zaženemo ga v orodju BIDS, tako da v kocki odpremo zavihek Partitions, izberemo želeno particijo in kliknemo na ikono Usage Based Optimization. Prikaže se pozdravno okno čarovnika, kjer kliknemo na gumb Next. V naslednjem koraku lahko filtriramo shranjene poizvedbe. Če ne želimo analizirati vseh, jih lahko omejimo glede na datum, uporabnika ali glede na določen delež najpogostejših poizvedb. Poleg tega so v tem koraku prikazani statistični podatki o shranjenih poizvedbah. Prikazano je število vseh poizvedb, število različnih poizvedb, število različnih uporabnikov in povprečen čas izvajanja poizvedb. Slika 40 prikazuje zaslonski posnetek tega koraka.



Slika 40. Izbira filtra poizvedb za analizo.

Ko določimo filter (lahko pustimo tudi praznega in izberemo vse poizvedbe), kliknemo na gumb Next. Prikaže se nam seznam vseh shranjenih poizvedb, ki ustrezajo izbranemu filtru.

Če želimo, lahko določene poizvedbe izključimo iz analize tako, da odstranimo kljukico pred njimi. Slika 41 prikazuje zaslonski posnetek tega koraka.



Slika 41. Izbira poizvedb za analizo.

Ko imamo želene poizvedbe izbrane, kliknemo na gumb Next. Koraki, ki sledijo, so identični korakom, ki smo jih srečali že pri čarovniku za oblikovanje agregacij in so opisani v dodatku B.

Dodatek Č

Poizvedba SQL za izračun odstotka redukcije pri razmerjih M2M

Pri optimizaciji razmerij M2M z uporabo matrike razmerij si lahko predhodno pomagamo s spodnjo poizvedbo SQL. Poizvedba izračuna pričakovan odstotek redukcije.

```
SELECT (SELECT COUNT(*) FROM [INT MG]) OriginalRecordCount
, COUNT(MatrixKey) * AVG(CAST(KeyCount AS FLOAT))
CompressedRecordCount
, (1 - CAST(COUNT(MatrixKey) * AVG(CAST(KeyCount AS FLOAT)) AS
FLOAT)/(SELECT COUNT(*) FROM [INT MG])) * 100 ReductionPerc
FROM ( SELECT DISTINCT COUNT(*) KeyCount, MatrixKey =
( SELECT [M2M DIM #1 KEYS] AS [data()]
FROM [INT MG]
WHERE [INT MG].[JOIN KEY #1] = s.[JOIN KEY #1]
AND [INT MG].[JOIN KEY #2] = s.[JOIN KEY #2]
ORDER BY [M2M DIM #1 KEYS]
FOR XML PATH ('')  )
FROM [INT MG] s
GROUP BY [JOIN KEY #1], [JOIN KEY #2]
) SUBQ
```

V tabeli 49 so zbrani opisi posameznih delov zgornje poizvedbe.

Del poizvedbe	**Opis**
[INT MG]	Vmesna tabela dejstev, ki je kandidat za matriko razmerij.
[M2M DIM #1 KEYS]	Ključ(i) dimenzije M2M (Predpostavljamo, da sta primarni in tuji ključ enako poimenovana).
[JOIN KEY #1]	Ključ(i), ki povezujejo vmesno tabelo dejstev s podatkovno tabelo dejstev.

Tabela 49. Opis delov poizvedbe, ki izračuna odstotek redukcije razmerij M2M.

Seznam slik

Slika 1. Trije ključi za učinkovito odločanje. 8
Slika 2. Rezultat enostavne poizvedbe MDX. 11
Slika 3. Arhitektura poizvedovanja v SSAS. 13
Slika 4. Shramba atributov dimenzij. 15
Slika 5. Shramba hierarhij. 16
Slika 6. Pridobivanje podatkov o merah. 17
Slika 7. Posli pri procesiranju dimenzij. 18
Slika 8. Primer izvajalnega drevesa. 19
Slika 9. Posli pri procesiranju particij. 20
Slika 10. SQL Server Profiler. 25
Slika 11. Database Engine Tuning Advisor. 26
Slika 12. Relacijski diagram zvezdaste sheme. 27
Slika 13. Relacijski diagram snežinkaste sheme. 27
Slika 14. Privzeta razmerja med atributi. 28
Slika 15. Optimizirana razmerja med atributi. 29
Slika 16. Dimenzije Artikel, Stranka in Datum naročila. 33
Slika 17. Kandidati za agregacije v dimenziji Artikel po uporabi vseh privzetih pravil. 35
Slika 18. Nastavitve procesiranja. 46
Slika 19. Primer referenčnega razmerja. 47
Slika 20. Model razmerja M2M. 48
Slika 21. Stiskanje vmesne tabele dejstev v razmerju M2M. 50
Slika 22. Nastavitve strežnika SSAS. 51
Slika 23. E-R diagram začetne zasnove podatkovnega skladišča. 59
Slika 24. Seznam skupin mer in dimenzij. 63
Slika 25. Uporaba dimenzij v kocki. 63
Slika 26. Primer poročila SSRS, ki služi gretju predpomnilnika. 69
Slika 27. Gibanje velikosti podatkovnega skladišča tekom testiranja. 72
Slika 28. Gibanje velikosti večdimenzijske podatkovne baze SSAS tekom testiranja. 73
Slika 29. Gibanje časa procesiranja tekom testiranja. 73
Slika 30. Gibanje časa izvajanja poizvedb tekom testiranja. 75
Slika 31. Nastavitveno okno sledenja v SQL Server Profiler-ju. 79
Slika 32. Glavno okno SQL Server Profiler-ja po končanem sledenju. 80
Slika 33. Glavno okno Database Engine Tuning Advisor-ja. 81
Slika 34. Zavihek Tuning Options. 82
Slika 35. Zavihek Recommendations. 83
Slika 36. Zavihek Reports. 83
Slika 37. Korak, kjer določimo števila objektov. 85
Slika 38. Korak, kjer določimo nastavitve izdelave agregacij. 86
Slika 39. Nastavitve SSAS strežnika. 87
Slika 40. Izbira filtra poizvedb za analizo. 88
Slika 41. Izbira poizvedb za analizo. 89

Seznam tabel

Tabela 1. Priporočila za nastavitev InstanceSelection. 31
Tabela 2. Rezultat poizvedbe brez uporabe NON EMPTY. 39
Tabela 3. Rezultat poizvedbe z uporabo NON EMPTY. 39
Tabela 4. Seznam konstruktov v SSAS 2005, ki podpirajo blokovno usmerjeno vrednotenje. 43
Tabela 5. Rezultati testiranja začetne zasnove (velikost). 63
Tabela 6. Rezultati testiranja začetne zasnove (čas). 63
Tabela 7. Rezultati testiranja po določitvi podatkovnih tipov (velikost). 64
Tabela 8. Rezultati testiranja po določitvi podatkovnih tipov (čas). 64
Tabela 9. Rezultati testiranja po izbrisu odvečnih podatkov (velikost). 64
Tabela 10. Rezultati testiranja po izbrisu odvečnih podatkov (čas). 64
Tabela 11. Rezultati testiranja po kreiranju ključev (velikost). 64
Tabela 12. Rezultati testiranja po kreiranju ključev (čas). 64
Tabela 13. Rezultati testiranja po kreiranju indeksov (velikost). 64
Tabela 14. Rezultati testiranja po kreiranju indeksov (čas). 65
Tabela 15. Rezultati testiranja po uporabi nastavitve read-only (velikost). 65
Tabela 16. Rezultati testiranja po uporabi nastavitve read-only (čas). 65
Tabela 17. Rezultati testiranja po izbrisu odvečnih atributov (velikost). 65
Tabela 18. Rezultati testiranja po izbrisu odvečnih atributov (čas). 65
Tabela 19. Rezultati testiranja po določenih povezavah (velikost). 65
Tabela 20. Rezultati testiranja po določenih povezavah (čas). 65
Tabela 21. Rezultati testiranja po definiranih hierarhijah (velikost). 66
Tabela 22. Rezultati testiranja po definiranih hierarhijah (čas). 66
Tabela 23. Rezultati testiranja po spremenjenih nastavitvah atributov (velikost). 66
Tabela 24. Rezultati testiranja po spremenjenih nastavitvah atributov (čas). 66
Tabela 25. Rezultati testiranja po odstranjenih odvečnih merah (velikost). 66
Tabela 26. Rezultati testiranja po odstranjenih odvečnih merah (čas). 66
Tabela 27. Rezultati testiranja po izbrisu podvojenih dimenzij (velikost). 66
Tabela 28. Rezultati testiranja po izbrisu podvojenih dimenzij (čas). 67
Tabela 29. Rezultati testiranja različnih načinov shranjevanja mer in agregacij (velikost). 67
Tabela 30. Rezultati testiranja različnih načinov shranjevanja mer in agregacij (čas). 67
Tabela 31. Rezultati testiranja po particioniranju (velikost). 67
Tabela 32. Rezultati testiranja po particioniranju (čas). 67
Tabela 33. Rezultati testiranja po uporabi agregacij (velikost). 68
Tabela 34. Rezultati testiranja po uporabi agregacij (čas). 68
Tabela 35. Rezultati testiranja po preoblikovanju poizvedb MDX (čas). 68
Tabela 36. Rezultati testiranja po uporabi SQL Native Provider-ja (čas). 69
Tabela 37. Rezultati testiranja načinov procesiranja kocke (čas). 69
Tabela 38. Rezultati testiranja pri različnih številih vzporednih poslov (čas). 70
Tabela 39. Rezultati testiranja po optimizaciji DistinctCount mere (velikost). 70
Tabela 40. Rezultati testiranja po optimizaciji DistinctCount mere (čas). 70

Tabela 41. Rezultati testiranja po optimizaciji razmerja mnogo proti mnogo (velikost)..... 70
Tabela 42. Rezultati testiranja po optimizaciji razmerja mnogo proti mnogo (čas)..... 70
Tabela 43. Rezultati testiranja po spremembi ravnanja s pomnilnikom (velikost). 71
Tabela 44. Rezultati testiranja po spremembi ravnanja s pomnilnikom (čas)..... 71
Tabela 45. Rezultati testiranja po povečanju paralelnosti (velikost). 71
Tabela 46. Rezultati testiranja po povečanju paralelnosti (čas). 71
Tabela 47. Rezultati testiranja po spremembi BufferMemoryLimit (čas). 71
Tabela 48. Pomen barv na grafikonih. 72
Tabela 49. Opis delov poizvedbe, ki izračuna odstotek redukcije razmerij M2M. 91

Literatura

[1] M. Carroll. (2007). *OLAP Design Best Practices for Analysis Services 2005, SQL Server Best Practices Article.* Dostopno na: http://technet.microsoft.com/en-us/library/cc966399.aspx

[2] A. Chigrik. (2003). *Analysis Services optimization tips*. Dostopno na: http://www.databasejournal.com/features/mssql/article.php/1598311/Analysis-Services-optimization-tips.htm

[3] Gartner, Inc. (2010). *Gartner EXP Worldwide Survey of Nearly 1,600 CIOs Shows IT Budgets in 2010 to be at 2005 Levels.* Dostopno na: http://www.gartner.com/it/page.jsp?id=1283413

[4] Gartner, Inc. (2009). *Gartner EXP Worldwide Survey of More than 1,500 CIOs Shows IT Spending to Be Flat in 2009*. Dostopno na: http://www.gartner.com/it/page.jsp?id=855612

[5] Gartner, Inc. (2008). *Gartner EXP Worldwide Survey of 1,500 CIOs Shows 85 Percent of CIOs Expect "Significant Change" Over Next Three Years*. Dostopno na: http://www.gartner.com/it/page.jsp?id=587309

[6] D. Hardan, E. Veerman, C. Rabeler. (2007). *Analysis Services Many-to-Many Dimensions: Query Performance Optimization Techniques*. Dostopno na: http://download.microsoft.com/download/6/b/8/6b858e1e-311a-4fb9-9b15-116821000804/Many-to-ManyQryOptTech.docx

[7] S. Harinath, S. R. Quinn, *Professional SQL Server Analysis Services 2005 with MDX*, Indianapolis: Wiley Publishing, Inc., 2006, poglavje 1

[8] Inštitut za slovenski jezik Frana Ramovša ZRC SAZU. (2009). *Slovar slovenskega knjižnega jezika.* Dostopno na: http://bos.zrc-sazu.si/sskj.html

[9] R. Jacobson, S. Misner, H. Consulting, *Microsoft® SQL Server™ 2005 Analysis Services Step by Step*, Washington: Microsoft Press, 2006, poglavje 1

[10] B. Larson, *Delivering Business Intelligence with Microsoft SQL Server 2005*, Emeryville: McGraw-Hill/Osborne, 2006, poglavja 1, 2, 3, 4

[11] D. Lee, N. Dristas, C. Rabeler. (2007). *Analysis Services Processing Best Practices, SQL Server Best Practices Article*. Dostopno na: http://technet.microsoft.com/en-us/library/cc966525.aspx

[12] B. McGehee. (2006). *Tips for Optimizing SQL Server OLAP/Analysis Services: Data Warehouse & Datamart Performance*. Dostopno na: http://www.sql-server-performance.com/tips/olap_performance_p1.aspx

[13] B. McGehee. (2006). *Tips for Optimizing SQL Server OLAP/Analysis Services -- Cube Performance*. Dostopno na: http://www.sql-server-performance.com/tips/olap_performance_cubes_p1.aspx

[14] B. McGehee. (2006). *Tips for Optimizing SQL Server OLAP/Analysis Services Hardware Performance*. Dostopno na: http://www.sql-server-performance.com/tips/olap_performance_hardware_p1.aspx

[15] Microsoft Corporation. (2002). *Best Practices for Business Intelligence Using the Microsoft Data Warehousing Framework*. Dostopno na: http://msdn.microsoft.com/en-us/library/aa902663(SQL.80).aspx

[16] Microsoft Corporation. (2007). *Analysis Services Query Performance Top 10 Best Practices*. Dostopno na: http://technet.microsoft.com/en-us/library/cc966527.aspx

[17] Y. Mor. (2006). *Storage Modes in SSAS 2005*. Dostopno na: http://www.sqlservercentral.com/articles/Analysis+Services+(SSAS)/2658/

[18] J. Mundy, W. Thornthwaite, R. Kimball, *The Microsoft Data Warehouse Toolkit—With SQL Server 2005 and the Microsoft Business Intelligence Toolset*, Indianapolis: Wiley Publishing, Inc., 2006, poglavja 2, 4, 7

[19] N. Pendse. (2008). *What is OLAP?* Dostopno na: http://www.bi-verdict.com/fileadmin/FreeAnalyses/fasmi.htm

[20] C. Rabeler, E. Jacobsen. (2007). *Identifying and Resolving MDX Query Performance Bottlenecks in SQL Server 2005 Analysis Services*. Dostopno na: http://download.microsoft.com/download/4/4/8/4483e661-4218-4072-b79e-77f8db199df2/IdentifyingAndResolvingMDXBottlenecksSSAS.doc

[21] R. L. Sallam, B. Hostmann, J. Richardson, A. Bitterer. (2010). *Magic Quadrant for Business Intelligence Platforms*. Dostopno na: http://www.gartner.com/technology/media-products/reprints/oracle/article121/article121.html

[22] G. Spofford, S. Harinath, C. Webb, D. H. Huang, F. Civardi, *MDX Solutions, Second Edition: With Microsoft ® SQL Server™ Analysis Services 2005 and Hyperion® Essbase*, Indianapolis: Wiley Publishing, Inc., 2006, poglavje 1

[23] E. Vitt. (2007). *Microsoft SQL Server 2005 Analysis Services Performance Guide*. Dostopno na: http://download.microsoft.com/download/8/5/e/85eea4fa-b3bb-4426-97d0-7f7151b2011c/SSAS2005PerfGuide.doc