|  | **Univerzitet u Nišu, Elektronski fakultet**  **Katedra za računarstvo** |  |
| --- | --- | --- |

**Fizičko projektovanje baze podataka i optimizacija podataka (SQL Server)**

**Seminarski rad**

Profesor: Student:

Prof. Aleksandar Stanimirović Anđela Kričak, 1498/22

**Niš, 2024. godina**

**Sadržaj**

[Uvod 3](#_Toc178423894)

[Fizičko projektovanje baza podataka 4](#_Toc178423895)

[Particionisanje podataka 4](#_Toc178423896)

[Distribucija podataka 6](#_Toc178423897)

[Normalizacija i denormalizacija 7](#_Toc178423898)

[Fizičko mapiranje tabela i indeksa 8](#_Toc178423899)

[Klasterovanje 10](#_Toc178423900)

[Optimizacija podataka u *SQL Serveru* 11](#_Toc178423901)

[Komprimovanje podataka 11](#_Toc178423902)

[Keširanje podataka 12](#_Toc178423903)

[Indeksiranje 13](#_Toc178423904)

[Fragmentacija i defragmentacija indeksa u *SQL Serveru* 16](#_Toc178423905)

[Defragmentacija indeksa: 16](#_Toc178423906)

[Zaključak 18](#_Toc178423907)

[Literatura 19](#_Toc178423908)

# Uvod

Fizičko projektovanje i optimizacija podataka u *SQL Serveru* predstavljaju fundamentalne aspekte u upravljanja bazama podataka, igrajući ključnu ulogu u postizanju visoke efikasnosti i performansi sistema. Ovaj proces uključuje strateško planiranje i implementaciju metoda za fizičko organizovanje podataka na diskovima, čime se optimizuje pristup i obrada tih podataka.

U svetu sve većih količina podataka i složenih aplikacija, efektivno fizičko projektovanje postaje neophodno kako bi se osigurala brzina izvršavanja upita, minimalizovalo vreme čekanja i smanjili resursi potrebni za obradu podataka. Za razliku od logičkog projektovanja koje se bavi definisanjem strukture podataka, njihovih međusobnih odnosa i ograničenja, fizičko projektovanje se fokusira na konkretne tehnike skladištenja podataka na fizičkim uređajima, kao što su hard diskovi i *SSD*-ovi.

U *SQL Serveru*, fizičko projektovanje obuhvata različite aspekte, uključujući particionisanje tabela, izbor pristupnih metoda, particionisanje podataka, distribuciju podataka i upravljanje fizičkom strukturom tabela. Svaka od ovih odluka ima direktan uticaj na performanse sistema i vreme izvršavanja upita. Bez pravilno dizajniranih struktura i optimizovanih šema, baza podataka može postati preopterećena, uzrokujući usporavanje rada aplikacija i smanjenje efikasnosti celog sistema.

Fizičko projektovanje takođe uključuje optimizaciju *I/O* operacija kroz podešavanje parametara kao što su veličina stranice, vrsta *RAID* konfiguracije, i podešavanje bafera. Sve ove aktivnosti imaju za cilj smanjenje latencije i povećanje propusnosti sistema.

**Optimizacija podataka** u *SQL Serveru* podrazumeva implementaciju strategija koje minimiziraju potrošnju resursa, poput memorije i I/O operacija, i maksimiziraju brzinu izvršavanja upita. Ove tehnike uključuju kompresiju podataka, keširanje, pravilno indeksiranje, particionisanje, kao i redovno održavanje baza podataka kroz reorganizaciju i rekonstrukciju indeksa. Optimizacija je ključna kako bi se obezbedio balans između performansi i pouzdanosti baze, naročito u scenarijima sa visokim brojem simultanih zahteva i velikim količinama podataka.

SQL Server pruža napredne alate i funkcionalnosti koje omogućavaju administratorima i projektantima baza podataka da efikasno upravljaju podacima. Korišćenjem tehnika fizičkog projektovanja i optimizacije podataka, baza podataka može postati robusna, brza i skalabilna, spremna da odgovori na izazove današnjih aplikacija koje zahtevaju sve brži pristup informacijama i obradu podataka u realnom vremenu. Dobro optimizovana baza podataka smanjuje troškove održavanja, povećava pouzdanost sistema i obezbeđuje bolju korisničku efikasnost.

U ovom radu biće detaljno objašnjeno fizičko projektovanje baze podataka u *SQL Serveru*, kao i optimizacija podataka na praktičnim primerima.

# Fizičko projektovanje baza podataka

Fizičko projektovanje baze podataka je proces dizajniranja i implementacije baze podataka na fizičkom nivou. Pod ovim se podrazumeva razmatranje načina skladištenja podataka na disku, ali i kako će podaci biti raspoređeni na različitim fizički uređajima u cilju povećanja performansi.

Osnovni cilj fizičkog projektovanja baza podataka je da se zadovolje nefunkcionalne specifikacije prikupljene u fazi analize sistema. Nefunkcionalne specifikacije najčešće definišu performanse koje ceo sistem treba da zadovolji, kao i posebne zahteve za neke pojedinačne ili grupe aplikacija.

Fizičko projektovanje relacionih baza podataka veoma zavisi od karakteristika konkretnog relacionog SUBP. Nije uobičajeno da se rade neki detaljni proračuni da bi se optimizovala fizička struktura baze. Umesto toga, primenjuju se ekspertska znanja u fazi fizičkog projektovanja, a zatim se stalno prate performanse bitnih aplikacija i vrši se povremeno podešavanje (*tuning*) fizičke strukture baze.

Fizičko projektovanje baze podataka obično obuhvata sledeće korake:

## Particionisanje podataka

Particionisanje je tehnika koja omogućava podelu velikih tabela i indeksa na manje fizičke delove, čime se poboljšava performansa upita i olakšava upravljanje podacima. *SQL Server* podržava dva glavna oblika particionisanja: horizontalno i vertikalno.

#### Horizontalno particionisanje

Horizontalno particionisanje podrazumeva podelu tabele na osnovu redova, gde se redovi razvrstavaju u particije prema određenom kriterijumu (npr. datum, ID ili lokacija). Ovo omogućava *SQL Serveru* da brzo pristupi podacima u određenoj particiji, umesto da pretražuje celu tabelu.

Pretpostavimo da imamo tabelu *Orders* koja beleži sve porudžbine u sistemu. Tabela sadrži milione zapisa i svakodnevno se povećava. Kako bi se poboljšale performanse upita i pretraga, moguće je podeliti ovu tabelu po godini porudžbine:

-- Kreiranje tabele za particionisanje prema godini

CREATE PARTITION FUNCTION OrderYearPartition (INT)

AS RANGE LEFT FOR VALUES (2020, 2021, 2022);

CREATE PARTITION SCHEME OrderPartitionScheme

AS PARTITION OrderYearPartition ALL TO ([PRIMARY]);

-- Kreiranje tabele sa particijama

CREATE TABLE Orders

(

OrderID INT PRIMARY KEY,

OrderDate DATE,

CustomerID INT,

Amount DECIMAL(10, 2)

)

ON OrderPartitionScheme (YEAR(OrderDate));

U ovom primeru, tabela Orders je podeljena na particije prema koloni OrderDate. Upiti koji se odnose na određeni datum sada mogu brže pristupiti relevantnoj particiji, umesto da pretražuju celu tabelu.

#### Vertikalno particionisanje

Vertikalno particionisanje se koristi za podelu tabele po kolonama. Kolone koje se često koriste mogu se čuvati u jednoj tabeli, dok se ređe korišćene kolone mogu premestiti u drugu tabelu. Ova tehnika smanjuje broj kolona koje *SQL Server* mora da učita pri izvršavanju upita, čime se poboljšavaju performanse.

Tabela *Customers*, koja sadrži informacije o korisnicima, može biti podeljena tako da često korišćeni podaci kao što su CustomerID, Name, Phone budu u jednoj tabeli, dok podaci poput Address i e-mail budu u drugoj:

-- Originalna tabela sa mnogo kolona

CREATE TABLE Customer

(

CustomerID INT PRIMARY KEY,

Name NVARCHAR(100),

Address NVARCHAR(200),

Phone NVARCHAR(15),

Email NVARCHAR(50),

DateOfBirth DATE

);

-- Vertikalno particionisanje

-- Tabela sa često korišćenim kolonama

CREATE TABLE CustomerMain

(

CustomerID INT PRIMARY KEY,

Name NVARCHAR(100),

Phone NVARCHAR(15)

);

-- Tabela sa ređe korišćenim kolonama

CREATE TABLE CustomerDetails

(

CustomerID INT PRIMARY KEY,

Address NVARCHAR(200),

Email NVARCHAR(50),

DateOfBirth DATE

);

Upiti koji ciljaju osnovne podatke o korisnicima sada će raditi brže jer *SQL Server* mora da obradi manje kolona.

## Distribucija podataka

Distribucija podataka koristi se u distribuiranim bazama podataka kako bi se podaci efikasno podelili između više servera ili čvorova. Ova tehnika postaje ključna kako baze podataka rastu i kako je neophodno skalirati sisteme za brži pristup i povećanje dostupnosti. Distribucija omogućava paralelnu obradu zahteva, smanjujući latenciju i optimizujući upotrebu resursa.

**Tipovi distribucije:**

1. **Distribucija po opsegu (*Range Partitioning*):** Podaci se distribuiraju između različitih čvorova na osnovu vrednosti iz određenog opsega (npr. godine, geografske lokacije). Ovaj pristup je koristan kada podaci imaju prirodno uređenje po vrednostima.

Ukoliko postoji velika bazu podataka korisnika (Users), koju je potrebno podeliti prema regionima (Evropa i Amerika), *SQL Server* može upućivati zahteve ka odgovarajućoj tabeli na osnovu geografske lokacije korisnika.

-- Distribucija korisnika po regionima

CREATE TABLE Users\_Europe

(

UserID INT PRIMARY KEY,

Name NVARCHAR(100),

Country NVARCHAR(50)

);

CREATE TABLE Users\_America

(

UserID INT PRIMARY KEY,

Name NVARCHAR(100),

Country NVARCHAR(50)

);

Prednosti:

* Skalabilnost: Baza raste horizontalno, omogućavajući dodavanje novih regiona ili vremenskih opsega bez većih problema.
* Optimizacija upita: Upiti koji se odnose na određeni region postaju brži jer *SQL Server* može direktno pristupiti odgovarajućoj particiji ili čvoru.

1. **Distribucija po heširanju (*Hash Partitioning*):** Podaci se distribuiraju između čvorova na osnovu rezultata heš funkcije. Ovaj pristup koristi se kada se podaci ne mogu prirodno podeliti na opsege (npr. ID korisnika).

Korisnička baza može biti podeljena na osnovu heša *UserID*-ja, što omogućava ravnomernu distribuciju podataka između različitih čvorova.

-- Distribucija korisnika po heširanju

CREATE TABLE Users\_Hash\_A

(

UserID INT PRIMARY KEY,

Name NVARCHAR(100),

Country NVARCHAR(50)

);

CREATE TABLE Users\_Hash\_B

(

UserID INT PRIMARY KEY,

Name NVARCHAR(100),

Country NVARCHAR(50)

);

-- Logika za distribuciju prema heširanju

IF (ABS(CHECKSUM(@UserID)) % 2 = 0)

INSERT INTO Users\_Hash\_A VALUES (@UserID, @Name, @Country)

ELSE

INSERT INTO Users\_Hash\_B VALUES (@UserID, @Name, @Country);

**Prednosti:**

* Ravnomerna raspodela podataka: Heš funkcija osigurava da podaci budu ravnomerno raspoređeni između različitih čvorova.
* Eliminisanje "vrućih tačaka": Sprečava da jedan čvor postane preopterećen, jer se podaci distribuiraju ravnomerno.

1. **Distribucija po listi (*List Partitioning*):** Ova tehnika koristi specifične liste vrednosti, da bi se podaci dodelili određenim čvorovima. Na primer, određeni skup vrednosti (npr. zemlje: "Srbija", "Hrvatska", "Bosna") može biti distribuiran ka jednom čvoru.

-- Distribucija korisnika po specifičnim zemljama

CREATE PARTITION FUNCTION CountryPartition (NVARCHAR(50))

AS RANGE RIGHT FOR VALUES ('Bosnia', 'Croatia', 'Serbia');

CREATE PARTITION SCHEME CountryPartitionScheme

AS PARTITION CountryPartition ALL TO ([PRIMARY]);

-- Kreiranje tabele sa listom vrednosti za particionisanje

CREATE TABLE Users

(

UserID INT PRIMARY KEY,

Name NVARCHAR(100),

Country NVARCHAR(50)

)

ON CountryPartitionScheme (Country);

## Normalizacija i denormalizacija

Normalizacija je proces uklanjanja redundancije podataka i osiguravanja integriteta podataka kroz pravilno organizovanje tabela i odnosa među njima.

-- Normalizovan model

CREATE TABLE Customers

(

CustomerID INT PRIMARY KEY,

Name NVARCHAR(100),

Email NVARCHAR(100)

);

CREATE TABLE Orders

(

OrderID INT PRIMARY KEY,

CustomerID INT,

OrderDate DATE,

Amount DECIMAL(10, 2),

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

U većini slučajeva, zbog performansi, nije pogodno da fizička struktura baze uvek odražava striktno logički model relacija, tj. njihovu normalnu formu. Umesto toga, zbog performansi se često radi denormalizacija - ispuštanje atributa iz torki relacija (redundantni podaci), kako bi se smanjio broj relacija koje se često koriste zajedno u nekoj aplikaciji. Prilagođavanje logičke strukture podataka zahteva iskustvenu procenu potreba konkretne aplikacije i performansi sistema.

-- Denormalizovan model

CREATE TABLE Orders\_Denormalized

(

OrderID INT PRIMARY KEY,

CustomerID INT,

CustomerName NVARCHAR(100),

CustomerEmail NVARCHAR(100),

OrderDate DATE,

Amount DECIMAL(10, 2)

);

Ovaj pristup smanjuje broj *JOIN* operacija, što ubrzava pretragu, ali povećava mogućnost za redundanciju podataka.

## Fizičko mapiranje tabela i indeksa

Fizičko mapiranje u *SQL Serveru* podrazumeva postavljanje tabela, indeksa i drugih baza podataka na različite fizičke diskove ili skladišne uređaje kako bi se povećale performanse. Ova tehnika može pomoći da se balansira opterećenje diska, poveća brzina pristupa podacima i optimizuje iskorišćenje skladišta.

*SQL Server* koristi koncept file grupa (*filegroups*) za organizaciju podataka unutar baze. Korisnik može kreirati više file grupa, koje su fizički locirane na različitim diskovima ili uređajima, kao što su *SSD* i *HDD*. Ova strategija omogućava administratorima baza podataka da smeštaju najbitnije i najčešće korišćene tabele i indekse na brže diskove (npr. *SSD*), kako bi ubrzali čitanje i pisanje podataka, dok se manje korišćeni podaci mogu postaviti na sporije diskove (npr. *HDD*), radi uštede prostora i troškova.

Pretpostavimo da postoje dve grupe podataka: jedna sa visoko frekventnim operativnim podacima i drugu sa arhivskim podacima koji se retko koriste. Visoko frekventni podaci će biti smešteni na *SSD* diskovima radi bržeg pristupa, dok će arhivski podaci biti skladišteni na *HDD*-u.

1. **Kreiranje file grupa**  
   Prvo je potrebno kreirati dve file grupe - jednu za brzi *SSD* disk i drugu za sporiji *HDD* disk.

-- Kreiranje file grupa

ALTER DATABASE MyDatabase ADD FILEGROUP FastStorage;

ALTER DATABASE MyDatabase ADD FILEGROUP SlowStorage;

1. **Dodavanje fajlova u file grupe**  
   Zatim se dodaju fajlovi (datoteke) u odgovarajuće *file* grupe. Fajlovi su fizički locirani na različitim diskovima.

-- Dodavanje fajlova u file grupe

ALTER DATABASE MyDatabase ADD FILE

(NAME = 'SSDFile', FILENAME = 'D:\FastStorage\MyDatabase\_SSD.ndf')

TO FILEGROUP FastStorage;

ALTER DATABASE MyDatabase ADD FILE

(NAME = 'HDDFile', FILENAME = 'E:\SlowStorage\MyDatabase\_HDD.ndf')

TO FILEGROUP SlowStorage;

1. **Kreiranje tabela u odgovarajućim *file* grupama**

Sada je moguće kreirati tabele i odrediti gde će biti smeštene, koristeći *file* grupe. Na primer, tabela koja sadrži frekventne transakcione podatke biće smeštena na *SSD*-u, dok će arhivska tabela biti smeštena na *HDD*-u.

-- Kreiranje tabele na bržem SSD-u

CREATE TABLE Transactions

(

TransactionID INT PRIMARY KEY,

TransactionDate DATE,

Amount DECIMAL(10, 2)

)

ON FastStorage;

-- Kreiranje tabele u sporijoj file grupi

CREATE TABLE ArchiveData

(

ArchiveID INT PRIMARY KEY,

ArchiveDate DATE,

Data NVARCHAR(MAX)

)

ON SlowStorage;

1. **Kreiranje indeksa u različitim *file* grupama**

Takođe, moguće je kreirati indekse u odgovarajućim *file* grupama, kako bi se optimizovale performanse. Na primer, indeks na tabeli sa transakcijama može biti postavljen na brži *SSD*, dok će indeks za arhivske podatke biti smešten na *HDD*.

-- Kreiranje indeksa na SSD-u

CREATE NONCLUSTERED INDEX IDX\_TransactionDate

ON Transactions(TransactionDate)

ON FastStorage;

-- Kreiranje indeksa na HDD-u

CREATE NONCLUSTERED INDEX IDX\_ArchiveDate

ON ArchiveData(ArchiveDate)

ON SlowStorage;

#### ****Prednosti fizičkog mapiranja:****

1. **Povećanje performansi:** Postavljanje najčešće korišćenih podataka na brže *SSD* diskove može značajno ubrzati pristup podacima i smanjiti vreme izvršavanja upita.
2. **Bolja iskorišćenost resursa:** Kombinovanjem različitih vrsta skladišnih uređaja (npr. *SSD* i *HDD*), organizacije mogu smanjiti troškove skladištenja, dok i dalje omogućavaju optimalne performanse za kritične podatke.
3. **Balansiranje opterećenja:** Smanjenje opterećenja na jednom fizičkom disku može pomoći u izbegavanju zagušenja prilikom velikog broja simultanih zahteva, jer podaci mogu biti distribuirani na više diskova.
4. **Fleksibilnost:** *SQL Server* omogućava lako dodavanje novih *file* grupa i premeštanje podataka između njih bez ometanja rada aplikacija.

## Klasterovanje

Pod pojmom klasterovanja u fizičkom projektovanju baza podataka podrazumeva se postupak preko koga se projektuje takva fizička struktura baze podataka, u kojoj su “fizički bliski” podaci relacija koje se često koriste zajedno u nekoj aplikaciji. “Fizilki blsiki” znači, na primer, da se n-torke različitih relacija koje se spajaju učitavaju u jednom fizičkom bloku. Većina relacionih SUBP-a poseduje mehanizme za ovakvo klasterovanje tabela, a sam izbor relacija koje će biti u jednom klasteru obično je stvar ekspertske proecene.

U kontekstu *SQL Servera*, klasterovanje se najčešće odnosi na **klasterovane indekse** (*clustered* *indexes*), gde su podaci u tabeli fizički organizovani po redosledu vrednosti jedne ili više kolona. Za razliku od neklasterovanih indeksa, gde se indeks samo referencira na podatke, klasterovani indeksi organizuju podatke na disku u skladu sa redosledom ključeva indeksa.

Više o klasterovanom indeksu biće u sledećem poglavlju.

# Optimizacija podataka u *SQL Serveru*

Optimizacija podataka u *SQL Serveru* obuhvata skup tehnika i postupaka čiji je cilj poboljšanje performansi baze podataka. Optimizacija se odnosi na različite aspekte rada baze, kao što su brzina izvršenja upita, efikasnost korišćenja resursa (memorija, *CPU*, disk), i smanjenje latencije pri pristupu podacima. *SQL Server* nudi niz alata i funkcionalnosti koje omogućavaju bolje performanse, kako za male baze podataka, tako i za velike, distribuirane sisteme.

## Komprimovanje podataka

Komprimovanje podataka u *SQL Serveru* je ključna tehnika koja se koristi kako bi se smanjila količina prostora koju podaci zauzimaju na disku, kao i za ubrzavanje operacija čitanja podataka. Komprimovanje se može primeniti na dve granularnosti:

* **Komprimovanje po redu (*ROW-level compression*)**: Smanjuje veličinu svake pojedinačne redovne strukture, koristeći efikasnije skladištenje za podatke tipova *CHAR, VARCHAR, NCHAR, NVARCHAR* i celobrojnih podataka. *SQL Server* automatski skladišti podatke na optimalniji način, koristeći manje bitova za skladištenje vrednosti.
* **Komprimovanje po stranici (*PAGE-level compression*)**: Ovo je dublji nivo kompresije, koji se primenjuje na čitave stranice podataka. Uključuje i redovno komprimovanje, ali dodatno koristi algoritme kao što su *prefix* i *dictionary* kompresija unutar stranica podataka, čime se postiže veća ušteda u prostoru.

#### ****Prednosti komprimovanja podataka:****

1. **Manja upotreba prostora na disku**: Komprimovanje smanjuje veličinu tabela i indeksa, čime se štedi na prostoru za skladištenje podataka.
2. **Povećane performanse**: Zbog manje količine podataka za čitanje, *SQL Server* može brže izvršavati operacije, smanjujući opterećenje memorije i I/O podsistema.
3. **Smanjena upotreba memorije**: Komprimovani podaci zauzimaju manje prostora u memoriji, što omogućava da više podataka bude u *cache*-u istovremeno.

Recimo da postoji tabela sa transakcijama *Transactions*, i potrebno je primeniti komprimovanje po stranici jer je potreban veći nivo kompresije zbog velike količine podataka:

-- Komprimovanje tabele po stranici

ALTER TABLE Transactions REBUILD WITH (DATA\_COMPRESSION = PAGE);

U slučaju da je tabela *Orders* manja i poželjan je manje agresivan pristup kompresiji, moguće je koristiti komprimovanje po redu:

-- Komprimovanje tabele po redu

ALTER TABLE Orders REBUILD WITH (DATA\_COMPRESSION = ROW);

*SQL Server* omogućava i komprimovanje indeksa, što dodatno može optimizovati skladištenje podataka i ubrzati pretrage.

-- Komprimovanje klasterizovanog indeksa po stranici

CREATE CLUSTERED INDEX IX\_CompressedTransactions

ON Transactions(TransactionID)

WITH (DATA\_COMPRESSION = PAGE);

-- Komprimovanje nekontrolisanog indeksa po redu

CREATE NONCLUSTERED INDEX IX\_CompressedOrders

ON Orders(OrderID)

WITH (DATA\_COMPRESSION = ROW);

Komprimovanje je poželjno koristiti kada baze veoma brzo rastu i zauzimaju veliki prostor na disku i kada se podaci često čitaju, ali ne previše često ažuriraju, jer komprimovanje može usporiti operacije pisanja i ažuriranja zbog dodatnog procesiranja.

## Keširanje podataka

Keširanje u *SQL Serveru* je automatska funkcija koja omogućava da najčešće korišćeni podaci budu čuvani u memoriji (*RAM* ), smanjujući pristup fizičkim diskovima i ubrzavajući operacije pretrage i manipulacije podacima. *SQL Server* koristi *Buffer Pool* za keširanje stranica podataka i*r* automatski upravlja time koji će podaci biti keširani, na osnovu frekventnosti pristupa i internim algoritmima.

Osim automatskog keširanja, *SQL Server* podržava ***In-Memory OLTP*** tehnologiju, koja omogućava kreiranje tabela optimizovanih za keširanje u memoriji. Tabele u memoriji su značajno brže jer potpuno eliminišu potrebu za pristupom disku tokom čitanja i pisanja podataka.

#### ****Prednosti keširanja podataka:****

1. **Brži pristup podacima**: Podaci u kešu se mogu vrlo brzo “dohvatiti”, smanjujući potrebu za čitanjem sa diska.
2. **Manje opterećenje na I/O podsistem**: Kako *SQL Server* manje puta mora da pristupa disku, to smanjuje opterećenje na *I/O* podsistem.
3. **Povećana propusnost**: Upiti koji pristupaju podacima iz keša mogu se izvršavati mnogo brže, omogućavajući veću propusnost baze podataka.

Pretpostavimo da je potrebno kreirati tabelu optimizovanu za keširanje u memoriji, kako bi ubrzali pristup podacima o klijentima:

-- Kreiranje tabele optimizovane za keširanje u memoriji

CREATE TABLE Customers\_InMemory

(

CustomerID INT PRIMARY KEY NONCLUSTERED HASH WITH (BUCKET\_COUNT = 10000),

Name NVARCHAR(100),

DateOfBirth DATE

)

WITH (MEMORY\_OPTIMIZED = ON, DURABILITY = SCHEMA\_ONLY);

**Opcijom *MEMORY\_OPTIMIZED = ON*** obezbeđuje se da se tabela skladišti u memoriji, a ne na disku. Ova tabela će uvek biti keširana, a *SQL Server* neće morati da je čita sa diska tokom izvršavanja upita.

***DURABILITY = SCHEMA\_ONLY*** znači da će podaci biti skladišteni samo u memoriji i neće biti trajno zapisani na disk. Ovo se kotisti kada je potrebna ekstremno brza obrada podataka, ali trajnost podataka nije kritična (npr. privremene tabele ili sesijski podaci).

Osim podataka, *SQL Server* kešira i rezultate prethodnih upita kako bi ubrzao naredne upite koji koriste iste ili slične podatke. Ovo se zove plan keširanje. Moguće je koristiti hintove da *SQL Server* zadrži rezultate u kešu ili da izbriše keš, kako bi se dobili tačniji rezultati.

Na primer, korišćenje ***OPTION (RECOMPILE)*** u upitu znači da *SQL Server* neće keširati plan izvršenja tog upita, što može biti korisno kada se podaci stalno menjaju.

-- Primer keširanja upita

SELECT CustomerID, Name, DateOfBirth

FROM Customers

WHERE DateOfBirth > '1980-01-01'

OPTION (RECOMPILE); -- Ovo će onemogućiti keširanje ovog upita

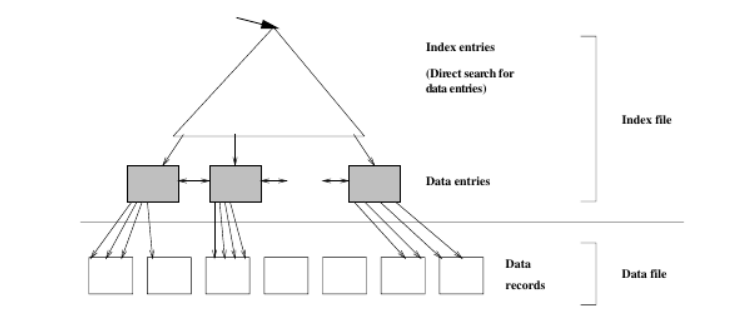
Keširanje treba koristiti prilikom rada sa podacima koji se često čitaju, sli se retko ažuriraju, u sistemima sa visokim brojem simultanih zahteva, gde je ključno da podaci budu brzo dostupni bez stalnog pristupa disku i kada su potrebni brzi odgovori na upite, a baze podataka imaju česte operacije čitanja (npr. u *OLAP* sistemima).

## Indeksiranje

Indeksiranje je kritičan deo optimizacije u *SQL Serveru*. Postoji nekoliko vrsta indeksa, uključujući klasterirane, neklasterirane, jedinstvene, kompozitne, prostorne, *full-text* i *columnstore* indekse. Svaka vrsta indeksa ima specifične prednosti i koristi se u zavisnosti od potreba aplikacije i tipa upita.

Različite vrste indeksa u *SQL Serveru* omogućavaju optimizaciju performansi upita i efikasno upravljanje podacima. Pravilno korišćenje indeksa može značajno poboljšati brzinu izvršavanja upita i smanjiti opterećenje sistema. Prilikom dizajniranja indeksa, važno je uzeti u obzir specifične potrebe aplikacije i obrasce pristupa podacima kako bi se postigli optimalni rezultati.

* **Klasterirani indeks**, kao što je rečeno u prethodnom poglavlju, određuje fizički redosled podataka u tabeli. Svaka tabela može imati samo jedan klasterirani indeks jer podaci mogu biti organizovani samo na jedan način. Na slici 1, ilustovana je upotreba klasterovanog indeksa.



Slika 1. Klasterovani indeks

Pretpostavimo da postoji tabela sa narudžbinama i da se izvršavaju upiti koji pretražuju podatke na osnovu datuma narudžbine. Da bi se optimizovale performanse tih upita, potrebno je kreirati klasterovani indeks na koloni *OrderDate.*

-- Kreiranje klasterovanog indeksa na koloni OrderDate

CREATE CLUSTERED INDEX IX\_OrderDate

ON Orders (OrderDate);

U ovom slučaju, *SQL Server* može brzo pronaći sve narudžbine između dva datuma, jer su redovi već fizički organizovani prema *OrderDate*.

Klasterovani indeks treba koristiti kada upiti često pretražuju podatke po kolonama koje su kontinuirane ili numeričke, kao što su datumi, brojevi faktura ili ID-jevi; kada tabela ima mnogo redova, a pretraga se često vrši prema određenom kriterijumu koji je lako sortiran (npr. datumi) i da bi se optimizovale performanse za opsežne upite, koji vraćaju mnogo podataka.

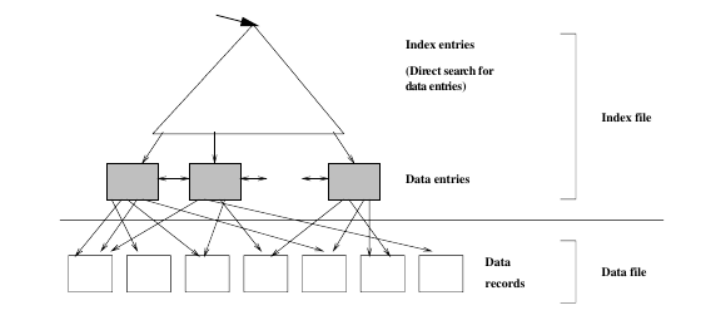
* **Neklasterirani indeks** pruža brži pristup podacima bez promene fizičkog redosleda. On sadrži ključne vrednosti iz tabele i pokazivače na fizičke lokacije podataka. Tabela može imati više neklasteriranih indeksa.

-- Kreiramo neklasterirani indeks na koloni CustomerID u tabeli Orders

CREATE NONCLUSTERED INDEX IX\_Orders\_CustomerID

ON Orders(CustomerID);

Na slici 2, prikazan je neklasterovani indeks.



Slika 2. Neklasterovani indeks

* **Jedinstveni indeks** osigurava da su vrednosti u koloni jedinstvene.

-- Kreiramo jedinstveni indeks na koloni Email u tabeli Customers

CREATE UNIQUE INDEX IX\_Customers\_Email

ON Customers(Email);

* + **Indeks kompozitnog ključa** koristi više kolona za kreiranje indeksa. Ovo može poboljšati performanse upita koji filtriraju ili sortiraju prema više kolona.

-- Kreiramo kompozitni neklasterirani indeks na kolonama CustomerID i OrderDate u tabeli Orders

CREATE NONCLUSTERED INDEX IX\_Orders\_CustomerID\_OrderDate

ON Orders(CustomerID, OrderDate);

* + **Prostorni indeksi** se koriste za optimizaciju upita koji uključuju prostorne podatke, kao što su geografske koordinate.

-- Kreiramo prostorni indeks na koloni Location u tabeli Locations

CREATE SPATIAL INDEX IX\_Locations\_Location

ON Locations(Location);

* + ***Full-text* indeks** omogućava pretraživanje teksta u tabeli. Koristi se za kompleksne pretrage teksta i ključnih reči.

-- Kreiramo full-text indeks na koloni Description u tabeli Products

CREATE FULLTEXT INDEX ON Products(Description)

KEY INDEX PK\_Products;

* + ***Columnstore* indeksi** su dizajnirani za analitičke upite i skladište podatke u kolonama umesto u redovima, što omogućava brže pretraživanje i agregaciju podataka.

-- Kreiramo clustered columnstore indeks na tabeli OrderDetails

CREATE CLUSTERED COLUMNSTORE INDEX CCI\_OrderDetails

ON OrderDetails;

## ****Fragmentacija i defragmentacija indeksa u SQL Serveru****

**Fragmentacija** u *SQL Serveru* nastaje kada redosled podataka u indeksima postane neorganizovan zbog umetanja, brisanja ili ažuriranja podataka. Kada se podaci ne dodaju u prirodnom redosledu indeksa (npr. umetanje podataka između već postojećih vrednosti u indeksu), *SQL Server* je primoran da podeli postojeće stranice indeksa kako bi smestio nove podatke. Ovaj proces se naziva **"*page split*"** ili podela stranica.

Posledica fragmentacije je da *SQL Server* mora da koristi više I/O operacija prilikom čitanja ili pisanja podataka, jer podaci koji bi inače bili kontinualno smešteni sada postaju podeljeni na različite stranice. Ova fragmentacija značajno utiče na performanse upita, posebno kod velikih tabela ili kada se učitava veliki broj redova.

**Vrste fragmentacije:**

1. **Interna fragmentacija:** Interna fragmentacija nastaje kada stranice indeksa nisu popunjene do kraja, odnosno sadrže previše praznog prostora. Ovo se dešava zbog umetanja podataka između postojećih redova, što zahteva deljenje stranica. Interna fragmentacija povećava broj stranica koje *SQL Server* mora da pročita, što može usporiti upite.
2. **Eksterna fragmentacija:** Eksterna fragmentacija se odnosi na slučajeve kada redovi podataka nisu fizički raspoređeni u kontinualnom redosledu na disku. Ovo znači da *SQL Server* mora da čita stranice koje nisu fizički blizu, što povećava broj I/O operacija.

### ****Defragmentacija indeksa:****

*SQL Server* pruža dva osnovna načina za rešavanje fragmentacije indeksa: **reorganizacija** i **rekonstrukcija** indeksa. Obe ove tehnike omogućavaju da se smanji fragmentacija i poboljšaju performanse upita.

**1. Reorganizacija indeksa (*Index Reorganize*):**

Reorganizacija indeksa je operacija sa niskim uticajem na performanse koja preuređuje podatke unutar indeksa, kako bi se smanjila interna fragmentacija bez premeštanja podataka između stranica. Ova tehnika ne pravi nove stranice, već samo preuređuje redosled podataka unutar postojećih stranica.

* Reorganizacija je **inkrementalni proces**, što znači da se stranice ne zaključavaju tokom reorganizacije, pa je manje disruptivna za aplikacije koje koriste bazu podataka.
* Ova tehnika je korisna kada je fragmentacija ispod 30%, jer efikasno smanjuje internu fragmentaciju bez velikih resursa.

-- Reorganizacija svih indeksa na tabeli Orders

ALTER INDEX ALL ON Orders REORGANIZE;

**2. Rekonstrukcija indeksa (*Index Rebuild*):**

Rekonstrukcija indeksa je potpun proces u kojem *SQL Server* kreira novi indeks od nule, reorganizuje sve stranice i uklanja svu fragmentaciju (i internu i eksternu). Rekonstrukcija indeksa zahteva više resursa i može privremeno zaključati tabelu dok traje proces.

* Rekonstrukcija indeksa je efikasnija od reorganizacije kada je fragmentacija visoka (preko 30%).
* U procesu rekonstrukcije, *SQL Server* kreira potpuno novi indeks, čime se vraća redosled stranica u optimalno stanje, eliminisući eksterne fragmentacije.
* **Online rekonstrukcija** je dostupna u naprednijim verzijama *SQL Servera*, što omogućava da proces rekonstrukcije ne ometa rad korisnika.

-- Rekonstrukcija svih indeksa na tabeli Orders

ALTER INDEX ALL ON Orders REBUILD;

Reorganizacija je preporučena kada je fragmentacija manja od 30% i kada administrator želiti operaciju koja ima minimalan uticaj na aplikaciju, a rekonstrukcija kada je fragmentacija veća od 30%, jer će potpuno eliminisati fragmentaciju, ali uz veću potrošnju resursa.

Nivo fragmentacije u tabeli proverava se pomoću *DMF* (*Dynamic Management Function*) *sys.dm\_db\_index\_physical\_stats*, koji prikazuje informacije o fragmentaciji indeksa.

-- Provera nivoa fragmentacije za sve indekse u tabeli Orders

SELECT

index\_id,

avg\_fragmentation\_in\_percent

FROM sys.dm\_db\_index\_physical\_stats(DB\_ID(N'MyDatabase'), OBJECT\_ID(N'Orders'), NULL, NULL, 'DETAILED');

# Zaključak

Fizičko projektovanje i optimizacija podataka predstavljaju ključne aspekte u izgradnji i održavanju baza podataka, naročito u *SQL Serveru*, gde performanse, pouzdanost i skalabilnost postaju ključni faktori u svakodnevnim operacijama. Kako organizacije sve više zavise od složenih sistema za upravljanje podacima, fizički dizajn baze postaje ključan za postizanje optimalne efikasnosti.

Jedan od glavnih izazova u upravljanju bazama podataka jeste balansiranje između brzine pristupa podacima i ekonomičnog korišćenja resursa, poput memorije, *CPU*-a i diskova. Fizičko projektovanje uključuje pažljivu analizu kako će se podaci skladištiti, pristupati i ažurirati, uzimajući u obzir specifične zahteve sistema i aplikacija. Tehnike kao što su particionisanje, distribucija podataka i kompresija omogućavaju *SQL Serveru* da efikasno upravlja velikim količinama podataka, istovremeno minimizirajući potrošnju resursa i maksimizirajući brzinu izvršavanja upita.

Particionisanje omogućava podelu velikih tabela na manje, logičke delove, čime se smanjuje vreme pretrage i olakšava održavanje baze podataka. Ove tehnike ne samo da poboljšavaju performanse, već i omogućavaju skalabilnost, što je od ključnog značaja u okruženjima gde baze podataka rastu eksponencijalno.

Još jedna važna tehnika je distribucija podataka, koja omogućava organizacijama da raspodele opterećenje između više servera ili diskova, čime se postiže bolja efikasnost u scenarijima sa velikim brojem istovremenih zahteva. Distribucija podataka može značajno poboljšati performanse u globalnim sistemima koji zahtevaju pristup podacima iz različitih geografskih lokacija.

Kompresija podataka igra ključnu ulogu u uštedi prostora na disku i ubrzavanju učitavanja podataka, dok keširanje omogućava *SQL Serveru* da najčešće korišćene podatke čuva u memoriji, smanjujući potrebu za pristupom disku i ubrzavajući upite. Kombinacijom ovih tehnika postiže se značajno smanjenje latencije i povećanje efikasnosti obrade podataka.

Optimizacija nije jednokratan proces, već kontinuirani napor. Redovno održavanje baza podataka, uključujući reorganizaciju i rekonstrukciju indeksa, ažuriranje statistika i analizu planova izvršenja upita, od suštinskog je značaja za održavanje performansi na visokom nivou. Kako se podaci razvijaju i aplikacije postaju složenije, neophodno je pratiti promene u performansama i prilagoditi fizičko projektovanje i optimizacione strategije.

Zaključno, fizičko projektovanje i optimizacija podataka u *SQL Serveru* su ključni za postizanje vrhunskih performansi i efikasnog upravljanja bazama podataka u modernim poslovnim okruženjima. Primena ovih tehnika ne samo da omogućava brži pristup podacima i smanjuje opterećenje sistema, već i obezbeđuje dugoročnu skalabilnost i održivost sistema. Pravilno implementirani principi fizičkog dizajna i optimizacije doprinose smanjenju troškova, poboljšanju korisničkog iskustva i povećanju ukupne produktivnosti organizacije.

# Literatura

1. „Baze podataka“ – Branislav Lazarević, Zoran Marjanović, Nenad Aničić, Slađan Babarogić
2. <https://people.vts.su.ac.rs/~peti/Baze%20podataka/Literatura/Skripta%20iz%20predmeta.%20Projektovanje%20baza%20podataka.pdf>
3. <https://repozitorij.etfos.hr/islandora/object/etfos%3A3453/datastream/PDF/view>
4. <https://books.google.rs/books?id=snh3_YCzAp8C&pg=PA53&hl=sr&source=gbs_toc_r&cad=2#v=onepage&q&f=false>
5. <https://en.wikipedia.org/wiki/Microsoft_SQL_Server>
6. <https://www.syncfusion.com/blogs/post/top-10-sql-query-optimization-techniques>
7. <https://medium.com/@guowili16/data-modeling-in-sql-584e56e3b6ff>