Univerzitet u Nišu, Elektronski fakultet
Katedra za računarstvo

Fizičko projektovanje baze podataka i optimizacija podataka (SQL Server)

Seminarski rad

Profesor:
Prof. Aleksandar Stanimirović

Student:
Anđela Kričak, 1498/22

Niš, 2024. godina

Sadržaj

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 dizajn indeksa, particionisanje tabela, raspodelu podataka na različite fizičke lokacije, optimizaciju *I/O* operacija i podešavanje parametara performansi. Indeksi, na primer, omogućavaju brži pristup podacima, ali zahtevaju dodatni prostor na disku i mogu usporiti operacije upisa. Stoga je neophodno pažljivo balansirati između potreba za brzim čitanjem i efikasnim pisanjem podataka.

Particionisanje tabela je još jedna ključna tehnika, gde se velike tabele dele na manje delove radi lakšeg upravljanja i bržeg pristupa. Ova tehnika ne samo da poboljšava performanse već i olakšava održavanje baze podataka. Pored toga, distribucija podataka na različite fizičke lokacije može pomoći u optimizaciji performansi tako što smanjuje konkurenciju za *I/O* resurse.

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* je kontinuiran proces koji zahteva stalno praćenje performansi, identifikaciju uskih grla i prilagođavanje konfiguracija kako bi se održala optimalna efikasnost.

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

SQL Server

SQL Server je sistem za upravljanje relacijskim bazama podataka razvijen od strane *Microsoft*-a. Namenjen je za skladištenje i upravljanje velikim količinama podataka, omogućavajući korisnicima da kreiraju, održavaju i manipulišu bazama podataka na jednostavan i efikasan način. Sa stalnim unapređenjima i podrškom za moderne tehnologije, *SQL Server* ostaje jedan od vodećih sistema za upravljanje bazama podataka na tržištu.

Ovaj sistem za upravljanje bazama podataka koristi se u poslovnoj analitici, e-trgovini, finansijama, zdravstvu i vladinim institucijama.

Ključne Karakteristike

1. Relacioni model:
    - SQL Server koristi relacioni model za organizaciju podataka u tabele koje su povezane relacijama. Ovo omogućava strukturalnu organizaciju podataka i lakoću pristupa.

2. *T-SQL*:
    - *Transact-SQL* (*T-SQL*) je proširenje *SQL* jezika, koje koristi *SQL Server* za upravljanje i manipulaciju podacima. *T-SQL* omogućava kompleksne upite, proceduralno programiranje i transakcije.

3. Integrisane usluge:
    - ***SQL Server Integration Services (SSIS)*:** Alat za ekstrakciju, transformaciju i učitavanje podataka (ETL proces).
    - ***SQL Server Reporting Services (SSRS)*:** Alat za kreiranje, izdavanje i upravljanje izveštajima.
    - ***SQL Server Analysis Services (SSAS)*:** Alat za analizu i izveštavanje podataka, uključujući *OLAP* i *data mining*.

4. Visoka Dostupnost i Oporavak:
    - *Always On Availability Groups*: Omogućava visoku dostupnost, oporavak od katastrofa i balansiranje opterećenja.
    - *Mirroring*, *Log Shipping* i *Clustering*: Tradicionalne metode za visoku dostupnost i oporavak.

5. Sigurnost:
    - *SQL Server* pruža robustne sigurnosne funkcije, uključujući autentifikaciju, autorizaciju, šifrovanje podataka, maskiranje podataka.

6. Skalabilnost i Performanse:

    - Podržava velike baze podataka, sa velikim brojem korisnika i visokom količinom transakcija.
    - *In-memory OLTP* (*Online Transaction Processing*) za ubrzanje transakcija.

7. Podrška za Različite Tipove Podataka:

    - Pored tradicionalnih numeričkih i tekstualnih podataka, *SQL Server* podržava rad sa *XML*, *JSON*, geolokacijskim podacima, grafovima i binarnim podacima.

8. *Cloud* Integracija:

    - SQL Server se može koristiti u oblaku putem *Azure SQL Database*, pružajući fleksibilnost i skalabilnost cloud okruženja.

Najčešće korišćeni alati za za upravljanje *SQL Server* istancama su *SQL Server Management Studio (SSMS)* - omogućava korisnicima da kreiraju, konfigurišu i održavaju baze podataka, *Azure Data Studio* - alat za razvoj i upravljanje bazama podataka u hibridnim i oblačnim okruženjima, *Visual Studio* - integracija sa *Visual Studi*o-om za razvoj aplikacija koje koriste *SQL Server* kao *backend*.

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, izbor vrste indeksa i ključeva koji će se koristiti za pretraživanje podataka, 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:

1. **Definisanje šeme baze podataka i identifikacija ključeva**

Proces fizičkog projektovanja baze podataka počinje definisanjem šeme baze podataka koristeći entitetsko-relacioni model (ER model). ER model se koristi za vizualizaciju strukture baze podataka kroz entitete (objekte) i njihove međusobne odnose. Ova faza obuhvata identifikaciju entiteta (npr. Klijenti, Naručbine) i njihovo međusobno povezivanje, što postavlja temelje za fizičku implementaciju baze u *SQL Serveru*.

Nakon identifikacije entiteta, potrebno je odrediti atribute koji su od značaja za isti. Na primer tabela Klijent će sadržati ID_Klijenta, Ime, Prezime... Neki od atributa će biti određeni kao ključevi. Ključ predstavlja jedan ili više atributa, čija vrednost jedinstveno identifikuje zapis i on može biti prost ili složen. Prost bi npr. bio samo ID_Klijenta, dok bi složeni ključ predstavljao kombinaciju dve ili više kolona koje zajedno čine jedinstveni ključ. Primer bi mogla biti tabela DetaljiPotudzbine, gde kombinacija *OrderID* i *ProductID* zajedno čini složeni primarni ključ za tabelu OrderDetails.:

```
CREATE TABLE OrderDetails
(
    OrderID int,
    ProductID int,
    Quantity int,
    PRIMARY KEY (OrderID, ProductID)
);
```

Vrste ključeva su: kandidat ključ, primarni ključ (jedan od kandidata ključeva), alternativni ključ i strani ključ (kolona u jednoj tabeli koja se odnosi na primarni ključ u drugoj tabeli, ova relacija omogućava uspostavljanje veze između podataka u različitim tabelama, što olakšava integritet podataka). Primarni ključ se još može podeliti na prirodni i veštački. Npr. ID_Klijenta bi bio veštački, dok bi JMBG_Klijenta bio prirodni.

Primer primarnog ključa:

```
CREATE TABLE Customers
(
    CustomerID int PRIMARY KEY,
    CustomerName varchar(255),
    ContactName varchar(255)
);
```

Primer stranog ključa:

```
CREATE TABLE Orders
(
    OrderID int PRIMARY KEY,
    OrderDate datetime,
    CustomerID int,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

2. **Izbor tipova podataka**

Pravilnim izborom tipova podataka u SQL Serveru omogućava se efikasnije korišćenje resursa i smanjenje zauzeća prostora. Na primer, korišćenje *VARCHAR* tipa umesto *CHAR* za promenljivu dužinu tekstualnih podataka smanjuje nepotrebno trošenje prostora. Osim toga, korišćenje odgovarajućih numeričkih tipova podataka, kao što su *INT* i *DECIMAL*, omogućava preciznost i efikasnost u skladištenju.

3. **Indeksiranje**

Indeksiranje je kritičan deo fizičkog projektovanja i 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** određuje fizički redosled podataka u tabeli. Svaka tabela može imati samo jedan klasterirani indeks jer podaci mogu biti organizovani samo na jedan način.

```
-- Kreiramo klasterirani indeks na koloni OrderID u tabeli Orders
CREATE CLUSTERED INDEX IX_Orders_OrderID
ON Orders(OrderID);
```

Ovaj indeks osigurava da su redovi u tabeli *Orders* fizički sortirani prema *OrderID*.

- **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);
```

- **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;
```

4. **Particionisanje**

Particionisanje omogućava bolje upravljanje velikim tabelama deljenjem na manje delove, što olakšava rad sa podacima i poboljšava performanse upita. U *SQL Serveru*, particionisanje može biti horizontalno (deljenje po redovima) ili vertikalno (deljenje po kolonama).

```
CREATE PARTITION FUNCTION myRangePF1 (int)
AS RANGE LEFT FOR VALUES (1, 100, 1000);

CREATE PARTITION SCHEME myRangePS1
AS PARTITION myRangePF1 TO ([PRIMARY], [FG1], [FG2], [FG3]);

CREATE TABLE Orders
(
    OrderID int,
    OrderDate datetime,
    CustomerID int
)
ON myRangePS1 (OrderID);
```

Ovaj primer prikazuje kako se tabela može particionisati na osnovu vrednosti u koloni *OrderID*.

5. ***Filegroup*-ovi i distribucija podataka**

SQL Server omogućava kreiranje *filegroup*-ova, što omogućava distribuciju podataka između različitih fizičkih diskova. Ovaj pristup omogućava bolju iskorišćenost resursa i ubrzava I/O operacije, posebno u slučajevima kada tabela ili indeksi zauzimaju značajan prostor.

Prvo, kreiraćemo dva *Filegroup*-a, jedan za tabelu *Customers*, a drugi za tabelu *Orders*.

```
ALTER DATABASE SalesDB
ADD FILEGROUP CustomersFG;

ALTER DATABASE SalesDB
ADD FILEGROUP OrdersFG;
```

Zatim, dodeljujemo fizičke fajlove svakom *Filegroup*-u, raspoređujući ih na različite diskove.

```
ALTER DATABASE SalesDB
ADD FILE
(
    NAME = CustomersFile,
    FILENAME = 'D:\SQLData\Customers.ndf',
    SIZE = 50MB,
    MAXSIZE = 500MB,
    FILEGROWTH = 50MB
) TO FILEGROUP CustomersFG;

ALTER DATABASE SalesDB
ADD FILE
(
    NAME = OrdersFile,
    FILENAME = 'E:\SQLData\Orders.ndf',
```

```
    SIZE = 100MB,
    MAXSIZE = 1GB,
    FILEGROWTH = 100MB
) TO FILEGROUP OrdersFG;
```

U ovom slučaju, podaci o kupcima biće smešteni na D:\ disku, dok će podaci o narudžbinama biti smešteni na E:\ disku.

Sada ćemo kreirati tabele Customers i Orders, pri čemu ćemo odrediti da se svaka tabela nalazi u odgovarajućem *Filegroup*-u.

```
CREATE TABLE Customers
(
    CustomerID int PRIMARY KEY,
    CustomerName varchar(100),
    ContactName varchar(100),
    Country varchar(50)
)
ON CustomersFG;

CREATE TABLE Orders
(
    OrderID int PRIMARY KEY,
    CustomerID int FOREIGN KEY REFERENCES Customers(CustomerID),
    OrderDate datetime,
    Amount decimal(18,2)
)
ON OrdersFG;
```

Ovim se postiže da podaci o kupcima budu fizički odvojeni od podataka o narudžbinama, što može pomoći u optimizaciji performansi, posebno kod sistema sa velikim brojem transakcija.

6. Normalizacija i denormalizacija

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

```
-- Normalizovana struktura
CREATE TABLE Customers
(
    CustomerID int PRIMARY KEY,
    CustomerName varchar(255)
);

CREATE TABLE Orders
(
    OrderID int PRIMARY KEY,
    OrderDate datetime,
    CustomerID int FOREIGN KEY 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.

```
-- Denormalizovana struktura
CREATE TABLE Orders
(
    OrderID int PRIMARY KEY,
    OrderDate datetime,
    CustomerName varchar(255)
);
```

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

Optimizacija podataka u *SQL Serveru*

Optimizacija *SQL* upita

Optimizacija *SQL* upita uključuje pravilno pisanje upita koji maksimalno koriste prednosti indeksa i minimiziraju nepotrebne operacije. Na primer, izbegavanje upotrebe *SELECT* * kada nije neophodno može značajno smanjiti količinu prenetih podataka i poboljšati brzinu izvršavanja upita.

Umesto:

```
SELECT * FROM Orders WHERE OrderDate = '2024-01-01';
```

Upotrebiti:

```
SELECT OrderID, CustomerID, Amount FROM Orders WHERE OrderDate = '2024-01-01';
```

Takođe, ako imate indeks na koloni koju koristite u WHERE klauzuli, SQL Server može mnogo brže pronaći odgovarajuće redove.

Umesto:

```
SELECT OrderID, CustomerID, Amount FROM Orders WHERE YEAR(OrderDate) = 2024;
```

Upotrebiti:

```
SELECT OrderID, CustomerID, Amount FROM Orders WHERE OrderDate BETWEEN '2024-01-01'
AND '2024-12-31';
```

Keširanje i memorijska optimizacija

SQL Server koristi keširanje upita i podatke iz *tempdb* baze za optimizaciju izvršavanja složenih upita.

```
EXEC sp_executesql N'SELECT OrderID, Amount FROM Orders WHERE CustomerID =
@CustomerID',
                   N'@CustomerID int', @CustomerID = 123;
```

Ovaj upit koristi parametre koji omogućavaju SQL Serveru da koristi keširane planove upita.

Memorijski optimizovane tabele koriste se za tabele kojima je potreban izuzetno brz pristup podacima, što omogućava poboljšanje performansi aplikacija.

```
CREATE TABLE dbo.OrdersMemoryOptimized
(
    OrderID int NOT NULL PRIMARY KEY NONCLUSTERED,
    CustomerID int NOT NULL,
```

```
    OrderDate datetime NOT NULL,
    Amount decimal(18, 2) NOT NULL
) WITH (MEMORY_OPTIMIZED = ON, DURABILITY = SCHEMA_ONLY);
```

Zadaci održavanja

Redovno održavanje baza podataka, uključujući reorganizaciju i rekonstrukciju indeksa, kao i ažuriranje statistika, ključno je za osiguranje optimalnih performansi u SQL Serveru. Ove aktivnosti omogućavaju optimizatoru upita da donosi bolje odluke i smanjuje vreme izvršavanja upita. SQL Server pruža alate kao što su Maintenance Plans i SQL Server Agent za automatizaciju ovih zadataka.

Zaključak

Fizičko projektovanje baze podataka i optimizacija podataka u *SQL Serveru* predstavlja složen i višeslojan proces koji direktno utiče na performanse i efikasnost poslovnih aplikacija. Pravilno projektovanje baze podataka nije samo stvar tehničke veštine već zahteva duboko razumevanje poslovnih potreba, očekivanih opterećenja i budućeg rasta sistema. Kroz pažljivo kreiranje i optimizaciju šema baze podataka, kao i preciznu definiciju ključnih entiteta i njihovih međusobnih odnosa, može se postići struktura koja ne samo da zadovoljava trenutne zahteve, već je i dovoljno fleksibilna da podrži buduće promene i proširenja.

Uloga indeksa u *SQL Serveru* je neprocenjiva u kontekstu performansi. Različite vrste indeksa omogućavaju različite vrste optimizacija, a pravilna primena indeksa može drastično smanjiti vreme izvršavanja upita, povećati efikasnost pretraga i poboljšati korisničko iskustvo. Klasterovani indeksi omogućavaju brze pretrage po primarnim ključevima, dok neklasterovani indeksi olakšavaju pretrage po sekundarnim kolonama, a specijalizovani indeksi kao što su *XML* i *Full-text* indeksi omogućavaju rad sa specifičnim tipovima podataka.

Optimizacija SQL upita i korišćenje tehnika kao što su keširanje i memorijska optimizacija pružaju dodatni sloj performansi koji je od ključnog značaja za visoko opterećene sisteme. Redovno održavanje baze podataka, kroz zadatke kao što su reorganizacija i rekonstrukcija indeksa, ažuriranje statistika i verifikacija integriteta, osigurava da sistem nastavi da radi optimalno čak i kako podaci rastu i sistem stari.

Na osnovu ovih razmatranja, možemo zaključiti da fizičko projektovanje baze podataka i optimizacija podataka u *SQL Server*u nisu samo tehničke discipline, već su i strateški procesi koji mogu značajno doprineti uspehu celokupnog IT sistema. Investiranje vremena i resursa u pravilno projektovanje i održavanje baze podataka može doneti dugoročne koristi u smislu povećanja performansi, smanjenja troškova održavanja i pružanja skalabilnog rešenja koje može da odgovori na sve izazove modernog poslovanja. Primenom najboljih praksi u svim fazama, od definisanja šeme baze i identifikacije ključeva, preko implementacije indeksa i optimizacije upita, do redovnog održavanja, moguće je postići visoko efikasan i stabilan sistem koji može zadovoljiti i najzahtevnije korisnike.

Literatura

[1] https://people.vts.su.ac.rs/~peti/Baze%20podataka/Literatura/Skripta%20iz%20predmeta.%20Projektovanje%20baza%20podataka.pdf

[2] https://repozitorij.etfos.hr/islandora/object/etfos%3A3453/datastream/PDF/view

[3] https://books.google.rs/books?id=snh3_YCzAp8C&pg=PA53&hl=sr&source=gbs_toc_r&cad=2#v=onepage&q&f=false

[4] https://en.wikipedia.org/wiki/Microsoft_SQL_Server

[5] https://www.syncfusion.com/blogs/post/top-10-sql-query-optimization-techniques

[6] https://medium.com/@guowili16/data-modeling-in-sql-584e56e3b6ff