УНИВЕРЗИТЕТ У НИШУ  
Електронски факултет

Семинарски рад  
Обрада упита код MS SQL базе података

Студенти: Ментор:  
Лазар Најдановић 1763 Александар Станимировић

Ниш, 2024.година

**САДРЖАЈ**

[**1.** **Увод** 3](#_Toc177754020)

[**2.** **Модели база података** 4](#_Toc177754021)

[**3.** **Microsoft SQL Server** 5](#_Toc177754022)

[3.1. Архитектура Microsoft SQL Server-a 6](#_Toc177754023)

[**4.** **Обрада упита** 7](#_Toc177754024)

[**4.1.** **Правила SQL-a** 7](#_Toc177754025)

[4.1.1. Правила за писање имена 8](#_Toc177754026)

[4.1.2. Правила о наредбама и изразима 8](#_Toc177754027)

[4.1.3. Типови података 8](#_Toc177754028)

[4.1.4. Дефиниција атрибута 9](#_Toc177754029)

[**4.2.** **Наредбе SQL-a** 9](#_Toc177754030)

[**4.3.** **SQL упити** 9](#_Toc177754031)

[4.3.1. Прост упит 10](#_Toc177754032)

[4.3.2. WHERE клаузула 10](#_Toc177754033)

[4.3.3. ORDER BY клаузула 10](#_Toc177754034)

[4.3.4. GROUP BY клаузула 11](#_Toc177754035)

[4.3.5. LIKE клаузула 11](#_Toc177754036)

[**4.4.** **Наредбе ажурирања** 11](#_Toc177754037)

[4.4.1. INSERT наредба 12](#_Toc177754038)

[4.4.2. Update наредба 13](#_Toc177754039)

[4.4.3. DELETE наредба 14](#_Toc177754040)

[**4.5.** **Оптимизација упита** 14](#_Toc177754041)

[4.5.1. Shared Pool check 15](#_Toc177754042)

[4.5.1.1. Катерогије парсирања 15](#_Toc177754043)

[4.5.1.2. Параметризовани упити 16](#_Toc177754044)

[4.5.2. Семантичка провера 17](#_Toc177754045)

[4.5.3. Синтаксна провера 18](#_Toc177754046)

[4.5.4. Индекси 18](#_Toc177754047)

[4.5.5. Execution plan 20](#_Toc177754048)

# 

# **Увод**

**База података** је интегрисани скуп података о неком систему и скуп поступака за њихово одржавање и коришћење, организован према потребама корисника. Представљају структуриране колекције података које омогућавају организацијама да складиште, управљају и приступају информацијама на ефикасан и поуздан начин. У суштини, базе података чине темељ на којем почива готово свака апликација, од најједноставнијих веб сајтова до сложених корпоративних система.

На њихов развој пресудно је утицао развој рачунара, рачунарских мрежа, као и клијент/сервер обраде. Истраживање, пројектовање и употреба база података су врло брзо показали низ својих добрих страна као што су: смањени трошкови одржавања, смањена потреба за мрежним ресурсима, побољшан интегритет података итд.

База података смештена у рачунару, представљена је низом битова, организованих у бајтове, док са више бајтова у одговарајућем формату записују се вредности појединих података и представљају једно поље базе података. Низ поља се организује у записе (рекорде) који представљају опис неког објекта из реалног света или неке везе између објеката реалног света. Записи истог формата се слажу и чине датотеке које су физички записане на диску.

База података обухвата више повезаних датотека, и може бити централизована (на једном рачунару) или дистрибуирана (на више међусобно удаљених рачунара).

Поред података који су записани у бази података, постоје и посебни подаци којима се описују појединачне датотеке, њене особине, карактеристични параметри из датотека, успостављене међусобне везе итд. Такви подаци се зову мета подаци тј. подаци о подацима. Користе се при покретању рада са базом података, како би се могли учитати сви конфигурациони подаци одговарајуће базе.

# **Модели база података**

За моделовање структуре података користе се различите технике. Одређени модели се лакше користе за неке типове система управљања базама података него други модели. Главни циљ је осмислити, дефинисати и имплементирати базе података.

Системи за управљање базама података могу се поделити у следеће основне моделе:

* Хијерархијски модел
* Мрежни модел
* Релациони модел
* Објектни модел

Хијерархијски модел базе података ограничен је структуром стабла где сваки запис има један надређени запис и више подзаписа. Овај модел је ефикасан за организовање података који имају јасну хијерархијску структуру, али може бити непрактичан за сложеније односе међу подацима.

Мрежни модел базе података проширује хијерархијски модел тако што омогућава сваком запису да има више надређених записа. Овај модел је флексибилнији од хијерархијског модела, али је сложен за имплементацију и управљање.

Објектни модел базе података интегрише карактеристике објектно-оријентисаног програмирања са базама података. Подаци су представљени као објекти који могу садржавати методе и особине, што олакшава рад са комплексним подацима.

Релациони модел базе података је најчешће коришћени модел данас.

Након што је дефинисан релациони модел, направљени су неформални модели да би се описали хијерархијски и мрежни модел. Хијерархијске и мрежне базе података су постојале пре релационих, али су као модели описани тек након што је релациони модел дефинисан, да би се направила основа за поређење.

У срцу релационог модела налази се концепт табеле (која се назива и релација) у којој су смештени сви подаци. Свака табела се састоји од редова који имају своја поља (атрибуте).

Суштина релационог модела је да се и класе објеката и класе веза између објеката представљају на јединствен начин то јест преко табела. Свака табела мора да има примарни кључ – један или више атрибута који на јединствен начин описују сваки рекорд у табели.

Овај семинарски рад је базиран на релационом моделу база података у Microsoft SQL Server-у.

# **Microsoft SQL Server**

SQL Server представља релациони систем за управљање базама података. Развијен је од стране Microsoft-а 1989. године као SQL Server 1.0 писан у C/C++. Овај систем за управљање базама података омогућава организацију, складиштење и манипулацију подацима на ефикасан и сигуран начин.

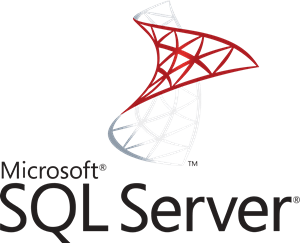
SQL Server нуди различите верзије, од бесплатних верзија попут SQL Server Express Edition-а до напредних верзија као што су SQL Server Standard и SQL Server Enterprise Edition. Свака верзија има своје карактеристике и ограничења, што омогућава корисницима да одаберу одговарајућу верзију у складу са потребама и захтевима њихових пројеката.

SQL Server подржава рад на више платформи, укључујући Windows и Linux. То омогућава већу флексибилност у избору оперативног система у зависности од специфичних потреба корисника и инфраструктуре.

Као и други софтвери за управљање релацијским базама података (RDBMS), Microsoft SQL Server је изграђен на основу језика за програмирање (SQL). То је стандардизовани језик који користе администратори база података и други IT стручњаци за управљање базама података и извршавање упита над подацима које оне садрже.

Неке од кључних карактеристика SQL Server су:

* Релациона база података: Организује податке у повезане табеле за ефикасно управљање.
* Скалабилност: Може руковати великим количинама података и корисника.
* Висока поузданост



Слика 1. - Microsoft SQL Server

## Архитектура Microsoft SQL Server-a

Са својом клијент-сервер архитектуром, SQL Server пружа могућност централизованог чувања података на серверу. Клијентске апликације могу приступити, управљати и ажурирати ове податке.

Microsoft SQL Server се састоји од два кључна дела:

* Database Engine
* SQLOS (SQL Server оперативни систем)

Database Engine служи за чување, обраду и обезбеђивање података, док SQLOS пружа оперативне услуге као што су управљање меморијом и I/O операцијама.

Database Engine се састоји од:

* Relational Engine-a (који обрађује упите)
* Storage Engine-a (који управља складиштењем података)

Relational Engine анализира, оптимизује и извршава упите, док Storage Engine управља приступом подацима на диску.

Relational Engine се састоји од:

* парсера команди (проверава синтаксне грешке и генерише стабло упита)
* оптимизатора (проналази оптимални план извршења упита)
* извршиоца упита (имплементира план и извршава упит)

Storage Engine се састоји од:

* Access Method (одређује како приступити подацима)
* Buffer Manager (управља кеширањем и управљањем меморијом)
* Transaction Manager (управља трансакцијама и записима у дневнику)

SQLOS пружа оперативне услуге као што су управљање меморијом, синхронизација нити и управљање изузецима. Такође контролише ресурсе које користи SQL Server.

# **Обрада упита**

Упити у контексту база података представљају инструкције или захтеве које корисници или апликације шаљу бази података како би добили одговарајуће информације или извршили одређене акције над подацима. Ови упити могу обухватати различите врсте операција, укључујући читање, писање, ажурирање или брисање података.

Главни циљ упита је да омогуће корисницима да ефикасно приступе подацима који су им потребни или да изврше одређене операције над тим подацима.

Обрада упита односи се на процес извршавања упита који су послати бази података ради добијања одговарајућих резултата. Кључни кораци обраде упита су:

* Анализа упита
* Оптимизација упита
* Извршавање упита
* Одговор на упит
* Обрада резултата

Анализа упита представља први корак у обради тог упита, тј. анализу самог упита како би разумели шта корисник жели да постигне. Овај корак укључује проверу синтаксе упита као и идентификацију табела и колона на које се тај упит односи.

## **Правила SQL-a**

Што се тиче синтаксе упита, важно је нагласити правила самог SQL-а. Она укључују:

* Правила за писање имена
* Правила о наредбама и изразима
* Типови података
* Дефиниција атрибута

## Правила за писање имена

Имена табела, погледа, атрибута, индекса и других објеката базе података морају да поштују следећа правила:

* Име не сме да садржи знак питања (?)
* Нема разлике између малих и великих слова
* Први знак мора бити слово
* Као имена не смеју се користити резервисане речи
* Дозвољени знаци су A-Z, 0-9, \_, $ и #
* Имена се не смеју понављати

## Правила о наредбама и изразима

Наредбе могу садржати следеће изразе у којима се појављују:

* Логичке операције: (AND, OR, NOT)
* Операције упоређивања: (=, <, >, IN, ANY, ALL, BETWEEN, IS NULL, LIKE)
* Скуповне операције: (UNION, INTERSECT, EXPECT)
* Агрегатне операције: (COUNT, SUM, AVG, MIN, MAX)

Изрази се могу груписати помоћу заграда, али исто тако могу садржати бројеве, текстуалне податке и остале врсте података.

## Типови података

SQL подржава следеће типове података:

* CHAR(n)
* VARCHAR(n)
* NUMBER
* DATE
* BOOLEAN

## Дефиниција атрибута

Атрибуте можемо да дефинишемо на следећи начин:

<**име\_атрибута**> <**тип\_атрибута**> <**додатна\_својства\_атрибута**>

Важно је напоменути да су додатна својства атрибута опциона, и нека од њих су:

* **DEFAULT** – задавање предефинисане вредности
* **NOT NULL** – вредност која не сме бити не задата
* **CHECK** – провера да је вредност атрибута у задатим границама
* **UNIQUE** – јединственост унутар релације
* **PRIMARY KEY** – примарни кључ
* **REFERENCES** – вредност мора бити међу вредностима из друге релације, обично је то кључ из друге релације.

## **Наредбе SQL-a**

Дефиниција базе података подразумева и могућност накнадне измене или уклањања те дефиниције. То се у SQL језику постиже са свега три наредбе:

* **CREATE**: Служи за креирање новог објекта
* **DROP**: Служи за уклањање дефиниције неког објекта из базе података
* **ALTER**: Служи за измену дефиниције неког објекта у бази података

## **SQL упити**

У овом поглављу, погледаћемо врсте SQL упита и примере кроз Microsoft SQL Server.

Код сваког упита се задаје:

* Које податке тражимо као резултат
* Из којих табела тражимо
* Који услов треба да задовоље подаци да би били укључени у резултат
* По ком редослед желимо приказ резултата

### Прост упит

Прост упит над једном табелом подразумева SELECT наредбу која као резултат даје ни један ред, један ред или низ редова података.

Примери:

* SELECT FirstName, LastName FROM Students WHERE Age > 20;
* SELECT COUNT(\*) FROM Students WHERE Age > 20;

### WHERE клаузула

WHERE клаузула служи за филтрирање упита. Приликом дефинисања критеријума можемо користити логичке и компаративне операторе. Важно је нагласити да WHERE клаузула није обавезна, а може се користити са SELECT, UPDATE и DELETE командама.

Примери:

* SELECT \* FROM Students WHERE Ocena > 8 AND (Ispit = “Sistemi za upravljane bazama podataka” OR Ispit = “Inteligentni sistemi”);
* UPDATE Students SET Ocena = 10 WHERE Indeks = 1763;
* DELETE FROM Students WHERE Indeks = 1763;

### ORDER BY клаузула

Помоћу ORDER BY клаузуле могуће је сортирати резултујућу табелу по једном или више атрибуту у растућем или опадајућем редоследу.

За спецификацију растућег редоследа користи се ASC клаузула, или DESC клаузула за спецификацију опадајућег редоследа.

Подразумевана вредност за ORDER BY клаузулу је ASC, тако да не морамо експлицитно да наводимо, док за опадајући редослед је то неопходно.

Још једна ствар коју је битно нагласити је то да је ORDER BY увек последња у SELECT блоку.

Примери:

* SELECT \* FROM Students ORDER BY Indeks;
* SELECT \* FROM Students ORDER BY Indeks ASC, Ocena DESC;

### GROUP BY клаузула

Клаузула GROUP BY користи се за груписање редова који имају исту вредност у одређеној колони и примењује агрегатне функције над тим групама.

GROUP BY се користи заједно са клаузулом WHERE, при чему WHERE увек иде пре GROUP BY. Помоћу WHERE се врши селекција н-торки, затим се те н-торке групишу помоћу GROUP BY.

Примери:

* SELECT AVG(Ocena) AS ProsecnaOcena FROM Students GROUP BY Ispit;
* SELECT Ispit, COUNT(\*) AS BrojPolozilih FROM Students WHERE Ispit = “Sistemi za upravljane bazama podataka” AND Ocena >= 6 GROUP BY Ispit;

### LIKE клаузула

Клаузула LIKE омогућава претраживањер на основу “узорка“, односно добијање информација и када не знамо потпун назив одређеног атрибута. Она користи два специјална карактера (“%”,”\_”) са следећим значењем:

• “%” представља стринг од 0 или више карактера, а

• “\_” представља позицију једног карактера. Остали карактери имају уобичајено значење.

## **Наредбе ажурирања**

Ажурирање у ширем смислу значења те речи обухвата додавање, измену садржаја и брисање реда или редова табеле. Те основне операције реализују се SQL наредбама: INSERT, UPDATE и DELETE са следећим значењем:

* **INSERT:** Додавање реда или редова у табелу
* **UPDATE:** Измена садржаја постојећег реда или редова табеле
* **DELETE:** Брисање постојећег реда или редова табеле

Tреба водити рачуна да су све три наведене наредбе - наредбе над једном табелом, тако да се њиховом применом не гарантује очување интегритета базе података.

### INSERT наредба

Постоје 3 случаја коришћења наредбе INSERT, и то када се врши:

* Унос вредности СВИХ атирибута н-торке
* Унос вредности НЕКИХ атрибута н-торке
* Унос података из једне табеле у другу

Унос вредности СВИХ атирибута н-торке:

Није потребно специﬁцирати називе атрибута, па INSERT наредба има облик:

INSERT INTO NazivTabele VALUES (value1, value2…);

За сваки атрибут МОРА постојати вредност, при чему је NULL дозвољена опција за сваки атрибут који није NOT NULL.

Пример:

INSERT INTO Students VALUES (1763, “Sistemi za upravljane bazama podataka”, 10);

Унос вредности НЕКИХ атрибута н-торке:

Ако желимо да унесемо вредност за само неке атрибуте, називи тих атрибута морају се експлицитно навести. У том случају наредба INSERT има облик:

INSERT INTO NazivTabele (atribut1, atribut2, …) VALUES (value1, value2,…);

Пример:

INSERT INTO Students (Indeks, Ocena) VALUES (1763, 10);

Унос података из једне табеле у другу:

Уколико обе табеле имају исти број атрибута и уколико су атрибути идентично деﬁнисани, наредба INSERT је облика:

INSERT INTO Tabela1 SELECT \* FROM Tabela2;

Пример:

INSERT INTO Students SELECT \* FROM Students2;

Уколико немају исти број атрибута, наредба је следећег облика:

INSERT INTO Tabela1 (atribut1, atribut2, ...) SELECT atribut1, atribut2

FROM Tabela2

WHERE kriterijum\_selekcije ;

Пример:

INSERT INTO Students (Indeks, Ocena) SELECT Indeks, Ocena FROM Students2;

### Update наредба

Уз ову наредбу мора се навести:

* У којој табели се врше измене
* За које колоне у реду мeњамо вредности
* Под којим условима мењамо вредности

Општи облик наредбе је:

UPDATE NazivTabele

SET atribut1 = izraz1 [,atribut2=izraz2] [WHERE kriterijum selekcije n-torki],

Пример:

UPDATE Students SET Ocena = 9 WHERE Ispit=“Sistemi za upravljane bazama podataka”;

односно,

UPDATE NazivTabele

SET(alribut1, atribut2,..)=(podupit) [WHERE kriterijum selekcije n-torki]

### DELETE наредба

Код ове наредбе наводимо:

* Из које табеле се врши уклањање
* Под којим условима се уклања неки ред

Синтакса наредбе:

DELETE FROM NazivTabelе WHERE R-Predikat

Пример:

DELETE FROM Students WHERE Indeks = 1763;

Претходне делове које смо обрадили (4.1. до 4.4.) представљају делове за анализу упита, како би разумели шта корисник жели да постигне тим упитом. Том приликом морамо да проверимо ситанксу упита, идентификујемо табеле и колоне над којима се он извршава итд.

Наредни корак у обради упита била би оптимизација упита.

## **Оптимизација упита**

Оптимизација упита представља кључни корак у развоју и одржавању перформантних база података. Њена сврха је побољшање брзине извршења упита, смањење оптерећења базе података и побољшање перформанси целокупне апликације.

Неколико разлога зашто је оптимизација упита важна су:

* Бржи одговори корисницима: Брзи одговори на упите омогућавају корисницима да добију информације које траже брже, побољшавајући корисничко искуство и задовољство.
* Смањење трошкова: Оптимизација упита може смањити потребу за додатним хардверским ресурсима, што може резултирати смањењем трошкова инфраструктуре.
* Скалабилност апликације: Ефикасни упити омогућавају апликацији да скалира и ефикасно обрађује већи број корисника и већи волумен података.
* Смањење оптерећења на бази података: Спори упити могу изазвати високо оптерећење на бази података, што може довести до проблема са перформансама и доступношћу целокупне апликације. Оптимизација упита може смањити оптерећење на бази података и побољшати њену отпорност.

Када је реч о самој оптимизацији упита, она обухвата низ техника и стратегија за побољшање перформанси упита. Ове технике могу укључивати индексирање, оптимизацију SQL упита, коришћење правилних типова података, праћење плана извршења упита, оптимизацију статистика и многе друге.

### Shared Pool check

Shared Pool check је важан корак у процесу обраде SQL упита у MSSQL бази података. База генерише хеш код за сваки SQL упит и проверава да ли већ постоји оптимизовани план извршења за тај код у дељеном меморијском сегменту, познатом као " shared pool ". Ако постоји, користи се постојећи план уместо генерисања новог, што убрзава обраду упита.

База података користи алгоритам хеширања за генерисање хеш кода за сваку SQL наредбу. Хеш код наредбе је SQL ID (јединствени идентификатор). Када корисник унесе SQL наредбу, база података ће прво генерисати хеш код за тај упит. Хеш код се генерише на основу самог текста упита, тако да два идентична упита дају исти хеш код. Након генерисања хеш код, база података ће проверити да ли већ постоји оптимизовани Execution plan за тај хеш код у Shared pool -у. Овај део меморије чува оптимизоване Execution plan-ове за недавно извршене SQL упите ради поновне употребе. Ако се пронађе оптимизовани план за дати хеш код у shared pool -у, база података ће користити тај план уместо поновног генерисања новог плана. То значи да се SQL упит неће поново парсирати, оптимизовати и извршавати, већ ће се користити већ постојећи Execution plan.

Пример упита:

SELECT \* FROM Clients WHERE FirstName = 'Marko' AND LastName = 'Markovic'

SQL Server ће генерисати hash код за упит и проверити да ли постоји оптимизовани план извршења за израчунати hash код.

### Катерогије парсирања

Код Shared Pool Check унутар MS SQL Server-а, постоје две главне категорије парсирања:

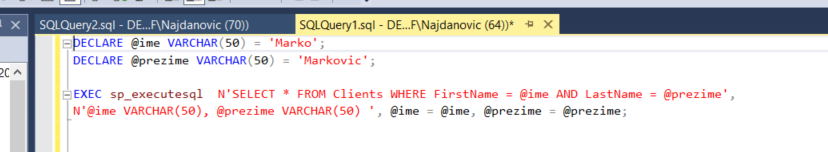
* **Hard parsing**: Када упит по први пут дође до SQL Server-а, сервер мора да изврши комплетну анализу упита, укључујући синтаксну анализу, оптимизацију плана извршења и генерисање извршног плана. Hard parsingје скупљи процес јер захтева више ресурса и времена због потребе за креирањем новог плана извршења.
* **Soft parsing**: Када SQL Server препозна да је упит већ претходно извршен и да се план извршења већ налази у Shared pool, може прескочити генерисање новог плана и директно искористити постојећи план из Shared pool-а. Овај процес захтева мање ресурса јер се прескаче већи део процеса анализе и генерисања плана.

Разлика између ове две категорије је у томе колико ресурса SQL Server троши да би припремио упит за извршавање.

### Параметризовани упити

Када SQL Server врши Shared Pool check, један од кључних фактора који одређује да ли ће се користити кеширани план или ће бити креиран нови је текст SQL упита. Ако проследите различите вредности за филтрирање (нпр. различите вредности у WHERE клаузули), SQL Server третира те упите као различите, јер се разликују у тексту.

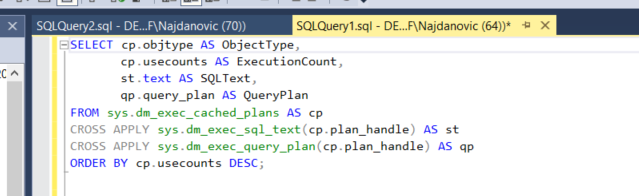
Потребно је користити параметризоване упите, да би се вредност параметара пренела независно од самог упита. На тај начин, Soft parsing може искористити исти hash код, јер је у основи исти текст упита, док само вредности параметара варирају.



Слика 2.

На овај начин смо креирали параметризовани упит, где ћемо добити исти hash код, за различите параметре упита.

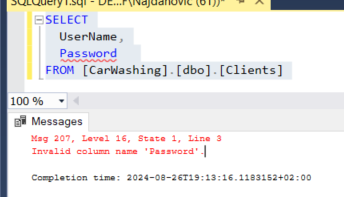
Уколико желимо да погледамо све упите који су сачувани унутар Shared Pool-а, можемо извршити ову команду:



Слика 3.

### Семантичка провера

Представља проверу смислености наредбе, тј. SQL идентификује које табеле и колоне су наведене у самом упиту и врши њихову валидацију.

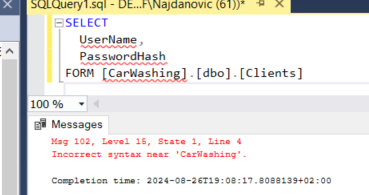


Слика 4.

SQL Server ће индентификовати грешку због проблема са колоном ‘Password’ која или не постоји у тој табели или је неправилно написана.

### Синтаксна провера

Представља проверу исправности синтаксе самог SQL упита како би се идентификовале могуће грешке.



Слика 5.

Синтаксна провера представља важан корак у самој анализији упита. SQL Server проверава да ли су све кључне речи, идентификатори, оператори исправно написани и да ли упит поштује правила SQL синтаксе. SQL Server ће генерисати грешку уколико се појави нека синтаксна грешка.

### Индекси

Индекси играју значајну улогу у брзини рада саме базе јер помажу DBMS-у да брже пронађе оно што тражимо. Омогућавају нам да брже приступимо одређеном реду у великој табели.

Основни типови индекса су:

* Clustered Index (кластеризовани индекс):
  + Кластеризовани индекс одређује физички редослед редова у табели на основу вредности кључа индекса.
  + Свака табела може имати само један кластеризовани индекс, јер одређује физички редослед података на диску.
* Non-clustered Index (некластеризовани индекс):
  + Некластеризовани индекс не мења физички редослед података у табели, већ се користи засебна структура индекса која садржи референцу на редове у табели.
  + Табела може имати више некластеризованих индекса.
* Unique Index (јединствени индекс):
  + Јединствени индекс обезбеђује јединственост вредности у одређеној колони или групи колона.
  + Може се применити на кластеризоване или некластеризоване индексе.
* Composite Index (сложени индекс):
  + Сложени индекс састоји се од више колона.
  + Омогућава ефикасније претраживање и филтрирање података који се ослањају на комбинацију вредности у више колона.
* Filtered Index (филтрирани индекс):
  + Филтрирани индекс садржи само редове који испуњавају одређени услов.
  + Корисно је када желите да индексирате само део података у табели који се често користи у упитима.

Постоје 3 корака приликом креирања индекса која треба испратити:

1. Идентификација колоне или група колона за индексирање: Прво морате одабрати које колоне желите да индексирате. То обично укључује колоне које се често користе у WHERE, JOIN или ORDER BY клаузулама.
2. Одабир типа индекса: На основу захтева перформанси и структуре ваше табеле, одаберите одговарајући тип индекса.
3. Писање СQЛ наредбе за креирање индекса: Користите CREATE INDEX наредбу да бисте креирали индекс.

Основна команда за креирање индекса је следећа:

CREATE INDEX Naziv\_Indeksa ON Naziv\_Tabele(Naziv\_Kolone);

Примери:

CREATE CLUSTERED/NONCLUSTERED/UNIQUE INDEX StudentIndex ON Students(Indeks);

CREATE INDEX StudentIndex ON Students(Indeks, Ocena); // Composite Index

CREATE INDEX StudentIndex ON Students(Indeks) WHERE Ocena >= 6; // Filtered Index

Након оптимизације упита, следи његово извршавање. MS SQL Server користи одабрани план за извршавање упита и израчунавање резултата који ће бити враћени кориснику.

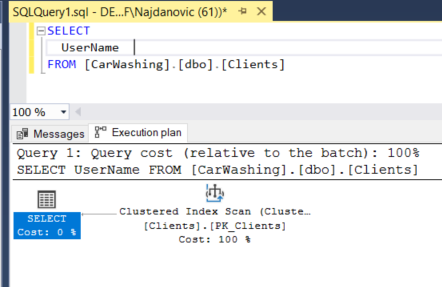
Када се упит изврши, подаци се враћају кориснику који их након тога може даље обрадити.

### Execution plan

План извршења (Execution Plan) представља стратегију коју база података користи како би ефикасно обрадила упит. Овај план дефинише како ће се подаци дохватати и обрађивати, укључујући употребу индекса, скенирање табела, редослед операција, као и методе за спајање и филтрирање података. План извршења је од суштинског значаја за оптимизацију перформанси упита, јер омогућава бази података да изабере најефикаснији пут за дохват резултата.

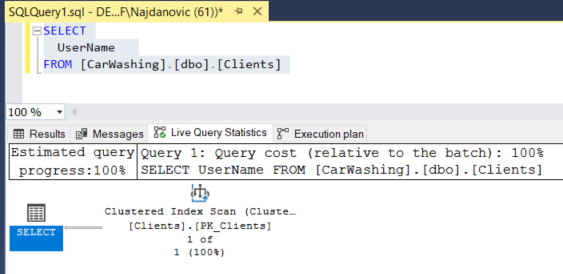
У SQL Serveru, постоји неколико начина за преглед планова извршења:

* **Процењени план извршења** (Estimated Execution Plan): Овај тип плана пружа предвиђање како SQL Server планира да изврши упит. Оптимизатор упита анализира упит и генерише стратегију пре него што се стварно изврши, заснивајући се на статистици и структури података. План је индикативан и не мора увек бити идентичан стварном извршењу.



Слика 6. – Estimated Execution Plan

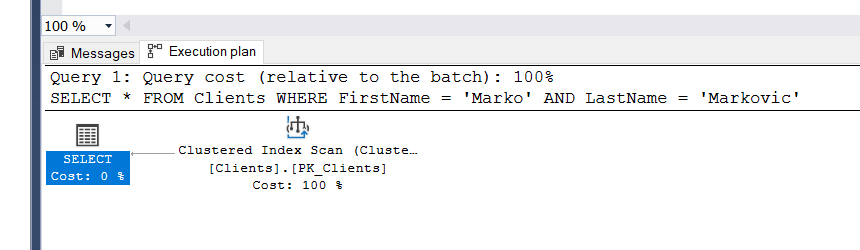
* **Стварни план извршења** (Actual Execution Plan): Овај план се генерише након извршења упита и пружа детаљне информације о томе како је упит заиста обрађен. Стварни план приказује све операције које су се десиле током извршавања, укључујући тачан број редова који су прошли кроз операторе, и може открити разлике у односу на процењени план.
* **Статистика упита уживо** (Live Query Statistics): Ова опција омогућава праћење напретка упита у реалном времену, док се извршава. Поред информација које нуди стварни план, статистика уживо пружа увид у временски ток извршавања и ажурира се током сваке секунде, показујући тренутни статус операција.



Слика 6. – Live Query Statistics

Унутар SSMS, постоје три формата прегледа планова извршења (Execution Plan):

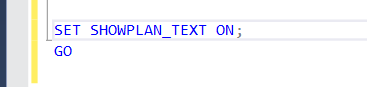
* **Графички приказ**: Ово је најчешће коришћен формат који приказује план извршења у визуелном облику. Операције у плану се приказују као чворови или иконе, које су повезане стрелицама, приказујући ток података кроз упит. Сваки оператор има своје визуално представљање, а детаљи о операцији се могу видети преласком миша преко иконе. Графички приказ омогућава лакшу анализу и идентификацију проблема са перформансама. Корисници могу брзо визуализовати најзахтевније операције (као што су скенирање табела или спајање) и идентификовати уска грла.



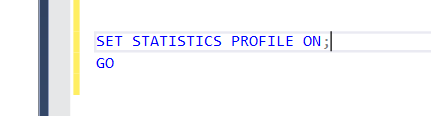
Слика 7. – Графички приказ

* **Текстуални приказ**: Приказује план извршења у чистом текстуалном формату, где се кораци извршења описују кроз низ линија текста. Свака линија описује операцију (као што је скенирање индекса, спајање табела, итд.) и приказује детаље попут процењеног броја редова и трошкова за сваки оператор. Текстуални приказ је користан за оне који желе да виде тачну структуру извршавања упита у једноставном формату без графичких елемената. Често се користи у окружењима где је приступ GUI (графичком интерфејсу) ограничен или када је потребно анализирати планове у командној линији.

Команда да се укључи текстуални приказ:

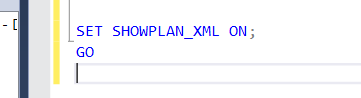


Слика 8. - Процењени план извршења



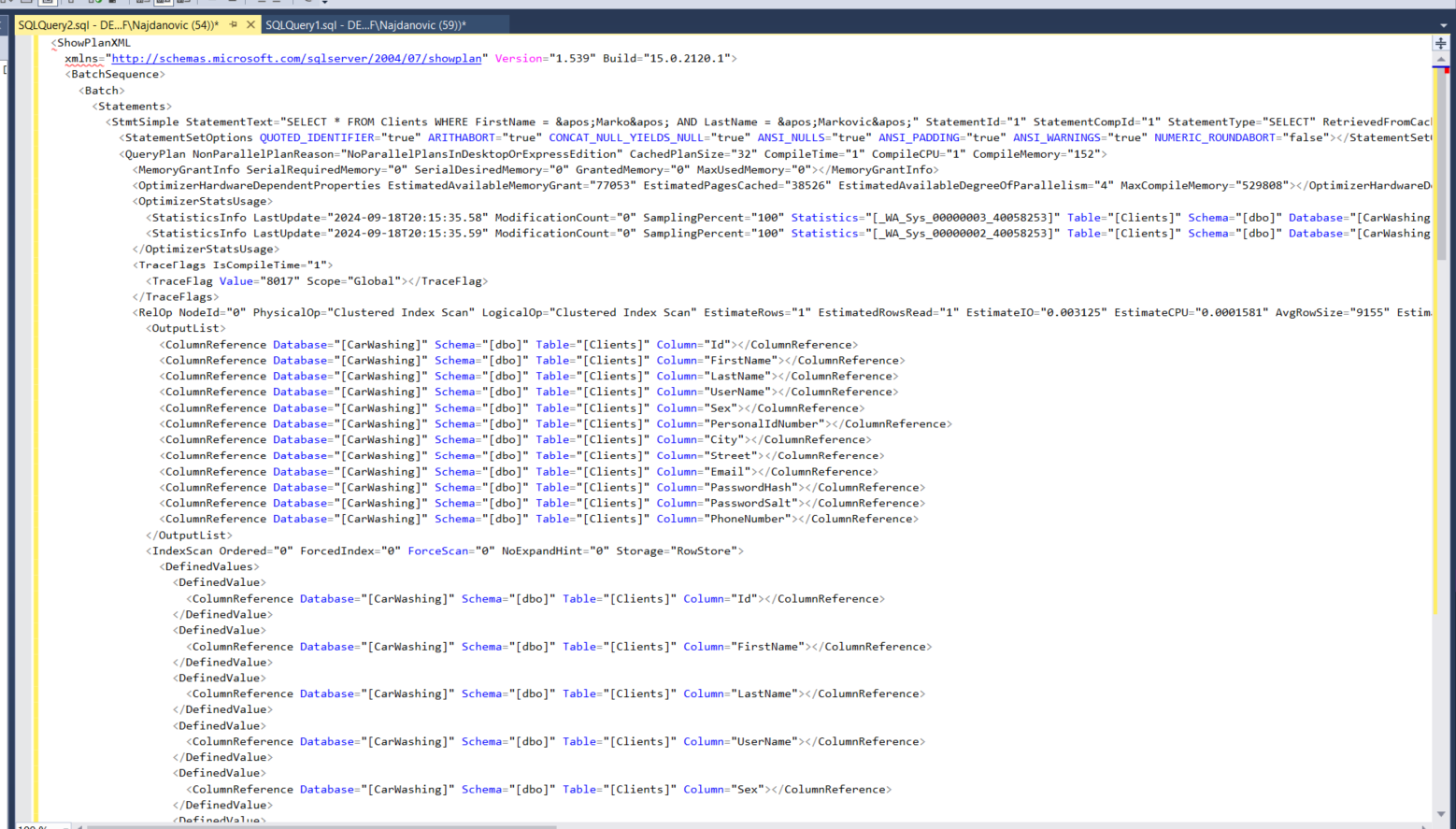
Слика 9. - Стварни план извршења

* **XML приказ**: Овај формат приказује план извршења у облику XML документа. XML план садржи све информације о плану извршења, укључујући операције, процене трошкова, број редова и друге релевантне параметре. XML приказ се користи за детаљну анализу и експорт планова извршења. Овај формат је посебно користан за аутоматску анализу планова помоћу алата или скрипти, јер се подаци лако могу обрадити програмима или анализирати користећи друге апликације.



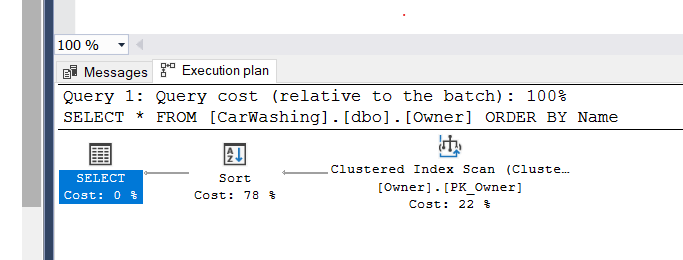
Слика 10. - Процењени план извршења

На слици испод, можемо да видимо како изгледа XML приказ.



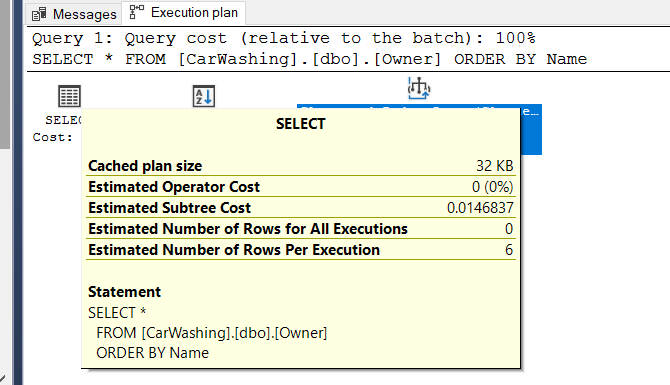
Када се приказује процењени план извршења (Estimated Execution Plan), фазе које си поменуо заправо представљају операције које SQL Server користи за извршавање упита. Те операције се приказују као кораци унутар графичког плана.

* **Clustered Index Scan**: SQL Server користи ову операцију када треба да скенира све редове унутар индексиране табеле или погледа, јер тражени подаци нису селективно дохваћени преко индекса. Када се користи Clustered Index Scan, сви подаци из табеле пролазе кроз претрагу. Ово се дешава када упит не користи ефикасан WHERE филтер или када не постоји одговарајући индекс.
* **Data Flow from Clustered Index Scan** (стрелица): Стрелице у графичком плану приказују ток података између различитих операција. Дебљина стрелице означава број редова који пролазе из једне операције у другу. Након што Clustered Index Scan добије податке, они се преносе на следећи оператор Sort.
* **Sort оператор**: Оператор сортира податке према одређеним критеријумима, обично по клаузули ORDER BY. Ако подаци нису већ сортирани, SQL Server користи овај оператор да их распореди у одговарајући редослед. Sort Operator може бити захтеван по питању ресурса, посебно ако се ради о великим скуповима података који нису унапред индексирани на начин који одговара критеријумима сортирања.
* **Data Flow from Clustered Index Scan** (стрелица): Као и код Clustered Index Scan, стрелица овде показује колико редова пролази из СSort Operator-a према следећем кораку, што је обично коначна селекција података или нека друга операција.
* **Select оператор**: Ово је завршни корак у плану извршења, који представља коначну селекцију података и враћање резултата кориснику. Након што сви подаци прођу кроз претходне операције, Select Operator враћа коначни скуп података као резултат упита.



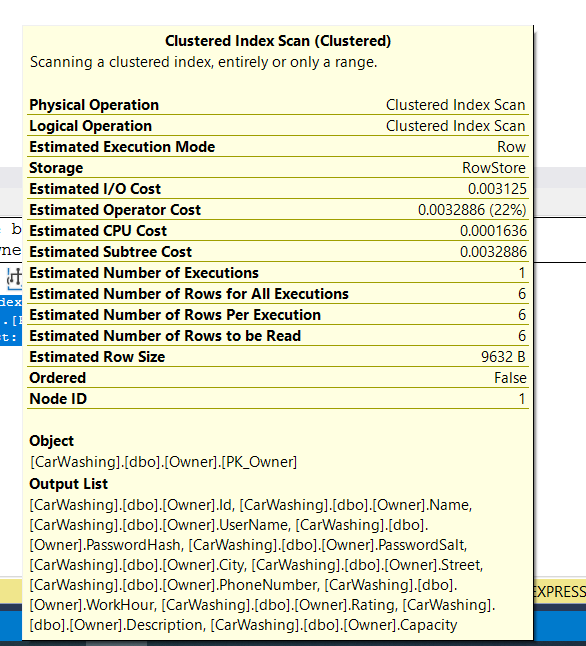
Слика 11. Приказ операција

Уколико пређемо преко Select оператор-а можемо да видимо процењени и стварни број редова, као и процењену величину података коју нам је упит вратио.



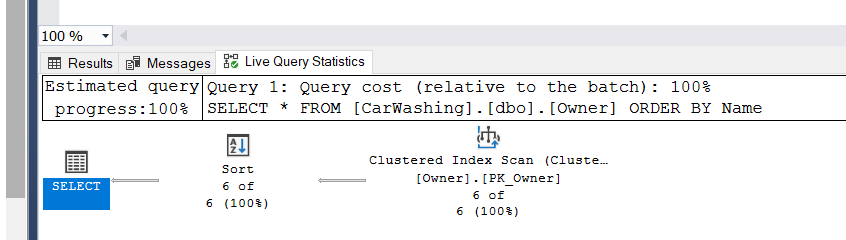
Слика 12. - Select Operator

Слика испод, нам показује детаљни приказ резултата упита.



Слика 13. – Детаљни приказ

Поред тога, можемо у укључити Live Execution Plan, који нам даје стварне податке.



Слика 14. – Live Execution Plan