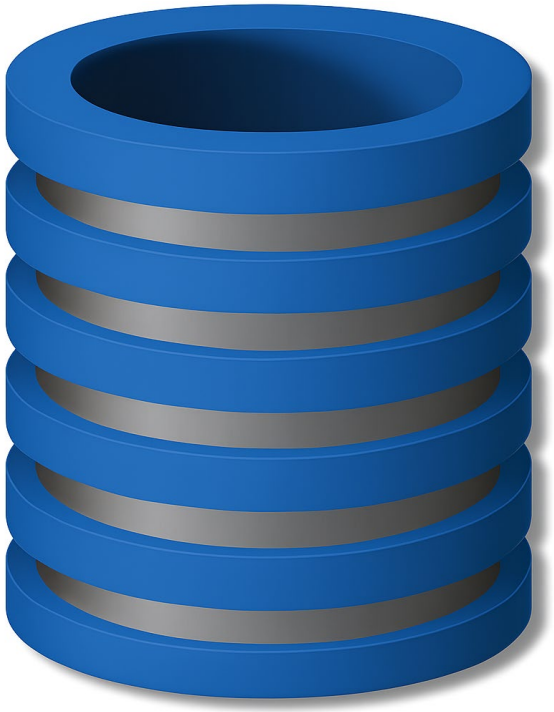


OPTIMIZACIJA UPITA

Predmet: Administriranje Baze Podataka

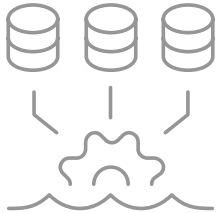
Predavač: dr Dušan Stefanović

UTICAJ KOMPLEKSNIH UPITA NA PERFORMANSE

U realnim aplikacijama, baze podataka se često suočavaju sa velikim brojem kompleksnih upita koji, ako nisu pravilno napisani, mogu dovesti do:

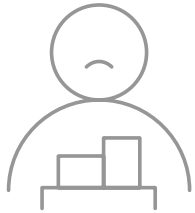
Sporo odazivanje

Sistem sporo odgovara na zahteve upućene bazi podataka.



Zagušenje baze

Preopterećenje baze dovodi do degradacije performansi.



Kašnjenje korisnika

Krajnji korisnici doživljavaju duža čekanja.

Preopterećenje servera

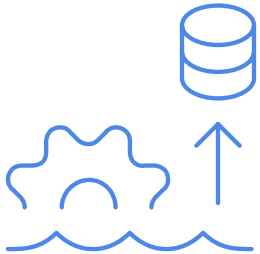
Server je opterećen, što potencijalno može uzrokovati prekid rada aplikacija.

OPTIMIZACIJA SQL UPITA

Optimizacija upita je proces identifikacije i primene najboljih SQL tehnika kako bi se:

Smanjeno vreme izvršavanja

Optimizacija upita smanjuje vreme potrebno za njihovo izvršavanje.



Smanjenje korišćenja resursa

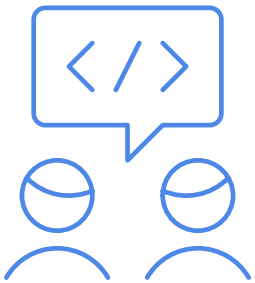
Optimizacija smanjuje korišćenje CPU-a, memorije, diska i mreže.

Poboljšana stabilnost sistema

Optimizacija poboljšava skalabilnost i ukupnu stabilnost sistema.

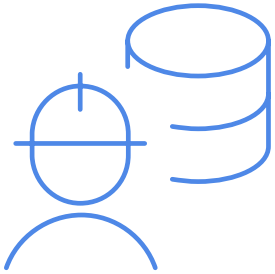
KO SE BAVI OPTIMIZACIJOM UPITA

Poznavanje optimizacije upita je vrlo bitna za:



Developeri

Žele da njihove aplikacije rade brzo i pouzdano



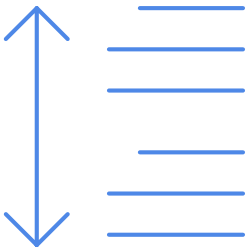
Administratori baza podataka

Administratori sistema koji održavaju performanse sistema.

PLAN IZVRŠENJA UPITA

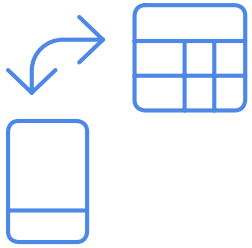
Query optimizer (plan izvršenja) je komponenta baze podataka koja automatski bira najbolju strategiju izvršenja upita.

Koristi različite algoritme za:



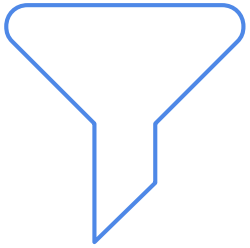
Sortiranje

Uređivanje podataka u određenom redosledu.



Spajanje tabela

Kombinovanje podataka iz više tabela.



Filtriranje

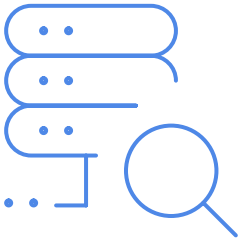
Odabir podataka na osnovu specifičnih kriterijuma.

Korišćenje indeksa

Iskorišćavanje indeksa za ubrzavanje upita.

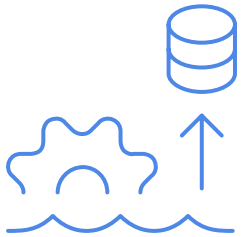
PLAN IZVRŠENJA UPITA

Da bi se pisali efikasniji upiti potrebno je razumeti:



Funkcija optimizatora upita

Razumevanje načina na koji optimizator upita funkcioniše je ključno.



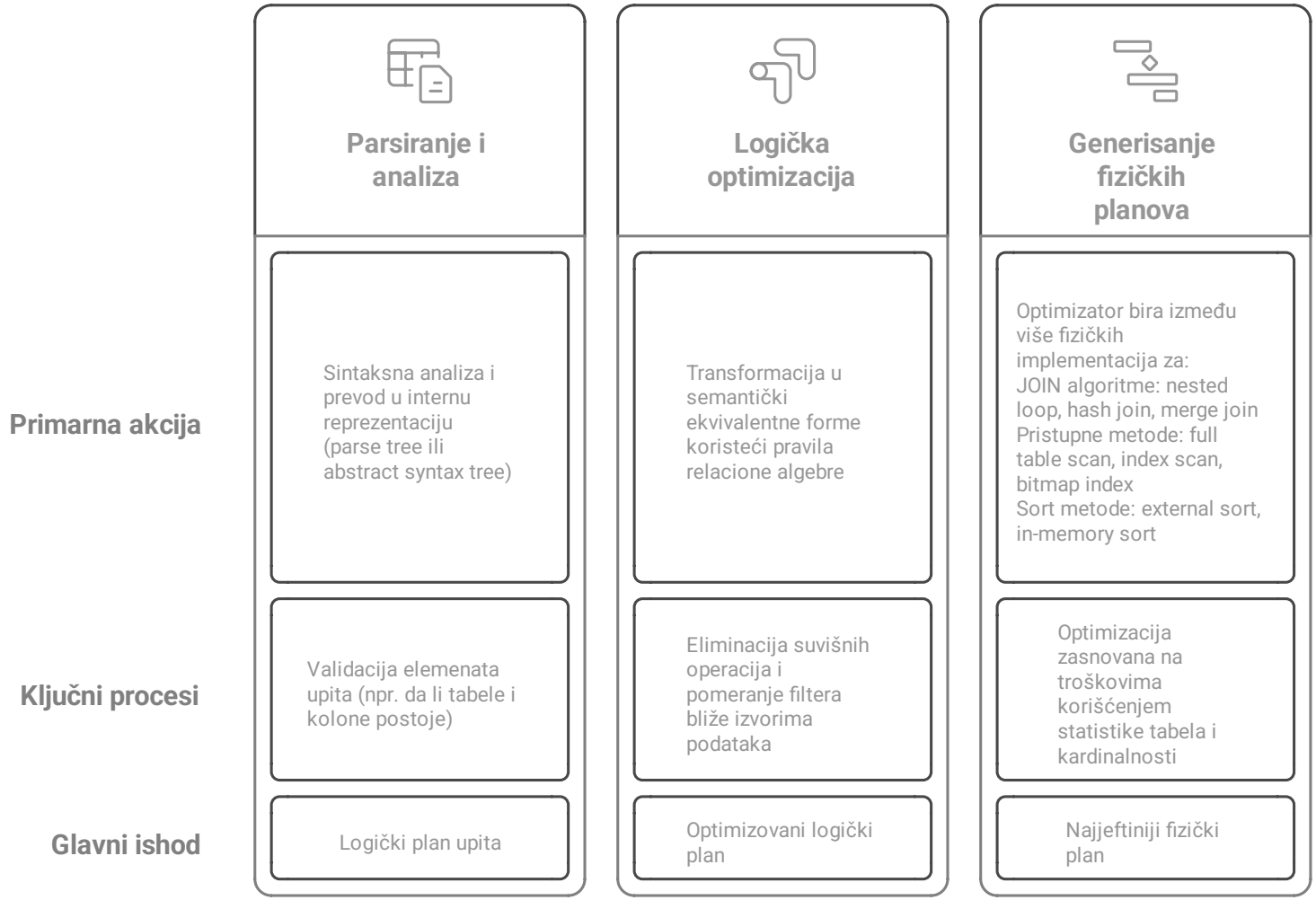
Korišćenje algoritama

Poznavanje algoritama koji se koriste za planove izvršenja upita.

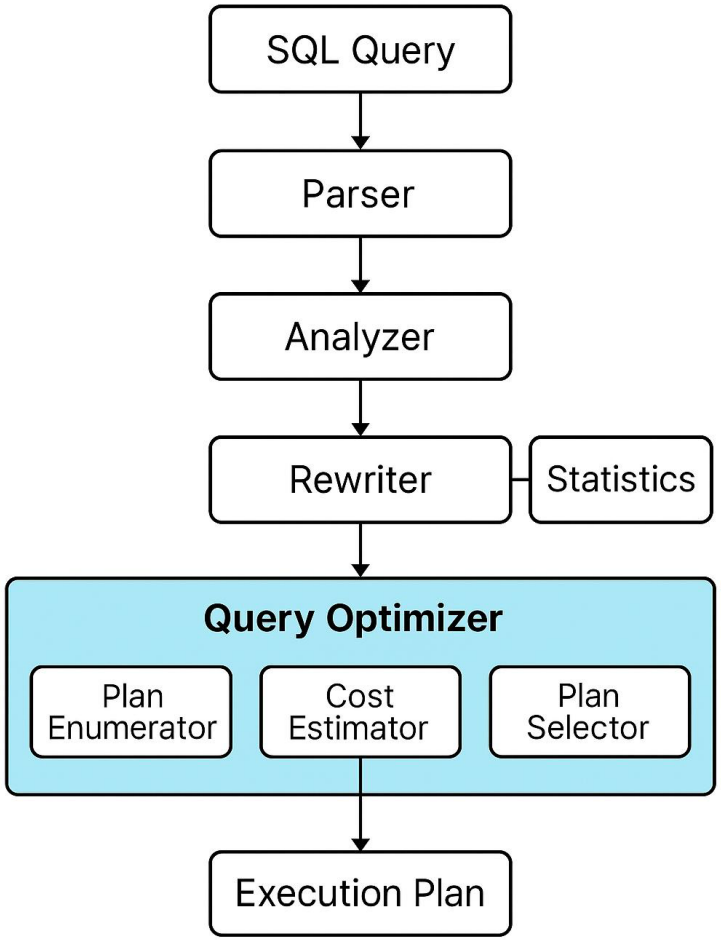
Poređenje planova izvršenja

Poređenje planova izvršenja različito napisanih upita.

ARHITEKTURA QUERY OPTIMIZERA



KOMPONENTE QUERY OPTIMIZERA



Komponenta	Opis
Parser	Pretvara SQL upit u interno stablo izraza
Analyzer	Proverava semantiku, tumači nazive tabela i kolona
Rewriter	Primenjuje pravila relacione algebre radi logičke optimizacije
Statistics Manager	Pruža statističke podatke (npr. broj redova, selektivnost kolone)
Plan Enumerator	Generiše sve moguće fizičke planove izvršenja
Cost Estimator	Izračunava procenjeni trošak svakog plana (I/O, CPU itd.)
Plan Selector	Bira najbolji plan na osnovu troška

SLUČAJ 1

Korišćenje kolona

Upotreba imena kolona umesto * u select iskazu

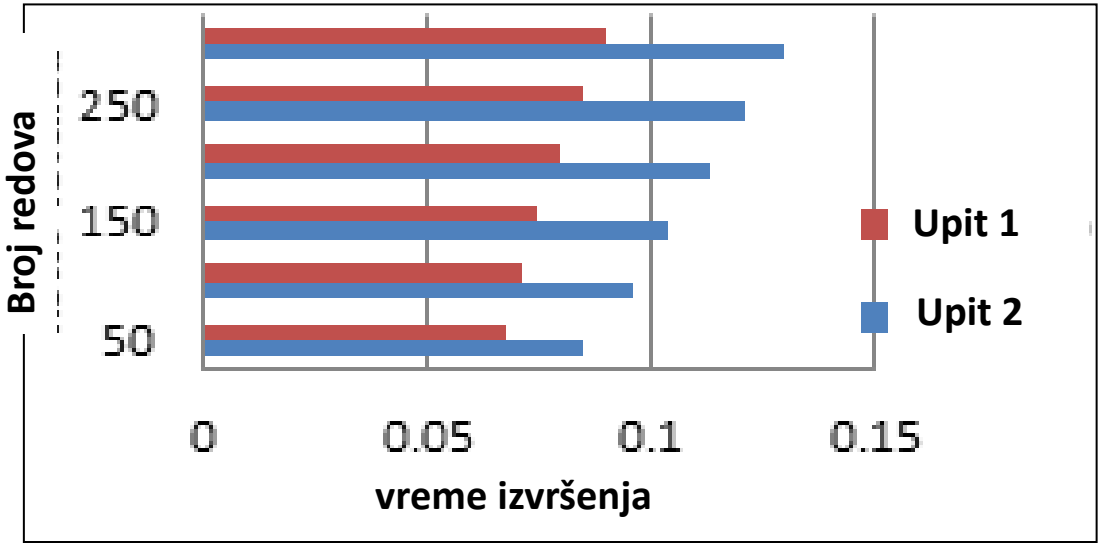
Obim podataka

Štedi se na obimu prikazanih podataka,

Pronalaženje podataka

Poboljšava ukupnu brzinu pronalaženja podataka.

27% je ušteda u vremenu izvršenja upita



Upit 1:
```
SELECT s.prod_id
FROM sales s;
```

Upit 2:
```
SELECT *
FROM sales;
```

SLUČAJ 2

VS

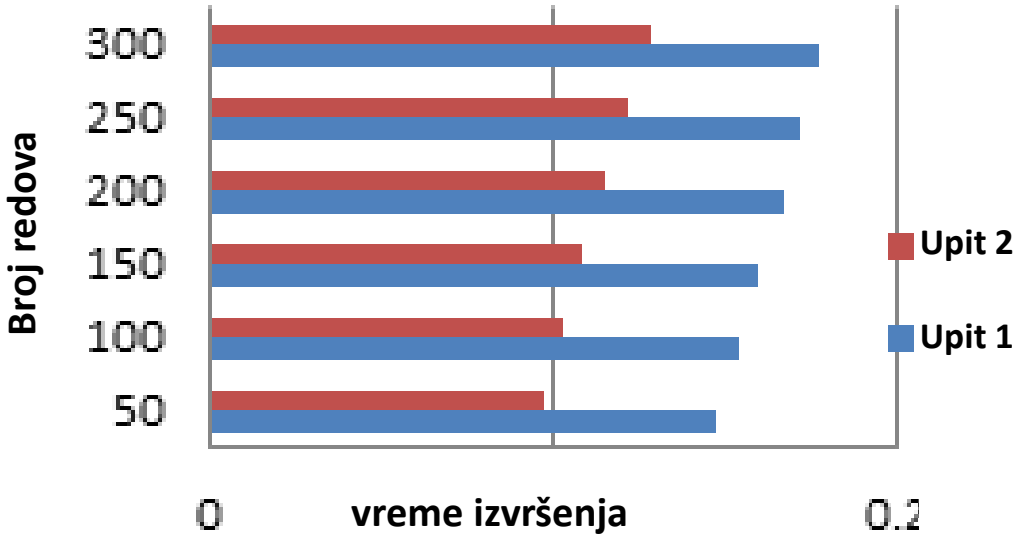
Izbegavajte HAVING

U select upitima gde nam postavka zadatka to dozvoljava

Koristite HAVING

Samo kada je neophodno u kompleksnijim zahtevima

31% je ušteda u vremenu izvršenja upita




```
SELECT s.cust_id, count(s.cust_id)
FROM sales s
GROUP BY s.cust_id
HAVING s.cust_id != '1660' AND s.cust_id != '2';
```

```
SELECT s.cust_id, count(cust_id)
FROM sales s
WHERE s.cust_id != '1660'  AND s.cust_id !='2'
GROUP BY s.cust_id;
```

SLUČAJ 3

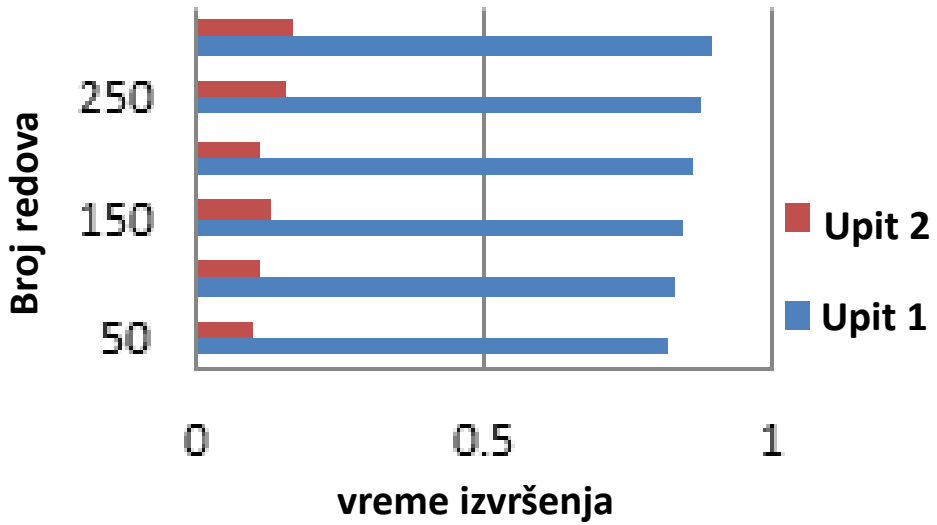
Upiti sa DISTINCT

Osigurava jedinstvene rezultate, ali može biti nepotrebno sa primarnim ključevima jer utiče negativno na performanse

Upiti bez DISTINCT

Smanjuje nepotrebnu obradu, posebno ako se koriste sa primarnim ključevima

85% je ušteda u vremenu izvršenja upita




```
SELECT DISTINCT * FROM sales s
JOIN customers c
ON s.cust_id= c.cust_id
WHERE c.cust_marital_status = 'single';
```

```
SELECT * FROM sales s JOIN
customers c
ON s.cust_id = c.cust_id
WHERE c.cust_marital_status='single';
```

SLUČAJ 4

VS

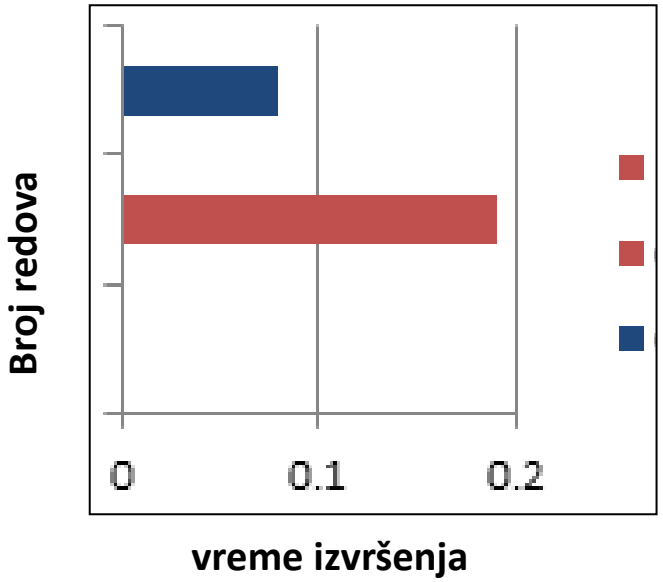
JOIN

Efikasniji u pogledu izvršenja upita. Omogućavaju optimizatoru upita da pravi efikasne planove izvršenja

Subquery

Kod podupita (posebno korelisanih), optimizer mora da izvršava unutrašnji upit za svaki red spoljnog upita.

61% je ušteda u vremenu izvršenja upita




```
SELECT *
FROM SH.products p
WHERE p.prod_id = (SELECT s.prod_id
FROM SH.sales s
WHERE s.cust_id = 100996 AND s.quantity_sold = 1 )
```

```
SELECT p.*
FROM SH.products p, sales s
WHERE p.prod_id = s.prod_id
AND s.cust_id = 100996 AND s.quantity_sold = 1;
```

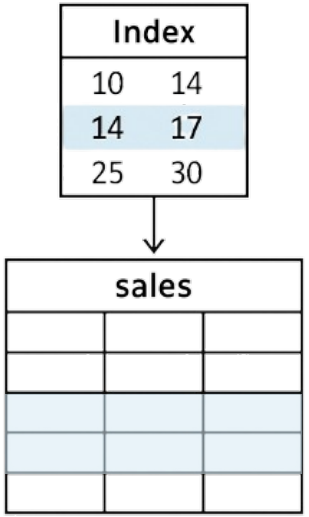
SLUČAJ 5

IN

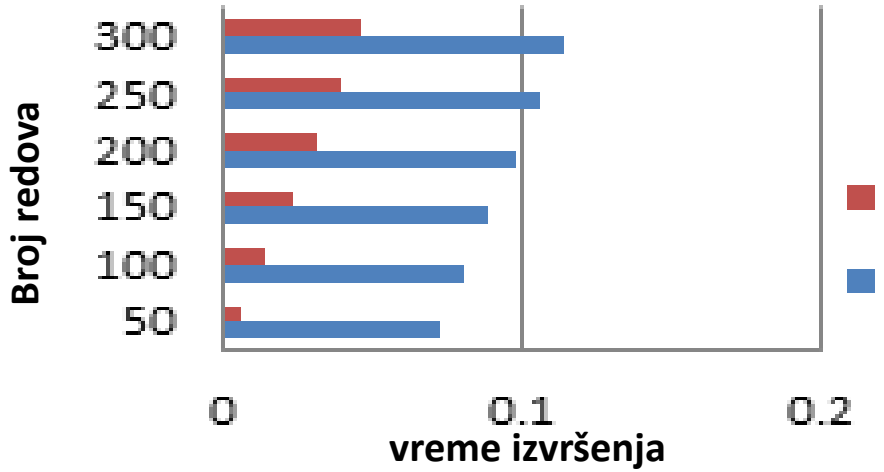
Primena IN iskaza nad indeksiranom kolonom je efikasnija jer optimajzer sortira IN listu da mečuje sortiranu sekvencu indeksa IN se često može paralelizovati kao grupna pretraga,

OR

Zahteva evaluaciju po pojedinačnom izrazu.



73% je ušteda u vremenu izvršenja upita




```
SELECT s.*
FROM sales s
WHERE s.prod_id = 14 OR s.prod_id = 17;
```

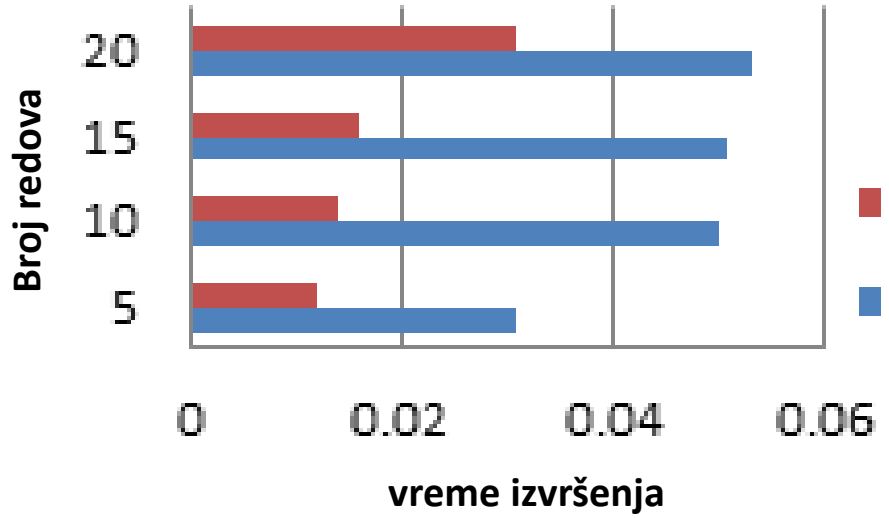
```
SELECT s.*
FROM sales s
WHERE s.prod_id IN (14, 17);
```

SLUČAJ 6

Koristiti EXISTS umesto DISTINCT kada se spajaju tabele u relaciji 1:M

DISTINCT prikazuje sve podatke a zatim izbacuje duplikate za razliku od podupita sa EXISTS gde se ne vraća cela tabela

61% je ušteda u vremenu izvršenja upita




```
SELECT DISTINCT c.country_id, c.country_name
FROM countries c, customers e
WHERE e.country_id = c.country_id;
```

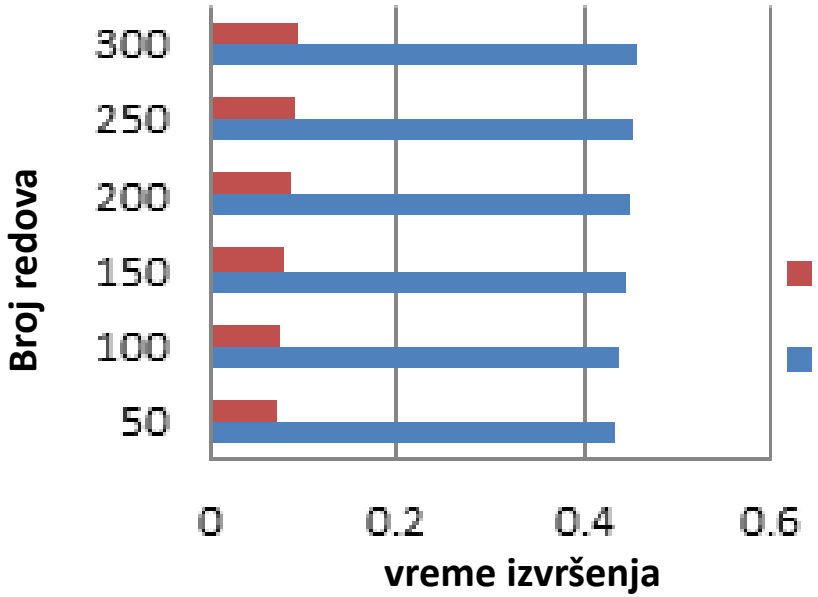
```
SELECT c.country_id, c.country_name
FROM countries c
WHERE EXISTS (SELECT 'X' FROM
                customers e
              WHERE e.country_id = c.country_id);
```

SLUČAJ 7

Koristiti UNION ALL umesto UNION

UNION ALL se brže izvršava jer ne traži duplikate za razliku od UNION koji traži duplikate bez obzira da li postoje ili ne postoje.

81% je ušteda u vremenu izvršenja upita




```
SELECT cust_id
FROM sales
UNION
SELECT cust_id
FROM customers;
```

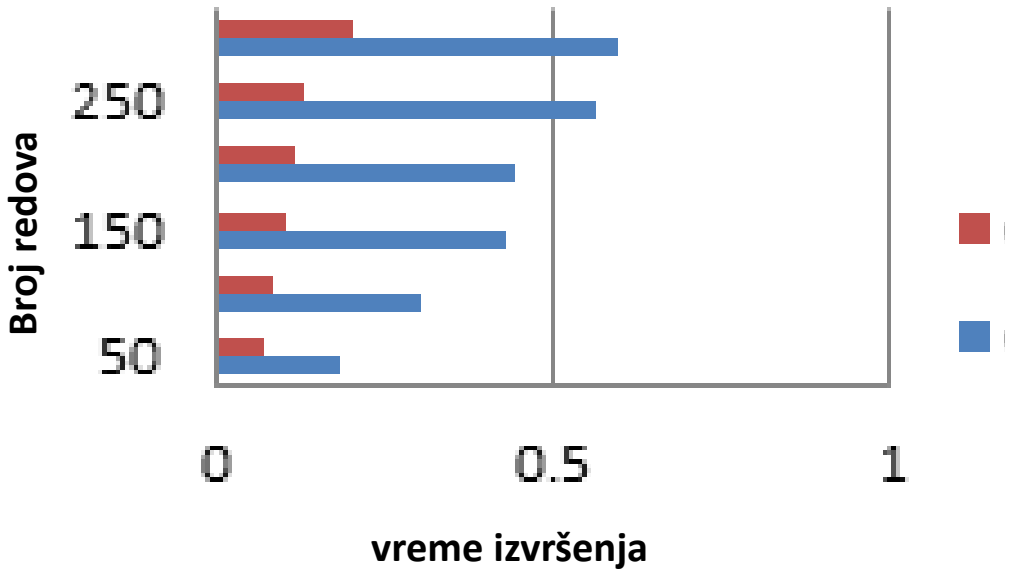
```
SELECT cust_id
FROM sales
UNION ALL
SELECT cust_id
FROM customers;
```

SLUČAJ 8

Izbegavanje OR u JOIN iskazima

Uvek kada se OR javi u JOIN, upit će se usporiti najmanje sa faktorom 2

70% je ušteda u vremenu izvršenja upita




```
SELECT *
FROM costs c
INNER JOIN products p ON c.unit_price =
p.prod_min_price OR c.unit_price =
p.prod_list_price;
```

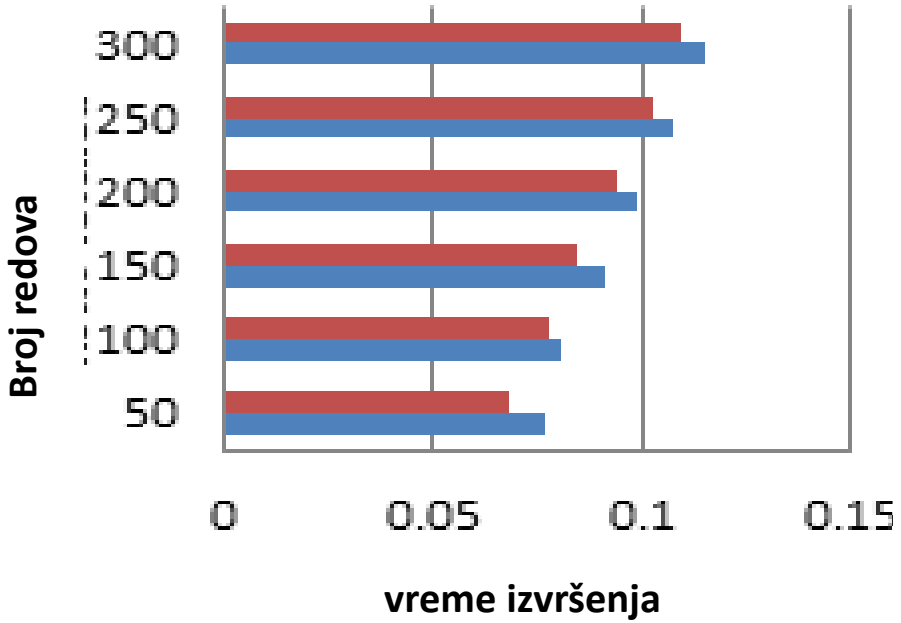
```
SELECT *
FROM costs c
INNER JOIN products p ON c.unit_price =
p.prod_min_price
UNION ALL
SELECT *
FROM costs c
INNER JOIN products p ON c.unit_price =
p.prod_list_price;
```

SLUČAJ 9

Izbegavati redudatnu matematiku

Matematika u SQL iskazu može znatno da smanji performanse upita jer se operacija primenjuje za svaki red.

11% je ušteda u vremenu izvršenja upita




```
SELECT *
FROM sales s
WHERE s.cust_id + 10000 < 35000;
```

```
SELECT *
FROM sales s
WHERE s.cust_id < 25000;
```

Indeksiranje

Indeksiranje kolone je najčešći način da se optimizuje čitanje podataka

Potrebno je detaljno razumeti kako indeksiranje radi u bazi da bi mogli u potpunosti da se iskoriste.

Aritmetički operatori

SELECT * FROM TABLE WHERE COLUMN > 16

optimizovan upit
SELECT * FROM TABLE WHERE COLUMN >= 17

Važi za slučaj da je kolona indeksirana

WILDCARD

Upotreba wildcard operatora znatno usporava izvršenje upita posebno ukoliko tabela sadrži veliki broj redova.

Upit se može optimizovati ukoliko se koristi **postfix** wildcard umesto **full** ili **pre** wildcard nad indeksiranom kolonom

```
SELECT * FROM TABLE WHERE COLUMN LIKE '%srt%';
SELECT * FROM TABLE WHERE COLUMN LIKE  ’srt%';
SELECT * FROM TABLE WHERE COLUMN LIKE  '%srt';
```

full wildcard

postfix wildcard

prefix wildcard

NOT OPERATOR

Izbegavati NOT operator u SQL-u

Čitanje je znatno brže ukoliko se koriste tzv. pozitivni operatori LIKE, IN, EXIST ili = umesto negativnih operatora NOT LIKE, NOT IN, NOT EXIST ili !=.

COUNT vs EXIST

Postoji slučajevi gde se COUNT operator koristi da bi se utvrdilo da li je podatak prisutan

Bolje je rešenje da se koristi EXIST koji će završiti pretragu čim pronađe prvi zapis.

Union umesto OR

Indeksi gube prednost u brzini kada se koriste u OR sutuacijama

```
SELECT * FROM TABLE WHERE COLUMN_A = 'value' OR COLUMN_B = 'value'
```

```
SELECT * FROM TABLE WHERE COLUMN_A = 'value'
UNION
SELECT * FROM TABLE WHERE COLUMN_B = 'value'
```

Indeksiranje Stringa

Nije neophodno da se indeksira ceo string ukoliko prefix ili postfix stringa mogu da se indeksiraju

Posebno ukoliko prefix ili postfix stringa obezbeđuju jedinstven identifikator za string, preporuka je da se koristi takvo indeksiranje

Kraći indeksi su brži ne samo što zahtevaju manje prostora na disku, već lakše mogu da se nađu u indeks kešu.

In Subquery

```
SELECT * FROM TABLE
WHERE COLUMN IN (SELECT COLUMN
                 FROM TABLE)
```

```
SELECT *
FROM TABLE, (SELECT COLUMN FROM TABLE) as dummytable
WHERE dummytable.COLUMN = TABLE.COLUMN;
```