Optimizacija Upita

- SQL je standardni jezik u radu sa bazama podataka
  - CRUD (Create, Read, Update i Delete)
- Problem - upiti koji se dugo izvršavaju (slow response time)
  - Zagušenje sistema
  - Upit nije algoritamski dobro napisan da se izvrši za kraće vreme

OPTIMIZACIJA UPITA

- Poznavanje optimizacije upita je vrlo bitna za SQL developere i DB administratore
- Da bi se pisali efikasniji upiti potrebno je razumeti na koji način Query optimajzer (query optimizer) radi i tehnike koje koristi za izradu plana izvršenja upita (query execution plan).
- Najčešće se upiti pišu na više načina a zatim se poredi njihov execution plan

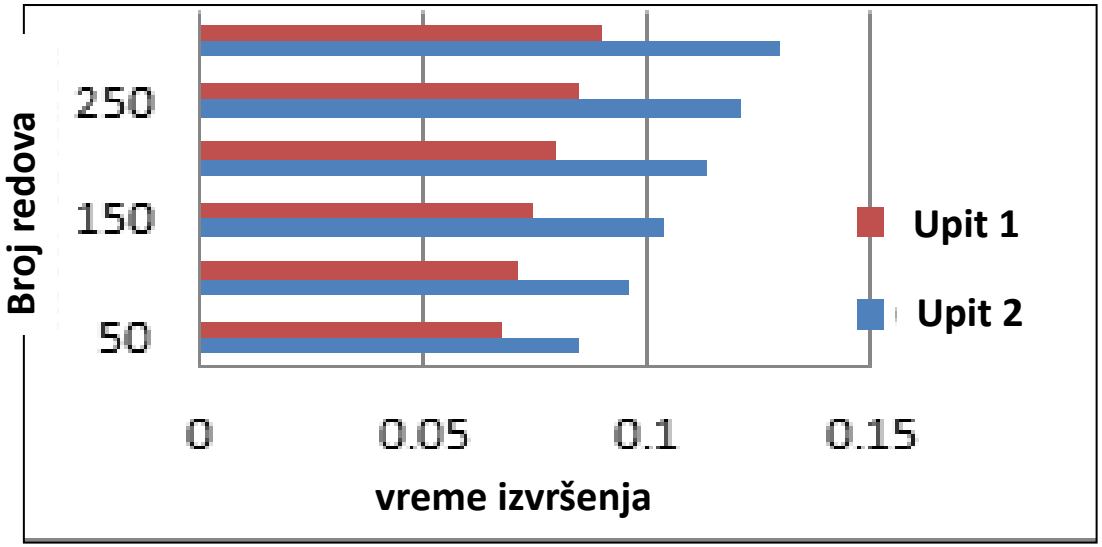
OPTIMIZACIJA UPITA

- Optimizaciju upita rade administratori baze podataka i dizajneri aplikacije da bi poboljšali performanse baze podataka.
- Performasne aplikacije mogu da budu značajno smanjene od strane neefikasnih upita.
- Optimizacija upita prvenstveno utiče na brži rad DBMS-a.

SLUČAJ 1

- Upotreba imena kolona umesto * u Select iskazu
- Ukoliko želimo da prikažemo samo određene kolone iz tabele poželjno je ne koristiti SELECT * i ako je ovu komandu lakše napisati
  - Štedi se na obimu prikazanih podataka, smanjuje se mrežni saobraćaj i ukupna brzina pronalaženja podataka

27% je ušteda u vremenu izvršenja upita



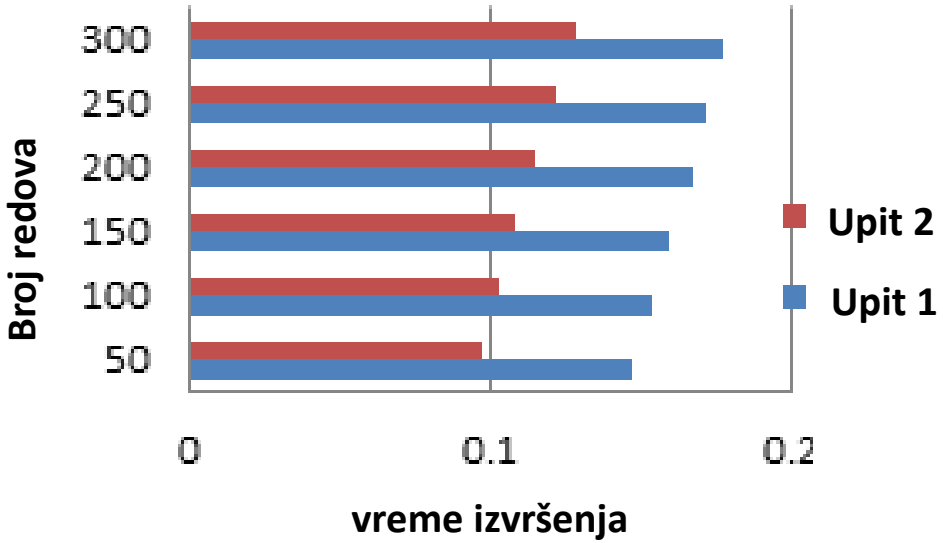
Upit 1:
SELECT s.prod_id FROM SH.sales s;

Upit 2:
SELECT * FROM SH.Sales;

SLUČAJ 2

- Izbegavanje HAVING iskaza u SELECT upitima gde nam postavka zadatka to dozvoljava

31% je ušteda u vremenu izvršenja upita



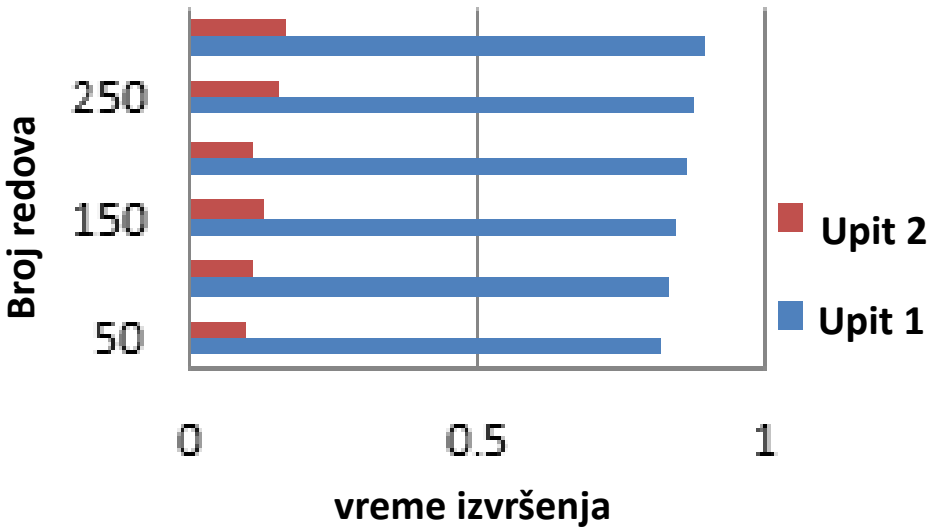

```
SELECT s.cust_id,count(s.cust_id)
FROM SH.sales s
GROUP BY s.cust_id
HAVING s.cust_id != '1660' AND s.cust_id != '2';
```

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

SLUČAJ 3

- Izbegavanje DISTINCT iskaza posebno u situacijama kada je rezlutat isti sa i bez DISTINCT ključne reči a to se dešava kada u rezlutatu imamo kolonu koja je primarni ključ

85% je ušteda u vremenu izvršenja upita



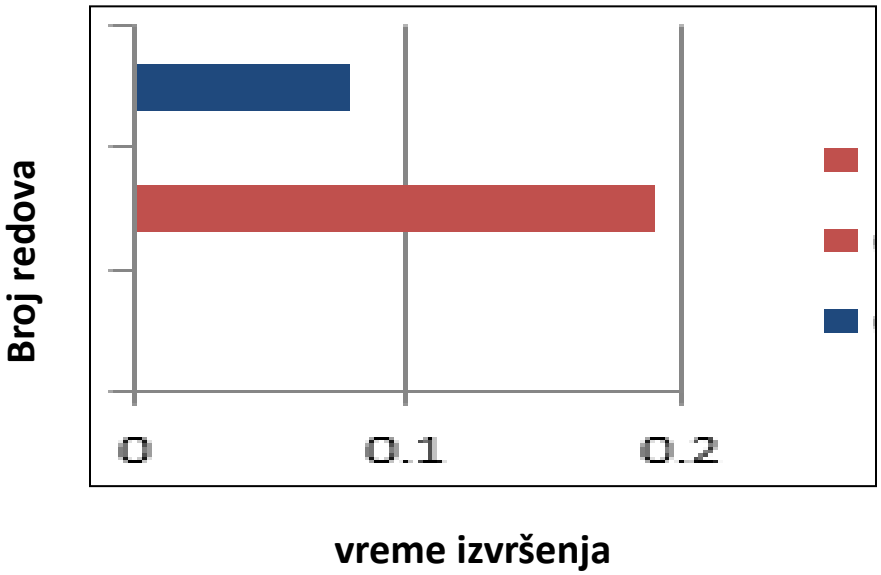

```
SELECT DISTINCT * FROM SH.sales s
JOIN SH.customers c
ON s.cust_id= c.cust_id
WHERE c.cust_marital_status = 'single';
```

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

SLUČAJ 4

- Korišćenje JOIN iskaza umesto ugnježdenih upita je efikasnije u pogledu izvršenja

61% je ušteda u vremenu izvršenja upita



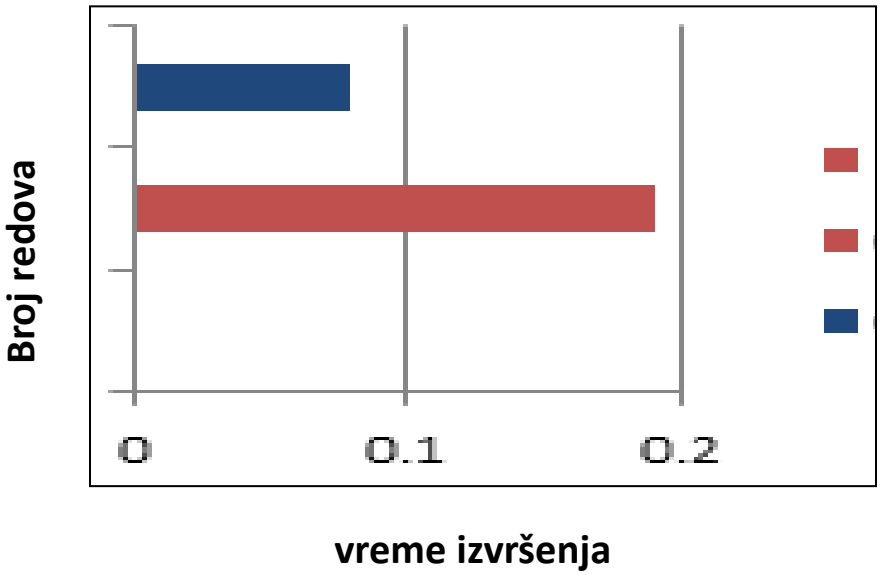

```
SELECT *
FROM SH.products p
WHERE p.prod_id =
(SELECT s.prod_id
FROM SH.sales s
WHERE s.cust_id = 100996 AND s.quantity_sold = 1 )
```

```
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

- Korišćenje JOIN iskaza umesto ugnježdenih upita je efikasnije u pogledu izvršenja

61% je ušteda u vremenu izvršenja upita



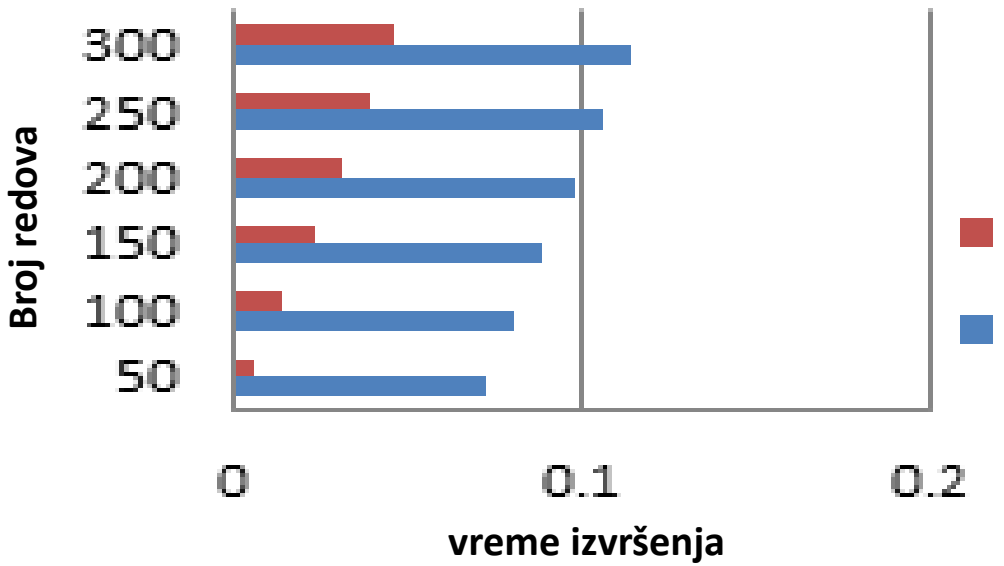

```
SELECT *
FROM SH.products p
WHERE p.prod_id =
(SELECT s.prod_id
FROM SH.sales s
WHERE s.cust_id = 100996 AND s.quantity_sold = 1 )
```

```
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 6

- Primena IN iskaza nad indeksiranom kolonom je efikasnija jer optimajzer sortira IN listu da mečuje sortiranu sekvencu indeksa

73% je ušteda u vremenu izvršenja upita



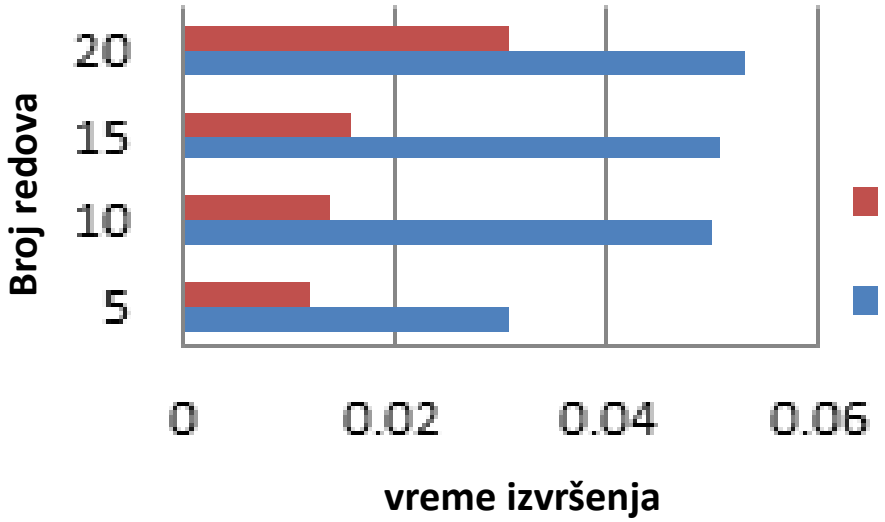

```
SELECT s.*
FROM SH.sales s
WHERE s.prod_id = 14 OR s.prod_id = 17;
```

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

SLUČAJ 7

- 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



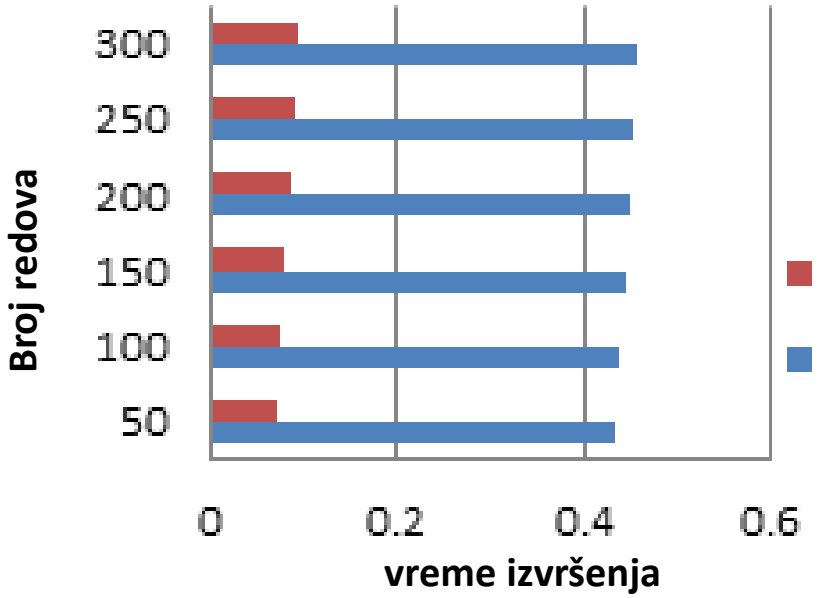

```
SELECT DISTINCT c.country_id, c.country_name
FROM SH.countries c,SH.customers e
WHERE e.country_id = c.country_id;
```

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

SLUČAJ 8

- 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



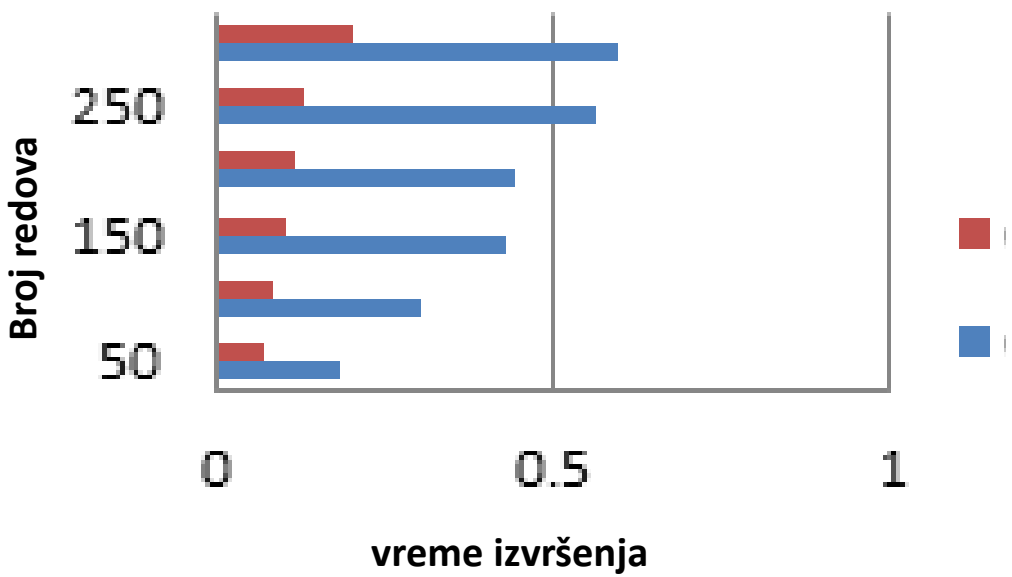

```
SELECT cust_id
FROM SH.sales
UNION
SELECT cust_id
FROM customers;
```

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

SLUČAJ 9

- 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



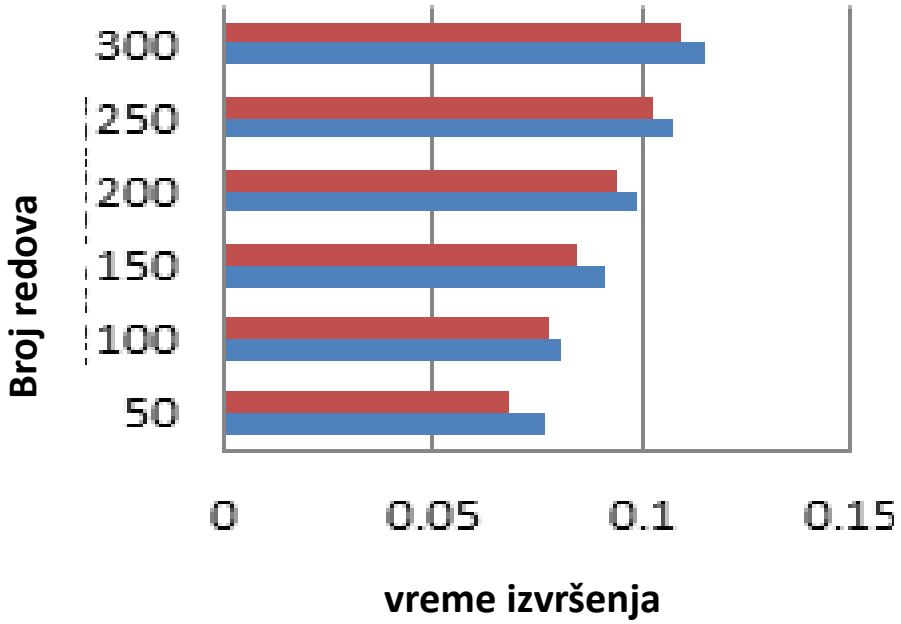

```
SELECT *
FROM SH.costs c
INNER JOIN SH.products p ON c.unit_price =
p.prod_min_price OR c.unit_price =
p.prod_list_price;
```

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

SLUČAJ 10

- 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 SH.sales s
WHERE s.cust_id + 10000 < 35000;
```

```
SELECT *
FROM SH.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.

```
SELECT COLUMN FROM TABLE
WHERE COUNT(COLUMN) > 0
```

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;
```