

1. Relacija *stipendija* sadrži podatke o rezultatima natječaja za stipendije. Za svakog studenta unaprijed je definiran iznos stipendije (*predvIznos*), a hoće li mu stipendija zaista biti odobrena ovisi o njegovom rangu i ukupnom raspoloživom iznosu sredstava.

Stipendije se odobravaju redoslijedom određenim rangom studenta. Odobravanje stipendija se prekida kada se ukupni raspoloživi iznos sredstava potroši u cijelosti ili preostali iznos nije dovoljan da bi podmirio predviđeni iznos stipendije studenta koji je na redu za dodjelu stipendije.

Napisati pohranjenu proceduru **odobriStip** koja prima ulazni parametar *raspolozivIznos*. Procedura odobrava stipendije studentima kojima stipendija još nije odobrena tako što im vrijednost atributa *odobrenaStip* postavlja na vrijednost 'D'. Procedura kao rezultat treba vratiti nedodijeljeni iznos.

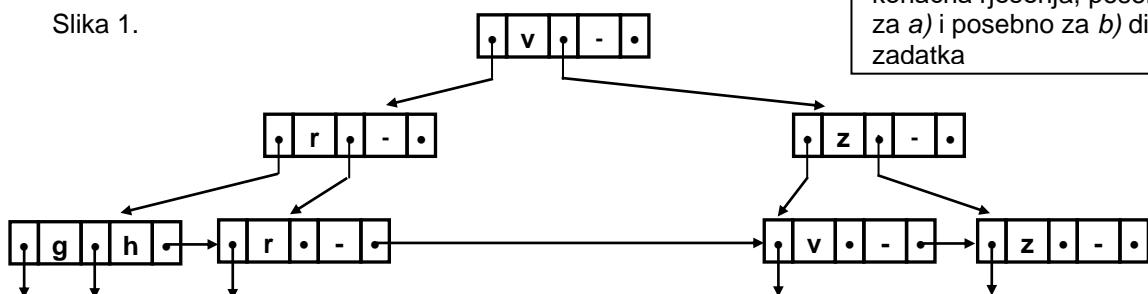
Proceduru napisati tako da se njenim izvršavanjem stipendija odobi svim studentima kojima treba biti odobrena ili niti jednom. Ako se pri obavljanju UPDATE naredbe u relaciji *stipendija* dogodi bilo koja vrsta pogreške, procedura u pozivajući program vraća pogrešku s tekstom '**Pogreška pri izmjeni.**' U slučaju pojave bilo koje druge pogreške na bilo kojem drugom mjestu u proceduri, procedura u pozivajući program mora proslijediti originalnu pogrešku. Nije dopuštena upotreba pomoćnih relacija niti SAVEPOINT mehanizma.

*Primjer:* za sadržaj relacije kao na slici, pozivom procedure s ulaznim parametrom 3500.00 dodijelit će se stipendija studentima s rangom 1 i 2, a nedodijeljeni iznos sredstava iznosiće 500.00.

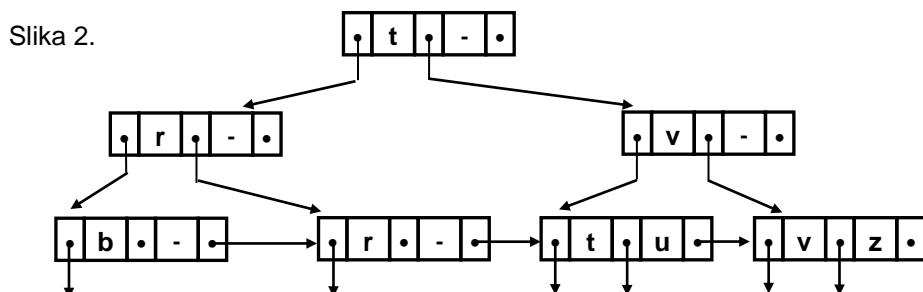
Ponovnim pozivom procedure s ulaznim parametrom 1200.00, stipendija će biti dodijeljena studentu s rangom 3, a nedodijeljeni iznos sredstava će iznositi 200.00. (4 boda)

2. Nacrtati B<sup>+</sup>-stablo:

- a) nakon unosa zapisa s ključem **a** u B<sup>+</sup>-stablu na slici 1.



- b) nakon brisanja zapisa s ključem **r** iz B<sup>+</sup>-stabla na slici 2.



(3 boda)

3. Za relaciju  $r(A, B)$  kreiran je samo indeks za atribut A. Za svaku od sljedećih operacija objasniti hoće li se za njeno izvršavanje koristiti indeks, pod kojim uvjetima i zašto.
- SELECT \* FROM r WHERE A = 100 AND B = 200**
  - SELECT \* FROM r WHERE A = 100 OR B = 200**
- (3 boda)

4. Slikom prikazati postupak sortiranja relacije (datoteke) prikazane na slici 3. metodom vanjskog sortiranja s uparivanjem (*external sort-merge*). Relaciju sortirati prema prvom atributu. Broj raspoloživih blokova glavne memorije  $M = 3$ . Broj n-torki (zapisa) u bloku je 3. Označiti faze i objasniti što se obavlja u kojoj fazi, te na koji se način u kojoj fazi koriste raspoloživi blokovi glavne memorije (odnosno međuspremnići). (4 boda)

2	a
7	b
9	c
4	d
6	e
11	f
1	g
10	h
13	i
8	j
12	k
5	l
3	m

Slika 3.

5. Zadane su relacije  $r(A, B, C)$  i  $s(D, E, F)$ .

Primarni ključevi su potcrtni. Nema indeksa. Na raspolažanju je 102 blokova primarne memorije. Svi međurezultati se materijaliziraju (zapisuju u sekundarnu memoriju). Kartezijev produkt se obavlja metodom *block nested-loop join*. Operacije se izvršavaju redoslijedom kako je navedeno u izrazu relacijske algebre, što znači da ne treba provoditi heurističku optimizaciju. Koristeći priložene podatke, procijeniti ukupni broj **U/I operacija** te broj blokova i broj n-torki u konačnom rezultatu. U rješenju prikazati postupak (ne samo konačni rezultat).

$$\sigma_{C=100 \wedge B>200}(r) \times \sigma_{E \text{ LIKE } 'R\%'}(s)$$

$V(B, r) = 200$
$V(E, s) = 20$
$V(F, s) = 5000$

$N(r) = 200000$
$N(s) = 10000$

$B(r) = 150000$
$B(s) = 5000$

$\min(B, r) = 100$
$\max(B, r) = 500$
$\min(C, r) = 0$
$\max(C, r) = 400000$
$\min(D, s) = 0$
$\max(D, s) = 50000$

veličina n-torke(r) = 0.5 blokova
veličina n-torke(s) = 0.2 blokova

(5 bodova)

6. a) Napisati definiciju striktnog parcijalnog poretku.  
 b) Napisati definiciju modela transakcije.

(4 boda)

## Rješenja odabralih zadataka:

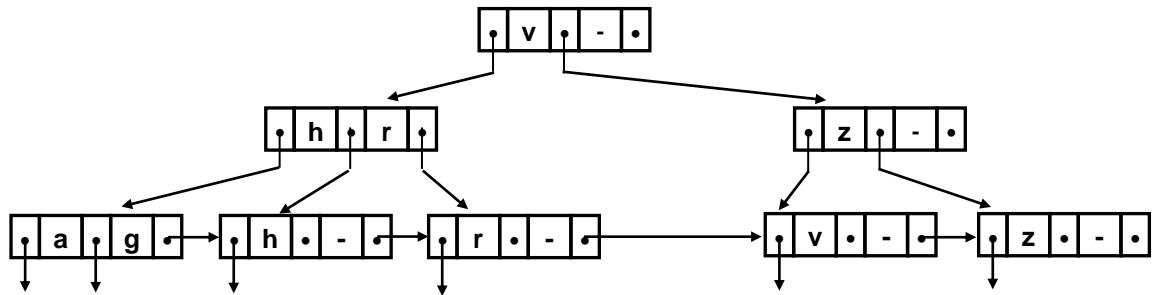
1.

```
CREATE FUNCTION odobriStip (p_iznos DECIMAL (10, 2))
RETURNS DECIMAL (10, 2) AS nedodijeljeno;
DEFINE sqle, isame INTEGER;
DEFINE errdata CHAR(80);
DEFINE p_rang LIKE stipendija.rang;
DEFINE p_predvIznos LIKE stipendija.predvIznos;

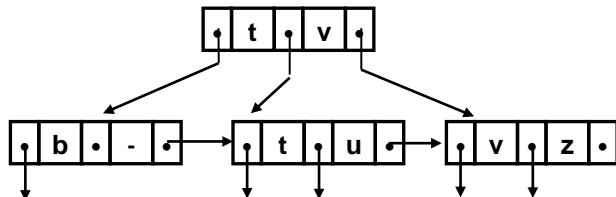
ON EXCEPTION SET sqle, isame, errdata
ROLLBACK WORK;
RAISE EXCEPTION sqle, isame, errdata;
END EXCEPTION

BEGIN WORK;
FOREACH SELECT rang, predvIznos
    INTO p_rang, p_predvIznos
    FROM stipendija
    WHERE odobrenaStip = 'N'
    ORDER BY rang
    IF (p_predvIznos > p_iznos) THEN
        EXIT FOREACH;
    END IF
    LET p_iznos = p_iznos - p_predvIznos;
    BEGIN
        ON EXCEPTION
            RAISE EXCEPTION -746, 0, 'Pogreška pri izmjeni.';
        END EXCEPTION
        UPDATE stipendija
            SET odobrenaStip = 'D'
            WHERE rang = p_rang;
    END
END FOREACH
COMMIT WORK;
RETURN p_iznos;
END FUNCTION;
```

2. a)



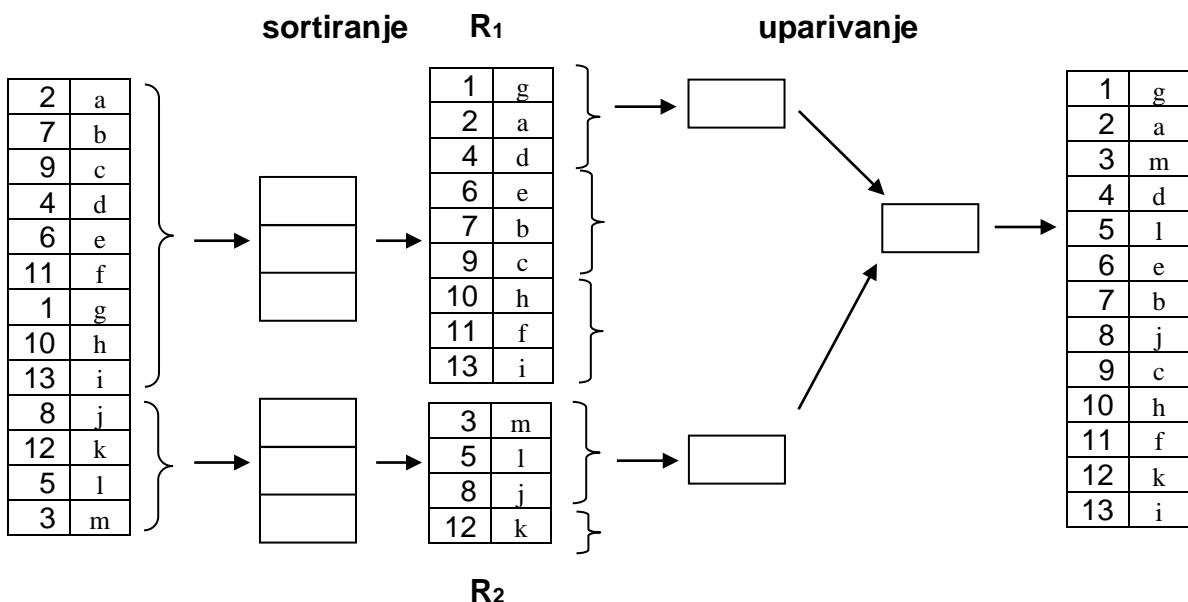
b)



3. a) *komentar odgovora:* indeks će se koristiti pod određenim uvjetima. Važno je navesti kako broj n-torki koje zadovoljavaju uvjet  $A=100$  ne smije biti prevelik (relativno u odnosu na ukupan broj n-torki). Stupanj grupiranja (*cluster factor*) može imati utjecaj, ali odgovor u kojem bi bio naveden samo stupanj grupiranja je neispravan jer se isključivo na temelju faktora grupiranja ne može ocijeniti isplativost korištenja indeksa. Ako se indeks koristi, operacijom *index-scan* čitaju se n-torce koji zadovoljavaju uvjet  $A=100$ , a nad onima koje taj uvjet zadovoljavaju primjenjuje se uvjet  $B=200$  (čime se ne povećava broj UI operacija).

b) *komentar odgovora:* indeks se sigurno neće koristiti. Uzaludno je korištenjem indeksa dohvatiti n-torce koje zadovoljavaju uvjet  $A=100$ , jer će se ionako za dohvat n-torki koje zadovoljavaju uvjet  $B=200$  morati pročitati svi blokovi relacije. Korištenje indeksa u ovom slučaju predstavlja bi samo dodatni trošak na ionako neizbjegjan trošak čitanja svih blokova relacije.

4.



U fazi sortiranja odjednom se čita M blokova (9 n-torki) i sortiraju, zapisuju u segment (*runfile*)  $R_i$ . Postupak se ponavlja dok se ne iscrpi cijela ulazna datoteka. Rezultat je  $N=2$  segmenata. Budući da je  $N < M$ , uparivanje se može provesti u samo jednom koraku. Jedan blok međuspremnika koristi se za čitanje  $R_1$ , jedan za  $R_2$ , a treći blok međuspremnika se koristi za pisanje rezultata.

5. Broj n-torki

- $f(C=100, r) = N(\sigma_{C=100}(r)) / N(r) = N(r) / 10 / N(r) = 0.1$
- $f(B \leq 200, r) = N(\sigma_{B \leq 200}(r)) / N(r) = N(r) \cdot (200 - 100) / (500 - 100) / N(r) = 0.25$
- $f(B > 200, r) = 1 - f(B \leq 200, r) = 0.75$
- $t_1 = \sigma_{C=100 \wedge B>200}(r)$
- $N(t_1) = N(r) \cdot f(C=100, r) \cdot f(B > 200, r) = 200\ 000 \cdot 0.75 \cdot 0.1 = 15\ 000$
- $f(E \text{ LIKE } 'R%', s) = N(\sigma_{E \text{ LIKE } 'R%'}(s)) / N(s) = N(s) / 5 / N(s) = 0.2$
- $t_2 = \sigma_{E \text{ LIKE } 'R%'}(s)$
- $N(t_2) = N(s) \cdot f(E \text{ LIKE } 'R%', r) = 2\ 000$
- $t_3 = t_1 \times t_2$
- $N(t_3) = N(t_1) \cdot N(t_2) = 30\ 000\ 000$

Broj U/I operacija

- veličina konačnog rezultata  $N(t_3) \cdot (0.5 + 0.2) = 30\ 000\ 000 \cdot 0.7 = 21\ 000\ 000$
- čitanje relacije r pri obavljanju  $\sigma_{C=100 \wedge B>200}(r) \Rightarrow B(r) = \mathbf{150\ 000}$
- veličina međurezultata  $B(t_1) = N(t_1) \cdot 0.5 = 7\ 500$
- zapisivanje međurezultata  $t_1 \Rightarrow \mathbf{7\ 500}$
- čitanje relacije s pri obavljanju  $\sigma_{E \text{ LIKE } 'R%'}(s) \Rightarrow B(s) = \mathbf{5\ 000}$
- veličina međurezultata  $B(t_2) = N(t_2) \cdot 0.2 = 400$
- zapisivanje međurezultata  $t_2 \Rightarrow \mathbf{400}$
- obavljanje  $t_1 \times t_2$  : niti jedna relacija ne stane cijela u međuspremnike
- $\Rightarrow B(t_2)/(M-2) \cdot B(t_1) + B(t_2) = 4 \cdot 7\ 500 + 400 = \mathbf{30\ 400}$
- ukupno (bez zapisivanja konačnog rezultata)  $\Rightarrow 150\ 000 + 7\ 500 + 5\ 000 + 400 + 30\ 400 = 193\ 300$

Sustavi baza podataka - Međuispit  
odabrani zadaci  
27. travnja 2015.

- odgovore na pitanja 1 - 6 napisati na vlastitim listovima papira. Netočni odgovori na ova pitanja ne donose negativne bodove.

- ~~1. U bazi podataka su kreirane relacije *racun* i *stavka*. Relacije su prazne.~~

~~Napisati pohranjenu proceduru **generiraj** kojom će se u relacije upisati podaci za testiranje. Procedura u relaciju *racun* treba pokušati upisati ukupno 100 n-torki s vrijednostima atributa *brRac* od 1 do 100, a za svaku takvu n-torku u relaciji *racun* upisati točno 10 pripadnih n-torki u relaciju *stavka*, s vrijednostima za atribut *rbrStavka* od 1 do 10.~~

```
CREATE TABLE racun (
    brRac      INTEGER,
    PRIMARY KEY (brRac) );

CREATE TABLE stavka (
    brRac      INTEGER,
    rbrStavka INTEGER,
    PRIMARY KEY
        (brRac, rbrStavka),
    FOREIGN KEY (brRac)
        REFERENCES racun (brRac);
```

~~Proceduru napisati tako da bude osigurano sljedeće:~~

~~ako je u relaciju *racun* upisana n-torka za neki račun, tada u relaciju *stavka* moraju biti upisane sve n-torce koje pripadaju tom računu (to ne znači da u relacije mora biti upisano ili svih 100+1000 n-torki ili niti jedna - vidjeti primjer).~~

~~Ako se pri obavljanju operacije unosa u relaciju *stavka* dogodi bilo koja vrsta pogreške, procedura u pozivajući program vraća pogrešku s tekstom '**Pogreška zbog stavke**'. U slučaju pojave bilo koje druge pogreške na bilo kojem drugom mjestu u proceduri, procedura u pozivajući program mora proslijediti originalnu pogrešku. Nije dopuštena upotreba pomoćnih relacija, okidača (*trigger*) niti *SAVEPOINT* mehanizma.~~

*Primjer:* prikazana su tri od ukupno stotinu i jednog mogućeg ishoda poziva procedure

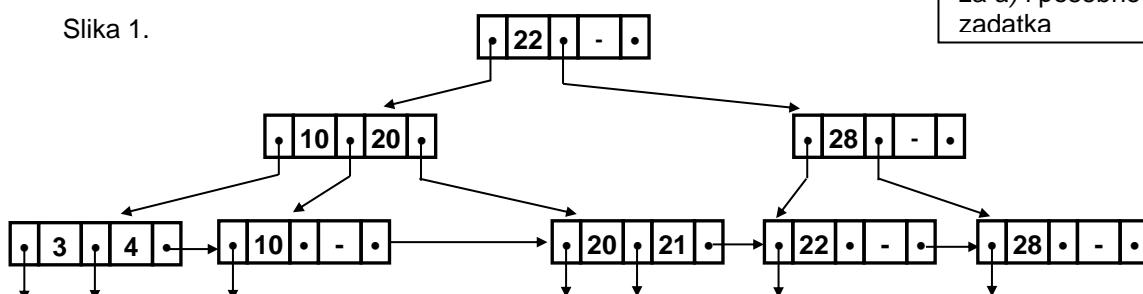
- obje relacije ostale su prazne
- u relaciji *racun* je 75 n-torki (brojevi računa 1-75), a u relaciji *stavka* 750 n-torki (za svaki broj računa od 1-75 točno po 10 stavki s rednim brojevima od 1-10). **Uočite:** nije dopušteno da se nakon završetka procedure u relaciji *racun* nalazi 75 n-torki, a u relaciji *stavka* 749 n-torki.
- u relaciji *racun* je 100 n-torki, a u relaciji *stavka* 1000 n-torki

(4 boda)

- ~~2. Nacrtati B<sup>+</sup>-stablo:~~

- a) nakon unosa zapisa s ključem **9** u B<sup>+</sup>-stablo na slici 1.

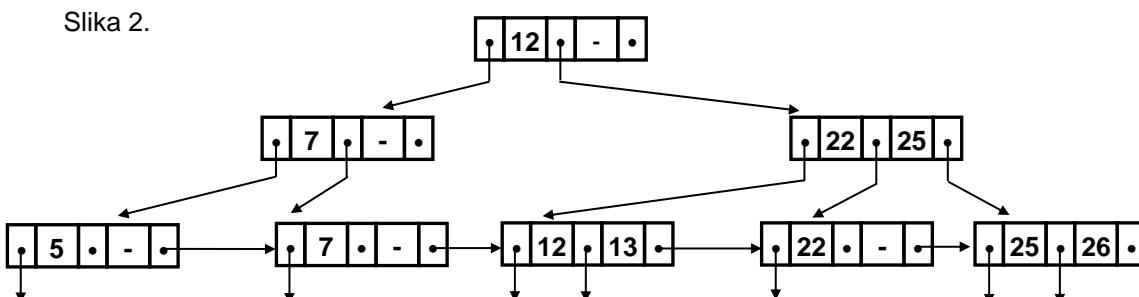
Slika 1.



Dovoljno je nacrtati samo konačna rješenja, posebno za a) i posebno za b) dio zadatka

- b) nakon brisanja zapisa s ključem **7** iz B<sup>+</sup>-stabla na slici 2.

Slika 2.



(4 boda)

3. a) Objasniti što je *cluster index* i što je stupanj grupiranja (*cluster factor*)  
b) Objasniti kada i zašto optimizator koristi informaciju o stupnju grupiranja (4 boda)
4. a) Definirati svojstva transakcije *atomarnost* i *izdržljivost*  
b) Navesti i objasniti tri vrste pogrešaka u sustavu za upravljanje bazama podataka (SUBP). Pojedinačno, za svaku vrstu pogreške navesti koje mehanizme SUBP koristi da bi osigurao svojstvo *atomarnosti*, a koje mehanizme koristi da bi osigurao svojstvo *izdržljivosti* (4 boda)
5. Zadane su relacije  $r(A, B, C, D)$  i  $s(E, F)$ .

Primarni ključevi su potcrtani. Nema indeksa. Na raspolaganju je 1002 blokova primarne memorije. Svi međurezultati se materijaliziraju (zapisuju u sekundarnu memoriju). Za operaciju spajanja uz uvjet koristi se spajanje ugnježđenim petljama (*nested-loop join*). Operacije se izvršavaju redoslijedom kako je navedeno u izrazu relacijske algebre, što znači da ne treba provoditi heurističku optimizaciju. Izvršava se sljedeća operacija relacijske algebre:

$$\sigma_{B \leq 1000 \vee C = 20}(r) \triangleright \triangleleft s \\ D = F$$

$V(B, r) = 200$	$N(r) = 5000$	$\min(B, r) = 500$
$V(C, r) = 5$	$N(s) = 1500$	$\max(B, r) = 2500$
$V(D, r) = 1000$	$B(r) = 2000$	$\min(C, r) = 0$
$V(F, s) = 100$	$B(s) = 1200$	$\max(C, r) = 100$
veličina n-torke( $r$ ) = 0.2 blokova		$\min(D, r) = 200$
veličina n-torke( $s$ ) = 0.5 blokova		$\max(D, r) = 400$
		$\min(F, s) = 50$
		$\max(F, s) = 250$

Koristeći priložene podatke iz rječnika podataka, za zadatu operaciju relacijske algebre:

- a) prvo procijeniti broj n-torki u međurezultatima i konačnom rezultatu
- b) zatim procijeniti ukupni broj **U/I operacija** tijekom izvršavanja operacije relacijske algebre i broj blokova u konačnom rezultatu.

U rješenju prikazati postupak (ne samo konačni rezultat). (4 boda)

6. Transakcija T je opisana sljedećim pseudokôdom:

```
begin work;
    read(x, p1);
    read(y, p2);
    read(z, p3);
    p4 ← p1 + p2 + p3;
    write(y, p4);
    write(z, 3.14); /* zapisuje se konstantna vrijednost */
commit work;
```

- a) nacrtati graf transakcije. U graf ucrtati isključivo one lûkove koji proizlaze iz semantike transakcije ili su nužni da bi se zadovoljila pravila konstrukcije grafa transakcije
- b) svaki lûk u grafu označiti proizvoljno odabranim rednim brojem, te referencirajući se na te brojeve, pojedinačno za svaki lûk navesti zašto je ucrtan u graf transakcije

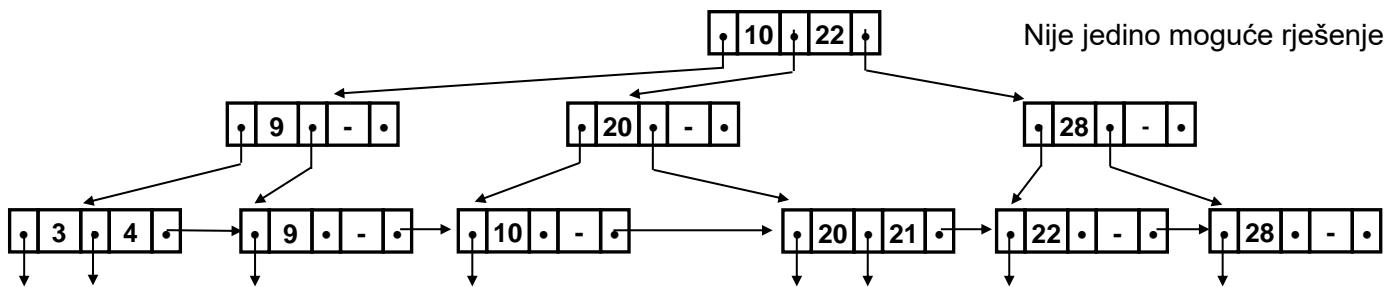
(3 boda)

## Rješenja:

1.

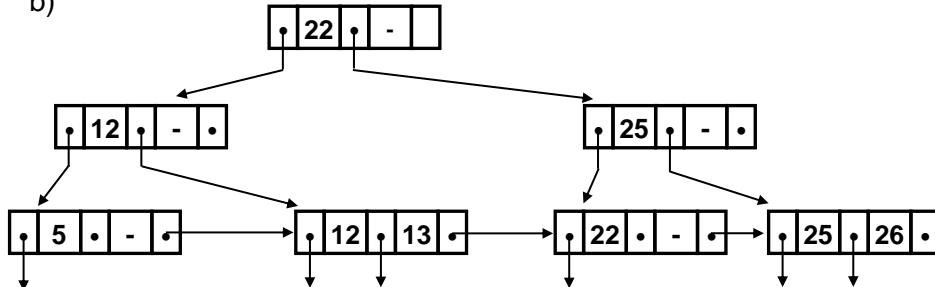
```
CREATE PROCEDURE generiraj ()  
  DEFINE pBrRac, pRbrStavka INTEGER;  
  DEFINE sqle, isame INTEGER;  
  DEFINE errdata CHAR(80);  
  FOR pBrRac = 1 TO 100  
    ON EXCEPTION SET sqle, isame, errdata  
    ROLLBACK WORK;  
    RAISE EXCEPTION sqle, isame, errdata;  
  END EXCEPTION  
  BEGIN WORK;  
  INSERT INTO racun VALUES (pBrRac);  
  FOR pRbrStavka = 1 TO 10  
    ON EXCEPTION  
      RAISE EXCEPTION -746, 0, 'Pogreška pri unosu stavke.';  
    END EXCEPTION  
    INSERT INTO stavka VALUES (pBrRac, pRbrStavka);  
  END FOR  
  COMMIT WORK;  
END FOR  
END PROCEDURE;
```

2. a)



Nije jedino moguće rješenje

b)



3. Komentar umjesto odgovora

- Indeks kod kojeg su podaci fizički poredani (manje-više) u skladu s ključevima iz listova B-stabla.  
Stupanj grupiranja je mjera koja govori o tome koliko su podaci poredani prema vrijednostima nekog (kompozitnog) ključa
- Selekcija i spajanje: hoću li koristiti indeks?

4. Komentar umjesto odgovora

a) vidjeti predavanja

b) **pogreška transakcije:**

atomarnost - dnevnik, poništavanje transakcije pomoću dnevnika  
izdržljivost - bespredmetno u slučaju pogreške transakcije

**pogreška sustava:**

atomarnost - dnevnik, brza obnova  
izdržljivost - dnevnik, brza obnova

**pogreška medija:**

atomarnost - arhivska kopija (ne i dnevnik, jer obnovom iz arhivske kopije dobivamo konzistentno stanje, stanje u kojem rezultati svih transakcija zadovoljavaju svojstvo atomarnosti)  
izdržljivost - arhivska kopija, dnevnik, obnova

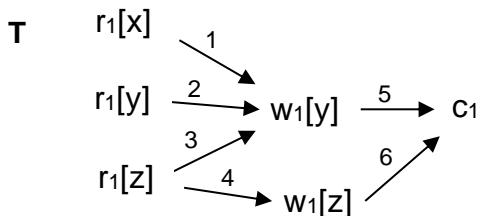
5. a) Broj n-torki

- $f(B \leq 1000, r) = N(\sigma_{B \leq 1000}(r)) / N(r) =$   
 $= N(r) \cdot (v - \min(B, r)) / (\max(B, r) - \min(B, r)) / N(r) =$   
 $= N(r) \cdot (1000 - 500) / (2500 - 500) / N(r) = 0.25$
- $f(C=20, r) = N(\sigma_{C=20}(r)) / N(r) = N(r) / V(C, r) / N(r) = 1 / 5 = 0.2$
- $f(B \leq 1000 \vee C=20, r) = 1 - (1 - f(B \leq 1000, r)) \cdot (1 - f(C=20, r)) =$   
 $= 1 - 0.75 \cdot 0.8 = 1 - 0.6 = 0.4$
- $t_1 = \sigma_{B \leq 1000 \vee C=20}(r)$
- $N(t_1) = N(r) \cdot f(B \leq 1000 \vee C=20, r) = 5000 \cdot 0.4 = 2000$
- $t_2 = t_1 \triangleright \triangleleft_{D=F} s$
- $N(t_2) = N(t_1) \cdot N(s) / \max(V(D, r), V(F, s)) = 2000 \cdot 1500 / 1000 = 3000$

b) Broj U/I operacija

- veličina konačnog rezultata  $B(t_2) = N(t_2) \cdot (0.2 + 0.5) = 3000 \cdot 0.7 = 2100$
- čitanje relacije  $r$  pri obavljanju  $\sigma_{B \leq 1000 \vee C=20}(r) \Rightarrow B(r) = \mathbf{2000}$
- veličina međurezultata  $B(t_1) = N(t_1) \cdot 0.2 = 2000 \cdot 0.2 = 400$
- zapisivanje međurezultata  $t_1 \Rightarrow \mathbf{400}$
- obavljanje  $t_1 \triangleright \triangleleft_{D=F} s$   
 $t_1$  cijela stane u međuspremnik  $\Rightarrow B(t_1) + B(s) = 400 + 1200 = \mathbf{1600}$
- ukupno (bez zapisivanja konačnog rezultata)  $\Rightarrow 2000 + 400 + 1600 = 4000$

6. a)



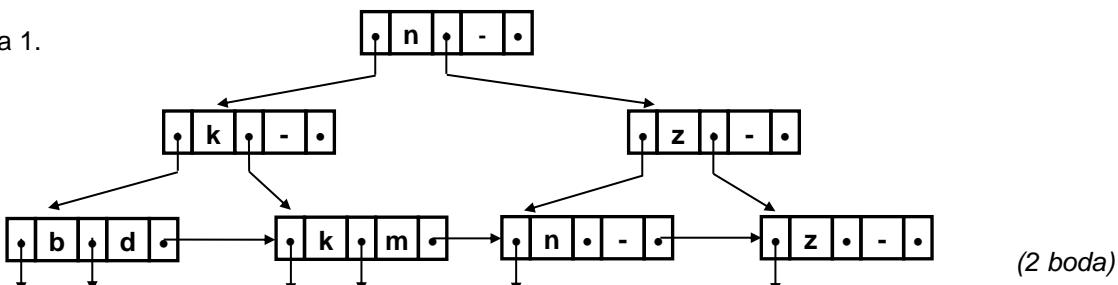
- b)
- 1 -  $y$  se izračunava na temelju  $x$
  - 2 -  $y$  se izračunava na temelju  $y$ ;  $r$  i  $w$  operacije nad istim elementom moraju biti poredane
  - 3 -  $y$  se izračunava na temelju  $z$
  - 4 -  $r$  i  $w$  operacije nad istim elementom moraju biti poredane
  - 5 - sve operacije moraju biti poredane u odnosu na operaciju commit
  - 6 - sve operacije moraju biti poredane u odnosu na operaciju commit

Sustavi baza podataka - Međuispit - Odabrani zadaci  
3. svibnja 2016.

- odgovore na pitanja 1 - 8 napisati na vlastitim listovima papira. Netočni odgovori na ova pitanja ne donose negativne bodove.

- Nacrtati B<sup>+</sup>-stablo nakon brisanja zapisa s ključem **n** iz B<sup>+</sup>-stabla na slici 1. Dovoljno je nacrtati samo konačno rješenje.

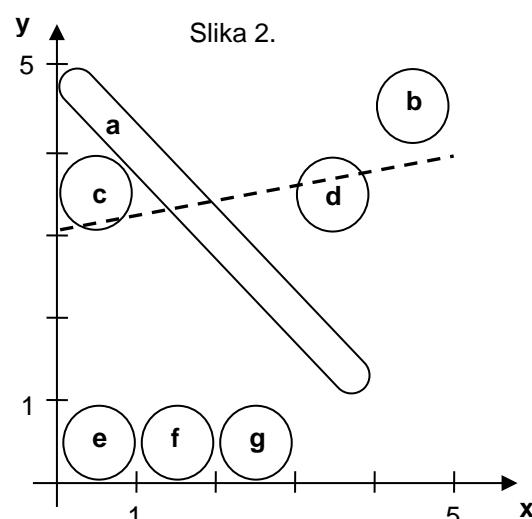
Slika 1.



- a) Nacrtati R-stablo za prostorne podatke (objekte, odnosno geometrijske likove **a** - **g**) prikazane na slici 2.

Najveći dopušteni broj zapisu (ključ+kazaljka) u listu i internom čvoru R-stabla je 3, a najmanji broj zapisu je 2. Rješenje a) dijela zadatka treba sadržavati dvije slike. Na prvoj slici nacrtati sve minimalne granične okvire (MBR). Na drugoj slici nacrtati R-stablo.

- b) Objasniti postupak kojim se pomoću R-stabla iz a) dijela zadatka pronalaze svi objekti koji su u presjeku s linijom koja se proteže od točke s koordinatama ( $x=0, y=3$ ) do točke s koordinatama ( $x=5, y=4$ ) (3 boda)



- U bazi podataka kreirane su relacije *racun* i *stavka*.

~~Napisati pohranjenu proceduru **brisanje** kojom se jedan po jedan brišu racun i njegove stavke za prvih 10 računa, redom prema broju računa. U trenutku kada se jedan račun i njegove stavke brišu, rezultat tog brisanja mora biti trajno zabilježen u bazi podataka. To znači da rezultat jednog poziva procedure može biti npr. da prvih sedam računa i njihove stavke budu uspješno obrisani, a sljedeća tri računa i njihove stavke ostanu u bazi podataka. Istovremeno, ne smije se dogoditi da za neki račun budu obrisane stavke, a ne bude obrisan zapis o računu, ili obratno.~~

~~Ako se pri obavljanju operacija brisanja iz računa ili stavke dogodi pogreška, procedura u pozivajući program vraća pogrešku s tekstom 'Pogreška brisanja'. U slučaju pojave bilo koje druge pogreške na bilo kojem drugom mjestu u proceduri, u pozivajući program se mora proslijediti originalna pogreška. Nije dopuštena upotreba pomoćnih relacija, okidača (trigger) niti SAVEPOINT mehanizma.~~ (3 boda)

```

CREATE TABLE racun (
    brRac    INTEGER,
    PRIMARY KEY (brRac);

CREATE TABLE stavka (
    brRac    INTEGER,
    rbrStavka INTEGER,
    PRIMARY KEY
        (brRac, rbrStavka)
);
  
```

- Objasnite na koji način SUBP izrađuje konzistentnu arhivu razine nula (full backup) bez prethodnog zaustavljanja sustava (dakle non-quiescent ili online ili hot backup). (3 boda)

5. U tijeku je obnova nakon pogreške sustava. SUBP za vraćanje baze podataka u konzistentno stanje koristi zapise iz logičkog dnevnika. Za jednu grupu transakcija potrebni su mu samo zapisi dnevnika tih transakcija upisani nakon kontrolne točke, a za drugu grupu transakcija potrebni su mu zapisi dnevnika tih transakcija upisani i prije i nakon kontrolne točke. Objasnite po čemu se razlikuju transakcije iz prve i druge grupe. (3 boda)

6. Zadane su relacije  $r(A, B, C)$  i  $s(D, E)$ .

Primarni ključevi su potcrtni. Nema indeksa. Na raspolaganju je 1000 blokova primarne memorije. Svi međurezultati se materijaliziraju (zapisuju u sekundarnu memoriju). Za operaciju spajanja uz uvjet koristi se spajanje ugnježđenim petljama (*nested-loop join*). Operacije se izvršavaju redoslijedom kako je navedeno u izrazu relacijske algebre, što znači da ne treba provoditi heurističku optimizaciju. Izvršava se sljedeća operacija relacijske algebre:

$$\sigma_{B \neq 15} (r \triangleright\triangleleft s) \\ C = E$$

$V(B, r) = 10$	$N(r) = 5000$	$\min(B, r) = 500$
$V(C, r) = 1000$	$N(s) = 4000$	$\max(B, r) = 600$
$V(E, s) = 2000$	$B(r) = 3000$	$\min(C, r) = 0$
	$B(s) = 800$	$\max(C, r) = 5000$
		$\min(E, s) = 1000$
		$\max(E, s) = 5000$
$veličina n-torke(r) = 0.3$ blokova		
$veličina n-torke(s) = 0.2$ blokova		

Koristeći priložene podatke iz rječnika podataka, za zadalu operaciju relacijske algebre:

- Prvo procijeniti broj n-torki u međurezultatima i konačnom rezultatu
- Zatim procijeniti ukupni broj **U/I operacija** tijekom izvršavanja operacije relacijske algebre i broj blokova u konačnom rezultatu.

U rješenju prikazati postupak (ne samo konačni rezultat). (4 boda)

7. Obavlja se operacija prirodnog spajanja  $r(R) \triangleright\triangleleft s(S)$ .

- $R = \{A, C\}$   $S = \{D, C\}$
- $N(r) = 1$   $N(s) = 10\ 000$
- $B(r) = 1$   $B(s) = 1\ 000$
- za relaciju  $s(S)$  postoji indeks za atribut  $C$
- za obavljanje operacije na raspolaganju je 50 blokova primarne memorije

Objasnite koji od navedenih i koji eventualno dodatni statistički podaci mogu biti korisni kada optimizator odlučuje treba li primijeniti fizički operator *indexed nested-loop join* ili *block nested-loop join*. Napomena: u rješenju se očekuje samo traženo objašnjenje, a ne procjena konkretnog broja U/I operacija koji bi se izvršio u jednom, odnosno drugom slučaju. (2 boda)

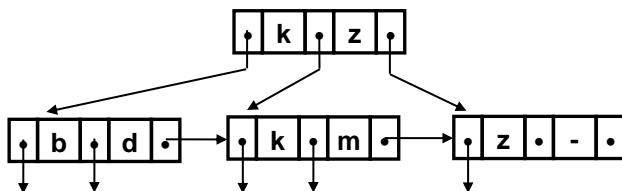
8. Transakcija  $T_1$  izvršava operacije nad elementima baze podataka  $x, y, z$ . Zadatak transakcije jest zamijeniti vrijednosti elemenata  $x$  i  $y$  (početnu vrijednost iz  $x$  upisati u  $y$ , početnu vrijednost iz  $y$  upisati u  $x$ ), te sumu početnih vrijednosti  $x$  i  $y$  upisati u element  $z$ .

- Nacrtati graf transakcije  $T_1$ . U graf ucrtati isključivo one lúkove koji proizlaze iz semantike transakcije ili su nužni da bi se zadovoljila pravila konstrukcije grafa transakcije.
- Svaki lúk u grafu označiti proizvoljno odabranim rednim brojem, te referencirajući se na te brojeve, za svaki lúk navesti zašto je ucrtan u graf transakcije.

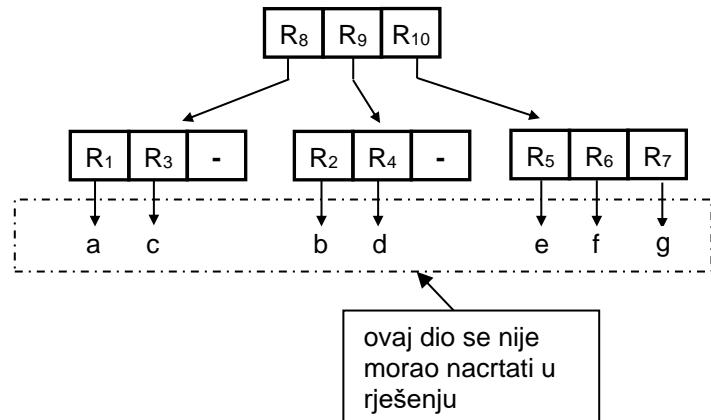
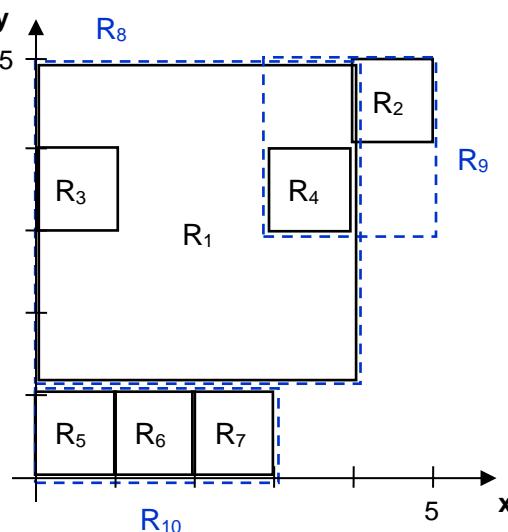
(3 boda)

## Rješenja odabralih zadataka:

1.



2.



- b) Odrediti minimalni granični okvir (MBR) za zadanu liniju (npr.  $MBR_x$ ). U korijenu stabla pronaći sve MBR-ove koji se sijeku s  $MBR_x$ . To su  $R_8$  i  $R_9$ . Zatim istodobno slijediti pokazivač uz  $R_8$  i  $R_9$ . Uz  $R_8$  utvrditi koji je od MBR-ova  $R_1$  i  $R_3$  u presjeku s  $MBR_x$ . To su  $R_1$  i  $R_3$ . Utvrditi jesu li objekti  $a$  ili  $c$  u presjeku sa zadanom linijom (ako neki od njih jest, dodati ga u rješenje). Uz  $R_9$  utvrditi koji je od MBR-ova  $R_2$  i  $R_4$  u presjeku s  $MBR_x$ . To je  $R_4$ . Utvrditi je li objekt  $d$  u presjeku sa zadanom linijom (ako jest, dodati ga u rješenje).

3.

```

CREATE PROCEDURE brisanje ()
  DEFINE pBrRac INTEGER;
  DEFINE sqle, isame INTEGER;
  DEFINE errdata CHAR(80);
  DEFINE brojac INTEGER;
  LET brojac = 0;
  FOREACH WITH HOLD SELECT brRac INTO pBrRac FROM racun ORDER BY brRac
    LET brojac = brojac + 1;
    IF brojac > 10 THEN
      EXIT FOREACH;
    END IF;
    BEGIN WORK;
    BEGIN
      ON EXCEPTION SET sqle, isame, errdata
        ROLLBACK WORK;
        RAISE EXCEPTION -746, 0, "Pogreška brisanja";
    END EXCEPTION;
    DELETE FROM stavka WHERE brRac = pBrRac;
    DELETE FROM racun WHERE brRac = pBrRac;
    END
    COMMIT WORK;
  END FOREACH;
END PROCEDURE;

```

4. Pogledati predavanja: stranica Arhiviranje baze podataka bez zaustavljanja sustava.

5. Transakcije koje su započele nakon kontrolne točke i završile na bilo koji način (ili nisu završile) prije kvara, trivijalno pripadaju grupi transakcija za koje su potrebni samo zapisi dnevnika nastali nakon kontrolne točke.

Preostaje razmotriti samo transakcije koje su bile aktivne za vrijeme posljednje kontrolne točke:

- za transakcije koje su bile aktivne za vrijeme kontrolne točke i koje treba ponovo obaviti potrebni su samo zapisи nakon kontrolne točke jer sve promjene koje su te transakcije obavile prije kontrolne točke su u kontrolnoj točki sigurno zapisane u postojanu memoriju
- za transakcije koje su bile aktivne za vrijeme kontrolne točke i koje treba poništiti potrebni su zapisi i prije i nakon kontrolne točke jer je potrebno poništiti sve promjene koje je transakcija načinila u postojanoj memoriji, i one koje su (možda) zapisane u postojanu memoriju poslije kontrolne točke i one koje su (sigurno) zapisane prije kontrolne točke

6. a) Broj n-torki

- $t_1 = r \triangleright \triangleleft_{C=E} s$
- $N(t_1) = N(r) \cdot N(s) / \max(V(C, r), V(E, s)) = 5\ 000 \cdot 4\ 000 / \max(1\ 000, 2\ 000) = 10\ 000$
- $V(B, r) = 10 \Rightarrow V(B, t_1) = 10 \Rightarrow f(B=15, t_1) = 1 / 10 = 0.1$
- $f(B \neq 15, t_1) = 1 - f(B=15, t_1) = 0.9$
- $t_2 = \sigma_{B \neq 15}(t_1)$
- $N(t_2) = N(t_1) \cdot f(B \neq 15, t_1) = 10\ 000 \cdot 0.9 = 9\ 000$

U tekstu zadatka pisalo je  $\min(B, r) = 500$ , što su rijetki studenti prepoznali kao "zamku" i zbog toga izračunali  $N(t_2) = 10\ 000$ .  
Nastojimo u zadacima ne postavljati "zamke" i ovdje se radilo o običnoj pogrešci u tekstu: trebalo je pisati  $\min(B, r) = 5$ .

- b) Broj U/I operacija

- veličina n-torke u konačnom rezultatu  $0.3 + 0.2 = 0.5$
- veličina konačnog rezultata  $B(t_2) = N(t_2) \cdot 0.5 = 9\ 000 \cdot 0.5 = 4\ 500$

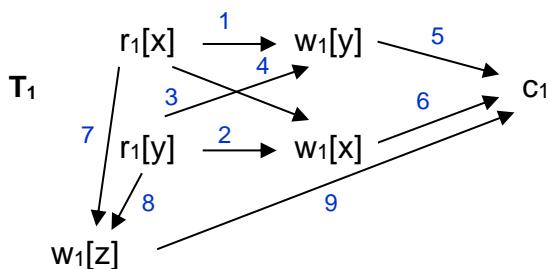
Priznaju se naravno oba rješenja. Prikazano je rješenje za  $\min(B, r) = 5$ .

- obavljanje  $r \triangleright \triangleleft_{C=E} s$   
s cijela stane u međuspremnik  $\Rightarrow B(r) + B(s) = 3\ 000 + 800 = 3\ 800$
- zapisivanje međurezultata  $t_1 \Rightarrow N(t_1) \cdot (0.3 + 0.2) = 10\ 000 \cdot 0.5 = 5\ 000$
- čitanje međurezultata  $t_1$  pri obavljanju  $\sigma_{B \neq 15}(t_1) \Rightarrow 5\ 000$
- ukupno (bez zapisivanja konačnog rezultata)  $\Rightarrow 3\ 800 + 5\ 000 + 5\ 000 = 13\ 800$

7. Iz  $N(r) = 1$  proizlazi da za n-torku iz r treba pronaći samo n-torce iz s za koje vrijedi  $t(C) = x$ . Indeks će se koristiti samo ako n-torce koje zadovoljavaju uvjet spajanja nisu smještene u prevelikom broju blokova, u suprotnom je bolje čitati cijelu relaciju s (tj. koristiti *block nested-loop join*).

Za procjenu koliko ima n-torki koje zadovoljavaju uvjet spajanja, korisno je poznavati  $N(s)$  i  $V(C, s)$  ili još bolje - histograme. Ako uvjet spajanja zadovoljava relativno mali broj n-torki, tada treba procijeniti u koliko su blokova smještene, pomoću stupnja grupiranja indeksa (jer i relativno mali broj n-torki može biti raspršen po velikom broju blokova). Dubina B-stabla (u ovom primjeru) je irelevantna jer se kroz B-stablo (ako se koristi) prolazi samo jednom.

8. a)



b)

- novi y se izračunava na temelju x
- novi x se izračunava na temelju y
- 3, 4 - r i w operacije nad istim elementom moraju biti poredane
- 7, 8 - novi z se izračunava na temelju x i y
- 5, 6, 9 - sve operacije moraju biti poredane u odnosu na operaciju commit

1. a) Nacrtati B<sup>+</sup>-stablo **reda 4** tako da **ukupni broj čvorova** stabla bude **minimalan**. Stablo treba sadržavati sljedeće vrijednosti ključeva: 1, 2, 3, 10, 11, 12, 13, 14, 15, 28, 29, 30 (ukupno 12 vrijednosti ključeva).
- b) Nacrtati stablo koje nastane upisivanjem zapisa s ključem **9** u stablo koje je nacrtano u a) dijelu zadatka.
- c) Nacrtati stablo koje nastane brisanjem zapisa s ključevima **10** i zatim **11** (dakle, brisanjem **dva** zapisa) iz stabla koje je nacrtano u b) dijelu zadatka. (4 boda)
2. U bazi podataka kreirane su relacije *racun* i *stavka*. Broj stavki računa redundantno je pohranjen u *brojStavki*. Napisati pohranjenu proceduru **brisanje** kojom se za jedan po jedan račun ciji je *brojStavki* veći od 5, brišu njegove stavke s rednim brojevima većim od 5, te se vrijednost atributa *brojStavki* postavlja na 5. Nakon što se se višak stavki jednog računa obriše i njegova vrijednost za *brojStavki* ažurira, rezultat tih dviju operacija mora ostati trajno zabilježen u bazi podataka.
- Ako se bilo koja pogreška dogodi pri obavljanju operacije **DELETE**, procedura u pozivajući program vraća pogrešku s tekstrom 'Pogreška-brisanje'. Ako se bilo koja pogreška dogodi pri obavljanju operacije **UPDATE**, procedura u pozivajući program vraća pogrešku s tekstrom 'Pogreška-izmjena'. U slučaju pojave bilo koje druge pogreške na bilo kojem drugom mjestu u proceduri, procedura u pozivajući program prosljeđuje originalnu pogrešku. Nije dopuštena upotreba pomoćnih relacija, okidača niti **SAVEPOINT** mehanizma. (4 boda)
3. U bazi podataka nalazi se relacija r(A, B, C). Za relaciju r je izgrađen samo jedan indeks za atribut A. Izvršava se SQL naredba **SELECT lista\_atributa FROM r WHERE A = konstanta**. Navesti koji faktori i na koji način utječu na odluku optimizatora hoće li se pri izvršavanju naredbe koristiti navedeni indeks. Odgovor treba napisati u sljedećem obliku:
- a) *lista\_atributa* utječe na sljedeći način: ako lista atributa [nadopuniti tekst] tada se [više/manje] isplati koristiti indeks zato jer [nadopuniti tekst]  
pod b), c), itd, navesti daljnje faktore i pripadna obrazloženja u sličnom obliku kao pod a) (4 boda)
4. Zadane su relacije r(A, B), s(C, D), t(E, F). Izvršava se upit koji je opisan sljedećim izrazom relacijske algebre:
- $$\sigma_{A=D \wedge B < C \wedge D=200 \wedge F \neq 200} ((r \times s) \times t)$$
- a) nacrtati inicijalno stablo upita (*query tree*) za prikazani izraz relacijske algebre  
b) nacrtati stablo upita i **pripadni izraz relacijske algebre** nakon provedene heurističke optimizacije. Nije potrebno crtati pojedine korake optimizacije (dovoljno je konačno rješenje). Fizičke operatore i način prosljeđivanja međurezultata nije potrebno ucrtavati u stablu.  
c) navesti dva (odabratи по жељи) različita pravila za transformaciju izraza relacijske algebre koji su korišteni u b) dijelu zadatka, te za svako od njih opisati kako je korišteno u konkretnom primjeru. (4 boda)

5. Zadane su relacije  $r(A, B, C)$  i  $s(D, E)$ .

Primarni ključevi su potcrtani. Nema indeksa. Na raspolaganju je 100 blokova primarne memorije. Svi međurezultati se materijaliziraju (zapisuju u sekundarnu memoriju). Za operaciju spajanja koristi se spajanje raspršenim adresiranjem (*hash join*). Operacije se izvršavaju redoslijedom kako je navedeno u izrazu relacijske algebre, što znači da ne treba provoditi heurističku optimizaciju. Izvršava se sljedeća operacija relacijske algebre:

$$\sigma_{B \leq 400} (r) \underset{C = D}{\triangleright \triangleleft} \sigma_{E \text{ LIKE } \%T\%} (s)$$

$V(B, r) = 200$	$N(r) = 8000$	$\min(B, r) = 100$
$V(C, r) = 1000$	$N(s) = 20000$	$\max(B, r) = 500$
$V(E, s) = 5000$	$B(r) = 3000$	$\min(C, r) = 0$
	$B(s) = 5000$	$\max(C, r) = 5000$
		$\min(E, r) = 1000$
		$\max(E, r) = 10000$
veličina n-torke(r) = 0.3 blokova		
veličina n-torke(s) = 0.2 blokova		

Koristeći priložene podatke iz rječnika podataka, za zadalu operaciju relacijske algebre:

- Prvo procijeniti broj n-torki u međurezultatima i konačnom rezultatu
- Zatim procijeniti ukupni broj **U/I operacija** tijekom izvršavanja operacije relacijske algebre i broj blokova u konačnom rezultatu.

U rješenju prikazati postupak (ne samo konačni rezultat).

(4 boda)

6. Transakcija T je opisana sljedećim pseudokodom:

```
begin work;
    write(z, 100);
    read(x, p1);
    read(z, p2);
    p3 ← p1 + p2;
    write(y, p3);
    write(x, p2);
commit work;
```

- Nacrtati graf transakcije T. U graf ucrtati isključivo one lúkove koji proizlaze iz semantike transakcije ili su nužni da bi se zadovoljila pravila konstrukcije grafa transakcije.
- Transakciju T iz a) dijela zadatka prikazati u obliku topološkog poretka operacija transakcija.
- Napisati definiciju modela transakcije (nije potrebno pisati definiciju parcijalnog poretka).

(4 boda)

- Što je to *statement-level-rollback*? Navesti jednostavan primjer kojim će se ilustrirati važnost tog mehanizma.
- Objasniti koji se efekti (i zašto) postižu povećanjem, odnosno smanjenjem, učestalosti obavljanja kontrolne točke (*checkpoint*).
- Zašto SUBP koji koristi UNDO tehniku obnove ne smije operaciju *flush log*, kojom se zapis dnevnika *<commit T>* upisuje u stabilnu memoriju, obaviti prije nego su obavljene sve *output(x)* operacije transakcije T?

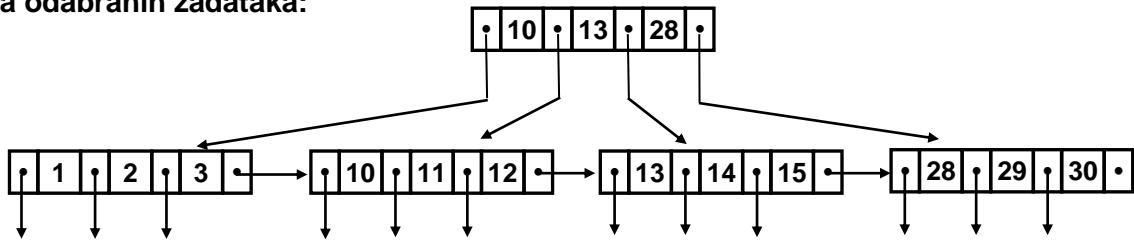
(2 boda)

(2 boda)

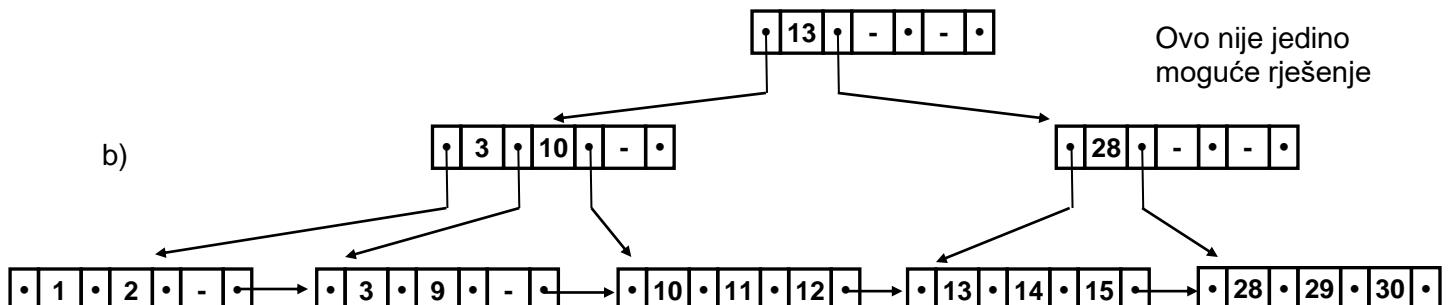
(2 boda)

## Rješenja odabralih zadataka:

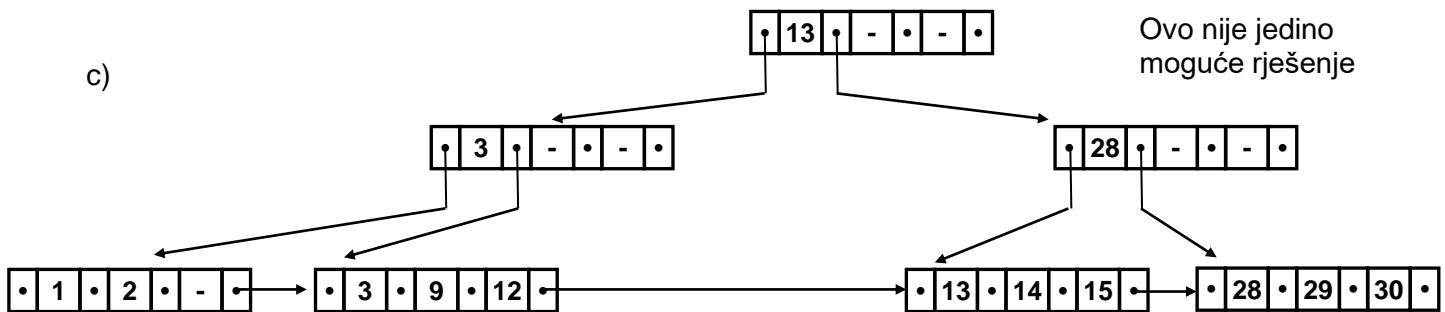
1. a)



b)



c)



2.

```

CREATE PROCEDURE brisanje ()
  DEFINE pBrRac INTEGER;
  DEFINE sqle, isame INTEGER;
  DEFINE errdata CHAR(80);
  FOREACH WITH HOLD
    SELECT brRac INTO pBrRac
      FROM racun WHERE brojStavki > 5
    BEGIN WORK;
    BEGIN
      ON EXCEPTION SET sqle, isame, errdata
        ROLLBACK WORK; -- mora biti ovdje, a ne na početku procedure!
        RAISE EXCEPTION -746, 0, "Pogreška-brisanje";
    END EXCEPTION
    DELETE FROM stavka WHERE brRac = pBrRac AND rbrStavka > 5;
    END
    BEGIN
      ON EXCEPTION SET sqle, isame, errdata
        ROLLBACK WORK; -- mora biti ovdje, a ne na početku procedure!
        RAISE EXCEPTION -746, 0, "Pogreška-izmjena";
    END EXCEPTION
    UPDATE racun SET brojStavki = 5 WHERE brRac = pBrRac;
    END
    COMMIT WORK;
  END FOREACH;
END PROCEDURE;

```

3. a) **lista\_atributa** utječe na sljedeći način: ako lista atributa sadrži samo atribut A tada se više isplati koristiti indeks zato jer se podaci ne trebaju dohvaćati iz blokova s podacima (dovoljan je *key-only-index-scan*)

- b) **stupanj grupiranja** utječe na sljedeći način: ako je stupanj grupiranja nepovoljan, tada se manje isplati koristiti indeks jer su n-torce koje zadovoljavaju uvjet selekcije raspršene po više blokova (treba obaviti više U/I operacija za dohvat blokova s podacima). [ima/nema cluster indeks nije točan odgovor]
- c) **procijenjeni broj n-torki rezultata selekcije** utječe na sljedeći način: ako je taj broj relativno velik, tada se manje isplati koristiti indeks jer trošak prolaska kroz indeks + dohvat blokova s podacima može nadmašiti trošak čitanja svih blokova (linearne pretrage) [broj različitih vrijednosti, raspršenje vrijednosti, ima/nema histogram, broj n-torki koje vraća upit, itd. nisu točni ili nisu dovoljni odgovori] /\* dubina stabla nije bitna \*/

4. a) -

- b) rezultat optimizacije (stablo): -  
rezultat optimizacije (izraz relacijske algebre):  $(\sigma_{A=200}(r) \triangleright \triangleleft_{A=D \wedge B < C} \sigma_{D=200}(r)) \times \sigma_{F \neq 200}(t)$
- c) -

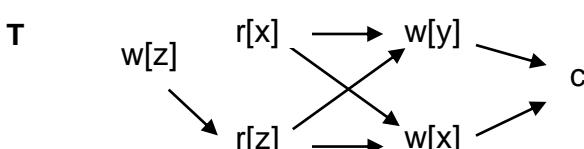
5. a) Broj n-torki

- $t_1 = \sigma_{B \leq 400}(r) = N(t_1) \Rightarrow 8000 \cdot (400-100)/(500-100) = 6000$
- $t_2 = \sigma_E \text{ LIKE } \%T\% (s) \Rightarrow N(t_2) = 20000 / 5 = 4000$
- $t_3 = t_1 \triangleright \triangleleft_{C=D} t_2 \Rightarrow D \text{ je ključ u } t_2 \Rightarrow N(t_3) = N(t_1) = 6000$

b) Broj U/I operacija

- veličina n-torce u konačnom rezultatu  $0.3 + 0.2 = 0.5$
- veličina konačnog rezultata  $B(t_3) = N(t_3) \cdot 0.5 = 6000 \cdot 0.5 = 3000$
- čitanje r pri obavljanju  $\sigma_{B \leq 400}(r) \Rightarrow 3000$
- zapisivanje međurezultata  $t_1 \Rightarrow N(t_1) \cdot 0.3 = 6000 \cdot 0.3 = 1800$
- čitanje s pri obavljanju  $\sigma_E \text{ LIKE } \%T\% (s) \Rightarrow 5000$
- zapisivanje međurezultata  $t_2 \Rightarrow N(t_2) \cdot 0.2 = 4000 \cdot 0.2 = 800$
- obavljanje  $t_1 \triangleright \triangleleft_{C=D} t_2$   
niti jedna ne stane u međuspremnik  $\Rightarrow 3 \cdot (B(t_1) + B(t_2)) = 3 \cdot (1800 + 800) = 7800$
- ukupno (bez zapisivanja konačnog rezultata)  $\Rightarrow 3000 + 1800 + 5000 + 800 + 7800 = 18400$  U/I operacija

6. a)



- b) npr.  $w[z] r[z] r[x] w[x] w[y] c$  (nije jedino moguće rješenje)  
c) -

7. Statement-level-rollback nije "korištenje implicitne transakcije za svaku operaciju"; statement-level-rollback nije "SAVEPOINT mehanizam". Za točan odgovor pogledati predavanja.

8. -

9. -

Sustavi baza podataka - Međuispit  
3. svibnja 2018.

s rješenjima odabralih zadataka

1. U relaciji  $r$  je za atribut A, koji je primarni ključ relacije  $r$ , izgrađeno B<sup>+</sup> stablo reda 11. Relacija sadrži 12 000 (dvanaest tisuća) n-torki.
  - a) Izračunati minimalnu dubinu stabla ("dubinu stabla "u najboljem slučaju"). U obzir uzeti da minimalni i maksimalni dopušteni broj ključeva u listovima i internim čvorovima stabla nije (uvijek) jednak. Nije potrebno izvoditi opći izraz (formulu) za izračunavanje dubine stabla. U rješenju mora biti vidljivo kako se do rezultata došlo: napisati samo npr. "dubina=10" nije prihvatljivo rješenje.
  - b) Ponoviti zadatak pod a), ali ovog puta umjesto *minimalne* dubine, izračunati *maksimalnu* dubinu stabla ("dubinu stabla "u najgorem slučaju").
  - c) Na temelju čega SUBP odabire red B<sup>+</sup> stabla? (4 boda)
  
2. a) Nacrtati R-stablo za prostorne podatke (objekte, odnosno geometrijske likove a - e) prikazane na slici.  
 Najveći dopušteni broj zapisa (ključ+kazaljka) u listovima i internim čvorovima R-stabla je 3, a najmanji je 2. Rješenje a) dijela zadatka treba sadržavati dvije slike: na prvoj slici nacrtati sve minimalne granične okvire (MBR). Na drugoj slici nacrtati R-stablo.  
 b) Objasniti postupak kojim se pomoću R-stabla iz a) dijela zadatka pronaže svi objekti koji su u presjeku s krugom polumjera 1 (promjera 2) čiji je centar u točki na koordinatama ( $x=3, y=0$ ) (4 boda)
  
3. Koji se problem povezan s međuspremnicima podataka, zašto i u kojoj tipičnoj situaciji može pojaviti u sustavu koji koristi REDO tehniku obnove? (3 boda)
  
4. Transakcija T je opisana sljedećim pseudokodom:  

```
begin work;
    read(x, p1);
    read(y, p2);
    p3 ← p1 + 100;
    write(z, p3);
    read(z, p4);
    p5 ← p2 + p4;
    write(x, p5);
commit work;
```

  - a) Nacrtati graf transakcije T. U graf ucrtati isključivo one lúkove koji proizlaze iz semantike transakcije ili su nužni da bi se zadovoljila pravila konstrukcije grafa transakcije.
  - b) Transakciju T iz a) dijela zadatka prikazati u obliku topološkog poretka operacija transakcija.
  - c) U topološkom poretku iz b) dijela zadatka navesti sve parove operacija koje smiju međusobno zamijeniti mesta. Zašto u topološkom poretku neke operacije smiju, a neke ne smiju zamijeniti međusobni poredak? (5 bodova)
  
5. a) Definirati svojstvo izdržljivosti transakcije.  
 b) Na koji način se performanse sustava mogu povećati na uštrb striktнog osiguranja svojstva izdržljivosti? Napomena: nije potrebno navoditi naredbu ili parametar za podešavanje nekog konkretnog sustava, dovoljno je ukratko objasniti princip. (4 boda)

6. U bazi podataka nalaze se relacije **r** i **s**. Izvršava se SQL naredba

```
SELECT lista_atributa FROM r, s WHERE uvjet_spajanja_r_i_s
```

Navesti koji faktori (u navedenoj SQL naredbi i relacijama r i s) i na koji način utječu na odabir koja je od fizičkih metoda najprikladnija za spajanje relacija. Odgovor treba napisati u sljedećem obliku:

- a) ako [nadopuniti tekst] tada je [više|manje] prikladna metoda spajanja [navesti naziv fizičke metode spajanja] zato jer [nadopuniti tekst]
- pod b), c), itd., navesti daljnje faktore i pripadna obrazloženja u sličnom obliku kao pod a) (4 boda)

7. Zadane su relacije r(A, B, C, D) i s(E, F).

Primarni ključevi su podcrtani. Nema indeksa. Na raspolaganju su 1002 bloka primarne memorije. Svi međurezultati se materijaliziraju (zapisuju u sekundarnu memoriju). Za operaciju spajanja koristi se fizički operator *block nested-loop join*. Operacije se izvršavaju redoslijedom kako je navedeno u izrazu relacijske algebre, što znači da ne treba provoditi heurističku optimizaciju. Izvršava se sljedeća operacija relacijske algebre:

$$\Pi_{C,D,E}^B (\sigma_{B=10 \vee C < 20}(r) \bowtie_{D=E} s)$$

$N(r) = 100\ 000$
$N(s) = 2\ 500$

$B(r) = 75\ 000$
$B(s) = 1\ 000$

$V(C, r) = 5$
$V(D, r) = 200$
$V(F, s) = 100$

veličina n-torce(r) = 0.5 blokova
veličina n-torce(s) = 0.1 blokova

Koristeći priložene podatke iz rječnika podataka, za zadanu operaciju relacijske algebre:

- a) Prvo procijeniti broj n-torki u međurezultatima i konačnom rezultatu
- b) Zatim procijeniti ukupni broj **U/I operacija** tijekom izvršavanja operacije relacijske algebre  
U rješenju prikazati postupak (ne samo konačni rezultat). (4 boda)

8. a) Zašto kontrolnu točku (CP) nije dobro obavljati niti prečesto niti prerijetko?

- b) Treba li (i zašto) povećati ili smanjiti učestalost obavljanja CP ako u sustav počne pristizati povećani broj transakcija koje obavljaju operacije pisanja?
- c) Treba li (i zašto) povećati ili smanjiti učestalost obavljanja CP ako u sustav počne pristizati povećani broj transakcija koje obavljaju operacije čitanja? (4 boda)

9. a) Zašto se zapisi logičkog dnevnika prvo upisuju u *on-line* stabilnu memoriju i zašto se ondje moraju zadržati određeno vrijeme?

- b) Zašto se zapisi logičkog dnevnika iz *on-line* stabilne memorije kopiraju u *off-line* stabilnu memoriju? (3 boda)

## Rješenja odabralih zadataka:

1. a) 11 kazaljki u korijenu

121 kazaljka u 11 čvorova

1331 kazaljka u 121 čvoru

13 310 kazaljki u 1331 listu, dovoljno

$\Rightarrow$  ukupno 4 razine, dubina je 4

b) 2 kazaljke u korijenu

12 kazaljki u 2 čvora

72 kazaljke u 12 čvorova

432 kazaljke u 72 čvora

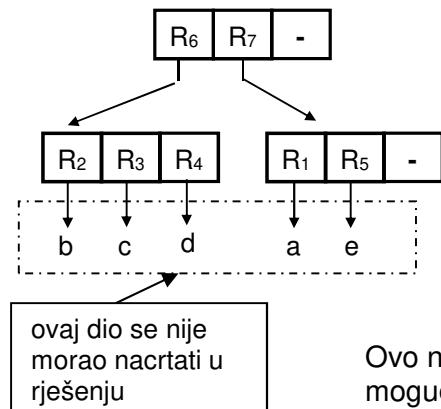
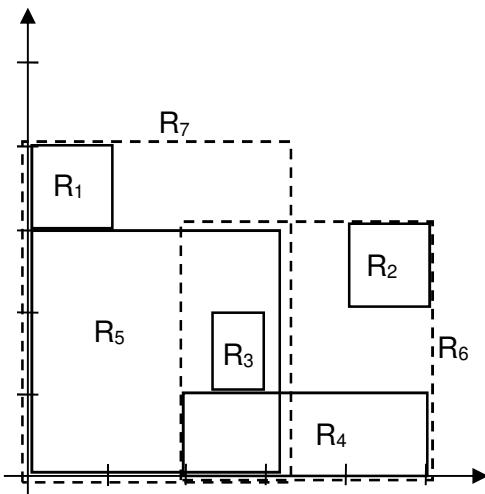
2592 kazaljke u 432 čvora

12 960 kazaljki u 2592 lista (previše)

$\Rightarrow$  stoga ukupno 5 razina, dubina je 5

c) veličina fizičkog bloka, veličina ključa, veličina pokazivača

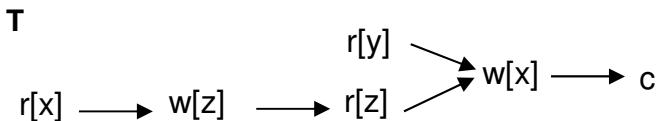
2.a)



Ovo nije jedino moguće rješenje

b) Odrediti minimalni granični okvir (MBR) za zadani krug (npr.  $MBR_x$ ). U korijenu stabla pronaći sve MBR-ove koji se sijeku s  $MBR_x$ . To su  $R_6$  i  $R_7$ . Zatim slijediti pokazivač uz  $R_6$  i utvrditi koji je od MBR-ova  $R_2$ ,  $R_3$ ,  $R_4$  u presjeku s  $MBR_x$ . To je  $R_4$ . Utvrditi je li objekt  $d$  u presjeku sa zadanim krugom (ako jest, dodati ga u rješenje). Zatim slijediti pokazivač uz  $R_7$  i utvrditi koji je od MBR-ova  $R_1$  i  $R_5$  u presjeku s  $MBR_x$ . To je  $R_5$ . Utvrditi je li objekt  $e$  u presjeku sa zadanim krugom (ako jest, dodati ga u rješenje).

4. a)



b) npr.  $r[x] w[z] r[z] r[y] w[x] c$  ili npr.  $r[x] r[y] w[z] r[z] w[x] c$  (nisu jedina moguća rješenja)

c) u prvoj varijanti topološkog poretka samo  $r[y]$  i  $r[z]$ ; u drugoj varijanti  $r[x]$  i  $r[y]$ ,  $r[y]$  i  $w[z]$

## 7. a) Broj n-torki

- $t_1 = \sigma_{B=10 \vee C <> 20}(r)$
- $f_1 = f(B=10, r) = N(\sigma_{B=10}(r)) / N(r) = N(r) / 10 / N(r) = 0.1$
- $f_2 = f(C=20, r) = N(\sigma_{C=20}(r)) / N(r) = N(r) / V(C, r) / N(r) = 0.2$
- $f_3 = f(C <> 20, r) = 1 - f_2 = 0.8$
- $N(t_1) = (1 - (1-f_1) \cdot (1-f_3)) \cdot N(r) = (1 - 0.9 \cdot 0.2) \cdot N(r) = 82\ 000$
- $t_2 = t_1 \triangleright \triangleleft_{D=E} s \Rightarrow E \text{ je ključ u } s \Rightarrow N(t_2) = N(t_1) = 82\ 000$
- $t_3 = \Pi^B_{C,D,E}(t_2) \Rightarrow \text{bag verzija projekcije} \Rightarrow N(t_3) = N(t_2) = 82\ 000$

## b) Broj U/I operacija

- čitanje r pri obavljanju  $\sigma_{B=10 \vee C <> 20}(r) \Rightarrow 75\ 000$
- zapisivanje međurezultata  $t_1 \Rightarrow N(t_1) \cdot 0.5 = 82\ 000 \cdot 0.5 = 41\ 000$
- obavljanje  $t_1 \triangleright \triangleleft_{D=E} s$   
s stane u međuspremnik  $\Rightarrow B(t_1) + B(s) = 41\ 000 + 1\ 000 = 42\ 000$
- zapisivanje međurezultata  $t_2 \Rightarrow N(t_2) \cdot (0.5 + 0.1) = 82\ 000 \cdot 0.6 = 49\ 200$
- čitanje  $t_2$  pri obavljanju  $\Pi^B \Rightarrow 49\ 200$
- ukupno (bez zapisivanja konačnog rezultata)  $\Rightarrow 75\ 000 + 41\ 000 + 42\ 000 + 49\ 200 + 49\ 200 = 256\ 400$  U/I operacija

## 8. Umjesto rješenja:

- tijekom kontrolne točke (dalje u tekstu: CP) ne odbijaju se sve operacije i transakcije
- tijekom CP ne zaustavlja se sustav
- tijekom CP ne zaustavlja se rad sustava za obnovu
- CP ne dovodi do "povećanih zapisa dnevnika"
- češćim obavljanjem CP ne skraćuje se vrijeme obnove nakon "pada", "rušenja", "kvara", "greške", "pogreške" ili pogreške medija (nego nakon pogreške sustava)
- rijetko obavljanje CP
  - ne znači da bi obnova trajala dana
  - ne može biti uzrok nekonzistentnosti
  - ne može narušiti integritet baze podataka
  - ne može uzrokovati gubitak podataka
  - ne "povećava mogućnost gubljenja više podataka"
- CP ne utječe na veličinu datoteke dnevnika
- CP ne utječe (u ovom kontekstu značajno) na veličinu dnevnika
- tijekom CP ne spremaju se "sve izmjene" u stabilnu memoriju
- nije istina da "izmjene obavljene nakon zadnje CP nisu pohranjene u bazu podataka".
- prečesto korištenje CP ne "usporava sustav zbog povećanog čekanja"
- nije istina da CP uzrokuje "praznjenje međuspremnika"
- prečesto ili prerijetko izvršavanje CP ne može "pregaziti željenu vrijednost" niti može "izazvati krivo izvršavanje transakcije"
- CP ne "uzrokuje memorjsko zauzeće"

1. Elementi baze podataka x, y i z nalaze se u različitim blokovima (*block*) postojane memorije. Tijekom obavljanja transakcije T, menadžer podataka (*data manager*) koji koristi tehniku obnove UNDO, redom izvršava sljedeće operacije:

1. start	8. read(z, p)
2. input(x)	9. output(y)
3. write(x, 7)	10. commit
4. input(y)	11. flush log
5. input(z)	12. output(x)
6. flush log	13. flush log
7. write(y, 8)	

Menadžer podataka u ovom primjeru ne poštuje neka od pravila propisanih tehnikom UNDO.

- a) Za svaku operaciju koja nije izvršena u skladu s pravilima navesti o kojoj se operaciji radi i koje pravilo narušava.
- b) Za svaku operaciju i prekršeno pravilo navedeno u rješenju pod a) navesti do kakvih bi problema moglo doći zbog kršenja tog pravila. (6 bodova)

2. Elementi u bazi podataka imaju oznake x, y i z i odnose se redom na vrijednosti atributa stanje u n-torkama sa šiframa 1000, 2000 i 3000.

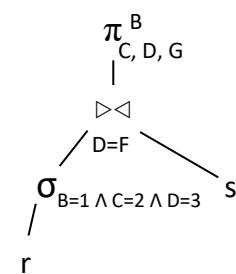
Transakcija T je opisana sljedećim programskim kôdom:

```
-- definiraj tri varijable: px, py, pz
DECLARE @px INTEGER, @py INTEGER, @pz INTEGER;
BEGIN TRANSACTION;
-- stanje iz n-torke sifra=2000 učitaj u varijablu py
SELECT @py = stanje FROM tab WHERE sifra = 2000;
UPDATE tab SET stanje = 199 WHERE sifra = 2000;
SELECT @px = stanje FROM tab WHERE sifra = 1000;
UPDATE tab SET stanje = @px + @py WHERE sifra = 1000;
UPDATE tab SET stanje = @px WHERE sifra = 3000;
SELECT @pz = 5 * stanje FROM tab WHERE sifra = 3000;
COMMIT TRANSACTION;
```

```
CREATE TABLE tab (
    sifra    INTEGER PRIMARY KEY
, stanje   INTEGER);
INSERT INTO tab VALUES (1000, 10);  
y
INSERT INTO tab VALUES (2000, 20);  
z
INSERT INTO tab VALUES (3000, 30);  
x
```

- a) Nacrtati graf transakcije T. U graf ucrtati isključivo one lúkove koji proizlaze iz semantike transakcije ili su nužni da bi se zadovoljila pravila konstrukcije grafa transakcije.
  - b) Transakciju T iz a) dijela zadatka prikazati u obliku nekog topološkog poretka operacija transakcija u kojem redoslijed obavljanja operacija neće biti potpuno jednak redoslijedu obavljanja operacija u prikazanom programskom kôdu. (6 bodova)
3. Zadane su relacije  $r(A, B, C, D)$  i  $s(E, F, G)$ . Primarni ključevi su podcrtni. Obični (*non-clustered*) indeksi su kreirani za sljedeće atribute: indeks  $i_1$  za atribut A,  $i_2$  za B,  $i_3$  za C,  $i_4$  za D,  $i_5$  za E,  $i_6$  za F i  $i_7$  za G. Faktor grupiranja za sve indekse je nizak (slabo, loše grupiranje). Na raspolaganju je 100 000 blokova primarne memorije. Optimizator na raspolaganju ima sljedeće statističke podatke:

$N(r)=50\ 000$	$N(s)=500\ 000$
$B(r)=1\ 000$	$B(s)=10\ 000$
$V(A, r)=50\ 000$	$V(E, s)=500\ 000$
$V(B, r)=10$	$V(F, s)=250\ 000$
$V(C, r)=20$	$V(G, s)=100\ 000$
$V(D, r)=50$	
$d(i_1)=d(i_2)=d(i_3)=d(i_4)=3$	$d(i_5)=d(i_6)=d(i_7)=5$



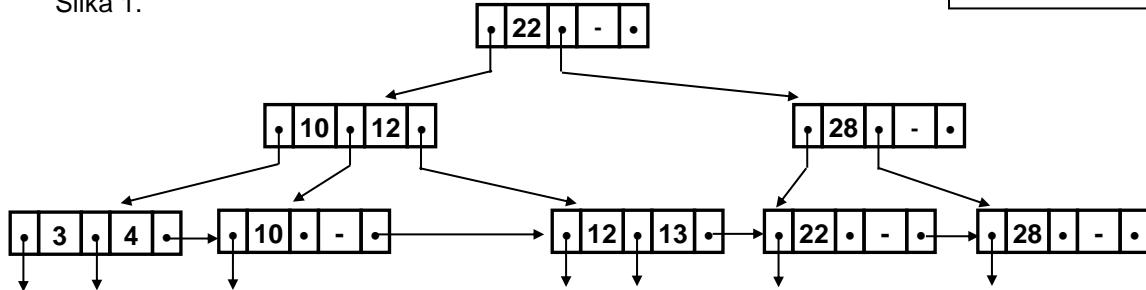
Napisati koje fizičke operatore će optimizator odabrati za izvršavanje operacije selekcije i operacije spajanja u planu izvršavanja na slici i objasniti zašto ih je optimizator odabrao. (6 bodova)

4. Nacrtati B<sup>+</sup>-stablo:

- a) nakon unosa zapisa s ključem **21** u originalno B<sup>+</sup>-stablo na slici 1.  
 b) nakon brisanja zapisa s ključem **22** iz originalnog B<sup>+</sup>-stabla na slici 1.  
 (6 bodova)

Dovoljno je nacrtati samo konačna rješenja, posebno za a) i posebno za b) dio zadatka

Slika 1.



5. Zadane su relacije r(A, B, C) i s(D, E). Primarni ključevi relacija su podcrtni. Nad relacijama nije izgrađen niti jedan indeks. Za operaciju Kartezijevog produkta koristi se fizički operator *block nested-loop join*. Za obavljanje fizičkih operacija na raspolaganju su 602 bloka primarne memorije. Svi međurezultati se materijaliziraju (zapisuju u sekundarnu memoriju). Operacije se izvršavaju redoslijedom kako je navedeno u izrazu relacijske algebre, što znači da ne treba provoditi heurističku optimizaciju.

$$\sigma_{B \neq 150 \vee C \leq 700}(r) \times s$$

$$\begin{aligned} V(B, r) &= 20 \\ V(C, r) &= 400 \\ V(E, s) &= 10 \end{aligned}$$

$$\begin{aligned} \text{veličina n-torke}(r) &= 0.1 \text{ blokova} \\ \text{veličina n-torke}(s) &= 0.4 \text{ blokova} \end{aligned}$$

$$\begin{aligned} N(r) &= 10000 \\ N(s) &= 1000 \end{aligned}$$

$$\begin{aligned} B(r) &= 4000 \\ B(s) &= 500 \end{aligned}$$

$$\begin{aligned} \min(B, r) &= 100 \\ \max(B, r) &= 500 \\ \min(C, r) &= 300 \\ \max(C, r) &= 800 \end{aligned}$$

Koristeći priložene podatke iz rječnika podataka, za zadanu operaciju relacijske algebre:

- a) Prvo procijeniti broj n-torki u međurezultatima i konačnom rezultatu  
 b) Zatim procijeniti ukupni broj U/I operacija tijekom izvršavanja operacije relacijske algebre i broj blokova u konačnom rezultatu.

U rješenju prikazati postupak (ne samo konačni rezultat).

(6 bodova)

6. Neke od sljedećih izjava o kontrolnoj točki (*checkpoint*, CP) nisu istinite:

- a) Neposredno nakon CP (nakon što se u dnevnik u stabilnoj memoriji upiše zapis `<chkpt L>`) međuspremni podataka su prazni.  
 b) Od početka do kraja postupka kojeg SUBP obavlja u vezi CP, sustav sigurno neće obaviti niti jednu SQL naredbu zadalu od strane korisnika.  
 c) Za poništavanje transakcija nikad neće biti potrebno koristiti zapise dnevnika koji se u dnevniku nalaze prije pretposljednjeg zapisa `<chkpt L>`.  
 d) U slučaju obnove baze podataka nakon pogreške sustava, za ponovno obavljanje (*redo*) neće biti potrebno koristiti zapise dnevnika koji se u dnevniku nalaze prije posljednjeg zapisa `<chkpt L>`.

Navesti koje od navedenih izjava nisu istinite i objasniti zašto.

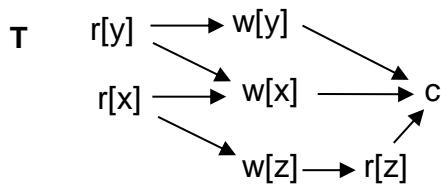
(5 bodova)

## Rješenja odabralih dijelova zadataka:

1. a)

1. Operacija `output(y)` obavljena je prije nego je zapis dnevnika koji se odnosi na operaciju `write(y, 8)` upisan u dnevnik (u stabilnu memoriju). (+ objašnjenje problema)
2. Operacija `flush log` koja je u dnevnik (u stabilnu memoriju) upisala zapis koji se odnosi na operaciju `commit`, obavljena je prije operacije `output(x)`. (+ objašnjenje problema)

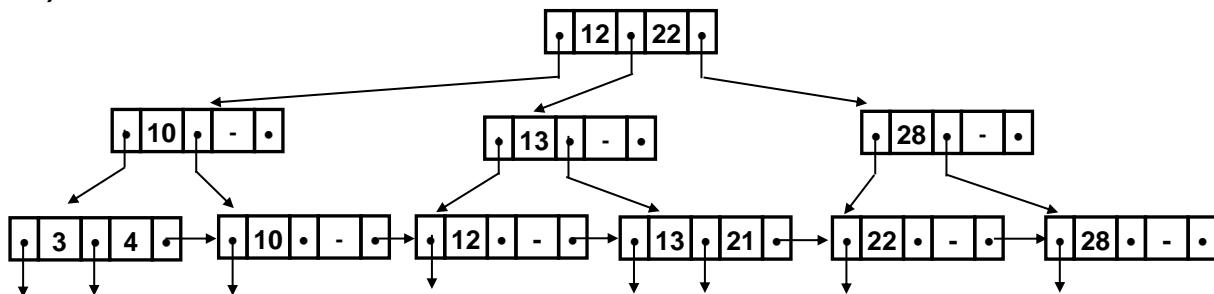
2. a)



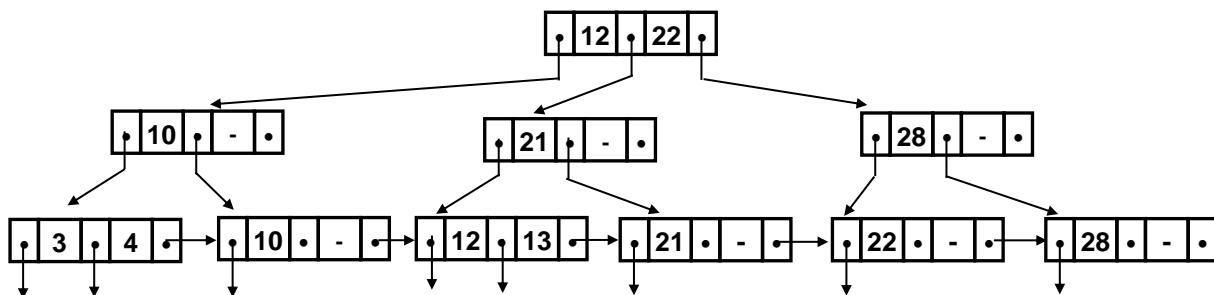
3. Za operaciju selekcije koristi se **table scan** (+ objašnjenje zašto).

Za operaciju spajanja koristi se **indexed nested-loop join** (+ objašnjenje zašto).

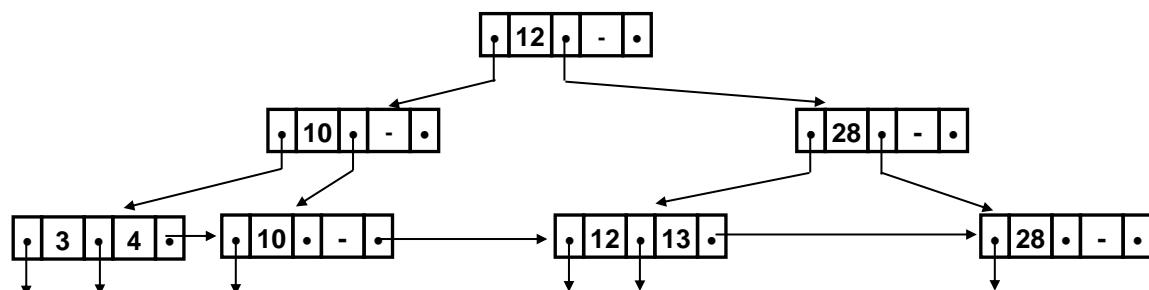
4. a)



ili



b)



**5. a) Broj n-torki**

- $t_1 = \sigma_{B \neq 150 \vee C \leq 700}(r)$
- $f_1 = f(B=150, r) = N(\sigma_{B=150}(r)) / N(r) = N(r) / 20 / N(r) = 0.05$
- $f_2 = f(B \neq 150, r) = 1 - f_1 = 0.95$
- $f_3 = f(C \leq 700, r) = (700 - \min(C, r)) / (\max(C, r) - \min(C, r)) = 0.8$
- $N(t_1) = (1 - (1-f_2) \cdot (1-f_3)) \cdot N(r) = (1 - 0.05 \cdot 0.2) \cdot N(r) = (1 - 0.01) \cdot N(r) = 9900$
- $t_2 = t_1 \times s$
- $N(t_2) = N(t_1) \cdot N(s) = 9900000$

**b) Broj U/I operacija**

- čitanje r pri obavljanju  $\sigma_{B \neq 150 \vee C \leq 700} \Rightarrow 4000$
- zapisivanje međurezultata  $t_1 \Rightarrow N(t_1) \cdot 0.1 = 9900 \cdot 0.1 = 990$
- obavljanje  $t_1 \times s$   
s cijela stane u međuspremnik  $\Rightarrow B(t_1) + B(s) = 990 + 500 = 1490$
- ukupno (bez zapisivanja konačnog rezultata)  $\Rightarrow 4000 + 990 + 1490 = 6480$  U/I operacija
- broj blokova u konačnom rezultatu  $N(t_2) \cdot (0.1 + 0.4) = 4950000$

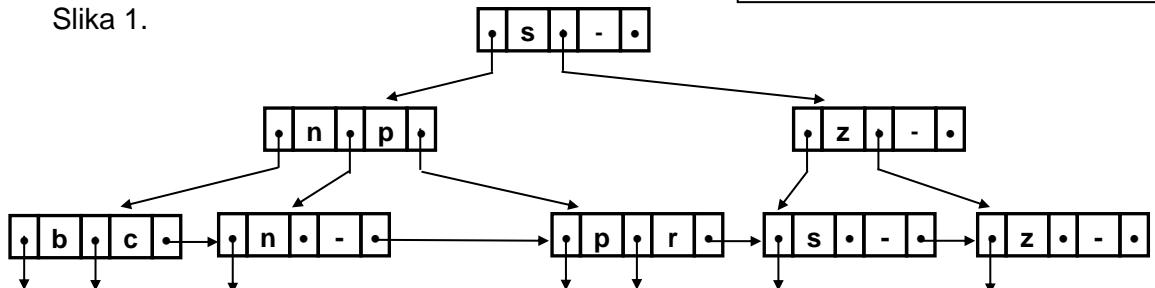
**6.** Nisu istinite izjave a), b), c) (+ objašnjenje zašto).

1. Nacrtati B<sup>+</sup>-stablo:

- a) nakon unosa zapisa s ključem **m** u B<sup>+</sup>-stablo na slici 1.

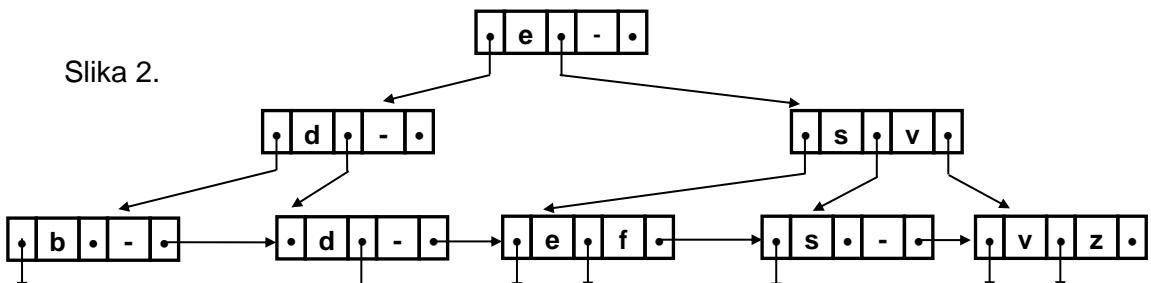
Dovoljno je nacrtati samo konačna rješenja, posebno za a) i posebno za b) dio zadatka

Slika 1.



- b) nakon brisanja zapisa s ključem **d** iz B<sup>+</sup>-stabla na slici 2.

Slika 2.



(6 bodova)

2. Zadane su relacije  $r(A, B, C)$  i  $s(D, E)$ .

Primarni ključevi su podcrtni. Nema indeksa. Na raspolažanju je 200 blokova primarne memorije. Svi međurezultati se materijaliziraju (zapisuju u sekundarnu memoriju). Operacije se izvršavaju redoslijedom kako je navedeno u izrazu relacijske algebre, što znači da ne treba provoditi heurističku optimizaciju.

$$\sigma_{B \leq 200 \wedge C = 100}(r) \times s$$

veličina n-torke(r) = 0.5 blokova
veličina n-torke(s) = 0.3 blokova

$V(B, r) = 100$
$V(C, r) = 50$
$V(E, s) = 20$

$N(r) = 20000$
$N(s) = 1000$

$B(r) = 15000$
$B(s) = 500$

$\min(B, r) = 50$
$\max(B, r) = 250$
$\min(E, s) = 0$
$\max(E, s) = 500$

Koristeći priložene podatke iz rječnika podataka, za zadanu operaciju relacijske algebre:

- a) prvo procijeniti **broj n-torki** u međurezultatima i konačnom rezultatu  
 b) zatim procijeniti broj **U/I operacija** tijekom izvršavanja operacije relacijske algebre i veličinu konačnog rezultata izraženu u broju blokova

U rješenju prikazati postupak (ne samo konačni rezultat).

(6 bodova)

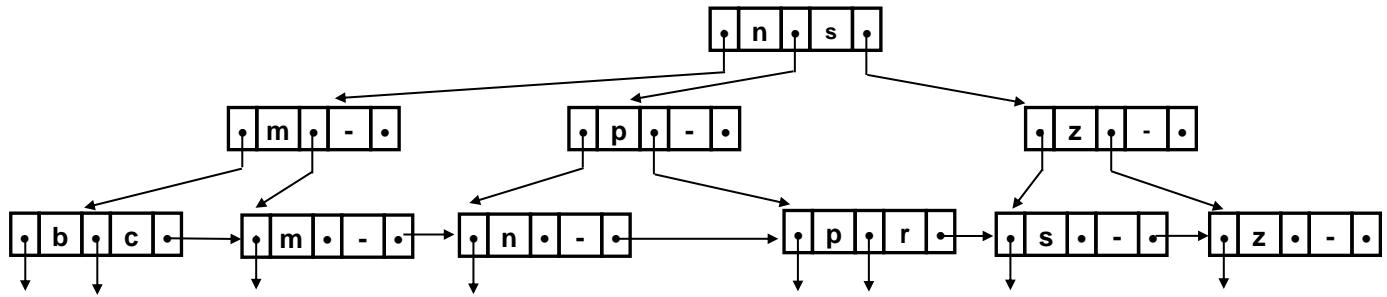
3. Što je inkrementalno arhiviranje baze podataka? Koji se problem ili problemi rješavaju korištenjem inkrementalnog arhiviranja u odnosu na situaciju kada se inkrementalno arhiviranje ne koristi?

(4 boda)

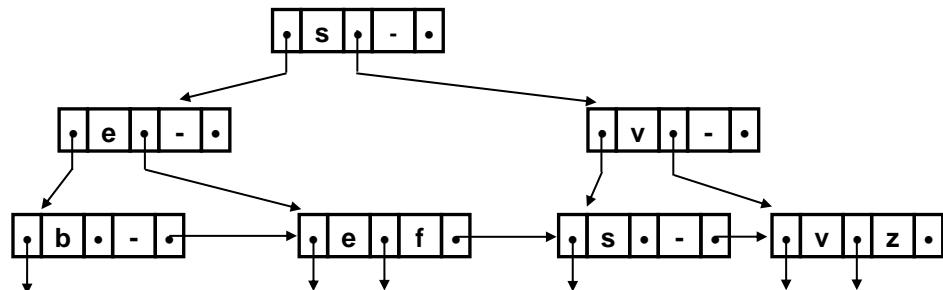
4. a) Zašto kontrolnu točku (CP) nije dobro obavljati niti prečesto niti prerijetko?
- b) Treba li (i zašto) povećati ili smanjiti učestalost obavljanja CP ako u sustav počne pristizati bitno povećani broj transakcija koje obavljaju operacije pisanja?
- c) Treba li (i zašto) povećati ili smanjiti učestalost obavljanja CP ako u sustav počne pristizati bitno povećani broj transakcija koje obavljaju operacije čitanja? (5 bodova)
5. Transakcija  $T_1$  izvršava operacije nad elementima baze podataka x, y, z, v. Zadatak transakcije jest: vrijednost elementa x uvećati za 5; vrijednost elementa y umanjiti za početnu vrijednost elementa x; na zaslon ispisati vrijednost elementa z; vrijednost 200 upisati u element v.
- a) Nacrtati graf transakcije  $T_1$ . U graf ucrtati isključivo one lûkove koji proizlaze iz semantike transakcije ili su nužni da bi se zadovoljila pravila konstrukcije grafa transakcije.
- b) Svaki lûk u grafu označiti proizvoljno odabranim rednim brojem, te referencirajući se na te brojeve, za svaki lûk navesti zašto je ucrtan u graf transakcije. (6 bodova)

Rješenja:

1. a)



b)



2. Broj n-torki

- selektivnost  $f(B \leq 200, r) = N(\sigma_{B \leq 200}(r)) / N(r) = N(r) \cdot (200 - 50) / (250 - 50) / N(r) = 0.75$
- selektivnost  $f(C=100, r) = N(\sigma_{C=100}(r)) / N(r) = N(r) / V(C, r) / N(r) = 0.02$
- $t_1 = \sigma_{B \leq 200 \wedge C=100}(r)$
- $N(t_1) = N(r) \cdot f(B \leq 200, r) \cdot f(C=100, r) = 20\ 000 \cdot 0.75 \cdot 0.02 = 300$
- $t_2 = t_1 \times s$
- $N(t_2) = N(t_1) \cdot N(s) = 300 \cdot 1\ 000 = 300\ 000$
- veličina konačnog rezultata  $N(t_2) \cdot (0.5 + 0.3) = 300\ 000 \cdot 0.8 = 240\ 000$  blokova

Broj U/I operacija

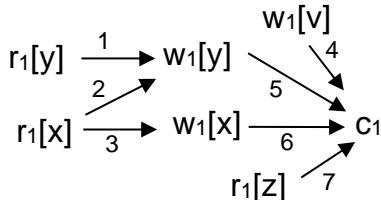
- čitanje relacije  $r$  pri obavljanju  $\sigma_{B \leq 200 \wedge C=100}(r) \Rightarrow B(r) = 15\ 000$
- veličina međurezultata  $B(t_1) = N(t_1) \cdot 0.5 = 300 \cdot 0.5 = 150$
- zapisivanje međurezultata  $t_1 \Rightarrow 150$
- obavljanje  $t_1 \times s$ :  $t_1$  stane cijela u međuspremnik
- $\Rightarrow$  za obavljanje operacije  $B(t_1) + B(s) = 150 + 500 = 650$
- ukupno (bez zapisivanja konačnog rezultata)  $\Rightarrow 15\ 000 + 150 + 650 = 15\ 800$

3. -

4. Umjesto rješenja:

- tijekom kontrolne točke (dalje u tekstu: CP) ne odbijaju se sve operacije i transakcije
- tijekom CP ne zaustavlja se sustav
- tijekom CP ne zaustavlja se rad sustava za obnovu
- CP ne dovodi do "povećanih zapisa dnevnika"
- češćim obavljanjem CP ne skraćuje se vrijeme obnove nakon "pada", "rušenja", "kvara", "greške", "pogreške" ili pogreške medija (nego nakon pogreške sustava)
- rijetko obavljanje CP
  - ne znači da bi obnova trajala danima
  - ne može biti uzrok nekonzistentnosti
  - ne može narušiti integritet baze podataka
  - ne može uzrokovati gubitak podataka
  - ne "povećava mogućnost gubljenja više podataka"
- CP ne utječe na veličinu datoteke dnevnika
- CP ne utječe (u ovom kontekstu značajno) na veličinu dnevnika
- tijekom CP ne spremaju se "sve izmjene u stabilnu memoriju"
- nije istina da "izmjene obavljene nakon zadnje CP nisu pohranjene u bazu podataka".
- prečesto korištenje CP ne "usporava sustav zbog povećanog čekanja"
- nije istina da CP uzrokuje "praznjenje međuspremnika"
- prečesto ili prerijetko izvršavanje CP ne može "pregaziti željenu vrijednost" niti može "izazvati krivo izvršavanje transakcije"
- CP ne "uzrokuje memorjsko zauzeće"
- CP ne određuje koji se zapisi čuvaju u dnevniku
- to što se dugo ne obavlja CP, ne znači da radi toga raste veličina dnevnika
- "gotove transakcije" se ne "spremaju u CP"
- CP ne uzrokuje "blokiranje čitanja"
- podaci ne "prestaju biti ažurni" zbog rijetkog obavljanja CP
- često obavljanje CP ne znači da će "transakcije biti često poništene"
- "vjerojatnost događanja greške" ne ovisi o učestalosti obavljanja CP
- učestalost CP ne utječe na "češće zapisivanje podataka u memoriju"

5. a)

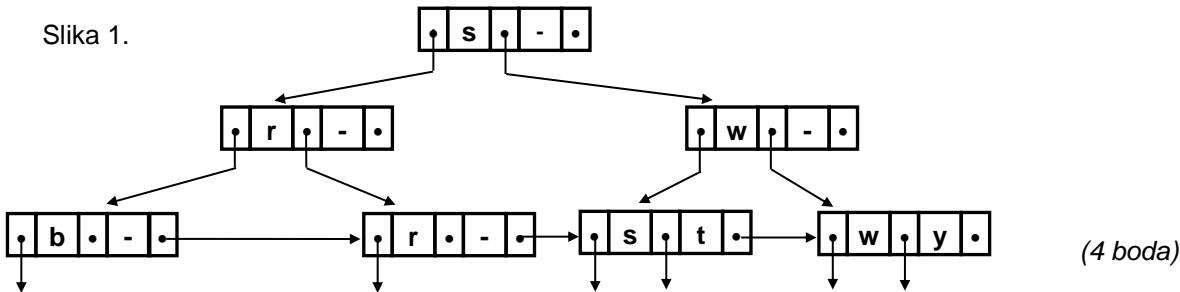


b)

1. na temelju definicije transakcije: operacije čitanja i pisanja nad istim elementom moraju biti poredane.  $r_1[y]$  se obavlja prije  $w_1[y]$  jer se nova vrijednost za  $y$  izračunava na temelju stare vrijednosti  $y$
2. na temelju semantike transakcije:  $r_1[x]$  se obavlja prije  $w_1[y]$  jer se nova vrijednost za  $y$  izračunava na temelju stare vrijednosti  $x$
3. na temelju definicije transakcije: operacije čitanja i pisanja nad istim elementom moraju biti poredane.  $r_1[x]$  se obavlja prije  $w_1[x]$  jer se nova vrijednost za  $x$  izračunava na temelju stare vrijednosti  $x$
- 4, 5, 6, 7. na temelju definicije transakcije: svaka operacija mora prethoditi operaciji *commit*

Sustavi baza podataka - Međuispit - 22. travnja 2022.

1. Nacrtati B<sup>+</sup>-stablo nakon brisanja zapisa s ključem **r** iz B<sup>+</sup>-stabla na slici 1. Dovoljno je nacrtati samo konačno rješenje.



2. a) Nacrtati R-stablo reda 3 za prostorne podatke (objekte, odnosno geometrijske likove **a - e**) prikazane na slici 2.

Najmanji broj zapisa (ključ+kazaljka) u listu i internom čvoru R-stabla je 2. Rješenje a) dijela zadatka treba sadržavati dvije slike. Na prvoj slici nacrtati i označiti sve minimalne granične okvire. Na drugoj slici nacrtati R-stablo.

- b) Objasniti postupak kojim se pomoću R-stabla iz a) dijela zadatka pronalaze oni od objekata **a - e** koji se sijeku s krugom **k** polumjera 1 s ishodištem u točki ( $x=2.5, y=1.0$ )  
(5 bodova)

3. Zadane su relacije  $r(A, B)$ ,  $s(C, D)$  i  $t(E, F)$ , za njih raspoloživi statistički podaci i operacija relacijske algebre.

$$(\sigma_{A=100}(r) \times \sigma_{D>700}(s)) \triangleright \triangleleft t$$

$$d(idx_A) = 4$$

$B(r) = 10\ 000$	$B(s) = 20\ 000$	$B(t) = 500$	$veličina n-torke(r) = 0.2$ blokova	$veličina n-torke(s) = 0.3$ blokova	$veličina n-torke(t) = 0.1$ blokova
$N(r) = 20\ 000$	$N(s) = 40\ 000$	$N(t) = 2\ 000$	$V(B, r) = 2\ 000$	$V(D, s) = 200$	$\min(D, s) = 600$ $\max(D, s) = 1\ 000$

Primarni ključevi relacija su podcrtni. Jedini kreirani indeks je indeks *idxA* za atribut A u relaciji r. Na raspolaganju su 1002 bloka primarne memorije. Svi međurezultati se materijaliziraju zapisivanjem u sekundarnu memoriju. Operacije se izvršavaju redoslijedom kako je navedeno u izrazu relacijske algebre, što znači da ne treba provoditi heurističku optimizaciju.

- a) prvo procijeniti **broj n-torki** u svim međurezultatima i konačnom rezultatu  
b) zatim temeljem raspoloživih statističkih podataka odabrati najbolji fizički operator za svaku od operacija relacijske algebre i navesti zašto je odabran baš taj fizički operator, te procijeniti broj **U/I operacija** tijekom izvršavanja svake od operacija relacijske algebre i veličinu svakog materijaliziranog međurezultata, te konačnog rezultata izraženih u broju blokova

U rješenju prikazati postupak (ne samo konačni rezultat).

(7 bodova)

4. Transakcija T je opisana sljedećim pseudo-kodom:

```

start;
  write(x, 100);
  read(x, p1);
  read(z, p2);
  p3 ← p1 + p2;
  write(y, p3);
  p4 ← p2 + 100;
  write(z, p4);
commit;

```

- a) Nacrtati graf transakcije T. U graf ucrtati isključivo one lúkove koji proizlaze iz semantike transakcije ili su nužni da bi se zadovoljila pravila konstrukcije grafa transakcije.
- b) Zašto se u definiciji modela transakcije zahtijeva da među operacijama *read* i *write* koje transakcija izvršava nad istim elementom x poredak uvijek mora biti utvrđen?

(5 bodova)

5. a) Zašto SUBP koji koristi tehniku obnove *UNDO* smije u stabilnu memoriju upisati zapis  $\langle \text{commit } T_i \rangle$  tek nakon što izvrši operacije *output(x)* za sve elemente x u koje je pisala transakcija  $T_i$ ?

Slika 3.

```

start
write(x, 7)
read(x, p)
write(y, p)
flush log
output(x)
commit
flush log
output(y)

```

- b) Postoji li operacija zbog koje je SUBP prekršio pravila vođenja dnevnika ako koristi tehniku obnove *REDO*, a izvršava operacije redoslijedom prikazanim na slici 3? Ako postoji, navesti o kojoj se operaciji i pravilu radi.
- c) Postoji li operacija zbog koje je SUBP prekršio pravila vođenja dnevnika ako koristi tehniku obnove *UNDO/REDO*, a izvršava operacije redoslijedom prikazanim na slici 3? Ako postoji, navesti o kojoj se operaciji i pravilu radi.

(6 bodova)

6. Definirati svojstva transakcije *atomarnost* i *izdržljivost*.

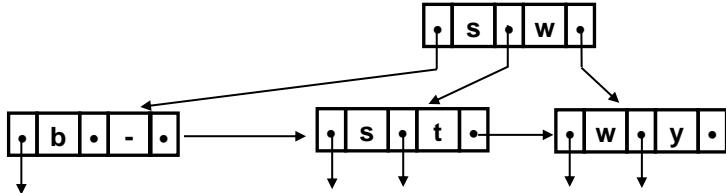
(3 boda)

7. Menadžer međuspremnika u SUBP-u u pravilu koristi algoritam LRU. SUBP neće koristiti LRU u slučaju u kojem bi njegovo korištenje bilo štetno po performanse. Opisati karakteristični slučaj u kojem SUBP neće primjeniti LRU, te dva različita pristupa (obrazložena na predavanjima) koja sustavi za upravljanje bazama podataka koriste u takvom slučaju.

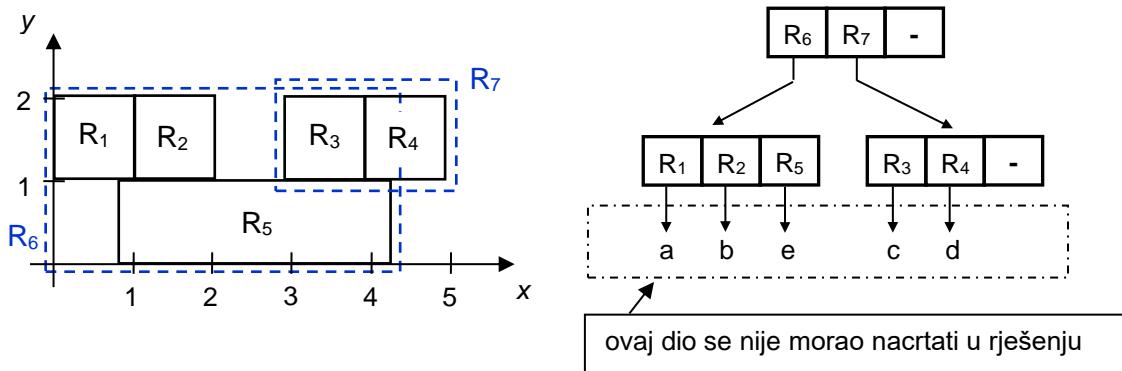
(5 bodova)

## Rješenja odabralih zadataka:

1.



2.



- b) Odrediti minimalni granični okvir (MBR) za krug  $k$  (npr.  $MBR_x$ ). U korijenu stabla pronaći sve MBR-ove koji se sijeku s  $MBR_x$ . To su  $R_6$  i  $R_7$ . Zatim slijediti pokazivače uz  $R_6$  i  $R_7$ . Uz  $R_6$  utvrditi koji su od MBR-ova  $R_1$ ,  $R_2$  i  $R_5$  u presjeku s  $MBR_x$ . To su  $R_2$  i  $R_5$ . Utvrditi jesu li objekti  $b$  ili  $e$  u presjeku s krugom  $k$  (ako neki od njih jest, dodati ga u rješenje). Uz  $R_7$  utvrditi koji su od MBR-ova  $R_3$  i  $R_4$  u presjeku s  $MBR_x$ . To je  $R_3$ . Utvrditi je li objekt  $c$  u presjeku s krugom  $k$  (ako jest, dodati ga u rješenje).

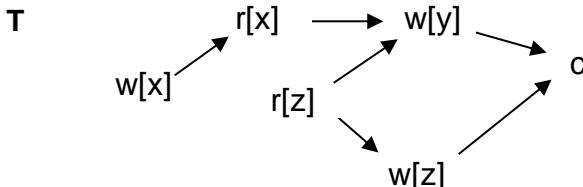
3. Broj n-torki

- međurezultat  $t_1 = \sigma_{A=100}(r)$
- selektivnost  $f(A=100, r) = N(\sigma_{A=100}(r)) / N(r) = 1 / 10\ 000 = 0.0001$
- $N(t_1) = N(r) \cdot f(A=100, r) = 10\ 000 \cdot 0.0001 = 1$
  
- međurezultat  $t_2 = \sigma_{D>700}(s)$
- selektivnost  $f(D>700, s) = 1 - N(\sigma_{D\leq 700}(s)) / N(s) = 1 - N(s) \cdot (700 - 600) / (1000 - 600) / N(s) = 0.75$
- $N(t_2) = N(s) \cdot f(D>700, s) = 40\ 000 \cdot 0.75 = 30\ 000$
  
- međurezultat  $t_3 = t_1 \times t_2$
- $N(t_3) = N(t_1) \cdot N(t_2) = 1 \cdot 30\ 000 = 30\ 000$
  
- konačni rezultat  $t_4 = t_3 \triangleright \triangleleft_{D=E} t$
- $E$  je ključ u  $t$ , stoga je  $N(t_4) = N(t_3) = 30\ 000$

Odabrani fizički operator, broj U/I operacija i veličina međurezultata

- međurezultat  $t_1 = \sigma_{A=100}(r)$
- index-scan jer postoji odgovarajući indeks i dohvaća se mali broj n-torki (samo jedna)
- broj UI operacija (pri izvršavanju operacije) = dubina indeksa + 1 = 5
- veličina materijaliziranog međurezultata  $B(t_1) = N(t_1) \cdot 0.2 = 1$  blok
  
- međurezultat  $t_2 = \sigma_{D>700}(s)$
- table-scan jer ne postoji indeks
- broj UI operacija (pri izvršavanju operacije) =  $B(s) = 20\ 000$
- veličina međurezultata  $B(t_2) = N(t_2) \cdot 0.3 = 30\ 000 \cdot 0.3 = 9\ 000$  blokova
  
- međurezultat  $t_3 = t_1 \times t_2$
- block nested-loop join jer jedna relacija sadrži tek jedan blok koji stane u prim. memoriju
- broj UI operacija (pri izvršavanju operacije) =  $B(t_1) + B(t_2) = 1 + 9\ 000 = 9\ 001$
- veličina međurezultata  $B(t_3) = N(t_3) \cdot (0.2 + 0.3) = 15\ 000$  blokova
  
- konačni rezultat  $t_4 = t_3 \triangleright \triangleleft_{D=E} t$
- hash-join ili block nested-loop join jer jedna relacija stane u prim. memoriju
- broj UI operacija (pri izvršavanju operacije) =  $B(t_3) + B(t) = 15\ 000 + 500 = 15\ 500$
- veličina konačnog rezultata  $B(t_4) = N(t_4) \cdot (0.2 + 0.1 + 0.3) = 18\ 000$  blokova

4. a)



b) Pogledati predavanja

5. a)

U trenutku kada je  $\langle \text{commit } T \rangle$  upisan u stabilnu memoriju, transakcija je dosegla točku potvrđivanja. Ako se nakon točke potvrđivanja dogodi kvar sustava ili kvar medija, a neka od output operacija transakcije se do tada nije obavila, sustav tijekom obnove u dnevniku neće moći naći novu vrijednost elementa s kojima bi izvršio nužno potrebnu operaciju redo, a kojom bi ponovio ono što je prije kvara trebala obaviti propuštena operacija output (jer u tehnici obnove UNDO u dnevnik se upisuju samo stare vrijednosti elemenata koji se mijenjaju).

b)

$\text{output}(x)$  se ne smije obaviti prije flush log zapisa dnevnika koji sadrži zapis commit te transakcije. Prema pravilu: bilo koja operacija  $\text{output}(x)$  transakcije  $T$  smije se obaviti tek nakon što su **svi** zapisi dnevnika povezani s tom transakcijom upisani u stabilnu memoriju

c)

nema takve operacije

6.

Pogledati predavanja

7.

Pogledati predavanja

# **Sustavi baza podataka**

Predavanja

**1. Arhitektura sustava**

ožujak 2023.

# **Sustavi baza podataka**

# Baza podataka

---

- BAZA PODATAKA je skup podataka koji su pohranjeni i organizirani tako da mogu zadovoljiti zahtjeve korisnika.  
*(M. Vetter, 1981.)*
- BAZA PODATAKA je skup perzistentnih podataka kojeg koriste aplikacijski sustavi neke organizacije.  
*(C. J. Date, 2000.)*
- BAZA PODATAKA je skup međusobno povezanih podataka, pohranjenih zajedno, uz isključenje bespotrebne zalihosti (redundancije), koji mogu zadovoljiti različite primjene. Podaci su pohranjeni na način neovisan o programima koji ih koriste. Prilikom dodavanja novih podataka, mijenjanja i pretraživanja postojećih podataka primjenjuje se zajednički i kontrolirani pristup. Podaci su strukturirani tako da služe kao osnova za razvoj budućih primjena.  
*(J. Martin, 1979.)*

# Sustav za upravljanje bazama podataka

---

- programski sustav koji upravlja istovremenim pristupom bazi podataka od strane više korisnika/aplikacija uz osiguravanje sigurnosti i integriteta baze podataka
  - složeni programski sustav - obuhvaća konglomerat znanja i desetljećima razvijanih koncepata i tehnologija
  - jedan od najranije razvijenih višekorisničkih sustava
    - utjecaj na aplikacijsku programsku potporu, operacijske sustave, mrežne servise
- relacijski sustav za upravljanje bazama podataka

# Zadaće sustava za upravljanje bazama podataka

---

- trajna pohrana podataka (*persistent storage*)
- osiguravanje programskog sučelja (*programming interface*)
  - DDL (Data Definition Language), DML (Data Manipulation Language)
- zaštita podataka
  - integritet podataka (*integrity*)
  - pristup podacima - autorizacija, sigurnost (*security*)
  - potpora za upravljanje transakcijama
    - upravljanje istodobnim pristupom (*concurrency control*)
    - obnova u slučaju razrušenja (*recovery*)
- optimiranje metoda pristupa podacima (*query optimization*)

# Sustav baza podataka (Database system)

---

- Sustav za upravljanje bazama podataka zajedno s bazom podataka (i aplikacijama) čini sustav baze podataka  
[Elmasri, 2000.]
- Sustav baze podataka je kolokvijalni naziv za sustav za upravljanje bazama podataka  
[Garcia-Molina, 2000.]

# Glavne zadaće administratora baze podataka

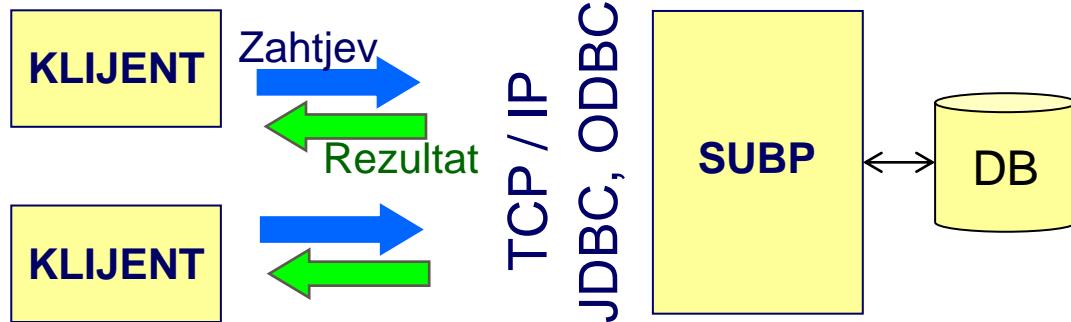
- instalacija i održavanje programske potpore koja se odnosi na sustav za upravljanje bazama podataka (*database engine + utilities*)
- održavanje sheme baze podataka
- nadzor pristupa podacima (autorizacija)
- planiranje i održavanje prostora za pohranu
  - sekundarna i tercijarna memorija
- nadzor performansi sustava
  - optimizacija upita
  - podešavanje parametara sustava (veličina međuspremnika, veličina i broj logičkih dnevnika, veličina tablice ključeva, alokacija procesora, metoda detekcije potpunih zastoja, ...)
- priprema procedura za obnovu podataka (i obnova podataka po potrebi)
  - upravljanje arhivskim kopijama i logičkim dnevnicima
- upravljanje i nadzor nad procesima replikacije i distribucije podataka
- komunikacija s korisnicima (aplikacijski programer i sofisticirani korisnici) i administratorom operacijskog sustava

# **Arhitektura sustava za upravljanje bazama podataka**

SUBP kao komponenta drugih sustava

# Klijent-poslužitelj arhitektura

- *Client-server (C/S)*



- sustav obuhvaća dvije komponente: klijent i poslužitelj
- koncept "zahtjev-odgovor" ("request-response"):
  - klijent postavlja zahtjev - poslužitelj odgovara
- komunikacija se odvija preko dobro definiranih sučelja: npr. TCP/IP i standardna sučelja namjenskih programa (Application Program Interface - API): ODBC (Open Database Connectivity), JDBC ("Java Database Connectivity"), OLE/DB (Object Linking and Embedding/Database), itd.

# Klijent-poslužitelj arhitektura i distribuirani sustavi

---

- postupak raslojavanja (distribucija) sustava - razdioba sustava na slojeve, od kojih svaki sadrži dio funkcionalnosti ukupnog sustava
- okruženje je tipično heterogeno
  - različito sklopolje (*hardware*) i programska oprema, često isporučena od strane različitih proizvođača
- slojevi imaju fundamentalno različite potrebe za računalnim resursima (brzina procesora, memorija, brzina i kapacitet diskova, U/I karakteristike)
  - npr. zahtjevi za sklopoljem i programskom potporom različiti su kod poslužitelja baze podataka i grafičke stanice

# Klijent-poslužitelj arhitektura

- tri sloja i njihovi podslojevi
  - različitim načinom grupiranja slojeva/podslojeva u komponente dobivaju se različiti tipovi arhitektura

Sloj korisničkog sučelja

Prezentacijski sloj

Sloj upravljanja dijalogom

Sloj obrade podataka

Sloj aplikacijske jezgre

Sloj upravljanja podacima

Sloj pristupa podacima

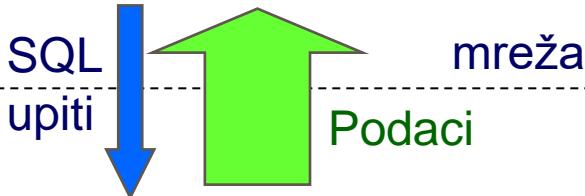
Sloj pohrane podataka

# Dvoslojna (*two-tiered*) klijent-poslužitelj arhitektura oslonjena na klijenta (*client-centric*) - *fat client, thick client*

## 1. komponenta:

sloj korisničkog sučelja +  
sloj obrade podataka

**KLIJENT**



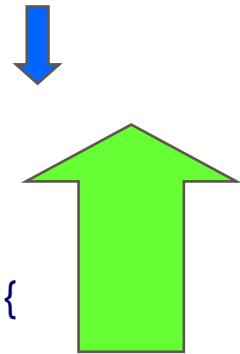
**POSLUŽITELJ**

## 2. komponenta:

sloj upravljanja podacima

*Client program:*

```
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(
    "SELECT * FROM racun");
int suma = 0;
while(rs.next()) {
    int brRac = rs.getInt(1);
    int iznos = rs.getInt(2);
    if (provjeraKontrolneZnamenke(brRac)) {
        suma = suma + iznos;
    }
}
...
ustmt = conn.prepareStatement(
    "INSERT INTO zbirno VALUES(?, ?)");
ustmt.setDate(1, "4.3.2004");
ustmt.setInt(2, suma);
ustmt.executeUpdate();
...
```



## Dvoslojna (*two-tiered*) klijent-poslužitelj arhitektura oslonjena na klijenta (*client-centric*) - *fat client, thick client*

---

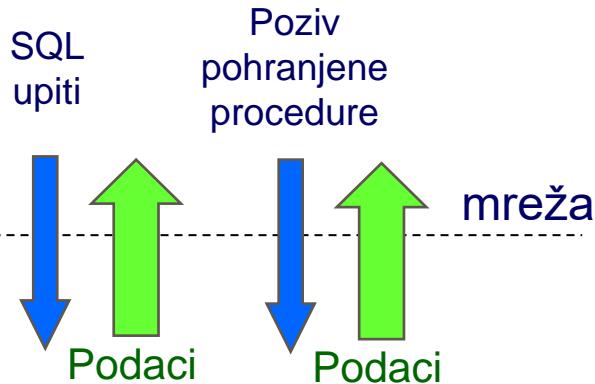
- korisničko sučelje (najčešće *GUI-graphical user interface*) usko je povezano s elementima za obradu podataka koji se u cijelosti implementiraju na klijentu
- Nedostaci:
  - *fat client sindrom* - budući da se obrada podataka obavlja na klijentu, klijent mora biti relativno snažno računalo
    - povećani troškovi opreme
    - degradacija performansi komunikacijske mreže, jer se podaci radi obrade moraju dobaviti na računalo-klijent
    - povećani troškovi održavanja programske potpore (promjene elemenata obrade podataka moraju se implementirati na svim računalima-klijentima)

# Dvoslojna (*two-tiered*) klijent-poslužitelj arhitektura oslonjena na poslužitelja (*server-centric*) - *thin client*

## 1. komponenta:

sloj korisničkog sučelja

**KLIJENT**



## 2. komponenta:

sloj za obradu podataka +  
sloj upravljanja podacima

*Client program:*

```
...
Statement stmt = conn.createStatement();
stmt.executeUpdate(
    "EXECUTE PROCEDURE proc1()");
...
```

*Stored procedure:*

```
PROCEDURE proc1 ()
DEFINE suma INTEGER; ...
LET suma = 0;
FOREACH
    SELECT brRac, iznos INTO
        pBrRac, plznos FROM racun
    IF (provjeraKontrolneZnamenke(pBrRac)) THEN
        LET suma = suma + iznos;
    END IF
END FOREACH
INSERT INTO zbirno VALUES(TODAY, suma);
END PROCEDURE
```

# Dvoslojna klijent-poslužitelj arhitektura oslonjena na poslužitelja (server-centric)

---

- Nedostaci:
  - poslužitelj može postati usko grlo - *bottle-neck*
  - najčešće se koristi poseban jezik za komponentu aplikacije na strani klijenta, poseban za komponentu aplikacije na strani poslužitelja
  - dio elemenata obrade podataka se ipak mora implementirati na klijentu, jer jezici koji se (za npr. pohranjene procedure) koriste na poslužitelju često nemaju sve mogućnosti koje imaju jezici koji se koriste u klijentskom sloju.

# Troslojna (*three-tiered*) klijent-poslužitelj arhitektura

**1. komponenta:**  
sloj korisničkog sučelja



**2. komponenta:**  
sloj za obradu podataka + sloj pristupa podacima



**3. komponenta:**  
sloj pohrane podataka



- primjer troslojne arhitekture:
  - klijent je zadužen za prezentaciju i upravljanje dijalogom: PC računalo s bogatim grafičkim korisničkim sučeljem (GUI) pisan u nekom prikladnom programskom jeziku (C#, C++, Java ...)
  - aplikacijski poslužitelj implementira sloj za obradu podataka i odgovoran je za upravljanje transakcijama, upravljanje opterećenjem sustava, sigurnost, itd. (BEA Tuxedo, IBM WebSphere Application server, Oracle Application Server, ...)
  - poslužitelj baze podataka je neki od sustava za upravljanje bazom podataka (Oracle, IBM DB2, IBM IDS, SQL Server...)

# Višeslojna (*multi-tiered*) klijent-poslužitelj arhitektura

1. komponenta:  
dio prezentacijskog  
sloja



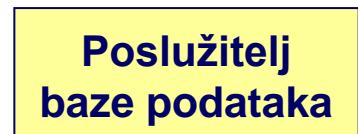
2. komponenta:  
dio prezentacijskog  
sloja + sloj upravljanja  
dijalogom



3. komponenta:  
sloj za obradu  
podataka + sloj  
pristupa podacima



4. komponenta:  
sloj pohrane  
podataka

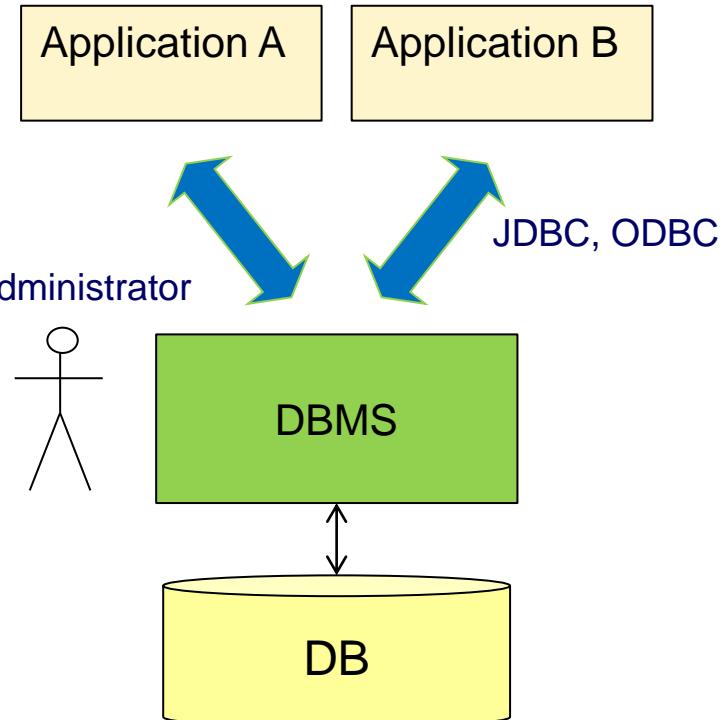


- primjer višeslojne arhitekture:
  - klijent je internet preglednik (*browser*) koji prezentira HTML stranice (ili mobilni uređaj, ...)
  - dio srednjeg sloja za upravljanje dijalogom je web poslužitelj podržan nekom od web tehnologija (JSP-Java ServerPages, ASP-Active Server Pages, ...)
  - dio srednjeg sloja s poslovним pravilima je aplikacijski poslužitelj koji podržava neku od tehnologija (EJB-Enterprise JavaBeans, CORBA-Common Object Request Broker Architecture, ...)
  - poslužitelj baze podataka je neki od sustava za upravljanje bazom podataka (Oracle, IBM DB2, ...)

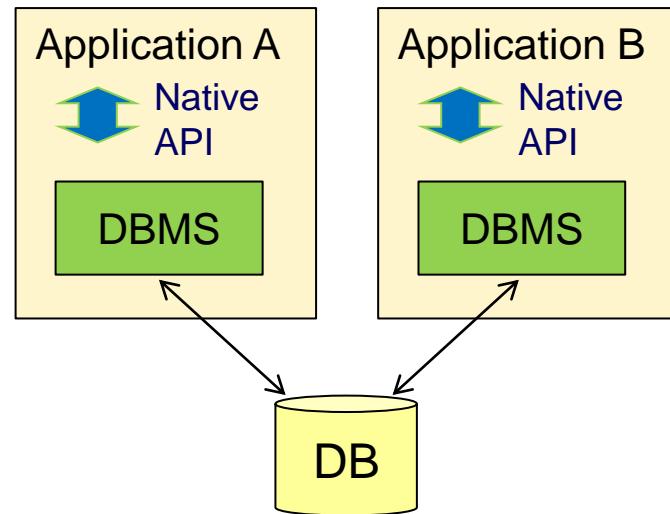
# Ugrađeni sustavi za upravljanje bazama podataka

- *Embedded database management system*
  - Sustav za upravljanje bazama podataka integriran je u aplikaciju (*software product*)

"Enterprise DBMS"



"Embedded DBMS"



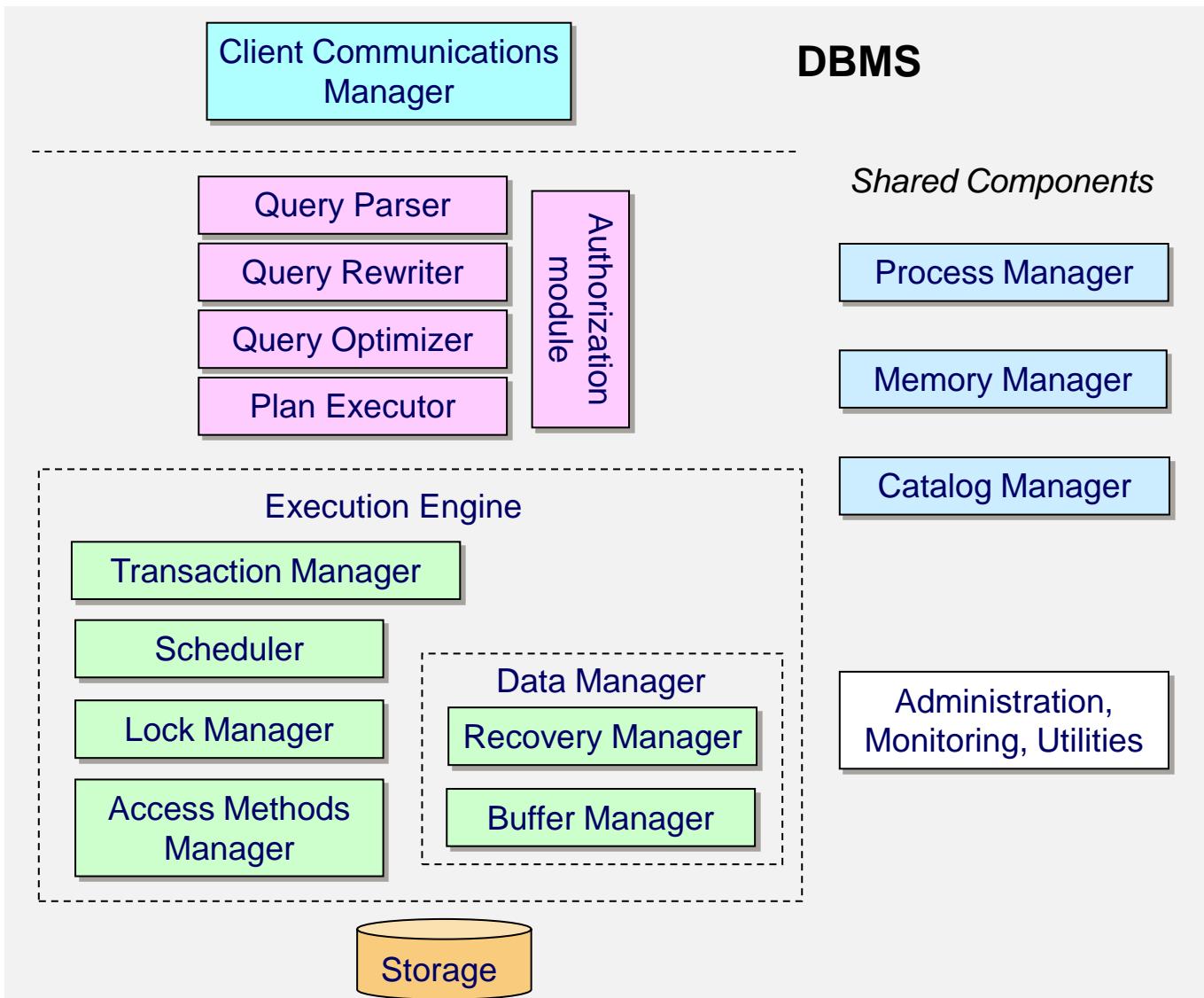
SUBP ugrađen u aplikaciju

- samoupravljav (self-managed DBMS)
- nezahtjevan prema resursima (small footprint)
- modularan (componentized DBMS)

## **Komponente SUBP-a**

# Komponente SUBP-a

*"The Life of a Transaction"*



# Komponente SUBP-a

---

- "The Life of a Transaction"

## Uspostava komunikacije s korisnikom

- klijentska aplikacija (namjenski program) putem sučelja otvara korisničku sjednicu (*user session*) s menadžerom komunikacija (*Client Communications Manager*) čije su zadaće:
  - utvrditi sigurnosni status korisnika (npr. *login, password*)
  - uspostaviti i održavati vezu (pamtiti stanje veze) prema klijentskoj aplikaciji
  - prihvaćati zahtjeve, vraćati rezultate (podatke, kontrolne poruke)
- menadžer procesa (*Process Manager*) pokreće *dretvu izvršavanja* (*thread of execution*) koja će biti zadužena za obrađivanje zahtjeva vezanih uz dotičnu korisničku sjednicu
  - nazivi u literaturi: *server process* ili *DBMS worker*
  - za svaku korisničku sjednicu postoji jedan *DBMS worker*

# Komponente SUBP-a

---

- *"The Life of a Transaction"* - nastavak

## Prevođenje i optimizacija

- operacije transakcije (npr. SQL upiti) predaje se parseru (*Query Parser*) i modulu za pojednostavljivanje i normalizaciju upita (*Query Rewriter*)
- rezultat optimiranja upita u modulu za optimizaciju (*Query Optimizer*) je interni plan izvršavanja (*query plan*) kojim je određen "optimalni" redoslijed izvršavanja tipičnih operacija (spajanje, selekcija, projekcija, agregacija, sortiranje, itd.)
- interni plan izvršavanja predaje se izvršitelju plana (*Plan Executor*) koji implementira operacije spajanja, selekcije, itd. koristeći usluge modula na nižim razinama
- modul za autorizaciju (*Authorization Module*) može u svakoj od navedenih faza prevođenja i optimizacije biti konzultiran u vezi dozvola za izvršavanje pojedinih dijelova upita

# Komponente SUBP-a

---

- "The Life of a Transaction" - nastavak

## Izvršavanje

- modul za izvršavanje (*Execution Engine*) izvršava *transakcije*:
  - menadžer transakcija (*Transaction Manager*) objedinjuje pojedinačne zahtjeve (upite) u transakcije na temelju transakcijskih naredbi. Pojedinačne database operacije (*read, write*) i transakcijske (*transactional*) operacije (*start, abort, commit*) upućuje u menadžer redoslijeda izvršavanja
  - menadžer redoslijeda izvršavanja (*Scheduler*) uz pomoć menadžera zaključavanja (*Locking Manager*) određuje redoslijed izvršavanja operacija kojim se uz "*istodobno*" izvršavanje transakcija neće narušiti konzistentnost baze podataka
  - menadžer pristupa (*Access Manager*) koristi prikladne algoritme i strukture podataka za organiziranje i pristupanje podacima u sekundarnoj memoriji
  - menadžer međuspremnika (*Buffer Manager*) ima važnu ulogu osiguravanja efikasnog prijenosa podataka između primarne i sekundarne memorije
  - menadžer obnove (*Recovery Manager*) osigurava mogućnost obnove baze podataka u slučaju pogreške (kvara)

# Komponente SUBP-a

---

- *"The Life of a Transaction"* - nastavak

## Rezultati

- modul za izvršavanje dobivene rezultate smješta u međuspremnike menadžera komunikacija koji ih proslijeđuje klijentskoj aplikaciji
  - ako se radi o relativno velikim skupovima podataka klijentska aplikacija može dohvaćati dijelove rezultata, što će rezultirati iterativnim aktiviranjem metoda u menadžeru komunikacija i modulu za izvršavanje

# Komponente SUBP-a

---

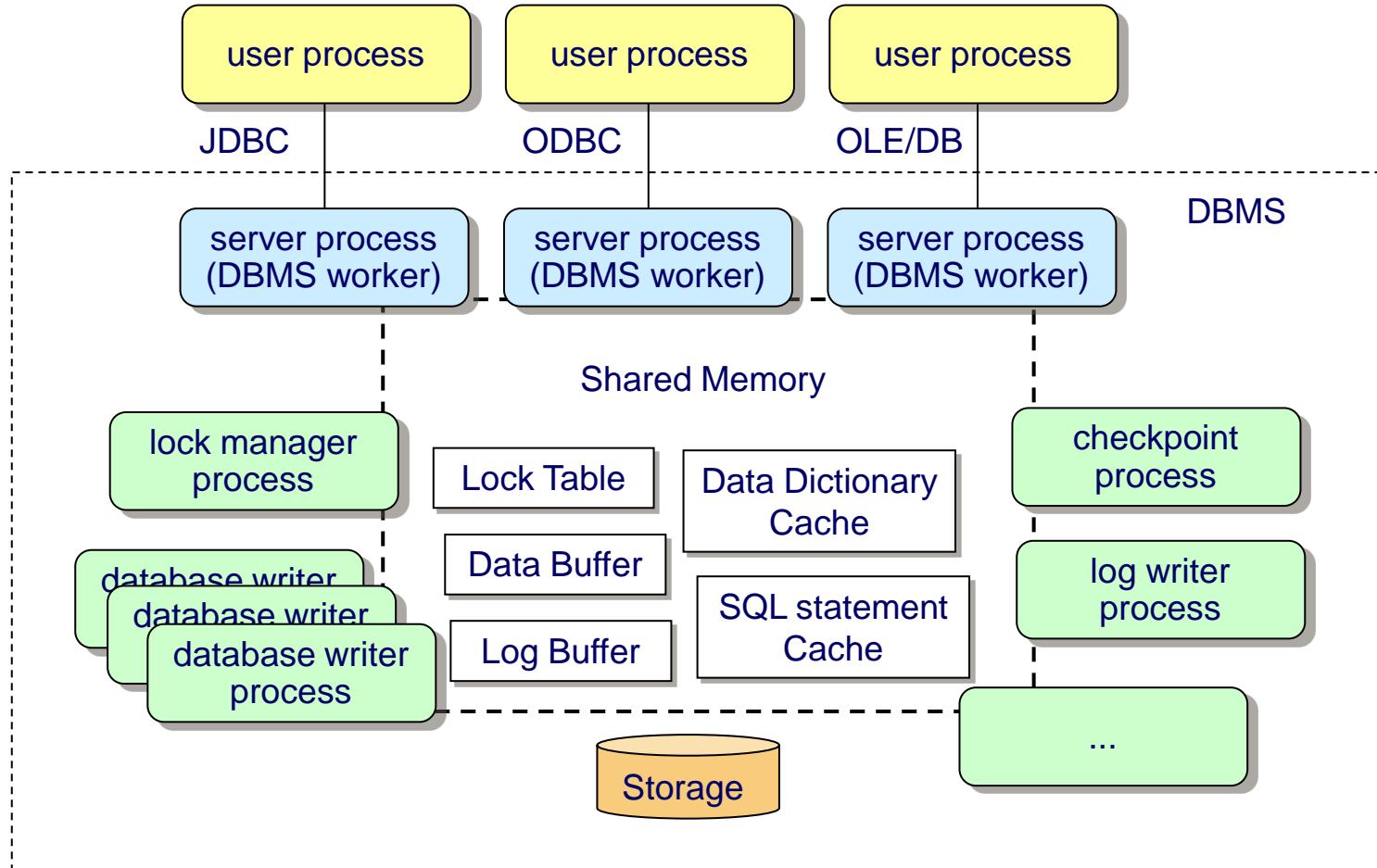
- "The Life of a Transaction" - nastavak

## Zajedničke komponente (*Shared Components*)

- zajedničke komponente koriste se kao servisi do sada opisanih modula
  - menadžer procesa (*Process Manager*) pokreće, zaustavlja i sinkronizira procese
  - menadžer memorije (*Memory Manager*) upravlja alokacijom i dealokacijom primarne memorije
  - menadžer kataloga (*Catalog Manager*) osigurava meta-podatke koji se koriste za autorizaciju, parsiranje i optimizaciju upita
- SUBP obuhvaća i niz pomoćnih (ali važnih) modula (*Administration, Monitoring, Utilities*) za administraciju i nadgledanje sustava, npr.
  - modul za upravljanje arhiviranjem i obnovom
  - modul za promjenu konfiguracijskih parametara
  - modul za nadzor i upravljanje prostorom u postojanoj memoriji
  - modul za nadzor i upravljanje procesima
  - itd.

# Procesi u SUBP-u

- Tipični sustav obuhvaća veliki broj procesa koji pristupaju podacima u dijeljenoj (*shared*) memoriji



# Procesi u SUBP-u

- *User process*
  - proces klijentske aplikacije (nije SUBP proces)
- *Server process (DBMS Worker)*
  - zaprima zahtjeve klijentskih aplikacija i vraća rezultate
  - upravlja korisničkom sjednicom (*user session*)
    - kontekst u kojem jedan korisnik obavlja niz SQL naredbi putem jedne veze (*SQL-Connection*) prema sustavu za upravljanje bazama podataka
- *Lock manager process*
  - implementira funkcionalnost menadžera zaključavanja
- *Database writer process*
  - jedan ili više procesa obavljaju prijenos podataka između primarne i sekundarne memorije
- *Log writer process*
  - upravlja zapisivanjem dnevnika u stabilnu memoriju
- *Checkpoint process*
  - periodički obavlja proces zapisivanja kontrolne točke

# Organizacija procesa u SUBP-u

- ***OS process per DBMS thread of execution***
- organizacije procesa u kojoj se svakoj dretvi izvršavanja (thread of execution, DBMS Worker, Server process) pridružuje jedan proces na razini operacijskog sustava (OS proces)
  - tip organizacije procesa korišten u ranijim implementacijama sustava
  - relativno jednostavna implementacija
    - koriste se mehanizmi operacijskog sustava
      - upravljanje procesima (*process switching*)
      - upravljanje dijeljenom memorijom, itd.
    - upitna skalabilnost (1000 korisnika → 1000 procesa)
      - povećanjem broja korisničkih sjednica raste broj OS procesa

# Organizacija procesa u SUBP-u

- danas sustavi u principu koriste organizaciju procesa u kojoj se dretva izvršavanja izvršava kao jedna od dretvi (*thread, lightweight process*) u okviru jednog OS procesa
  - jedan OS proces obavlja više dretvi
  - dretve dijele zajednički adresni prostor
  - trajanje zamjene konteksta dretve je znatno kraće od zamjene konteksta procesa
  - visoki stupanj skalabilnosti
- ***OS thread per DBMS thread of execution***
  - koristi se postojeći (opći) OS API za upravljanje dretvama
- ***DBMS thread per DBMS thread of execution***
  - SUBP koristi vlastiti (specijalizirani) API za upravljanje dretvama

# **Mediji za pohranu podataka**

# Hijerarhija medija za pohranu podataka

- Priručna memorija (*cache*)
  - Glavna memorija (*main memory*)
  - Sekundarna memorija  
(online)
  - Tercijarna memorija  
(offline)
- primarna memorija
- RAM
- DRAM
- Flash-SSD, DRAM-SSD
- magnetski disk (HDD)
- optički medij (CD, DVD)
- magnetska traka
- Tape silo/jukebox* sustavi ( $10^6$  TB)
- 
- ```
graph TD; A[cache] --- B[main memory]; B --- C[secondary memory]; C --- D[tertiary memory]; B --- E[primarna memorija]; E --- F[RAM]; E --- G[DRAM];
```

# Karakteristike medija za pohranu podataka

---

- Priručna memorija (*cache*)
  - tipično SRAM
  - memorija u (ili pri) središnjoj procesorskoj jedinici (CPU), vrlo velike brzine pristupa podacima (~10 ns), vrlo skupa, vrlo ograničenog kapaciteta (~ MB)
- Glavna memorija (*main memory*)
  - tipično DRAM
  - velika brzina pristupa podacima (10-100 ns), relativno skupa, kapacitet (~ GB)
  - neprikladnost za (trajnu) pohranu baze podataka:
    - nepostojanost (gubitak napajanja, pogreška sustava)
    - cijena
    - kapacitet

# Karakteristike medija za pohranu podataka

- Sekundarna memorija
  - postojana, sporija, većeg kapaciteta u odnosu na primarnu memoriju
- Magnetski disk (HDD)
  - kapacitet: do 4TB (uz brzi napredak tijekom vremena)
  - osjetljivost na mehaničke udare ( $\sim 300\text{g}$ )
  - vrijeme pristupa (*seek time + rotational latency*): 5-15 ms (uz spor napredak tijekom vremena)
  - brzina prijenosa podataka: 50-300 MB/s
    - *disk-to-buffer, buffer-to-computer*
    - manja za podatke s unutarnjih staza



# Karakteristike medija za pohranu podataka

---

- karakteristike medija za pohranu podataka (postojanost, kapacitet, brzina pristupa podacima, cijena) uvjetuju da se većina današnjih baza podataka pohranjuje na magnetskim diskovima
- Sekundarna memorija (nastavak)
  - Flash-SSD
    - manji utrošak energije: ~ HDD/3
    - vrijeme pristupa podacima: ~ 0.1ms
    - brzina prijenosa podataka: ~ HDD
    - otpornost na vanjske utjecaje (-60/+90°C, otpornost na udarce ~ 1500g)
    - nedostaci: cijena, ograničeni broj ciklusa pisanja ( $3 \times 10^5$  -  $10^6$ )

# Karakteristike medija za pohranu podataka

---

- Tercijarna memorija
  - sporija, većeg kapaciteta
    - za izradu sigurnosnih kopija (*backup*) ili za vrlo velike baze podataka
  - Optički mediji (CD, DVD)
    - kapacitet do ~ 17 GB
    - vrijeme pristupa podacima ~100 ms
  - Magnetske trake
    - kapacitet 40 GB - 2 TB
  - *Jukebox* ili *tape silo* (robotizirani sustavi)
    - za pohranu vrlo velikih baza podataka (100 TB - 1000 PB)
    - robotizirani sustav omogućava korištenje većeg broja jedinica medija (trake, CD, ...) u kombinaciji s manjim brojem uređaja za čitanje/pisanje
    - vrijeme pristupa: od nekoliko sekundi do nekoliko minuta

# Karakteristike medija za pohranu podataka

- u kontekstu sustava za upravljanje bazama podataka treba razlikovati
  - **nepostojana (*volatile*) memorija**
    - sadržaj memorije se gubi u slučaju *pogreške sustava* (kvar napajanja, pogreška memorije, operacijskog sustava, ...)
    - SRAM, DRAM
  - **postojana (*non-volatile*) memorija**
    - sadržaj memorije ostaje sačuvan u slučaju pogreške sustava
    - međutim, pogreške medija (npr. *disk crash*) će ipak izazvati gubitak sadržaja
    - HDD, Flash-SSD, DRAM-SSD, traka, DVD, ...
  - **stabilna (*stable*) memorija**
    - sadržaj memorije se *nikad\** ne gubi

\* garancija za "nikad" ne postoji, ali moguće je postići razumno razinu zaštite

# Stabilna memorija

---

- implementacija: replikacijom podataka na medijima s postojanom memorijom
- najčešće: diskovi organizirani u RAID sustave - *on-line* stabilna memorija
  - *hot-pluggable, hot-spare*: eliminira se potreba za zaustavljanjem sustava u slučaju kvara medija
  - RAID ne garantira "stabilnost" memorije npr. u slučaju požara
- korištenje trake, CD, DVD (napraviti kopiju i pohraniti medij u drugom gradu)
  - pogodno za arhivske kopije (*backup*), ali ne omogućava kontinuiranu zaštitu sadržaja: *off-line* stabilna memorija
- korištenje SAN (*Storage Area Network*) arhitekture
  - polje RAID organiziranih diskova fizički odvojenih od poslužitelja
    - povezivanje SCSI ili Fibre Channel sučeljem

# Organizacija podataka u sekundarnoj memoriji

- logički zapisi (n-torke) pohranjuju se u blokove (fizičke zapise, stranice) sekundarne memorije
  - block, physical record, page*
  - veličina bloka ovisi o sklopoljtu, operacijskom sustavu, primijenjenom tipu datotečnog sustava (tipično: 512 B, 1 kB, 2 kB, 4 kB, 8 kB)
    - u jedan blok najčešće je smješteno više logičkih zapisa (npr. n-torki relacije)

Primjer:

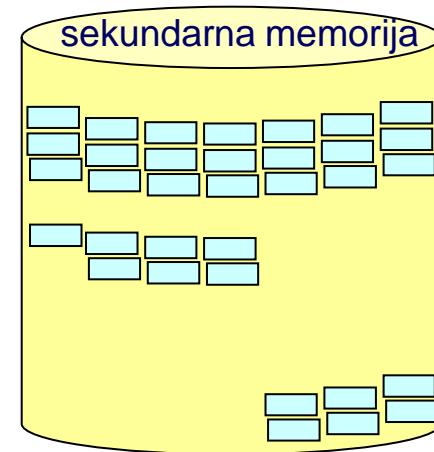
| mbr     | ime  | prez   |
|---------|------|--------|
| 1999999 | Ivan | Novak  |
| 1000315 | Ana  | Horvat |
| ...     |      |        |
| 1999998 | Jura | Kolar  |
| 1000001 | Tea  | Ban    |

56 B

$10^6$  n-torki

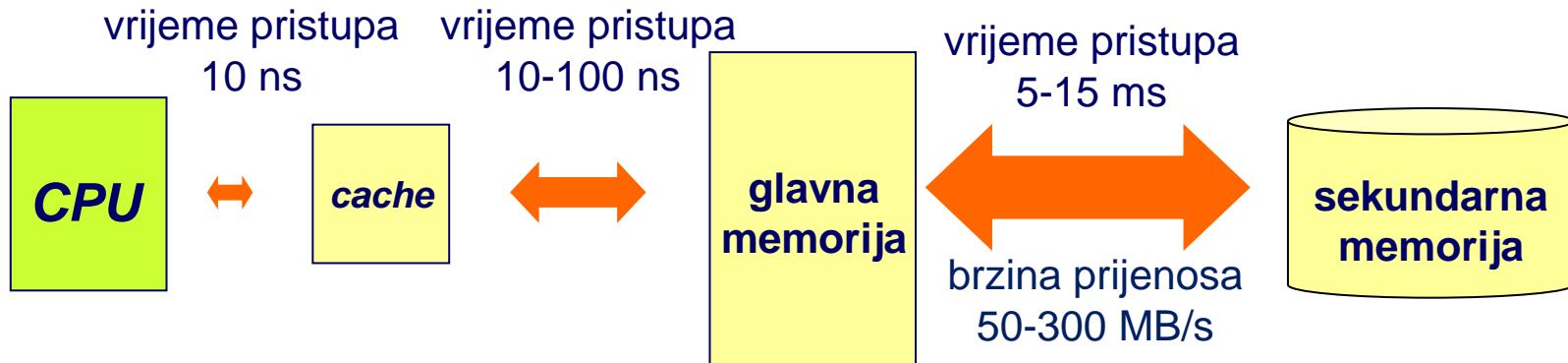
⇒ uz pretpostavku  
veličine bloka 1kB

⇒ broj potrebnih  
blokova u  
sekundarnoj  
memoriji: najmanje  
**54688**



# Pristup podacima u sekundarnoj memoriji

- koliko je podatak "daleko od procesora"



- operacije čitanje/pisanje
  - read, write*
- U/I - ulazno/izlazne operacije
  - obavljaju se u jedinicama blokova
  - input, output (I/O)*
  - disk-read, disk-write*
- zbog velike razlike u brzini (~ ns : ~ ms)
  - ⇒ **dominiraju troškovi U/I operacija**

# Utjecaj broja U/I operacija na performanse sustava

- ukupno vrijeme potrebno za prijenos blokova iz sekundarne u primarnu memoriju najviše ovisi o sljedećim karakteristikama medija
  - *seek time*
  - *rotational delay (latency)*
  - *transfer time*
- znatan utjecaj ima stupanj fragmentacije (koliko su blokovi koji se prenose raspršeni po cilindrima i sektorima na disku)

## Primjer:

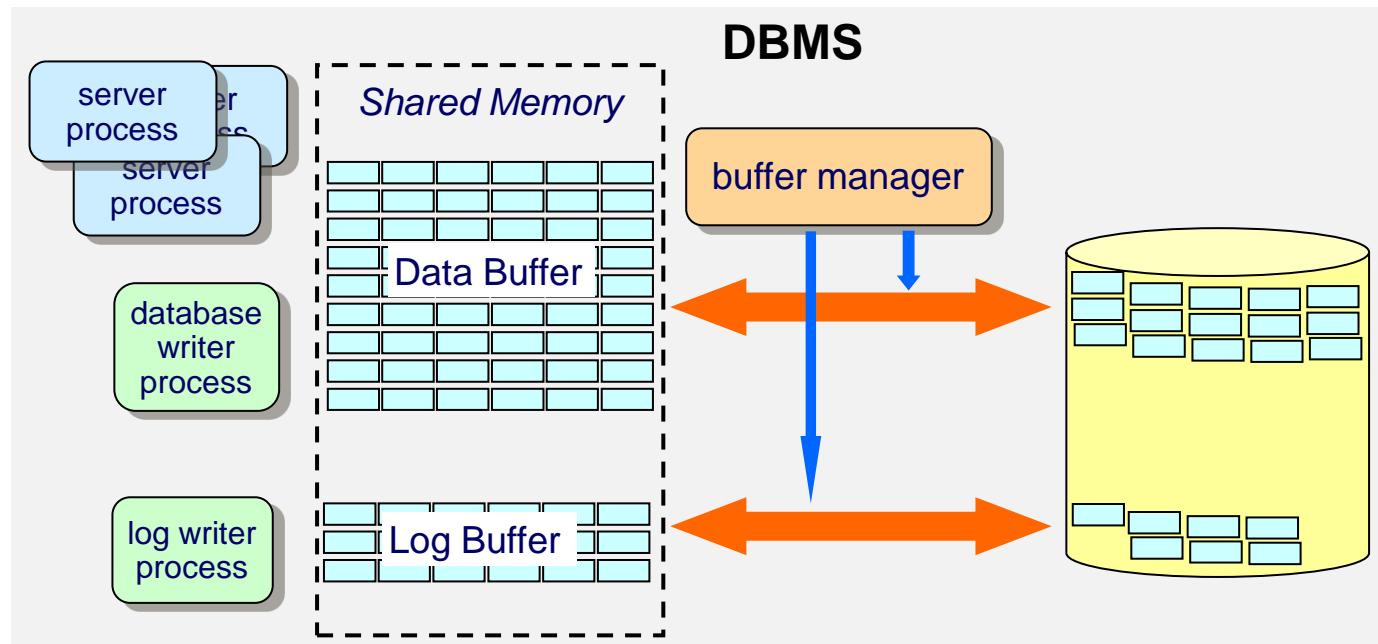
- neka se nad podacima iz prethodnog primjera obavlja upit

```
SELECT * FROM osoba  
WHERE mbr BETWEEN 1222000 AND 1222300;
```

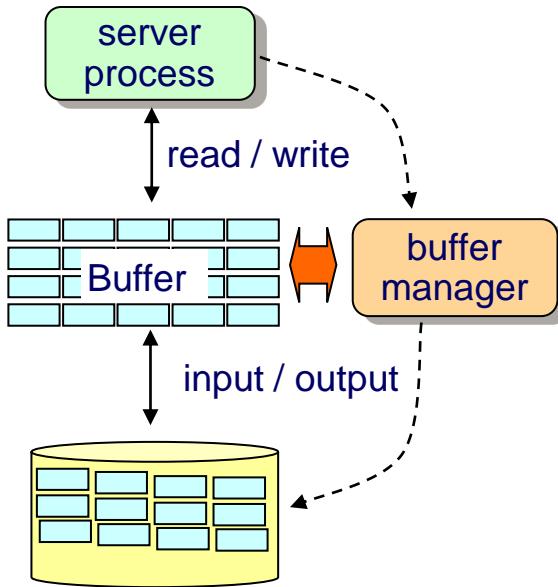
- uz pretpostavku korištenja slijednog pretraživanja, svaki od 54688 blokova mora se prenijeti u glavnu memoriju i pretražiti
  - za prijenos podataka očekuje se utrošak vremena mјeren sekundama
  - za pretraživanje svih blokova u glavnoj memoriji očekuje se utrošak vremena mјeren milisekundama

# Organizacija podataka u glavnoj memoriji

- međuspremnik (*buffer* ili *buffer pool*)
  - raspoloživa primarna memorija partitionirana je u blokove koji svojom veličinom u pravilu odgovaraju blokovima u sekundarnoj memoriji
- komponente SUBP-a koriste sekundarnu memoriju posredstvom menadžera međuspremnika

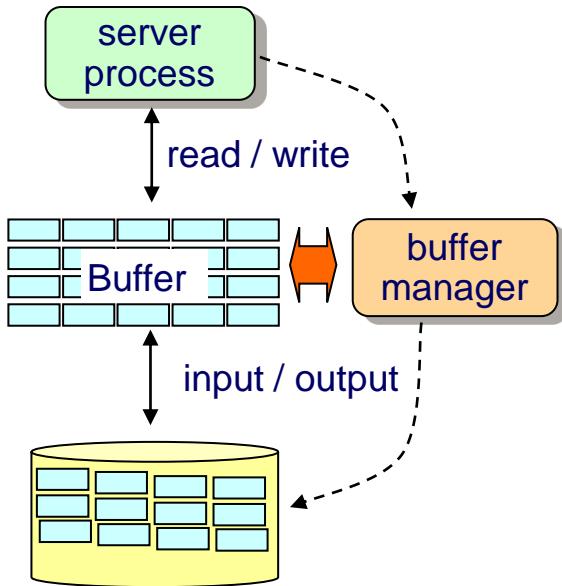


# Uloga menadžera međuspremnika (za operaciju read)



- proces koji treba obaviti operaciju čitanja postavlja *read* zahtjev menadžeru međuspremnika
- ako se traženi blok već nalazi u međuspremniku, obavlja se operacija read
- inače, menadžer prethodno mora obaviti operaciju *input* kojom se traženi blok iz sekundarne memorije kopira u međuspremnik
  - iskorišteni blok može ostati u međuspremniku
- blokove je poželjno čim dulje zadržati u međuspremniku, međutim u međuspremniku ne mogu ostati zauvijek zbog ograničenog kapaciteta međuspremnika

# Uloga menadžera međuspremnika (za operaciju write)



- proces koji treba obaviti operaciju pisanja postavlja *write* zahtjev menadžeru međuspremnika
- ako se traženi blok već nalazi u međuspremniku, obavlja se operacija *write*
- inače, menadžer prethodno mora obaviti operaciju *input* kojom se traženi blok iz sekundarne memorije kopira u međuspremnik
  - iskorišteni blok može ostati u međuspremniku, ali prije ili kasnije morat će se obaviti operacija *output*, svakako prije trenutka u kojem će se taj blok (*dirty page*) izbaciti iz međuspremnika
  - dakle, operacija *output* se ne mora (uvijek) obaviti odmah nakon *write*
    - *vidjet ćemo u poglavlju o obnovi baze podataka*

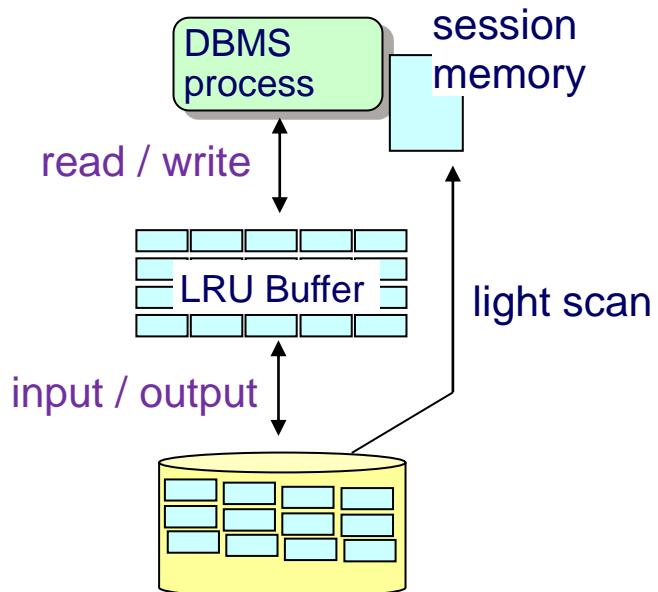
# Strategije zamjene stranica

- algoritam koji određuje koje blokove (i kada) izbaciti iz međuspremnika (*buffer-replacement policy*) ima veliki utjecaj na efikasnost sustava
  - Least-Recently Used (LRU) - nisu korišteni tijekom najduljeg vremena
  - Most-Recently Used (MRU) - proteklo najmanje vremena od zadnjeg korištenja
  - First-In-First-Out (FIFO)
  - Clock Algorithm
- u današnjim sustavima najčešće se koristi LRU algoritam\*
  - pretpostavka: blok kojeg je neki proces koristio nedavno (*recently*) vjerojatno će uskoro opet koristiti isti ili neki drugi proces
  - iz međuspremnika izbacivati (po potrebi uz operaciju *output*) blokove koje tijekom najduljeg perioda nije koristio niti jedan proces
  - menadžer međuspremnika mora za svaki blok u međuspremniku evidentirati zadnji trenutak u kojem je neki proces koristio taj blok

\* u nekim slučajevima se operacija *output* mora obaviti "odmah", bez obzira na LRU algoritam  
→ u poglavlju o obnovi baze podataka

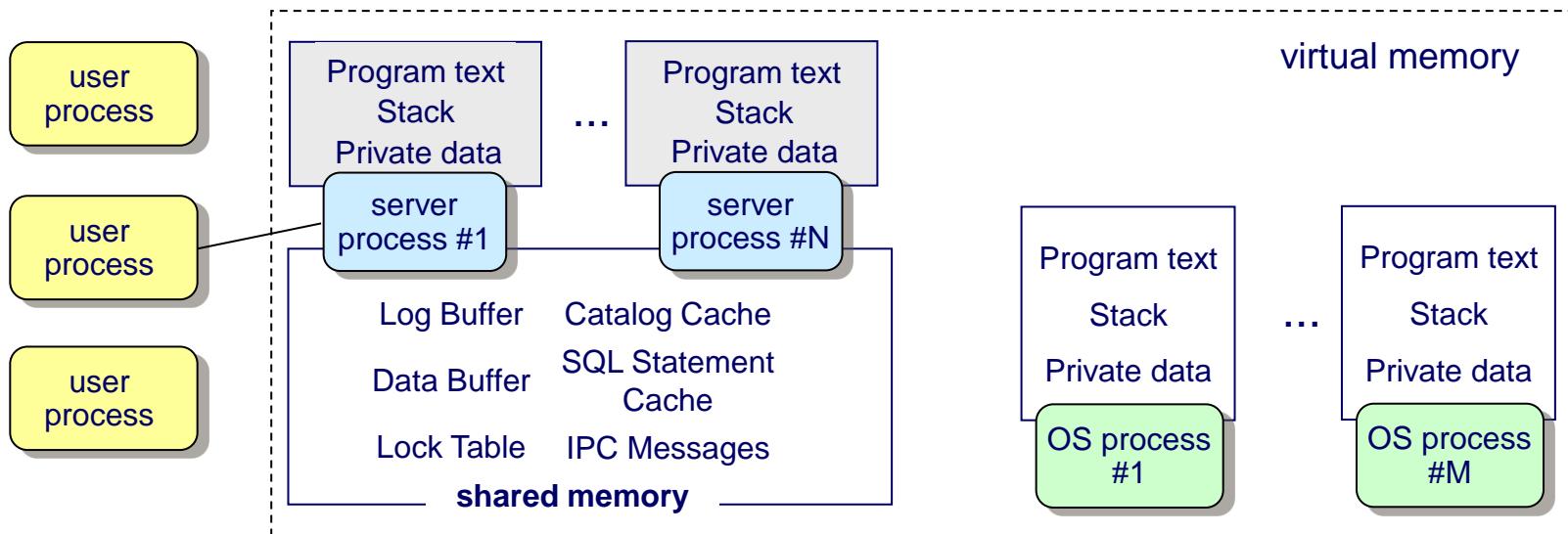
# LRU - kada se ne koristi?

- slijedno čitanje relacije čija je veličina sumjerljiva ili veća od raspoložive veličine međuspremnika. Npr. čitanje relacija od 4 500 blokova, uz raspoloživu veličinu međuspremnika od 5 000 blokova
  - posljedica: pročitane stranice (koje se u međuspremniku trebaju nalaziti tek kratko vrijeme) nepotrebno uzrokuje izbacivanje svih drugih, vjerojatno često korištenih stranica
  - Oracle rješenje: koristi uobičajeni (LRU) međuspremnik, ali se tom operacijom pročitane stranice smjesta označavaju kao *least recently used* stranice, odnosno kao stranice koje dugo vremena nisu korištene
  - IBM Informix rješenje: *light scans*: međuspremnik se zaobilazi i podaci se iz sekundarne memorije kopiraju izravno u memoriju koja je u tu svrhu alocirana za korisničku sjednicu u kojoj se izvršava ta operacija



# Uloga dijeljene memorije (*shared memory*)

- dijeljena memorija je mehanizam operacijskog sustava koji iz perspektive SUBP-a omogućava:
  - smanjenje broja U/I operacija
    - upravljanje međuspremnicima na temelju zajedničkih potreba svih procesa, umjesto na temelju svakog procesa pojedinačno
  - visoku brzinu komunikacije među procesima i dretvama (IPC - *inter-process communication*)
    - razmjena poruka među procesima se obavlja brzinom prijenosa u memoriji, nasuprot sporijim mehanizmima, npr. cjevovodima (*pipes*)



# Utjecaj mehanizama operacijskog sustava

---

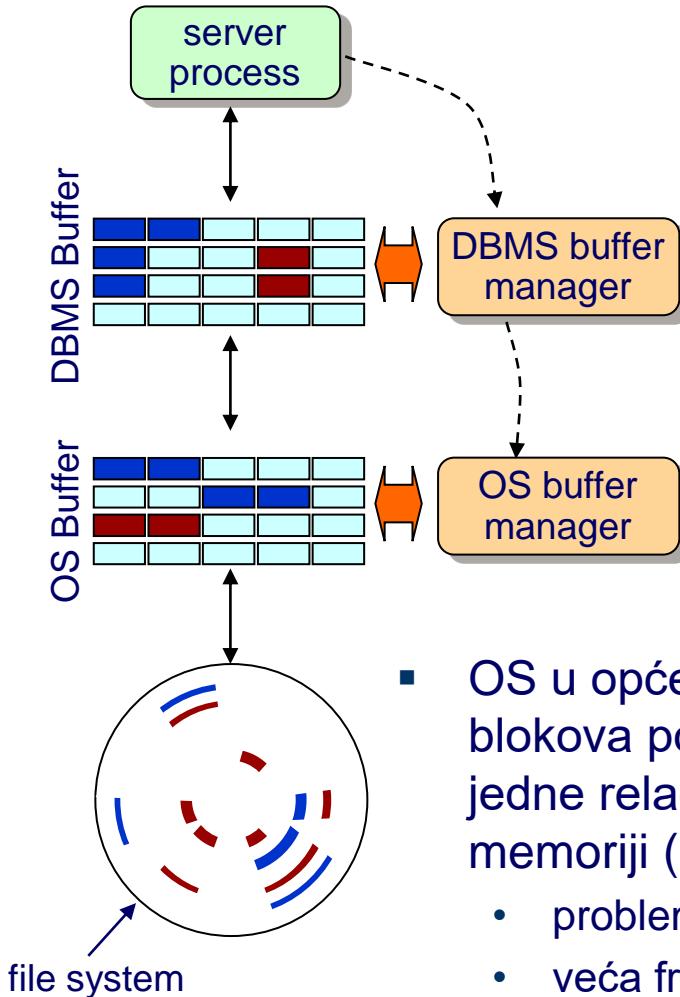
- operacijski sustav koristi vlastite mehanizme za organizaciju podataka i prijenos podataka između radnog spremnika i sekundarne memorije
  - podaci su u sekundarnoj memoriji organizirani u datotečni sustav (*file system*)
  - prenose se u blokovima, uz korištenje međuspremnika operacijskog sustava (često LRU algoritam) s ciljem poboljšanja performansi U/I operacija
- za SUBP koji za pohranu podataka u sekundarnoj memoriji koristi datotečni sustav i procedure operacijskog sustava (*system calls*) kaže se da koristi cooked files ili buffered files
- problemi:
  - korištenje dvostrukih međuspremnika, *double buffering*
  - moguće je da će logički kontinuirani podaci biti smješteni u fizički diskontinuiranom prostoru (fragmentacija relacije u fizičkom prostoru)

# **Double buffering**

---

- budući da SUBP upravlja međuspremnicima optimalno u odnosu na operacije koje on obavlja, međuspremni operacijskog sustava su u kombinaciji s međuspremnicima SUBP-a redundantni i štetni.
- Pored toga
  - troše primarnu memoriju koja se mogla bolje iskoristiti
  - troše procesorsko vrijeme na nepotrebno kopiranje podataka
    - pisanje
      - međuspremnik SUBP → međuspremnik OS → sekundarna memorija
    - čitanje
      - sekundarna memorija → međuspremnik OS → međuspremnik SUBP

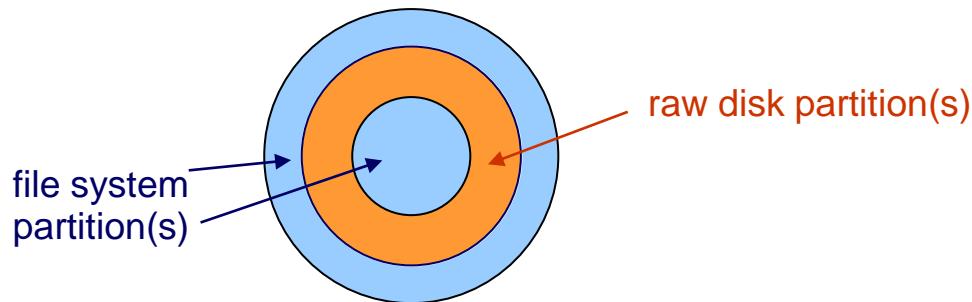
# Cooked files



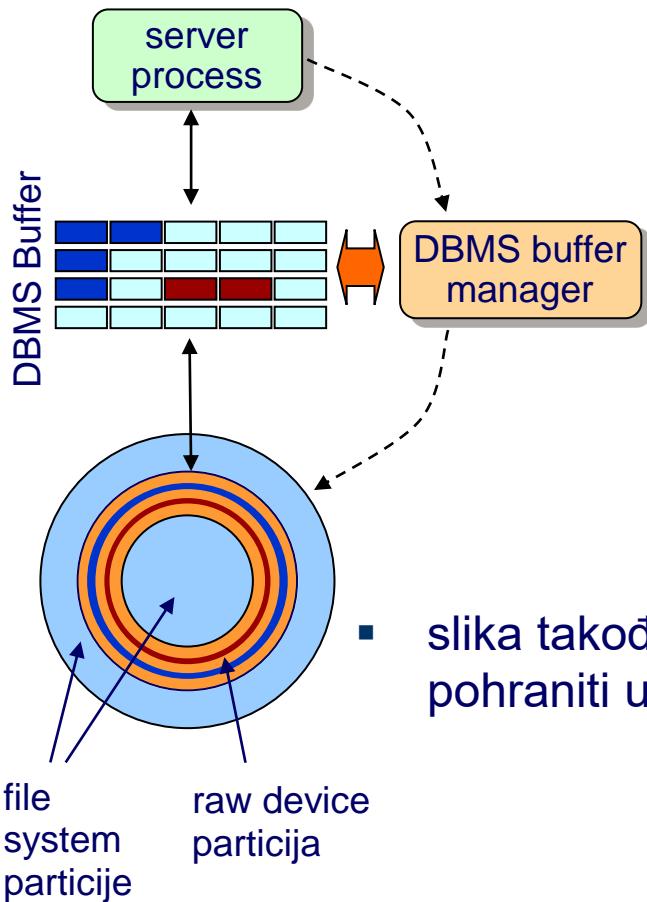
- korištenje međukoraka: disk  $\Leftrightarrow$  OS  
međuspremniči  $\Leftrightarrow$  SUBP međuspremniči  
 $\Rightarrow$  lošije performanse
- trošenje resursa s vrlo malim ili nikakvim  
dobitkom, korištenje procedura opće  
namjene (operacijski sustav) za visoko  
specijalizirane zadaće koje obavlja SUBP  
 $\Rightarrow$  lošije performanse
- OS u općem slučaju ne osigurava pohranu logički kontinuiranih  
blokova podataka (npr. blokova podataka koji sadrže n-torce  
jedne relacije) u fizički kontinuirani prostor u sekundarnoj  
memoriji (npr. u nekoliko susjednih traka)
  - problem: *seek time, rotational latency*
  - veća fragmentacija  $\Rightarrow$  sporiji pristup podacima

# Raw disk I/O

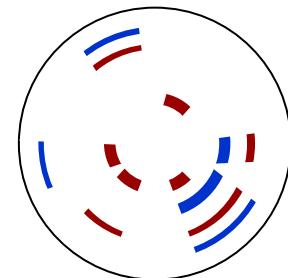
- termin *raw disk I/O* odnosi se na način pristupa podacima u sekundarnoj memoriji: podacima se pristupa direktno, umjesto putem datotečnog sustava. Operacijski sustav omogućuje *raw disk I/O* putem posebnog sučelja
    - *raw disk device*
  - korištenjem *raw disk I/O* SUBP
1. zaobilazi mehanizme međuspremnika i straničenja (*paging*) operacijskog sustava
    - SUBP u potpunosti upravlja procesima prijenosa podataka
      - npr. blokovi će se u sekundarnu memoriju zapisivati točno onda kada za to postoji potreba iz perspektive SUBP-a
  2. particija sekundarne memorije za koju se koristi *raw disk device* predstavlja fizički kontinuirani prostor



# Raw disk I/O



- nema međukoraka: disk  $\Leftrightarrow$  OS  
međuspremniči  $\Leftrightarrow$  SUBP međuspremniči  
 $\Rightarrow$  bolje performanse
- logički kontinuirani blokovi podataka  
pohranjuju se u fizički kontinuiranom  
prostoru  $\Rightarrow$  bolje performanse
- moguće poboljšanje performansi 10-30%
- slika također ilustrira kako se n-torce jedne relacije mogu  
pohraniti u fizički kontinuiranom prostoru
- za razliku od pohrane u  
moguće diskontinuiranom  
fizičkom prostoru u  
datotečnom sustavu



# Direct I/O

---

- danas se mehanizam raw disk nešto rjeđe koristi jer operacijski sustavi i pomoću mehanizma **direct I/O** omogućuju korištenje direktnog pristupa memoriji u datotečnom sustavu (tj. zaobilaženje međuspremnika operacijskog sustava)
- performanse sustava za upravljanje bazama podataka kada koristi mehanizam **direct I/O** uglavnom su sumjerljive performansama sustava koji koristi mehanizam *raw disk*

# Upravljanje prostorom za pohranu u SUBP-u

- različiti SUBP omogućuju različite razine mogućnosti upravljanja prostorom za pohranu
- ovdje su na primjeru sustava IBM Informix prikazani temeljni koncepti

## Fizičke jedinice pohrane

- jedinice koje predstavljaju segmente sekundarne memorije gledano iz perspektive operacijskog sustava
  - grumen (*chunk*)
  - područje (*extent*)

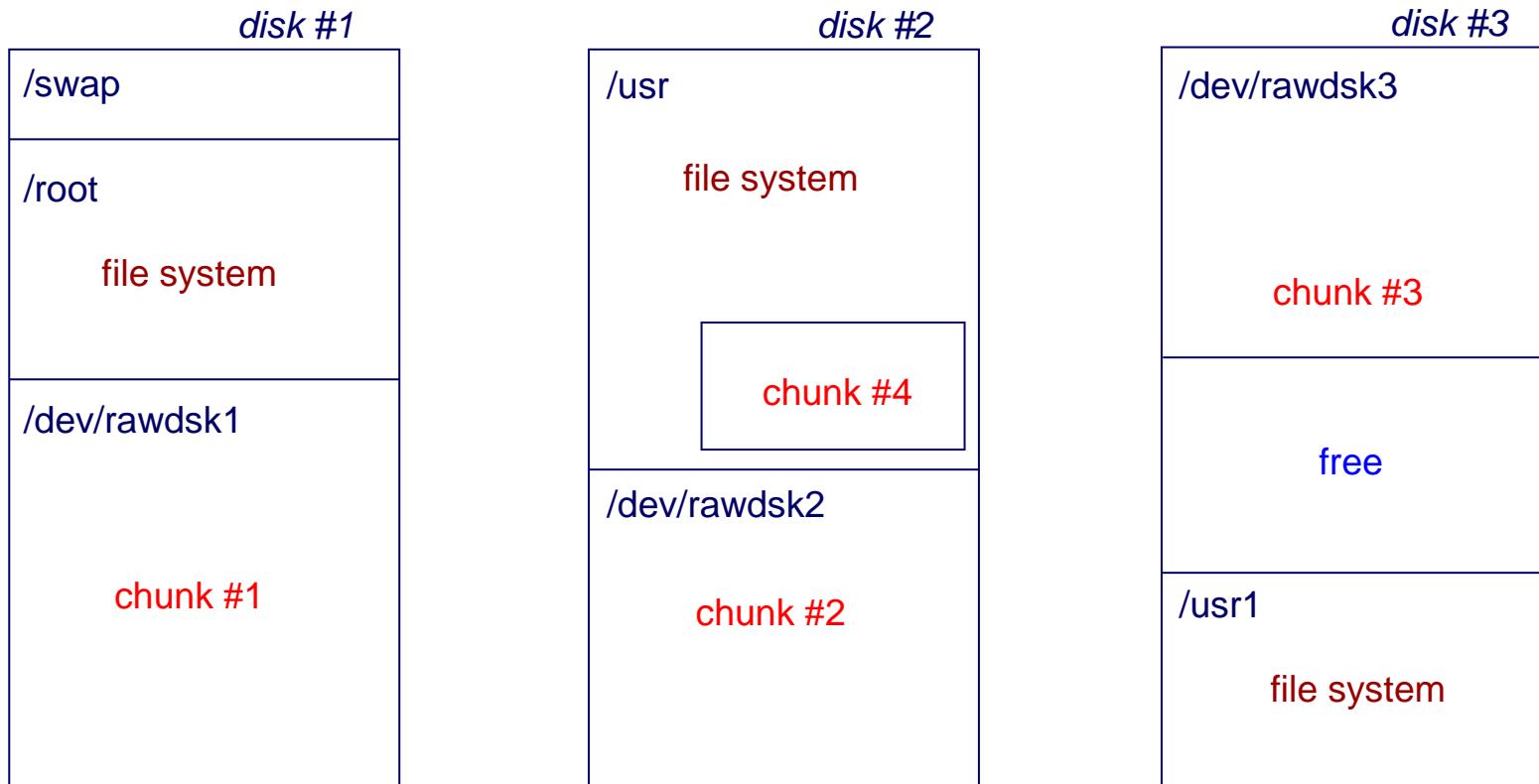
## Logičke jedinice pohrane

- jedinice koje predstavljaju logički povezane dijelove baze podataka (iz perspektive sustava za upravljanje bazama podataka). Npr.
  - prostor baze podataka (*database space*)
- logičke jedinice pohrane se na razini SUBP-a mogu povezati s fizičkim jedinicama pohrane. Na taj način administrator SUBP-a može utjecati na fizički smještaj logičkih elemenata baze podataka

## Grumen (*chunk*)

- grumen je najveća jedinica fizičkog prostora koja se s razine operacijskog sustava dodjeljuje sustavu za upravljanje bazama podataka. Predstavlja unaprijed alocirani prostor, u *raw device* obliku ili u obliku datoteke u datotečnom sustavu (*cooked file*).
- grumene definira administrator operacijskog sustava u dogovoru s administratorom sustava baza podataka (uobičajeno prilikom instalacije operacijskog sustava ili ugradnje novog diska)
  - particije diska koje će se koristiti kao *raw device*
  - datoteke koje će se koristiti kao *cooked file*

Primjer:



- fizički kontinuirani prostor u grumenima 1, 2 i 3
- grumen 4 (datoteka) nije nužno fizički kontinuirani prostor
- dodavanje novih grumena je moguće samo na slobodnim particijama, na diskovima koji se naknadno dodaju ili u datotečnom sustavu u formi datoteka

# Područje (extent)

## Područje (extent)

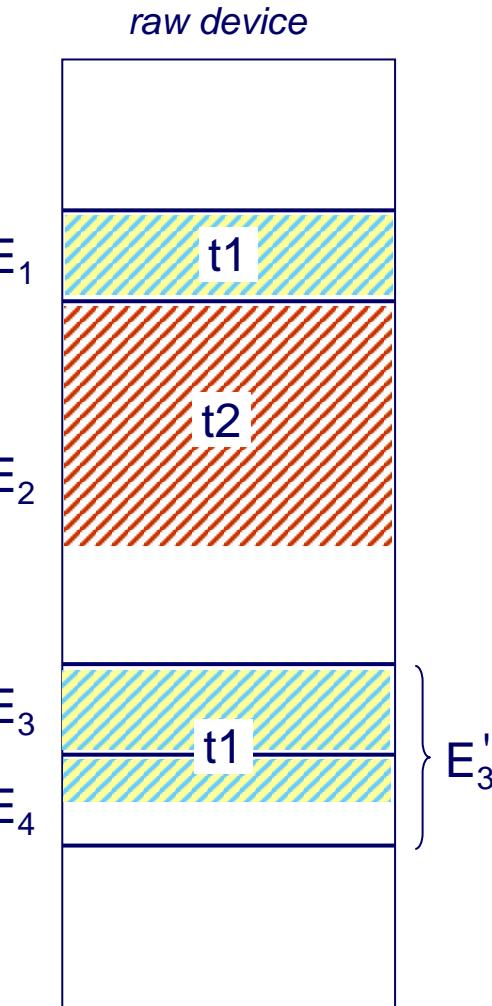
- SUBP nastoji zadržati podatke iste relacije u kontinuiranom prostoru sekundarne memorije
- u trenutku kreiranja relacije, SUBP rezervira fiksnu količinu prostora u sekundarnoj memoriji. Kada se taj inicijalno rezervirani prostor popuni, SUBP će alocirati sljedeći dio sekundarne memorije. Fizička jedinica prostora koja se rezervira u trenutku kreiranja relacije ili kasnije tijekom njenog proširenja, naziva se područje (*extent*)
- ako se koristi *raw device*, jedno područje je uvijek kontinuirani segment sekundarne memorije, inače, područje može biti više ili manje fragmentirano

```
CREATE TABLE ... (
    ...
) EXTENT SIZE m NEXT SIZE n;
```

- **m** i **n** su veličine područja izražene u kB
  - **m** veličina inicijalnog područja kod kreiranja relacije
  - **n** inkrementalna veličina dodatnog područja kod proširenja prostora relacije

# Područje (extent)

Primjer:



**CREATE TABLE t1 (...)**  
**EXTENT SIZE 16 NEXT SIZE 16**  
 $\Rightarrow$  alocira se  $E_1 = 16\text{k}$

**CREATE TABLE t2 (...)**  
**EXTENT SIZE 64 NEXT SIZE 32**  
 $\Rightarrow$  alocira se  $E_2 = 64\text{k}$

**INSERT INTO t1 (40 kB podataka)**  
 $\Rightarrow$  popunjava se  $E_1$  (16 kB)  
 $\Rightarrow$  alocira se  $E_3$  (16 kB)  
 $\Rightarrow$  popunjava se  $E_3$  (16 kB)  
 $\Rightarrow$  alocira se  $E_4$  (16 kB)  
 $\Rightarrow E_3$  i  $E_4$  se spajaju u  $E'_3$   
 $\Rightarrow$  popunjava se dio  $E'_3$  (8 kB)

**INSERT INTO t2 (40 k podataka)**  
 $\Rightarrow$  popunjava se dio  $E_2$  (40 kB)

# Područje (extent)

- **CILJ:** optimirati između
  - nepotrebnog zauzeća prostora
  - broja područja koja se pojedinačno alociraju
- ako broj fragmenata neke relacije postane prevelik, moguće je obaviti defragmentaciju

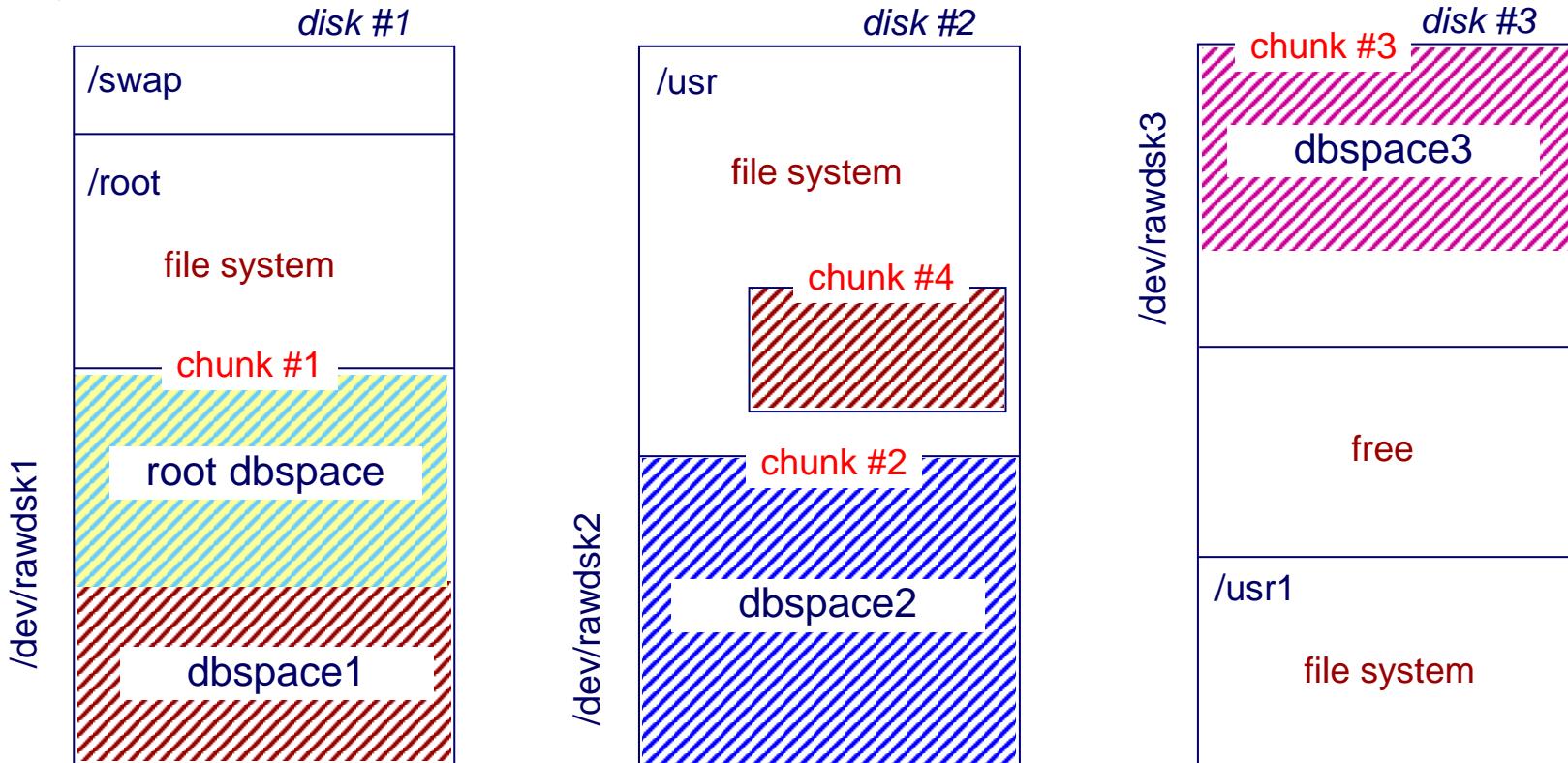
# Logičke jedinice pohrane

## Prostor baze podataka (*database space*)

- naziv, veličinu i smještaj prostora baze podataka određuje administrator SUBP-a
- prostor baze podataka formira se u jednom ili više dijelova jednog ili više grumena. Na taj način se najveća logička jedinica pohrane povezuje s najvećom fizičkom jedinicom pohrane
- administrator baze podataka će kasnije moći odabrati prostor baze podataka u kojem će kreirati objekt baze podataka (npr. relaciju). Na taj način indirektno će odrediti i fizički smještaj tog objekta

# Prostor baze podataka

Primjer:



- *root dbspace* je kreiran u dijelu grumena #1
- *dbspace1* je kreiran u dijelu grumena #1 i grumenu #4
- *dbspace2* je kreiran u grumenu #2
- *dbspace3* je kreiran u dijelu grumena #3

# Prostor baze podataka

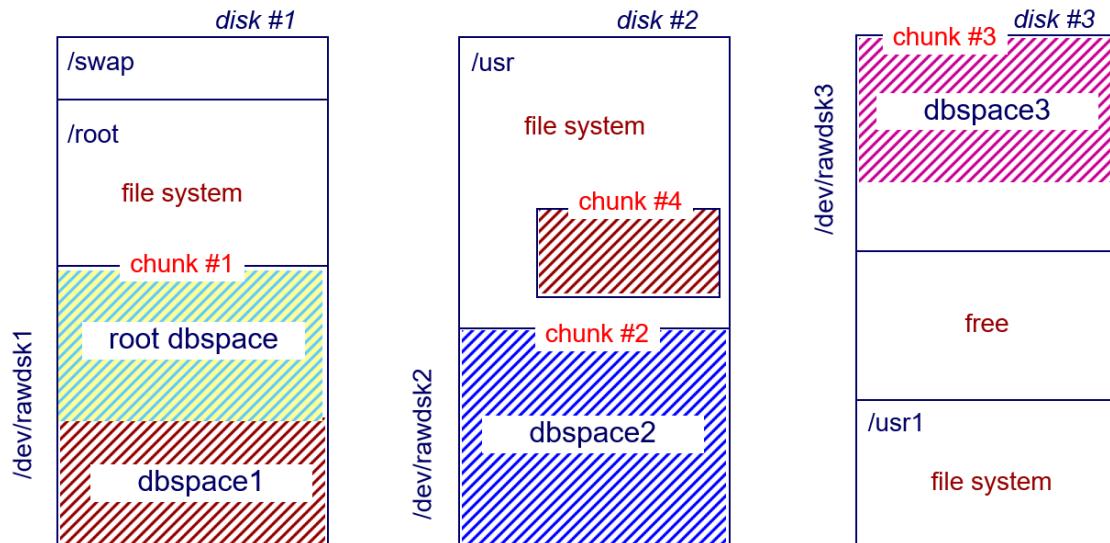
- administrator baze podataka može odrediti gdje će biti smještena baza podataka i pojedine relacije

`CREATE DATABASE dbName IN dbspace3;`

- baza podataka se pohranjuje u prostor *dbspace3*, a također i sve pripadne relacije za koje se CREATE TABLE naredbom ne specificira drugačije

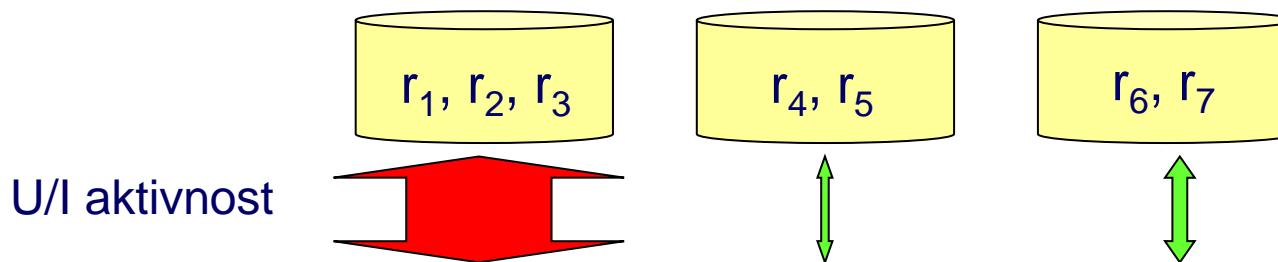
`CREATE TABLE tablename (...) IN dbspace2;`

- relacija se pohranjuje u prostor *dbspace2*, bez obzira na oblik pripadne CREATE DATABASE naredbe

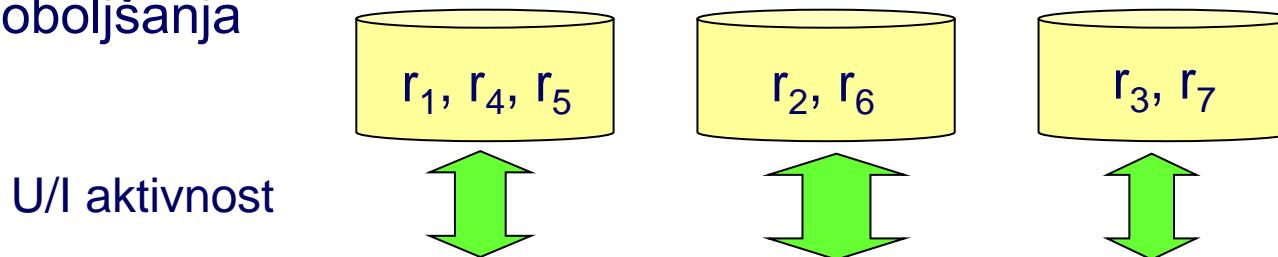


# Utjecaj fizičkog smještaja na performanse sustava

- U/I uređaji (*I/O devices*) često predstavljaju uska grla. Npr. neka su
  - relacije  $r_1, r_2, r_3$  relacije s visokom U/I aktivnošću
  - relacije  $r_4, r_5, r_6, r_7$  relacije s niskom U/I aktivnošću
- visoka razina natjecanja (*contention*) za pristup U/I uređajima rezultira pojavom "vrućih točaka", *hotspots*



- boljim planiranjem prostora za pohranu, mogu se postići velika poboljšanja



# Kako fizičkim smještajem podataka utjecati na performanse?

- nekoliko pravila kojima se treba poslužiti pri planiranju prostora za pohranu.  
Relacije s najvišom U/I aktivnošću:
  - smjestiti na zasebne U/I jedinice
  - ako postoje U/I jedinice različite brzine, smjestiti ih na najbrže U/I jedinice
  - smjestiti ih čim bliže središnjim cilindrima diska
    - kreirati *raw device* u blizini središnjih cilindara
    - nad njim kreirati grumen (*chunk*)
    - u njemu kreirati prostor baze podataka
    - u prostoru baze podataka kreirati relaciju
- kamo smjestiti jednu relaciju koja se među ostalima ističe visokom U/I aktivnošću?
  - fragmentacija (*fragmentation, partitioning*) omogućuje smještaj podataka jedne relacije na više U/I uređaja. Koriste se sljedeće sheme fragmentacije:
    - *round-robin*
    - *expression based*

# Shema fragmentacije relacije: *round-robin*

- n-torce se smještaju u fragmente na "slučajan" način. Pri tome se fragment određuje kao rezultat hash funkcije nad brojem dobivenim generatorom slučajnih brojeva ili se fragmenti ciklično izmjenjuju (jedna n-torka u prvi fragment, jedna u drugi fragment, ...). Postiže se jednolika razdioba n-torki.

```
CREATE TABLE ispit (
    jmbag      CHAR(10)
, sifPred   INTEGER
, datIspit DATE
, sifNast   INTEGER
, ocjena    SMALLINT
, PRIMARY KEY (jmbag
                , sifPred
                , datIspit)
    CONSTRAINT pkIspit
) FRAGMENT BY ROUND ROBIN
IN dbs1, dbs2, dbs3, dbs4;
```

```
SELECT sifPred, AVG(ocjena)
  FROM ispit
 WHERE YEAR(datIspit) = 2002
 GROUP BY sifPred;
```

- operacije prijenosa blokova (*input*) mogu se obavljati paralelno nad svim U/I uređajima u kojima je pohranjena relacija ispit
- shema je prikladna kada se izvršavaju upiti koji pristupaju relativno velikom broju n-torki

# Shema fragmentacije relacije: *expression based*

- n-torke se smještaju u fragmente na temelju predikata kojeg zadovoljavaju

```
CREATE TABLE student (
    jmbag      CHAR(10)
,  ime        CHAR(20)
,  prez       CHAR(20)
,  sifFakul  INTEGER
,  PRIMARY KEY (jmbag)
              CONSTRAINT pkStudent
) FRAGMENT BY EXPRESSION
  (sifFakul=36) IN dbs1
,  (sifFakul=120) IN dbs2
,  (sifFakul=210) IN dbs3
,  REMAINDER IN dbs4;
```

```
SELECT * FROM student
 WHERE prez LIKE 'Hor%'
   AND sifFakul=120;
```

- SUBP pristupa samo onim fragmentima koji sadrže podatke koji zadovoljavaju uvjet selekcije (ako uvjet sadrži kriterij iz sheme fragmentacije)
- shema je prikladna kada se često izvršavaju upiti čiji uvjeti za selekciju sadrže kriterij naveden u shemi fragmentacije

# Vertikalna fragmentacija relacija

- u relaciji se nalazi  $10^6$  n-torki

```
CREATE TABLE osoba (
    oib          CHAR(10)
, prez_ime    CHAR(86)
, visina      INTEGER
, zivotopis  CHAR(10000)
, PRIMARY KEY (oib)
    CONSTRAINT pkOsoba) ;
```

- u grubo procijeniti: koliko blokova veličine 2K je potrebno prenijeti (sekundarna→primarna memorija) pri obavljanju sljedećeg upita:

```
SELECT AVG(visina)
  FROM osoba;
```

$$\rightarrow 10^6 * 10100 / 2048 \approx 4.9 * 10^6 \text{ blokova}$$

# Vertikalna fragmentacija relacija

- vertikalna fragmentacija
  - stvara se nova relacija s primarnim ključem jednakim kao u originalnoj relaciji
  - u novu relaciju prebacuju se atributi koji zauzimaju veliki prostor, a u upitima se ne koriste često

```
CREATE TABLE osoba (
    oib        CHAR(10)
, prez_ime   CHAR(86)
, visina     INTEGER
, PRIMARY KEY (oib)
);
```

```
CREATE TABLE biogr (
    oib        CHAR(10)
, zivotopis CHAR(10000)
, PRIMARY KEY (oib)
, FOREIGN KEY ... osoba...
);
```

- u grubo procijeniti: koliko blokova veličine 2K je potrebno prenijeti (sekundarna→primarna memorija) pri obavljanju sljedećeg upita:

```
SELECT AVG(visina)
FROM osoba;
```

$$\rightarrow 10^6 * 100 / 2048 \approx 4.9 * 10^4 \text{ blokova}$$

# Literatura:

---

- H. Garcia-Molina, J. Ullman, J. Widom. Database system implementation. Prentice-Hall International, Inc. Upper Saddle River, New Jersey. 2000.
- A. Silberschatz, H. Korth, S. Sudarshan. Database system concepts, 6th ed. McGraw-Hill. 2010.
- R. Elmasri, S. Navathe. Fundamentals of database systems, 3rd ed. Addison-Wesley Publishing Company. Reading, Massachusetts. 2000.
- IBM Informix Dynamic Server Administrator's Reference, Version 11.1, IBM, 2007.
- IBM Informix Dynamic Server Administrator's Guide, Version 11.1, IBM, 2007.
- Oracle® Database Performance Tuning Guide, 11g Release 2 (11.2)
- Oracle® Database Concepts 11g Release 2 (11.2)

# **Sustavi baza podataka**

Predavanja

**2. Fizička organizacija**

ožujak 2023.

# Fizička organizacija

---

- fizička organizacija podataka ima vrlo veliki utjecaj na efikasnost (*efficiency*) sustava za upravljanje bazama podataka
- pojam fizičke organizacije podataka odnosi se na:
  - organizacija datoteke (*file organization*): strukture podataka primijenjene pri pohrani podataka u sekundarnoj memoriji
  - metode pristupa (*access methods*): postupci koji se primjenjuju pri obavljanju operacija nad podacima
- primjenjivost pojedinih metoda pristupa podacima ovisi o primijenjenim strukturama podataka
  - za pohranu sadržaja relacije nastojati primijeniti strukturu podataka koje će omogućiti efikasno obavljanje onih operacija koje se nad tom relacijom najčešće obavljaju
  - često je potreban kompromis

# Važniji ciljevi fizičke organizacije

---

- minimizirati broj U/I operacija pri pohrani i dohvatu podataka, minimizirati utrošak prostora za pohranu
  - u koji fizički blok pohraniti logički zapis odnosno n-torku
  - koje je dodatne informacije potrebno pohraniti da bi se omogućio efikasni pristup podacima
- omogućiti različite metode pristupa koje se koriste za pronalaženje bloka u kojem se nalazi traženi zapis (na temelju vrijednosti ključa pretrage)
  - ključ pretrage (*search key*) ne mora nužno biti primarni ili alternativni ključ. Ključ pretrage može biti bilo koji atribut ili skup atributa relacije ("sekundarni ključ").

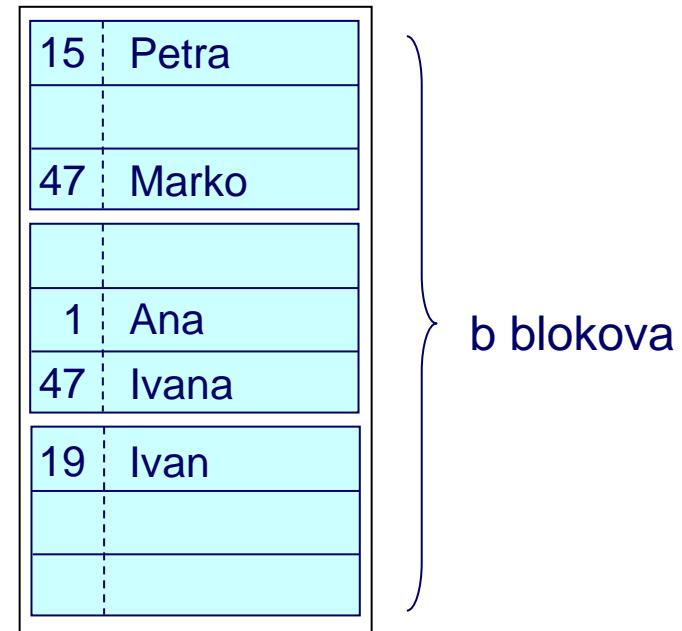
# Strukture podataka i metode pristupa podacima

---

- ne postoji "najbolja" metoda fizičke organizacije, ali neke od njih se u današnjim sustavima za upravljanje bazama podataka češće koriste
  - **Neporedana datoteka**
  - **B-stablo**
  - **Raspršena datoteka**
  - Poredana datoteka (*sorted file*)

# Neporedana datoteka

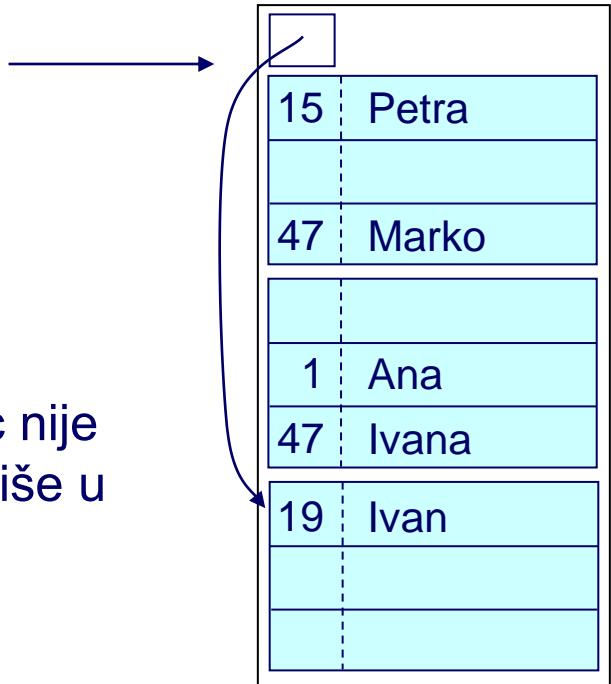
- *heap file, pile file, file of unordered records*
- pozicija zapisa u datoteci ovisi o redoslijedu unosa
- datoteka sadrži  $b$  blokova
- pristup podacima (dohvat podatka sa zadanim vrijednošću ključa pretrage) moguć je isključivo linearnim (slijednim) pretraživanjem (*linear search*)
- traženje zapisa sa zadanim ključem
  - ako se radi o primarnom ključu zapisa koji postoji tada je prosječan broj potrebnih U/I operacija  $b/2$
  - ako ključ potrage nije primarni ključ, broj obavljenih U/I operacija je  $b$



# Neporedana datoteka

u zaglavlju datoteke (*file header*) nalazi se adresa zadnjeg nepotpunjeneog bloka

- operacija unosa je vrlo efikasna: zadnji blok datoteke prenosi se u međuspremnik (ako već nije ondje), u blok se upiše novi zapis, blok se zapiše u sekundarnu memoriju (ne nužno odmah)
- brisanje zapisa je najčešće logičko (broj U/I operacija približno kao kod traženja zapisa)
- ova organizacija podataka se često koristi u kombinaciji s drugim organizacijama (npr. B<sup>+</sup>-stablo)

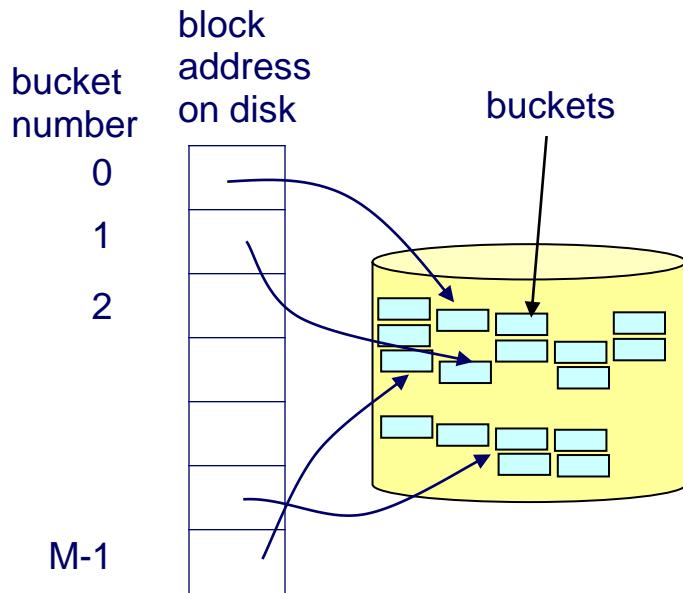


The diagram illustrates an unsorted file structure. On the left, a horizontal arrow points from the text "u zaglavlju datoteke (*file header*) nalazi se adresa zadnjeg nepotpunjeneog bloka" to a small rectangular box at the top left of a data block. The data block itself is a table with a light blue background and dark blue borders, divided into two columns by a vertical dashed line. It contains five records, each consisting of a number and a name:

|    |       |
|----|-------|
| 15 | Petra |
| 47 | Marko |
| 1  | Ana   |
| 47 | Ivana |
| 19 | Ivan  |

# Raspršena datoteka

- *hash files*: vrlo slično raspršenoj organizaciji u primarnoj memoriji
  - *external hashing, secondary storage hashing*
- pretinac (*bucket*): jedan blok ili više uzastopnih blokova u sekundarnoj memoriji
- *hash-funkcijom* se iz ključa izračunava relativna adresa pretinca

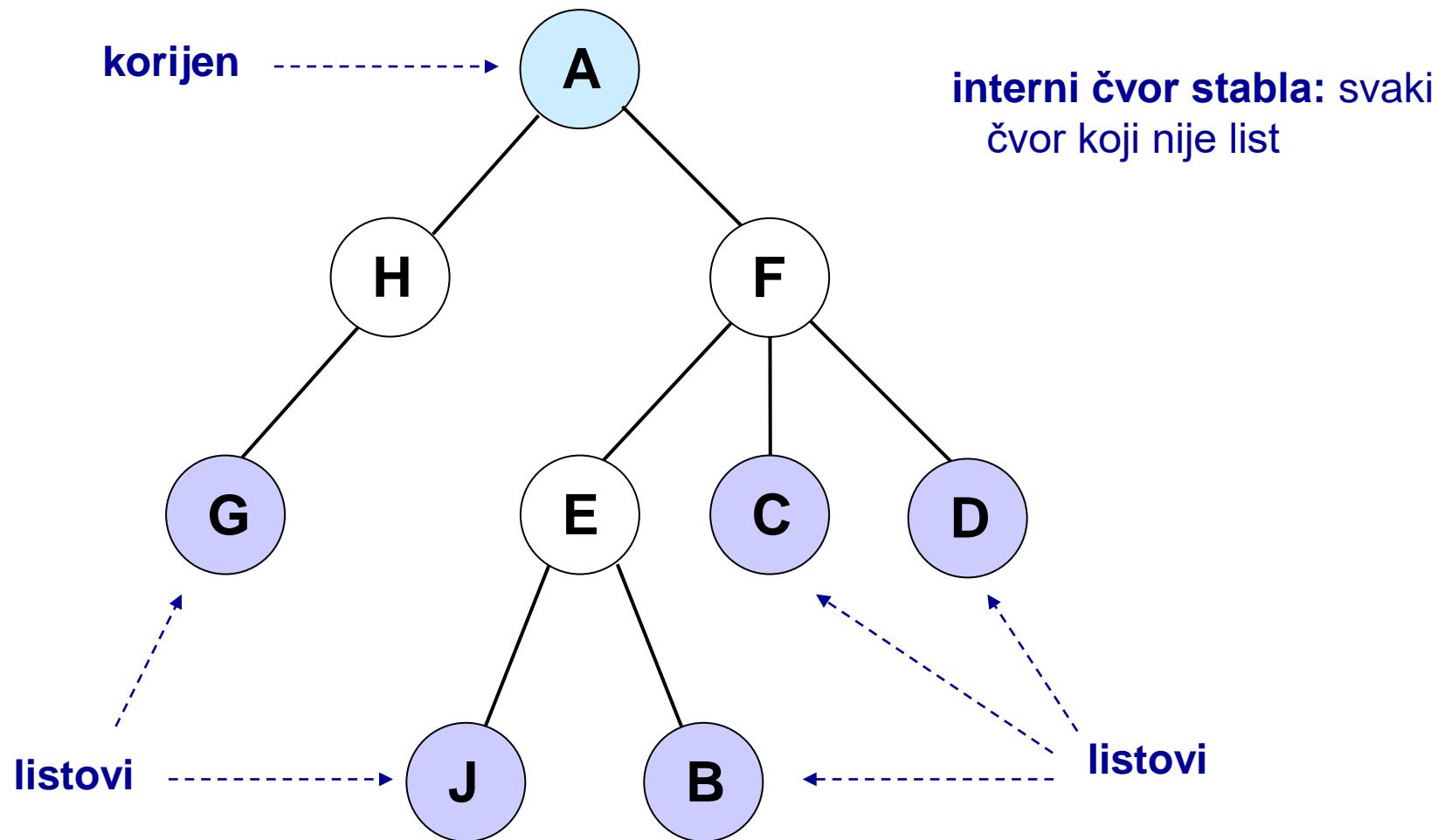


## Važna svojstva

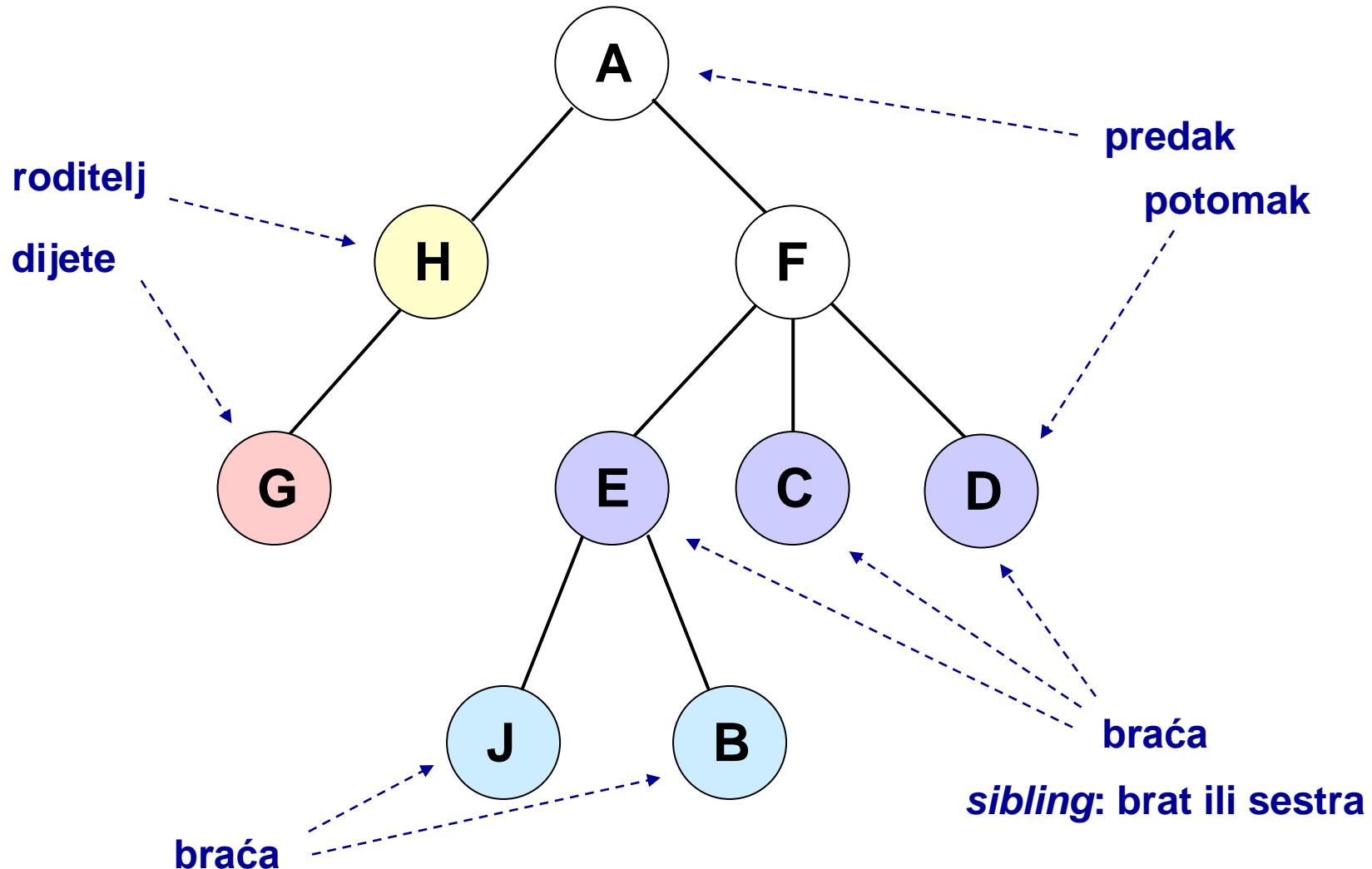
- najbrža metoda za operaciju pretraživanja prema ključu (najčešće samo jedna U/I operacija)
- efikasno dodavanje i brisanje zapisa
- problem: eventualni preljevi → ulančavanje
- pretraživanje moguće samo prema ključu (ne intervalu) jer zapisi nisu sortirani (osim za posebne *hash-funkcije*)

**B<sup>+</sup>-stablo**

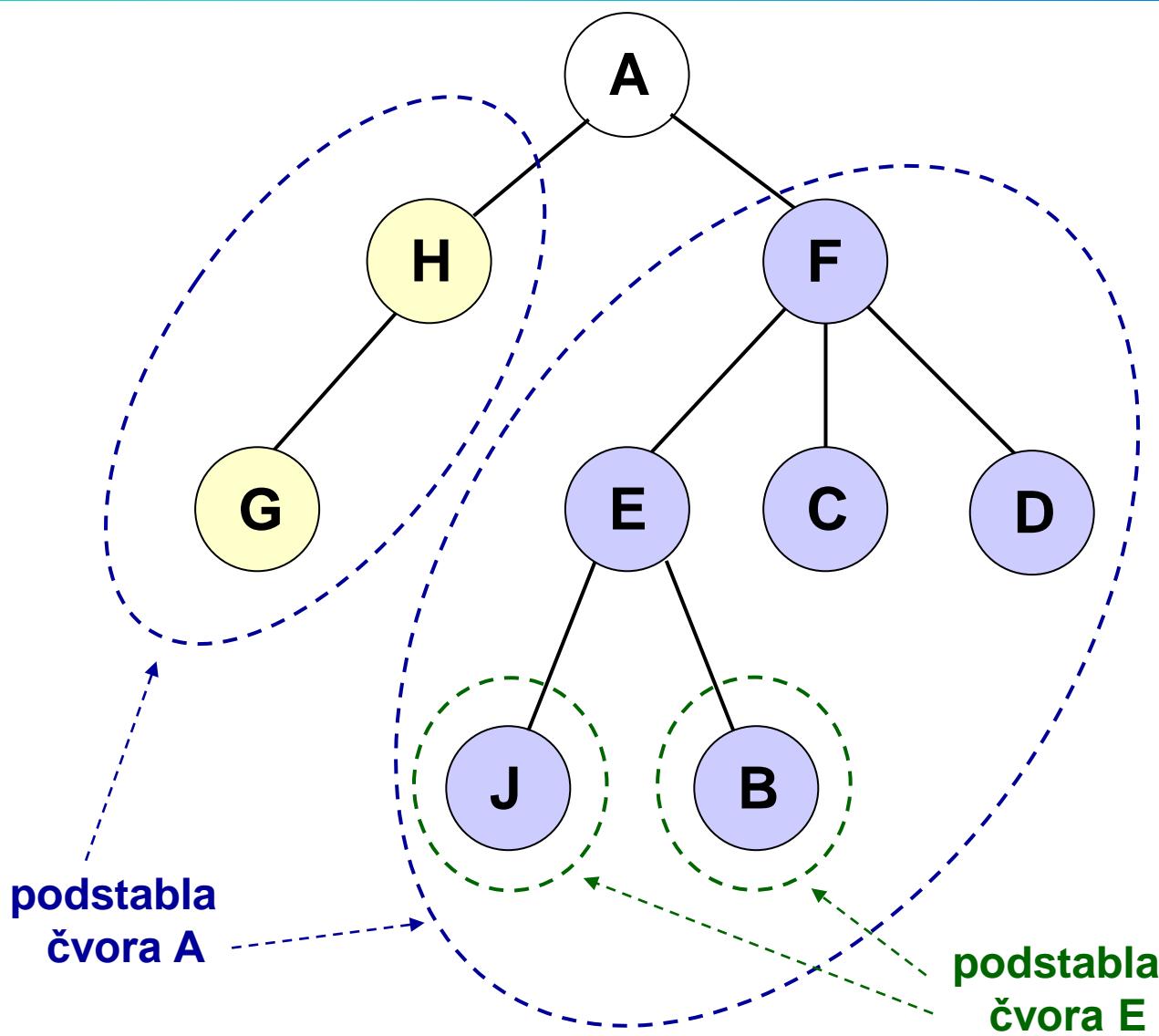
# Stablo kao struktura podataka



# Stablo kao struktura podataka



# Stablo kao struktura podataka



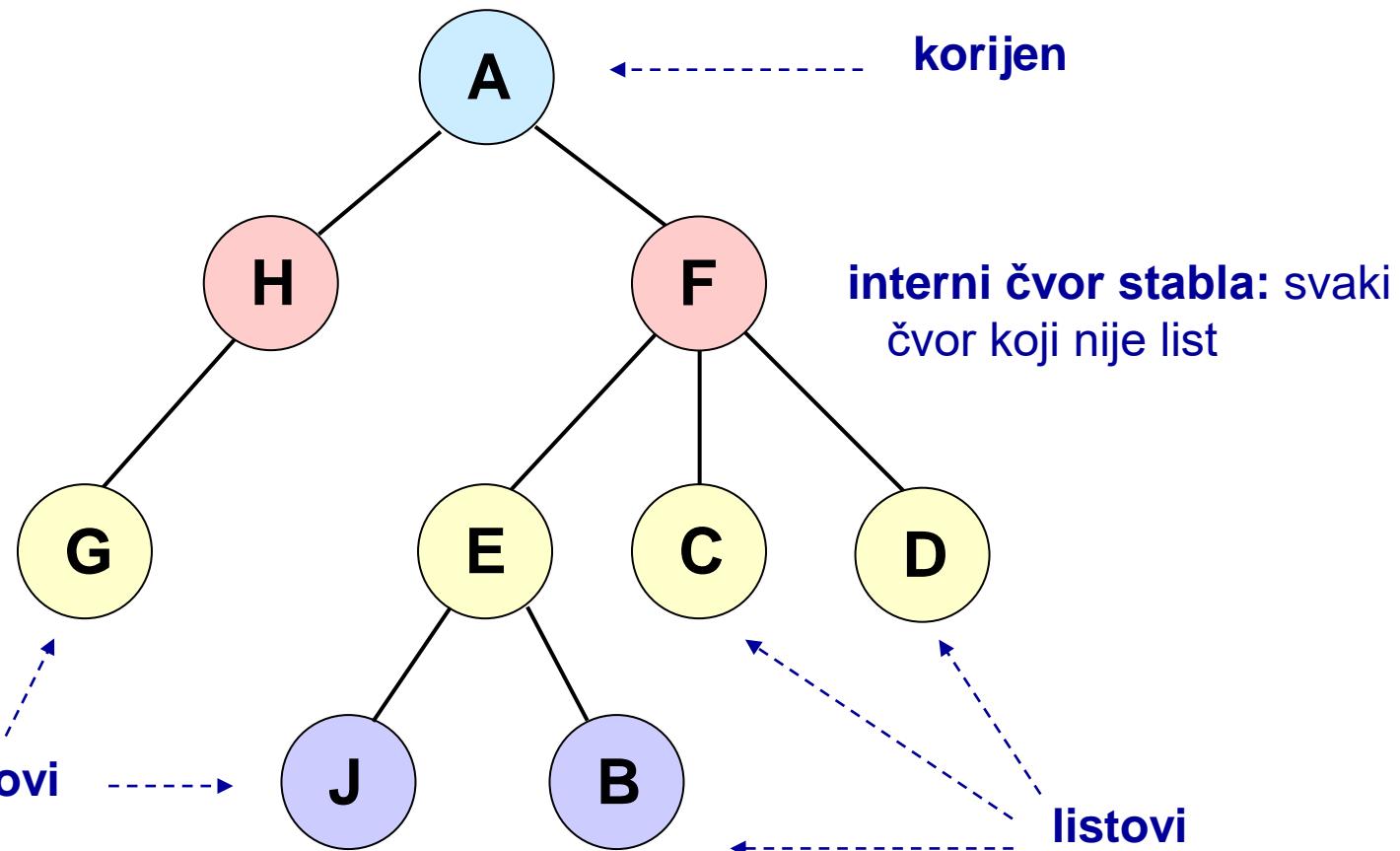
# Stablo kao struktura podataka

razina 0

razina 1

razina 2

razina 3

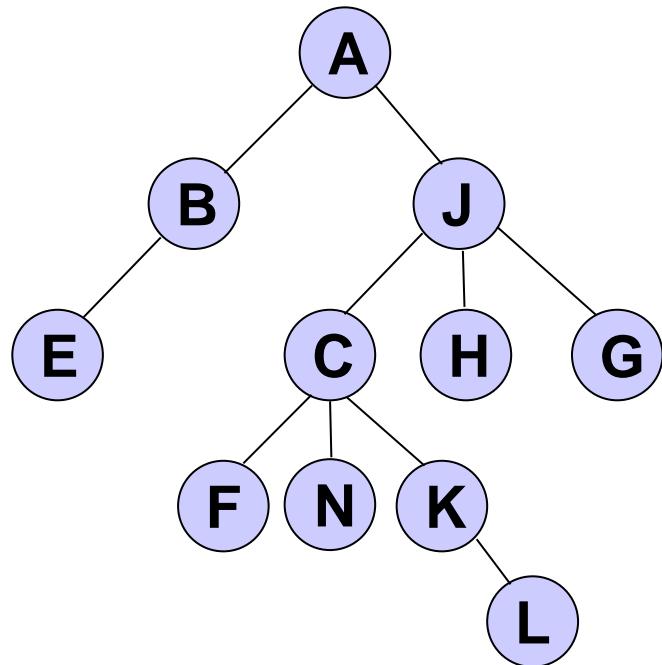


- **razina čvora (level):** duljina puta od korijena do čvora
- **dubina stabla (depth):** najveća duljina puta od korijena do lista
- **red stabla (order):** najveći broj djece koje čvor može imati

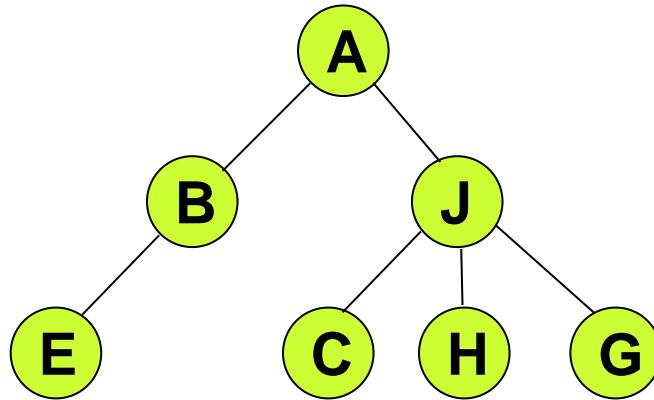
# Stablo kao struktura podataka

Stablo je **balansirano (*balanced*)** ako je duljina puta od korijena do lista jednaka za svaki list u stablu

stablo nije balansirano



stablo je balansirano



Oznaka **B** u B-stablo znači "**balansirano!**"!

## O nekonzistentnim i nepreciznim definicijama u literaturi

NIST - National Institute of Standards and Technology / Software and Systems Division / Information Technology Laboratory

**Balanced tree:** A tree where no leaf is *much farther* away from the root than any other leaf. Different balancing schemes allow different definitions of "much farther".

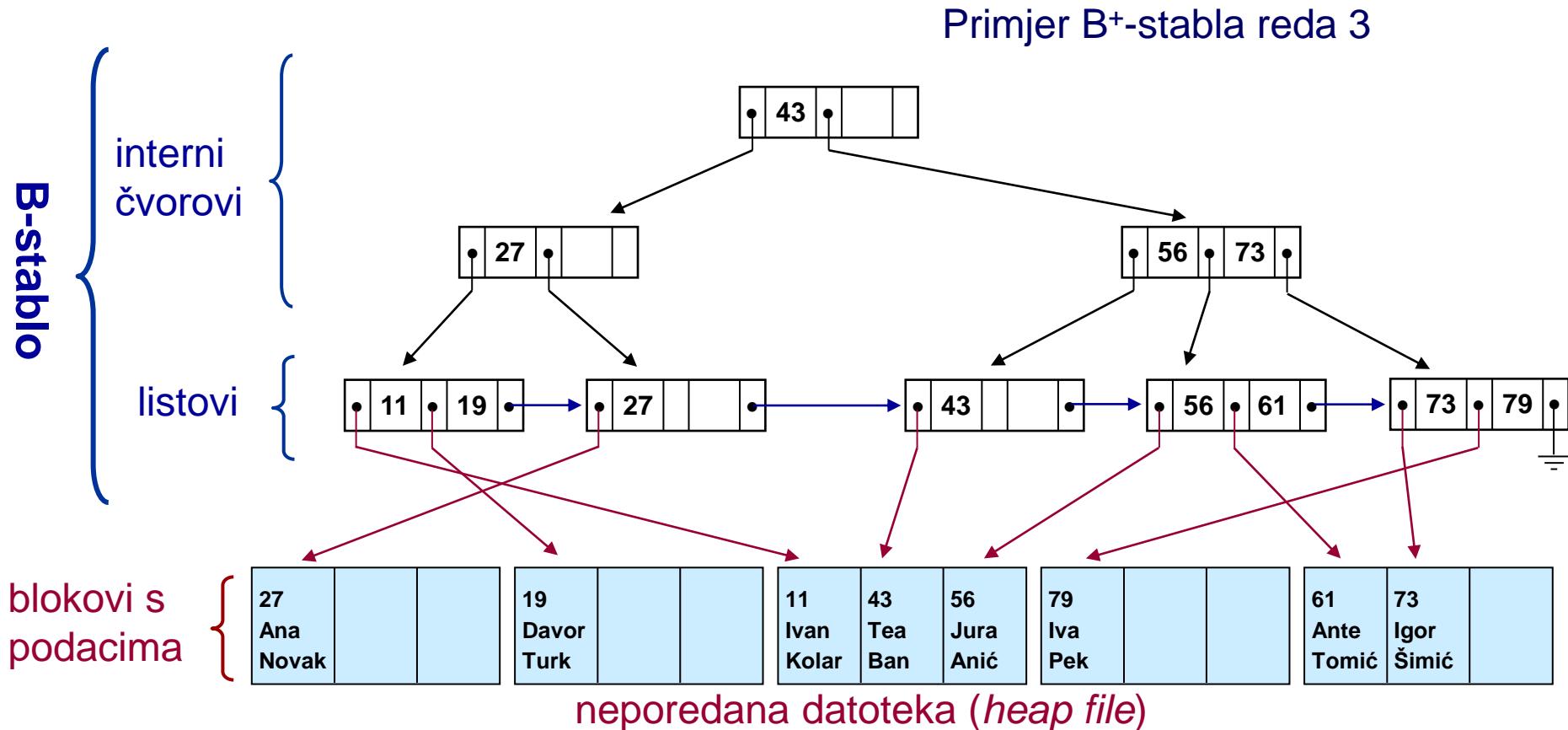
**AVL tree:** a *balanced binary* search tree where the height of the two subtrees (children) of a node differs by at most one.

R. Elmasri, S. Navathe. Fundamentals of database systems, 3rd ed.  
Addison-Wesley Publishing Company. Reading, Massachusetts. 2000.

[...] B-tree [...] that a search tree is **balanced**, meaning that all of its leaf nodes are at the same level.<sup>7</sup>

7. The definition of *balanced* is different for binary trees. Balanced binary trees are known as AVL trees.

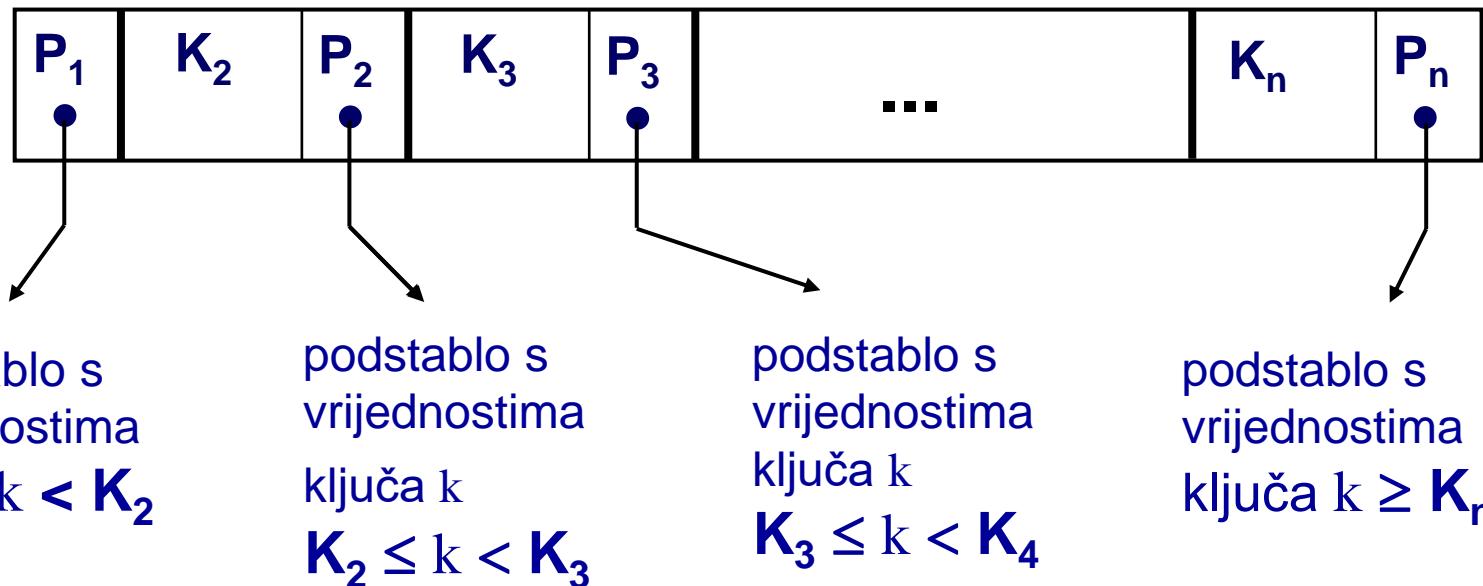
# Struktura B<sup>+</sup>-stabla



- Shema: mbr, ime, prezime; B<sup>+</sup>-stablo je izgrađeno za atribut mbr
- Moguće metode pristupa podacima (za bilo koji ključ pretrage):
  - linearnim pretraživanjem (kao kod neporedane datoteke)
  - ako je ključ pretrage mbr, može se koristiti ovo B-stablo

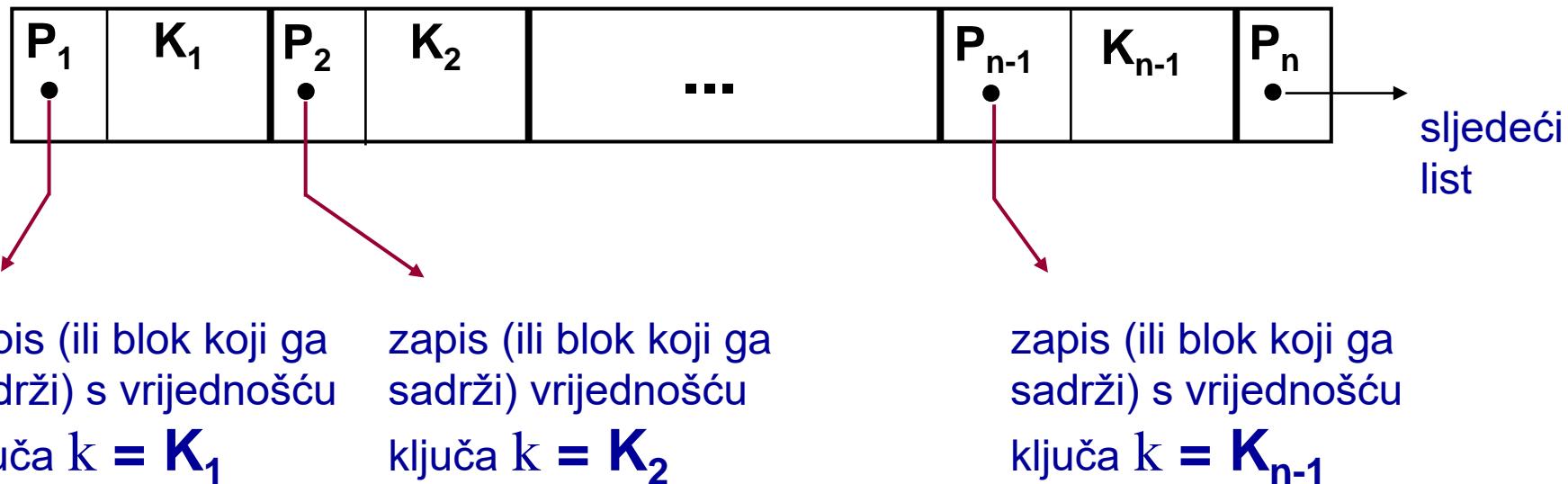
# Struktura internog čvora $B^+$ -stabla

- U  $B^+$ -stablu reda  $n$ , **interni čvor** sadrži:
  - najviše  $n$  kazaljki
  - najmanje  $\lceil n/2 \rceil$  kazaljki  $\rightarrow \lceil a \rceil$  je najmanji cijeli broj  $\geq a$ 
    - ovo ograničenje ne vrijedi za korijen (gdje je najmanji broj kazaljki 2)
  - uz  $p$  kazaljki u čvoru, broj pripadnih vrijednosti  $K_i$  u čvoru je  $p-1$ 
    - $K_i$  je vrijednost ključa



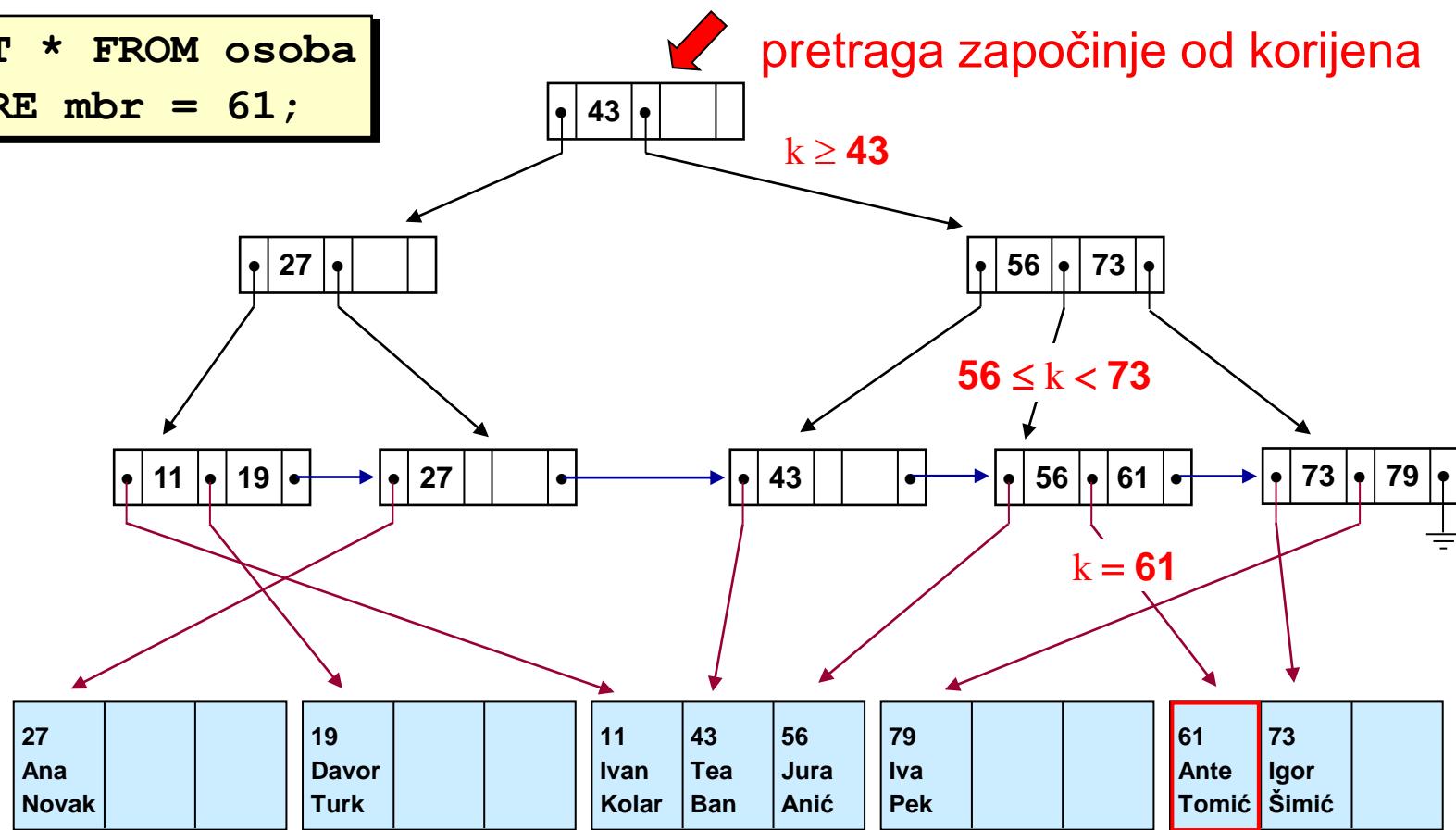
# Struktura lista B<sup>+</sup>-stabla

- U B<sup>+</sup>-stablu reda n, list sadrži:
  - najviše  $n-1$  vrijednosti  $K_i$  i pripadnih kazaljki na zapise (ili blokove)
  - najmanje  $\lceil(n-1)/2\rceil$  vrijednosti  $K_i$  i pripadnih kazaljki na zapise
  - svi listovi (osim zadnjeg) sadrže kazaljku na sljedeći list
    - omogućuje upite tipa od-do (prema zadanim granicama intervala)



# Algoritam za pronalaženje zapisa putem B<sup>+</sup>-stabla

```
SELECT * FROM osoba  
WHERE mbr = 61;
```

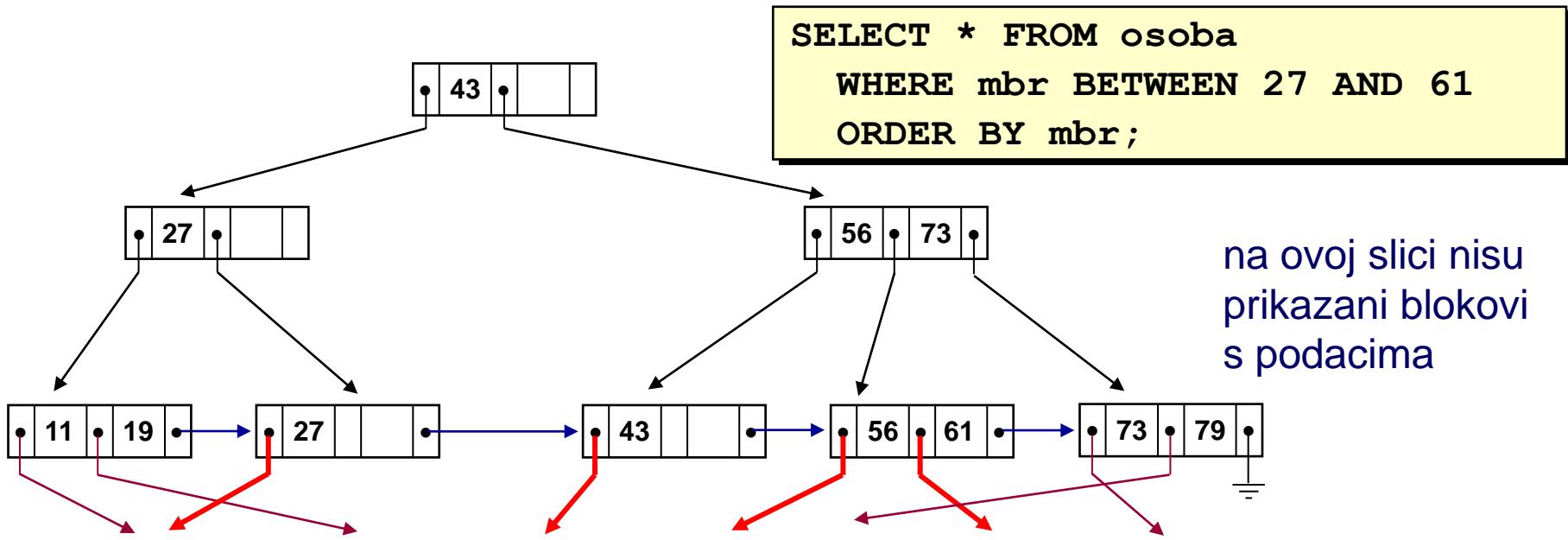


- slijediti odgovarajuću kazaljku do sljedeće razine
- postupak se ponavlja dok se ne dođe do lista u kojem će se naći kazaljka na zapis u bloku s podacima (ako n-torka s tim ključem uopće postoji)

# Algoritam za pronalaženje zapisa putem B<sup>+</sup>-stabla

- algoritam za traženje zapisa s ključem vrijednosti  $k$  je rekurzivan
  - cilj je u svakom koraku rekurzije (pretraga i-te razine) pronaći čvor na nižoj,  $(i+1)$ -voj razini, koji će voditi prema listu u kojem se nalazi ključ čija je vrijednost  $k$
- traženje zapisa započinje od korijena (0-te razine)
- u čvoru i-te razine potrebno je pronaći najveću vrijednost ključa koja je manja ili jednaka traženoj vrijednosti  $k$ 
  - za prvu kazaljku internog čvora nije navedena vrijednost ključa, pa ona "pokriva" sve vrijednosti ključeva manje od vrijednosti ključa  $K_2$
- nakon pronalaženja odgovarajuće vrijednosti ključa, slijedi se pripadna kazaljka i time se obavlja pozicioniranje na  $(i+1)$ -vu razinu
- postupak se ponavlja rekursivno sve dok se ne dođe do lista. U njemu se mora nalaziti, ako postoji, ključ čija je vrijednost  $k$ , te pripadna kazaljka prema traženom zapisu ( $n$ -torki)

# Dohvat podataka iz intervala, sortiranje



- u listu pronaći kazaljku na zapis s ključem **27**
- redom dohvaćati kazaljke i pripadne zapise dok se ne dođe do kazaljke na zapis s ključem **61**
  - taj postupak omogućuju kazaljke među listovima
- dobiveni su svi traženi zapisi, pri tome su poredani prema mbr
- Ako se obavlja **SELECT \* ... ORDER BY mbr DESC**
  - pronađene zapise jednostavno ispisati obrnutim redoslijedom

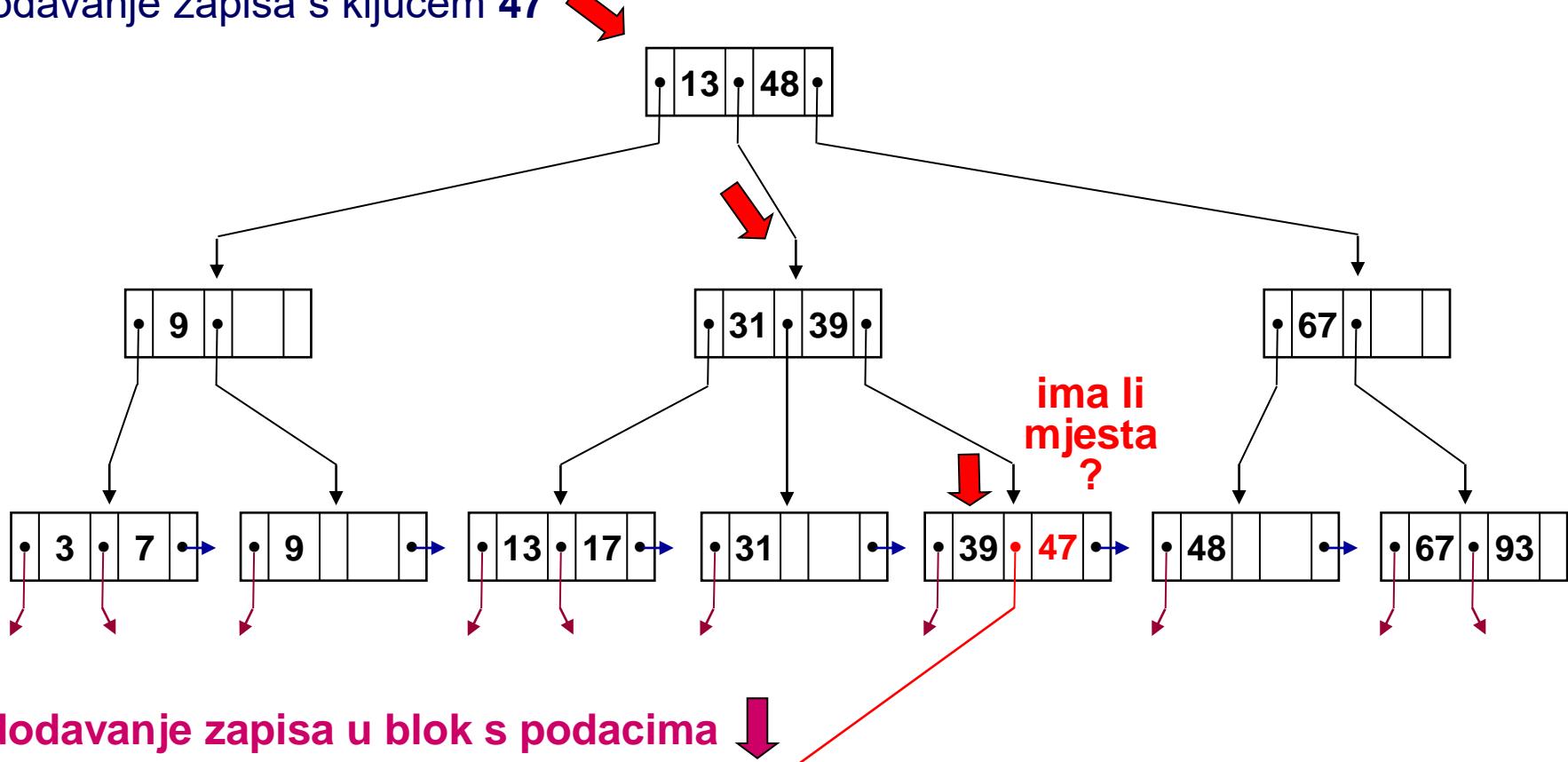
# Algoritam za dodavanje zapisa u B<sup>+</sup>-stablu

---

- zapis (podaci) se upisuje u jedan od slobodnih blokova s podacima
- obavlja se procedura za pronalaženje lista L u koji pripada vrijednost ključa k
- ako čvor L nije popunjen, u čvor se dodaje zapis s ključem k i kazaljkom na pripadni zapis u datoteci, uz očuvanje poretku vrijednosti ključeva
  - nova vrijednost nikad nije prva u čvoru, osim u slučaju krajnjeg lijevog čvora
- ako je čvor L popunjen
  - stvara se novi čvor i zapisi među njima se podijele, pri čemu svakom od čvorova pripadne polovica zapisa
  - budući da je dodan novi čvor, u nadređeni čvor potrebno je dodati zapis s kazaljkom i najmanjom vrijednošću ključa u novom čvoru
  - za dodavanje novog zapisa u nadređeni čvor koristi se ista procedura kao za dodavanje zapisa u čvor na nižoj razini
- postupak je rekurzivan i mora se obaviti za svaku nadređenu razinu (sve dok se ne dođe do korijena ili se na nekoj od razina nađe dovoljno mesta za upis vrijednosti ključa i kazaljke, bez dodavanja novih čvorova).
- ako se dođe do korijena, može se desiti da u korijenu nema mesta za novi zapis. Tada se dodaje novi čvor i formira se novi korijen na višoj razini

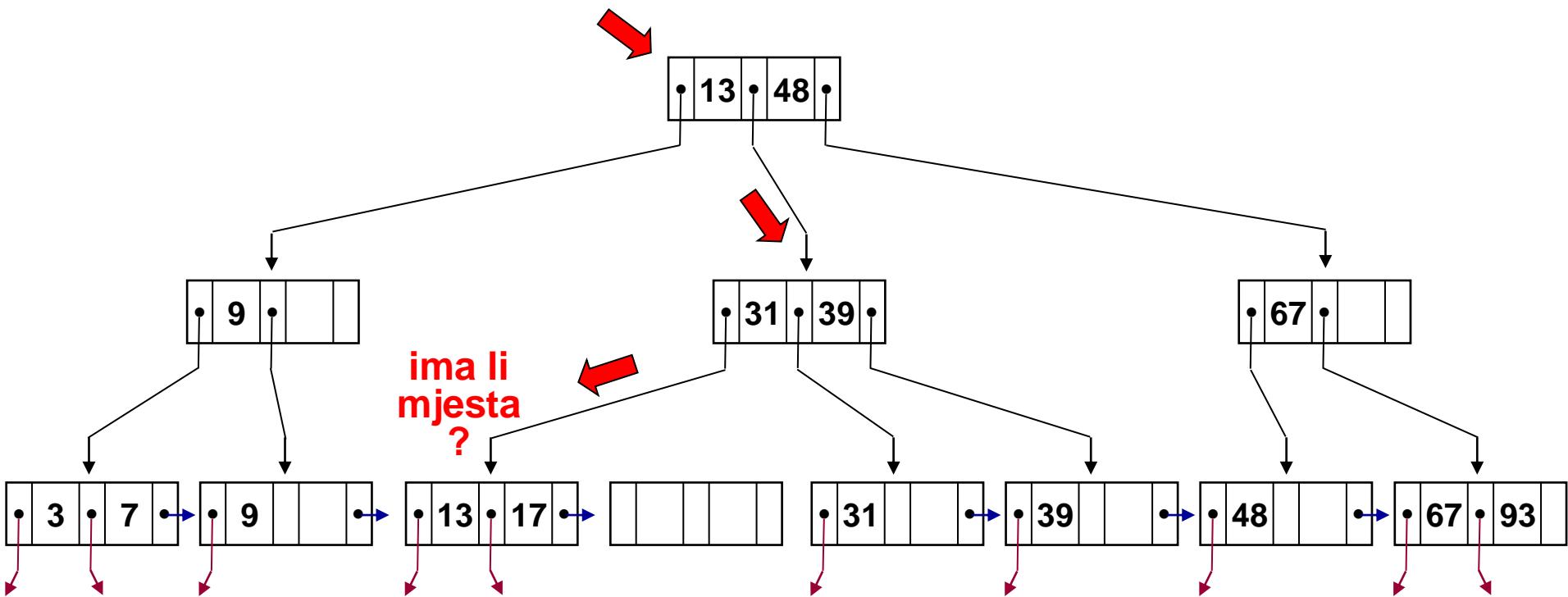
# Dodavanje zapisa u B<sup>+</sup>-stablu

dodavanje zapisa s ključem 47



# Dodavanje zapisa u B<sup>+</sup>-stablu

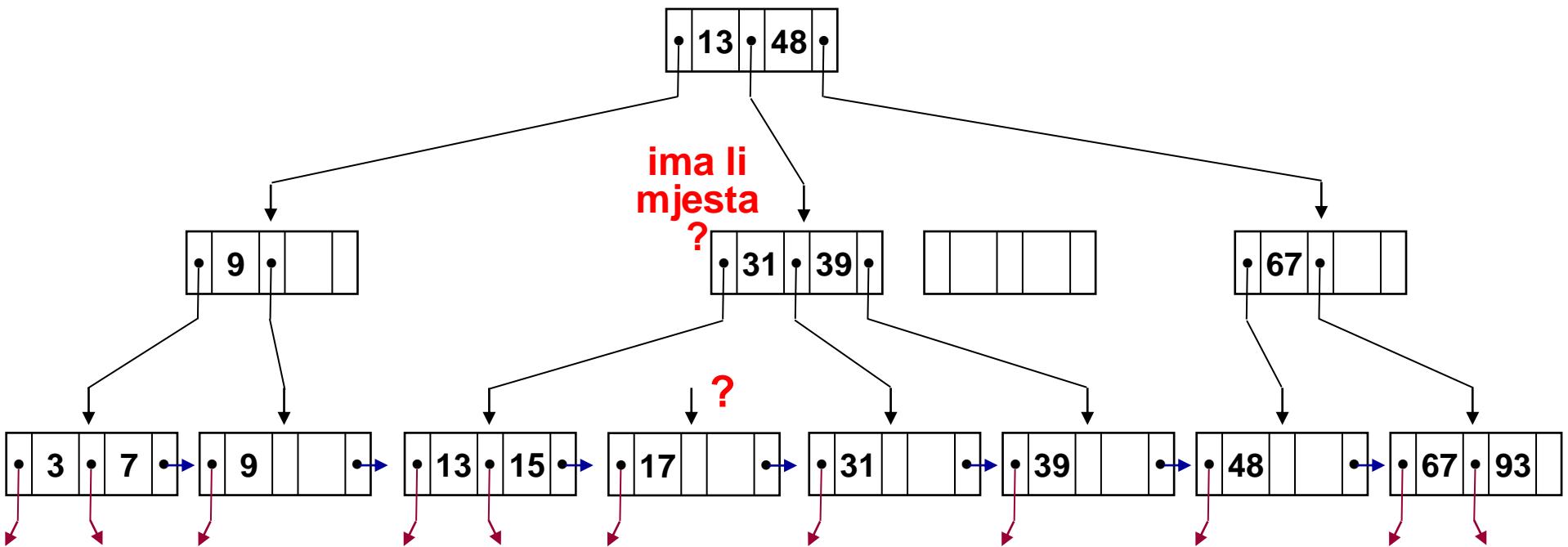
dodavanje zapisa s ključem 15



dodati novi čvor i podijeliti zapise  
13, 15, 17 među čvorovima

# Dodavanje zapisa u B<sup>+</sup>-stablu

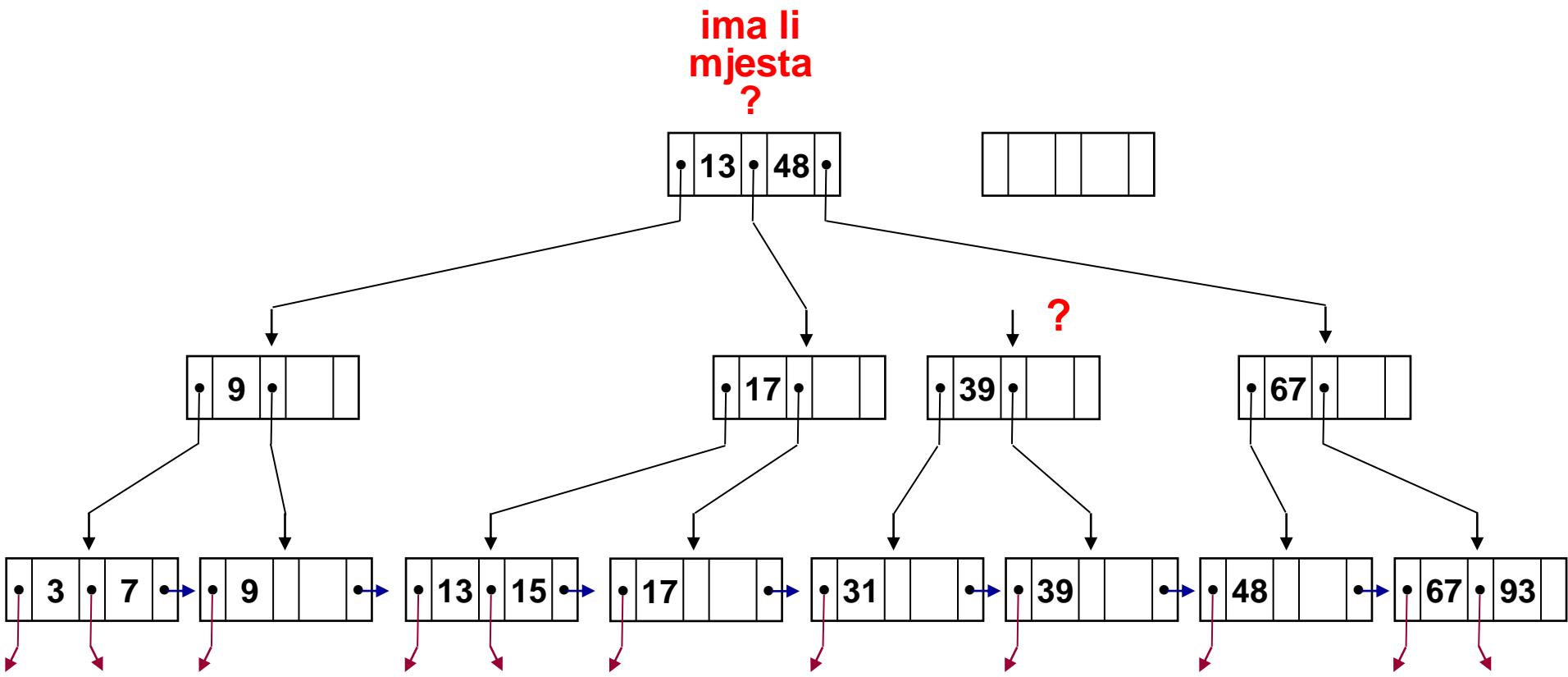
dodavanje zapisa s ključem 15



dodati novi čvor i podijeliti zapise  
13, 17, 31, 39 među čvorovima

# Dodavanje zapisa u B<sup>+</sup>-stablu

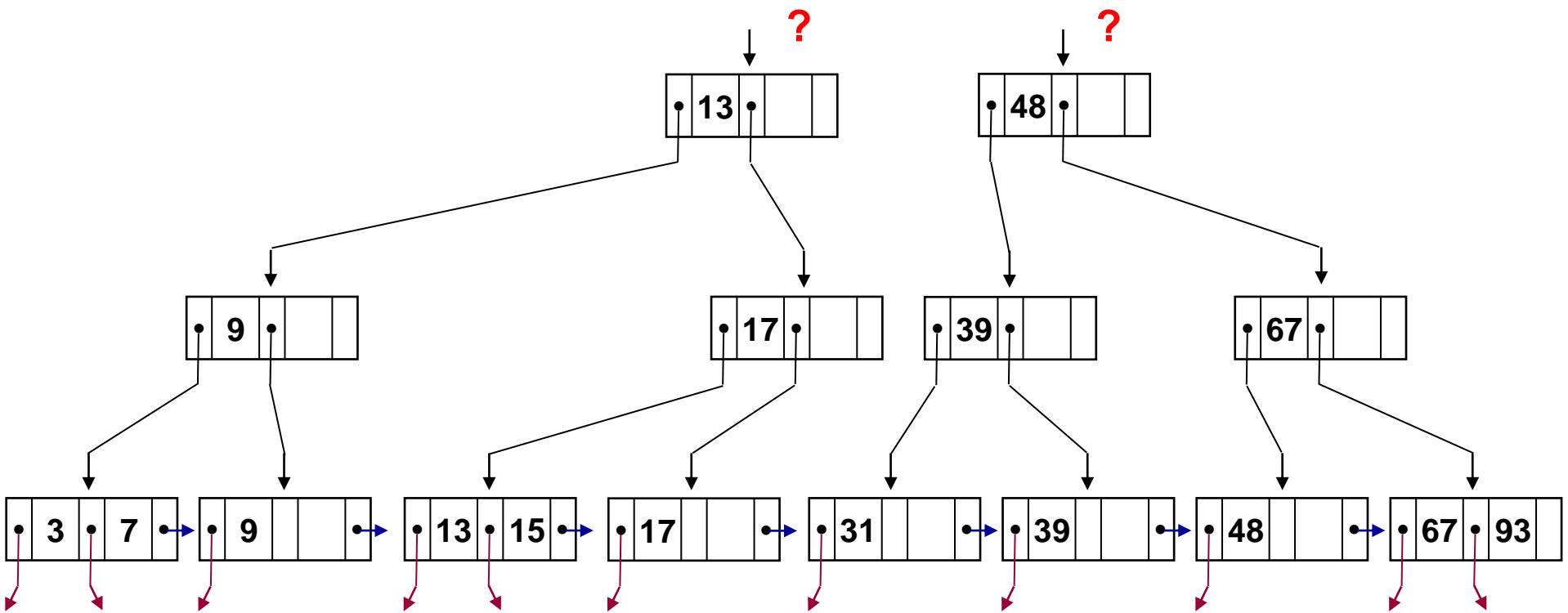
dodavanje zapisa s ključem 15



dodati novi čvor i podijeliti zapise  
-∞, 13, 31, 48 među čvorovima

# Dodavanje zapisa u B<sup>+</sup>-stablu

dodavanje zapisa s ključem 15



dodati novi korijen i upisati zapise  $-\infty, 31$

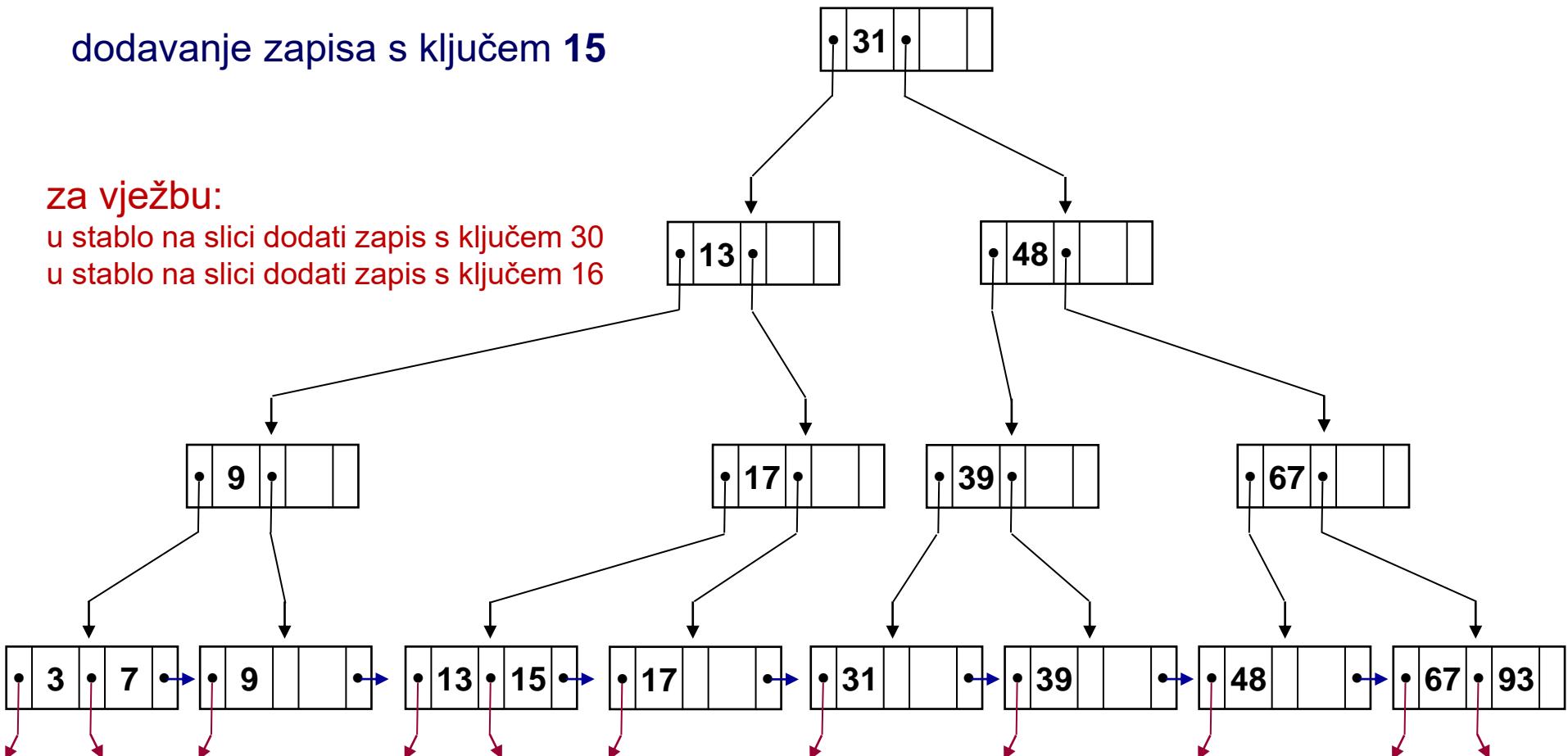
# Dodavanje zapisa u B<sup>+</sup>-stablu

dodavanje zapisa s ključem 15

za vježbu:

u stablo na slici dodati zapis s ključem 30

u stablo na slici dodati zapis s ključem 16



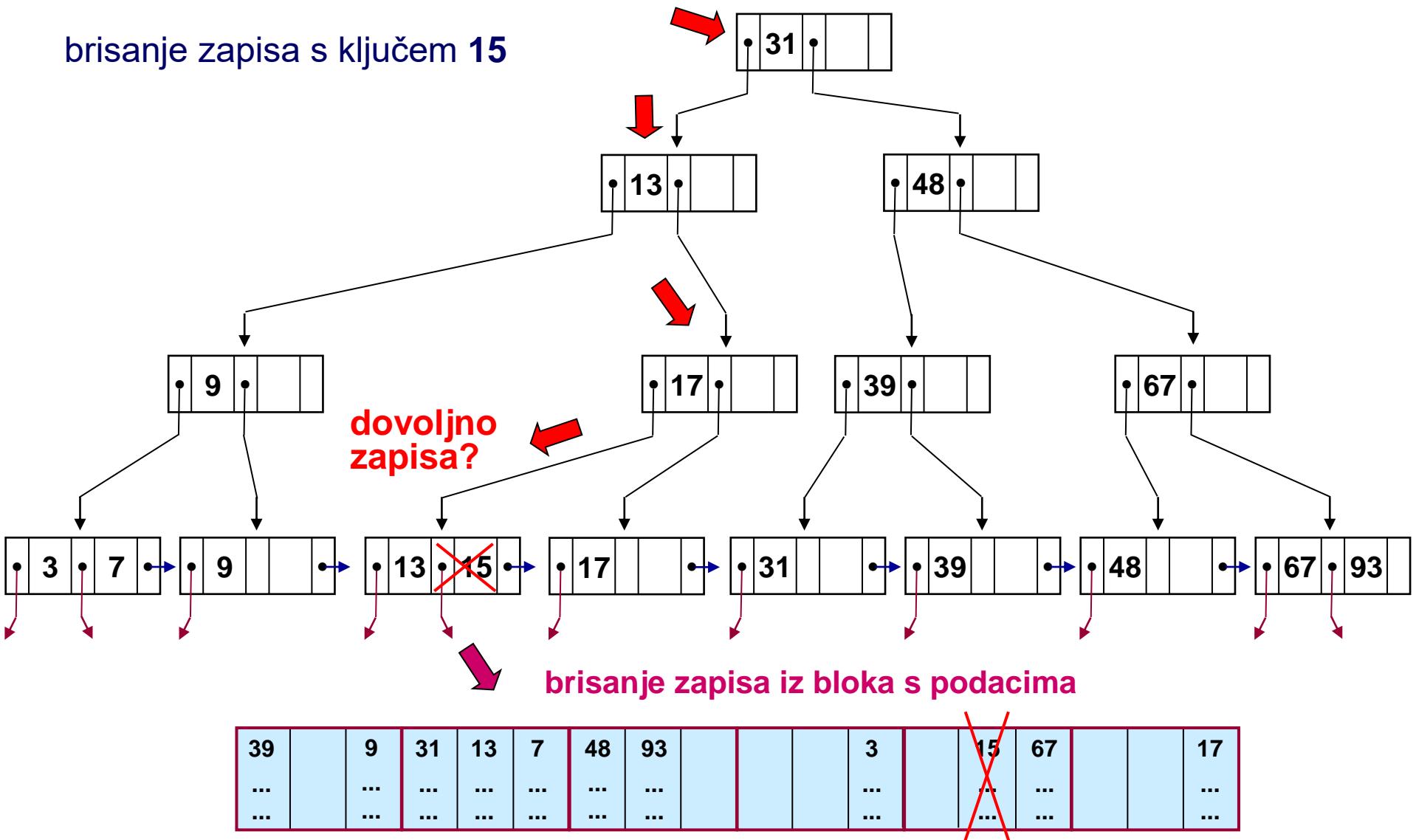
rezultat je balansirano stablo veće dubine

# Algoritam za brisanje zapisa iz B<sup>+</sup>-stabla

- obavlja se procedura za pronalaženje lista L u kojem se nalazi ključ k
- zapis se briše iz bloka s podacima, a vrijednost ključa i kazaljka iz čvora L
- ako čvor L nakon brisanja sadrži dovoljan broj zapisa
  - samo ako je potrebno (onda kada se iz lista obriše prvi zapis), trebat će promijeniti vrijednost ključa u nekom od predaka
- inače (tj. ako čvor L nakon brisanja nema dovoljan broj zapisa)
  - ako postoji čvor-brat  $L_x$  koji se nalazi neposredno s lijeva ili s desna čvoru L i ima za barem jedan više od dovoljnog broja zapisa
    - zapise podijeliti između čvorova L i odabranog  $L_x$ . Po potrebi obaviti korekciju ključa u nekom od predaka
  - inače odabrati jednog od braće koji se nalazi neposredno s lijeva ili s desna čvoru L (taj sigurno ima najmanji dopušteni broj zapisa)
    - čvorovi L i  $L_x$  se spajaju u jedan čvor. U nadređenom čvoru briše se jedan zapis te po potrebi mijenja ključ u nekom od predaka
    - brisanje zapisa u nadređenom čvoru svodi se na rekurzivno izvođenje procedure za brisanje. Ako se putem prema korijenu dođe u situaciju da treba spojiti jedina dva čvora-djeteta korijena, tada se oni spajaju, postaju novi korijen stabla, a stari se korijen briše

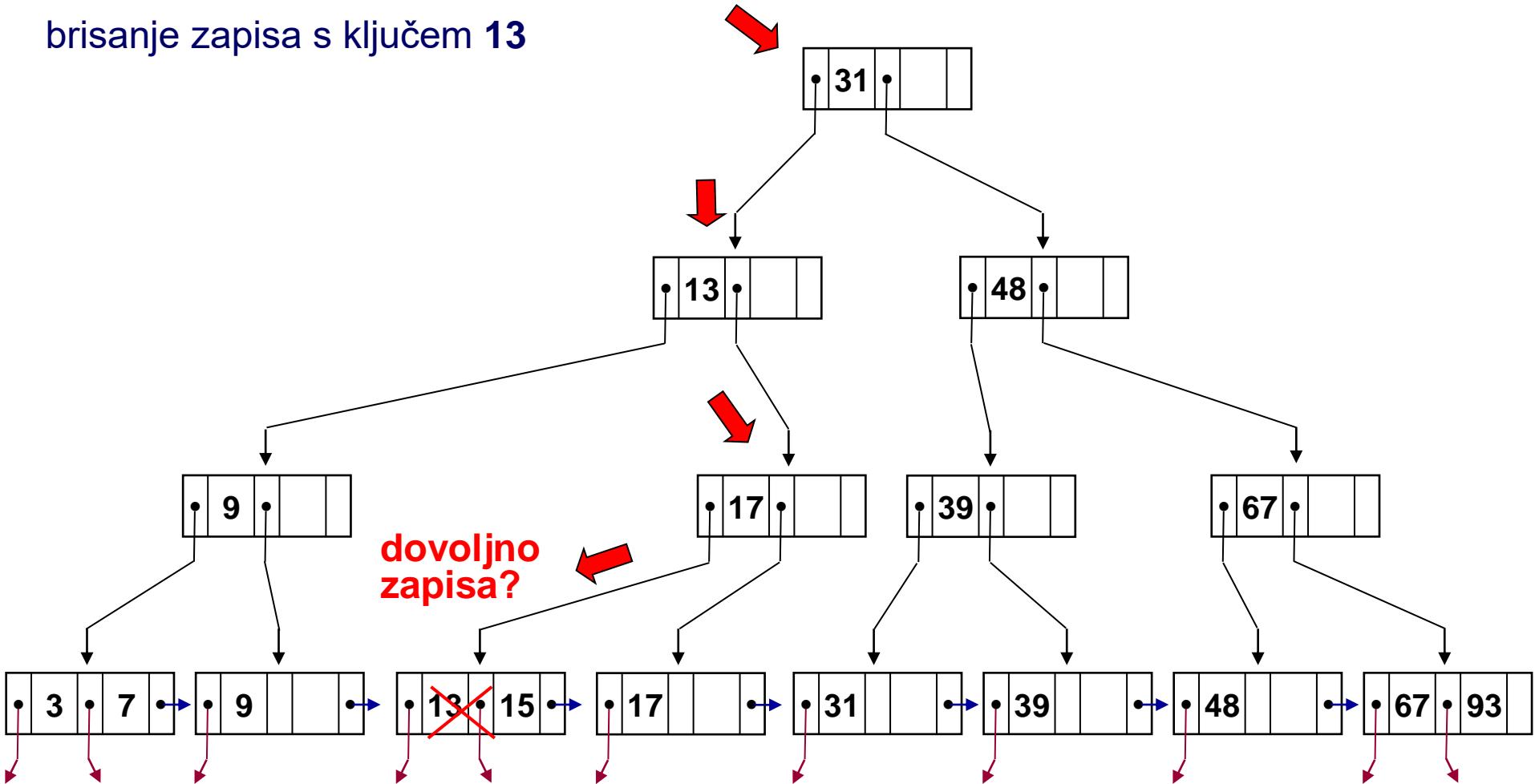
# Brisanje zapisa iz B<sup>+</sup>-stabla

brisanje zapisa s ključem 15



# Brisanje zapisa iz B<sup>+</sup>-stabla

brisanje zapisa s ključem 13

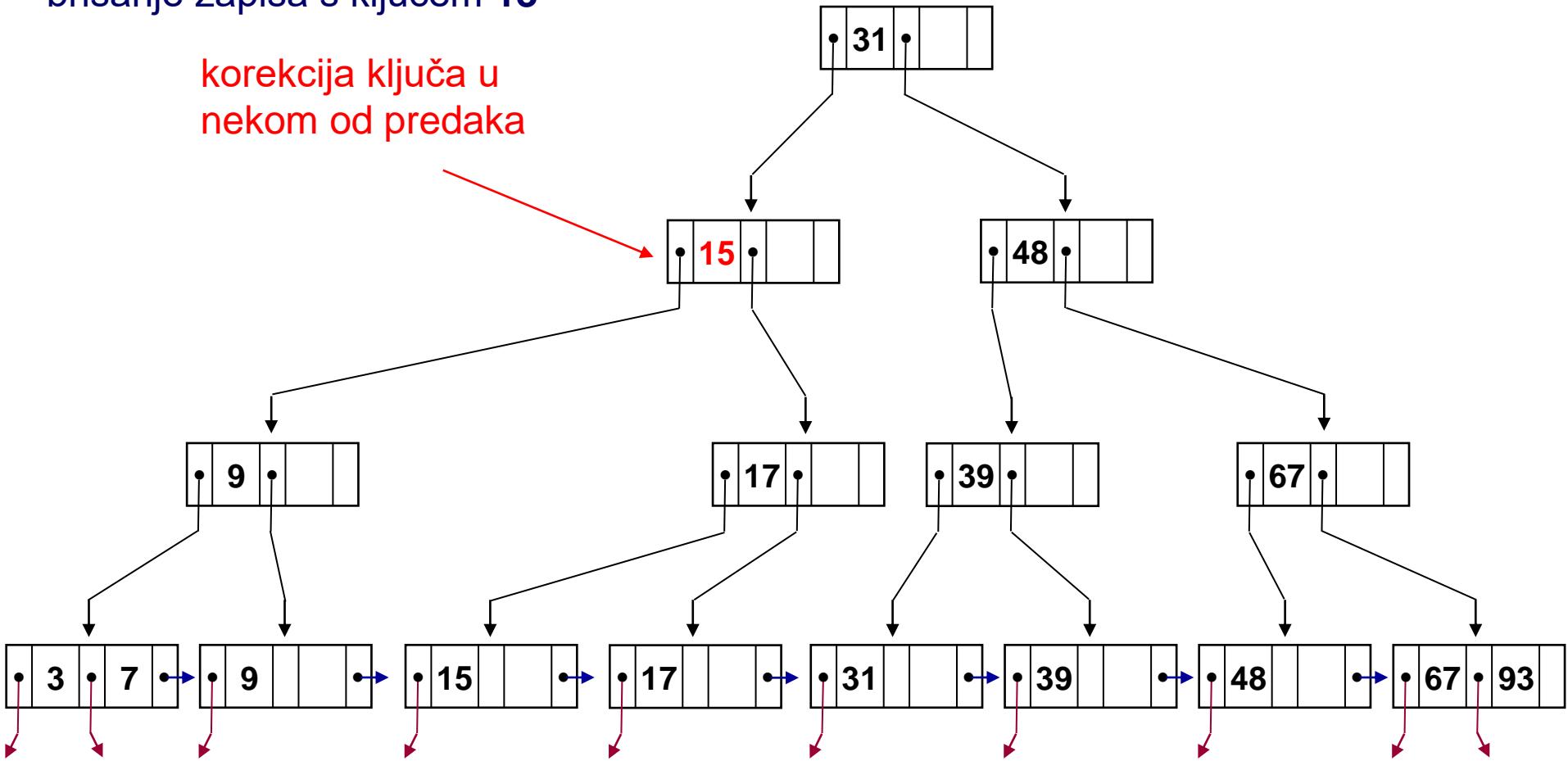


obaviti korekciju ključa u nekom od predaka  
(jer sada je prvi zapis u listu 15, umjesto 13)

# Brisanje zapisa iz B<sup>+</sup>-stabla

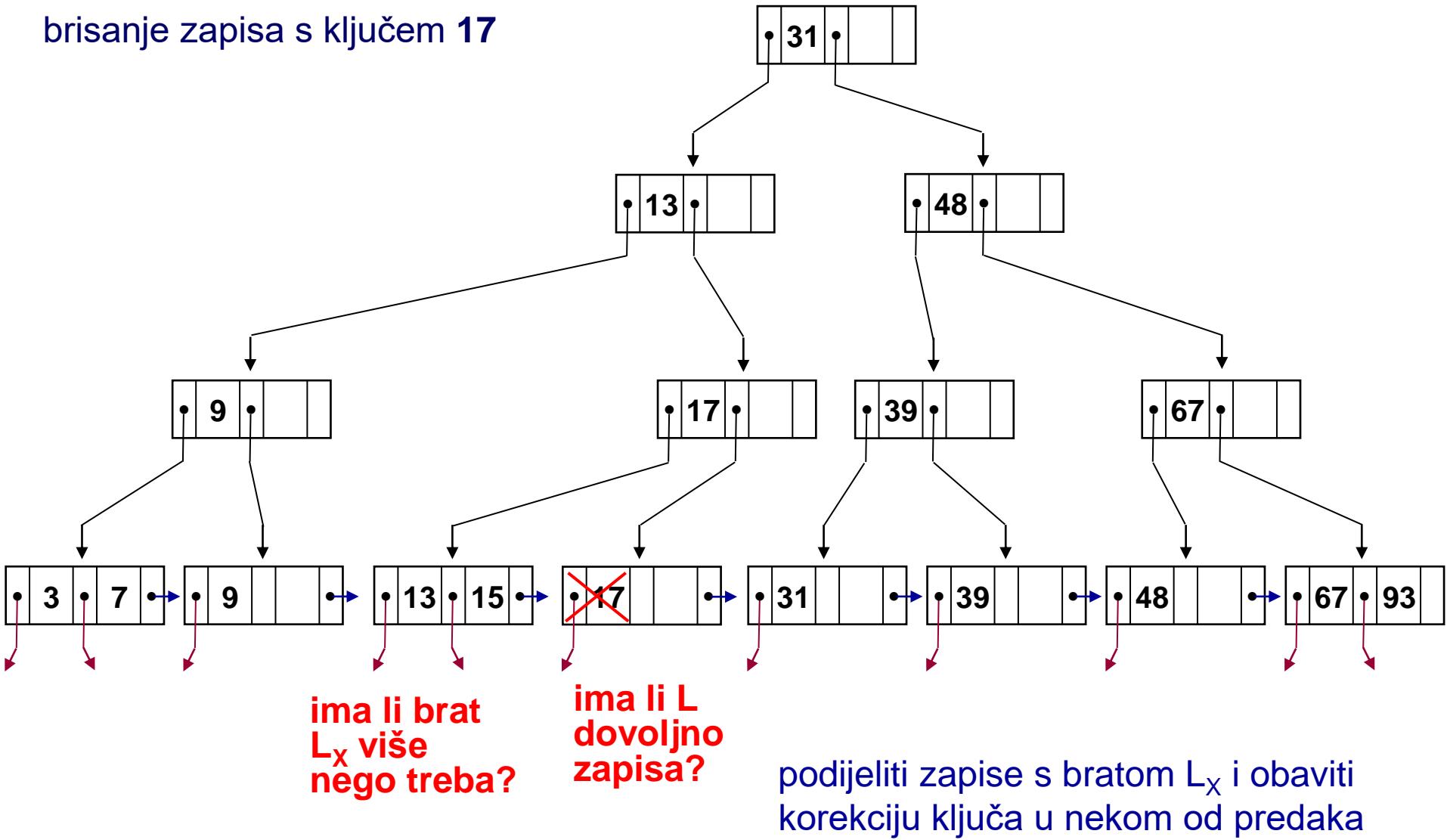
brisanje zapisa s ključem 13

korekcija ključa u  
nekom od predaka



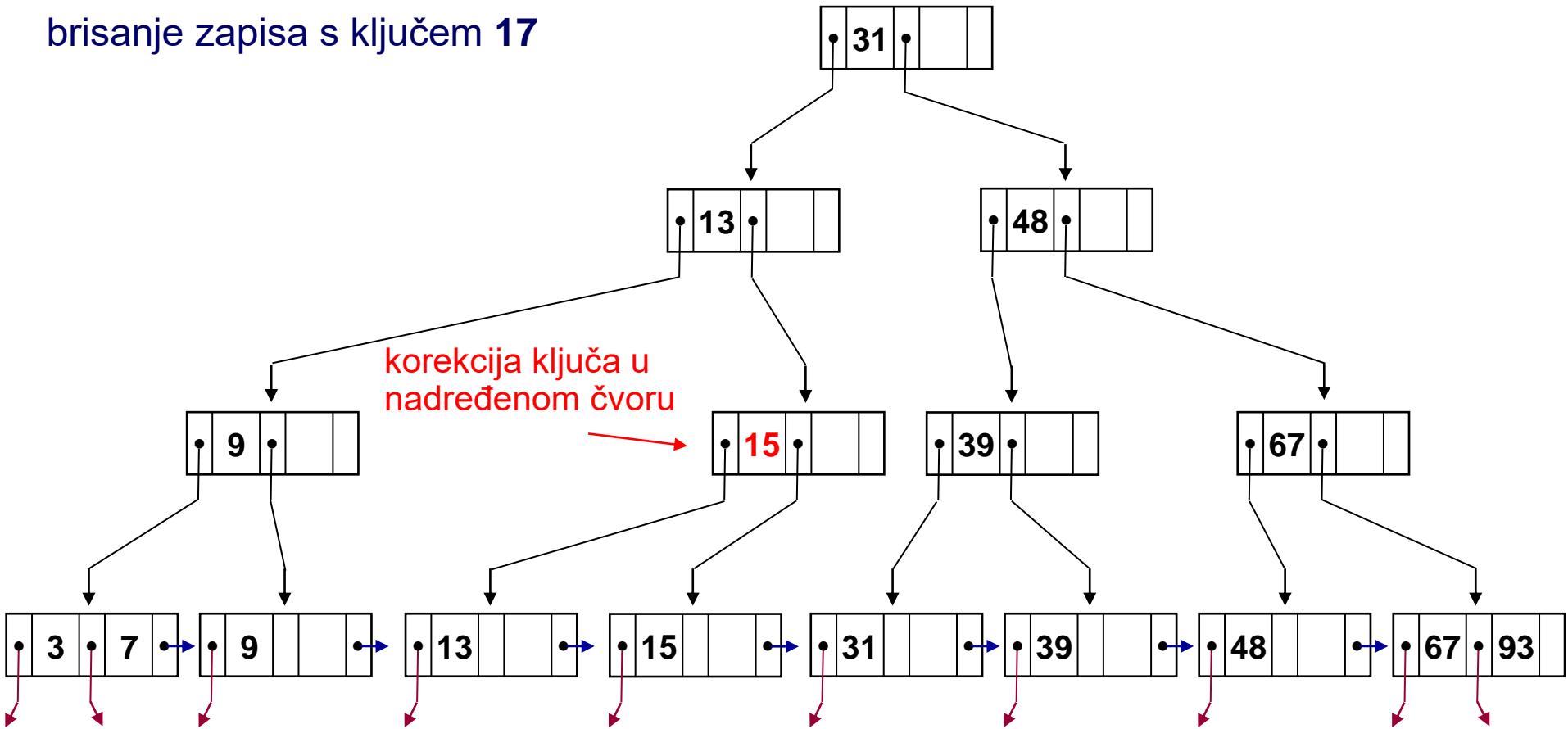
# Brisanje zapisa iz B<sup>+</sup>-stabla

brisanje zapisa s ključem 17



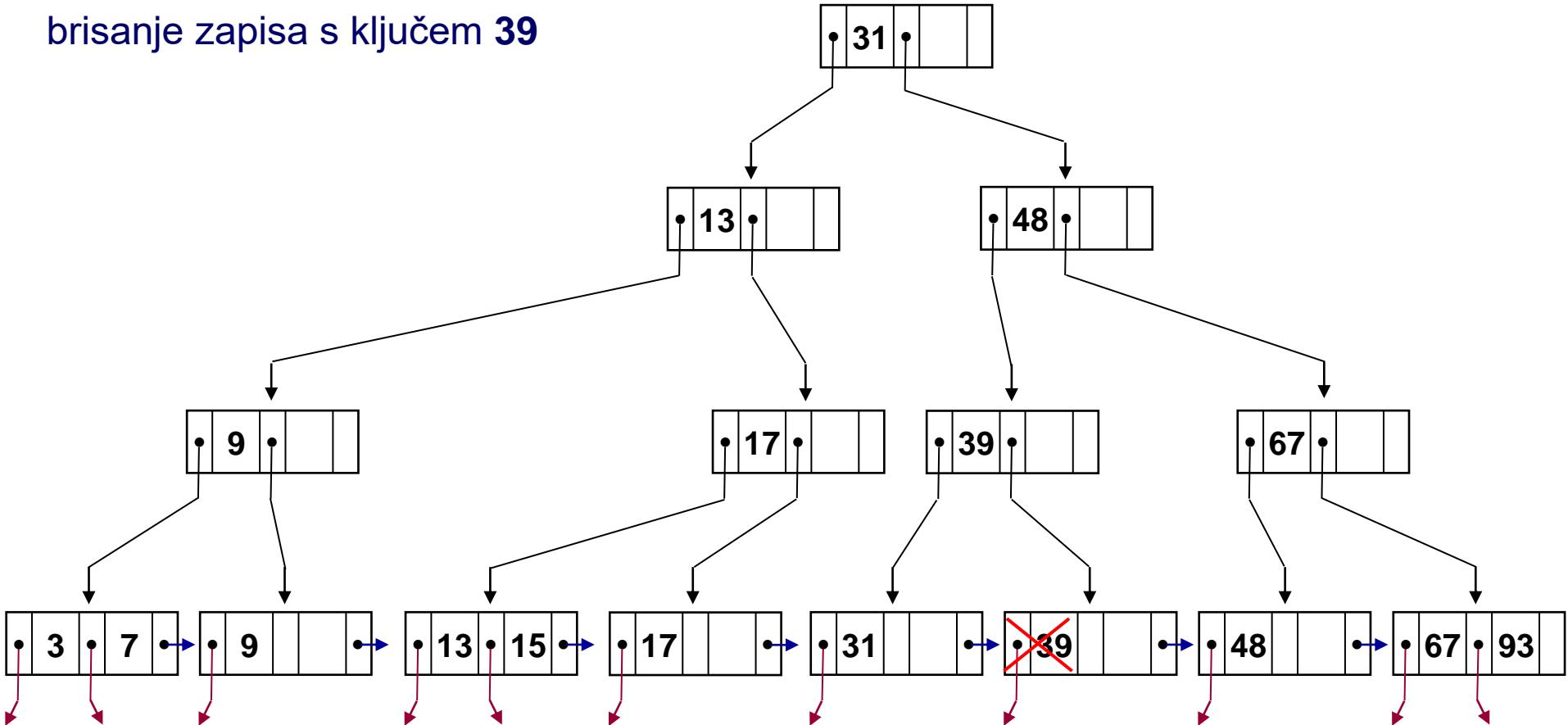
# Brisanje zapisa iz B<sup>+</sup>-stabla

brisanje zapisa s ključem 17



# Brisanje zapisa iz B<sup>+</sup>-stabla

brisanje zapisa s ključem 39



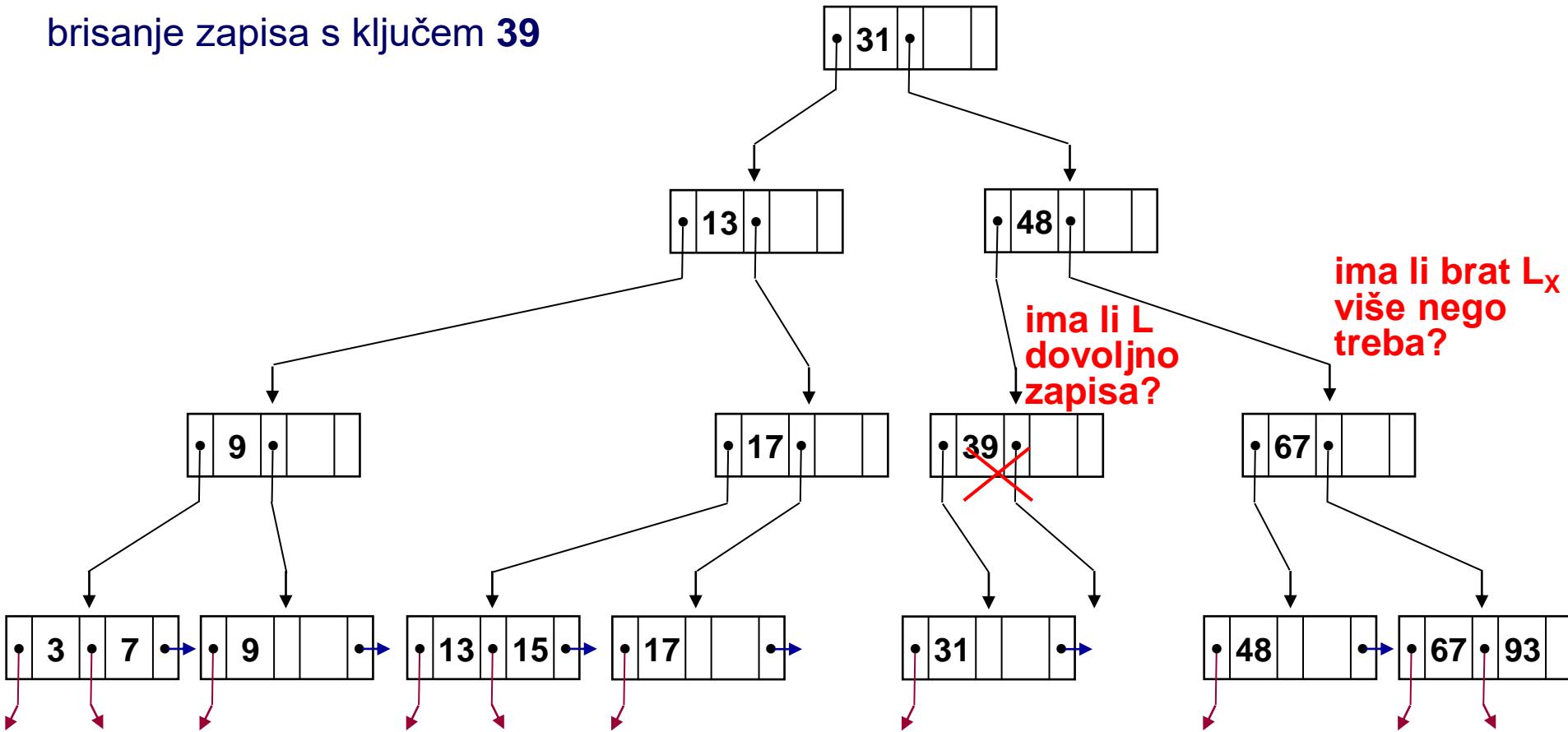
spojiti L i  $L_x$ . Obrisati zapis u nadređenom čvoru i po potrebi promijeniti vrijednost ključa u nekom od predaka.

ima li brat  
 $L_x$  više  
nego treba?

ima li L  
dovoljno  
zapisa?

# Brisanje zapisa iz B<sup>+</sup>-stabla

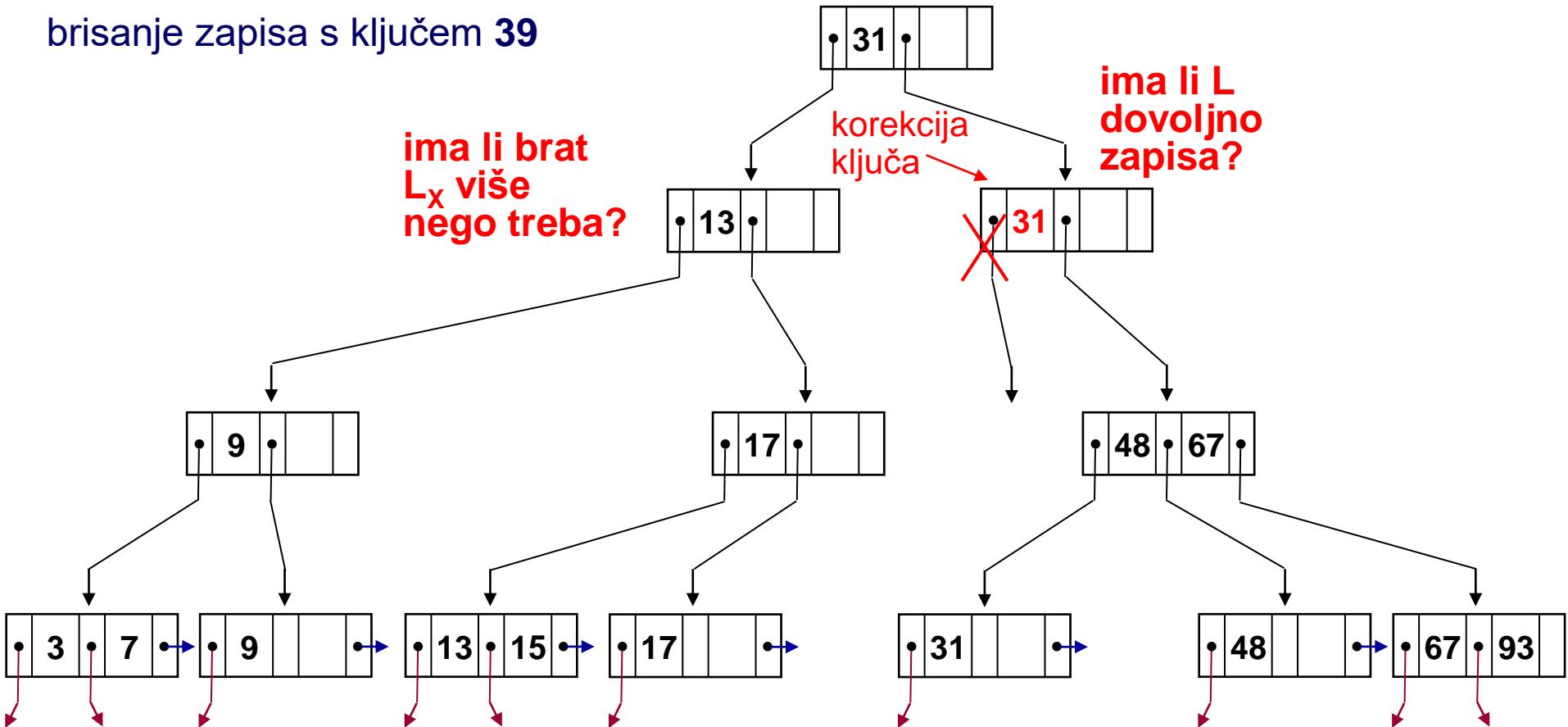
brisanje zapisa s ključem 39



spojiti L i L<sub>x</sub>. Obrisati zapis iz nadređenog čvora i po potrebi promijeniti vrijednost ključa u nekom od predaka.

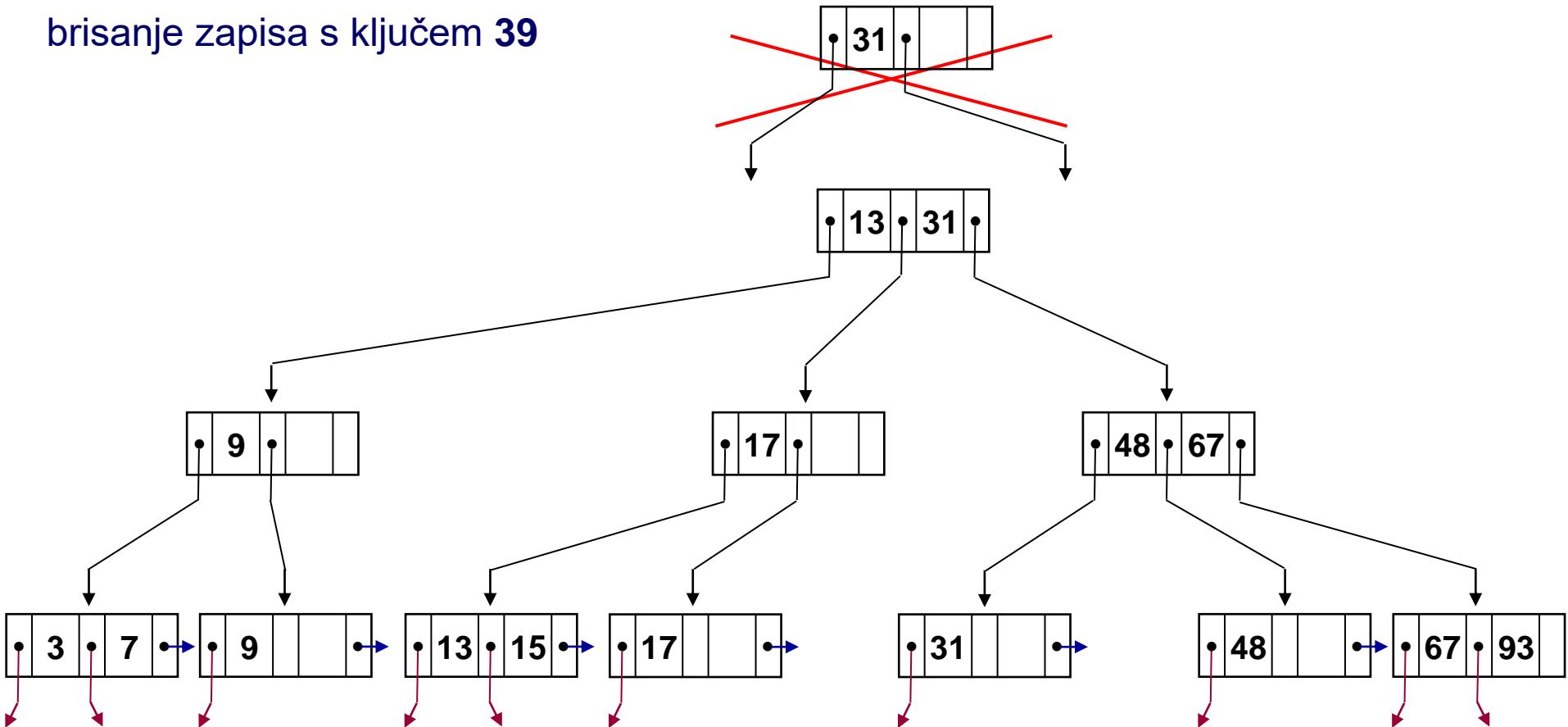
# Brisanje zapisa iz B<sup>+</sup>-stabla

brisanje zapisa s ključem 39



# Brisanje zapisa iz B<sup>+</sup>-stabla

brisanje zapisa s ključem 39



za vježbu:

u stablu na slici obrisati zapis s ključem 13

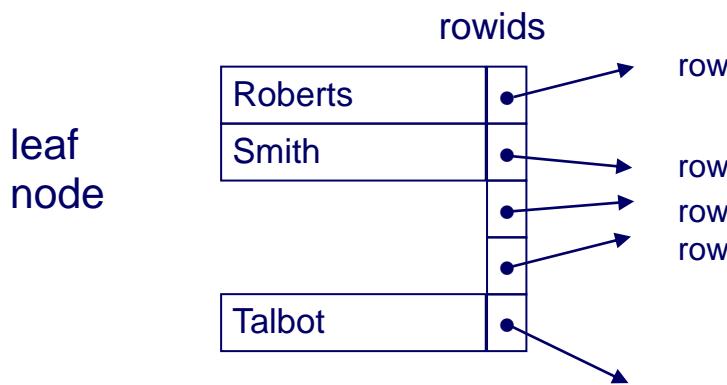
u stablu na slici obrisati zapis s ključem 48

u stablu na slici obrisati zapis s ključem 17 i nakon toga zapis s ključem 13

rezultat je balansirano stablo manje dubine

# Zapisi podataka s jednakim vrijednostima ključa

- *non-unique index*
- opisani algoritmi za B-stablo se moraju modificirati. Problemu se pristupa na različite načine. Jedan od načina jest da se u listovima omogući pohrana više od jednog pokazivača na zapise s podacima. Npr:



# Efikasnost operacije pretrage u B<sup>+</sup>-stablu

---

- pretpostavka: stablo reda **n** sadrži kazaljke na **m** zapisa podataka
- **n** se odabire tako da se sadržaj čvora može smjestiti u jedan fizički blok  
⇒ za dohvat **jednog** čvora potrebna je **jedna** U/I operacija
- broj U/I operacija u stablu pri traženju zapisa ovisi o broju razina u stablu jer se pri dohvatu zapisa mora obaviti po jedna U/I operacija za svaki čvor B-stabla na putu od korijena do lista
- B-stablo ima najveći broj razina onda kada su čvorovi najmanje popunjeni  
⇒ moguće je odrediti koliko će U/I operacija biti potrebno obaviti u najlošijem slučaju

# Efikasnost operacije pretrage u B<sup>+</sup>-stablu

- **Primjer:** za broj n-torki  $m = 1\ 000\ 000$ , za red stabla  $n = 70$ , pokazat će se da je ukupni broj razina (uključujući i razinu korijena) u najlošijem slučaju jednak 4:

1.  točno 1 čvor, najmanje 2 kazaljke
2.  najmanje 2 čvora, najmanje 70 kazaljki
3.  najmanje 70 čvorova, najmanje 2 450 kazaljki
4.  najmanje 2 450 čvorova, najmanje 85 750 kazaljki
5.  najmanje 85 750 čvorova, najmanje 3 001 250 kazaljki

- B<sup>+</sup>-stablo koje bi imalo ukupno 5 razina, moralo bi imati **najmanje 3 001 250** kazaljki na zapise. To znači da B-stablo reda 70 čije kazaljke u listovima pokazuju na 1 000 000 n-torki smije imati najviše 4 razine.  
⇒ Za dohvatzanja prema vrijednosti ključa potrebno je najviše 5 U/I operacija (4 U/I operacije za dohvatzanje lista u kojem se nalazi kazaljka na zapis + 1 U/I operacija za dohvatzanje bloka s podacima)
- **često se korijen i njegova djeca zadržavaju u međuspremniku SUBP-a**

# Efikasnost operacije pretrage u B<sup>+</sup>-stablu

---

- kako na općeniti način izračunati dubinu stabla u najgorem slučaju
- pretpostavka: stablo reda **n** sadrži kazaljke na **m** zapisa podataka
- stablo će imati najveći broj razina ako su čvorovi najmanje popunjeni
  - najmanja popunjenošć korijena je **2**
  - najmanja popunjenošć internog čvora:  $\lceil n / 2 \rceil$
  - najmanja popunjenošć lista:  $\lceil (n - 1) / 2 \rceil \approx \lceil n / 2 \rceil$ , za dovoljno veliki **n**
- u korijenu (1. razina) ima 1 čvor i najmanje **2** kazaljke
- na 2. razini ima najmanje 2 čvora i zato najmanje  $2 \cdot \lceil n / 2 \rceil$  kazaljki
- u čvorovima 3. razine ima najmanje  $2 \cdot \lceil n / 2 \rceil \cdot \lceil n / 2 \rceil$  kazaljki
- u čvorovima i-te razine ima najmanje  $2 \cdot \lceil n / 2 \rceil^{i-1}$  kazaljki
- za broj zapisa u podatkovnim blokovima stabla koje ima **d** razina vrijedi:
  - $m \geq 2 \cdot \lceil n / 2 \rceil^{d-1}$
- iz toga slijedi
  - $d \leq \log_{\lceil n/2 \rceil} (m / 2) + 1$
- **Primjer:** za  $m = 1\ 000\ 000$  zapisa,  $n = 70$ , ukupni broj razina (uključujući i razinu korijena) u najgorem slučaju je 4

# Popunjenošt B<sup>+</sup>-stabla u stacionarnom stanju

---

- analitički je dokazano i simulacijama potvrđeno:
  - ako se nad B<sup>+</sup>-stabлом obavi veliki broj operacija unosa i brisanja (slučajno odabranih vrijednosti ključeva), tada će popunjenošt stabla biti  $\approx \ln 2 \approx 69\%$
- zadatak za vježbu: grubo procijeniti kolika je (u najlošijem slučaju) popunjenošt B<sup>+</sup>-stabla nakon što se u prazno stablo unese velik broj zapisu poredanih prema ključu

# Efikasnost ostalih operacija u B<sup>+</sup>-stablu

---

- na efikasnost operacija unosa i brisanja utječe trošak reorganizacije B-stabla
- u rijetkim slučajevima dijeljenje ili spajanje čvorova se propagira sve do korijena stabla
- u B<sup>+</sup>-stablima koja imaju razumno velik red stabla (npr. 10 ili više) takvi su slučajevi rijetki
  - npr. za red stabla 101: koliko puta se list može podijeliti (u najgorem slučaju) ako se u stablo unese 100 zapisa
    - najgori slučaj: list je bio pun, 100 zapisa
    - 1. uneseni zapis dijeli list na dva: list od 50 i list od 51 zapisa
    - 2. - 50. uneseni zapis neće uzrokovati dijeljenje lista
    - 51. zapis uzrokuje dijeljenje lista: listovi s 50 zapisa i 51 zapisom
    - 52. - 100. uneseni zapis neće uzrokovati dijeljenje lista
  - 100 unesenih zapisa uzrokovalo je samo dvije reorganizacije na razini listova (eventualno, ali vrlo rijetko, reorganizacija će biti potrebna i na višim razinama)

# Efikasnost ostalih operacija u B<sup>+</sup>-stablu

- može se zaključiti da će se za operacije unosa ili brisanja najčešće koristiti samo dvije U/I operacije više nego kod pretrage
  - **pretraga (što treba dodatno obaviti nakon pronalaska lista):**
    - *lookup* u *heap* datoteku - 1 U/I operacija
  - **unos ili brisanje (što treba dodatno obaviti nakon pronalaska lista):**
    - *lookup* u *heap* datoteku (1 U/I operacija)
    - *output data block* (1 U/I operacija)
    - *output leaf block* (1 U/I operacija)
- originalni algoritam za B<sup>+</sup>-stablo pri brisanju obavlja reorganizaciju onda kada broj zapisa u čvoru padne ispod polovice kapaciteta čvora
  - strategija *merge-at-half*
- većina današnjih sustava reorganizaciju pri brisanju provodi tek onda kada u čvoru ne preostane niti jedan zapis
  - strategija *merge-at-empty*
  - značajno niža frekvencija reorganizacija uz neznatno lošije iskorištenje prostora

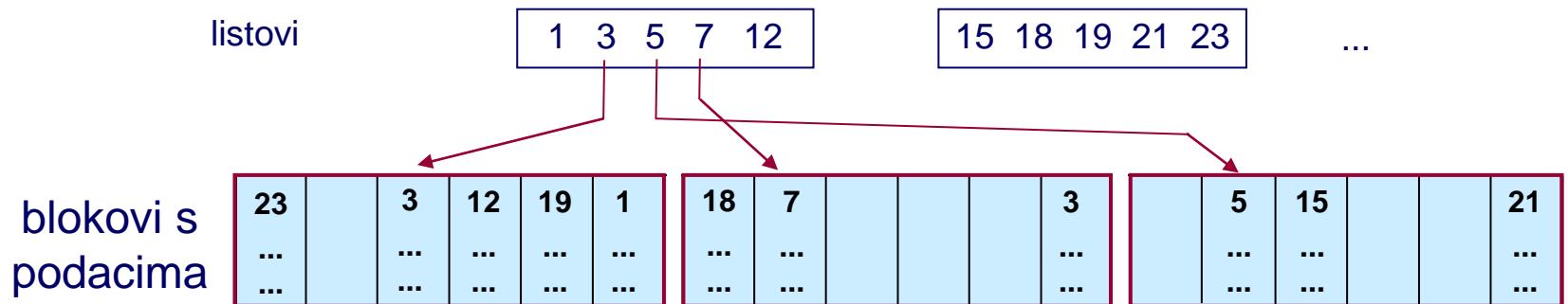
# Trošak korištenja indeksa

- Izgradnja i održavanje indeksa može biti skupo (prostor za pohranu, CPU, U/I aktivnosti). Administrator sustava mora osigurati da dobici pri korištenju indeksa nadmašuju gubitke koji nastaju zbog održavanja indeksa
- gruba procjena: obavljanje DML operacije INSERT ili DELETE zbog koje je potrebno obaviti održavanje jednog indeksa rezultira trostruko većim dodatnim utroškom resursa. Npr, INSERT ili DELETE operacija nad relacijom za koju su izgrađena tri indeksa je desetak puta skuplja (CPU, U/I) od iste operacije nad relacijom za koju nije izgrađen niti jedan indeks
- posebnost UPDATE operacije: primjer
  - relacija  $r(A, B)$ , za atribut A je kreiran indeks
  - **UPDATE r SET B=x WHERE A=y**
    - nema troška održavanja indeksa, dobar odabir
  - **UPDATE r SET A=x WHERE B=y**
    - indeks ne donosi dobitak, postoji samo trošak održavanja indeksa, loš odabir

# Efikasnost dohvata iz intervala vrijednosti

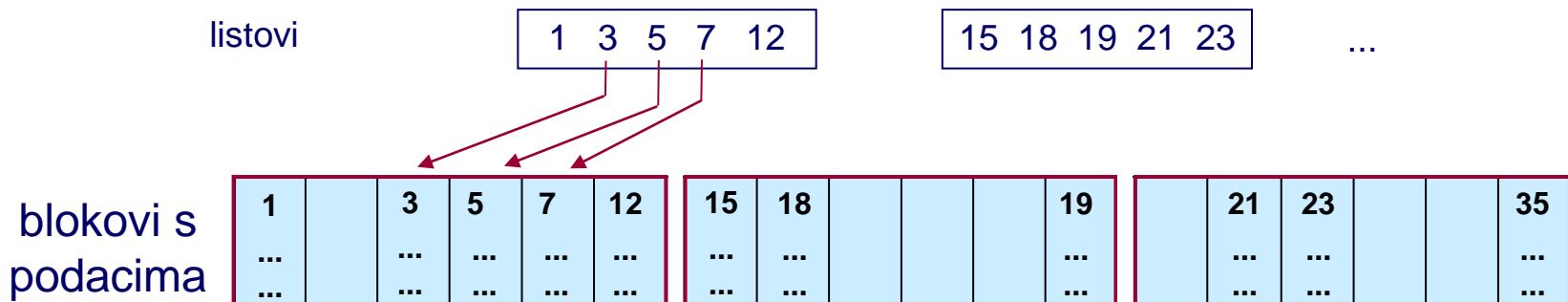
## Indeks s poredanim zapisima podataka (*IBM Informix: Cluster index*)

- dohvat podataka u intervalu vrijednosti [3, 7]



- Cluster index*: podaci (u podatkovnim blokovima) su poredani prema ključu

```
CREATE CLUSTER INDEX idxName ON tablename (colname, ...);
```



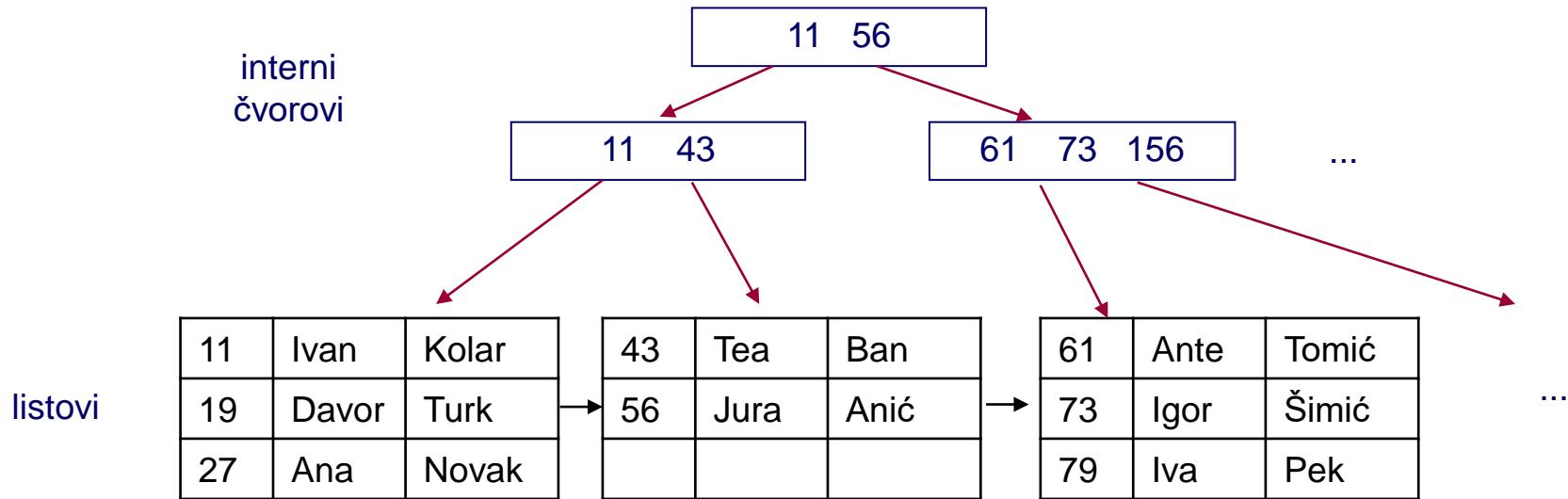
- usporediti broj U/I operacija u odnosu na non-clustered indekse
- za jednu relaciju moguć je najviše jedan indeks s poredanim zapisima podataka

# Efikasnost dohvata iz intervala vrijednosti

## Indeks sa zapisima podataka u listovima (SQL Server: *Clustered index*)

- naredba kreira indeksnu strukturu sličnu B<sup>+</sup>-stablu. Podaci (n-torce) su smješteni u listovima stabla

```
CREATE CLUSTERED INDEX idxName ON tablename (colname, ...);
```



- očito, i ovdje je moguć samo jedan *clustered index* po relaciji
- za razliku od *non-clustered* indeksa, gdje n-torka nakon promjene ključa ostaje u istom podatkovnom bloku (treba "popraviti" samo interne čvorove i listove), kod *clustered* indeksa će trebati fizički preseliti cijelu n-torku
  - npr. kod promjene matičnog broja 27 u 58

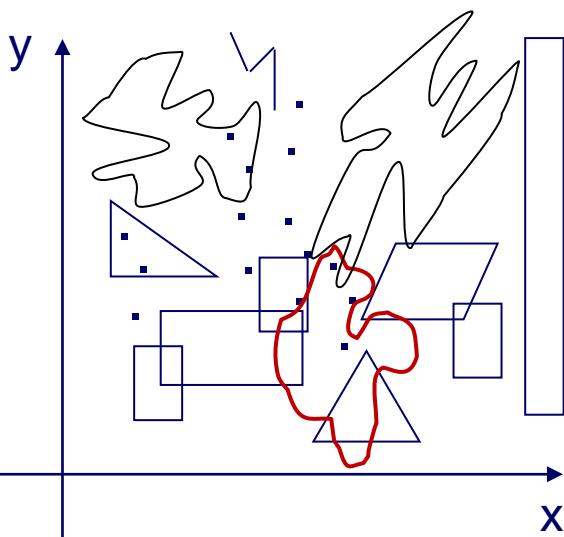
# Efikasnost dohvata iz intervala vrijednosti

## Indeks sa zapisima podataka u listovima (SQL Server: *Clustered index*)

- preporuča se koristiti
  - za atribute ili skupove atributa (složeni ili kompozitni indeks) prema kojima se često obavlja dohvatzanje većeg broja n-torki iz intervala vrijednosti ili prema kojima se često obavlja sortiranje. Npr. korisno je kreirati *clustered* indeks za jmbag ako se često obavljaju upiti oblika
    - `SELECT * FROM student WHERE jmbag BETWEEN 100 AND 200;`
    - `SELECT * FROM student ORDER BY jmbag;`
- ne preporuča se koristiti
  - za atribute čije se vrijednosti često mijenjaju (to vrijedi i za ostale indekse, ali ovdje naročito)
  - naročito ne za atribute ili skupove atributa čije vrijednosti zauzimaju *puno* prostora (*wide keys*). Npr. kompozitni indeks za `ime_stud + prez_stud`
    - ostali indeksi kreirani nad tom relacijom, umjesto uobičajene kazaljke na blok u kojem se nalazi n-torka, moraju koristiti ključ *clustered* indeksa. Zašto: jer n-torka može promijeniti svoju fizičku poziciju (preseliti se u neki drugi list) kada se promijeni vrijednost ključa *clustered* indeksa. Stoga je drugi indeksi mogu izravno adresirati jedino preko vrijednosti ključa *clustered* indeksa i uz upotrebu tog indeksa.

# Višedimenzionalni indeksi (*Multidimensional indexes*)

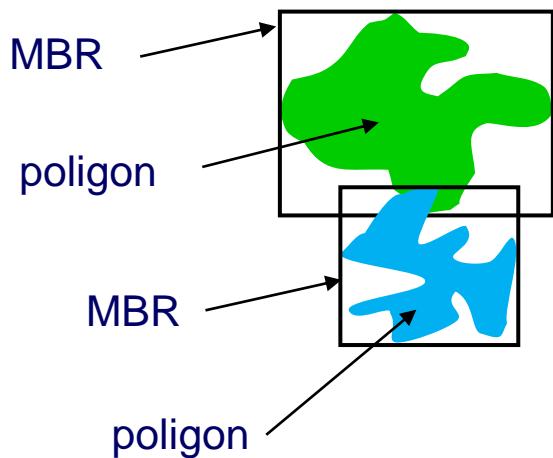
- U B-stablu ključ pretrage je uvijek neka skalarna vrijednost - zadana vrijednost ključa pretrage uspoređuje se s vrijednostima ključeva u zapisima B-stabla i tako se pronalaze blokovi u kojima se nalazi traženi zapis (n-torka)
- ako umjesto skalarnih vrijednosti treba indeksirati složene ili višedimenzionalne vrijednosti, indeksi temeljeni na B-stablu više neće biti dovoljni. Potrebni su tzv. višedimenzionalni indeksi



- R-stabla
- točke (*point data*), linije (*lines*), regije (*region data*)
- primjeri prostornih upita (*spatial query*)
  - dohvati sve objekte koji se nalaze na zadanoj udaljenosti od zadanog objekta
  - dohvati objekte čije se granice sijeku sa zadanim objektom

# R-stablo

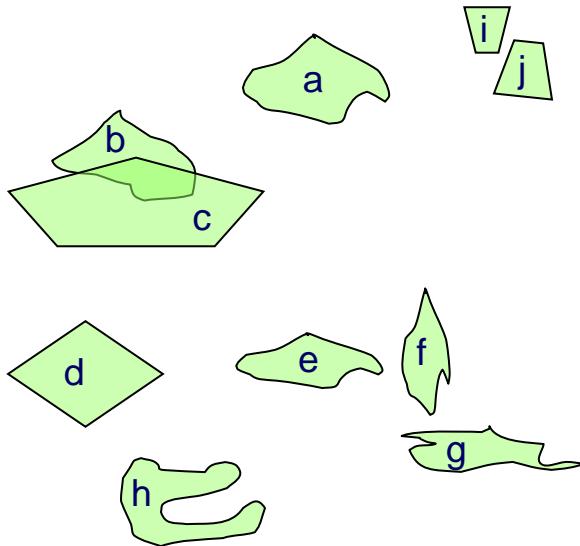
- prema osnovnoj ideji slično je B<sup>+</sup>-stablu
- R-stablo je balansirano stablo u kojem su ključevi uz kazaljke u stablu i ključevi pretrage minimalni granični okviri, *minimum bounding rectangle*, MBR
  - minimalni pravokutnik (2D) ili minimalni kvadar (3D) koji u cijelosti obuhvaća lik (2D) ili tijelo (3D)



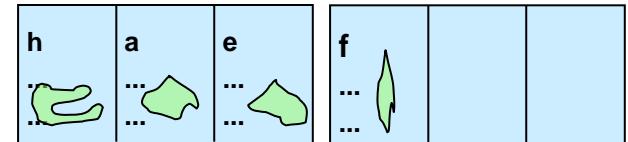
- MBR: x-low, y-low; x-high, y-high

# R-stablo

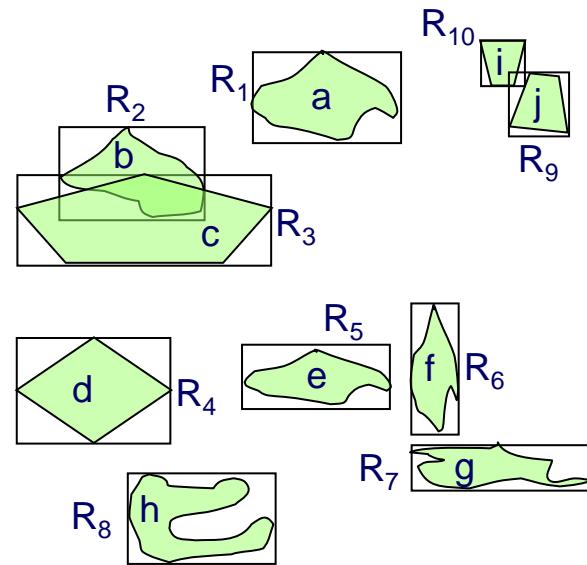
- primjer: izgraditi R-stablo reda 3 za objekte na sljedećoj slici



blokovi s podacima



- odrediti MBR-ove za sve objekte



# R-stablo

- što sadrži svaki pojedini zapis u listu R-stabla?

jedan zapis iz  
lista R-stabla



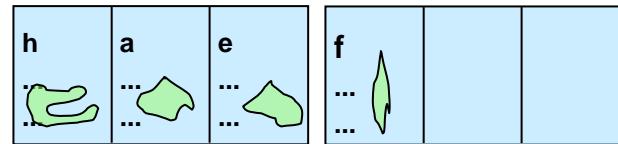
jedan zapis iz  
lista R-stabla



jedan zapis iz  
lista R-stabla



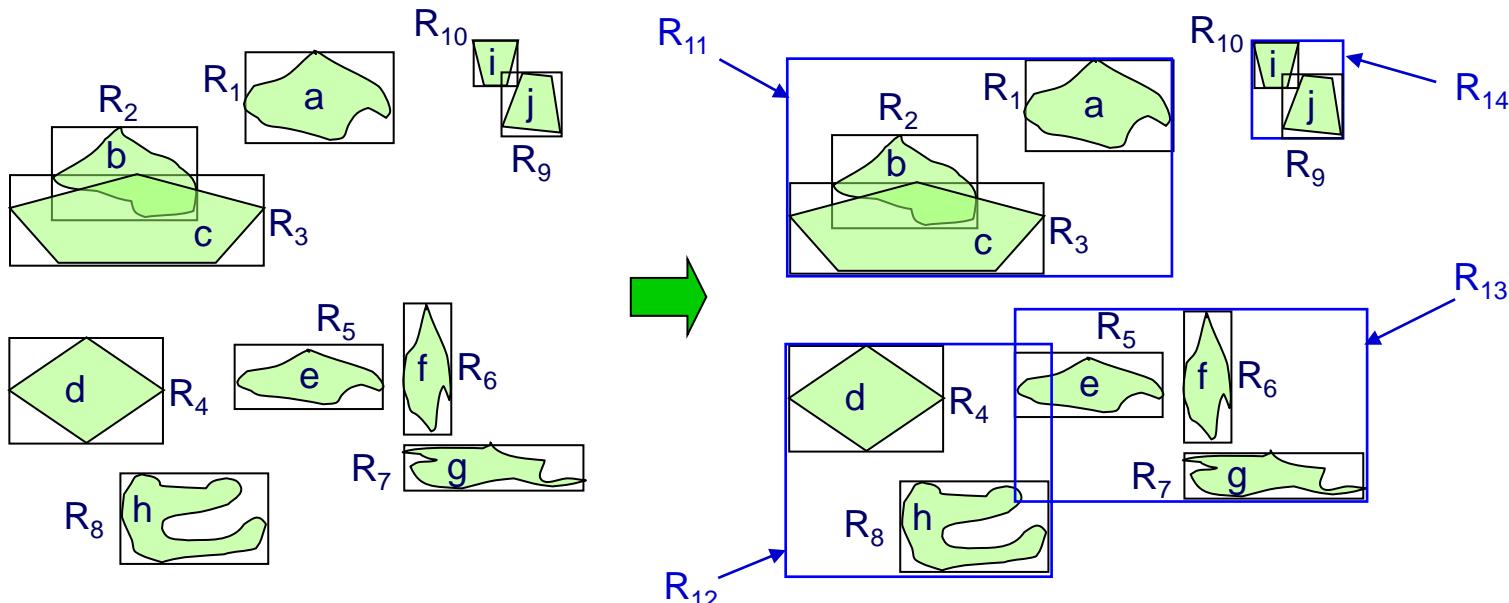
blokovi s  
podacima



- jedan zapis u listu R-stabla sadrži podatke o MBR-u za objekt (x-low, y-low; x-high, y-high) i pokazivač na blok podataka (ili eventualno dodatno i poziciju zapisa u bloku) u kojem se nalaze podaci o objektu za taj MBR
  - slično kao što jedan zapis u listu B-stabla sadrži vrijednost ključa i pokazivač na blok podataka (ili eventualno dodatno i poziciju zapisa u bloku) u kojem se nalaze podaci za n-torku s dotičnim ključem

# R-stablo

- primjer (nastavak):
- grupirati međusobno bliske MBR-ove u po jedan MBR koji ih prekriva. Broj MBR-ova u jednoj grupi ograničen je kapacitetom čvora, tj. redom stabla
  - R-stablo je balansirano stablo, minimalna popunjenošć čvorova (osim korijena) je 50%

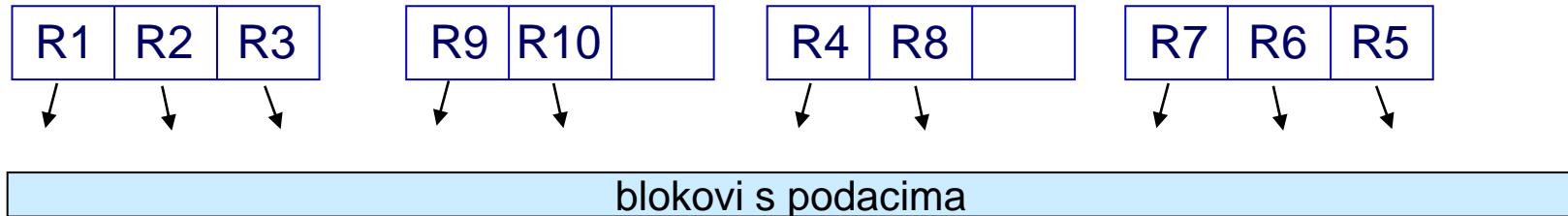


- način grupiranja nije jednoznačan, ali nastoji se minimizirati površina MBR-a

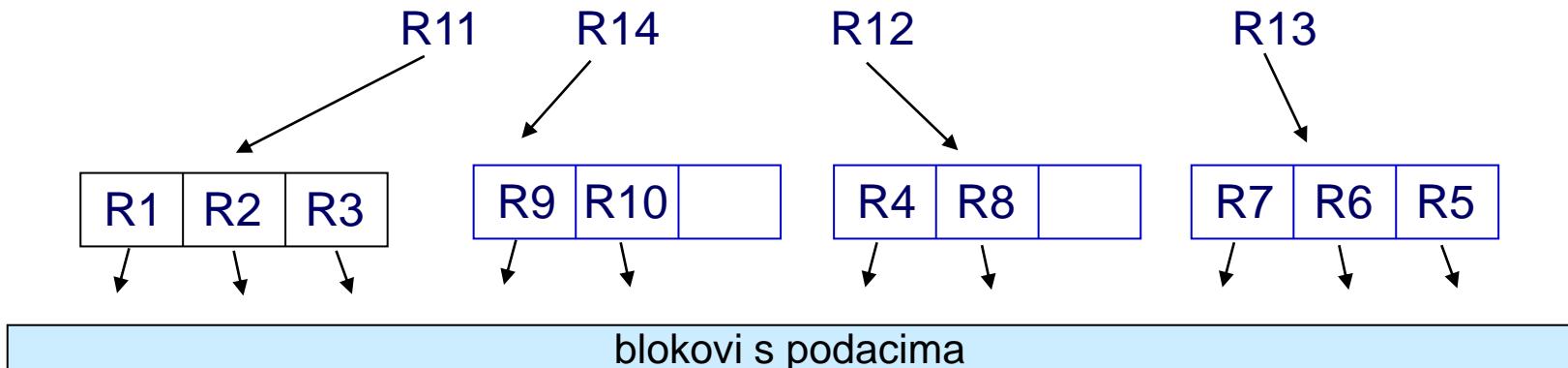
\* na sljedećim slikama, radi preglednosti, objekti nisu ucrtani u MBR-ove

# R-stablo

- primjer (nastavak):
- zapisi za MBR-ove jedne grupe upisuju se u jedan list

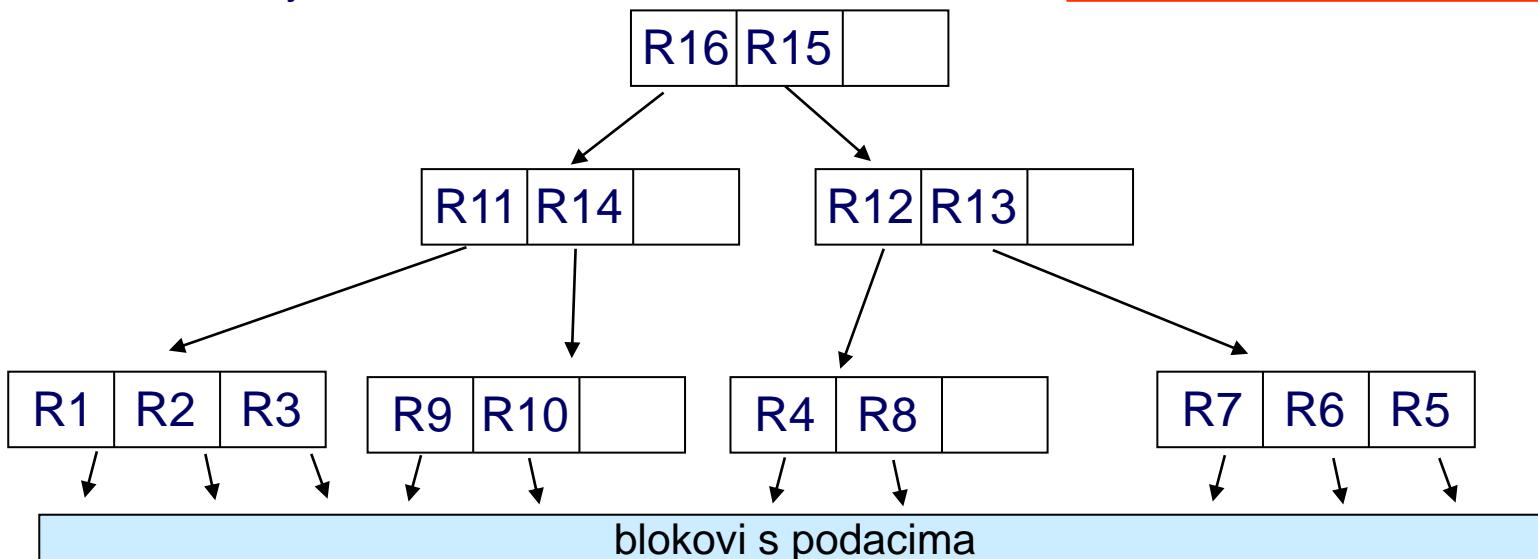
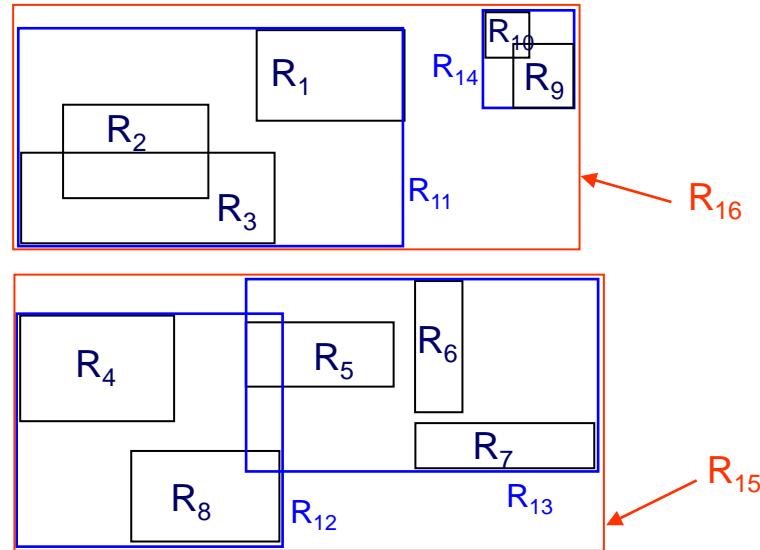


- na svaki list pokazuje po jedan zapis s više razine stabla. U zapisu se nalaze podaci o MBR-u (x-low, y-low; x-high, y-high) koji obuhvaća grupu MBR-ova iz lista i pokazivač na taj list



# R-stablo

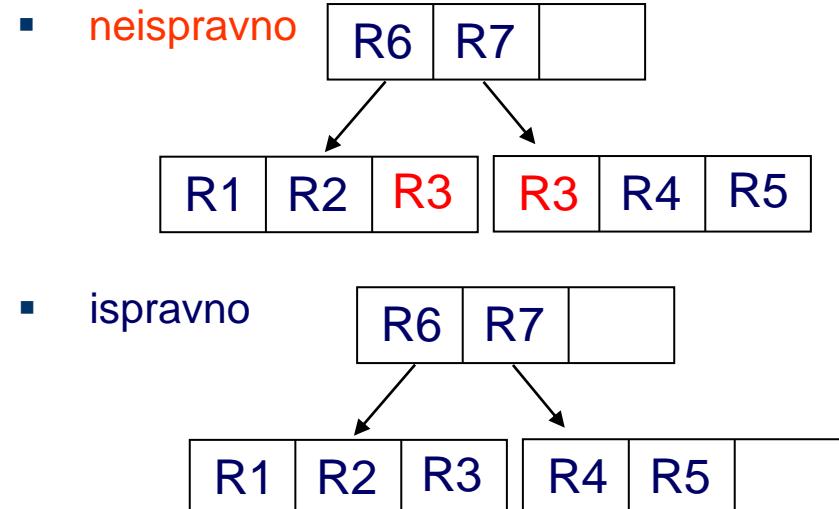
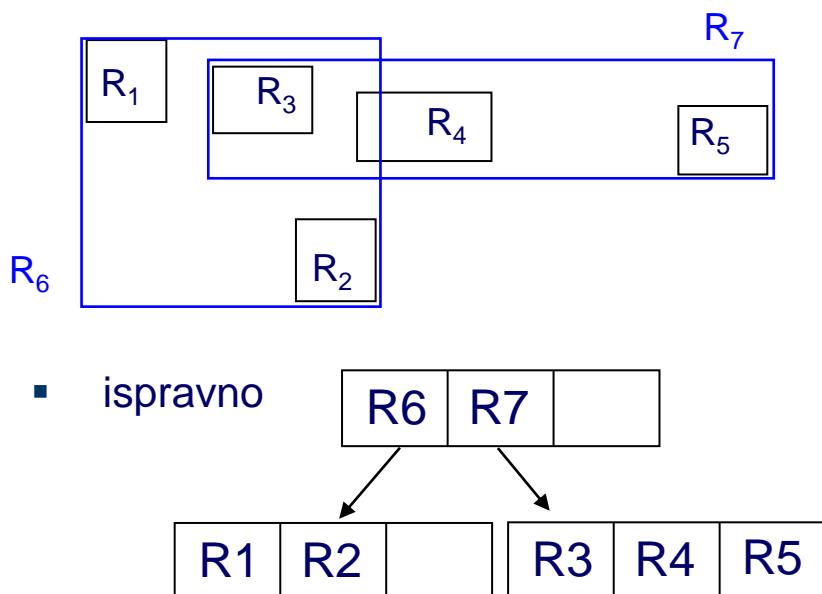
- primjer (nastavak):
- postupak se ponavlja rekurzivno. MBR-ovi s razine iznad lista grupiraju se u grupe međusobno bliskih MBR-ova, zapisi jedne grupe upisuju se u jedan interni čvor, na višoj razini se za svaku grupu formira jedan zapis (MBR + kazaljka) itd.
- postupak se ponavlja dok se ne dođe do razine korijena R-stabla



# R-stablo

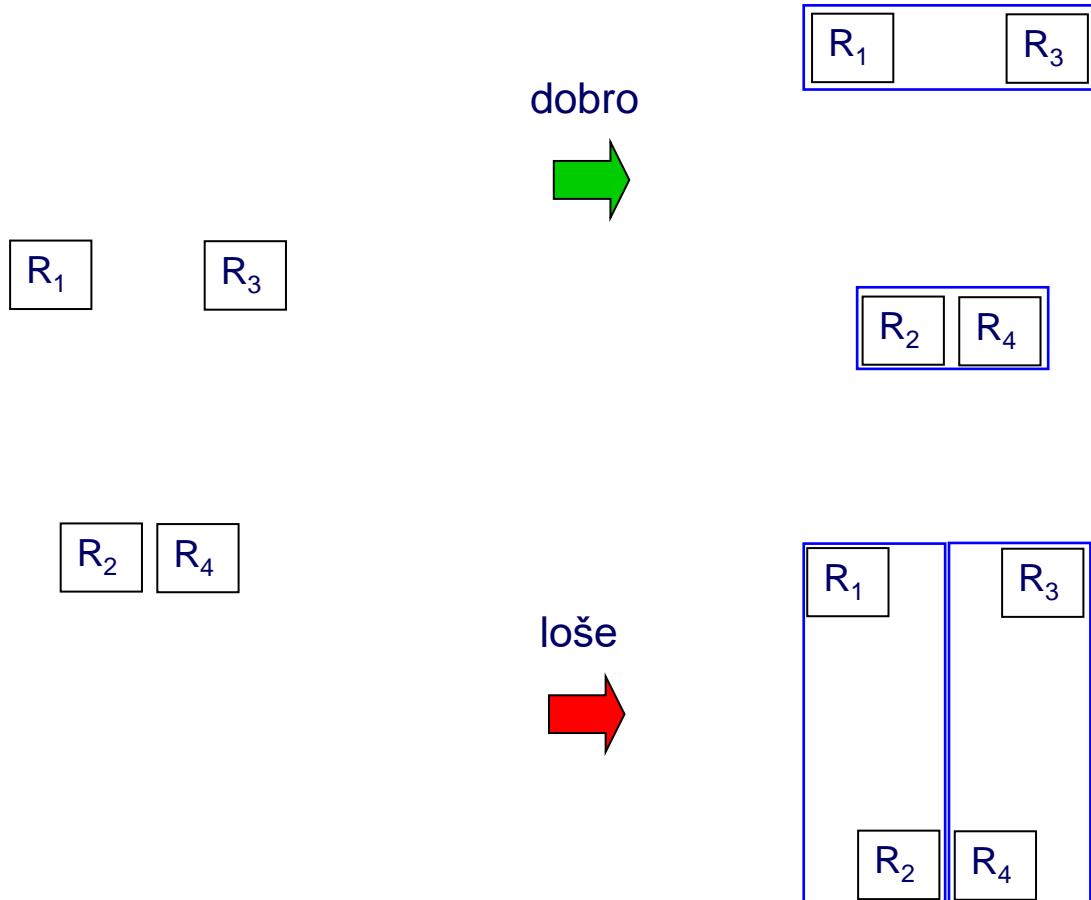
- uočiti:

- MBR-ovi iste razine smiju se preklapati (ali čim manje - to bolje). Npr. preklapaju se R12 i R13 u prethodnom primjeru, ili R6 i R7 na slici dolje
- zapis za jedan MBR u R-stablu smije se nalaziti u samo jednom čvoru
  - ako se  $MBR_X$  i  $MBR_Y$  na  $i$ -toj razini preklapaju, moguće je da će neki  $MBR_Z$  na razini  $i+1$  biti obuhvaćen i sa  $MBR_X$  i sa  $MBR_Y$ . U takvom slučaju,  $MBR_Z$  se dodaje ili u čvor na kojeg pokazuje  $MBR_X$  ili u čvor na kojeg pokazuje  $MBR_Y$  (ne u oba)

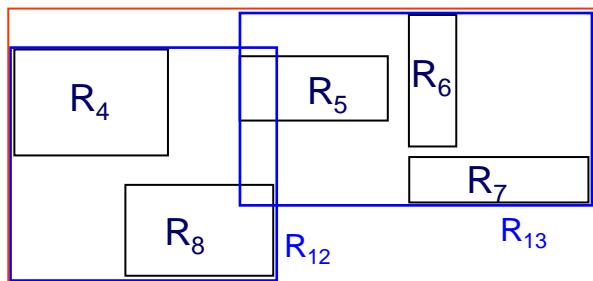
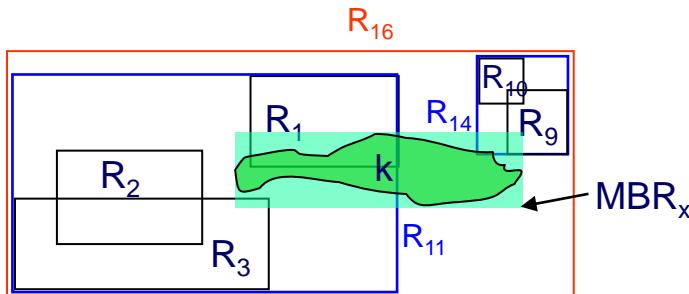


# R-stablo

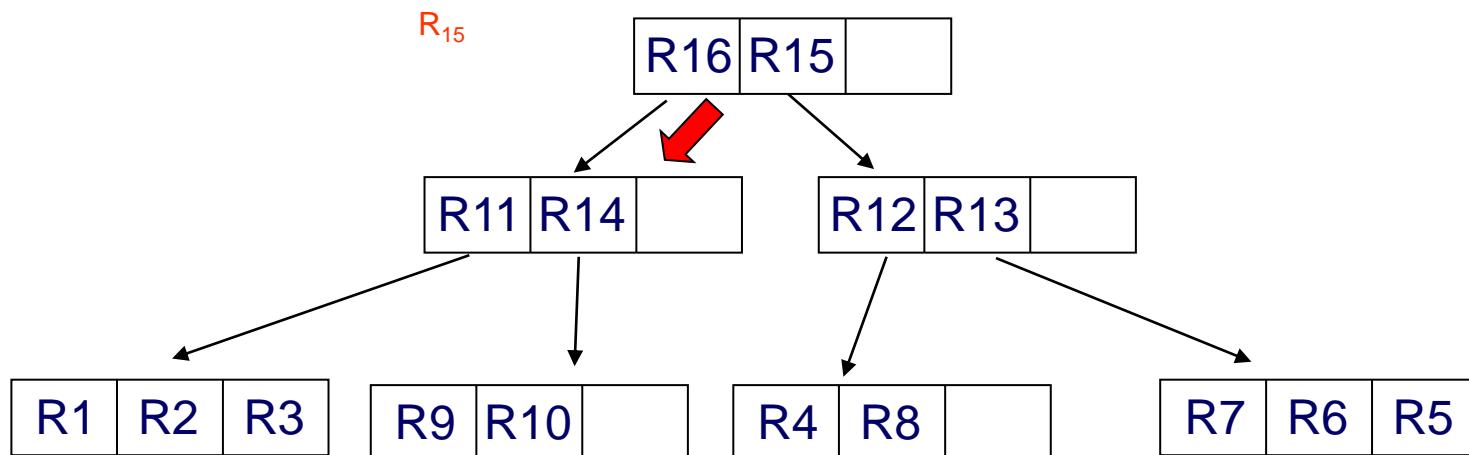
- algoritmi za grupiranje minimalnih graničnih okvira su relativno kompleksni
  - nastojati minimizirati površinu MBR-ova
- primjer dobrog i lošeg odabira



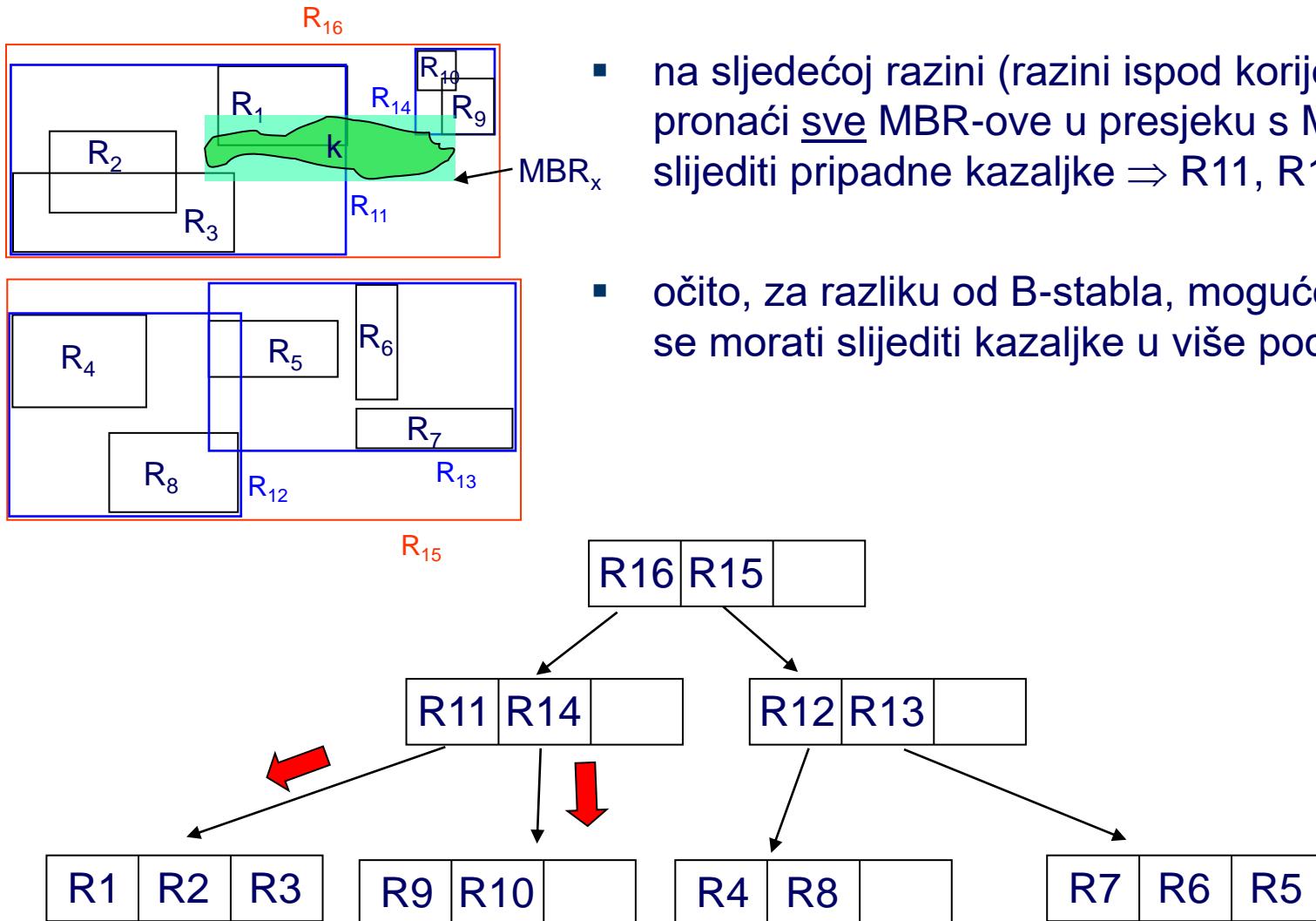
# R-stablo - pretraga



- primjer prostornog upita: dohvatiti sve objekte koji se sijeku sa zadanim objektom  $k$  (pripadni MBR objekta  $k$  je  $MBR_x$ )
- pretraga kreće od korijena
- u korijenu naći sve MBR-ove u presjeku s  $MBR_x$  i slijediti pripadne kazaljke  $\Rightarrow R_{16}$
- ponavljati na svakoj razini do lista

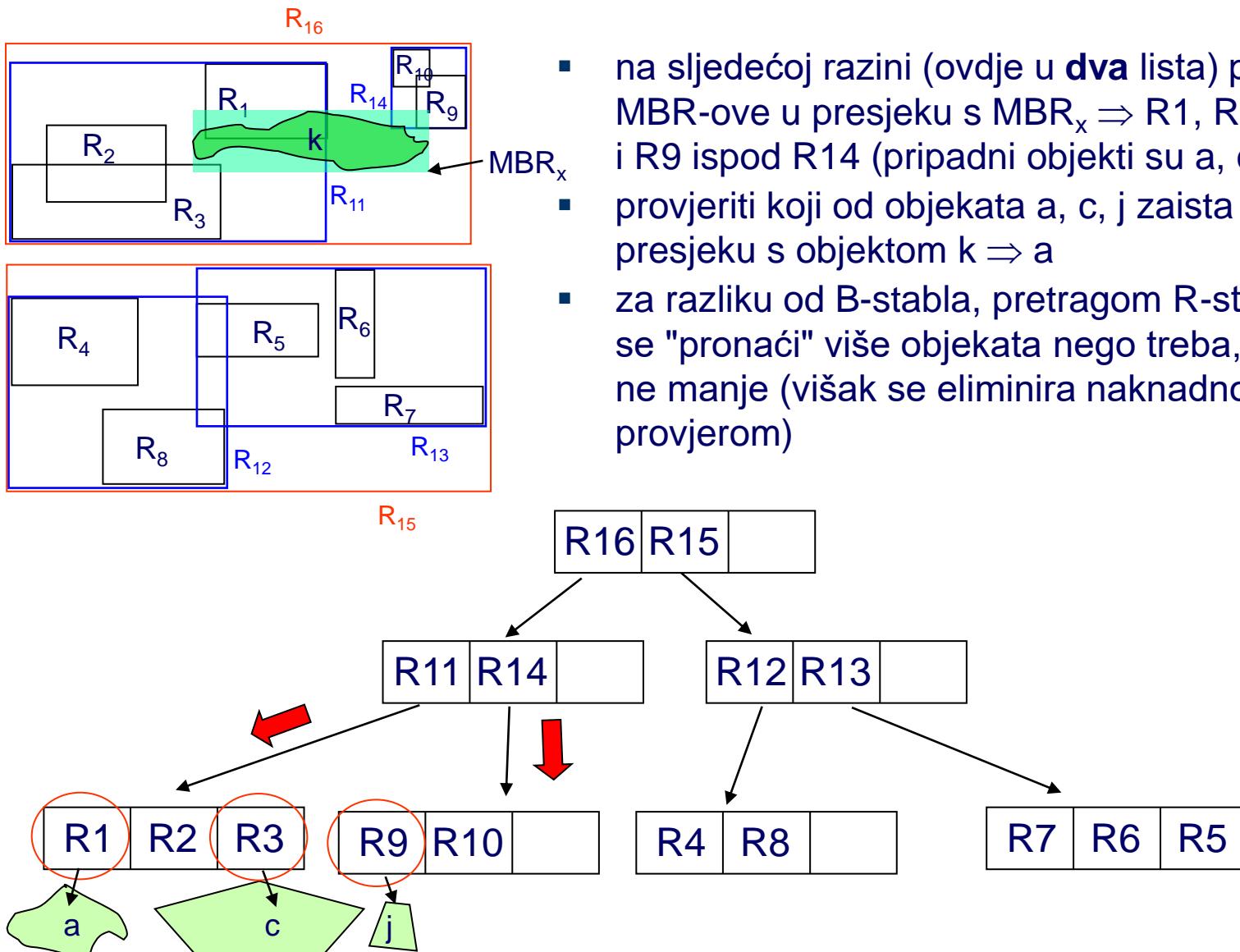


# R-stablo - pretraga



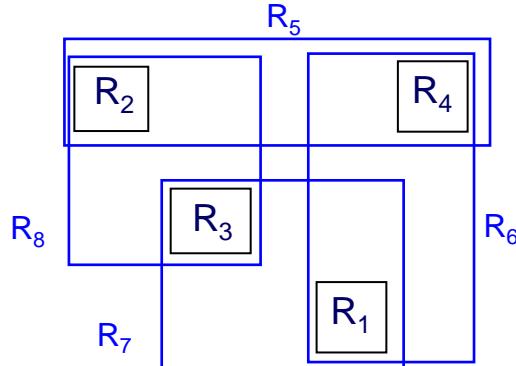
- na sljedećoj razini (razini ispod korijena) pronaći sve MBR-ove u presjeku s  $MBR_x$  i slijediti pripadne kazaljke  $\Rightarrow R11, R14$
- očito, za razliku od B-stabla, moguće je da će se morati slijediti kazaljke u više podstabala

# R-stablo - pretraga

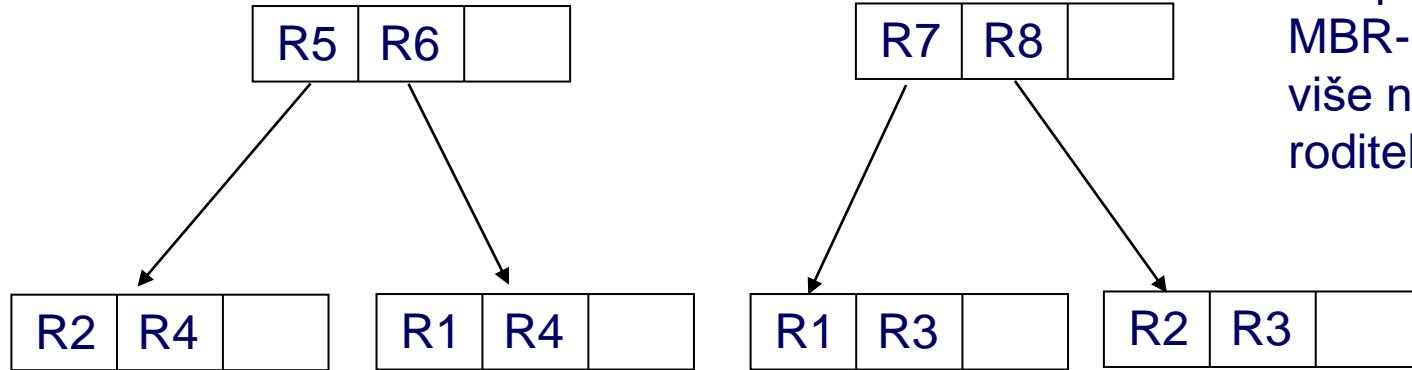


# R-stablo - primjer

- svaki roditelj kompletno prekriva svoju djecu
  - MBR-ovi se smiju preklapati
- ALI
- ako se MBR-ovi preklapaju i neko dijete je prekriveno s dva (ili više MBR-ova), u stablu se dijete smije pridružiti **samo jednom** od roditelja koji ga prekrivaju

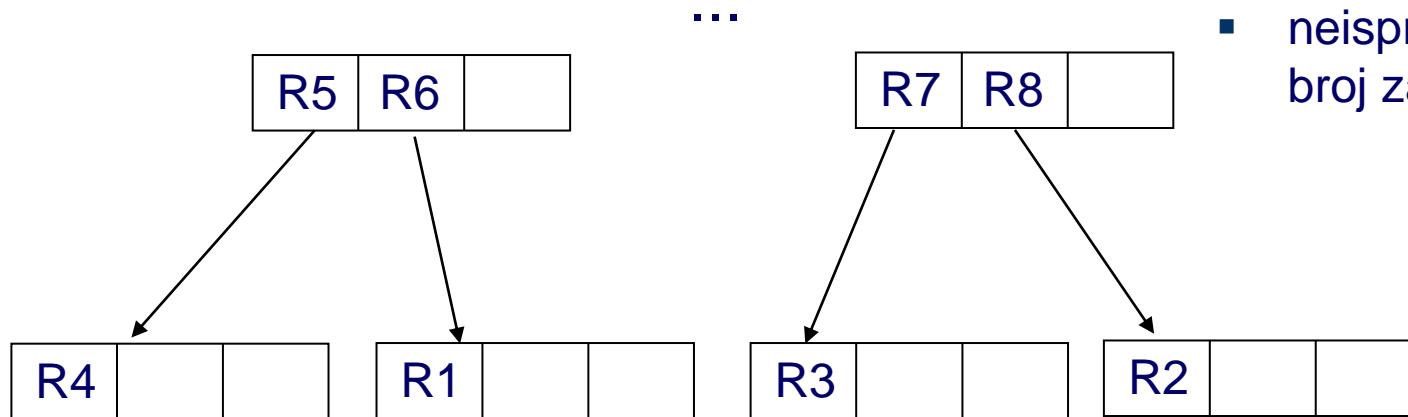
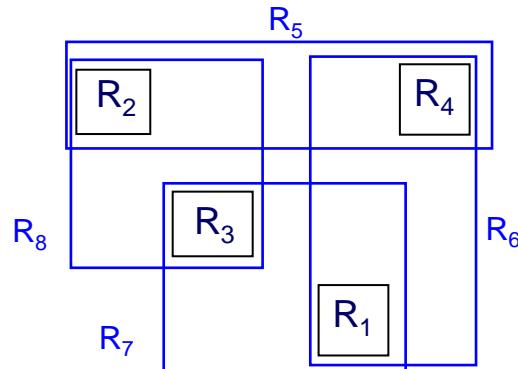


...



- neispravno - neki MBR-ovi su pridruženi više nego jednom roditelju

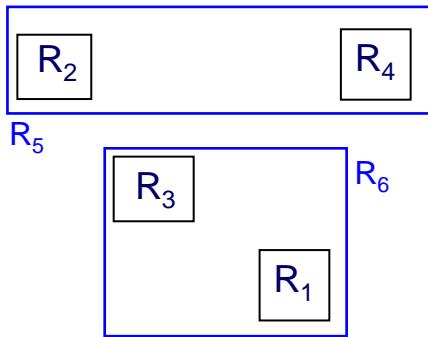
# R-stablo - primjer (nastavak)



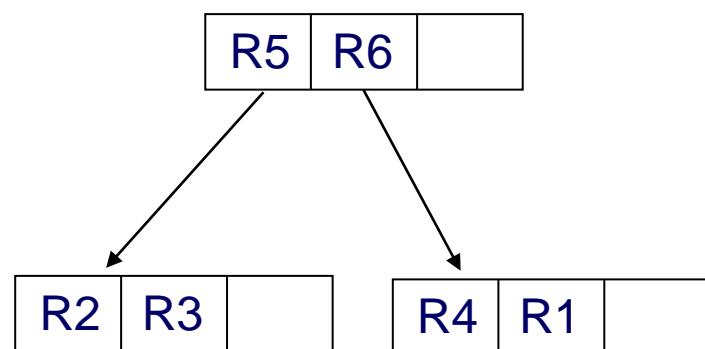
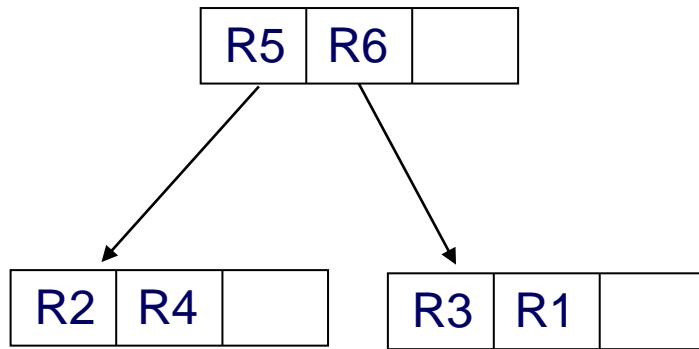
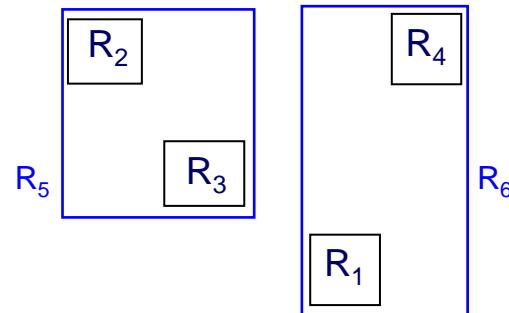
- neispravno - preveliki broj zapisa u listovima

# R-stablo - primjer (nastavak)

- ispravno



- ispravno



# Literatura:

---

- H. Garcia-Molina, J. Ullman, J. Widom. Database system implementation. Prentice-Hall International, Inc. Upper Saddle River, New Jersey. 2000.
- A. Silberschatz, H. Korth, S. Sudarshan. Database system concepts, 6th ed. McGraw-Hill. 2010.
- R. Elmasri, S. Navathe. Fundamentals of database systems, 3rd ed. Addison-Wesley Publishing Company. Reading, Massachusetts. 2000.

# **Sustavi baza podataka**

Predavanja

**3. Obavljanje upita**

**(1. dio)**

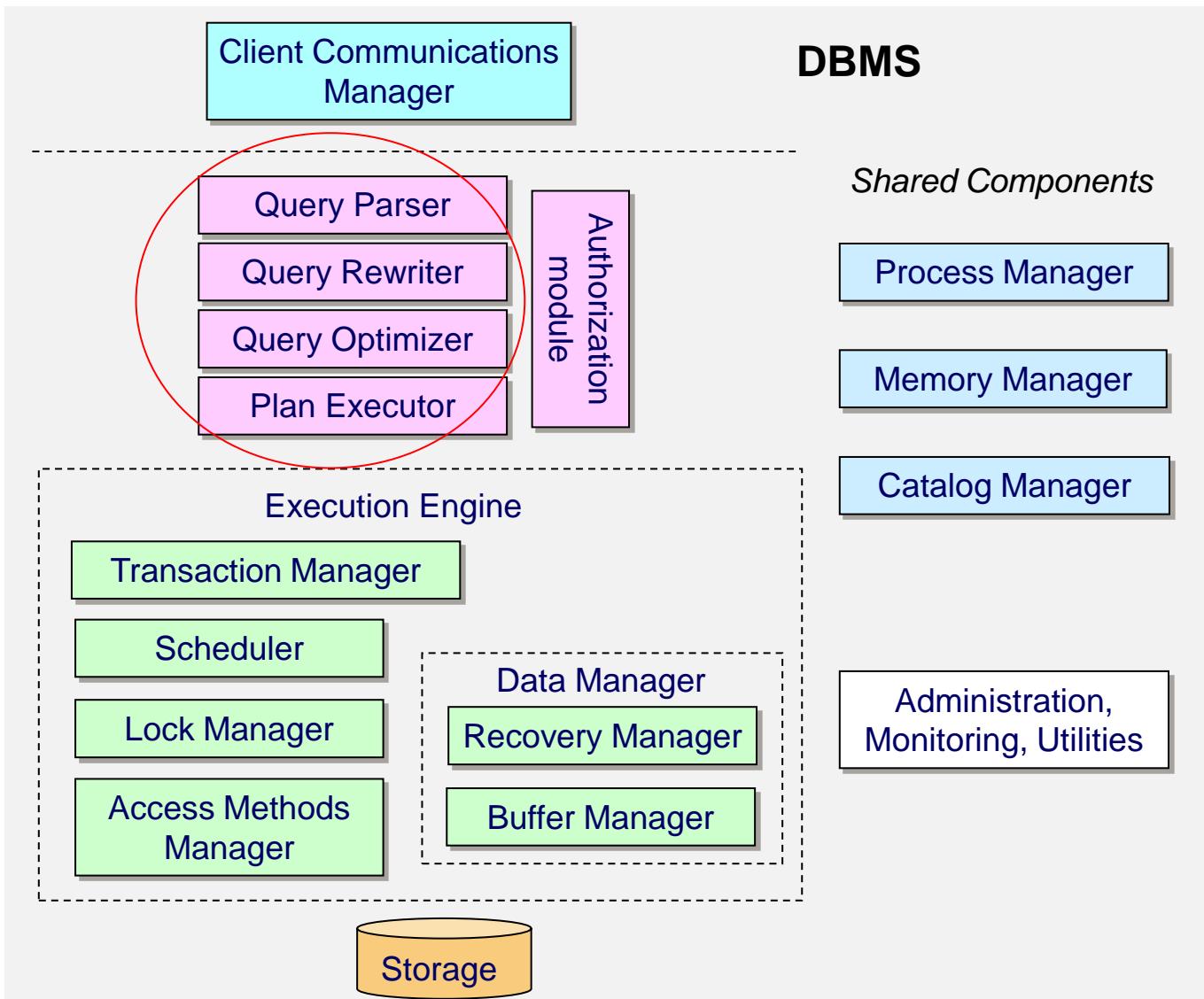
**ožujak 2023.**

# Obavljanje upita

---

- obavljanje upita (*query processing*)
  - skup aktivnosti vezanih uz ekstrakciju podataka iz baze podataka
- starije generacije sustava za upravljanje bazama podataka
  - mrežni, hijerarhijski model podataka
  - upitni jezik ugniježđen u proceduralni jezik treće generacije (npr. COBOL, PL/I). Navigacijski upitni jezici: *Network DML*, *Hierarchical DML*
    - odgovornost programera pri odabiru "optimalne" strategije ekstrakcije podataka
- moderni sustavi za upravljanje bazama podataka
  - relacijski i objektno-relacijski modeli podataka
  - SQL je neproceduralni jezik. Korisnik/programer definira *što* rezultat treba sadržavati, a ne *kako* do rezultata doći
    - SUBP odabire najpogodniju strategiju izvršavanja upita
    - oslobođa korisnika/programera odgovornosti (i potrebe da zna *kako*)
    - korisnik/programer nije u prilici odabrati lošu strategiju

# Komponente SUBP-a



# Obavljanje upita

Primjer:

```
SELECT DISTINCT imeStud, prezStud
  FROM stud
    JOIN ispit ON stud.mbr = ispit.mbrStud
    JOIN mjesto ON stud.pbrStan = mjesto.pbr
 WHERE nazMjesto = 'Zagreb'
   AND ocjena = 4;
```

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. DISTINCT
6. UNION
7. ORDER BY

$$\pi_{imeStud, prezStud}(\sigma_{nazMjesto='Zagreb' \text{ AND } ocjena=4}((stud \bowtie ispit) \bowtie mjesto))$$

```
SELECT DISTINCT imeStud, prezStud
  FROM mjesto CROSS JOIN stud
    CROSS JOIN ispit
 WHERE nazMjesto = 'Zagreb'
   AND ocjena = 4
   AND stud.pbrStan = mjesto.pbr
   AND ispit.mbrStud = stud.mbr;
```

$$\pi_{imeStud, prezStud}(\sigma_{nazMjesto='Zagreb' \text{ AND } ocjena=4 \text{ AND } pbrStan=pbr \text{ AND } mbrStud=mbr} (mjesto \times stud \times ispit))$$

SUBP će u oba slučaja obaviti istu operaciju:

$$\pi_{imeStud, prezStud}(\sigma_{ocjena=4}(ispit) \bowtie (stud \bowtie \sigma_{nazMjesto='Zagreb'}(mjesto)))$$

# Obavljanje upita

```
SELECT AVG(ocjena)
  FROM stud
    JOIN ispit ON stud.mbr = ispit.mbrStud
    JOIN mjesto ON stud.pbrStan = mjesto.pbr
 WHERE nazMjesto = 'Zagreb';
```

$$G_{\text{AVG}(\text{ocjena})}(\sigma_{\text{nazMjesto}='\text{Zagreb}'}(\text{mjesto}) \bowtie \text{stud} \bowtie \text{ispit})$$

pbr=pbrStan      mbr=mbrStud

card(mjesto) = 10 000  
card(stud) = 50 000  
card(ispit) = 1 000 000  
bez indeksa

→ 1.8 sec

Eksplicitni zahtjev optimizatoru za promjenom plana izvršavanja istog upita

- ORDERED: relacije spajati redoslijedom kojim su navedene u listi FROM

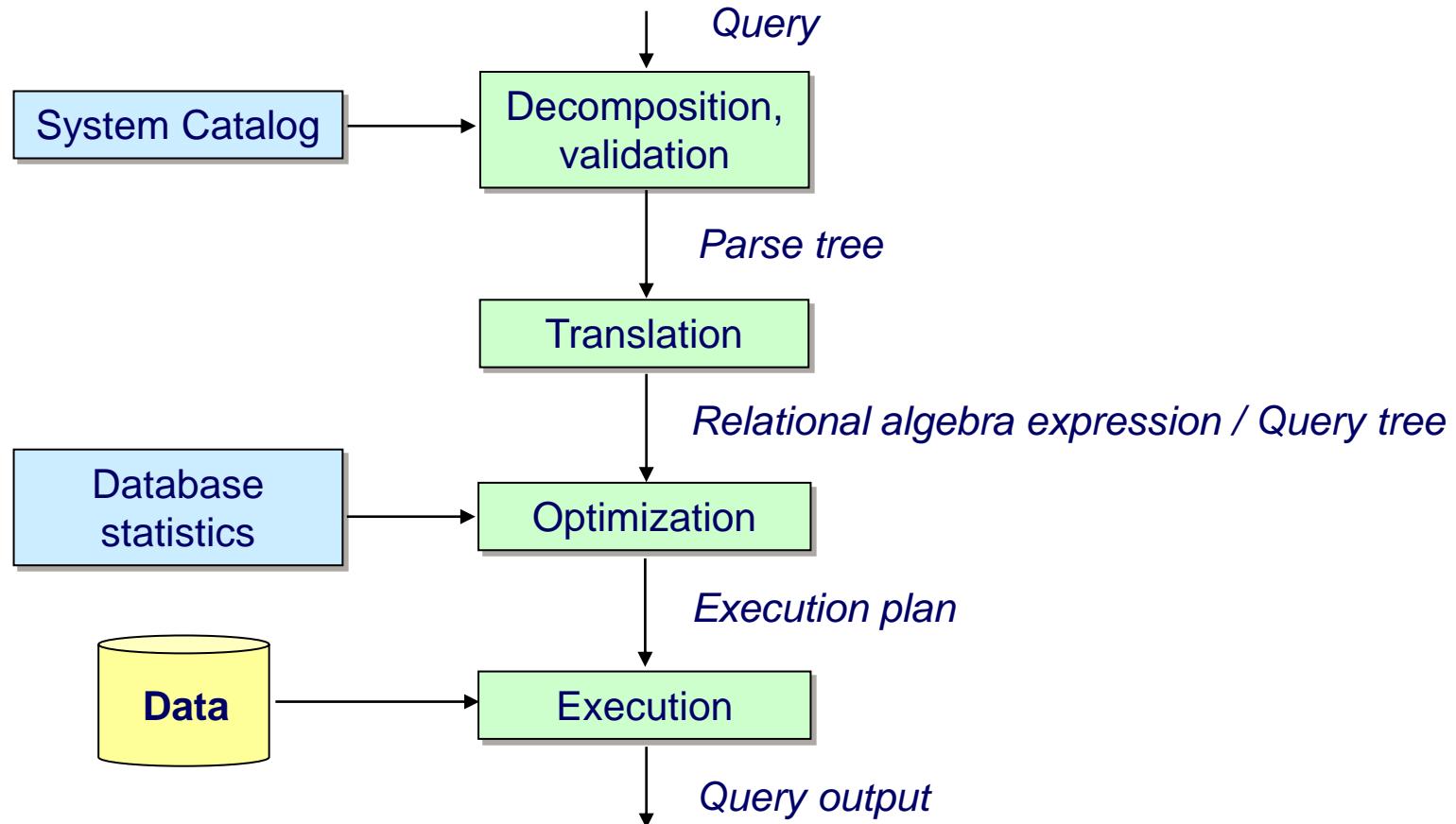
```
SELECT {+ORDERED} AVG(ocjena)
  FROM stud
    JOIN ispit ON stud.mbr = ispit.mbrStud
    JOIN mjesto ON stud.pbrStan = mjesto.pbr
 WHERE nazMjesto = 'Zagreb';
```

$$G_{\text{AVG}(\text{ocjena})}((\text{stud} \bowtie \text{ispit}) \bowtie \sigma_{\text{nazMjesto}='\text{Zagreb}'}(\text{mjesto}))$$

mbr=mbrStud      pbrStan=pbr

→ 10.5 sec

# Obavljanje upita



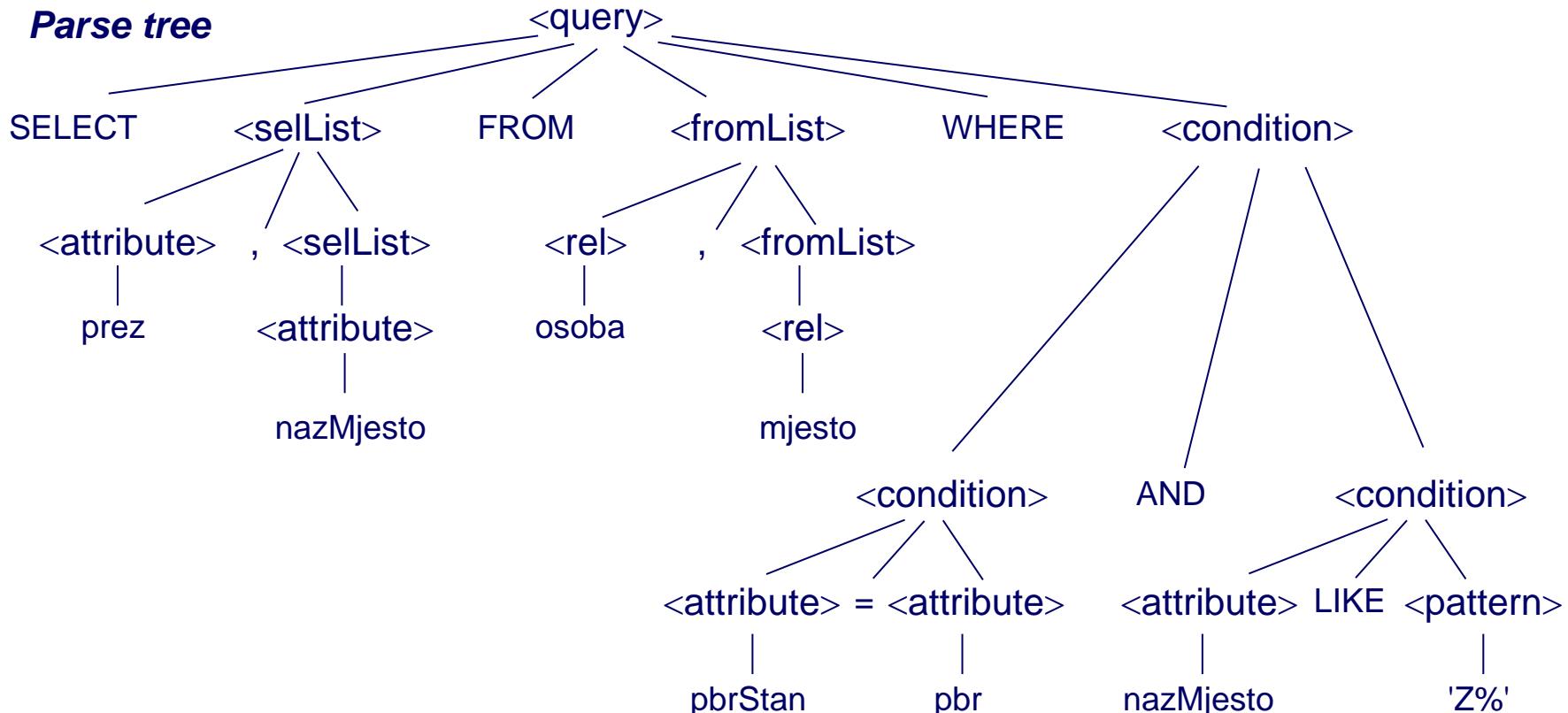
# Dekompozicija i validacija

- leksička i sintaktička analiza (tehnike prevođenja programskih jezika)

Primjer:

```
SELECT prez, nazMjesto FROM osoba, mjesto  
WHERE osoba.pbrStan = mjesto.pbr  
AND mjesto.nazMjesto LIKE 'Z%'
```

Parse tree



# Dekompozicija i validacija

---

- obavlja se zamjena virtualne relacije (*view*) stablom (*parse tree*) koje proizlazi iz definicije virtualne relacije u rječniku podataka (modifikacija upita)
  - materijalizirane virt. relacije se promatraju jednako kao temeljne relacije
- obavlja se transformacija podupita
- validacija (semantička analiza):
  - ispitati korištenje naziva relacija. Npr. svaka "relacija" iz FROM liste mora biti relacija opisana u rječniku podataka
  - verificirati ispravno korištenje naziva atributa
    - razriješiti imena atributa (kojoj relaciji pripada koji atribut, postoje li dvosmislenosti)
    - nalazi li se atribut u dosegu (*scope*) relacije spomenute u pripadnoj FROM listi (→Baze podataka→podupiti)
  - provjeriti tipove podataka i primjenjivost operacija nad objektima (npr. množenje cijelog broja i niza znakova)
- pojednostavljivanje i normalizacija    (*query rewriter*)
  - npr. predikati za selekciju → konjunktivna normalna forma
  - eliminacija tautologija i kontradikcija

# Translacija u inicijalni izraz relacijske algebre

---

- *parse tree* se transformira u inicijalni izraz relacijske algebre koji se može prikazati kao stablo upita (*query tree*)
- pravilo za transformaciju stabla (*parse tree*) u inicijalni izraz relacijske algebre - za jednostavne upite
  1. načiniti Kartezijev produkt svih relacija navedenih u `<fromList>`
  2. nad rezultatom prvog koraka obaviti operaciju selekcije  $\sigma_F$  gdje je  $F$  uvjet naveden u `<condition>`
  3. nad rezultatom drugog koraka obaviti operaciju projekcije  $\pi_L$  gdje je  $L$  lista izraza navedenih u `<selList>`
    - *bag* verziju operacije, ako nije navedeno **DISTINCT**

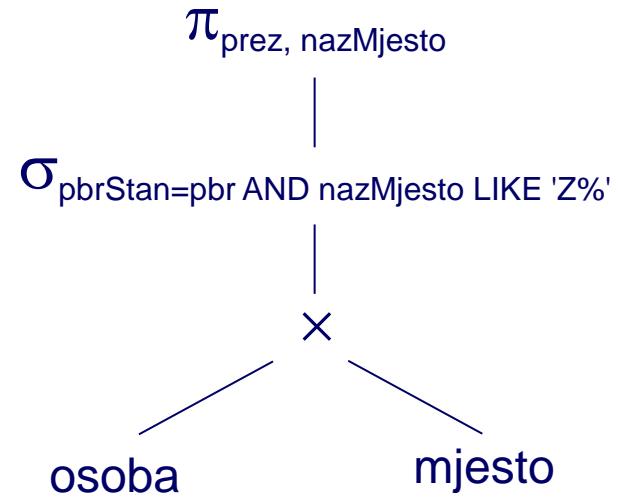
# Translacija u inicijalni izraz relacijske algebre

Primjer:

```
SELECT DISTINCT prez, nazMjesto  
  FROM osoba, mjesto  
 WHERE osoba.pbrStan = mjesto.pbr  
   AND mjesto.nazMjesto LIKE 'Z%'
```

- inicijalni izraz relacijske algebre:

$$\pi_{\text{prez, nazMjesto}}(\sigma_{\text{pbrStan=pbr AND nazMjesto LIKE 'Z%'}}(\text{osoba} \times \text{mjesto}))$$

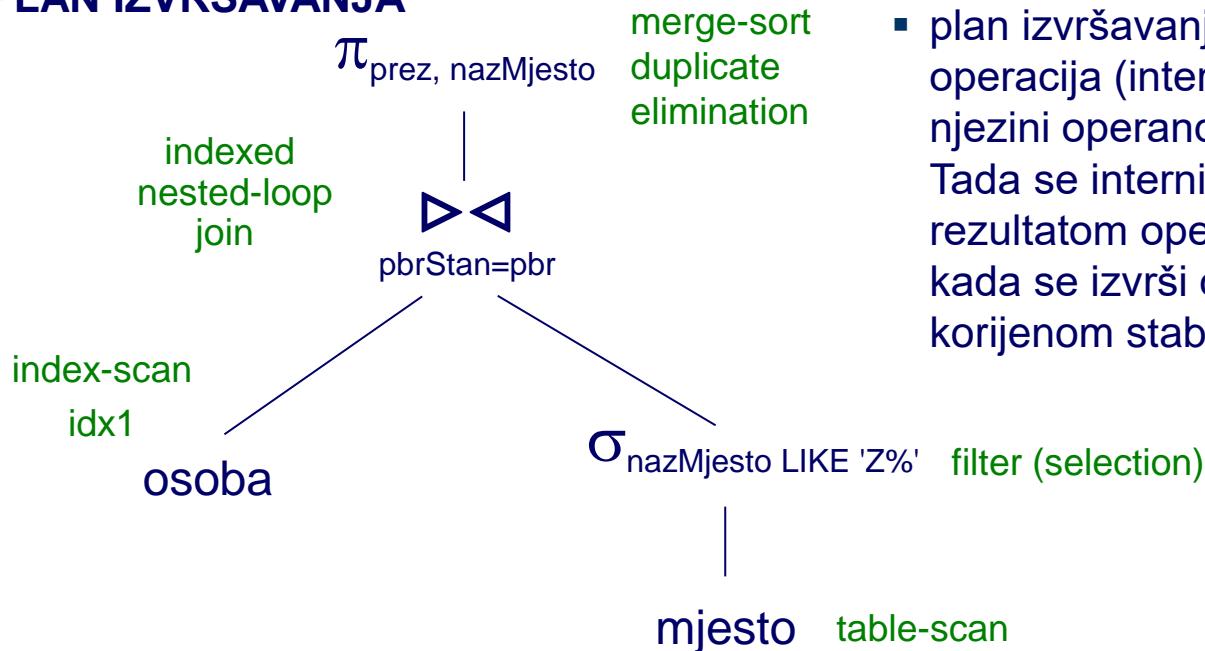


- stablo upita (*query tree*) je način prikaza izraza relacijske algebre
- listovi stabla su relacije, a interni čvorovi su operacije relacijske algebre
- konačni rezultat dobije se obavljanjem operacije u korijenu stabla

# Stablo upita → optimizacija → Plan izvršavanja upita

- plan izvršavanja upita (*execution plan, execution strategy*) obuhvaća
  - stablo upita: operande i operacije, redoslijed njihovog izvršavanja
  - fizičke metode izvršavanja algebarskih operacija, odnosno fizičke operatore (*physical operators*)
    - index-scan, table-scan, nested-loop join, merge join, hash join, ...*
  - način prosljeđivanja međurezultata

## PLAN IZVRŠAVANJA



- plan izvršavanja upita se evaluira tako da se operacija (interni čvor) izvršava onda kada njezini operandi (djeca) postanu raspoloživi. Tada se interni čvor zamjenjuje relacijom - rezultatom operacije. Izvršavanje završava kada se izvrši operacija predstavljena korijenom stabla

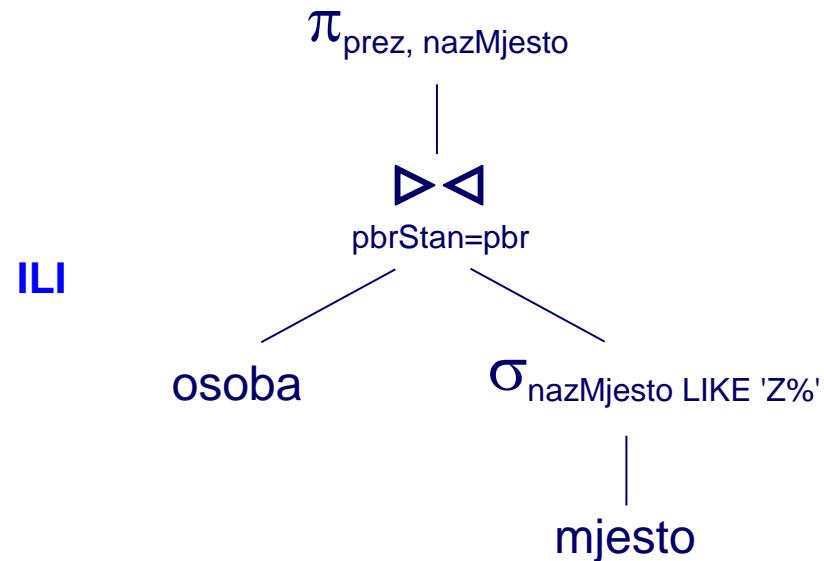
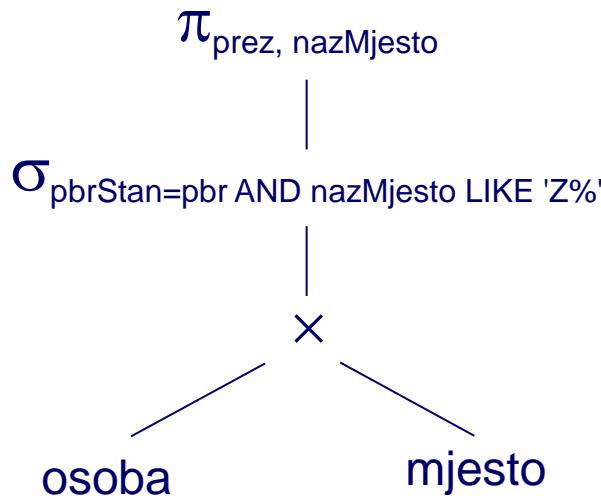
- na slici, zbog preglednosti, nisu opisani načini prosljeđivanja međurezultata

# Plan izvršavanja nije jednoznačno određen upitom

## 1. operandi i operacije relacijske algebre, redoslijed njihovog izvršavanja

- u općem slučaju se izraz relacijske algebre može zamijeniti ekvivalentnim, alternativnim izrazom relacijske algebre (jednim od mnogih)
  - *dva izraza relacijske algebre su ekvivalentni ako primijenjeni nad svakom instancom baze podataka daju međusobno jednak rezultat*
- alternativni izrazi (stabla upita) se određuju na temelju pravila za transformaciju izraza relacijske algebre

Primjer:

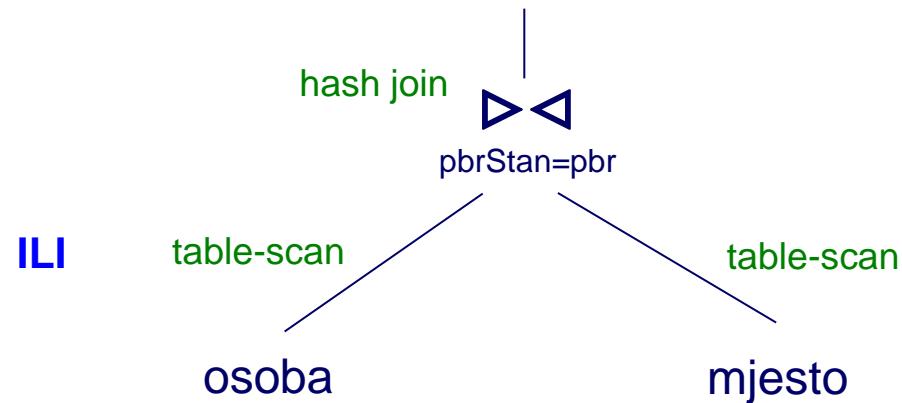
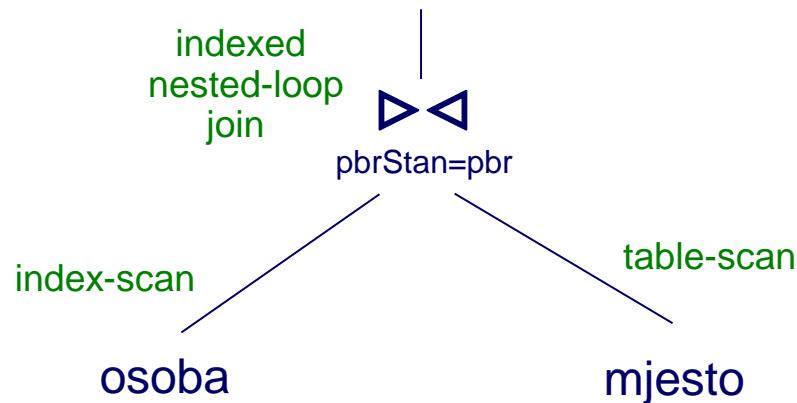


# Plan izvršavanja nije jednoznačno određen upitom

## 2. fizičke metode izvršavanja algebarskih operacija

- SUBP implementira različite metode izvršavanja karakterističnih algebarskih operacija (selekcija, spajanje, projekcija, ...)
- mogućnost primjene pojedine metode ovisi o konkretno primjenjenoj fizičkoj organizaciji (npr. postoji li odgovarajući indeks u konkretnoj relaciji)
- razmatraju se alternativne metode izvršavanja algebarskih operacija, njihova primjenjivost i efikasnost

Primjer:



## 3. način proslijedivanja međurezultata (kasnije)

# Optimizacija

---

⇒ za jedan upit postoji više (potencijalno mnogo) alternativnih planova izvršavanja

- izvršavanjem "lošeg" plana (umjesto "dobrog" plana), utrošak resursa se može povećati za nekoliko redova veličine
- procjenjuju se troškovi za
  - UI operacije (značajno za velike baze podataka)
  - CPU (značajno za male i *in-memory* baze podataka)
  - komunikaciju (značajno za distribuirane SUBP)
- proces odabira *najprikladnijeg* plana izvršavanja naziva se optimizacija upita (*query optimization*)
  - donekle pogrešni termin: odabrani plan nije uvijek optimalan
  - međutim, određivanje optimalnog plana moglo bi biti vremenski prezahtjevno ili neizvedivo zbog nedostatka potrebnih informacija
  - cilj "optimizacije" upita jest odabrati *razumno efikasan* plan izvršavanja

# Dinamička i statička optimizacija upita

---

- *dynamic query optimization*
  - optimizacija se provodi svaki put kada se pokrene izvršavanje upita
  - dobro: informacije na temelju kojih se provodi optimizacija su ažurne
  - loše: veći utrošak resursa za postupke optimizacije
- *static query optimization*
  - upit se optimizira jednom, generirani plan izvršavanja ("izvršni kôd") se pohranjuje, te se koristi pri pokretanju svakog "sličnog" upita
    - neovisno o SQL sjednici
  - dobro: optimizacija se obavlja relativno rijetko pa se može provesti uz razmatranje većeg broja alternativnih planova
  - loše: podaci na temelju kojih je provedena optimizacija (statistički podaci) se mijenjaju tijekom vremena. U jednom trenutku dobar plan može u promijenjenim okolnostima postati loš
- *hybrid approach*
  - koristi se statička optimizacija, ali sustav ponavlja postupak optimizacije upita kada se statistički podaci značajnije promijene

## Fizički operatori

- implementacija i procjena troškova

# Implementacija operacija relacijske algebre

---

- fizički operator je konkretna implementacija (u SUBP-u) algoritama za obavljanje operacija relacijske algebre:
  - selekcija, spajanje, Kartezijev produkt, projekcija
  - grupiranje i agregatne funkcije
  - unija, presjek, razlika
- za obavljanje dodatnih operacija iz SQL jezika:
  - eliminacija duplikata
  - sortiranje
- ostalih fizičkih operacija, npr:
  - čitanje n-torki slijednim čitanjem blokova podataka (*table-scan*)
  - čitanje n-torki uz korištenje indeksa (*index-scan*)
  - čitanje samo vrijednosti ključeva iz indeksa (*key-only index-scan*)

- specifičnost SQL-a u odnosu na relacijski model podataka
  - tablica (*table*) u SQL-u nije nužno relacija (skup n-torki), nego multiskup (*multiset, bag*) n-torki. Stoga se i skup operacija mora proširiti operacijama koje kao operande koriste multiskupove

## "bag" verzije operacija unije, presjeka i razlike

- neka su  $r$  i  $s$  "relacije" u kojima je dopuštena pojava više istih n-torki
- ako se n-torka  $t$  u "relaciji"  $r$  nalazi  $m$  puta, u "relaciji"  $s$  se nalazi  $n$  puta, tada
  - u rezultatu  $r \cup^B s$  n-torka  $t$  se nalazi  $m+n$  puta **UNION ALL**
  - u rezultatu  $r \cap^B s$  n-torka  $t$  se nalazi  $\min(m, n)$  puta **INTERSECT ALL**
  - u rezultatu  $r \setminus^B s$  n-torka  $t$  se nalazi  $\max(0, m-n)$  puta **EXCEPT ALL**

## "bag" verzija operacija projekcije

- $\pi_L^B(r)$  - izdvajanje vertikalnog podskupa relacije prema listi atributa  $L$  (ali bez eliminacije duplikata)

# Bag (multiset, multiskup) verzije operacija

Podsjetnik  
(uzeti u obzir)

Primjer:

| r | E | F |
|---|---|---|
|   | 1 | A |
|   | 1 | A |
|   | 2 | B |
|   | 2 | B |
|   | 2 | B |

| s | E | F |
|---|---|---|
|   | 1 | A |
|   | 1 | A |
|   | 2 | B |

| t | F | G |
|---|---|---|
|   | A | 5 |
|   | A | 6 |
|   | C | 7 |

$r \cup^B s$

|  | E | F |
|--|---|---|
|  | 1 | A |
|  | 1 | A |
|  | 1 | A |
|  | 1 | A |
|  | 2 | B |
|  | 2 | B |
|  | 2 | B |
|  | 2 | B |

$r \cap^B s$

|  | E | F |
|--|---|---|
|  | 1 | A |
|  | 1 | A |
|  | 2 | B |

$r \setminus^B s$

|  | E | F |
|--|---|---|
|  | 2 | B |
|  | 2 | B |

slično i za ostale operacije:

$\sigma_{E=1}^B(s)$

|  | E | F |
|--|---|---|
|  | 1 | A |
|  | 1 | A |

$s \triangleright \triangleleft t$

|  | E | F | G |
|--|---|---|---|
|  | 1 | A | 5 |
|  | 1 | A | 5 |
|  | 1 | A | 6 |
|  | 1 | A | 6 |

$\pi_E^B(s)$

|  | E |
|--|---|
|  | 1 |
|  | 1 |
|  | 2 |

# Selekcija

---

- način obavljanja operacije ovisi o uvjetu selekcije (*filter*) i raspoloživim metodama pristupa (*access methods*)
  - *filter* - u ovom području često korišten alternativni pojam za uvjet selekcije
- **jednostavna selekcija**  $\sigma_{A > 5} (r)$ 
  - jednostavna selekcija linearom pretragom
  - jednostavna selekcija uz pomoć indeksa
- **složena selekcija**
  - konjunktivna forma  $\sigma_{A > 5 \wedge B = 100} (r)$
  - disjunktivna forma  $\sigma_{A > 5 \vee B = 100} (r)$

# Jednostavna selekcija

$\sigma_F(r)$  gdje je  $F$  jednostavna formula oblika  $a \theta c$

- $a$  atribut
- $\theta$  je operacija iz skupa  $\{ <, \leq, >, \geq, = \}$
- $c$  je konstanta

## Mogući načini obavljanja

1. linearna pretraga (*table-scan, sequential scan, linear search, brute force*) za čitanje svih n-torki uz izdvajanje onih koje zadovoljavaju formulu  $F$
2. indeks za dohvat skupa n-torki prema formuli  $a \theta c$  ako za atribut  $a$  postoji indeks

Indeks se također može koristiti za formule oblika *range query*:  $a \text{ BETWEEN } c1 \text{ AND } c2$

Ako postoji odgovarajući indeks, ne znači da će se nužno i koristiti.

U nekim uvjetima optimizator će izbjegći korištenje indeksa i radije koristiti linearnu pretragu!

# Jednostavna selekcija pomoću indeksa

---

## a) čitanje n-torki pomoću indeksa (B-stabla)

- *index-scan*
  - indeks se koristi za pronalaženje pokazivača (RID, row-id) na n-torke koje zadovoljavaju uvjet selekcije, n-torke (tj. dijelovi n-torki čiji se atributi ne nalaze u ključu B-stabla) dohvaćaju se iz blokova s podacima
- u sustavu SQL Server ta vrsta operacije naziva se *index-seek*, a u planu izvršavanja prikazuje se kao *index-seek+RID lookup*
  - indeks se koristi za pronalaženje pokazivača (RID) na razini lista (*index-seek*), zatim se iz blokova s podacima dohvaćaju n-torke (*RID lookup*)
- ova vrsta operacije neće se koristiti ako se iz relacije dohvaća relativno veliki broj n-torki, naročito ako je stupanj grupiranja loš. U takvom slučaju linearna pretraga je prikladnija.
  - stupanj grupiranja: što su n-torke bolje poredane prema vrijednosti ključa indeksa, stupanj grupiranja je veći (ili bolji)

# Jednostavna selekcija pomoću indeksa

---

b) čitanje samo vrijednosti ključeva iz listova indeksa (B-stabla)

- *key-only index-scan*
  - vrijednosti atributa koje treba pročitati odnose se samo na atribute koji se nalaze u listovima B-stabla. To znači da SUBP neće trebati dohvaćati n-torce ili dijelove n-torki iz blokova s podacima
  - nije važno koliko je velik skup n-torki koje se čitaju. Čak i u slučaju čitanja vrijednosti ključeva iz listova za sve n-torce iz relacije (kada se indeks očito ne koristi za svoju primarnu funkciju, a to je filtriranje n-torki prema uvjetu selekcije), operacija je ipak prikladnija od linearног pretraživanja jer više se isplati čitati sve blokove listova nego sve blokove s podacima (zašto? koliki je broj listova u odnosu na broj blokova s podacima?)
- pojmovi u sustavu SQL Server
  - *index-scan*: ako se čitaju vrijednosti iz listova za sve n-torce iz relacije (ne dohvaćaju se blokovi)
  - *index-seek (bez spominjanja + RID lookup)*: ako se indeks koristi za filtriranje prema uvjetu selekcije (dakle, dohvaćaju se vrijednosti iz samo nekih listova, ne dohvaćaju se blokovi)

# Jednostavna selekcija pomoću indeksa

Primjeri za a) i b) - *index-scan* i *key-only index-scan*

```
SELECT ime FROM osoba WHERE sifra < 10;
```

*index-scan*

SQL Server: *index-seek + "RID lookup"*

```
SELECT ime FROM osoba WHERE sifra > 10;
```

*table-scan*

*zašto ne index-scan?*

```
SELECT sifra FROM osoba WHERE sifra < 10;
```

*key-only index-scan*

SQL Server: *index-seek (bez "+RID lookup")*

```
SELECT sifra FROM osoba WHERE sifra > 10;
```

*key-only index-scan*

SQL Server: *index-seek (bez "+RID lookup")*

```
SELECT sifra FROM osoba;
```

*key-only index-scan*

SQL Server: *index-scan*

```
CREATE INDEX idxOso  
ON osoba(sifra);
```

osoba

| sifra | ime   |
|-------|-------|
| 1     | Pero  |
| 2     | Ana   |
| 3     | Ivo   |
| ...   | ...   |
| 10000 | Marko |

10 000

# Jednostavna selekcija pomoću indeksa

c) (SQL Server) čitanje n-torki iz listova *clustered* indeksa

- *index-seek (clustered)*
- *index-scan (clustered)*

Primjeri:

```
CREATE CLUSTERED INDEX idxOso  
ON osoba(sifra);
```

```
SELECT ime FROM osoba WHERE sifra < 10;
```

SQL Server: *clustered index-seek*

```
SELECT ime FROM osoba WHERE sifra > 10;
```

SQL Server: *clustered index-seek*

```
SELECT ime FROM osoba;
```

SQL Server: *clustered index-scan*

| osoba |       |
|-------|-------|
| sifra | ime   |
| 1     | Pero  |
| 2     | Ana   |
| 3     | Ivo   |
| ...   | ...   |
| 10000 | Marko |

zašto se ovdje indeks koristi čak i onda kada se dohvaća veliki broj n-torki?

# Jednostavna selekcija pomoću indeksa

d) *index self-join (IBM Informix)*

- složeni indeks (A, B, C) se (ipak) može koristiti i za uvjete selekcije oblika  $B=x$ 
  - ali samo ako su vrijednosti atributa A slabo raspršene

Primjer:

```
CREATE INDEX idxOso ON osoba (spol, sifra);
SELECT * FROM osoba
    WHERE sifra BETWEEN 20 AND 30;
```

- SUBP će u prvom čitanju indeksa dohvatiti različite vrijednosti  $v$  atributa **spol**, a zatim po jednom, za svaku dohvaćenu vrijednost  $v \in \{ 'M' , 'Ž' \}$  obaviti:

```
SELECT * FROM osoba
    WHERE spol = v AND sifra BETWEEN 20 AND 30;
```

- ako vrijednosti atributa A nisu slabo raspršene, SUBP neće koristiti index self-join!

| osoba |       |
|-------|-------|
| spol  | sifra |
| Ž     | 1     |
| Ž     | 14    |
| Ž     | 15    |
| Ž     | 19    |
| Ž     | 21    |
| Ž     | 24    |
| Ž     | 37    |
| Ž     | 59    |
| M     | 12    |
| M     | 23    |
| M     | 64    |
| M     | 72    |
| M     | 83    |
| M     | 91    |
| M     | 94    |

# Složena selekcija

---

## Konjunktivna forma

- Uvjet selekcije je oblika  $F_1 \wedge F_2 \wedge F_3 \dots$ , gdje su  $F_i$  jednostavne formule
- 1. konjunktivna selekcija korištenjem jednog indeksa
  - ako se za neki  $F_i$  može (i isplati) primijeniti indeks, tada se pomoću indeksa dohvate n-torce koji zadovoljavaju  $F_i$ , a nad svakom tako dohvaćenom n-torkom primjenjuje se *filter* sastavljen od preostalih izraza  $F_1 \wedge F_2 \dots$
  - ako su na raspolaganju indeksi koji se mogu koristiti za više od jedne formule  $F_i$ , odabire se onaj  $F_i$  (i koristi pripadajući indeks) kojim će se dohvatiti najmanji broj n-torki (ili možda onaj s boljim stupnjem grupiranja - razmislite zašto)
- 2. konjunktivna selekcija korištenjem složenog (kompozitnog) indeksa
  - ako se u konjunktivnom obliku formule s relacijskim operatorima usporedbe ( $<$ ,  $\leq$ ,  $>$ ,  $\geq$ ,  $=$ ) koriste atributi za koje postoji složeni indeks, npr.
  - ako postoji index `idx(a, b)`, može se iskoristiti za predikat:
    - `a = 7 AND b > 'P'`
- 3. linearna pretraga

# Složena selekcija

## Konjunktivna forma

Primjer:

```
SELECT * FROM ispit
  WHERE ocjena = 3
    AND sifPred = 201
    AND datIspit = TODAY
```

- broj n-torki u relaciji 100000
- broj različitih predmeta 100
- broj različitih ocjena 5

Ako bi postojao, bilo bi moguće

- koristiti složeni indeks (**sifPred, ocjena**) (+ filter za ostale uvjete)
- koristiti indeks za **sifPred** (+ filter za ostale uvjete)
- koristiti (ali pitanje je isplati li se) indeks za **ocjena** (+ filter za ostale uvjete)
- koristiti kompozitni indeks za (**sifpred, datIspit**) (+ filter za ostale uvjete)
- koristiti kompozitni indeks za (**datIspit, ocjena, sifPred**)
- međutim, ako postoji indeks za **sifpred** i indeks za **ocjena**, koristi se samo indeks za **sifPred** (+ filter za ostale uvjete)
  - zašto?

# Složena selekcija

## Disjunktivna forma (disjunktivna selekcija)

- Uvjet selekcije je oblika  $F_1 \vee F_2 \vee F_3 \dots$ , gdje su  $F_i$  jednostavne formule
  - samo u slučaju ako postoji (isplativi) indeks za svaki  $F_i$ , zasebno se dohvaća po jedan skup identifikatora n-torki (RID, row-id) za svaki uvjet  $F_i$  uz korištenje pripadnog indeksa. Rezultat se dobije unijom dobivenih skupova
  - inače, linearna pretraga
- ovaj "trik" (dohvat skupova row-identifikatora n-torki na temelju više indeksa za svaki  $F_i$ ) u rijetkim slučajevima može se koristiti i za konjunktivni oblik selekcije
  - ako primjena svakog pojedinačnog  $F_i$  kao rezultat daje relativno veliki skup n-torki, a u presjeku tih skupova se očekuje mali broj n-torki, moguće je da će se isplatiti iz listova svih indeksa dohvatiti skupove row-identifikatora koji zadovoljavaju pojedinačne  $F_i$ , pa u primarnoj memoriji napraviti presjek tih skupova. Na taj način izbjegao bi se dohvat velikog broja n-torki (koje zadovoljavaju samo jedan od odabranih  $F_i$ ) iz blokova s podacima

# Složena selekcija

## Disjunktivna forma (disjunktivna selekcija)

Primjer:

```
SELECT * FROM ispit
  WHERE datIspit = TODAY
    OR sifPred = 201
```

- ako postoji samo kompozitni indeks (**sifPred** , **datIspit**) , koristi se linearna pretraga
- ako postoji samo indeks za **sifPred** ili samo indeks za **datIspit**, linearna pretraga!
- ako postoji indeks za **sifpred** i indeks za **datIspit**, dohvaća se skup identifikatora n-torki za koje je **datIspit=TODAY**, skup identifikatora n-torki za koje je **sifPred=201**, nakon čega se obavlja unija tih skupova
  - u ovakvim slučajevima, ako su skupovi veliki, optimizator može ipak odabrat linearnu pretragu

# Statistički podaci iz rječnika podataka

---

- $B(r)$  - broj blokova relacije  $r$
- $N(r)$  - broj n-torki relacije  $r$
- $d(idx)$  - dubina B-stabla za indeks  $idx$
- $V(A, r)$  - broj različitih vrijednosti atributa  $A$  u relaciji  $r$ 
  - $V(A, r) = G_{COUNT(*)}(\pi_A(r))$
  - $A$  može biti skup atributa
- ostali statistički podaci (na sljedećem predavanju)
  
- SUBP ne ažurira statističke podatke pri svakoj promjeni podataka, već
  - u unaprijed propisanim vremenima (npr. u vremenu smanjenog opterećenja sustava), reakcijom na neki događaj u sustavu (unos veće količine podataka), itd.
  - odlukom administratora sustava, npr. naredbom oblika

**UPDATE STATISTICS FOR TABLE  $r$**

# Selekcija - procjena troškova

- procjena troška: procjenjuje se broj U/I operacija koje će biti potrebno obaviti u najlošijem slučaju
- bez indeksa, linearna pretraga:  $B(r)$
- pomoću indeksa idx (B-stabla)
  - jedna n-torka (preko ključa relacije):  $d(idx) + 1$
  - više n-torki
    - *clustered index*:  $d(idx) + \text{broj blokova u kojima se nalaze tražene n-torke}$
    - *non-clustered index*: vrijednosti u listovima indeksa jesu sortirane, ali tražene n-torke mogu biti više ili manje raspršene među blokovima s podacima. Ekstremni slučaj: za dohvat svake n-torke potrebna je jedna U/I operacija. Ako je broj n-torki koje se dohvaćaju velik, trošak može biti i veći nego trošak linearne pretrage pa se tada indeks ne koristi

Za svaki indeks SUBP u rječniku podataka može voditi podatak o "stupnju grupiranja" (*degree of clustering*) koji govori u kojoj mjeri fizički poredak n-torki u blokovima s podacima odgovara poretku vrijednosti ključeva u dotičnom indeksu.

Što se može reći o stupnju grupiranja *clustered* indeksa u sustavu SQL Server?

# Primjeri za *clustered* indeks (IBM Informix)

```
CREATE INDEX idxOso ON osoba (sifra);
```

Prema slici se može zaključiti da je stupanj grupiranja vrlo nizak:

```
SELECT ime FROM osoba WHERE sifra < 500;
```

table-scan

osoba

| sifra | ime    |
|-------|--------|
| 354   | Jura   |
| 2     | Tomo   |
| 8753  | Marija |
| ...   | ...    |
| 14    | Jana   |

```
DROP INDEX idxOso;  
CREATE CLUSTER INDEX idxOso ON osoba (sifra);
```

```
SELECT ime FROM osoba WHERE sifra < 500;
```

index-scan

# Sortiranje

- sortiranje je jedna od temeljnih fizičkih operacija
  - zbog ORDER BY u SQL naredbi
  - zbog fizičkih operatora koji zahtijevaju da argumenti budu sortirani (vidjeti kasnije npr. *spajanje uparivanjem sortiranih relacija*)
- relacija  $r(R)$  i lista atributa  $L \subseteq R$  prema kojima treba obaviti sortiranje
  - ako postoji odgovarajući indeks (koji sadrži  $L$  u *odgovarajućem* poretku)
    - *index-seek (clustered)* i *index-scan (clustered)* (SQL Server) prikladni su neovisno o broju n-torki
    - *key-only index-scan* prikladan je ako ključ indeksa sadrži sve attribute navedene u *select-listi*, neovisno o broju n-torki
    - *index-scan* je prikladan samo ako je stupanj grupiranja relativno visok i broj n-torki je relativno malen
  - ako ne postoji odgovarajući indeks ili ga se ne isplati primijeniti
    - *table-scan* + sortiranje u glavnoj memoriji ili, ako n-torce koje treba sortirati ne stanu u primarnu memoriju koja je na raspolaganju dotičnoj korisničkoj sjednici, **vanjsko sortiranje**

```
SELECT * FROM osoba  
ORDER BY prezime
```

# Vanjsko sortiranje

---

- koristi se za sortiranje relacija koje se ne mogu odjednom smjestiti u segment glavne memorije kojeg korisnička sjednica ima na raspolaganju
- najpoznatiji i najčešći oblik vanjskog sortiranja je vanjsko sortiranje s uparivanjem: *external sort-merge*
- obavlja se u dvije faze
  - faza sortiranja
  - faza uparivanja

# External sort-merge - 1. faza - sorting phase

---

- $B(r)$  - broj blokova relacije  $r$
- $M$  - raspoloživi broj blokova glavne memorije
- relacija  $r$  se dijeli na  $N$  segmenata veličine  $M$ ,  $N = \lceil B(r) / M \rceil$ 
  - svaki segment (*run file*) se sortira zasebno i zapisuje u postojanu memoriju

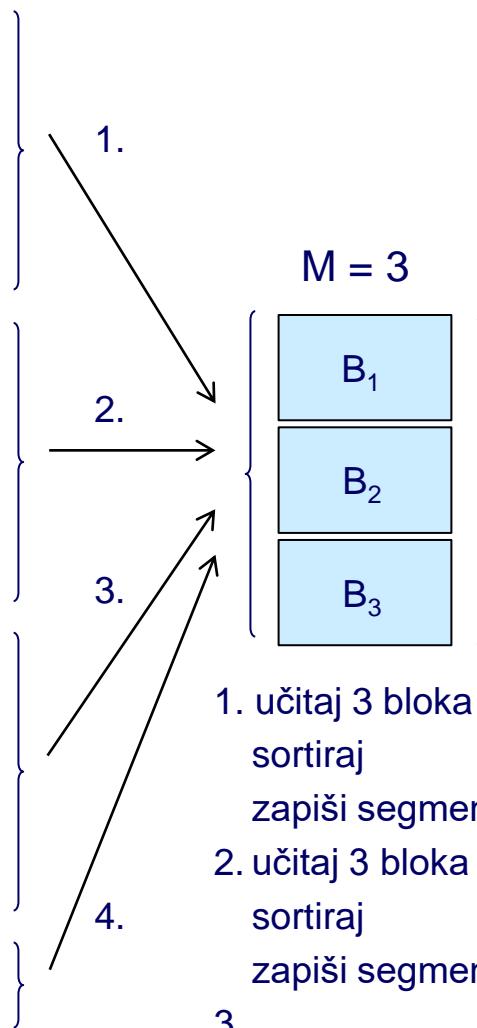
```
i = 0
repeat
    read M blocks of r or the rest of the r
    sort the in-memory part
    write the sorted data to run file Ri
    i = i + 1
until the end of the r
```

# External sort-merge - 1. faza - sorting phase

Primjer:  
zapisa u bloku = 2  
 $B(r) = 10$   
 $M = 3$   
sortiranje prema  
prvom atributu

početna relacija r

|   |    |
|---|----|
| j | 1  |
| o | 2  |
| d | 3  |
| b | 4  |
| t | 5  |
| i | 6  |
| e | 7  |
| v | 8  |
| m | 9  |
| f | 10 |
| a | 11 |
| p | 12 |
| k | 13 |
| s | 14 |
| n | 15 |
| g | 16 |
| c | 17 |
| h | 18 |
| r | 19 |
| u | 20 |



|   |   |
|---|---|
| b | 4 |
| d | 3 |
| i | 6 |
| j | 1 |
| o | 2 |
| t | 5 |

sortirani  
segmenti  
(run files)

$R_1$

|   |    |
|---|----|
| a | 11 |
| e | 7  |
| f | 10 |
| m | 9  |
| p | 12 |
| v | 8  |

$R_2$

|   |    |
|---|----|
| c | 17 |
| g | 16 |
| h | 18 |
| k | 13 |
| n | 15 |
| s | 14 |

$R_3$

|   |    |
|---|----|
| r | 19 |
| u | 20 |

$R_4$

## *External sort-merge - 2. faza - merging phase*

---

- uparivanje sortiranih segmenata
- odjednom se uparuju zapisi iz  $d_m$  segmenata ( $d_m$  je *degree of merging*)
- $d_m = \min(N, M-1)$
- ako je  $N < M$ , tada je  $d_m = N$ , te je za uparivanje dovoljan jedan korak
  - u stvarnosti, najčešće je dovoljan taj jedan korak
- ako je  $N \geq M$ , uparivanje je potrebno provesti u više koraka
  - uparuje se po  $M-1$  segmenata. Time se broj segmenata u svakom koraku reducira za faktor  $M-1$
  - postupak se ponavlja dok broj segmenata ne bude manji od  $M$
  - u stvarnosti: najviše ukupno dva koraka

## *External sort-merge - 2. faza - merging phase*

---

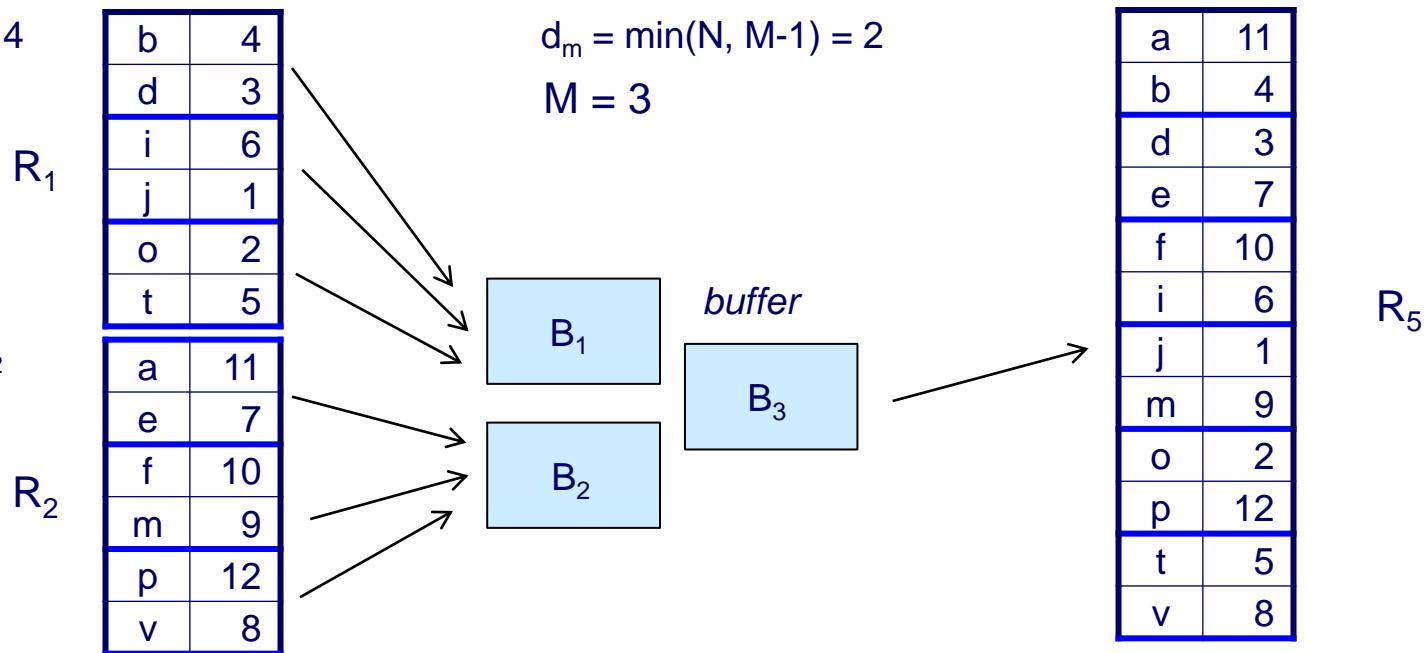
- u glavnoj memoriji alocirati po jedan ulazni blok za svaki od segmenata + jedan izlazni blok za (*buffered!*) zapisivanje rezultata, ukupno ne više od M blokova

```
read first block of each run  $R_i$  into main memory block  $B_i$ 
repeat
    choose the first tuple (in sort order) among all  $B_i$ 
    write the tuple to the output* and delete it from  $B_i$ 
    if  $B_i$  is empty and not end-of-file( $R_i$ )
        read the next block of  $R_i$  into  $B_i$ 
until all blocks  $B_i$  are empty
```

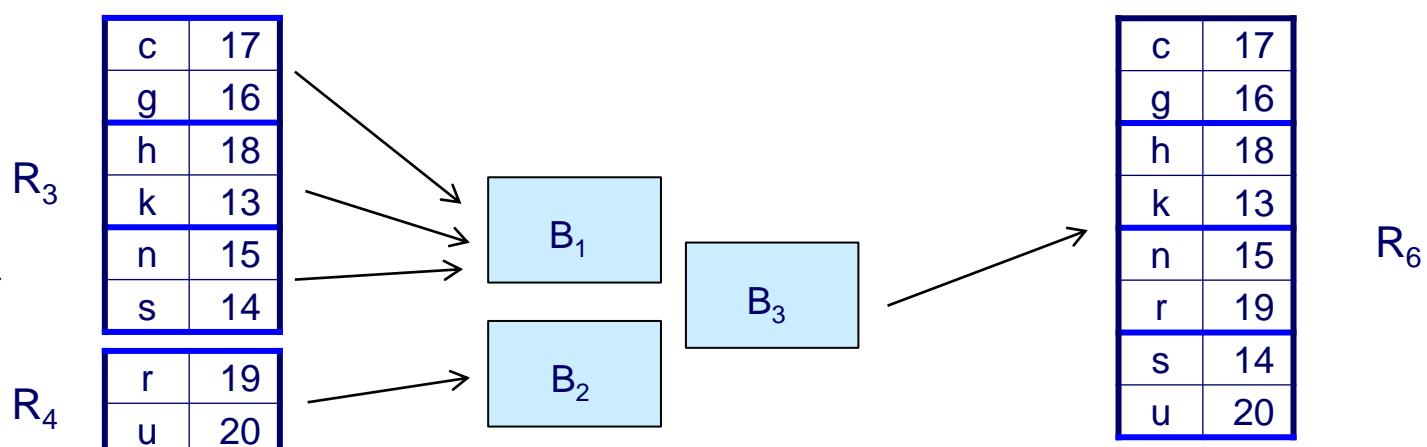
\* *write the tuple to the output*: zapisivanje u postojanu memoriju (uz korištenje međuspremnika) ili proslijedivanje na ulaz u sljedeću operaciju (*pipelining*)

# External sort-merge - 2. faza - merging phase

1. korak  $N = 4$

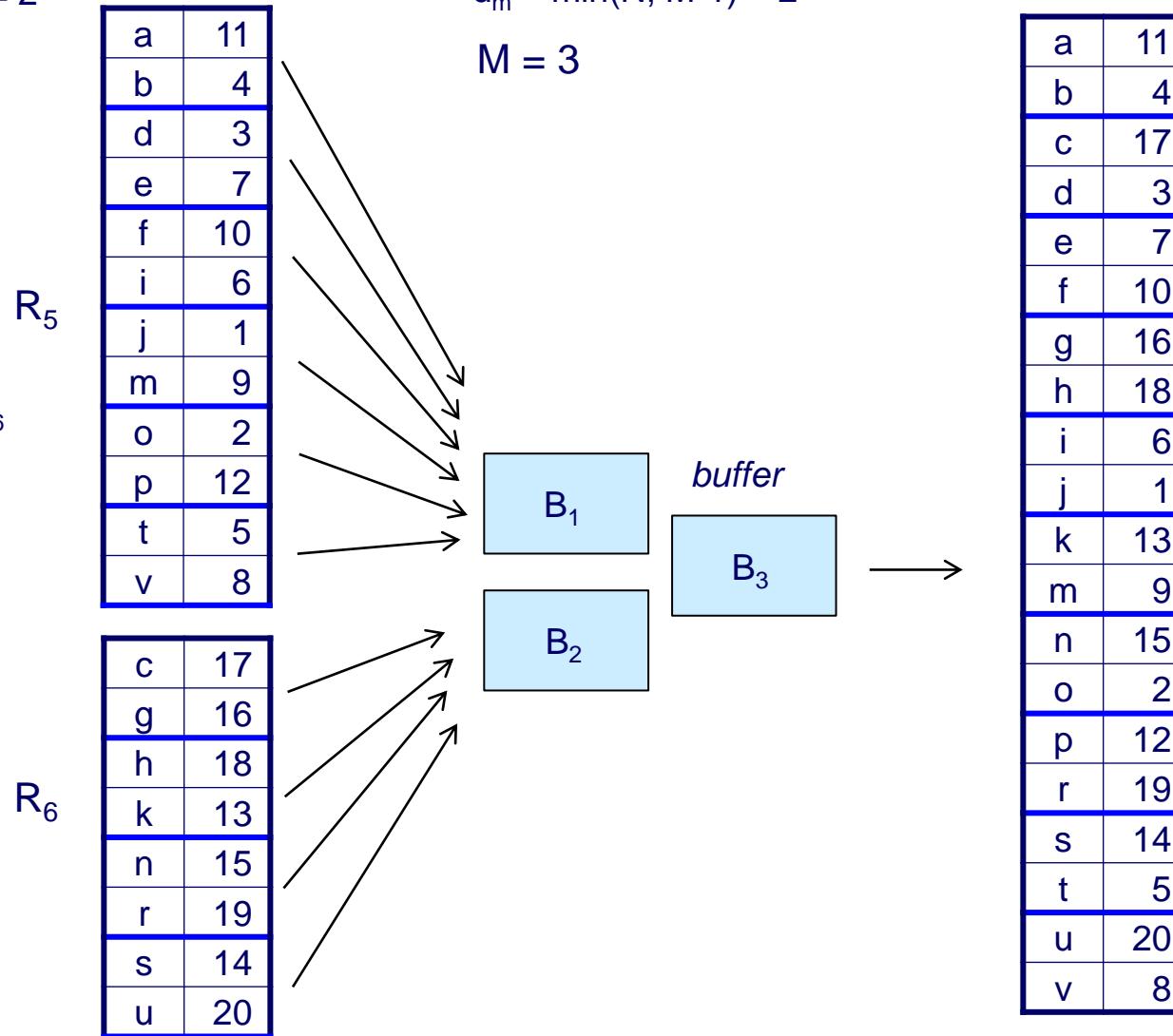


uparivanje  $R_1$  i  $R_2$



# External sort-merge - 2. faza - merging phase

2. korak  $N = 2$



# *External sort-merge - procjena troškova*

---

- Zbog primjera s ekstremno malom raspoloživom primarnom memorijom, čini se da je ova metoda sortiranja vrlo skupa, međutim u realnim situacijama je izuzetno efikasna.
- Primjer
- Sortiranje relacije od 1 000 000 blokova (npr. blok od 2K, ukupno 2GB), pri čemu je na raspolaganju samo 1000+1 blok primarne memorije (približno 2MB).
- U postupku će nastati 1 000 sortiranih segmenata, za koje će biti potreban samo jedan korak uparivanja
  - to znači da se sa samo 2MB primarne memorije, 2GB podataka može sortirati uz utrošak od samo 4 000 000 U/I operacija
  - (to je onoliko U/I operacija koliko je potrebno da bi se datoteka dva puta kopirala iz jednog mesta na drugo mjesto u sekundarnoj memoriji)

# External sort-merge - procjena troškova

---

- broj U/I operacija za 1. fazu (sortiranje):  $2 B(r)$ 
  - $B(r)$  *input* +  $B(r)$  *output* operacija
- svaki korak druge faze (uparivanje) zahtijeva  $2 B(r)$  U/I operacija
- u prvom koraku druge faze uparuje se  $\lceil B(r) / M \rceil$  segmenata
- u drugom koraku uparuje se  $\lceil B(r) / M \rceil / (M-1)$  segmenata
- u  $i$ -tom koraku uparuje se  $\lceil B(r) / M \rceil / (M-1)^{i-1}$  segmenata
- u svakom koraku sigurno se uparuje barem  $M-1$  segmenata
  - $\lceil B(r) / M \rceil / (M-1)^{i-1} \geq M-1$
  - $i \leq \log_{M-1} \lceil B(r) / M \rceil$
  - $\Rightarrow$  potreban broj koraka u drugoj fazi:  $\lceil \log_{M-1} \lceil B(r) / M \rceil \rceil$
- ukupan broj U/I operacija za 2. fazu:  $2 B(r) \lceil \log_{M-1} \lceil B(r) / M \rceil \rceil$
- ukupan broj potrebnih U/I operacija:  $2 B(r) + 2 B(r) \lceil \log_{M-1} \lceil B(r) / M \rceil \rceil$ 
  - uz uključen trošak zapisivanja konačnog rezultata, inače za  $B(r)$  manje
- u primjeru za  $M=3$ : broj potrebnih U/I operacija = 60 (uz uračunati trošak zapisivanja rezultata)

# Spajanje

---

1. spajanje s ugniježđenim petljama: *nested-loop join*
2. blokovsko spajanje s ugniježđenim petljama: *block nested-loop join*
3. indeksirano spajanje s ugniježđenim petljama: *indexed nested-loop join*
4. spajanje uparivanjem sortiranih relacija: *sort-merge join*
5. spajanje raspršenim adresiranjem: *hash join*

# Nested-loop join

- par ugniježdenih petlji s unaprijed poznatim brojem ponavljanja

$r \triangleright\triangleleft S$   
F

```
for each  $t_r$  in r
    for each  $t_s$  in s
        if F( $t_r, t_s$ )
            add  $t_r \cdot t_s$  to the result
```

vanjska relacija (*outer relation*)  
unutarnja relacija (*inner relation*)

- veliki trošak (uspoređuje se  $N(r) \cdot N(s)$  n-torki)
- najlošiji slučaj: samo po jedan blok svake relacije stanu u glavnu memoriju
  - procjena troška:  $N(r) \cdot B(s) + B(r)$  (nisu uključeni troškovi zapisivanja rezultata)
- najbolji slučaj: obje relacije ili samo manja relacija (manja relacija se tada koristi kao unutarnja) stanu u glavnu memoriju
  - procjena troška:  $B(r) + B(s)$  (nisu uključeni troškovi zapisivanja rezultata)

$t_r \cdot t_s$  : n-torka dobivena ulančavanjem n-torki  $t_r$  i  $t_s$

$N(r), N(s)$  : broj n-torki u relacijama r i s

$B(r), B(s)$  : broj blokova u relacijama r i s

# Block nested-loop join

---

```
for each block  $B_r$  in r
    for each block  $B_s$  in s
        for each  $t_r$  in  $B_r$ 
            for each  $t_s$  in  $B_s$ 
                if  $F(t_r, t_s)$ 
                    add  $t_r \cdot t_s$  to the result
```

- najlošiji slučaj: samo po jedan blok svake relacije stanu u glavnu memoriju
  - procjena troška:  $B(r) \cdot B(s) + B(r)$  (nisu uključeni troškovi zapisivanja rezultata)
- najbolji slučaj: (jednako kao u *nested-loop join*)
  - procjena troška:  $B(r) + B(s)$  (nisu uključeni troškovi zapisivanja rezultata)

Ako je za operaciju spajanja na raspolaganju glavna memorija veličine M blokova

- 1 blok za čitanje unutarnje relacije, 1 blok za izlazni međuspremnik
- vanjska petlja: umjesto po jedan blok,  $\lceil B(r) / (M - 2) \rceil$  puta učitava po M-2 blokova
- procjena troška:  $\lceil B(r) / (M - 2) \rceil \cdot B(s) + B(r)$   
(nisu uključeni troškovi zapisivanja rezultata)

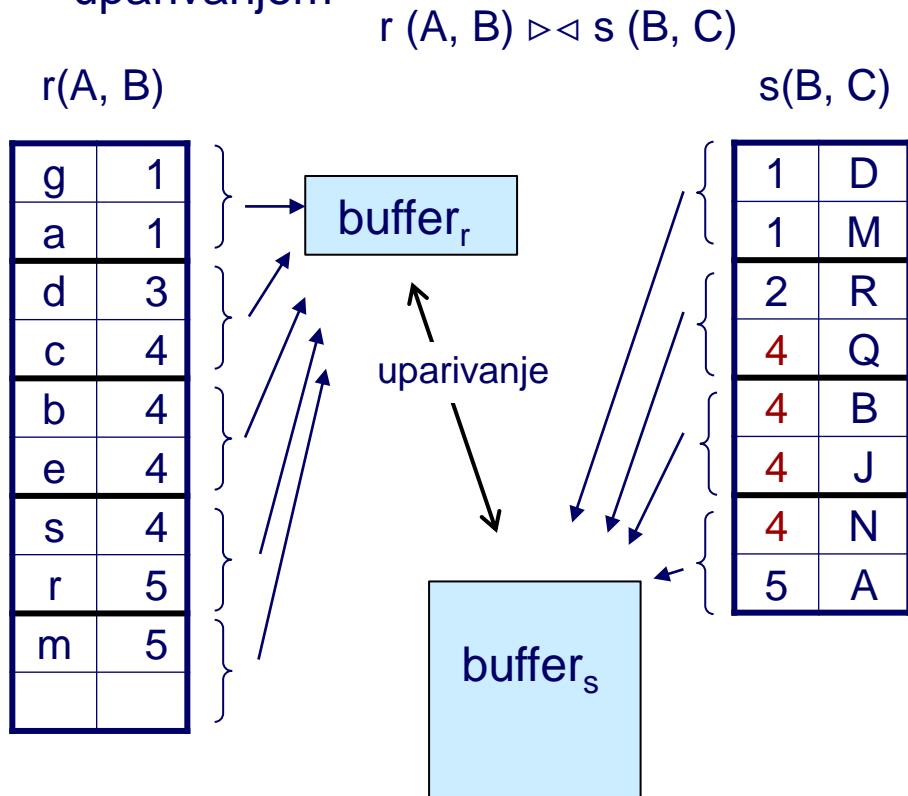
## *Indexed nested-loop join*

---

- preduvjet: na raspolaganju je indeks nad unutarnjom relacijom za atribute prema kojima se obavlja spajanje
- za svaku  $t_r$  iz  $r$ , pomoću indeksa se pronalaze odgovarajuće n-torce iz  $s$
- procjena troška:  $N(r) \cdot c + B(r)$  (nisu uključeni troškovi zapisivanja rezultata)  
 $c$  : broj U/I operacija za dohvrat n-torce iz  $s$  pomoću indeksa (~ dubina stabla)
- što ako obje relacije imaju indeksirane atribute prema kojima se obavlja spajanje?
  - u općem slučaju bolje je za vanjsku relaciju odabrati onu s manje n-torki

# Sort-merge join

- može se koristiti za prirodno spajanje i spajanje uz uvjet s izjednačavanjem (*equi-join*)
- r i s su sortirane prema atributima prema kojima se obavlja spajanje
- postupak spajanja je sličan drugoj fazi u algoritmu za vanjsko sortiranje s uparivanjem



- pretpostavka: u međuspremnik relacije  $s$  moguće je odjednom pohraniti sve n-torke s jednakim vrijednostima atributa prema kojima se obavlja spajanje. U primjeru na slici: drugi, treći i četvrti blok zbog vrijednosti **4**
- međuspremnik relacije  $r$  može biti veličine jednog ili više blokova
- moguće korisna nuspojava:
  - rezultat je sortiran prema atributima prema kojima se obavlja spajanje

# Sort-merge join

---

- procjena troška:
  - $B(r) + B(s)$  (nisu uključeni troškovi zapisivanja rezultata)
  - ako relacije nisu unaprijed sortirane, dodati i broj U/I operacija potrebnih za sortiranje relacija r i s, koji ovisi o veličini relacija u odnosu na raspoloživu primarnu memoriju (vidjeti npr. trošak za *external sort-merge*)
- u rijetkim slučajevima u kojima u međuspremnik relacije **s** neće biti moguće odjednom pohraniti sve n-torke s jednakim vrijednostima atributa prema kojima se obavlja spajanje, trošak će se povećati jer će se spajanje takvih grupa n-torki morati obaviti nepovoljnim oblikom metode *block nested-loop join*

# Hash join

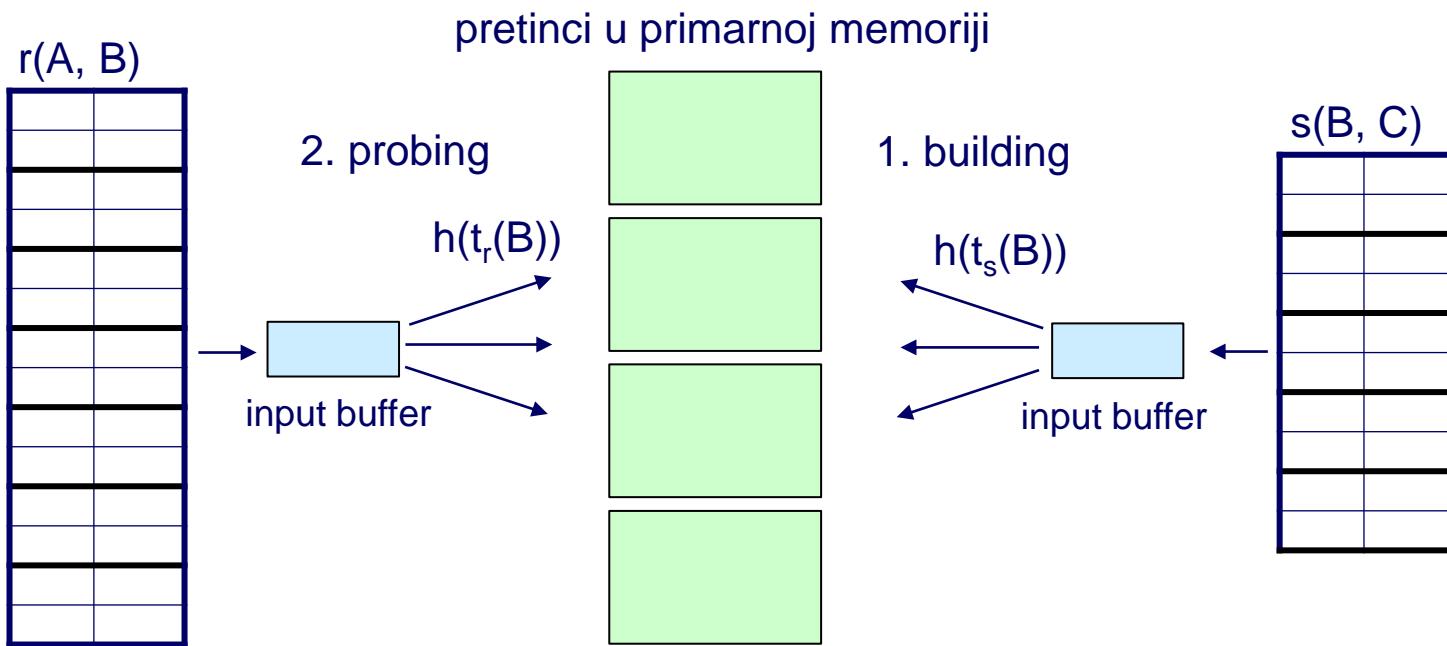
---

- može se koristiti za prirodno spajanje i spajanje uz uvjet s izjednačavanjem (equi-join)  $r(A, B) \bowtie s(B, C)$
- funkcija raspršenja h preslikava B-vrijednost n-torke u adresu pretinca (*bucket*)
  - adresa pretinca =  $h(t(B))$
  - B može biti jedan atribut ili skup atributa

# Hash join

- ako se relacija **s** (manja u paru) može odjednom pohraniti u primarnoj memoriji

- *building phase*
  - n-torce  $t_s$  relacije s raspršuju se u pretince u primarnoj memoriji, adresa =  $h(t_s(B))$
- *probing phase*
  - za svaku n-torku  $t_r$  relacije r izračunava se adresa pretinca  $h(t_r(B))$
  - za svaku  $t_s$  iz tog pretinca, ako je  $t_s(B) = t_r(B)$ , dodaj  $t_r \cdot t_s$  u rezultat. Provjera je nužna jer funkcija raspršenja za različite  $t(B)$  može izračunati istu adresu pretinca



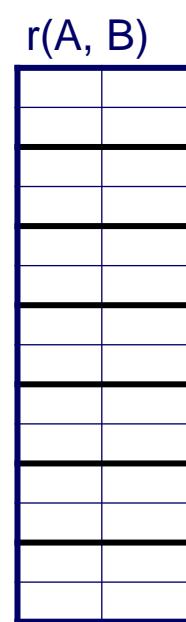
- procjena troška:  $B(r) + B(s)$  (nisu uključeni troškovi zapisivanja rezultata)

# Hash join

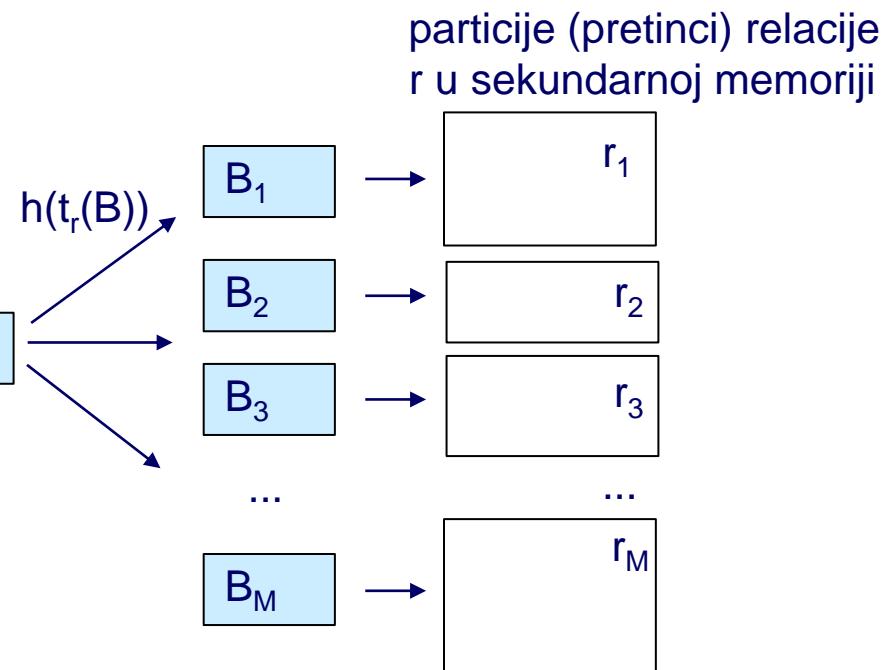
- ako se relacija **s** (manja u paru) ne može odjednom pohraniti u primarnoj memoriji

- ako se relacija **s** ne može smjestiti u glavnu memoriju  $\Rightarrow$  *partitioned hash join*
- partitioning phase:** primjenom funkcije raspršenja  $h(t_r(B))$ , zasebno se particioniraju relacije **r** i **s**. U sekundarnoj memoriji nastaju particije (pretinci):
  - $r_1, r_2, \dots, r_M$  i  $s_1, s_2, \dots, s_M$

1. partitioning



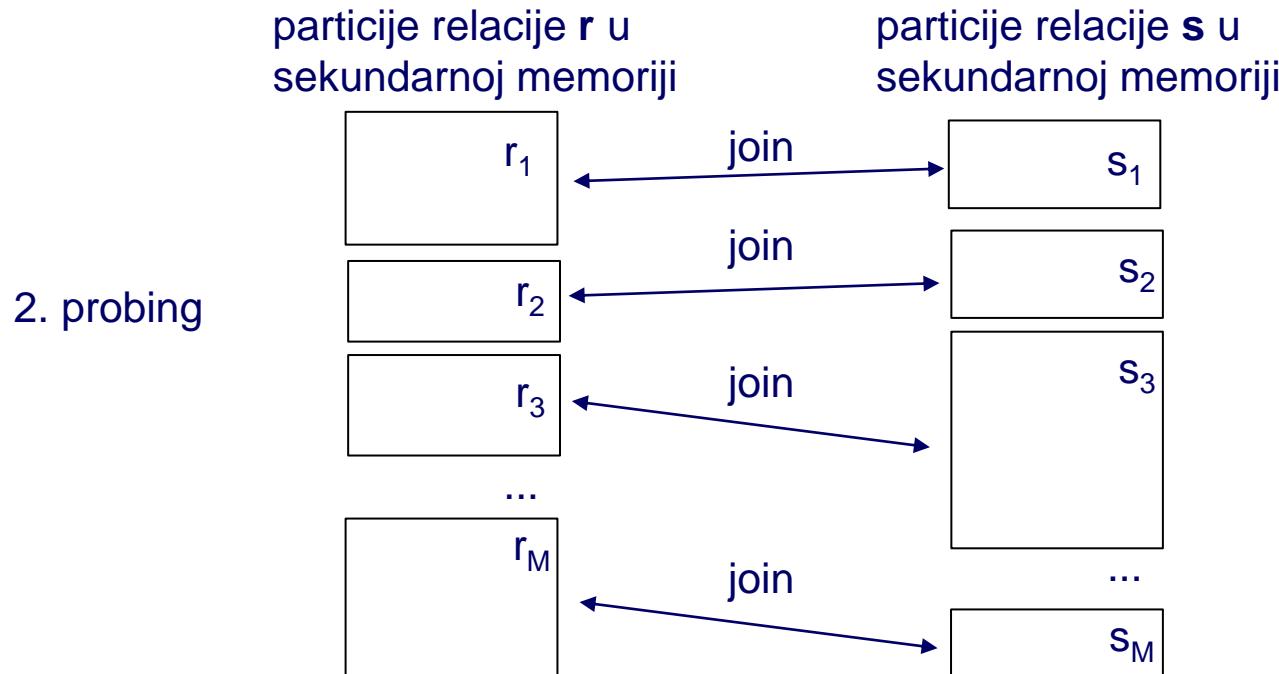
Slika prikazuje  
particioniranje relacije **r**.  
Isti postupak se mora  
provesti i za relaciju **s**.



# Hash join

- ako se relacija **s** (manja u paru) ne može odjednom pohraniti u primarnoj memoriji

- **probing phase:** za svaki  $i = 1..M$ , obavlja se spajanje n-torki iz particija  $r_i, s_i$ 
  - npr. primjenom *nested-loop join* ili *hash-join* (ako barem manja od dviju particija sada stane u primarnu memoriju) ili *block nested-loop join*



- procjena ukupnog troška:  $3 * (B(r) + B(s))$  (nisu uključeni troškovi zapisivanja rezultata)
  - za vježbu objasniti zašto. Također, uočiti da će trošak biti veći od navedenog u slučaju kada se za spajanje nekog para particija koristi *block nested-loop join*)

# Usporedba: *Hash join* i (*auto-index*) *nested-loop join*

- *hash join* u općem slučaju brže evaluira ukupan rezultat
- *nested-loop* brže evaluira "prve n-torke"
  - koristi se ako je važno da korisnik čim prije dobije barem dio rezultata
- *hash join* je efikasniji, ali se može koristiti samo za "*equi-join*" uvjete spajanja

Primjer:

| linija |            |
|--------|------------|
| let    | udaljenost |
| CA-825 | 700        |
| LH-412 | 4800       |
| BA-722 | 15000      |
| CA-311 | 13000      |

| zrakoplov |       |
|-----------|-------|
| tip       | dolet |
| B747      | 13000 |
| A320      | 5400  |
| DC-9      | 3100  |

- sljedeća operacija spajanja se ne može obaviti pomoću raspršenog adresiranja

```
SELECT *
  FROM linija JOIN zrakoplov
    ON dolet >= udaljenost;
```

# Vanjsko spajanje

---

$$r(A, B) \underset{F}{\text{*}\triangleright\triangleleft} s(B, C)$$

- koristi se modificirani oblik jednog od prethodno opisanih algoritama
- npr. modificira se *nested-loop join*
  - u slučaju lijevog vanjskog spajanja, lijeva relacija mora se koristiti kao vanjska relacija
  - ako se za  $t_r$  iz vanjske relacije  $r$  ne pronađe odgovarajuća  $t_s$  iz unutarnje relacije  $s$ , u rezultat se dodaje  $t_r \cdot t_{NULL}$  (gdje je  $t_{NULL}$  n-torka čije su vrijednosti atributa NULL)
- prirodno vanjsko spajanje (*natural outer join*) i vanjsko spajanje s izjednačavanjem (*outer join with equi-join condition*) se (osim pomoću *nested-loop*) također mogu realizirati modificiranim oblicima *sort-merge join* i *hash-join* algoritama

# Eliminacija duplikata

$\delta(r)$

(ako se relacija  $r$  ne može smjestiti u primarnu memoriju)

- pomoću vanjskog sortiranja s uparivanjem (*external sort-merge*)
  - već za vrijeme faze sortiranja, eliminiraju se duplikati pronađeni unutar istog segmenta (smanjuje se inicijalna veličina segmenata)
  - preostali duplikati se jednostavno eliminiraju tijekom faze uparivanja
  - procjena troška: jednak kao kod sortiranja
- pomoću raspršenog adresiranja
  - primjenom funkcije raspršenja  $h(t)$  nastanu particije  $r_1, r_2, \dots, r_M$  (u sek. mem.)
  - za svaku particiju  $r_i$  provodi se postupak:
    - formira se nova *in-memory hash table*, pri čemu funkcija raspršenja mora biti drugačija od  $h$  (jasno je zašto)
    - za svaku  $n$ -torku  $t_r$  iz  $r_i$ 
      - $t_r$  se dodaje u *hash table* samo ako se već ne nalazi u njoj
    - $n$ -torke iz *hash table* se dodaju u rezultat
  - procjena troška:  $3 * B(r)$  (nisu uključeni troškovi zapisivanja rezultata)

# Projekcija

---

$$\pi_L(r) = \delta(\pi_L^B(r)) \quad (\text{ako se relacija } r \text{ ne može smjestiti u primarnu memoriju})$$

- *bag* verzija projekcije (izdvajanje atributa) je jednostavna
- za eliminaciju duplikata se koristi jedan od opisanih postupaka (*external sort-merge, hash*)
- procjena troška: jednaka kao kod operacije eliminacije duplikata

# Agregacija i grupiranje

---

$A_1, A_2, \dots, A_m G_{\mathcal{AF}_1(B_1), \mathcal{AF}_2(B_2), \dots, \mathcal{AF}_n(B_n)}(r)$

## Ako je rezultat prevelik za raspoloživi prostor главне memorije

- postupak kao kod eliminacije duplikata raspršenim adresiranjem
  - umjesto eliminacije duplikata, za svaku grupu izračunavaju se agregatne funkcije
  - procjena troška:  $3 * B(r)$  (nisu uključeni troškovi zapisivanja rezultata)

## Ako se rezultat može pohraniti u raspoloživi prostor главне memorije

- rezultat se pohranjuje u *in-memory hash table*
- za svaku n-torku s ključem  $A_1, A_2, \dots, A_m$ 
  - ako se još ne nalazi u *in-memory hash table*
    - dodati n-torku  $A_1, A_2, \dots, A_m, \mathcal{AF}_1, \mathcal{AF}_2, \dots, \mathcal{AF}_n$
    - izračunati nove vrijednosti  $\mathcal{AF}_1, \mathcal{AF}_2, \dots, \mathcal{AF}_n$
  - procjena troška:  $B(r)$  (nisu uključeni troškovi zapisivanja rezultata)

# Unija, presjek, razlika

(ako se manja relacija ne može smjestiti u primarnu memoriju)

- **pomoću vanjskog sortiranja s uparivanjem**
  - obje relacije sortirati po svim atributima
  - tijekom uparivanja
    - ▷ n-torku iz **r** ili **s** dodati u rezultat, preskočiti sve jednake n-torke u obje relacije
    - ▷ n-torku iz **r** koja se nalazi i u **s** dodati u rezultat, preskočiti sve jednake n-torke u obje relacije
    - \\ n-torku koja se nalazi u **r**, a ne nalazi se u **s**, dodati u rezultat, preskočiti sve jednake n-torke u obje relacije
- procjena troška:
  - ako su **r** i **s** već prije sortirane
    - $B(r) + B(s)$
  - ako **r** i **s** nisu sortirane
    - trošak sortiranja (**r**) + trošak sortiranja (**s**) +  $B(r) + B(s)$

# Unija, presjek, razlika

(ako se manja relacija ne može smjestiti u primarnu memoriju)

- **pomoću raspršenog adresiranja**

- primjenom funkcije raspršenja  $h(t)$  nastanu particije
    - $r_1, r_2, \dots, r_M, S_1, S_2, \dots, S_M$
  - ponavljati za svaki  $r_i$ 
    - $r_i \rightarrow \text{in-memory hash table}$
- ▷ ▪ za svaku n-torku  $t_s$  iz  $S_i$  koja se već ne nalazi u *in-memory hash table*
  - dodati  $t_s$  u *in-memory hash table*
- sadržaj *in-memory hash table* dodati u konačni rezultat
- ∩ ▪ za svaku n-torku  $t_s$  iz  $S_i$  koja se nalazi u *in-memory hash table*
  - dodati  $t_s$  u rezultat
- \ ▪ za svaku n-torku  $t_s$  iz  $S_i$  koja se nalazi u *in-memory hash table*
  - izbaciti  $t_s$  iz *in-memory hash table*
- sadržaj *in-memory hash table* dodati u konačni rezultat

# Literatura:

---

- H. Garcia-Molina, J. Ullman, J. Widom. Database system implementation. Prentice-Hall International, Inc. Upper Saddle River, New Jersey. 2000.
- A. Silberschatz, H. Korth, S. Sudarshan. Database system concepts, 6th ed. McGraw-Hill. 2010.
- R. Elmasri, S. Navathe. Fundamentals of database systems, 3rd ed. Addison-Wesley Publishing Company. Reading, Massachusetts. 2000.
- IBM Informix Dynamic Server Performance Guide, Version 11.50, IBM, 2008.

# **Sustavi baza podataka**

Predavanja

**4. Obavljanje upita**

**(2. dio)**

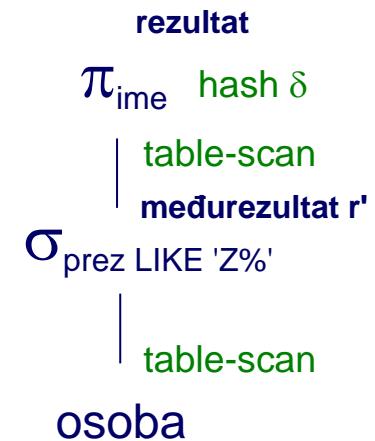
**ožujak 2023.**

## **Složeni izrazi relacijske algebre**

**- način prosljeđivanja međurezultata -**

# Način prosljeđivanja međurezultata

- rezultat složenog izraza relacijske algebre evaluira se kao niz pojedinačnih operacija relacijske algebre
- osim redoslijeda obavljanja operacija i odabira fizičkih operatora, planom se treba definirati i način postupanja s rezultatima pojedinačnih operatora (međurezultatima) jer rezultat jednog operatora predstavlja ulazni argument sljedećeg operatora



## A) Materijalizacija (*materialization*)

- međurezultat se kao cjelina pohranjuje
  - ili u međuspremniku
  - ili u sekundarnu memoriju (ako je međurezultat prevelik za raspoložive međuspremniku). Problemi: trošak U/I operacija radi zapisivanja (a kasnije opet čitanja) međurezultata; nije moguća rana produkcija "prvih rezultata"

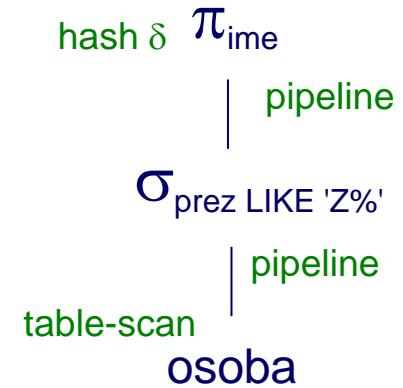
### Primjer (sa slikom):

- obavi  $\sigma_{prez \text{ LIKE } 'Z\%'"}$  i pohrani međurezultat  $r'$   
*(materijalizirati ga u sekundarnoj memoriji)*
- obavi  $\pi_{ime} ( r' )$

# Način prosljeđivanja međurezultata

## B) Cjevovodi (*pipelining, stream-based processing*)

- fizički operator s niže razine (proizvođač, *producer*) producira pojedinačne n-torce koje koristi operator na višoj razini (potrošač, *consumer*)
  - *demand-driven (pulling)* - češća implementacija
  - *producer-driven (pushing)* - redovi, sinkronizacija



### Primjer (*pulling*):

- operator  $\pi$  zahtijeva od operatora  $\sigma$  sljedeću n-torku, operator  $\sigma$  zahtijeva od operatora *table-scan* sljedeću n-torku, operator *table-scan* čita n-torku i predaje u  $\sigma$ ,  $\sigma$  obrađuje n-torku i predaje u  $\pi$ ,  $\pi$  obrađuje n-torku i predaje kao rezultat
- koji će se model (materijalizacija ili cjevovodi) koristiti u konkretnom slučaju najviše ovisi o obliku fizičkog operatora, npr.
  - *external sort-merge* nije naročito pogodan za primjenu u modelu cjevovoda: ne može producirati n-torce dok ne prihvati cijeli međurezultat s prethodne razine i započne s posljednjom fazom uparivanja
  - operator selekcije, spajanje s ugniježđenim petljama, *bag* verzija unije, primjeri su operatora pogodnih za primjenu u modelu cjevovoda

# Prosljeđivanje rezultata i iteratori

---

- kod implementacije *pulling* modela, SUBP koristi poseban oblik sučelja: *iterator*
- korištenjem uvijek istog sučelja, zakrivaju se interni implementacijski detalji za fizički operator. Time se omogućuje jednostavna zamjena operadora koji obavljaju istu operaciju, a implementirani su na različiti način (optimalno za svaki specifični slučaj)
- Za svaki operator implementiraju se tri metode:
  - `open()`: metoda koja inicijalizira stanje iteratora, alocira potrebnu memoriju (npr. ulazne i izlazne redove)
  - `getNext()`: metoda pozivatelju vraća "sljedeću n-torku"
  - `close()`: oslobađa zauzete resurse
- iteratori se mogu izvršavati svaki unutar svojeg vlastitog procesa ili dretve
  - omogućava se paralelno izvršavanje upita

# Iteratori - *table scan* iterator

```

TableScan {
    open(r) {
        bR := the first block of r;
        tR := the first tuple of bR;
        FOUND := true; /* global variable */
    }
    getNext() {
        if tR is past last tuple on block bR
            increment bR to the next block;
        if there is no next block in r
            FOUND := false;
            return;
        else
            tR := the first tuple of bR;
            return_tR := tR;
            increment tR to the next tuple of bR;
            return return_tR;
    }
    close() { ... }
}

```

- pseudokod je znatno pojednostavljen - nedostaju oznake programskih blokova {}, definicije varijabli, rukovanje praznim relacijama, alokacija memorije u open() i oslobađanje memorije u close(), ...
- vidi se da je i implementacija također vrlo pojednostavljenja. Ovdje se npr. pretpostavilo da u zadanoj relaciji sigurno postoji barem jedna n-torka

# Iteratori - *union (bag) iterator*

```
BagUnion {  
    open(r, s) {  
        tscanR := new TableScan(); tscanR.open(r);  
        currentRel := r;  
    }  
    getNext() {  
        if currentRel = r  
            t := tscanR.getNext();  
            if FOUND  
                return t;  
            else  
                tscanR.close();  
                currentRel := s;  
                tscanS := new TableScan(); tscanS.open(s);  
                return tscanS.getNext();  
    }  
    close() {  
        tscanS.close(); ...  
    }  
}
```

$$r \cup^B s$$

## **Složeni izrazi relacijske algebre**

**- procjena troškova -**

# Složeni izrazi relacijske algebre - procjena troškova

Primjer:

```
SELECT *
  FROM stud AS r, ispit AS s
 WHERE mbr = mbrStud
   AND datIspit = '17.6.2008';
```

$$\sigma_{\text{datIsp}='17.6.2008' \wedge \text{mbr}=\text{mbrStud}}(\text{stud} \times \text{ispit})$$
$$\sigma_{\text{datIsp}='17.6.2008'} \underset{\text{mbr}=\text{mbrStud}}{(\text{stud} \bowtie \text{ispit})}$$
$$\text{stud} \bowtie (\sigma_{\text{datIsp}='17.6.2008'} \underset{\text{mbr}=\text{mbrStud}}{(\text{ispit})})$$

ekvivalentni izrazi  
relacijske algebre

- procijeniti broj U/I operacija koje će se obaviti za svaki od navedenih ekvivalentnih izraza relacijske algebre

# Složeni izrazi relacijske algebre - procjena troškova

## Pretpostavke:

| stud                                    | ispit                                   |
|-----------------------------------------|-----------------------------------------|
| ▪ $N(r) = 5\ 000$                       | ▪ $N(s) = 100\ 000$                     |
| ▪ $B(r) = 500$                          | ▪ $B(s) = 3000$                         |
| ▪ $V(mbr, r) = 5000$                    | ▪ $V(datlsp, s) = 200$                  |
| ▪ veličina n-torke $t_r = 0.05$ blokova | ▪ veličina n-torke $t_s = 0.02$ blokova |

**Uočiti:**  $B(r)$  može biti veći od  $N(r) \cdot$  veličina n-torke  $t_r$  jer blokovi ne moraju biti u cijelosti popunjeni n-torkama

- svi međurezultati se zapisuju u sekundarnu memoriju
- za spajanje (i Kartezijev produkt) koristiti *block nested-loop join*. Prepostaviti najlošiji slučaj: na raspolaganju je samo  $M=3$  (broj blokova glavne memorije).
- nad relacijama nema kreiranih indeksa
- primarni ključ relacije  $r$  je  $mbr$

# Složeni izrazi relacijske algebre - procjena troškova

---

$\sigma_{\text{datisp}='17.6.2008' \wedge \text{mbr}=\text{mbrStud}}(\text{stud} \times \text{ispit})$

- trošak za  $r_1 = \text{stud} \times \text{ispit}$  - *block nested-loop*
  - čitanje:  $B(r) + B(r) \cdot B(s) = 500 + 1.5 \cdot 10^6 \approx 1.5 \cdot 10^6$  U/I operacija
  - pisanje međurezultata ( $r_1$ ):
    - $N(r_1) = N(r) \cdot N(s) = 500 \cdot 10^6$
    - veličina n-torke rezultata: 0.07 blokova (zašto?)
    - $B(r_1) = 35 \cdot 10^6$  blokova  $\Rightarrow 35 \cdot 10^6$  U/I operacija
- trošak za  $r_2 = \sigma_{\text{datisp}='17.6.2008' \wedge \text{mbr}=\text{mbrStud}}(r_1)$  - *table-scan*
  - $35 \cdot 10^6$  U/I operacija
- ukupni trošak (bez troškova zapisivanja rezultata):  $\approx 71.5 \cdot 10^6$  U/I operacija

# Složeni izrazi relacijske algebre - procjena troškova

$$\sigma_{\text{datisp}='17.6.2008'} (\text{stud} \bowtie \text{ispit})$$

mbr=mbrStud

- trošak za  $r_1 = \text{stud} \bowtie \text{ispit}$  - *block nested-loop join*
  - čitanje:  $B(r) + B(r) \cdot B(s) = 500 + 1.5 \cdot 10^6 \approx 1.5 \cdot 10^6$  U/I operacija
  - pisanje međurezultata ( $r_1$ ):
    - $N(r_1) = 100\ 000$  - jer se svaka  $t_s$  može spojiti s najviše jednom  $t_r$
    - veličina n-torke rezultata: 0.07 blokova
    - $B(r_1) = 7000$  blokova  $\Rightarrow 7000$  U/I operacija
- trošak za  $r_2 = \sigma_{\text{datisp}='17.6.2008'}(r_1)$  - *table-scan*
  - 7000 U/I operacija
- ukupni trošak (bez troškova zapisivanja rezultata):  $\approx 1.51 \cdot 10^6$  U/I operacija

# Složeni izrazi relacijske algebre - procjena troškova

---

stud  $\bowtie$  ( $\sigma_{\text{datlsp}='17.6.2008'}(\text{ispit})$ )  
mbr=mbrStud

- trošak za  $r_1 = \sigma_{\text{datlsp}='17.6.2008'}(\text{ispit})$  - *table-scan*
  - čitanje:  $B(s) = 3000$  U/I operacija
  - pisanje međurezultata ( $r_1$ ):
    - $N(r_1) = 100\ 000 / V(\text{datlsp}, s)$  - pretp: jednolika distribucija po datlsp
    - veličina n-torke rezultata: 0.02 blokova
    - $B(r_1) = 10$  blokova  $\Rightarrow 10$  U/I operacija
- trošak za  $r_2 = \text{stud} \bowtie r_1$  - *block nested-loop join*
  - $B(r_1) + B(r_1) \cdot B(r) = 5010$  U/I operacija
- ukupni trošak (bez troškova zapisivanja rezultata): 8020 U/I operacija
- za vježbu: procijeniti trošak za svaki od prikazanih primjera ako su na raspolaganju  $M=102$  blokova glavne memorije

# Složeni izrazi relacijske algebre - procjena troškova

---

- trošak izvršavanja operacija relacijske algebre nad temeljnim relacijama i materijaliziranim pogledima ovisi o primjenjenom fizičkom operatoru
- u složenim izrazima relacijske algebre važna je i veličina međurezultata
- međurezultati se (u osnovi) pohranjuju kao neporedane datoteke
  - posljedica: broj U/I operacija koje se obavljaju nad međurezultatima je proporcionalan veličini međurezultata
- prethodni primjer pokazuje kako je za procjenu veličine međurezultata važan i broj n-torki i veličina n-torke
  - veličina n-torke je podatak lako dostupan iz rječnika podataka. Ako se radi o n-torci varijabilne duljine, može se koristiti podatak o prosječnoj veličini n-torke
  - broj n-torki u rezultatu obavljanja neke operacije nije moguće saznati prije nego se operacija obavi
- budući da se ne može unaprijed znati koliko će n-torki imati međurezultat, koriste se pravila za procjenu broja n-torki u rezultatu operacije relacijske algebre
- cilj procjene NIJE predvidjeti točan broj n-torki u međurezultatu

# Procjena broja n-torki u rezultatu operacije rel. algebre

---

- za relaciju  $r(A, B, \dots)$  u rječniku podataka pohranjeni su statistički podaci:
  - $N(r)$  - broj n-torki relacije  $r$
  - $V(A, r)$  - broj različitih vrijednosti atributa  $A$  u relaciji  $r$ 
    - $V(A, r) = G_{\text{COUNT}(\star)}(\pi_A(r))$ 
      - pri tome  $A$  može biti jedan atribut ili skup atributa
  - $\min(A, r), \max(A, r)$  - najmanja i najveća vrijednost atributa  $A$  u relaciji  $r$
  - histogrami

# Jednostavna selekcija

$\sigma_{A=v}(r)$

- ako je  $V(A, r)$  poznat, uz pretpostavljenu jednoliku razdiobu vrijednosti atributa A procijenjeni broj n-torki  $= N(r) / V(A, r)$
- ako  $V(A, r)$  nije poznat:  $= N(r) / 10$
- ako razdioba nije jednolika, mogu pomoći histogrami (ako su na raspolaganju)

## Histogrami:

- za svaki raspon vrijednosti atributa A (granice raspona ovise o primjenjenoj rezoluciji) u rječniku podataka se evidentira broj n-torki, npr.
- ispit(mbrStud, datIspit, sifPred, ocjena)
  - $N(ispit) = 100 000$
  - $V(ocjena, ispit) = 5$

Procjena broja n-torki za  $\sigma_{ocjena=2}(ispit)$

Pravila za procjenu veličine rezultata nisu univerzalna. Ovdje je prikazan podskup pravila koja se koriste u SUBP IBM Informix.



- bez stat. podataka: 10 000
- pomoću  $V(A, r)$ : 20 000
- pomoću histograma: 30 000

# Jednostavna selekcija

---

$$\sigma_{A \leq v}(r)$$

- ako su vrijednosti  $v$ ,  $\min(A, r)$ ,  $\max(A, r)$  poznate u trenutku optimizacije
  - za  $v < \min(A, r)$   $= 0$
  - za  $v \geq \max(A, r)$   $= N(r)$
  - inače  $= N(r) \cdot (v - \min(A, r)) / (\max(A, r) - \min(A, r))$
- ako vrijednost  $v$  ili  $\min(A, r)$ ,  $\max(A, r)$  nisu poznate u trenutku optimizacije
  - $= N(r) / 3$ 
    - u tom slučaju se isti izraz koristi i za nejednakosti oblika  $<$ ,  $\geq$ ,  $>$

$$\sigma_{A \text{ LIKE } \text{izraz}}(r) = N(r) / 5$$

# Selekcija - uvjet selekcije s negacijom

---

$$\sigma_{\neg F}(r)$$

- ako se procijenjeni broj n-torki za operaciju  $\sigma_F(r)$  označi s  $N_F(r)$  tada je procijenjeni broj n-torki u rezultatu operacije  $\sigma_{\neg F}(r)$   
 $= N(r) - N_F(r)$
- prethodni izraz se može iskoristiti za procjenu npr.  
 $\sigma_{A \neq v}(r) = \sigma_{\neg(A=v)}(r) \Rightarrow N(\sigma_{A \neq v}(r)) = N(r) - N(\sigma_{(A=v)}(r))$
- slično također za  
 $\sigma_{A \text{ NOT LIKE } \text{izraz}}(r) = \sigma_{\neg(A \text{ LIKE } \text{izraz})}(r)$
- slično također za  
 $\sigma_{A > v}(r) = \sigma_{\neg(A \leq v)}(r)$ 
  - ali ako  $v$  ili  $\min(A, r)$ ,  $\max(A, r)$  nisu poznati u trenutku optimizacije, tada se opet procjenjuje na  $N(r) / 3$ , a ne  $N(r) - N(r) / 3$

# Selektivnost predikata (*selectivity*)

---

- $f(F, r)$ : broj iz intervala  $[0, 1]$
- što je selektivnost predikata  $F$ ? Omjer broja n-torki iz  $r$  koje zadovoljavaju predikat  $F$  i ukupnog broja n-torki u  $r$
- $f(F, r) = N(\sigma_F(r)) / N(r)$        $\Rightarrow N(\sigma_F(r)) = N(r) \cdot f(F, r)$

npr. za  $\sigma_{A=v}(r)$

- ako je  $V(A, r)$  poznat,  $f(A=v, r) = N(r) / V(A, r) / N(r) = 1 / V(A, r)$
- ako  $V(A, r)$  nije poznat,  $f(A=v, r) = N(r) / 10 / N(r) = 0.1$
- u oba slučaja vrijedi:
  - $N(\sigma_{A=v}(r)) = N(r) \cdot f(A=v, r)$

# Složena selekcija - konjunktivna forma

---

$$\sigma_{F_1 \wedge F_2 \wedge \dots \wedge F_m}(r)$$

- za svaki  $F_i$  se procjenjuje veličina rezultata  $N_{F_i}(r) = N(\sigma_{F_i}(r))$
- $N_{F_i}(r)$  je procijenjeni broj n-torki za operaciju  $\sigma_{F_i}(r)$ 
  - procjena se obavlja prema prethodno opisanim pravilima
- vjerojatnost da n-torka zadovoljava pojedinačni uvjet  $F_i$  je  $N_{F_i}(r) / N(r)$ 
  - odnosno  $f(F_i, r)$  - ovako je lakše napisati istu vrijednost
- ako su  $F_i$  nezavisni, vjerojatnost da n-torka zadovoljava sve  $F_i$  jest
$$(N_{F_1}(r) / N(r)) \cdot (N_{F_2}(r) / N(r)) \cdot \dots \cdot (N_{F_m}(r) / N(r))$$
  - odnosno  $f(F_1, r) \cdot f(F_2, r) \cdot \dots \cdot f(F_m, r)$
- procijenjeni broj n-torki u rezultatu je dakle
$$N(r) \cdot (N_{F_1}(r) \cdot N_{F_2}(r) \cdot \dots \cdot N_{F_m}(r)) / (N(r))^m$$
  - odnosno  $N(r) \cdot f(F_1, r) \cdot f(F_2, r) \cdot \dots \cdot f(F_m, r)$

# Nezavisnost pojedinačnih uvjeta u složenim predikatima

- prethodno navedeni izrazi podrazumijevaju nezavisnost pojedinačnih uvjeta  
osoba

| mbr | pbrRod | pbrStan | spol |
|-----|--------|---------|------|
| 101 | 51000  | 51000   | Ž    |
| 102 | 51000  | 51000   | M    |
| 103 | 51000  | 51000   | Ž    |
| 104 | 51000  | 21000   | M    |
| 105 | 51000  | 51000   | Ž    |
| 106 | 51000  | 51000   | M    |
| 107 | 51000  | 51000   | Ž    |
| 108 | 51000  | 51000   | M    |
| 109 | 21000  | 21000   | Ž    |
| 110 | 21000  | 21000   | M    |
| 111 | 21000  | 21000   | M    |
| 112 | 21000  | 21000   | Ž    |
| 113 | 21000  | 21000   | Ž    |
| 114 | 21000  | 21000   | M    |
| 115 | 21000  | 51000   | Ž    |
| 116 | 21000  | 21000   | M    |

- N(osoba) = 16
- V(pbrRod, osoba) = 2
- V(pbrStan, osoba) = 2
- ocijeniti nezavisnost pojedinačnih uvjeta u složenim predikatima sljedećih algebarskih operacija

$$s = \sigma_{pbrRod=51000 \wedge pbrStan=21000} (\text{osoba})$$

$$N(s) = 16 \cdot 0.5 \cdot 0.5 = 4$$

a koliko je zapravo n-torki u rezultatu ?

$$s = \sigma_{pbrRod=51000 \wedge spol='\checkmark'} (\text{osoba})$$

$$N(s) = 16 \cdot 0.5 \cdot 0.5 = 4$$

a koliko je zapravo n-torki u rezultatu ?

# Složena selekcija - disjunktivna forma

---

$$\sigma_{F_1 \vee F_2 \vee \dots \vee F_m}(r)$$

- vjerojatnost da n-torka ne zadovoljava pojedinačni uvjet  $F_i$  je  $1 - N_{F_i}(r) / N(r)$ 
  - odnosno  $1 - f(F_i, r)$
- ako su  $F_i$  nezavisni, vjerojatnost da n-torka ne zadovoljava niti jedan  $F_i$  jest
$$(1 - N_{F_1}(r) / N(r)) \cdot (1 - N_{F_2}(r) / N(r)) \dots \cdot (1 - N_{F_m}(r) / N(r))$$
  - odnosno  $(1 - f(F_1, r)) \cdot (1 - f(F_2, r)) \cdot \dots \cdot (1 - f(F_m, r))$
- vjerojatnost da zadovoljava barem jedan od  $F_i$ 
$$1 - (1 - N_{F_1}(r) / N(r)) \cdot (1 - N_{F_2}(r) / N(r)) \dots \cdot (1 - N_{F_m}(r) / N(r))$$
  - odnosno  $1 - (1 - f(F_1, r)) \cdot (1 - f(F_2, r)) \cdot \dots \cdot (1 - f(F_m, r))$
- procijenjeni broj n-torki u rezultatu je dakle
$$N(r) \cdot (1 - (1 - N_{F_1}(r) / N(r)) \cdot (1 - N_{F_2}(r) / N(r)) \dots \cdot (1 - N_{F_m}(r) / N(r)))$$
  - odnosno  $N(r) (1 - (1 - f(F_1, r)) \cdot (1 - f(F_2, r)) \cdot \dots \cdot (1 - f(F_m, r)))$

# Spajanje

$r(R) \bowtie s(S)$

- ako je  $R \cap S = \emptyset \Rightarrow N(r) \cdot N(s)$
- ako je  $R \cap S = \text{ključ u } R \Rightarrow N(s)$  (jer se svaka  $t_s$  može spojiti s najviše jednom  $t_r$ )
- ako je  $R \cap S = \text{ključ u } S \Rightarrow N(r)$  (jer se svaka  $t_r$  može spojiti s najviše jednom  $t_s$ )
- ako je  $R \cap S = A$ , A je atribut ili skup atributa koji nije ključ niti za R niti za S
  - procjena za  $r \bowtie s$
  - promatra se jedna n-torka  $t_r$  (ona ima neku konkretnu A-vrijednost)
  - broj n-torki  $t_s$  koje imaju A-vrijednost jednaku A-vrijednosti dotične n-torke  $t_r$   
 $= N(s) / V(A, s)$
  - svaka  $t_r$  će se spojiti sa  $N(s) / V(A, s)$  n-torki  $t_s$ . Procijenjeni broj n-torki  
 $N(r \bowtie s) = N(r) \cdot N(s) / V(A, s)$
- simetrično:  $N(s \bowtie r) = N(s) \cdot N(r) / V(A, r)$
- očito, ako je  $V(A, r) \neq V(A, s)$ , procjene se razlikuju! U takvom slučaju, kao konačna procjena uzima se manja od dviju prikazanih, dakle  
$$N(r \bowtie s) = N(r) \cdot N(s) / \max(V(A, r), V(A, s))$$

# Unija, presjek, razlika, projekcija, agregacija

---

$\sigma_{F_1}(r) \cup \sigma_{F_2}(r)$  unija podskupova iste relacije

- procjenjuje se jednako kao rezultat operacije  $\sigma_{F_1 \vee F_2}(r)$

$\sigma_{F_1}(r) \cap \sigma_{F_2}(r)$  presjek podskupova iste relacije

- procjenjuje se jednako kao rezultat operacije  $\sigma_{F_1 \wedge F_2}(r)$

$r \cup s \Rightarrow N(r) + N(s)$

$r \cap s \Rightarrow \min(N(r), N(s))$

$r \setminus s \Rightarrow N(r)$

$\pi_L(r) \Rightarrow V(L, r)$

$\llcorner G_{\mathcal{AF}_1(B_1), \mathcal{AF}_2(B_2), \dots, \mathcal{AF}_n(B_n)}(r) \Rightarrow V(L, r)$

# Primjer

```
CREATE TABLE ispit (
    mbrStud
, sifPred
, sifNast
, datIspit
, ocjena);
```

- procijeniti broj n-torki u rezultatu operacije:

$$\sigma_{\text{ocjena}=2 \wedge \text{datIspit}='16.4.2008'}(\text{ispit})$$

$$N(\text{ispit}) = 200\ 000$$

$$V(\text{ocjena}, \text{ispit}) = 5$$

$$V(\text{datIspit}, \text{ispit}) = 1500$$

$$V(\text{sifNast}, \text{ispit}) = 100$$

$$V(\text{sifPred}, \text{ispit}) = 200$$

$$\min(\text{sifNast}, \text{ispit}) = 101$$

$$\max(\text{sifNast}, \text{ispit}) = 200$$

$$N(\sigma_{\text{ocjena}=2}(\text{ispit})) = N(\text{ispit}) / V(\text{ocjena}, \text{ispit})$$

$$N(\sigma_{\text{datIspit}='16.4.2008'}(\text{ispit})) = N(\text{ispit}) / V(\text{datIspit}, \text{ispit})$$

$$N(r) \cdot ( N_{F_1}(r) \cdot N_{F_2}(r) \cdot \dots \cdot N_{F_m}(r) ) / ( N(r) )^m$$

$$N(\text{rez}) = N(\text{ispit}) / ( V(\text{ocjena}, \text{ispit}) \cdot V(\text{datIspit}, \text{ispit}) ) \approx 27$$

- za vježbu (ovaj i sljedeća dva zadatka) riješite korištenjem notacije selektivnosti predikata

# Primjer

```
CREATE TABLE ispit (
    mbrStud
, sifPred
, sifNast
, datIspit
, ocjena);
```

- procijeniti broj n-torki u rezultatu operacije:

$$\sigma_{\text{ocjena}=2 \wedge \text{sifNast} \leq 120}(\text{ispit})$$

$$N(\sigma_{\text{ocjena}=2}(\text{ispit})) = N(\text{ispit}) / V(\text{ocjena}, \text{ispit}) = 40\ 000$$

$$N(r) \cdot (v - \min(A, r)) / (\max(A, r) - \min(A, r))$$

$$N(\sigma_{\text{sifNast} \leq 120}(\text{ispit})) = N(\text{ispit}) \cdot 19 / 99 \approx 38\ 380$$

$$N(r) \cdot (N_{F_1}(r) \cdot N_{F_2}(r) \cdot \dots \cdot N_{F_m}(r)) / (N(r))^m$$

$$N(\text{rez}) \approx N(\text{ispit}) \cdot (40\ 000 \cdot 38\ 380) / N(\text{ispit})^2 \approx 7677$$

# Primjer

```
CREATE TABLE ispit (
    mbrStud
, sifPred
, sifNast
, datIspit
, ocjena);
```

- procijeniti broj n-torki u rezultatu operacije:

$$\sigma_{\text{sifNast}=120 \vee \text{sifPred}=320}(\text{ispit})$$

$$\begin{aligned} N(\text{ispit}) &= 200\,000 \\ V(\text{ocjena}, \text{ispit}) &= 5 \\ V(\text{datIspit}, \text{ispit}) &= 1500 \\ V(\text{sifNast}, \text{ispit}) &= 100 \\ V(\text{sifPred}, \text{ispit}) &= 200 \\ \min(\text{sifNast}, \text{ispit}) &= 101 \\ \max(\text{sifNast}, \text{ispit}) &= 200 \end{aligned}$$

$$N(\sigma_{\text{sifNast}=120}(\text{ispit})) = N(\text{ispit}) / V(\text{sifNast}, \text{ispit}) = 2\,000$$

$$N(\sigma_{\text{sifPred}=320}(\text{ispit})) = N(\text{ispit}) / V(\text{sifPred}, \text{ispit}) = 1\,000$$

$$N(r) \cdot (1 - (1 - N_{F_1}(r) / N(r)) \cdot (1 - N_{F_2}(r) / N(r)) \cdots (1 - N_{F_m}(r) / N(r)))$$

$$N(\text{rez}) = N(\text{ispit}) \cdot (1 - (1 - 2000 / 200000) \cdot (1 - 1000 / 200000)) = 2990$$

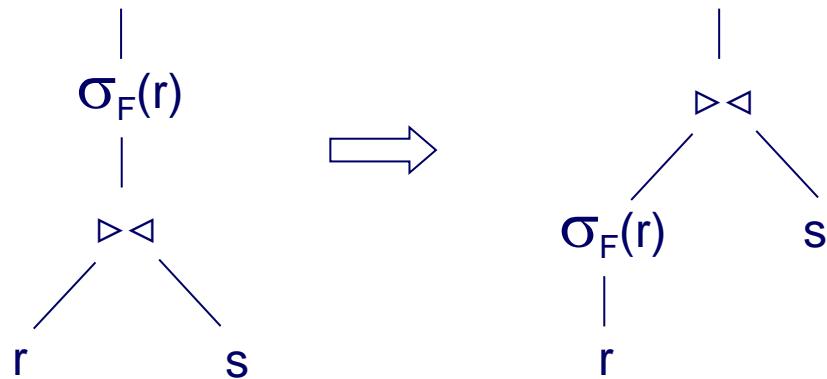
# **Optimizacija**

# Tehnike optimizacije

- **optimizacija temeljena na primjeni heurističkih pravila**
  - *heuristic optimization*
  - iz relacijske algebre poznata su pravila ekvivalencije (*equivalence rules*) kojima se jedan izraz relacijske algebre može transformirati u ekvivalentni izraz relacijske algebre
  - ovdje će se koristiti ona pravila ekvivalencije čijom se primjenom, u većini slučajeva (ali ne uvijek!), dobije bolji plan izvršavanja

**Primjer:** oznake:  $r(R)$  relacija  $r$  sa shemom  $R$ ;  $s(S)$  relacija  $s$  sa shemom  $S$

- potiskivanje selekcije: operaciju selekcije obaviti u čim ranijoj fazi
- ako  $F$  sadrži samo atribute iz  $R$ , tada  $\sigma_F(r \bowtie s) \equiv \sigma_F(r) \bowtie s$



# Tehnike optimizacije

---

- **optimizacija temeljena na procjeni troškova**
  - *cost based optimization*
  - optimizator izračunava ukupni trošak svakog ekvivalentnog plana izvršavanja (ili mnogih ekvivalentnih planova). Pri tome procjenjuje:
    - broj U/I operacija
    - veličina korištene glavne i sekundarne memorije
    - trošak procesorskog vremena
    - komunikacijski troškovi (za distribuirane baze podataka)
  - odabire plan izvršavanja s najmanjim procijenjenim ukupnim troškom
  - nedostatak: procjena troška obavlja se za veliki broj alternativnih planova izvršavanja. Trošak same optimizacije postaje značajan faktor
- kao jednostavan primjer optimizacije temeljene na procjeni troškova može poslužiti primjer procjene troškova u složenim operacijama relacijske algebre. U primjeru su troškovi procijenjeni za (samo) tri alternativna plana, te se usporedbom ukupnog troška moglo procijeniti koji je plan najbolji

# Tehnike optimizacije

---

- najčešće se ove dvije tehnike kombiniraju: primjenom heurističkih pravila se odredi ograničeni broj "potencijalno dobrih" planova izvršavanja, te se na tako ograničenom skupu planova provodi optimizacija temeljena na procjeni troškova
- sustavi za upravljanje bazama podataka omogućuju određeni stupanj kontrole nad odabirom tehnike optimizacije

## Primjer:

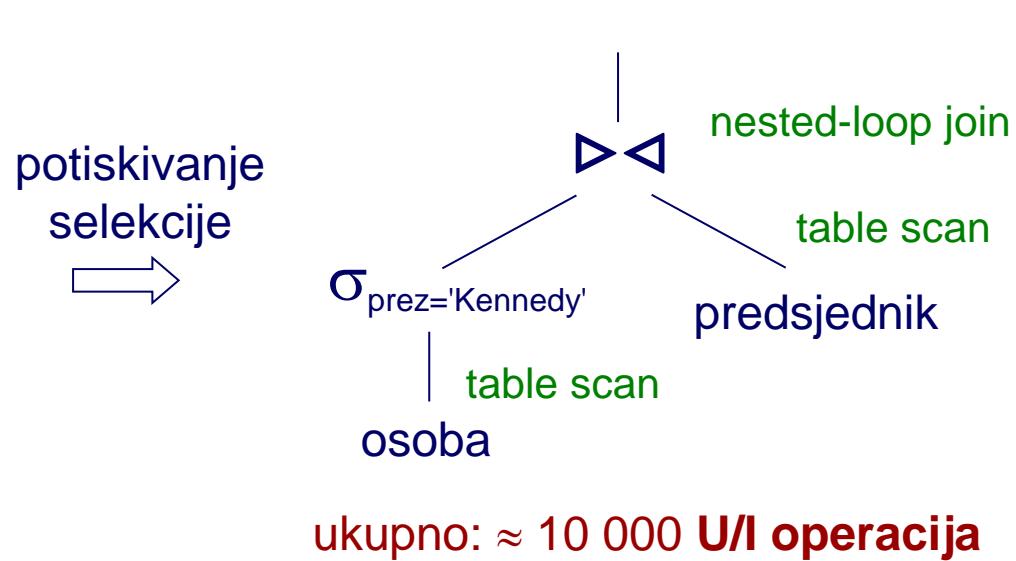
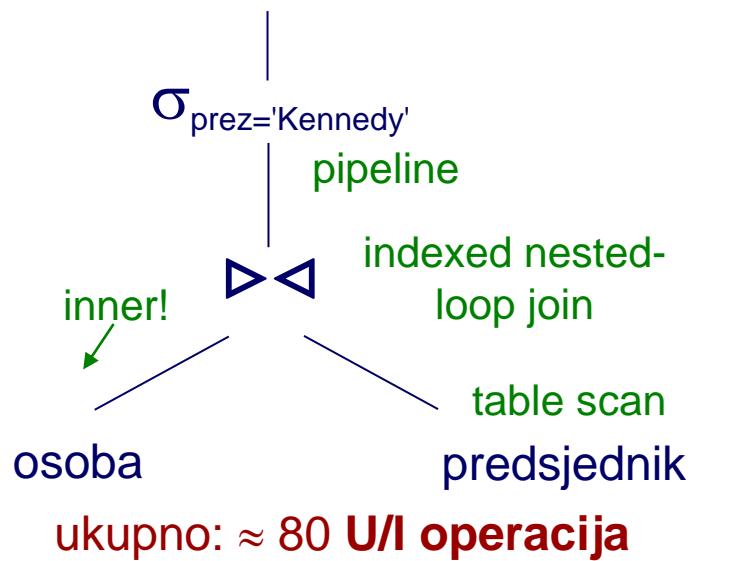
- *The algorithm that a SET OPTIMIZATION HIGH statement invokes is a sophisticated, cost-based strategy that examines all reasonable query plans and selects the best overall alternative. For large joins, this algorithm can incur more overhead than desired. In extreme cases, you can run out of memory.*
- *The alternative algorithm that a SET OPTIMIZATION LOW statement invokes eliminates unlikely join strategies during the early stages, which reduces the time and resources spent during optimization. However, when you specify a low level of optimization, the optimal strategy might not be selected because it was eliminated from consideration during early stages of the algorithm.*

# Samo heuristička optimizacija nije dovoljna

- optimizator se ne bi trebao osloniti samo na pravila za heurističku optimizaciju
- primjer pokazuje kako se nekriticom primjenom pravila heurističke optimizacije može dobiti loš plan

```
SELECT * FROM osoba JOIN predsjednik  
    ON osoba.sif = predsjednik.sifOso  
    WHERE prez = 'Kennedy' ;
```

- $N(\text{osoba}) = 1\ 000\ 000$
- $N(\text{predsjednik}) = 20$
- $V(\text{prez}, \text{predsjednik}) = 20$
- $V(\text{prez}, \text{osoba}) = 5\ 000$
- $\text{idx za } \text{osoba}.\text{sif}, d(\text{idx}) = 4$
- $B(\text{osoba}) = 10\ 000$
- $B(\text{predsjednik}) = 1$



# **Heuristička optimizacija**

# Pravila za transformaciju izraza relacijske algebre

- transformacije se temelje na pravilima ekvivalentnosti izraza relacijske algebre
- *equivalence rules, algebraic laws*
- ovdje će biti navedena samo najvažnija pravila za transformaciju, ona koja su važna u jednostavnijim postupcima heurističke optimizacije
- označke
  - relacije  $r(R)$ ,  $s(S)$ ,  $t(T)$ , ...
  - formule  $F$ ,  $F_1$ ,  $F_2$ ,  $F_3$ ,  $G$ , ...

## 1. Pravila vezana uz operaciju selekcije

$$1.1. \quad \sigma_{F_1 \wedge F_2}(r) \equiv \sigma_{F_1}(\sigma_{F_2}(r))$$

dekompozicija operatora selekcije s konjunktivnim oblikom formule

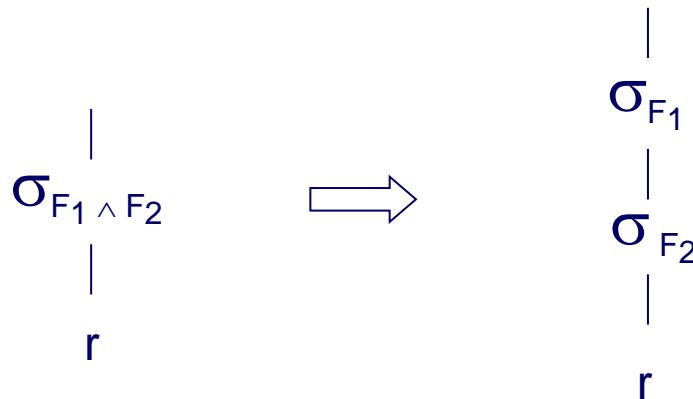
$$1.2. \quad \sigma_{F_1}(\sigma_{F_2}(r)) \equiv \sigma_{F_2}(\sigma_{F_1}(r))$$

promjena redoslijeda operacija selekcije

# Primjena na stablu upita

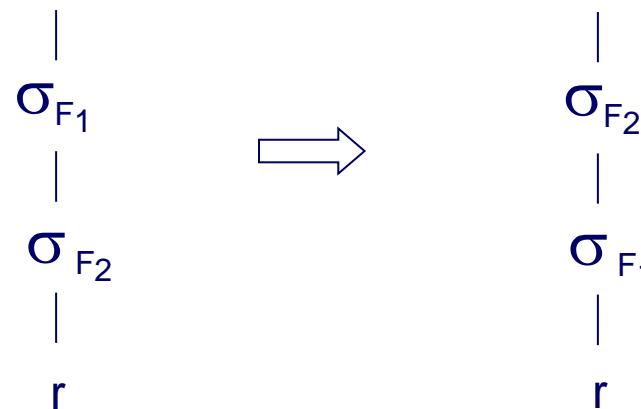
dekompozicija operatora selekcije s konjunktivnim oblikom formule

$$\sigma_{F_1 \wedge F_2}(r) \equiv \sigma_{F_1}(\sigma_{F_2}(r))$$



promjena redoslijeda operacija selekcije

$$\sigma_{F_1}(\sigma_{F_2}(r)) \equiv \sigma_{F_2}(\sigma_{F_1}(r))$$



# Pravila za transformaciju izraza relacijske algebre

- ako  $F$  sadrži samo atribute iz  $R$

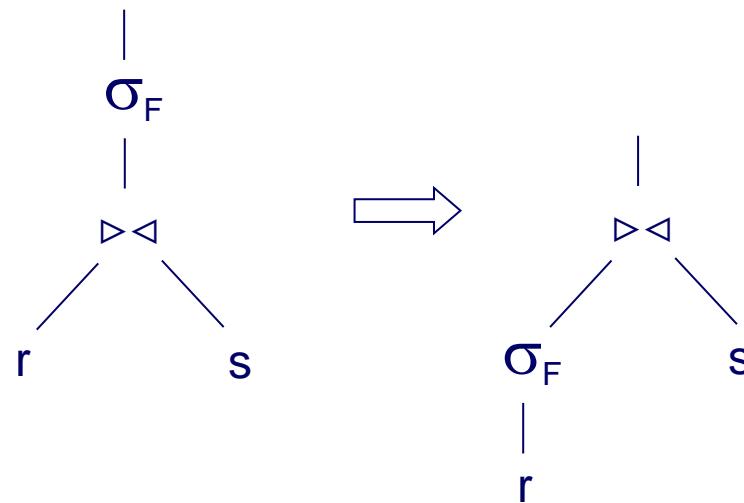
$$1.3. \quad \sigma_F(r \triangleright\!\triangleleft s) \equiv \sigma_F(r) \triangleright\!\triangleleft s$$

$$1.4. \quad \sigma_F \begin{matrix} (r \triangleright\!\triangleleft s) \\ G \end{matrix} \equiv \sigma_F \begin{matrix} (r) \\ G \end{matrix} \triangleright\!\triangleleft s$$

$$1.5. \quad \sigma_F(r \times s) \equiv \sigma_F(r) \times s$$

Prikaz često korištene transformacije: potiskivanje selekcije

$$\sigma_F(r \triangleright\!\triangleleft s) \equiv \sigma_F(r) \triangleright\!\triangleleft s$$

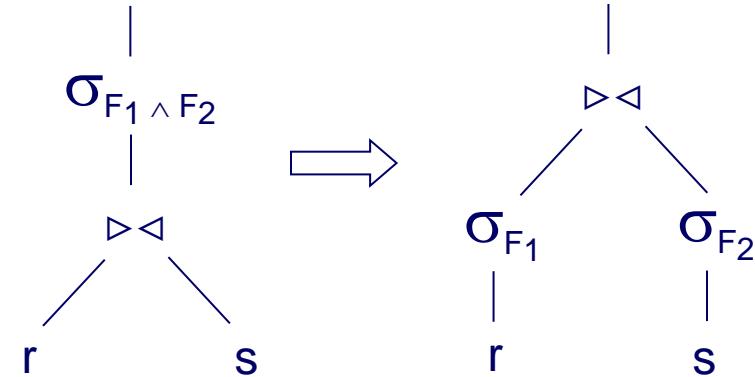


# Pravila za transformaciju izraza relacijske algebre

- slično, ako  $F_1$  sadrži samo attribute iz  $R$ , a  $F_2$  sadrži samo attribute iz  $S$

$$1.6. \quad \sigma_{F_1 \wedge F_2}(r \bowtie s) \equiv \sigma_{F_1}(r) \bowtie \sigma_{F_2}(s)$$

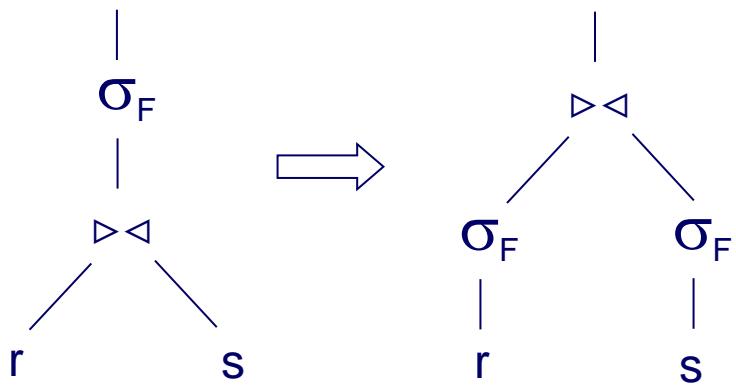
$$1.7. \quad \sigma_{F_1 \wedge F_2}(r \bowtie s) \underset{G}{\equiv} \sigma_{F_1}(r) \bowtie \sigma_{F_2}(s)$$



- slično, ako i  $R$  i  $S$  sadrže sve attribute iz  $F$

$$1.8. \quad \sigma_F(r \bowtie s) \equiv \sigma_F(r) \bowtie \sigma_F(s)$$

$$1.9. \quad \sigma_F(r \bowtie s) \underset{G}{\equiv} \sigma_F(r) \bowtie \sigma_F(s)$$

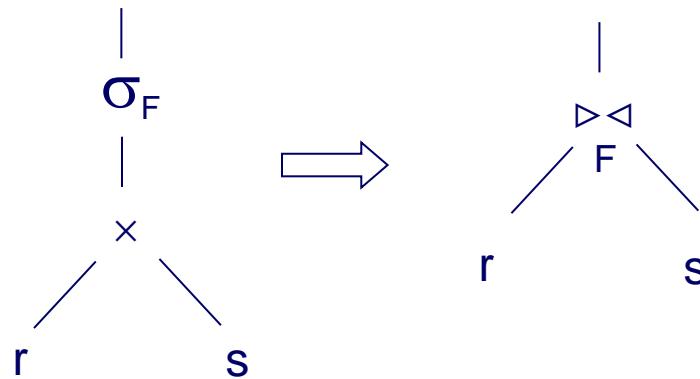


# Pravila za transformaciju izraza relacijske algebre

- važno pravilo o selekciji, Kartezijevom produktu i spajanju

1.10.

$$\sigma_F(r \times s) \equiv r \triangleright\!\!\! \triangleleft_s^F$$



# Pravila za transformaciju izraza relacijske algebre

## 2. Komutativnost

$$2.1. \quad r \times s \equiv s \times r$$

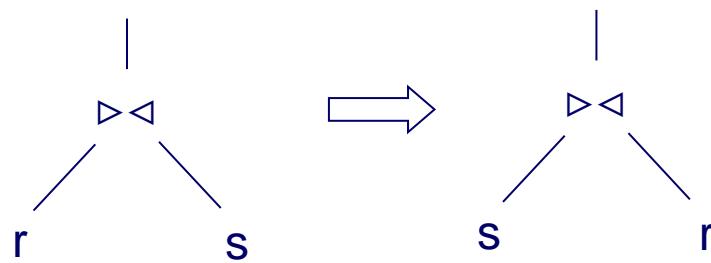
$$2.2. \quad r \triangleright\triangleleft s \equiv s \triangleright\triangleleft r$$

$$2.3. \quad r \triangleright\triangleleft s \equiv s \triangleright\triangleleft r$$

F

F

Primjer: slobodna promjena redoslijeda operanada



# Pravila za transformaciju izraza relacijske algebre

## 3. Asocijativnost

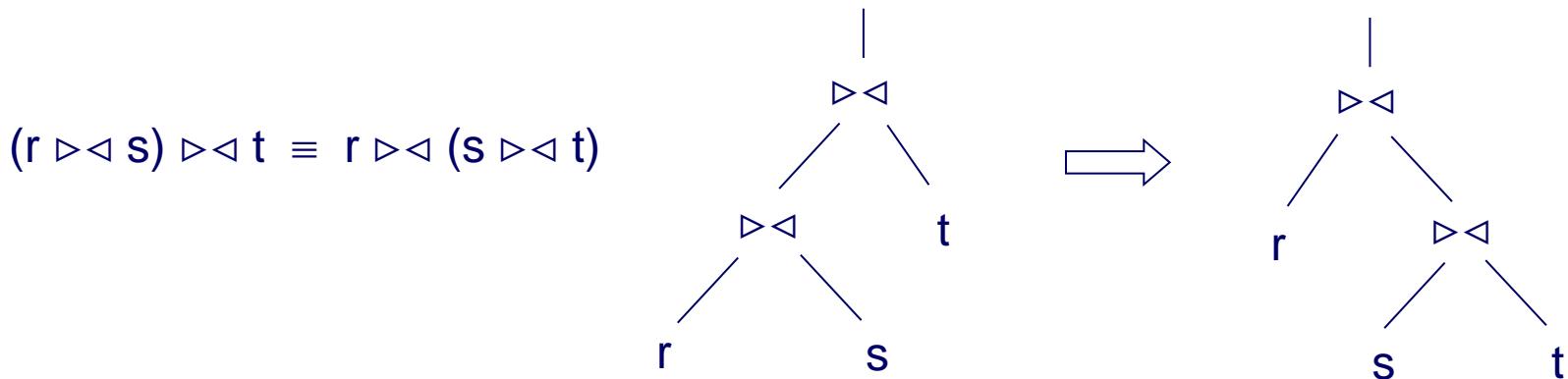
$$3.1. \quad (r \times s) \times t \equiv r \times (s \times t)$$

$$3.2. \quad (r \triangleright\triangleleft s) \triangleright\triangleleft t \equiv r \triangleright\triangleleft (s \triangleright\triangleleft t)$$

$$3.3. \quad (r \triangleright\triangleleft s) \triangleright\triangleleft t \equiv r \triangleright\triangleleft (s \triangleright\triangleleft t) \quad \text{uz uvjet da } F_2 \text{ sadrži samo atributе iz } S \text{ i } T$$

$F_1 \quad F_2 \wedge F_3 \quad F_1 \wedge F_3 \quad F_2$

**Primjer:** često korištena transformacija: promjena redoslijeda spajanja



# Osnovni koraci u heurističkoj optimizaciji upita

1. dekomponirati operatore selekcije koji sadrže konjunktivne izraze
  - postiže se veći stupanj slobode u pomicanju operatora selekcije unutar stabla upita
2. potisnuti operatore selekcije što je moguće dublje prema listovima stabla upita
  - obavljanjem operacija selekcije u najranijim mogućim fazama smanjuje se veličina međurezultata koji se koristi u dalnjim operacijama što najčešće dovodi do smanjenja ukupnog troška
3. Karteziјev produkt kombinirati s operacijom selekcije u operaciju spajanja uz uvjet ( $\theta$ -join)
4. redoslijed spajanja odrediti tako da se prvo obavljaju operacije spajanja koje rezultiraju najmanjim brojem n-torki (ili veličinom međurezultata)
  - ovdje su navedena tek najvažnija (najčešće korištena) pravila

Navedena pravila koristiti za rješavanje zadataka u kojima se traži heuristička optimizacija

# Primjer

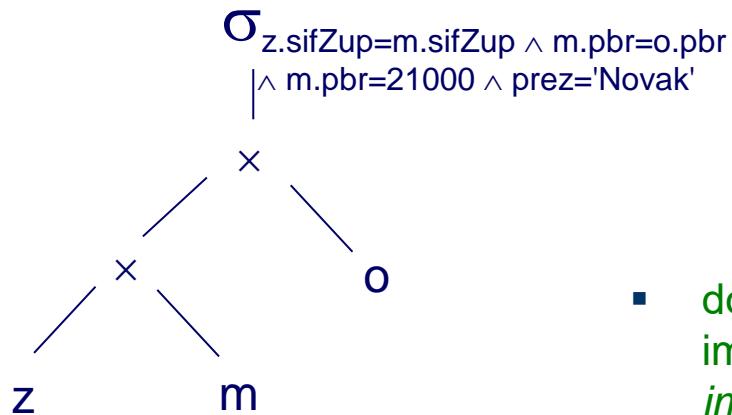
```
SELECT *
  FROM zup, mjesto, osoba
 WHERE zup.sifZup = mjesto.sifZup
   AND mjesto.pbr = osoba.pbr
   AND mjesto.pbr = 21000
   AND osoba.prez = 'Novak' ;
```

**zup**  
sifZup  
nazZup

**mjesto**  
pbr  
nazMj  
sifZup

**osoba**  
sifOso  
nazMj  
ime  
prez  
pbr

- nacrtati inicijalno stablo upita. Redoslijed operacija Kartezijevog produkta (uočiti da originalni SQL upit ne sadrži operacije spajanja!) u inicijalnom stablu mora odgovarati redoslijedu kojim su relacije navedene u upitu
- provesti heurističku optimizaciju



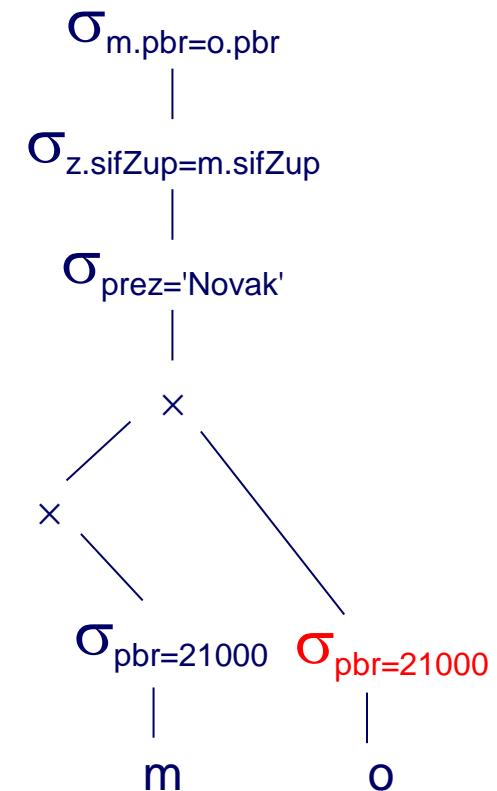
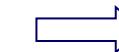
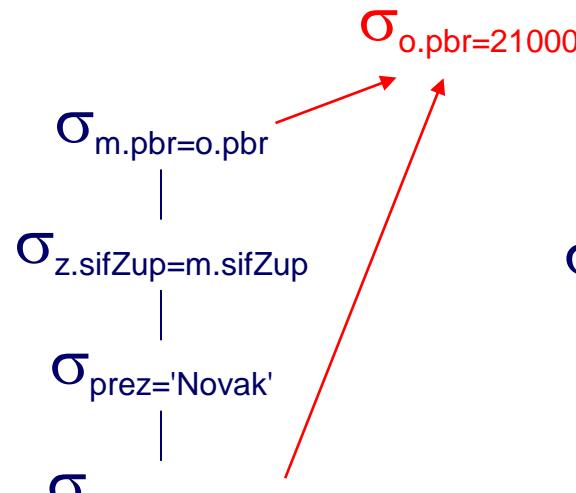
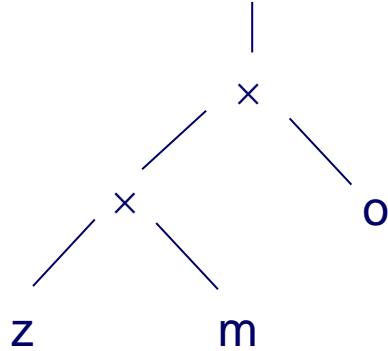
**z** : **zup**  
**m** : **mjesto**  
**o** : **osoba**

- dogovor: tamo gdje je to nužno, imena atributa se kvalificiraju s *imeTablice.imeAtributa*

# Primjer

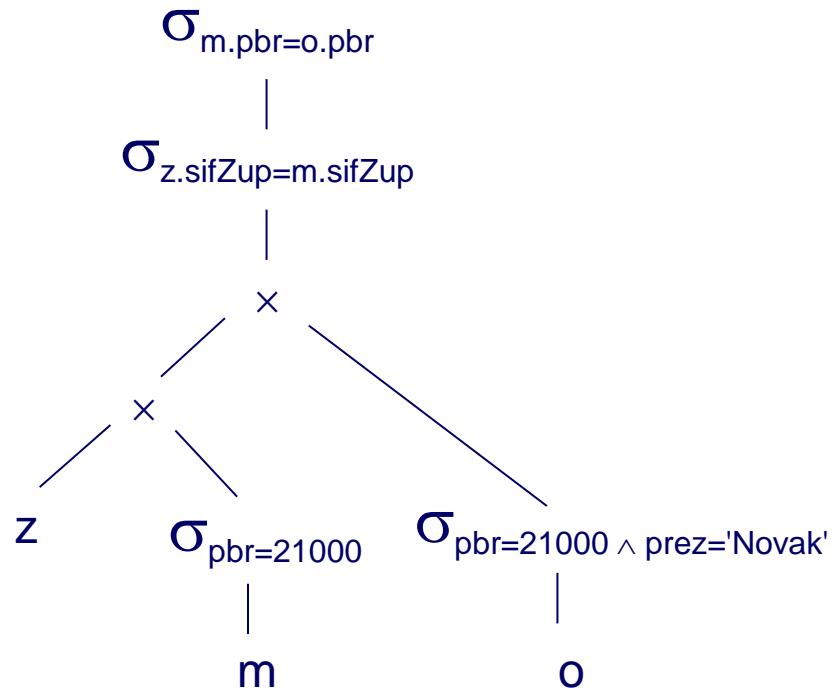
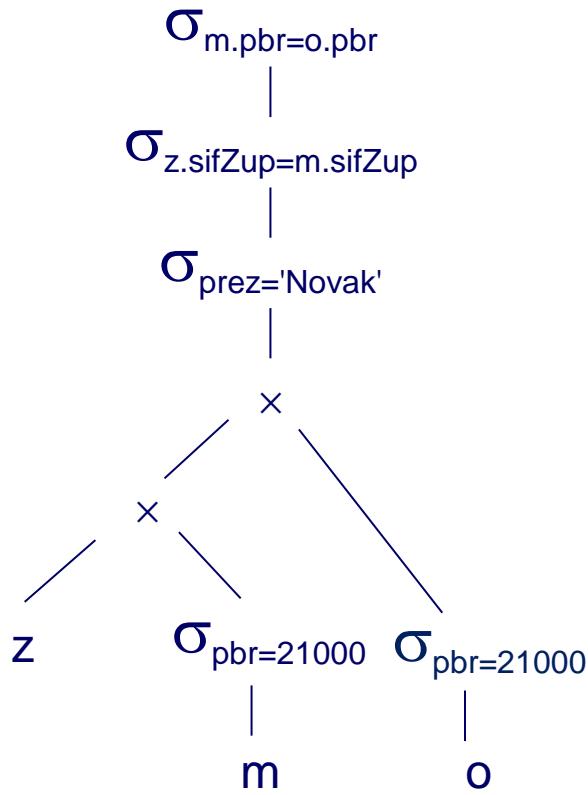
$z : zup$   
 $m : mjesto$   
 $o : osoba$

$$\sigma_{z.sifZup=m.sifZup \wedge m.pbr=o.pbr \wedge m.pbr=21000 \wedge prez='Novak'}$$

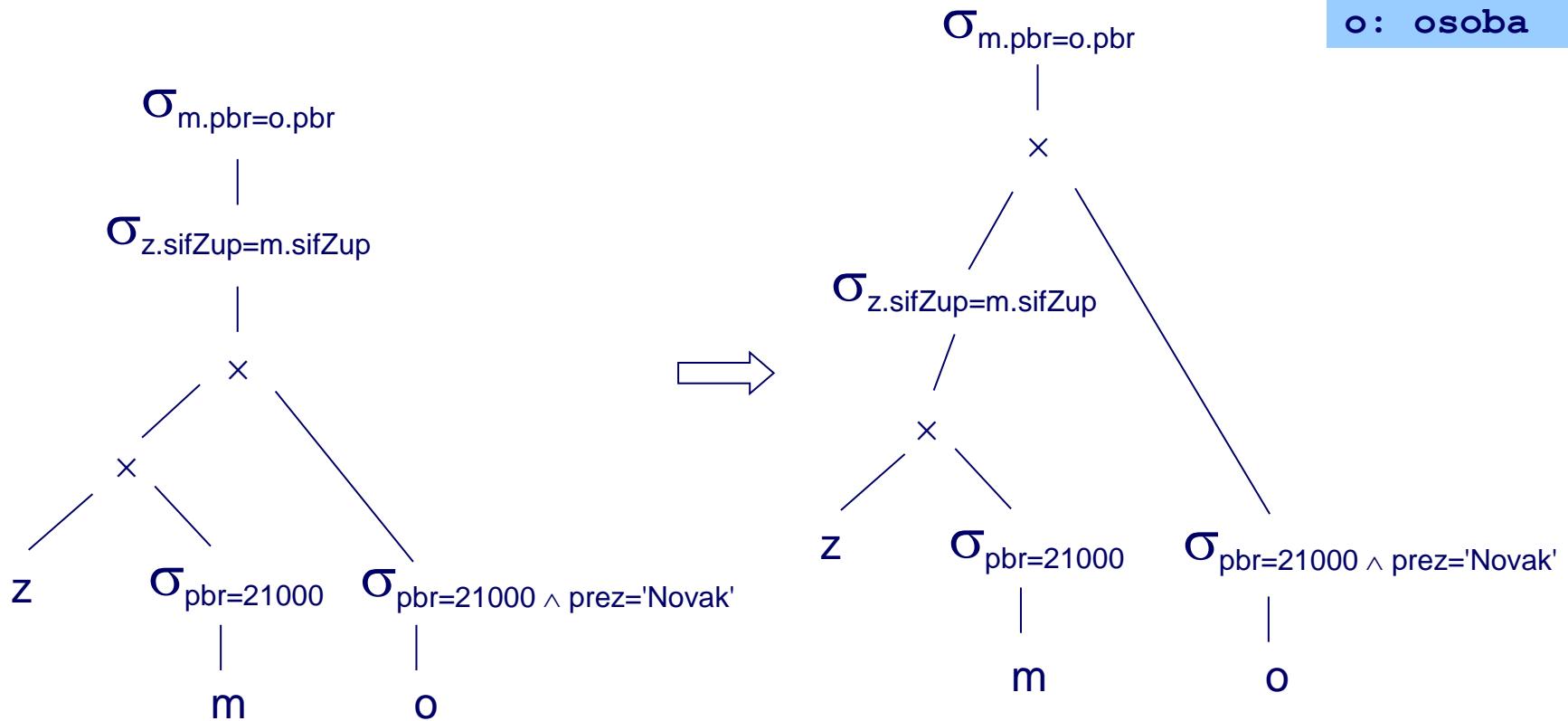


# Primjer

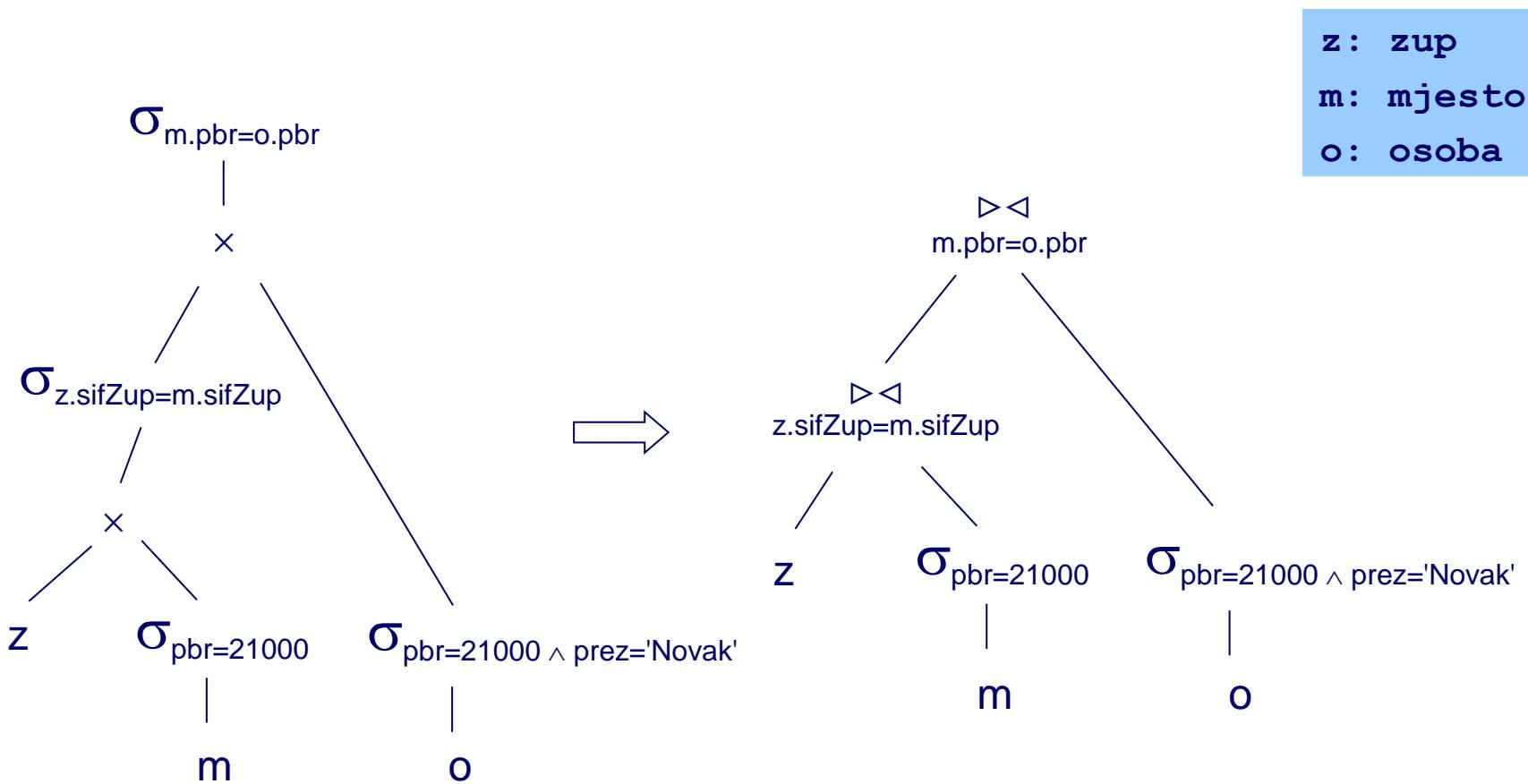
**z : zup  
m : mjesto  
o : osoba**



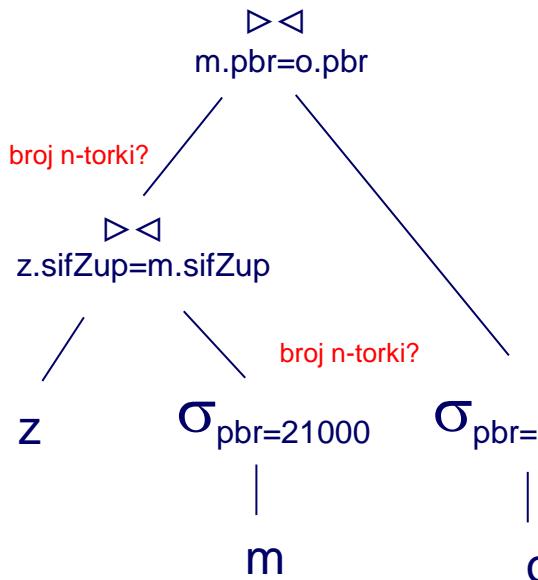
# Primjer



# Primjer



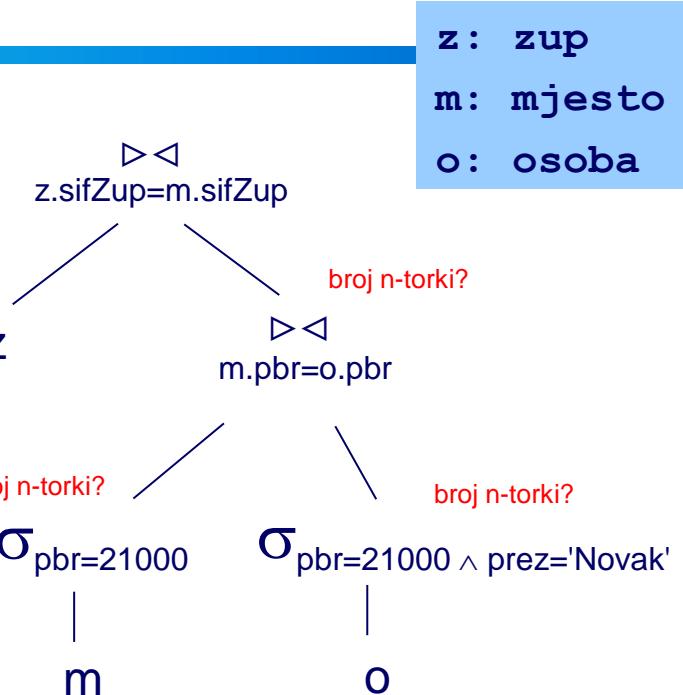
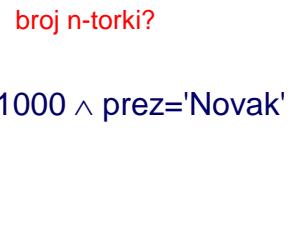
# Primjer



**redoslijed spajanja?**

Prema asocijativnosti  
- pravilo 3.3. na str. 40

ILI



**z : zup**  
**m : mjesto**  
**o : osoba**

- za vježbu! Radi pojednostavljenja, pri procjeni zanemariti sve troškove osim troškova pisanja međurezultata. Pretpostaviti da se svaki međurezultat mora materijalizirati. Zanemariti veličinu n-torke te trošak izraziti u broju n-torki. Usporediti ukupni broj n-torki u međurezultatima ovih planova. Na raspolaganju su sljedeći podaci iz rječnika podataka:

$$PK(z) = \text{sifZup}$$

$$N(z) = 20$$

$$V(pbr, o) = 10$$

$$PK(m) = pbr$$

$$N(m) = 2\ 000$$

$$V(\text{prez}, o) = 50$$

$$PK(o) = \text{sifOso}$$

$$N(o) = 200\ 000$$

# Primjer prikaza plana izvršavanja (IBM Informix)

Informativno

```
SELECT *
  FROM ispit
    , stud
    , mjesto
 WHERE mjesto.pbr = stud.pbrStan
   AND stud.mbr = ispit.mbrStud
   AND mjesto.pbr = 10805
   AND ispit.ocjena = 1;
```

Estimated Cost: 8409

Estimated # of Rows Returned: 42

```
1) informix.stud: SEQUENTIAL SCAN
   Filters: informix.stud.pbrstan = 10805
2) informix.mjesto: SEQUENTIAL SCAN
   Filters:
     Table Scan Filters: informix.mjesto.pbr = 10805
DYNAMIC HASH JOIN
  Dynamic Hash Filters: informix.mjesto.pbr = informix.stud.pbrstan
3) informix.ispit: SEQUENTIAL SCAN
   Filters:
     Table Scan Filters: informix.ispit.ocjena = 1

DYNAMIC HASH JOIN (Build Outer)
  Dynamic Hash Filters: informix.stud.mbr = informix.ispit.mbrstud
```

- Procijenjeni trošak: **8409**
- Procijenjeni broj n-torki u rezultatu: **42**
- (1) *table scan* za stud, uz filter pbrstan = 10805 → međurezultat 1
- (2) *table scan* za mjesto, uz filter pbr = 10805 → međurezultat 2
  - prethodna dva međurezultata (1 i 2) spoji pomoću *hash join* → međurezultat 3
- (3) *table scan* za ispit, uz filter ocjena = 1 → međurezultat 4
  - prethodna dva međurezultata (3 i 4) spoji pomoću *hash join*

# Primjer prikaza plana izvršavanja (IBM Informix)

Informativno

Query statistics:

Table map :

| Internal name | Table name |
|---------------|------------|
| t1            | stud       |
| t2            | mjesto     |
| t3            | ispit      |

| type | table | rows_prod | est_rows | rows_scan | time     | est_cost |
|------|-------|-----------|----------|-----------|----------|----------|
| scan | t1    | 8         | 16       | 5000      | 00:00.01 | 369      |

| type | table | rows_prod | est_rows | rows_scan | time     | est_cost |
|------|-------|-----------|----------|-----------|----------|----------|
| scan | t2    | 1         | 1        | 1000      | 00:00.00 | 46       |

| type  | rows_prod | est_rows | rows_bld | rows_prb | novrflo | time     | est_cost |
|-------|-----------|----------|----------|----------|---------|----------|----------|
| hjoin | 8         | 9        | 1        | 8        | 0       | 00:00.01 | 417      |

...

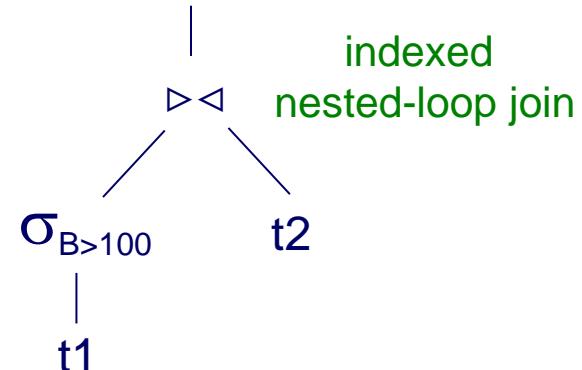
- nakon izvršavanja upita, optimizator u izvještaj o planu izvršavanja dodaje i statističke podatke o upitu, npr.
- za dio koji se odnosi na operaciju *table scan* nad relacijom stud
  - procijenjeni trošak: 369
  - procijenjeni broj n-torki u rezultatu: 16
  - stvarni broj n-torki u rezultatu (poznato tek nakon izvršavanja upita): 8
  - broj pročitanih n-torki: 5000

# Upute optimizatoru (*optimizer hints*, *optimizer directives*)

- mogućnost koju treba koristiti vrlo štedljivo jer:
  - uvodenjem uputa optimizatoru odgovornost za "optimalno" izvršavanje upita prebacuje se sa SUBP-a na administratora (ili programera ili korisnika)
  - ne postoji standardna sintaksa

Primjer:

```
SELECT *
  FROM t1 JOIN t2
    ON t1.A = t2.A
   WHERE t1.B > 100;
```



- postoji indeks nad atributom  $t2.a$
- ako optimizator procijeni da će uvjet  $t1.B > 100$  vratiti veliki broj n-torki iz  $t1$ , neće koristiti *indexed nested-loop join* i radije primjeniti *hash join*. **Zašto?**
- ako administrator ili programer (ili vrlo napredni korisnik), prema značenju podataka ocijeni da će rezultat selekcije ipak sadržavati relativno malen broj n-torki, može uz SELECT naredbu optimizatoru uputiti sljedeću direktivu:

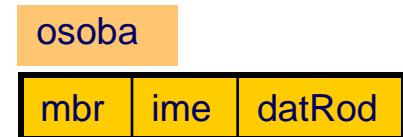
```
SELECT {+USE_NL(t2)} *
  FROM t1 JOIN t2
    ON t1.A = t2.A
   WHERE t1.B > 100;
```

- uputa: spajanje s  $t2$  obavi pomoću *nested-loop join*

# Funkcijski indeks

- koristi li se indeks `idxDat` u sljedećem upitu?

```
CREATE INDEX idxDat ON osoba (datRod);
SELECT * FROM osoba
    WHERE DAY(datRod) BETWEEN 30 AND 31;
```



- NE, slijedna pretraga

- postoji rješenje - koristiti funkciji indeks:

```
CREATE FUNCTION danMjeseca(datum DATE)
    RETURNING INTEGER WITH(NOT VARIANT)
    RETURN DAY(datum);
END FUNCTION;
CREATE INDEX idxDat ON osoba (danMjeseca(datRod));
SELECT * FROM osoba
    WHERE danMjeseca(datRod) BETWEEN 30 AND 31;
```

- Ključevi u čvorovima B-stabla su cijeli brojevi koji predstavljaju redni broj dana u mjesecu
- Prikazan je primjer za sustav IBM Informix. Ne treba učiti sintaksu, važno je samo razumijevanje ideje funkcijskog indeksa

# Literatura:

---

- H. Garcia-Molina, J. Ullman, J. Widom. Database system implementation. Prentice-Hall International, Inc. Upper Saddle River, New Jersey. 2000.
- A. Silberschatz, H. Korth, S. Sudarshan. Database system concepts, 6th ed. McGraw-Hill. 2010.
- R. Elmasri, S. Navathe. Fundamentals of database systems, 3rd ed. Addison-Wesley Publishing Company. Reading, Massachusetts. 2000.
- IBM Informix Dynamic Server Performance Guide, Version 11.50, IBM, 2008.

# **Sustavi baza podataka**

Predavanja

**5. Transakcija**

ožujak 2023.

# Transakcija

- Izvršavanje programa koji pristupa bazi podataka radi obavljanja neke administrativne (poslovne) funkcije  
Bernstein, Newcomer: Principles of Transaction Processing
- Niz SQL naredbi koje mijenjaju podatke i koje se moraju izvršiti u cijelosti ili se (u cijelosti) ne smiju izvršiti  
IBM Informix Guide to SQL: Tutorial
- Logička jedinica posla (*logical unit of work*) koja sadrži jednu ili više SQL naredbi. Transakcija grupira SQL naredbe tako da su te naredbe potvrđene (primijenjene na bazi podataka), ili poništene.  
Oracle 11g Release 2 (11.2)

**Definicija:** Transakcija je niz logički povezanih operacija koje se izvršavaju kao cjelina i prevode bazu podataka iz jednog u drugo **konzistentno stanje\***

\* baza podataka je konzistentna ako zadovoljava sva definirana integritetska ograničenja

# Transakcija

## Primjer:

- zadani iznos sredstava prebaciti s prvog zadanog na drugi zadani račun (stanje prvog računa umanjiti, a drugog računa uvećati).

| racun |         |
|-------|---------|
| brRac | stanje  |
| 101   | 1000.00 |
| 102   | 500.00  |
| 103   | 2500.00 |
| 104   | 200.00  |

Logički povezane operacije:

umanjiti stanje na prvom računu (SQL: UPDATE)  
uvećati stanje na drugom računu (SQL: UPDATE)

Ili obje UPDATE operacije moraju biti obavljene, ili niti jedna!

# Sintaksa - sa stanovišta korisnika

---

- transakcija se opisuje kao program ili odsječak programa, SPL, PL/SQL, Transact SQL, Java+JDBC, C+ODBC, itd, kojim se izvršavaju SQL naredbe:
  - **SELECT, UPDATE, INSERT, DELETE, ...**
  - **BEGIN [WORK], COMMIT [WORK], ROLLBACK [WORK]**
- **BEGIN [WORK]** nije u skladu s ISO/ANSI SQL standardom
  - ISO/ANSI SQL: **START TRANSACTION**
- transakcije se ne mogu ugnježđivati - svaka korisnička sjednica (*user session*), može u jednom trenutku imati najviše jednu aktivnu transakciju

# Sintaksa - sa stanovišta korisnika

## Primjer:

- napisati programski kôd za transakciju kojom se zadani iznos sredstava prebacuje s prvog zadanog na drugi zadani račun (stanje prvog računa umanjiti, a drugog računa uvećati).

|                  |               |                               |
|------------------|---------------|-------------------------------|
| <b>variable:</b> | <b>brRac1</b> | ← broj prvog računa           |
|                  | <b>brRac2</b> | ← broj drugog računa          |
|                  | <b>iznos</b>  | ← iznos kojeg treba prebaciti |

| racun | brRac | stanje  |
|-------|-------|---------|
|       | 101   | 1000.00 |
|       | 102   | 500.00  |
|       | 103   | 2500.00 |
|       | 104   | 200.00  |

# Sintaksa - ako se koristi implicitni početak transakcije

Oracle (PL/SQL) ili IBM Informix (SPL)

- **implicitni početak transakcije:** početkom transakcije smatra se prva SQL naredba izvršena u okviru korisničke sjednice ili prva SQL naredba izvršena neposredno nakon završetka prethodne transakcije
- **kraj transakcije:** mora biti eksplicitno zadan
  - COMMIT [ WORK ] ili ROLLBACK [ WORK ] naredbe iz ISO/ANSI SQL standarda
  - ako korisnička sjednica završi prije nego se obavi COMMIT ili ROLLBACK, podrazumijeva se ROLLBACK

Primjer:

```
-- započela je korisnička sjednica
--> ovdje se početak transakcije podrazumijeva
UPDATE racun SET stanje = stanje - iznos WHERE brRac = brRac1;
UPDATE racun SET stanje = stanje + iznos WHERE brRac = brRac2;
COMMIT WORK; -- ili ROLLBACK WORK
--> ovdje se početak transakcije podrazumijeva
sljedeća SQL naredba;
```

# Sintaksa - ako se koristi eksplisitni početak transakcije

IBM Informix (SPL)

- **početak transakcije:** mora biti eksplisitno zadan
  - BEGIN [ WORK ] ili START TRANSACTION
- **kraj transakcije:** mora biti eksplisitno zadan
  - COMMIT [ WORK ] ili ROLLBACK [ WORK ]
  - ako korisnička sjednica završi prije nego se obavi COMMIT ili ROLLBACK, SUBP će transakciju poništiti (kao da je obavljena naredba ROLLBACK)

**Primjer:**

```
BEGIN WORK;  
UPDATE racun SET stanje = stanje - iznos WHERE brRac = brRac1;  
UPDATE racun SET stanje = stanje + iznos WHERE brRac = brRac2;  
COMMIT WORK;
```

# Sintaksa - bez određivanja granica transakcija

IBM Informix (SPL)

- ako granice transakcija nisu određene (niti implicitnim početkom i eksplisitnim završetkom, niti eksplisitnim početkom i završetkom), smatra se da transakcija započinje neposredno prije početka, a završava neposredno nakon završetka svake zasebne SQL naredbe. Podrazumijeva se COMMIT ako je naredba uspješno završila, inače se podrazumijeva ROLLBACK. Sljedeći odsječak stoga ne garantira konzistentnost baze podataka.

Primjer:

```
-- ovdje se podrazumijeva početak transakcije (kao da je zaista obavljen BEGIN)
UPDATE racun SET stanje = stanje - iznos WHERE brRac = brRac1;
-- ovdje se podrazumijeva kraj transakcije i to:
    COMMIT ako je SQL naredba obavljena bez pogreške, ROLLBACK inače

-- ovdje se podrazumijeva početak transakcije (kao da je zaista obavljen BEGIN)
UPDATE racun SET stanje = stanje - iznos WHERE brRac = brRac1;
-- ovdje se podrazumijeva kraj transakcije i to:
    COMMIT ako je SQL naredba obavljena bez pogreške, ROLLBACK inače
```

- `setAutoCommit(false)` : implicitni početak transakcije. Kraj transakcije i ovdje mora biti eksplisitno zadan

Java varijable:

`brRac1` ← broj prvog računa  
`brRac2` ← broj drugog računa  
`iznos` ← iznos kojeg treba prebaciti  
`conn` ← *Connection* objekt

```
Statement stmt = conn.createStatement();
conn.setAutoCommit(false);

// transakcija započinje prvom izvršnom SQL naredbom
stmt.executeUpdate("UPDATE racun SET stanje = stanje - " + iznos +
                    " WHERE brRac = " + brRac1 );
stmt.executeUpdate("UPDATE racun SET stanje = stanje + " + iznos +
                    " WHERE brRac = " + brRac2 );
conn.commit(); // ili conn.rollback()

// pozivom sljedeće execute... metode implicitno započinje nova transakcija
...
```

- `setAutoCommit(true)` : granice transakcija nisu određene. Svaka DML naredba predstavlja zasebnu transakciju. Sljedeći odsječak stoga ne garantira konzistentnost!

### Java varijable:

|                     |                               |
|---------------------|-------------------------------|
| <code>brRac1</code> | ← broj prvog računa           |
| <code>brRac2</code> | ← broj drugog računa          |
| <code>iznos</code>  | ← iznos kojeg treba prebaciti |
| <code>conn</code>   | ← <i>Connection</i> objekt    |

```
Statement stmt = conn.createStatement();
conn.setAutoCommit(true);

// ovdje se podrazumijeva BEGIN
stmt.executeUpdate("UPDATE racun SET stanje = stanje - " + iznos +
                   " WHERE brRac = " + brRac1 );
// ovdje se podrazumijeva COMMIT/ROLLBACK

// ovdje se podrazumijeva BEGIN
stmt.executeUpdate("UPDATE racun SET stanje = stanje + " + iznos +
                   " WHERE brRac = " + brRac2 );
// ovdje se podrazumijeva COMMIT/ROLLBACK
```

# Sintaksa - sa stanovišta SUBP-a

- sa stanovišta SUBP-a, transakcija je niz operacija čitanja i/ili pisanja (u ili iz elemenata baze podataka) koje su omeđene transakcijskim operacijama
  - operacije čitanja ili pisanja (*database operations*)
    - **read(x, p)**, **write(y, p)**
    - u varijablu p učitaj vrijednost elementa x, u element y upiši vrijednost varijable p
  - transakcijske operacije (*transaction operations*)
    - **start**: početak nove transakcije
    - **commit**: završetak transakcije uz trajnu pohranu svih efekata transakcije
    - **abort**: prekid transakcije (kojeg prati poništavanje svih efekata transakcije)

```
start;  
read(x, p);  
r ← p * 10;  
write(x, r);  
commit;
```

r[x], w[x], c

**Model transakcije**

r[x] → w[x] → c

# ACID svojstva transakcije

---

- SUBP mora osigurati sljedeća svojstva transakcija:

***Atomicity*** Atomarnost

***Consistency*** Konzistentnost

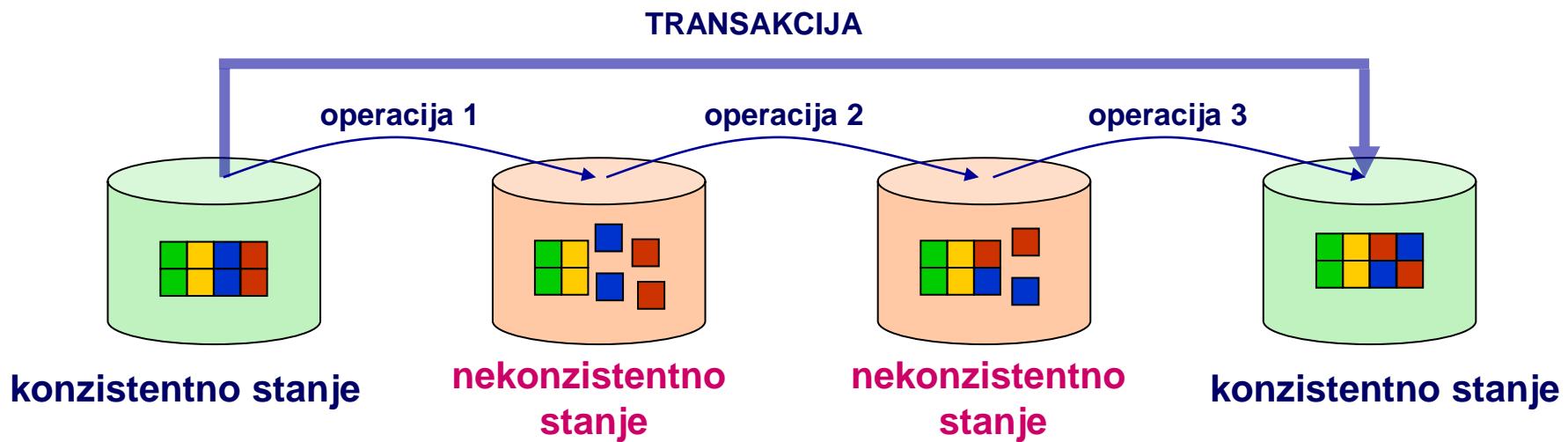
***Isolation*** Izolacija

***Durability*** Izdržljivost

# Konzistentnost

- transakcija je **korektna** ako konzistentnu bazu podataka, u odsustvu drugih transakcija i pogrešaka\*, prevodi iz jednog u drugo konzistentno stanje

\* pogreške će se detaljnije razmatrati u poglavlju o obnovi sustava



- Konzistentnost (*Consistency*)** - transakcija zadovoljava svojstvo konzistentnosti ako je korektna

# Konzistentnost

## Primjer:

- prebaciti zadani **iznos** sredstava s računa **brRac1** na račun **brRac2**
- korektna transakcija

```
BEGIN WORK;  
UPDATE racun SET stanje = stanje - iznos WHERE brRac = brRac1;  
UPDATE racun SET stanje = stanje + iznos WHERE brRac = brRac2;  
COMMIT WORK;
```

- nije korektna transakcija (posljedica npr. logičke pogreške u programu)

```
BEGIN WORK;  
UPDATE racun SET stanje = stanje - iznos WHERE brRac = brRac1;  
UPDATE racun SET stanje = stanje - iznos WHERE brRac = brRac2;  
COMMIT WORK;
```

# Konzistentnost

---

- na koji način osigurati svojstvo konzistentnosti?
- osigurati korektnost transakcije:
  - kad god je moguće koristiti eksplicitno opisana integritetska ograničenja
    - CHECK, PRIMARY, UNIQUE, FOREIGN KEY
    - okidači
    - pohranjene procedure
  - odgovornost programera
  - ili naprednog korisnika

# Izolacija

## Primjer:

- početno stanje računa A je 1000 kn. Jedna iza druge se obavljaju dvije transakcije
- transakcija kojom se 600 kn isplaćuje s računa A

```
BEGIN WORK;
SELECT stanje INTO p1 FROM racun WHERE brRac = 'A';
LET p1 = p1 - 600;
UPDATE racun SET stanje = p1 WHERE brRac = 'A';
COMMIT WORK;  novo stanje na računu A: 400 kn
```

- transakcija kojom se 2000 kn uplaćuje na račun A

```
BEGIN WORK;
SELECT stanje INTO p2 FROM racun WHERE brRac = 'A';
LET p2 = p2 + 2000;
UPDATE racun SET stanje = p2 WHERE brRac = 'A';
COMMIT WORK;  novo stanje na računu A: 2400 kn
```

- transakcije su korektne - nakon njihovog obavljanja baza podataka je konzistentna

# Izolacija

- ako SUBP operacije tih istih transakcija izvodi istodobno (naizmjence operacije jedne i druge transakcije u dvije korisničke sjednice)

```
BEGIN WORK;
SELECT stanje INTO p1 FROM racun WHERE brRac = 'A';
LET p1 = p1 - 600;
BEGIN WORK;
SELECT stanje INTO p2 FROM racun WHERE brRac = 'A';
LET p2 = p2 + 2000;
UPDATE racun SET stanje = p2 WHERE brRac = 'A';      -- stanje=3000
UPDATE racun SET stanje = p1 WHERE brRac = 'A';      -- stanje=400
COMMIT WORK;
COMMIT WORK;
```

novo stanje na računu A: 400 kn

- unatoč tome što su pojedinačne transakcije korektne, ako se njihove operacije obavljaju naizmjence, mogu narušiti konzistentnost baze podataka
- **Izolacija (Isolation)** - učinak transakcija koje se obavljaju istodobno (paralelno) mora biti jednak učinku tih istih transakcija koje bi se obavljale jedna iza druge (serijski)

# Atomarnost

## Primjer:

- prebaciti zadani **iznos** sredstava s računa **brRac1** na račun **brRac2**

```
BEGIN WORK;  
UPDATE racun SET stanje = stanje - iznos WHERE brRac = brRac1;  
UPDATE racun SET stanje = stanje + iznos WHERE brRac = brRac2;  
COMMIT WORK;
```

Primijetiti: baza podataka u ovom trenutku nije konzistentna

- bez obzira na moguće događaje tijekom izvršavanja transakcije (nestanak napajanja, kvar računala, ...) dopušten je samo jedan od dva ishoda:
  - efekti obje naredbe pohranjeni su u bazi podataka
  - baza podataka je u stanju u kakvom je bila prije početka transakcije
- Atomarnost (Atomicity)** - ili su efekti svih operacija transakcije pravilno pohranjeni u bazi podataka ili transakcija nije utjecala na stanje baze podataka
  - all or nothing*

# Atomarnost transakcije i interakcija s "vanjskim svijetom"

---

- budući da su operacije pisanja koje se obavljaju tijekom transakcije pod nadzorom SUBP-a, njihove efekte SUBP po potrebi može i poništiti (pojednostavljeni: vraćanjem starih vrijednosti elemenata). Kako to SUBP obavlja opisano je u sljedećim predavanjima
- problem nastaje kada se u okviru transakcije obavljaju i operacije koje utječu na okolinu SUBP-a (terminal ili slična U/I naprava). Tada svojstvo atomarnosti transakcije (za operacije izvršene izvan sustava) može biti dovedeno u pitanje

# Atomarnost transakcije i interakcija s "vanjskim svijetom"

Primjer:

- Što bi se moglo dogoditi kada bi bankomat (ATM) radio ovako?

```
BEGIN WORK
IF (SELECT stanje FROM racun
    WHERE brRac = p_brRac) >= p_iznos THEN
    UPDATE racun SET stanje = stanje - p_iznos
        WHERE brRac = p_brRac
    SYSTEM "prebroji_i_predaj " || iznos
    COMMIT WORK
ELSE
    ROLLBACK WORK
END IF
```

- bi li pomoglo kad bi se operacija **SYSTEM** premjestila  
iza naredbe **COMMIT**?

# Izdržljivost

- neka je sljedeća transakcija uspješno obavljena, tj. njezini efekti su u cijelosti pohranjeni u bazi podataka:

```
BEGIN WORK;
```

```
UPDATE racun SET stanje = stanje - iznos WHERE brRac = brRac1;
```

```
UPDATE racun SET stanje = stanje + iznos WHERE brRac = brRac2;
```

```
COMMIT WORK;
```

- ako se nakon obavljanja naredbe COMMIT dogodi kvar medija (*disk failure*) na kojem je pohranjena baza podataka, efekti potvrđene transakcije, koju korisnik smatra obavljenom, mogli bi biti izgubljeni
- **Izdržljivost (Durability)** - izmjene u bazi podataka koje su rezultat potvrđenih transakcija ne smiju biti izgubljene neovisno o vrsti kvara sustava koji bi se mogao dogoditi nakon što je transakcija potvrđena

# Osiguravanje ACID svojstava transakcije

---

- svojstvo konzistentnosti transakcije (iz skupa ACID svojstava) odnosi se samo na korektnost transakcije
  - pojam svojstva transakcije *konzistentnost* ne treba brkati s pojmom konzistentnosti baze podataka: konzistentnost baze podataka može (osim izvršavanjem transakcija koje ne zadovoljavaju svojstvo konzistentnosti), također biti narušena izvršavanjem transakcijama koje ne zadovoljavaju svojstva atomarnosti, izolacije ili izdržljivosti
- svojstva atomarnosti i izdržljivosti osigurava poseban dio SUBP-a: sustav za obnovu
- svojstvo izolacije osigurava poseban dio SUBP-a: sustav za kontrolu istodobnog pristupa
- bez tih mehanizama koji su ugrađeni u SUBP, programski kôd svake pojedinačne transakcije bi bio izuzetno složen jer bi se u svakoj transakciji zasebno morala implementirati svojstva atomarnosti, izdržljivosti i izolacije

# Dodatak uz svojstvo konzistentnosti: odgođena provjera ugrađenih integritetskih ograničenja

---

- baza podataka može biti u nekonzistentnom stanju tijekom obavljanja transakcije
  - što se za to vrijeme dešava s ugrađenim integritetskim ograničenjima?
- provjera integritetskih ograničenja može se odgoditi do kraja transakcije SQL naredbama:

```
SET CONSTRAINTS constrName { IMMEDIATE | DEFERRED }
SET CONSTRAINTS ALL { IMMEDIATE | DEFERRED }
```

- **IMMEDIATE**: integritetska ograničenja provjeravaju se nakon svake izvedene DML naredbe (ne za vrijeme obavljanja DML naredbe!). Ako konačni rezultat DML naredbe narušava integritetska ograničenja, efekti DML naredbe se automatski poništavaju (*statement-level rollback*)
- **DEFERRED**: integritetska ograničenja se provjeravaju pri izvršavanju naredbe **COMMIT**
  - ako se utvrdi da je neko integritetsko ograničenje narušeno, transakcija se poništava

# Dodatak uz svojstvo konzistentnosti: odgođena provjera ugrađenih integritetskih ograničenja

## Primjer:

Svaki nastavnik predaje u točno jednoj grupi. Zamijeniti grupe za nastavnike 107 i 102

| nastava | sifNast | sifGrupa |
|---------|---------|----------|
|         | 107     | 1.01     |
|         | 102     | 1.02     |
|         | 105     | 1.03     |

```
CREATE TABLE nastava ( IBM Informix
    sifNast      INTEGER
    , sifGrupa   CHAR(4)
    , PRIMARY KEY (sifNast)
        CONSTRAINT pkNastava
    , UNIQUE (sifGrupa)
        CONSTRAINT unqNastava
);
```

```
CREATE TABLE nastava ( Oracle
    sifNast      INTEGER
    , sifGrupa   CHAR(4)
    , CONSTRAINT pkNastava
        PRIMARY KEY (sifNast)
    , CONSTRAINT unqNastava
        UNIQUE (sifGrupa) DEFERRABLE
);
```

```
BEGIN WORK;
UPDATE nastava SET sifGrupa = '1.02'
    WHERE sifNast = 107; → POGREŠKA
UPDATE nastava SET sifGrupa = '1.01'
    WHERE sifNast = 102;
COMMIT WORK;
```

```
BEGIN WORK;
SET CONSTRAINTS unqNastava DEFERRED;
UPDATE nastava SET sifGrupa = '1.02'
    WHERE sifNast = 107;
UPDATE nastava SET sifGrupa = '1.01'
    WHERE sifNast = 102;
COMMIT WORK; → O.K.
```

Što će se dogoditi ako transakcija nije korektna?

```
BEGIN WORK;
SET CONSTRAINTS unqNastava DEFERRED;
UPDATE nastava SET sifGrupa = '1.02'
    WHERE sifNast = 107;
COMMIT WORK; → POGREŠKA i ROLLBACK
```

# Prekid i poništavanje (*abort* i *rollback*) transakcije

---

- prekid transakcije iniciran od strane programa ili korisnika
  - u kontroliranim uvjetima utvrđeno je da transakcija ne može biti obavljena. Npr. u PL/SQL proceduri je utvrđeno da bi dalnjim obavljanjem transakcije došlo do narušavanja integritetskih ograničenja, procedurom se obavlja naredba ROLLBACK\*)
  - korisnik je odlučio putem SQL sučelja izazvati prekid transakcije
  - abnormalni prekid programa (korisničke sjednice): SUBP nastavlja s radom ali poništava transakciju koja se izvršavala u toj korisničkoj sjednici
- prekid transakcije iniciran od strane sustava za upravljanje bazama podataka
  - abnormalni prekid rada SUBP-a (npr. nestanak napajanja). Sustav je prestao raditi, baza podataka je u nekonzistentnom stanju, ali će se efekti eventualno nezavršenih transakcija poništiti tijekom obnove, koja se obavlja prije nego sustav počne zaprimati nove transakcije.
  - sustav prekida i poništava jednu ili više transakcija tijekom razrješavanja potpunog zastoja

\* SQL naredba ROLLBACK [WORK] predstavlja zahtjev za prekidom transakcije (*abort*) nakon kojeg SUBP odmah obavlja i poništavanje efekata transakcije (*rollback*).

# Primjer: Oracle PL/SQL

- zadani iznos sredstava prebacuje s prvog zadanog na drugi zadani račun (stanje prvog računa umanjiti, a drugog računa uvećati), uz integritetsko ograničenje: stanje niti jednog računa ne smije biti manje od nule

**PL/SQL varijable:**

|                |                               |
|----------------|-------------------------------|
| <b>brRac1</b>  | ← broj prvog računa           |
| <b>brRac2</b>  | ← broj drugog računa          |
| <b>iznos</b>   | ← iznos kojeg treba prebaciti |
| <b>stanje1</b> | ← novo stanje na prvom računu |

```
-- transakcija započinje prvom izvršnom SQL naredbom
UPDATE racun SET stanje = stanje - iznos WHERE brRac = brRac1;
UPDATE racun SET stanje = stanje + iznos WHERE brRac = brRac2;
SELECT stanje INTO stanje1 FROM racun WHERE brRac = brRac1;
IF stanje1 < 0.00 THEN
    ROLLBACK;
ELSE
    COMMIT;
END IF;
```

# Primjer: JAVA + JDBC API

Java varijable:

**brRac1** ← broj prvog računa  
**brRac2** ← broj drugog računa  
**iznos** ← iznos kojeg treba prebaciti  
**stanje1** ← novo stanje na prvom računu  
**conn** ← *Connection* objekt

```
Statement stmt = conn.createStatement();
conn.setAutoCommit(false);
stmt.executeUpdate("UPDATE racun SET stanje = stanje - " + iznos +
                    " WHERE brRac = " + brRac1 );
stmt.executeUpdate("UPDATE racun SET stanje = stanje + " + iznos +
                    " WHERE brRac = " + brRac2 );
ResultSet rs = stmt.executeQuery("SELECT stanje FROM racun " +
                                " WHERE brRac = " + brRac1);
rs.next();
BigDecimal stanje1 = rs.getBigDecimal(1);
if (stanje1.doubleValue() >= 0.0) {
    conn.commit();
} else {
    conn.rollback();
}
```

# Prekid i poništavanje na razini SQL naredbe

---

- neovisno o tome jesu li granice transakcije eksplicitno ili implicitno zadane, svaka pojedinačna SQL naredba se obavlja u cijelosti ili se njezini efekti poništavaju u cijelosti (ANSI Standard: *SQL-statement atomicity*)
- ako se SQL naredba ne može uspješno obaviti u cijelosti, obavlja se poništavanje na razini SQL naredbe (*statement-level rollback*)

# Prekid i poništavanje na razini SQL naredbe

```
CREATE TABLE ispit (
    mbrSt  INTEGER
,  ocj    INTEGER
    CHECK (ocj BETWEEN 1 AND 5)
);
```

| ispit | mbrSt | ocj |
|-------|-------|-----|
|       | 101   | 3   |
|       | 102   | 4   |
|       | 103   | 5   |

```
UPDATE ispit SET ocj = ocj + 1;
```

neke ocjene su možda bile promijenjene tijekom izvršavanja naredbe, ali u trenutku kada je naredba prekinuta zbog pogreške (narušavanja integritetskog ograničenja), svi njezini efekti su poništeni

```
BEGIN WORK;
...
ostale SQL naredbe
UPDATE ispit SET ocj = ocj + 1;
...
ostale SQL naredbe
COMMIT ili ROLLBACK WORK;
```

već u ovom trenutku poništeni su efekti naredbe UPDATE - sve su ocjene vraćene na početne vrijednosti. Efekti ostalih SQL naredbi iz ove transakcije ovog trenutka još nisu poništeni. Efekti ostalih SQL naredbi mogu se poništiti obavljanjem naredbe ROLLBACK

# Primjer transakcije (IBM Informix)

korisnik *admin* obavlja:

```
CREATE DATABASE testTran
    IN dbspace1 WITH LOG;
CREATE TABLE racun (
    sifRacun      INTEGER
, stanje        DECIMAL(9,2)
, dopustMinus  DECIMAL(9,2)
, PRIMARY KEY (sifRacun)
    CONSTRAINT pkRacun
, CHECK (stanje + dopustMinus >= 0)
    CONSTRAINT chkRacunStanje
);

```

```
CREATE TABLE promet (
    sifPromet     SERIAL
, sifRacun      INTEGER
, iznos         DECIMAL(9,2)
, PRIMARY KEY (sifPromet)
    CONSTRAINT pkPromet
, FOREIGN KEY (sifRacun)
    REFERENCES racun (sifRacun)
    CONSTRAINT fkPrometRacun
);

```

- narušavanje integritetskih ograničenja izazvat će pogrešku koja je opisna s tri parametra (SQL error, ISAM error, Error data):

(-268, -100, admin.pkracun)

(-530, 0, admin.chkracunstanje)

(-268, -100, admin.pkpromet)

(-691, -111, admin.fkprometracun)

# Primjer transakcije (IBM Informix) - nastavak

---

- napisati pohranjenu proceduru **unosPromet**
- ulazni argumenti: šifra računa i iznos. Unijeti n-torku u relaciju promet (serijski broj, zadanu šifru računa i zadani iznos). Promijeniti stanje zadanog računa tako da postojećem stanju pribroji zadani iznos). Ove dvije operacije se moraju obaviti u okviru transakcije
- ako stanje na računu padne ispod dopuštenog (ograničenje chkRacunStanje) tada u pozivajući program treba dojaviti pogrešku -746, 0, 'Nedopušten minus'
- u slučaju bilo koje druge pogreške, u pozivajući program dojaviti originalnu pogrešku
- procedura ne vraća ništa: smatra se da je uspješno obavljena ako ne vrati pogrešku
- procedura samostalno upravlja granicama transakcije, tj. započinje i potvrđuje/poništava transakciju: započeta transakcija po završetku procedure mora biti u stanju završena (ili potvrđena ili poništena)

# Primjer transakcije (IBM Informix) - nastavak

- primjer izvršavanja iz interaktivnog alata (npr. SQL Editor)

| racun | sifRacun | stanje | dopustMinus |
|-------|----------|--------|-------------|
|       | 1        | 100.00 | 200.00      |
|       | 2        | 500.00 | 500.00      |

| promet | sifPromet | sifRacun | iznos |
|--------|-----------|----------|-------|
|        |           |          |       |

```
EXECUTE PROCEDURE unosPromet(2, -1000.00);
```



| racun | sifRacun | stanje  | dopustMinus |
|-------|----------|---------|-------------|
|       | 1        | 100.00  | 200.00      |
|       | 2        | -500.00 | 500.00      |

| promet | sifPromet | sifRacun | iznos    |
|--------|-----------|----------|----------|
|        | 1         | 2        | -1000.00 |

```
EXECUTE PROCEDURE unosPromet(1, -500.00);
```



**-746, 0, Nedopušten minus**

procedura je "uhvatila" pogrešku (-530, 0, admin.chkracunstanje) i umjesto nje proslijedila novu pogrešku (-746, 0, Nedopušten minus)

```
EXECUTE PROCEDURE unosPromet(3, 500.00);
```



**-691, -111, admin.fkprometracun**

procedura nije "uhvatila" pogrešku pa je originalna pogreška proslijedena pozivajućem programu (u ovom slučaju SQL editoru)

# Primjer transakcije (IBM Informix) - nastavak

```
CREATE PROCEDURE unosPromet(pSifRacun LIKE promet.sifRacun
                            , pIznos LIKE promet.iznos)
  DEFINE sqle, isame INTEGER;
  DEFINE errdata CHAR(80);
  ON EXCEPTION SET sqle, isame, errdata
    ROLLBACK WORK;
    IF sqle = -530 AND errdata LIKE '%chkracunstanje%' THEN
      RAISE EXCEPTION -746, 0, 'Nedopušten minus';
    ELSE
      RAISE EXCEPTION sqle, isame, errdata;
    END IF
  END EXCEPTION;                                što ako izazove pogrešku
  BEGIN WORK;                                    (-691, -111, admin.fkprometracun)
  INSERT INTO promet VALUES (0, pSifRacun, pIznos);
  UPDATE racun SET stanje = stanje + pIznos
    WHERE sifRacun = pSifRacun;                 što ako izazove pogrešku
  COMMIT WORK;                                   (-530, 0, admin.chkracunstanje)
END PROCEDURE;
```

# Primjer transakcije (IBM Informix) - nastavak

- primjer poziva procedure **unosPromet** iz java klijentske aplikacije

```
...
Statement stmt = conn.createStatement();
conn.setAutoCommit(true); // granicama transakcije ne upravlja klijent
try {
    stmt.executeUpdate("EXECUTE PROCEDURE unosPromet(sifRac, iznos)");
} catch (SQLException exc) {
    System.out.println(exception.getErrorCode() + " " + exception.getMessage());
}
```

- poziv procedure **unosPromet** iz java klijentske aplikacije uz ***sifRac=1, iznos=-500***  
→ **-746 Nedopusťen minus**
- poziv procedure **unosPromet** iz java klijentske aplikacije uz ***sifRac=3, iznos=500***  
→ **-691 admin.fkprometracun**

# Primjer transakcije (IBM Informix) - nastavak

- transakcija realizirana pomoću java + JDBC API

```
void unosPromet(Connection connection
                 , Integer sifRacun
                 , BigDecimal iznos) throws SQLException {
    Statement stmt = connection.createStatement();

    connection.setAutoCommit(false); // granicama transakcije upravlja klijent
    try {
        stmt.executeUpdate("INSERT INTO promet VALUES (0, " + sifRacun + ", " + iznos + ")");
        stmt.executeUpdate("UPDATE racun SET stanje = stanje + " + iznos +
                           " WHERE sifRacun = " + sifRacun);
    }
    catch (SQLException exception) {
        connection.rollback();
        if (exception.getErrorCode() == -530 &&
            exception.getMessage().indexOf("chkracunstanje") != -1) {
            throw new SQLException("Nedopušten minus", exception.getSQLState(), -746);
        }
        else {
            throw exception;
        }
    }
    connection.commit();
}
```

# Primjer transakcije (IBM Informix) - nastavak

- primjeri poziva metode unosPromet

```
// podrazumijeva se da je objektom connection uspostavljena sjednica  
...  
try {  
    unosPromet(connection, sifRacun, iznos);  
}  
catch (SQLException exception) {  
    System.out.println(exception.getErrorCode() + " " + exception.getMessage());  
}
```

- ako su vrijednosti za sifRacun i iznos: 1, -500.00



**-746 Nedopusťen minus**

- ako su vrijednosti za sifRacun i iznos: 3, 500.00



**-691 admin.fkprometracun**

# Subtransakcije

## Ugnježđivanje transakcija nije dopušteno:

- pokušaj parcijalnog poništavanja transakcije na pogrešan način

```
BEGIN WORK;  
SQL_1;  
SQL_2;  
BEGIN WORK;  
SQL_3;  
SQL_4;  
IF (uvjet_1) THEN  
    ROLLBACK WORK;  
ELSE  
    COMMIT WORK;  
END IF;  
SQL_5;  
COMMIT WORK;
```

- ispravan način za poništavanje samo određenih dijelova transakcije: *savepoints*

# Savepoints

---

- naredba ROLLBACK poništava sve promjene u bazi podataka koje su transakcijom obavljene nakon početka transakcije
- u transakciji T je moguće označiti jednu ili više "kontrolnih" točki (*savepoints*)
- unutar transakcije T je moguće poništiti promjene koje su obavljene nakon bilo koje od označenih "kontrolnih" točaka (*savepoint*), a da se time ne prekine transakcija niti ponište efekti ostalih obavljenih DML naredbi
- SQL naredbe
  - `SAVEPOINT savepoint_id;`
  - `ROLLBACK TO SAVEPOINT savepoint_id;`

# Savepoints

Primjer:

```
BEGIN WORK;  
SQL_1;  
SQL_2;  
BEGIN WORK;  
SQL_3;  
SQL_4;  
IF (uvjet_1) THEN  
    ROLLBACK WORK;  
ELSE  
    COMMIT WORK;  
END IF;  
SQL_5;  
COMMIT WORK;
```



```
BEGIN WORK;  
SQL_1;  
SQL_2;  
SAVEPOINT dioTrans;  
SQL_3;  
SQL_4;  
IF (uvjet_1) THEN  
    ROLLBACK TO SAVEPOINT dioTrans;  
ELSE  
    -- ništa  
END IF;  
SQL_5;  
COMMIT WORK;
```

Napomena: ROLLBACK TO SAVEPOINT nije niti po čemu slična naredbi za kontrolu toka programa *goto* u nekim programskim jezicima!

# Savepoints

Primjer:

```
BEGIN WORK;
SQL_1;
SAVEPOINT t_1;
SQL_2;
SAVEPOINT t_2;
SQL_3;
IF (uvjet_1) THEN
    ROLLBACK TO SAVEPOINT t_2;
    SQL_4;
END IF;
SQL_5;
IF (uvjet_2) THEN
    ROLLBACK TO SAVEPOINT t_1;
    SQL_6;
END IF;
SQL_7;
IF (uvjet_3) THEN
    COMMIT WORK;
ELSE
    ROLLBACK WORK;
END IF
```

- obavlja se naredba SQL\_1
- označava se *savepoint t\_1*
- obavlja se SQL\_2
- označava se *savepoint t\_2*
- obavlja se SQL\_3
- ako vrijedi uvjet\_1 poništavaju se promjene koje je obavila SQL\_3 i obavlja se SQL\_4
- obavlja se SQL\_5
- ako vrijedi uvjet\_2 poništavaju se promjene koje su obavile sve SQL naredbe obavljene nakon t\_1 i obavlja se SQL\_6
- obavlja se SQL\_7
- ako vrijedi uvjet\_3 potvrđuju se sve obavljene promjene (koje u međuvremenu nisu poništene)
- ako ne vrijedi uvjet\_3, poništavaju se sve promjene (koje u međuvremenu već nisu poništene)

# Klasifikacija transakcija

---

- ***on-line (short-life) transakcije***: obavljaju se u relativno kratkom vremenu (tipično ne više od sekunde) i pristupaju malom broju elemenata baze podataka
- ***batch (long-life) transakcije***: trajanje takvih transakcija je izraženo u minutama ili satima, praćeno pristupanjem velikom broju elemenata baze podataka
- ***konverzacijske ili interaktivne transakcije***: za vrijeme obavljanja transakcije postoji interakcija s korisnikom

# **Model transakcije**

# Pojmovi i oznake

---

- transakcije se označavaju oznakama  $T_1, T_2, \dots, T_i, T_j, \dots$
- baza podataka se promatra kao statički skup (u smislu nepromjenjivosti kardinalnog broja skupa) imenovanih elemenata ili objekata
- kao element baze podataka može se promatrati dio n-torke (*field*), n-torka (*record*), fizička stranica (*page*, *block*), relacija, baza podataka. Veličina promatranog elementa određena je granulacijom (*granularity*). U većem dijelu razmatranja koje slijedi granulacija je irelevantna
- kao oznake elemenata baze podataka koriste se mala slova x, y, z, itd.
- sve operacije nad elementima baze podataka odnose se na dvije osnovne vrste operacija: operaciju čitanja (*read*) i operaciju izmjene, odnosno pisanja (*write*). Navodi se vrsta operacije i naziv objekta. Vrijednost elementa je irelevantna
- operacije koje transakcija s oznakom  $T_i$  obavlja nad elementom x baze podataka označavat će se općenito s  $o_i[x]$ , a konkretno za operacije čitanja i pisanja, koristit će se oznake  $r_i[x]$  i  $w_i[x]$
- transakcijske operacije koje obavlja transakcija s oznakom  $T_i$  označavaju se s  $a_i$  (*abort*) i  $c_i$  (*commit*)

# Parcijalni poredak

- Parcijalni poredak\*
  - Parcijalni poredak (*partial order*)  $L$  je uređeni par  $(\Sigma, <)$ . Skup  $\Sigma$  je domena parcijalnog poretka; relacija  $<$  je relacija poretka. Relacija poretka je irefleksivna, asimetrična i tranzitivna relacija na skupu  $\Sigma$ . Ako za elemente  $a, b \in \Sigma$  vrijedi  $a < b$ , odnosno uređeni par  $(a, b) \in <$ , tada se kaže da  $a$  prethodi  $b$ , odnosno da  $b$  slijedi nakon  $a$ . Ako ne vrijedi  $a < b$  niti vrijedi  $b < a$ , tada su  $a$  i  $b$  neusporedivi (neporedani).

\*formalno ispravnije - *striktni* parcijalni poredak

- Relacija  $\rho$  nad skupom  $P$  je:
  - irefleksivna ako  $(\forall a \in P) \neg (a \rho a)$
  - asimetrična ako  $(\forall a, b \in P) (a \rho b \Rightarrow \neg (b \rho a))$
  - tranzitivna ako  $(\forall a, b, c \in P) (a \rho b \wedge b \rho c \Rightarrow a \rho c)$

## Primjer:

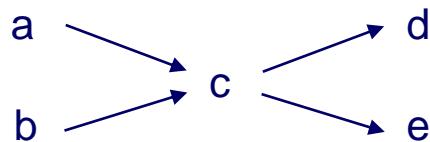
- Domena  $\Sigma = \{ a, b, c, d, e \}$
- Relacija poretka  $< = \{ (a, c), (a, d), (a, e), (b, c), (b, d), (b, e), (c, d), (c, e) \}$
- Parcijalni poredak  $L = (\Sigma, <)$

# Parcijalni poredak i usmjereni graf

- Usmjereni graf i usmjereni aciklički graf
  - Usmjereni graf  $G = (N, E)$  sastoji se od skupa elemenata  $N$  koji se nazivaju čvorovi, te skupa  $E$  uređenih parova koji se nazivaju lúkovi. Lük između elemenata  $a$  i  $b$  označavat će se s  $(a, b)$ . Usmjereni aciklički graf je usmjereni graf koji ne sadrži petlje.
    - Parcijalni poredak  $L = (\Sigma, <)$  se može prikazati kao usmjereni aciklički graf  $G = (N, E)$  pri čemu je  $N = \Sigma$ , dok je  $E$  skup lúkova  $(a, b)$ ,  $a, b \in \Sigma$ , takvih da je  $a < b$ .

## Primjer:

- usmjereni aciklički graf za prikaz parcijalnog poretku iz prethodnog primjera:



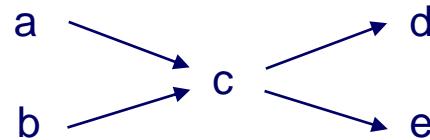
- lúkove implicirane svojstvom tranzitivnosti nije potrebno crtati

# Topološki poredak usmjerenog acikličkog grafa

- Topološki poredak
  - Topološki poredak usmjerenog acikličkog grafa  $G = (N, E)$  je niz sastavljen od svih elemenata iz  $N$  poredanih tako da se element  $a$  nalazi u nizu ispred elementa  $b$  samo ako u  $G$  ne postoji put od čvora  $b$  do čvora  $a$ .

## Primjer:

- prikaz mogućih topoloških poredaka za usmjereni aciklički graf iz prethodnog primjera:
  - a, b, c, d, e
  - b, a, c, d, e
  - a, b, c, e, d
  - b, a, c, e, d
- jednostavan postupak: iz grafa uzastopno uklanjamo čvorove koji nemaju prethodnike među preostalim čvorovima u grafu. Svaki iz grafa uklonjeni čvor dodajemo na kraj niza koji predstavlja topološki poredak



# Model transakcije

---

- Transakcija se modelira kao (striktni) parcijalni poredak
- Neka je  $\Sigma_i$  skup operacija transakcije  $T_i$ ,  $<_i$  je relacija koja određuje međusobni poredak operacija iz  $\Sigma_i$ . Transakcija  $T_i = (\Sigma_i, <_i)$  pri čemu vrijedi:
  1.  $\Sigma_i \subseteq \{ r_i[x], w_i[x] \mid x \text{ je objekt u bazi podataka} \} \cup \{ a_i, c_i \}$
  2.  $a_i \in \Sigma_i$  ako i samo ako  $c_i \notin \Sigma_i$
  3. ako je operacija  $t \in \{ a_i, c_i \}$ , tada za sve operacije  $p$ ,  $p \in \Sigma_i \wedge p \neq t$  vrijedi  $p <_i t$
  4. za svake dvije operacije  $r_i[x], w_i[x] \in \Sigma_i$ , vrijedi ili  $r_i[x] <_i w_i[x]$  ili  $w_i[x] <_i r_i[x]$

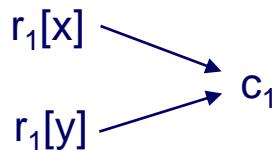
## Objašnjenje:

- 1: definira koje su operacije transakcije dopuštene
- 2: u transakciji se nalazi jedna i samo jedna od operacija *abort* ili *commit*
- 3: operacija *abort* ili *commit* mora biti posljednja operacija svake transakcije
- 4: ako transakcija obavlja operaciju čitanja i operaciju pisanja nad istim elementom  $x$ , redoslijed obavljanja tih operacija utječe na rezultat operacije čitanja. Svojstvom se garantira da je za operaciju pisanja elementa  $x$  određen poredak u odnosu na operaciju čitanja istog elementa  $x$

# Parcijalni poredak, graf transakcije

Primjer:

- $T_1 = (\Sigma_1, <_1)$ 
  - $\Sigma_1 = \{ r_1[x], r_1[y], c_1 \}$
  - $<_1 = \{ (r_1[x], c_1), (r_1[y], c_1) \}$
- transakcija  $T_1$ , odnosno parcijalni poredak  $(\Sigma_1, <_1)$  također se može prikazati sljedećim usmjerenim acikličkim grafom:

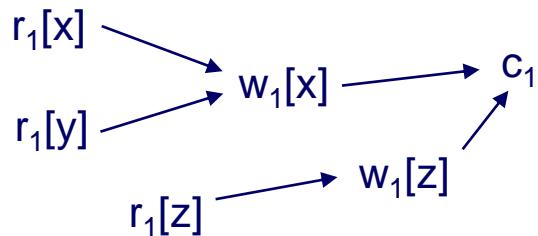


- operacije  $r_1[x]$  i  $r_1[y]$  su neusporedivi elementi transakcije - nije utvrđen njihov međusobni poredak
- operacija  $c_1$  se mora dogoditi nakon  $r_1[x]$  i  $r_1[y]$

# Parcijalni poredak, graf transakcije

Primjer:

- $T_1 = (\Sigma_1, <_1)$ 
  - $\Sigma_1 = \{ r_1[x], r_1[y], w_1[x], r_1[z], w_1[z], c_1 \}$
  - $<_1 = \{ (r_1[x], w_1[x]), (r_1[y], w_1[x]), (r_1[z], w_1[z]), (r_1[x], c_1), (r_1[y], c_1), (w_1[x], c_1), (r_1[z], c_1), (w_1[z], c_1) \}$
- transakcija  $T_1$ , odnosno parcijalni poredak  $(\Sigma_1, <_1)$  također se može prikazati sljedećim usmjerenim acikličkim grafom:



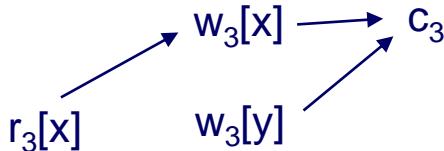
- $w_1[x]$  se mora dogoditi nakon  $r_1[x]$  i  $r_1[y]$
- nije određeno kojim redoslijedom se moraju obaviti npr.  $r_1[x]$  i  $r_1[y]$ .
- iako u relaciji poretku postoji par  $(r_1[x], c_1)$ , u grafu taj lûk nije potrebno crtati jer iz  $r_1[x] < w_1[x]$  i  $w_1[x] < c_1$ , temeljem svojstva tranzitivnosti proizlazi  $r_1[x] < c_1$ .

# Parcijalni poredak, graf transakcije

## Dodatna razmatranja:

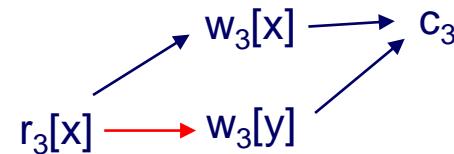
- osim lúkova koje graf **mora** sadržavati prema definiciji modela transakcije (sve operacije čitanja i pisanja nad istim elementom moraju biti poredane, sve operacije čitanja i pisanja prethode operacijama *commit/abort*), u graf se moraju ucrtati i lúkovi koji proizlaze iz semantike transakcije

Primjer:



vrijednost koju zapisuje operacija  $w_3[y]$  ne ovisi o rezultatu operacije  $r_3[x]$

```
read(x, p);  
display(p);  
write(x, 35);  
write(y, 27);  
commit;
```



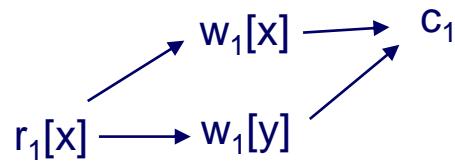
ako transakcija novu vrijednost za  $y$  određuje na temelju pročitane vrijednosti elementa  $x$ , dodaje se lúk između operacija  $r_3[x]$  i  $w_3[y]$

```
read(x, p);  
write(x, 35);  
p ← p + 10;  
write(y, p);  
commit;
```

# Topološki poredak

Primjer:

- Transakcija  $T_1$  je prikazana sljedećim usmjerenim acikličkim grafom



- Topološkim poretkom čvorova grafa može se dobiti prikaz transakcije u obliku niza operacija
  - $r_1[x], w_1[y], w_1[x], c_1$   
ili
  - $r_1[x], w_1[x], w_1[y], c_1$

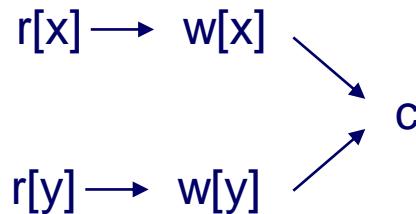
# Graf transakcije

Primjer: zadani iznos sredstava prebaciti s prvog zadanog na drugi zadani račun (stanje prvog računa umanjiti, a drugog računa uvećati).

```
x   y  
SELECT stanje INTO p FROM racun WHERE brRac = zadBrRac1;  
SELECT stanje INTO q FROM racun WHERE brRac = zadBrRac2;  
LET r = p - zadIznos;  
LET s = q + zadIznos;  
UPDATE racun SET stanje = r WHERE brRac = zadBrRac1;  
UPDATE racun SET stanje = s WHERE brRac = zadBrRac2;  
COMMIT;
```

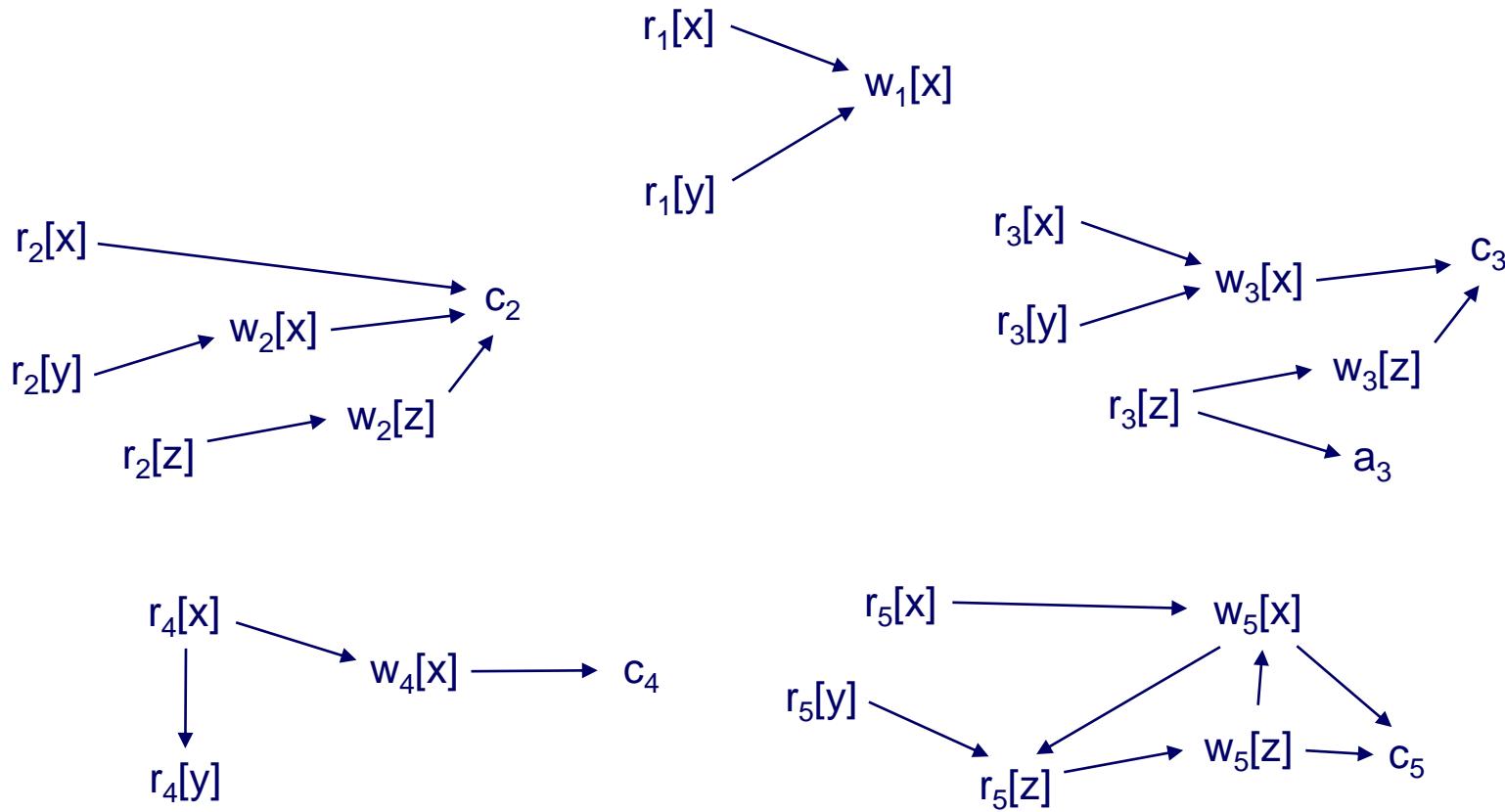
```
start;  
read(x, p);  
read(y, q);  
-- izracunaj r, s  
write(x, r);  
write(y, s);  
commit;
```

## Model transakcije



# Zadatak za vježbu

- za svaki graf utvrditi zašto se ne radi o ispravnom grafu transakcije



- rješenje zadatka se može provjeriti na sljedećoj stranici

# Rješenje

---

1. u grafu ne postoji niti operacija  $a_1$  niti operacija  $c_1$
2. operacije  $r_2[x]$  i  $w_2[x]$  su neusporedive (neporedane), odnosno nije utvrđen poredak između  $r_2[x]$  i  $w_2[x]$
3. graf istovremeno sadrži operacije  $a_3$  i  $c_3$ . Dodatno,  $a_3$  nije poredana u odnosu na sve ostale operacije transakcije.
4. operacija  $c_4$  nije poredana u odnosu na sve ostale operacije transakcije
5. graf nije acikličan (postoji ciklus  $w_5[x] \rightarrow r_5[z] \rightarrow w_5[z] \rightarrow w_5[x]$ ), odnosno relacija porekta nije asimetrična (npr. vrijedi  $w_5[x] < w_5[z]$  i  $w_5[z] < w_5[x]$ ). Uočiti da zbog toga na temelju ovog grafa nije moguće konstruirati ispravan topološki poredak operacija

# Literatura:

---

- P. Bernstein, V. Hadzilacos, N. Goodman. Concurrency control and recovery in database systems. Addison-Wesley Publishing Company. Reading, Massachusetts. 1987.
- H. Garcia-Molina, J. Ullman, J. Widom. Database system implementation. Prentice-Hall International, Inc. Upper Saddle River, New Jersey. 2000.
- R. Elmasri, S. Navathe. Fundamentals of database systems, 3rd ed. Addison-Wesley Publishing Company. Reading, Massachusetts. 2000.
- A. Silberschatz, H. Korth, S. Sudarshan. Database system concepts, 6th ed. McGraw-Hill. 2010.

# **Sustavi baza podataka**

Predavanja

**6. Sustav za obnovu  
(1. dio)**  
travanj 2023.

# Pogreške u SUBP-u

---

- različiti uzroci pogrešaka:
  - pogreške računalne opreme (softverske, hardverske)
  - prekid napajanja
  - požar, potres, eksplozija, sabotaža
  - itd.
- svaki od tih događaja može dovesti do gubitka podataka
- takve događaje nije moguće spriječiti korištenjem "besprijeckornih" komponenti i "besprijeckornim" dizajnom sustava → previsoka cijena

## Rješenje:

- korištenje sustava za obnovu (*recovery system*)

# Zadatak sustava za obnovu

- u slučaju pogreške dovesti SUBP u stanje ekvivalentno zadnjem konzistentnom stanju sustava prije pogreške. Pri tome je važno:
  1. ostvariti čim veću sposobnost oporavka (*resilience*) SUBP-a
    - sustav treba biti u stanju oporaviti se nakon različitih vrsta pogrešaka
  2. ostvariti čim veću raspoloživost (*availability*) SUBP-a minimiziranjem vremena potrebnog za oporavak sustava nakon pogreške
    - omjer vremena tijekom kojeg je sustav sposoban obavljati predviđenu funkciju i ukupnog promatranog vremena koje obuhvaća i vrijeme potrebno za oporavak

$$Av = MTTF / (MTTF + MTTR)$$

Mean Time To Failure

Mean Time To Repair

- MTTF ovisi o pouzdanosti (*reliability*)

| sustav u kvaru | raspoloživost |
|----------------|---------------|
| 1 sat/dan      | 95.8 %        |
| 1 sat/mjesec   | 99.86 %       |
| 1 sat/godinu   | 99.9886 %     |

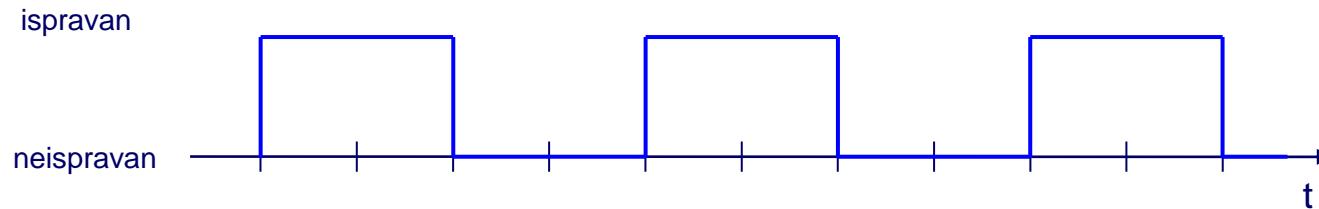
Pouzdanost (*reliability*): svojstvo sustava koje se odnosi na vjerojatnost da će sustav funkcionirati ispravno tijekom određenog vremenskog intervala.

Sustav za obnovu ne može utjecati na pouzdanost ("ne može spriječiti kvar").

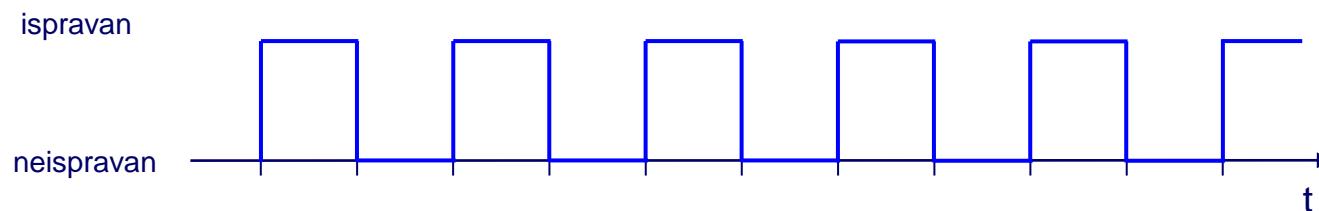
# Raspoloživost, pouzdanost, srednje vrijeme oporavka

- usporediti raspoloživost, srednje vrijeme potrebno za oporavak i pouzdanost sljedećih sustava

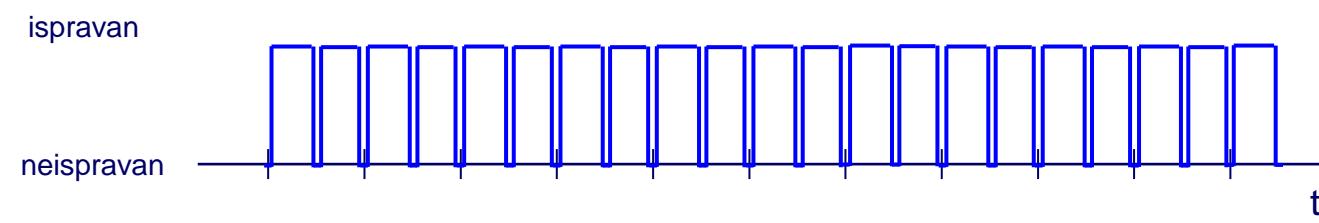
1.



2.

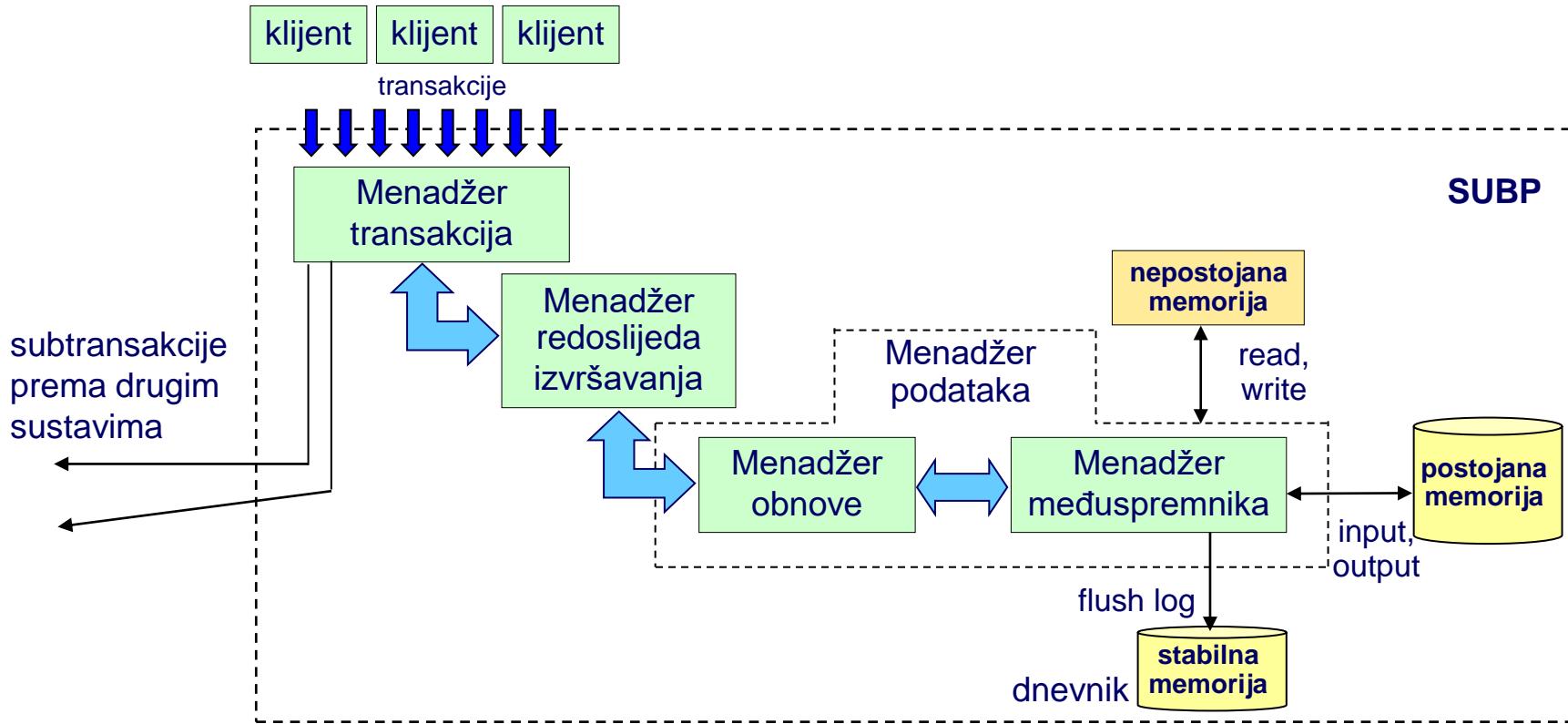


3.



# Model SUBP-a

- apstraktni model sustava za upravljanje bazama podataka [Bernstein]



- na slici su prikazani samo oni dijelovi sustava za upravljanje bazama podataka koji upravljaju obnovom i istodobnim pristupom (npr. nedostaje *Query optimizer*, *Access Methods Manager*, itd.)

# Model SUBP-a

---

- Menadžer transakcija (*transaction manager*) upravlja početkom i završetkom transakcije, zaprimljene operacije transakcija upućuje prema menadžeru redoslijeda izvršavanja (*scheduler*). U distribuiranim sustavima ima dodatnu zadaću usmjeravanja operacija transakcija prema drugim SUBP
- Menadžer redoslijeda izvršavanja (*scheduler*) upravlja redoslijedom obavljanja operacija transakcija: obavljanjem, odbijanjem ili odgađanjem obavljanja pojedinih operacija. Upravlja istodobnim (ili "paralelnim") pristupom
- Menadžer obnove (*recovery manager*) osigurava pouzdano potvrđivanje ili poništavanje transakcija. Podsustav je odgovoran za to da baza podataka sadrži sve efekte potvrđenih transakcija i niti jedan efekt poništenih transakcija. Između ostalog, mora biti neosjetljiv na gubitak sadržaja nepostojane (*volatile*) memorije ili na gubitak sadržaja ili kvar postojane (*non-volatile*) memorije.
- Menadžer međuspremnika (*buffer manager*) je podsustav zadužen za prebacivanje blokova memorije između međuspremnika i postojane memorije.
- Menadžer obnove i menadžer međuspremnika čine veću cjelinu koja se uobičajeno naziva menadžer podataka (*data manager*)

# Uspostavljanje sustava za obnovu

- identificirati pogreške koje se u pojedinim komponentama sustava mogu dogoditi, te utjecaj pojedine vrste pogreške na sposobnost oporavka i raspoloživost sustava
- definirati postupke obnove (*recovery algorithms*) koji će osigurati otpornost sustava na pogreške, odnosno omogućiti očuvanje konzistentnosti baze podataka bez obzira na pogreške

## Postupci obnove

1. postupci koji se provode tijekom redovitog funkciranja sustava i osiguravaju prikupljanje dovoljno podataka za obnovu baze podataka nakon pogreške
2. postupci pomoću kojih se u slučaju pogreške sustav na pouzdan način vraća u konzistentno stanje

## Temeljna ideja implementacije

Korištenje redundancije - svaki podatak koji je izgubljen zbog pogreške u jednom dijelu sustava, mora se moći rekonstruirati iz podataka redundantno pohranjenih u nekom drugom dijelu sustava

# Vrste pogrešaka

---

- pogreške koje utječu na funkcioniranje SUBP-a klasificiraju se na sljedeći način:
  - pogreške transakcija (*transaction failures*)
  - pogreške sustava (*system failures, crashes*)
  - pogreške medija (*media failures*)

# Pogreške transakcije

---

- nepredviđena pogreška koja je uzrokovana ulaskom transakcije u stanje koje onemogućuje daljnje ispravno obavljanje operacija transakcije. SUBP **implicitno** prekida i poništava dotičnu transakciju
  - pogreške koje nastaju zbog interakcije transakcije s drugim transakcijama
    - detekcija stvarnog ili potencijalnog potpunog zastoja
    - narušavanje serijalizabilnog izvršavanja
  - neočekivani prekid korisničke sjednice (npr. prekid klijentske aplikacije)
- u navedenim i sličnim slučajevima SUBP mora prekinuti i poništiti transakciju
  - poništavanje transakcije obavlja sustav za obnovu

# Pogreške transakcije

---

- važne karakteristike pogrešaka transakcija:
  - ne dolazi do gubitka sadržaja postojane niti nepostojane memorije
  - ne dolazi do zastoja u funkcioniranju SUBP-a (transakcije koje nisu sudjelovale u pogrešci nastavljaju se izvršavati bez zastoja)

# Pogreške transakcije

Primjer:

```
begin work
    read(x, p)
    read(y, q)
    p ← p / q
    write(x, q)
    write(y, p)
commit work
```



- Ako u trenutku **t** dođe do prekida korisničke sjednice, dogodit će se **pogreška transakcije**. SUBP će prekinuti i poništiti takvu transakciju.

# Eksplisitno rukovanje pogreškama

---

- pogrešku koja se dogodi pri obavljanju neke operacije transakcije SUBP signalizira procesu koji izvršava transakciju. Time se omogućuje **eksplicitno rukovanje pogreškom**, npr. tako da se pod određenim programskim uvjetima transakcija eksplisitno prekine ili se obavi niz kompenzacijskih operacija i nastavi s izvršavanjem transakcije
- ako transakcija (tj. programski kôd transakcije) eksplisitno rukuje pogreškom, tada se pogreška **ne smatra pogreškom transakcije**, bez obzira obavi li se pri tome eksplisitno prekidanje transakcije ili se transakcija nastavi nakon obavljanja kompenzacijskih operacija
  - iako se ne radi o pogrešci transakcije, za poništavanje transakcije i ovdje će se koristiti mehanizmi sustava za obnovu (koji su to mehanizmi objašnjeno je kasnije)

# Eksplicitno rukovanje pogreškama

Primjer:

```
begin work
    read(x, p)
    read(y, q)
    p ← p / q
    write(x, p)
commit work
on exception (division by zero)
    p ← p / 2
end exception with continue
```

```
begin work
    read(x, p)
    read(y, q)
    p ← p / q
    write(x, p)
commit work
on exception (division by zero)
    rollback work
end exception
```

Ako je vrijednost elementa y jednaka 0, transakcija će obaviti kompenzaciju akciju. **Ne smatra se pogreškom transakcije.**

Ako je vrijednost elementa y jednaka 0, transakcija će se eksplicitno prekinuti te poništiti. **Ne smatra se pogreškom transakcije.**

- iako se ne radi o pogrešci transakcije, za poništavanje transakcije i ovdje se (u primjeru na desnoj strani) koriste mehanizmi sustava za obnovu

# Pogreške sustava

---

- SUBP dio podataka privremeno pohranjuje u nepostojanoj memoriji, u međuspremnicima baze podataka
- pogreška sklopolja, pogreška programske podrške (operacijskog sustava ili programske podrške SUBP-a), nestanak napajanja i slično, uzrokuje
  - trenutačni prekid rada sustava (SUBP procesa ili dretvi izvršavanja)
  - gubitak sadržaja međuspremnika
- prepostavka o prekidu rada sustava u slučaju pogreške (*fail-stop* ili *fail-fast assumption*) je važan mehanizam kojim se osigurava da hardverske ili softverske pogreške izazivaju trenutačno zaustavljanje sustava (*system halt*), s ciljem sprječavanja korupcije postojane memorije
  - djelovanje sukladno ovoj prepostavci omogućeno je mehanizmima samoprovjere na hardverskoj i softverskoj razini koje posjeduju današnji sustavi (*assertions*, *CPU checks*, *memory parity*, RAID)
- važne karakteristike pogrešaka sustava:
  - gubitak sadržaja nepostojane memorije
  - prekid rada sustava
  - postojana memorija je očuvana

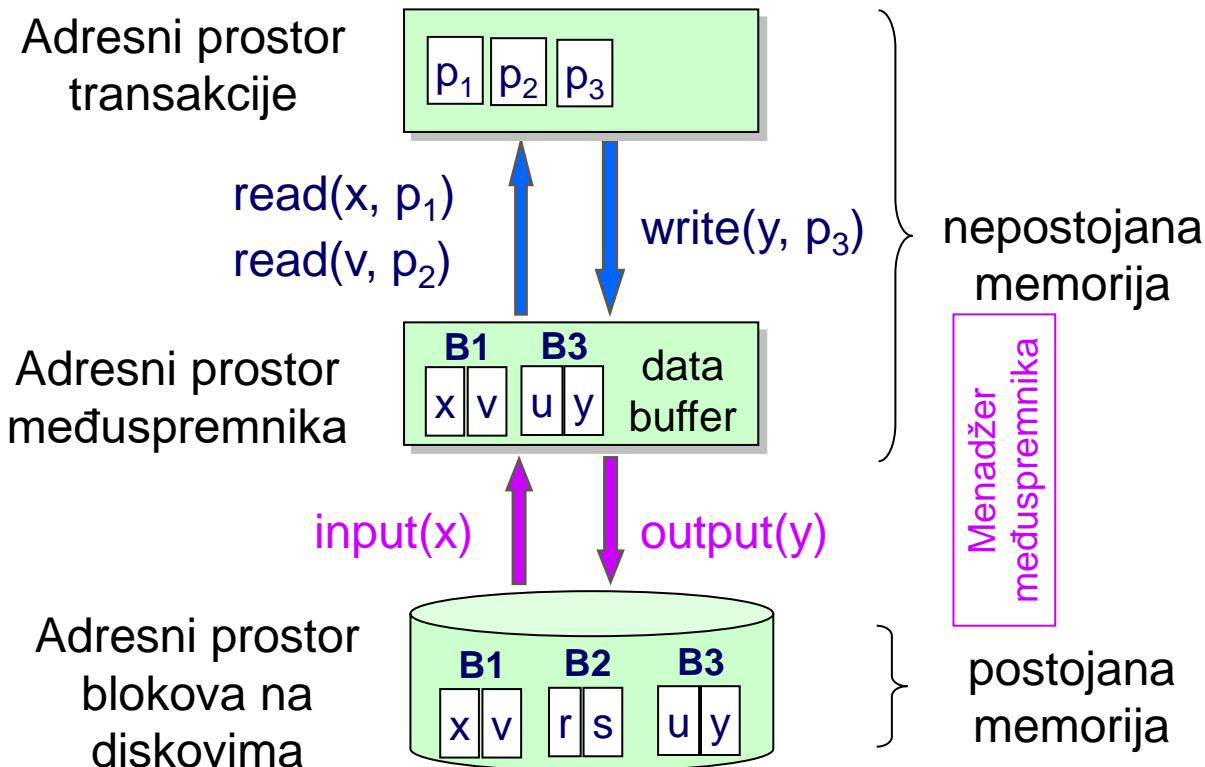
# Pogreške medija

---

- Pogreške medija odnose se na djelomični ili potpuni gubitak podataka u postojanoj memoriji
  - također mogu biti posljedica neispravnosti sklopolja, programske potpore, operacijskog sustava, SUBP-a
    - pogreške tijekom prijenosa podataka između nepostojane i postojane memorije
    - tipično za magnetske diskove: *disk crash, head crash*
  - požar, potres, vandalizam, teroristički napad, posljedice ratnog djelovanja
  - sabotaža, namjerno uništavanje podataka
- Pogreška medija u loše podešenom sustavu za obnovu u pravilu izaziva *katastrofu*
  - katastrofa (*disaster*) je tehnički pojam kojim se opisuje pogreška nakon koje nije moguća obnova

# Temeljne operacije (*primitive operations*) transakcije

- SUBP koristi tri adresna prostora
  - lokalni adresni prostor transakcija
  - adresni prostor međuspremnika baze podataka
  - adresni prostor blokova na diskovima



# Temeljne operacije (*primitive operations*) transakcije

---

## Temeljne operacije

- $\text{input}(x)$  - blok koji sadrži element  $x$  iz adresnog prostora diska kopiraj u međuspremnik
- $\text{read}(x, p)$  - ako blok u kojem se nalazi  $x$  nije u međuspremniku, obavi  $\text{input}(x)$ . Kopiraj sadržaj  $x$  iz međuspremnika u lokalnu varijablu transakcije  $p$
- $\text{write}(x, p)$  - ako blok koji sadrži element  $x$  nije u međuspremniku, obavi  $\text{input}(x)$ . Kopiraj sadržaj lokalne varijable transakcije  $p$  u međuspremnik
- $\text{output}(x)$  - kopiraj blok međuspremnika u kojem se nalazi element  $x$  u odgovarajući blok na disku

# Temeljne operacije (*primitive operations*) transakcije

- u cilju smanjenja broja U/I operacija, menadžer međuspremnika dopušta da se rezultat obavljene izmjene neko vrijeme (i uz zadovoljene određene uvjete) zadrži u nepostojanoj memoriji

Primjer: ▪ dva **objekta** u bazi podataka ( $x = 3$ ,  $y = 5$ )

- integritetsko ograničenje:  $y = x + 2$

- $p$  - lokalna varijabla transakcije T

- radi pojednostavljenja, x i y su jedini elementi u svojim blokovima

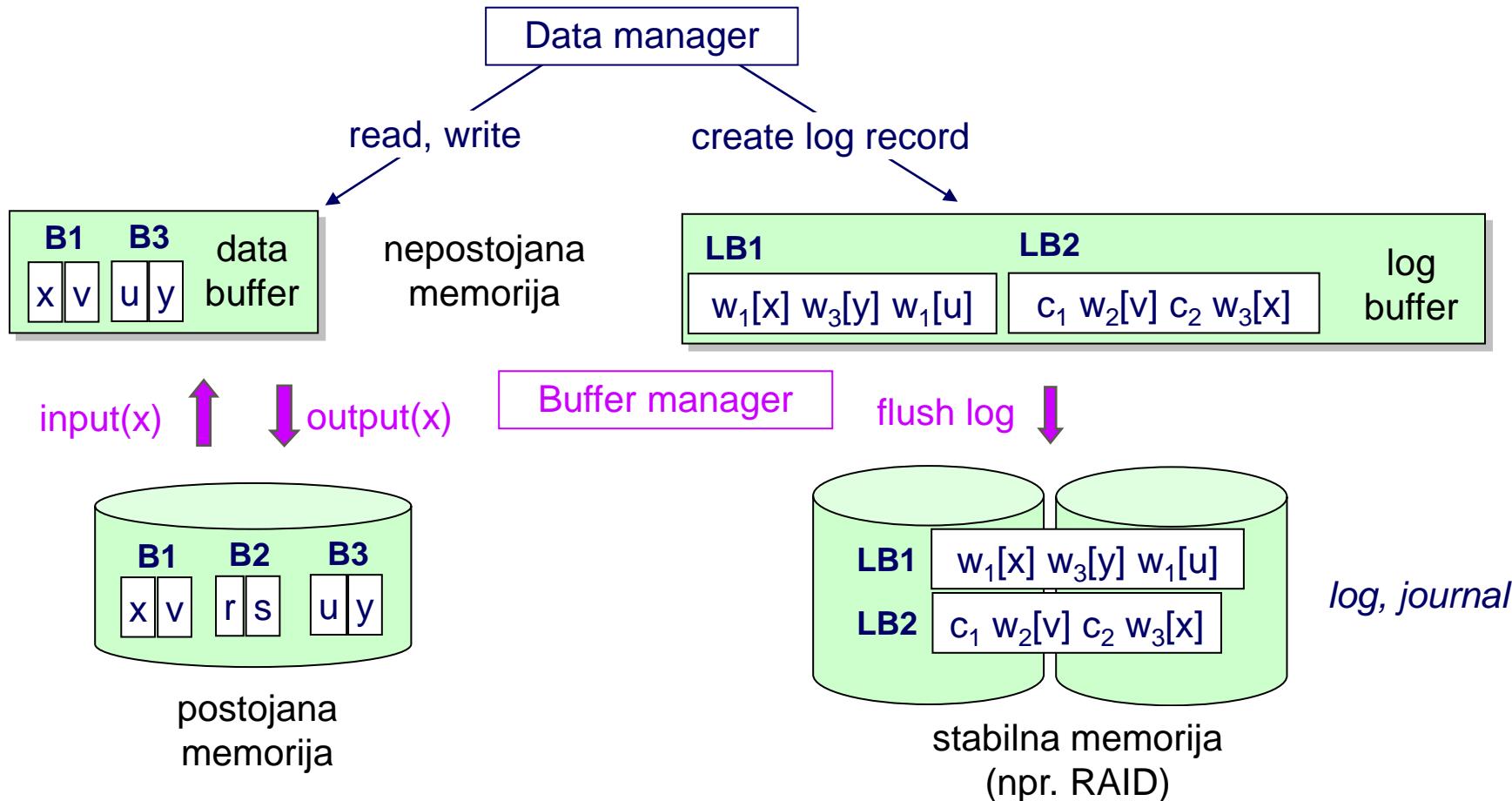
- ako se pogreška sustava dogodi prije obavljanja  $\text{output}(x)$  ili nakon obavljanja  $\text{output}(y)$ , konzistentnost baze podataka je sačuvana
- ako se pogreška sustava dogodi nakon obavljanja  $\text{output}(x)$ , a prije obavljanja  $\text{output}(y)$ , baza podataka ostaje u nekonzistentnom stanju

| T                    | Buffer manager | p | buff x | buff y | disk x | disk y |
|----------------------|----------------|---|--------|--------|--------|--------|
| begin                |                |   |        |        |        |        |
| read(x, p)           | input(x)       | 3 | 3      |        | 3      | 5      |
| $p \leftarrow p * 2$ |                | 6 | 3      |        | 3      | 5      |
| write(x, p)          |                | 6 | 6      |        | 3      | 5      |
| $p \leftarrow p + 2$ |                | 8 | 6      |        | 3      | 5      |
|                      | output(x)      | 8 | 6      |        | 6      | 5      |
| write(y, p)          | input(y)       | 8 | 6      | 8      | 6      | 5      |
| commit               |                | 8 | 6      | 8      | 6      | 8      |
|                      | output(y)      | 8 | 6      | 8      | 6      | 8      |

# Dnevnik, zapis dnevnika

- aktivnosti transakcija se bilježe u **dnevniku** (*log, journal*)

$w_1[x], r_2[v], w_3[y], r_3[u], w_1[u], c_1, r_3[x], w_2[v], c_2, r_3[v], w_3[x], a_3, r_4[v], w_4[x] \dots$



# Dnevnik, zapis dnevnika

---

- aktivnosti transakcija se bilježe u **dnevniku** (*log, journal*)
  - zapisi dnevnika se stvaraju (*create log record*) u međuspremniku dnevnika (*log buffer*) u nepostojanoj memoriji
  - dnevnik sadrži zapise o operacijama *write, commit, abort* (ali ne i *read*) svih transakcija koje su u tijeku
    - u dnevniku se zapisi različitih transakcija međusobno isprepliću (*interleaves*) jer se upisuju redom kojim se operacije izvršavaju
- menadžer međuspremnika upravlja zapisivanjem međuspremnika podataka u postojanu memoriju (operacije *output*) i međuspremnika dnevnika u **stabilnu** memoriju (operacije *flush log*)
  - kada se točno obavljaju operacije *output*, a kada operacije *flush log*, ovisi o primjenjenoj tehnici obnove (tehnike obnove objašnjene su kasnije)
- može se uočiti da će dnevnik u stabilnoj memoriji sadržavati niz zapisa dnevnika (*log records*) u obliku datoteke u koju se zapisi dodaju isključivo na začelje (*append-only file*), što znači da pri zapisivanju dnevnika u stabilnu memoriju ne dolaze do izražaja loše karakteristike magnetskih diskova (*seek time + rotational latency*)

# Dnevnik izmjena, zapis dnevnika izmjena

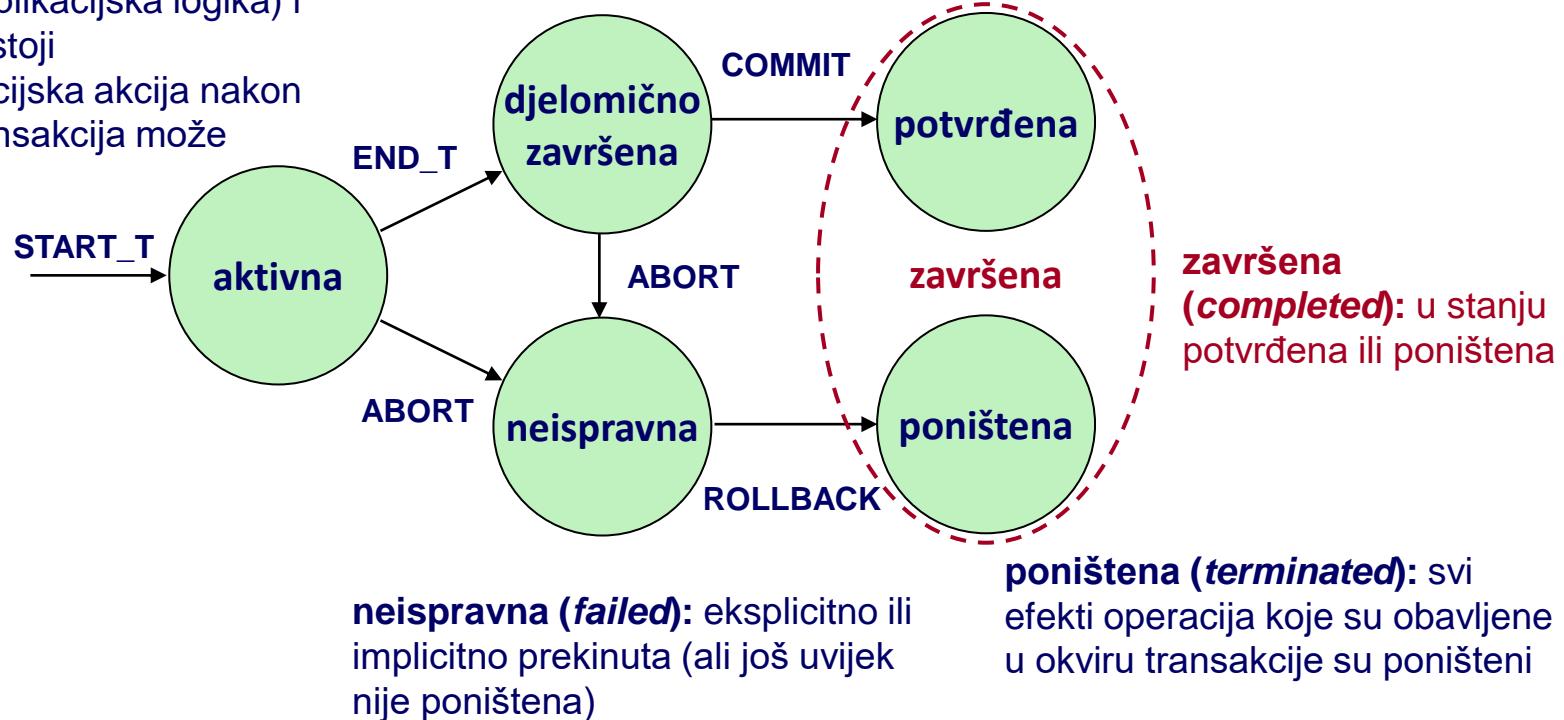
---

- **dnevnik izmjena** (*update log*) sadrži niz **zapisa dnevnika izmjena** (*update log record*)
  - različite vrsta zapisa dnevnika izmjena
    - zapis koji opisuje početak transakcije:  $\langle \text{start } T_i \rangle$ 
      - započela je transakcija s identifikatorom  $T_i$
    - zapis koji opisuje operaciju pisanja:  $\langle T_i, x_j, V_{\text{old}}, V_{\text{new}} \rangle$ 
      - transakcija s identifikatorom  $T_i$  promijenila je vrijednost elementa  $x_j$  iz prethodne vrijednosti  $V_{\text{old}}$  u novu vrijednost  $V_{\text{new}}$
      - zapis dnevnika izmjene stvara se kao posljedica operacije *write* (a ne operacije *output*)
    - zapis koji opisuje potvrđivanje transakcije:  $\langle \text{commit } T_i \rangle$
    - zapis koji opisuje prekid transakcije:  $\langle \text{abort } T_i \rangle$

# Dijagram stanja transakcije

- sa stanovišta SUBP-a, transakcija se u svakom trenutku nalazi u jednom od stanja prikazanih na slici

**aktivna (active)**: operacije se obavljaju bez pogreške ili se dešavaju samo one pogreške kojima transakcija eksplicitno upravlja (aplikacijska logika) i za koje postoji kompenzacijnska akcija nakon koje se transakcija može nastaviti.

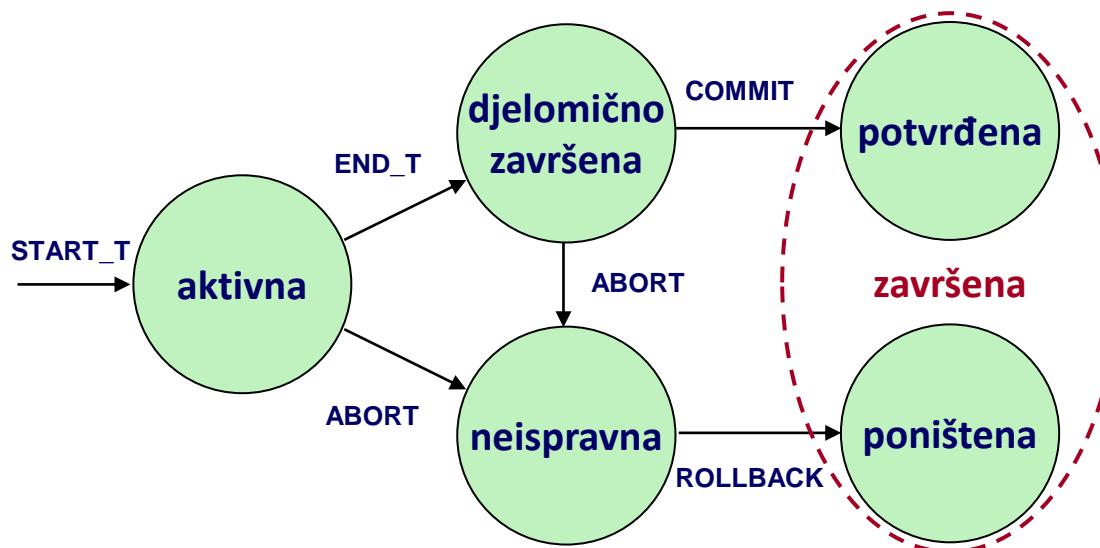


# Dijagram stanja transakcije

---

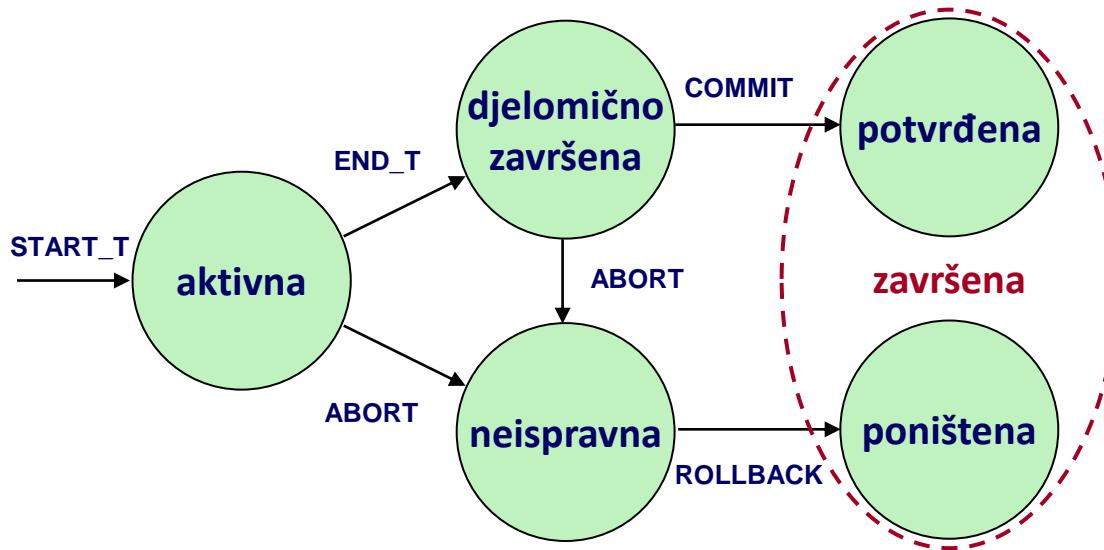
- prekid transakcije (*abort*) - korisnik ili SUBP je prekinuo transakciju
- poništavanje transakcije (*rollback*) - postupak kojim sustav poništava efekte operacija koje je transakcija eventualno provela prije prekida
- pomoću END TRANSACTION (na slici END\_T) sustavu se signalizira da su sve *read* i *write* operacije transakcije završene
- provjere integritetskih ograničenja (ako se koristi odgođena provjera integritetskih ograničenja) obavljaju se za vrijeme dok je transakcija u stanju "djelomično završena" (dakle, nakon operacije END TRANSACTION)
- najčešće se ne koriste posebne naredbe END TRANSACTION i ABORT, odnosno END TRANSACTION i COMMIT
  - SQL naredbom ROLLBACK transakcija se i prekida i poništava
  - SQL naredbom COMMIT transakcija prelazi iz stanje "aktivna" u stanje "djelomično završena", a zatim (ako za potvrđivanje transakcije ne postoji prepreka) u stanje "potvrđena"

# Završena (kompletna) i nekompletna transakcija



- završena transakcija (*completed transaction*) - u dnevniku je zabilježen ishod transakcije (*commit* ili *abort*). Pri tome je irelevantno jesu li blokovi **podataka** izmijenjeni tijekom transakcije također pohranjeni u postojanu memoriju
- nekompletna transakcija (*incomplete transaction*) - u dnevniku ne postoji niti zapis *abort*, niti zapis *commit*. Pri tome je irelevantno jesu li blokovi **podataka** izmijenjeni tijekom transakcije pohranjeni u postojanu memoriju.

# Točka potvrđivanja



- točka potvrđivanja (*commit point*) - transakcija  $T_i$  je dosegla točku potvrđivanja u trenutku kada je u stabilnu memoriju upisan zapis dnevnika  $\langle \text{commit } T_i \rangle$ . Sve izmjene koje je transakcija načinila prije točke potvrđivanja mogu se smatrati provizornim ili tentativnim (*tentative*)
- ako se poštuje svojstvo izdržljivosti transakcija, efekti transakcije koja je dosegla točku potvrđivanja niti u kojem slučaju ne bi smjeli biti izgubljeni (bez obzira na vrstu kvara)

# Korištenje dnevnika za obnovu nakon pogreške sustava

- u općem slučaju, tijekom postupka obnove nakon pogreške sustava
  - operacije nekih transakcija treba ponovo obaviti
  - operacije nekih transakcija treba poništiti
- ako se pogreška sustava dogodila prije točke potvrđivanja, efekte transakcije T treba pomoću zapisa iz dnevnika poništiti
- ako se pogreška dogodila nakon točke potvrđivanja, pomoću zapisa iz dnevnika osigurati da su svi efekti transakcije T sigurno upisani u postojanu memoriju
- mehanizam poništavanja efekata transakcije pomoću dnevnika također se koristi **na jednak način**:
  - u slučaju implicitnog ili eksplicitnog poništavanja transakcije
  - tijekom obnove nakon pogreške medija

# Korištenje dnevnika za obnovu nakon pogreške medija

---

- u općem slučaju, tijekom postupka obnove nakon pogreške medija
  - obnoviti bazu podataka pomoću arhivske kopije
    - sustav se vraća u stanje u kojem je bio u trenutku početka izrade arhivske kopije
  - korištenjem dnevnika izmjena obaviti operacije transakcija koje su potvrđene nakon trenutka u kojem je započela izrada arhivske kopije

# Tehnike obnove

---

- postoje različite tehnike obnove. O primjenjenoj tehnici ovisit će:
  - što sadrži zapis dnevnika izmjene
  - redoslijed obavljanja operacija *write*, *output* i *flush log*
  - postupak obnove
- *undo (undo/no-redo)*
  - tijekom obnove, operacije nekih transakcija se poništavaju
- *redo (no-undo/redo)*
  - tijekom obnove, operacije nekih transakcija se ponovno obavljaju
- *undo/redo*
  - tijekom obnove, neke se poništavaju, neke ponovno obavljaju
  - najčešće korištena tehnika u današnjim SUBP
- *no-undo/no-redo ili shadow paging* - neće se razmatrati
  - bez dnevnika: koriste se kopije blokova s podacima (stari blok, novi blok, *current directory*, *shadow directory*)

# **Undo - sadržaj dnevnika i pravila vođenja dnevnika**

- zapis koji opisuje operaciju pisanja sadrži samo staru vrijednost elementa:  
 $\langle T_i, x_j, V_{old} \rangle$ 
  - transakcija s identifikatorom  $T_i$  promijenila je vrijednost elementa  $x_j$  čija je **prethodna** vrijednost bila  $V_{old}$  na novu vrijednost  $V_{new}$
- pravila vođenja dnevnika:
  1. ako transakcija  $T$  obavlja operaciju  $write(x, V_{new})$ , operacija *flush log* koja će upisati zapis dnevnika  $\langle T, x, V_{old} \rangle$  mora biti obavljena prije operacije *output(x)*
  2. operacija *flush log* kojom se zapis dnevnika  $\langle commit T \rangle$  upisuje u stabilnu memoriju obavlja se isključivo nakon što su obavljene sve *output(x)* operacije transakcije  $T$

Pravilo koje zahtijeva da se zapisi dnevnika upisuju u stabilnu memoriju **prije** nego se u postojanu memoriju upiše promijenjena vrijednost elementa naziva se *write-ahead logging (WAL)*

# Undo - sadržaj dnevnika i pravila vođenja dnevnika

Primjer: ■ integritetska ograničenja:  $y = x * 2$ ;  $z = x + 4$ ;

|     | T <sub>1</sub>            | Buffer manager | buff x | buff y | buff z | disk x | disk y | disk z |
|-----|---------------------------|----------------|--------|--------|--------|--------|--------|--------|
| 1.  | begin <sub>1</sub>        |                |        |        |        | 1      | 2      | 5      |
| 2.  | write <sub>1</sub> (x, 3) |                | 3      |        |        | 1      | 2      | 5      |
| 3.  |                           | flush log      | 3      |        |        | 1      | 2      | 5      |
| 4.  | write <sub>1</sub> (y, 6) |                | 3      | 6      |        | 1      | 2      | 5      |
| 5.  |                           | output(x)      | 3      | 6      |        | 3      | 2      | 5      |
| 6.  | write <sub>1</sub> (z, 7) |                | 3      | 6      | 7      | 3      | 2      | 5      |
| 7.  |                           | flush log      | 3      | 6      | 7      | 3      | 2      | 5      |
| 8.  |                           | output(y)      | 3      | 6      | 7      | 3      | 6      | 5      |
| 9.  |                           | output(z)      | 3      | 6      | 7      | 3      | 6      | 7      |
| 10. | commit <sub>1</sub>       |                | 3      | 6      | 7      | 3      | 6      | 7      |
| 11. |                           | flush log      | 3      | 6      | 7      | 3      | 6      | 7      |

1.↓      2.↓      4.↓      6.↓      10.↓

log buffer      <start T<sub>1</sub>> <T<sub>1</sub>,x,1> <T<sub>1</sub>,y,2> <T<sub>1</sub>,z,5> <commit T<sub>1</sub>>

3.↓      7.↓      11.↓

log      <start T<sub>1</sub>> <T<sub>1</sub>,x,1> <T<sub>1</sub>,y,2> <T<sub>1</sub>,z,5> <commit T<sub>1</sub>>

# **Undo - postupak obnove nakon pogreške sustava**

---

- *restart recovery*
- za vrijeme obnove, sustav ne zaprima nove transakcije
- čitati dnevnik **od kraja prema početku** te za svaki zapis iz dnevnika
  - ako pročitaš zapis  $\langle \text{commit } T_i \rangle$  dodaj  $T_i$  u listu potvrđenih transakcija
  - ako pročitaš zapis  $\langle T_i, x, V \rangle$ 
    - ako je  $T_i$  u listi potvrđenih transakcija, činiti ništa\*
    - inače ako  $T_i$  nije u listi potvrđenih transakcija, obavi operaciju  $\text{undo}(x, V)$ , tj. promijeni vrijednost elementa  $x$  na vrijednost  $V$ \*\*
- \* ako je u dnevniku zabilježen zapis  $\langle \text{commit } T_i \rangle$ , tada su sve *output* operacije transakcije  $T_i$  sigurno uspješno izvršene već prije pogreške
- \*\* ako u dnevniku nije zabilježen zapis  $\langle \text{commit } T_i \rangle$ , možda su neke *output* operacije transakcije  $T_i$  obavljene, a neke nisu. Poništiti sve efekte transakcije  $T_i$
- što ako se pogreška sustava opet dogodi za vrijeme obnove?
  - ponavljanje postupka obnove, bez obzira u kojem je trenutku prekinut, neće izazvati probleme jer je operacija *undo* idempotentna
    - $\text{undo}(x, V), \text{undo}(x, V), \dots \equiv \text{undo}(x, V)$

# **Undo - postupak obnove nakon pogreške sustava**

---

## **Primjer:**

- neka se u prethodnom primjeru pogreška sustava dogodila nakon 5. koraka
- na početku obnove sadržaj dnevnika jest: .... <start T<sub>1</sub>> <T<sub>1</sub>,x,1>
- tijekom obnove
  - u element x, čija je vrijednost promijenjena na 3, zapisuje se vrijednost 1

## **Za vježbu odgovoriti na pitanja:**

- što će se desiti ako se pogreška sustava dogodi nakon 8. koraka?
- što će se desiti ako se pogreška sustava dogodi nakon 10. koraka?
  - smije li SUBP korisniku dojaviti "Transakcija je potvrđena" nakon 10 koraka?
- što će se desiti ako se pogreška sustava dogodi nakon 11. koraka?
  - smije li SUBP korisniku dojaviti "Transakcija je potvrđena" nakon 11 koraka?
- što će se desiti ako se pogreška sustava dogodila nakon 8. koraka, a za vrijeme obnove se opet dogodila pogreška nakon obavljanja operacije *undo(y, 2)*

# Undo - nedostaci

---

- potreba obavljanja većeg broja U/I operacija prije potvrđivanja transakcije:
  - zapis dnevnika  $\langle \text{commit } T_i \rangle$  mora se u dnevnik upisati čim je prije moguće (zbog izdržljivosti transakcije). S druge strane,  $\langle \text{commit } T_i \rangle$  smije biti upisan u dnevnik tek nakon što su obavljene sve pripadne *output* operacije. Zbog toga je menadžer međuspremnika prisiljen najkasnije prije točke potvrđivanja transakcije obaviti sve *output* operacije koje su povezane s izmijenjenim elementima. Time se uvećava broj U/I operacija koje se moraju obaviti tijekom transakcije
  - primjer: mnogo vrlo kratkih transakcija mijenja isti element u bazi podataka. LRU bi vjerojatno odgodio obavljanje operacije *output* za taj element jer ga stalno koriste mnogi procesi, ali je zbog gore navedenih razloga prisiljen obaviti operaciju *output* prije dosezanja točke potvrđivanje svake od transakcija
- ograničena mogućnost obnove baze podataka pomoću arhivske kopije
  - u slučaju pogreške medija (baza podataka je uništena) koristi se arhivska kopija. Dnevnik nije upotrebljiv za ponovno obavljanje transakcija koje su izvršene nakon posljednjeg arhiviranja baze

# Redo

---

## Redo - sadržaj dnevnika

- zapis koji opisuje operaciju pisanja sadrži samo novu vrijednost elementa:  
 $\langle T_i, x_j, V_{new} \rangle$ 
  - transakcija s identifikatorom  $T_i$  promijenila je vrijednost elementa  $x_j$  na **novu** vrijednost  $V_{new}$

## Redo - pravila vođenja dnevnika

1. bilo koja operacija  $output(x)$  transakcije  $T$  smije se obaviti tek nakon što su **svi** zapisi dnevnika  $\langle T, x, V_{new} \rangle$  **i zapis**  $\langle commit T \rangle$  transakcije upisani u stabilnu memoriju

# Redo

---

## Redo - postupak obnove

- identificirati potvrđene transakcije (za koje u dnevniku postoji zapis `<commit>`)
- čitati dnevnik **od početka prema kraju**
  - za svaki zapis  $\langle T_i, x_j, V \rangle$ 
    - ako je  $T_i$  potvrđena, obaviti operaciju  $redo(x_j, V)$
    - ako  $T_i$  nije potvrđena, činiti ništa\*

\* ako u dnevniku ne postoji zapis `<commit Ti>` tada niti jedna  $output(x)$  operacija transakcije  $T_i$  sigurno nije obavljena

## Redo - pogreška sustava tijekom obnove

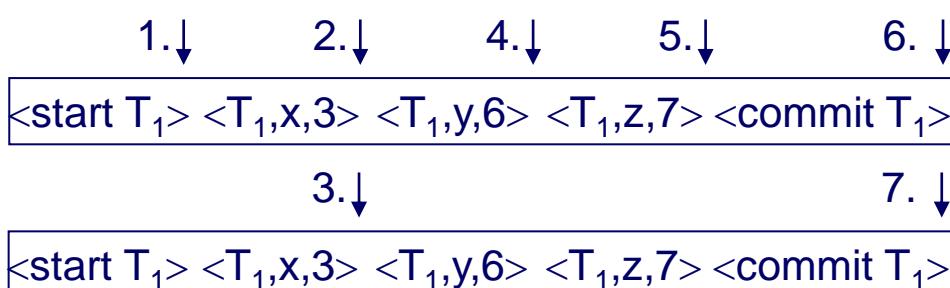
- ponovno obavljanje opisanog postupka obnove, bez obzira u kojem je trenutku prekinut, neće izazvati probleme jer je operacija *redo* idempotentna
  - $redo(x, V), redo(x, V), \dots \equiv redo(x, V)$

# Redo - sadržaj dnevnika i pravila vođenja dnevnika

Primjer: ■ integrietska ograničenja:  $y = x * 2$ ;  $z = x + 4$ ;

|     | $T_1$                     | Buffer manager | buff x | buff y | buff z | disk x | disk y | disk z |
|-----|---------------------------|----------------|--------|--------|--------|--------|--------|--------|
| 1.  | begin <sub>1</sub>        |                |        |        |        | 1      | 2      | 5      |
| 2.  | write <sub>1</sub> (x, 3) |                | 3      |        |        | 1      | 2      | 5      |
| 3.  |                           | flush log      | 3      |        |        | 1      | 2      | 5      |
| 4.  | write <sub>1</sub> (y, 6) |                | 3      | 6      |        | 1      | 2      | 5      |
| 5.  | write <sub>1</sub> (z, 7) |                | 3      | 6      | 7      | 1      | 2      | 5      |
| 6.  | commit <sub>1</sub>       |                | 3      | 6      | 7      | 1      | 2      | 5      |
| 7.  |                           | flush log      | 3      | 6      | 7      | 1      | 2      | 5      |
| 8.  |                           | output(x)      | 3      | 6      | 7      | 3      | 2      | 5      |
| 9.  |                           | output(y)      | 3      | 6      | 7      | 3      | 6      | 5      |
| 10. |                           | output(z)      | 3      | 6      | 7      | 3      | 6      | 7      |

log  
buffer  
  
log



- *output* niti jednog elementa se ne smije obaviti prije nego je commit zapis dnevnika za transakciju zapisan u stabilnu memoriju

# Redo - postupak obnove nakon pogreške sustava

## Primjer:

- neka se u prethodnom primjeru pogreška sustava dogodila nakon 4. koraka
- na početku obnove sadržaj dnevnika jest:

```
<start T1> <T1,x,3>
```

- u dnevniku nema zapisa <commit T<sub>1</sub>>   ⇒ činiti ništa

## Primjer:

- neka se u prethodnom primjeru pogreška sustava dogodila nakon 8. koraka
  - na početku obnove sadržaj dnevnika jest:
- ```
<start T1> <T1,x,3> <T1,y,6> <T1,z,7> <commit T1>
```
- tijekom obnove u x se (ponovo) zapisuje vrijednost 3, u y se zapisuje 6, u z se zapisuje 7

# Redo - prednosti i nedostaci

---

## Nedostaci (u odnosu na *undo* tehniku):

- potencijalna potreba za velikim međuspremnikom podataka:
  - vrijednosti iz međuspremnika s podacima koje je promijenila transakcija  $T_i$  se ne smije upisati u postojanu memoriju (i time potencijalno osloboditi međuspremnik) prije nego se zapis dnevnika  $\langle \text{commit } T_i \rangle$  zapise u stabilnu memoriju. Zbog tog se pravila uvećava potrebna veličina međuspremnika
  - primjer: transakcija koja je izmjenila veliki broj elemenata i duže vrijeme ne obavi naredbu commit. LRU možda zatreba izbaciti te stranice iz međuspremnika (prije toga obaviti output), ali ne smije prije nego dotična transakcija dosegne točku potvrđivanja

## Prednosti (u odnosu na *undo* tehniku):

- manja ograničenja redoslijeda obavljanja operacija
  - menadžer međuspremnika može po volji odgađati obavljanje operacija *output* i time optimizirati broj obavljenih U/I operacija
    - transakcija može biti potvrđena (i njezina izdržljivost garantirana) i onda kada nisu obavljene sve pripadne *output* operacije

# Redo - prednosti i nedostaci

## Prednosti (u odnosu na *undo* tehniku):

- veća mogućnost obnove baze podataka iz arhivske kopije
  - u slučaju pogreške medija (baza podataka je uništena) koristi se arhivska kopija. Obavljanjem *redo* operacija za zapise dnevnika potvrđenih transakcija baza podataka se može dovesti u stanje koje odgovara stanju neposredno prije pogreške medija

*Redo* tehnikom (jednako tako i *undo/redo* tehnikom prikazanom kasnije) omogućeno je odgađanje svih *output* operacija tijekom vrlo dugog vremena. U ekstremnom slučaju, SUBP bi mogao odgoditi obavljanje svih *output* operacije sve do trenutka zaustavljanja.

Zašto takvu mogućnost ipak ne bi trebalo koristiti?

Radi mogućnosti brze obnove nakon pogreške sustava. Ako bi se tijekom obnove sustava morao obaviti veliki broj *redo* operacija, obnova bi dugo trajala. Kako se rješava taj problem  
→ kontrolna točka (u sljedećim predavanjima)

# **Undo/Redo**

---

## **Undo/Redo - sadržaj dnevnika**

- zapis koji opisuje operaciju pisanja:  $\langle T_i, x_j, V_{old}, V_{new} \rangle$ 
  - transakcija s identifikatorom  $T_i$  promijenila je vrijednost elementa  $x_j$  iz prethodne vrijednosti  $V_{old}$  u novu vrijednost  $V_{new}$

## **Undo/Redo - pravila vođenja dnevnika**

1. ako transakcija  $T$  obavlja operaciju  $write(x, V_{new})$ , operacija *flush log* kojom je zapis dnevnika  $\langle T, x, V_{old}, V_{new} \rangle$  upisan u stabilnu memoriju mora biti obavljena prije operacije  $output(x)$
- 
- razlika u odnosu na *redo*:
    - nije propisan redoslijed između zapisivanja  $\langle commit T_i \rangle$  u stabilnu memoriju i zapisivanja blokova s podacima u postojanu memoriju

# **Undo/Redo - postupak obnove nakon pogreške sustava**

- **čitati dnevnik od početka prema kraju**
  - ako za transakciju  $T_i$  u dnevniku postoji zapis  $\langle \text{commit } T_i \rangle$  tada obaviti operaciju  $\text{redo}(x_j, V_{new})$  za svaki zapis  $\langle T_i, x_j, V_{old}, V_{new} \rangle$  iz dnevnika\*
- **čitati dnevnik od kraja prema početku**
  - ako za transakciju  $T_i$  u dnevniku ne postoji zapis  $\langle \text{commit } T_i \rangle$  tada obaviti operaciju  $\text{undo}(x_j, V_{old})$  za svaki zapis  $\langle T_i, x_j, V_{old}, V_{new} \rangle$  iz dnevnika\*\*

\* zato jer transakcije jest potvrđena, a neke od njenih *output* operacije možda nisu obavljene

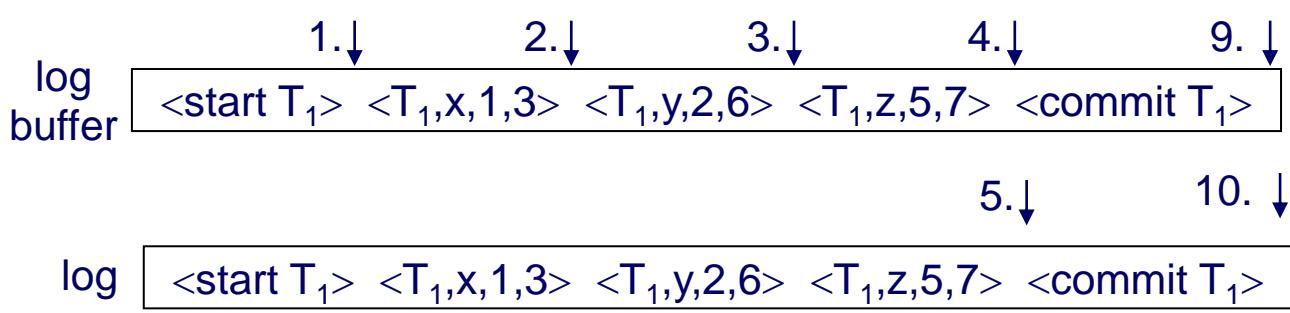
\*\* zato jer transakcija nije potvrđena, a neke od njenih *output* operacija možda jesu obavljene

- što ako se pogreška sustava opet dogodi za vrijeme obnove?
  - ponavljanje opisanog postupka obnove, bez obzira u kojem je trenutku prekinut, neće izazvati probleme jer su operacije *undo* i *redo* idempotentne

# Undo/Redo - sadržaj dnevnika i pravila vođenja dnevnika

Primjer: ■ integritetska ograničenja:  $y = x * 2$ ;  $z = x + 4$ ;

	T <sub>1</sub>	Buffer manager	buff x	buff y	buff z	disk x	disk y	disk z
1.	begin <sub>1</sub>					1	2	5
2.	write <sub>1</sub> (x, 3)		3			1	2	5
3.	write <sub>1</sub> (y, 6)		3	6		1	2	5
4.	write <sub>1</sub> (z, 7)		3	6	7	1	2	5
5.	flush log		3	6	7	1	2	5
6.	output(x)		3	6	7	3	2	5
7.	output(y)		3	6	7	3	6	5
8.	output(z)		3	6	7	3	6	7
9.	commit <sub>1</sub>		3	6	7	3	6	7
10.	flush log		3	6	7	3	6	7



- ako se pogreška sustava dogodi nakon koraka 7. tijekom obnove obavit će se operacije *undo(z,5)*, *undo(y,2)* i *undo(x,1)*

# Undo/Redo - sadržaj dnevnika i pravila vođenja dnevnika

Primjer: ■ integritetska ograničenja:  $y = x * 2$ ;  $z = x + 4$ ;

	T <sub>1</sub>	Buffer manager	buff x	buff y	buff z	disk x	disk y	disk z
1.	begin <sub>1</sub>					1	2	5
2.	write <sub>1</sub> (x, 3)		3			1	2	5
3.	write <sub>1</sub> (y, 6)		3	6		1	2	5
4.	write <sub>1</sub> (z, 7)		3	6	7	1	2	5
5.	commit <sub>1</sub>		3	6	7	1	2	5
6.	flush log		3	6	7	1	2	5
7.	output(x)		3	6	7	3	2	5
8.	output(y)		3	6	7	3	6	5
9.	output(z)		3	6	7	3	6	7

1. ↓

2. ↓

3. ↓

4. ↓

5. ↓

log  
buffer

<start T<sub>1</sub>> <T<sub>1</sub>,x,1,3> <T<sub>1</sub>,y,2,6> <T<sub>1</sub>,z,5,7> <commit T<sub>1</sub>>

6. ↓

log

<start T<sub>1</sub>> <T<sub>1</sub>,x,1,3> <T<sub>1</sub>,y,2,6> <T<sub>1</sub>,z,5,7> <commit T<sub>1</sub>>

- ako se pogreška sustava dogodi nakon koraka 7. tijekom obnove obaviti će se operacije  $redo(x,3)$ ,  $redo(y,6)$  i  $redo(z,5)$

# ***Undo/Redo - prednosti i nedostaci pred redo tehnikom***

---

## **Prednost (u odnosu na redo tehniku):**

- nema potrebe za velikim međuspremnikom podataka
  - menadžer međuspremnika može (ali ne mora) obaviti *output* operacije neke transakcije T odmah nakon upisivanja pripadnih zapisu dnevnika u stabilnu memoriju. Time što ne mora čekati završetak transakcije (kao kod *redo* tehnike) smanjuje se potreba za velikim međuspremnikom podataka

## **Nedostatak (u odnosu na undo i redo tehniku):**

- tijekom obnove, potreban je prolazak kroz dnevnik u oba smjera
- povećani utrošak prostora za pohranu dnevnika
  - bilježe se i stare i nove vrijednosti elementa koji je promijenjen

# Literatura:

---

- H. Garcia-Molina, J. Ullman, J. Widom. Database system implementation. Prentice-Hall International, Inc. Upper Saddle River, New Jersey. 2000.
- R. Elmasri, S. Navathe. Fundamentals of database systems, 3rd ed. Addison-Wesley Publishing Company. Reading, Massachusetts. 2000.
- A. Silberschatz, H. Korth, S. Sudarshan. Database system concepts, 6th ed. McGraw-Hill. 2010.
- P. Bernstein, V. Hadzilacos, N. Goodman. Concurrency control and recovery in database systems. Addison-Wesley Publishing Company. Reading, Massachusetts. 1987.
- Archive and Backup Guide for Informix Dynamic Server, Version 7.3, Informix Press. 1998.
- Oracle Database Backup and Recovery Advanced User's Guide 10g Release 2 (10.2)

# **Sustavi baza podataka**

Predavanja

**7. Sustav za obnovu  
(2. dio)**  
travanj 2023.

# Dnevnik, dnevnik izmjena, fizički dnevnik, logički dnevnik

---

- aktivnosti transakcija se bilježe u **dnevniku** (*log, journal*)
- jedan oblik implementacije dnevnika je **dnevnik izmjena** (*update log*).
  - sadrži zapise dnevnika izmjena
    - $\langle \text{start } T_i \rangle, \langle T_i, x_j, V_{\text{old}}, V_{\text{new}} \rangle, \langle \text{commit } T_i \rangle, \langle \text{abort } T_i \rangle$
- **novi pojmovi: fizički dnevnik, logički dnevnik**
  - nekonzistentna upotreba ovih pojmoveva u literaturi
  - (Silberschatz, 2010.)
    - fizički dnevnik sadrži zapise oblika *old-value, new-value*
    - logički dnevnik sadrži opise *operacija* koje su obavljene
      - za obavljanje kompenzacijskih operacija tijekom obnove
  - (Garcia-Molina, 2000.)
    - fizički dnevnik sadrži zapise oblika: *old-block, new-block*
    - logički dnevnik sadrži zapise oblika: *old-value, new-value*
    - u nastavku se koriste termini iz (Garcia-Molina, 2000.)

# Primjer sadržaja logičkog dnevnika u sustavu IBM Informix

Informativno

## osoba

oib	ime	prez
12345678901	Ivica	Horvat

Korisnici **miha** i **jure** obavljaju naredbe:

```
(m) BEGIN WORK;
(m) UPDATE osoba SET
    ime='Marko', prez='Ban'
    WHERE oib = '12345678901';
(j) BEGIN WORK;
(j) INSERT INTO osoba
    VALUES ('22233322233',
    'Mendo', 'Slavica');
(m) INSERT INTO osoba
    VALUES ('3334444555',
    'Ivica', 'Horvat');
(m) COMMIT WORK;
(j) COMMIT WORK;
```

# onlog -l -n 3360						
addr	len	type	xid	id	link	
48018	52	BEGIN	22	3360		12/22/2008 19:40:17 31 miha
	340000000	200d0100	00000000	00000000	4...	.....
		[skraćeno]				....
4804c	104	HUPDAT	22		48018	200f5e 101 0 31 31 2
	680000000	00004900	12000000	00000000	h.....I.	.....
	000000000	00000000	16000000	18800400		.....
	7e16e100	5e0f2000	5e0f2000	01010000	~...^.. . ^.	.....
	000000000	1f001f00	02000000	00000000		.....
	7075626c	000b0006	49766963	61204d61	publ.... Ivica Ma	
	726b6f20	00150007	486f7276	61742042	rko .... Horvat B	
	616e2020	20202020				an
480b4	52	BEGIN	24	3360		12/22/2008 19:40:29 41 jure
	340000000	200d0100	00000000	00000000	4...	.....
		[skraćeno]				....
480e8	96	HINSERT	24		480b4	200f5e 201 31
		[skraćeno]				
	32323233	33333232	3233334d	656e646f	22233322 233Mendo	
	20202020	20536c61	76696361	20202000	Sla vica .	
48148	96	HINSERT	22		4804c	200f5e 102 31
		[skraćeno]				
	33333334	34343435	35352049	76696361	33344445 55 Ivica	
	20202020	20486f72	76617420	20202000	Hor vat .	
481a8	48	COMMIT	22		48148	12/22/2008 19:40:36
		[skraćeno]				

addr – Log record address

len – record length

type – record type name

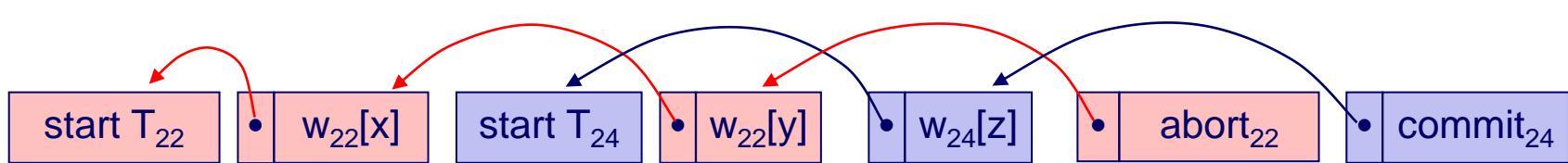
xid – transaction number

id – logical log number

link – link to the previous record in the transaction

# Efikasno poništavanje transakcija

## punjene dnevnika



- zapisi različitih transakcija su u dnevnik upisani sekvencijalno i naizmjence
- poništavanje transakcija zbog pogreške transakcije ili eksplisitnog zahtjeva za poništavanjem transakcije obavlja se izvršavanjem operacije *undo* za sve zapise izmjene te transakcije
- operacije *undo* se obavljaju u smjeru od kraja dnevnika prema početku
- sustav mora imati mogućnost brzog pronalaženja svih prethodnih zapisa određene transakcije (počevši od zapisa *abort/rollback*) - to omogućuje pokazivač prema prethodnom zapisu iste transakcije u dnevniku

# Kontrolna točka

---

## Problem:

- za sve do sada prikazane tehnike pretpostavljali smo da se u slučaju **pogreške sustava** obnova mora provesti na temelju **cijelog** dnevnika
- koliko je dnevnik velik?
  - ovisi o učestalosti operacija izmjena i veličini podataka koji se mijenjaju

ISVU - Informacijski sustav visokih učilišta u RH (stanje 2009. godine)

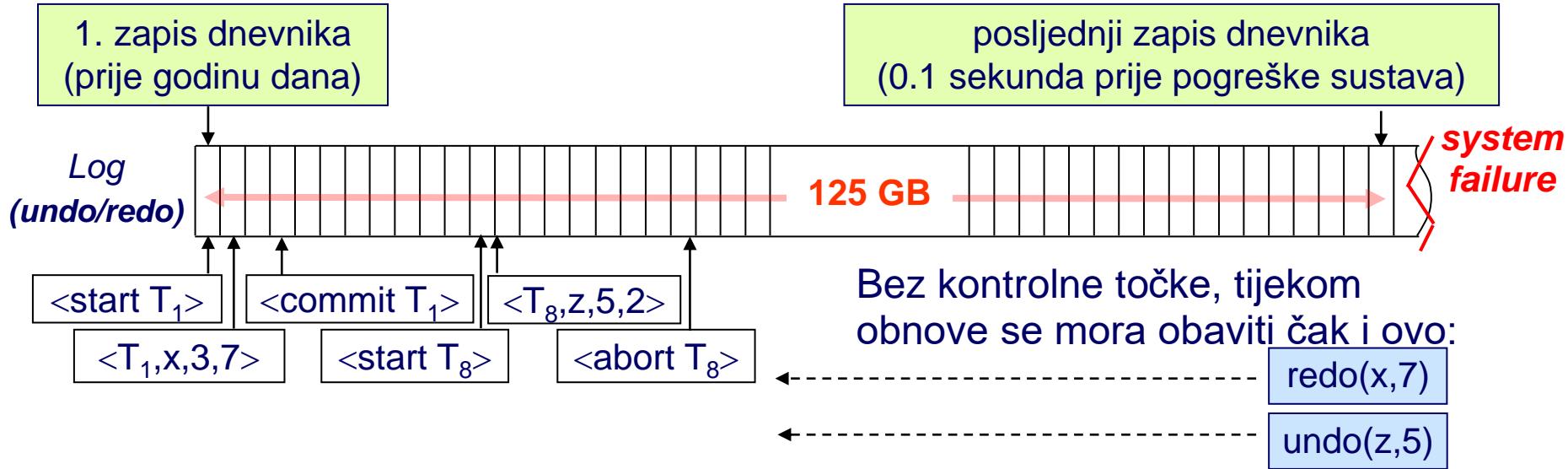
350 000 studenata

1 000 000 izdanih potvrda

5 000 000 ispita

prirast dnevnika: prosječno 350 MB/dan  $\Rightarrow$  125 GB/godinu

# Kontrolna točka



## Problemi koji se rješavaju upotrebom kontrolne točke

1. Trajanje obnove: satima? danima?
  - Kako bi postupak obnove korištenjem cijelog dnevnika utjecao na srednje vrijeme potrebno za oporavak (MTTR) i raspoloživost sustava (*availability*)?
2. Čuvanje cijelog dnevnika u stabilnoj memoriji: godinama?

# Kontrolna točka

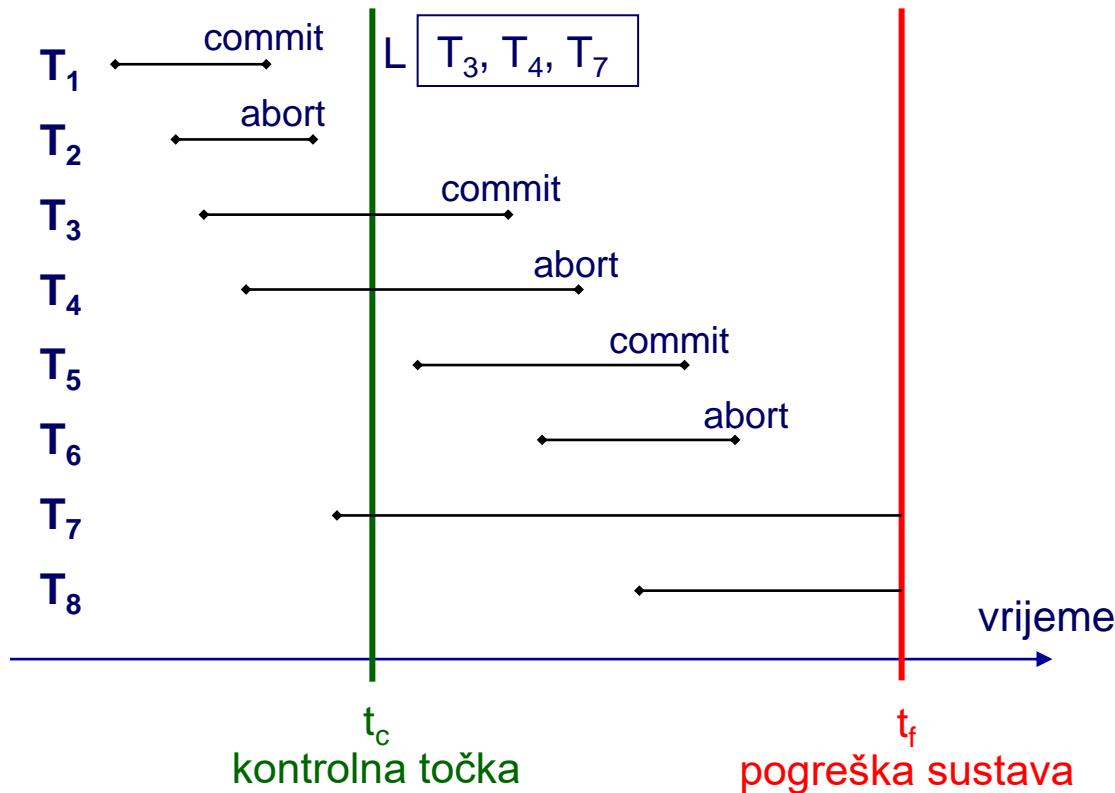
---

## Postavljanje/obavljanje kontrolne točke

- periodički (npr. svakih 10 minuta) sustav određuje kontrolnu točku (*checkpoint*)
  1. SUBP prestaje obavljati operacije koje zahtijevaju pisanje u međuspremnik podataka ili međuspremnik dnevnika
  2. sadržaj međuspremnika dnevnika (*log buffer*) se zapisuje u stabilnu memoriju
  3. stranice međuspremniku podataka (*data buffer*) čiji je sadržaj promijenjen (*dirty pages*) zapisuje se u postojanu memoriju
  4. u dnevnik se upisuje zapis <chkpt L>\*
  5. SUBP nastavlja obavljati operacije koje zahtijevaju pisanje u međuspremnik podataka ili dnevnika

\* L je lista identifikatora svih transakcija  $T_1, T_2, \dots, T_k$  koje su bile aktivne u trenutku kontrolne točke.

# Kontrolna točka



- $T_1$  i  $T_2$  su započele i završile prije  $t_c$ .  $T_1$  je potvrđena,  $T_2$  je poništена
- $T_3$  i  $T_4$  su započele prije  $t_c$ , završile između  $t_c$  i  $t_f$ .  $T_3$  je potvrđena,  $T_4$  je poništена
- $T_5$  i  $T_6$  su započele i završile između  $t_c$  i  $t_f$ .  $T_5$  je potvrđena,  $T_6$  je poništена
- $T_7$  je započela prije  $t_c$  i bila je aktivna u trenutku  $t_f$
- $T_8$  je započela između  $t_c$  i  $t_f$  i bila je aktivna u trenutku  $t_f$

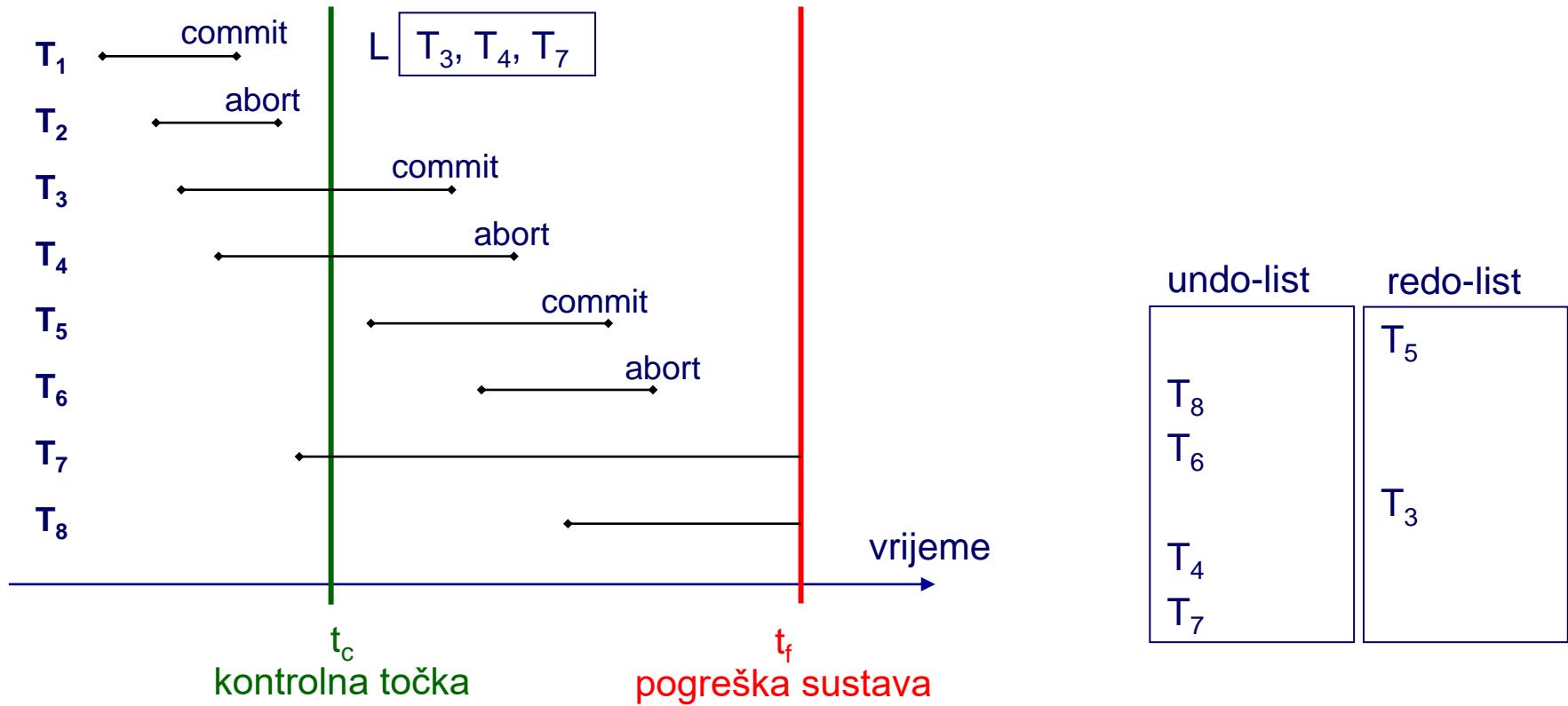
# Kontrolna točka - postupak obnove nakon pogreške sustava

---

- formiraj dvije prazne liste: redo-list i undo-list
- čitaj dnevnik unatrag od kraja dnevnika do prvog  $\langle \text{chkpt } L \rangle$  zapisa
  - za svaki zapis oblika  $\langle \text{commit } T_i \rangle$  dodaj  $T_i$  u redo-list
  - za svaki zapis oblika  $\langle \text{start } T_i \rangle$ , ako se  $T_i$  ne nalazi u redo-list, dodaj  $T_i$  u undo-list
- svaku transakciju  $T_i$  iz  $L$  koja nije u redo-list dodaj u undo-list
  
- čitaj dnevnik unatrag od kraja
  - za svaki zapis  $\langle T_i, x, V_1, V_2 \rangle$ , ako je  $T_i$  u undo-list, obavi  $undo(x, V_1)$
  - čitanje dnevnika unatrag prestaje kad se pronađu zapisi  $\langle \text{start } T_i \rangle$  za svaku transakciju  $T_i$  iz undo-list
- čitaj dnevnik od zadnje kontrolne točke do kraja dnevnika (*roll forward*)
  - za svaki zapis  $\langle T_i, x, V_1, V_2 \rangle$ , ako je  $T_i$  u redo-list, obavi  $redo(x, V_2)$

# Kontrolna točka - postupak obnove nakon pogreške sustava

Primjer:



- *undo* operacije će se obaviti za sve zapise dnevnika transakcija  $T_4, T_6, T_7$  i  $T_8$
- *redo* operacije će se obaviti za zapise dnevnika transakcija  $T_3$  i  $T_5$  koji su nastali nakon  $t_c$

# Kontrolna točka

---

O redoslijedu koraka koji se obavljaju u kontrolnoj točki

1. SUBP prestaje obavljati operacije koje zahtijevaju pisanje u međuspremnik podataka ili međuspremnik dnevnika
2. sadržaj međuspremnika dnevnika (*log buffer*) se zapisuje u stabilnu memoriju
3. sadržaj međuspremnika podataka (*data buffer*) se zapisuje u postojanu memoriju
4. u dnevnik (u stabilnu memoriju!) upisuje se zapis <chkpt L>
5. SUBP nastavlja obavljati operacije koje zahtijevaju pisanje u međuspremnik podataka ili dnevnika

Za vježbu odgovoriti na pitanja:

- zašto se korak 1 mora obaviti prije koraka 2?
- zašto se korak 2 mora obaviti prije koraka 3?
- zašto se korak 3 mora obaviti prije koraka 4?

# Upravljanje logičkim dnevnicima

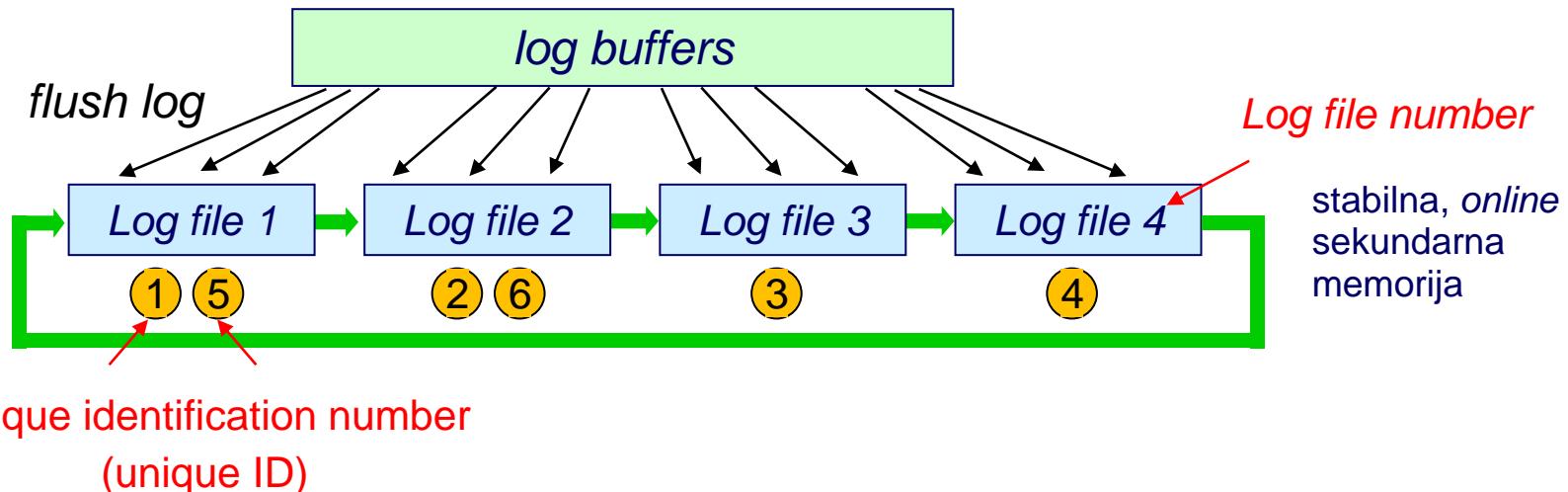
---

- s obzirom na mogući veliki prirast dnevnika, potreban je efikasan mehanizam upravljanja dnevnicima
- zapisi dnevnika se upisuju u (*online*) stabilnu memoriju (*raw* ili *cooked device*)
  - *online* memorija: povezana s računalom, neprekidno i odmah dostupna bez intervencije operatera
  - *offline* memorija: uglavnom se odnosi na trake, optičke medije, *pluggable HDD*, *pluggable SSD* i slično

# Upravljanje logičkim dnevnicima

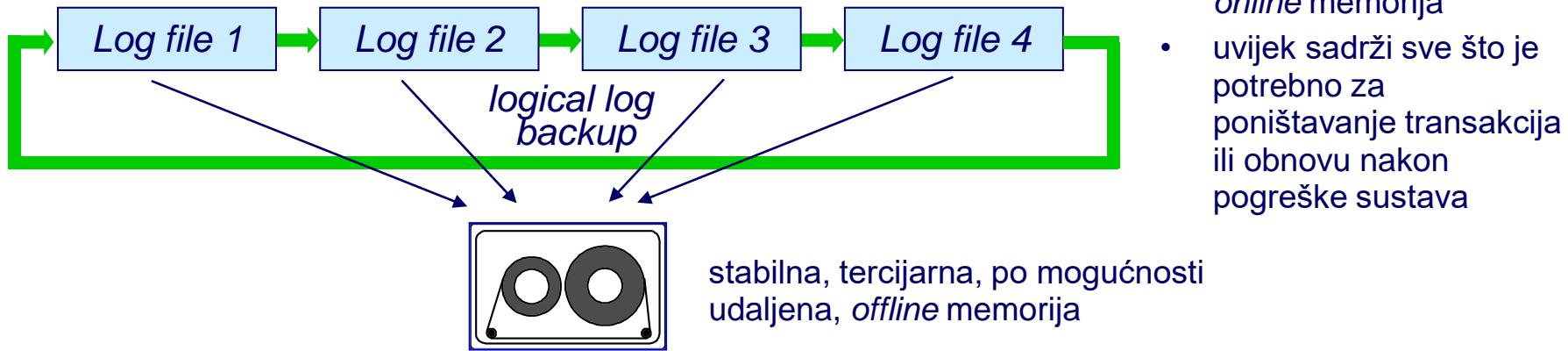
## IBM Informix

- prostor za pohranu dnevnika (*logical log*) smješten je u jedan od prostora baze podataka (npr. *rootdbspace*), u stabilnoj (*online*) memoriji
- prostor za pohranu dnevnika je podijeljen u zasebne cjeline koje se nazivaju datoteke dnevnika (*logical log file*). Oracle: *redo log group*
- sustav posjeduje najmanje tri datoteke dnevnika, ali se najčešće koristi veći broj, npr. 50 ili 500
- datoteke dnevnika se ciklički izmjenjuju: kad se jedna datoteka dnevnika popuni, zapisi dnevnika se počinju upisivati u sljedeću. Kad se popuni posljednja datoteka dnevnika u nizu, ponovo se koristi prva



# Upravljanje logičkim dnevnicima

- čim se popuni, sadržaj datoteke dnevnika se može arhivirati (*logical log backup*)
  - na traku, optički medij
  - na udaljeni sustav za pohranu podataka (*remote backup site*)



Arhiviranjem dnevnika:

- sadržaj dnevnika se (dodatno) štiti od pogreške medija
  - jer dnevnik je vrlo važan za mogućnost obnove nakon pogreške medija (kasnije)
- omogućava se ponovno korištenje prostora za pohranu dnevnika
- može li se prostor datoteke dnevnika koja je arhivirana odmah početi puniti sa zapisima nove datoteke dnevnika?
  - **ne uvijek!**

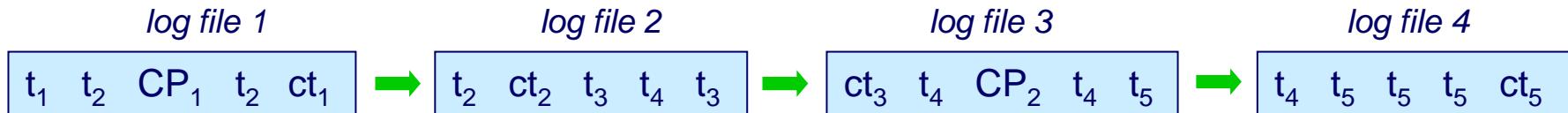
# Koliko dugo čuvati dnevnik u dostupnoj stabilnoj memoriji

---

- zapis logičkih dnevnika koji se odnose na trenutno aktivne transakcije potrebni su za poništavanje neispravne (*failed*) transakcije (bilo zbog pogreške transakcije, bilo zbog izvršene naredbe ROLLBACK), a u postupku obnove nakon pogreške sustava nužni su svi zapisi dnevnika o transakcijama koje su bile aktivne tijekom posljednje kontrolne točke ili kasnije
- stoga se u uvijek dostupnoj (*online*), stabilnoj sekundarnoj memoriji moraju čuvati:
  - datoteke dnevnika koje još uvijek nisu arhivirane i sve datoteke dnevnika koje slijede iza njih
  - datoteka dnevnika koja sadrži posljednju kontrolnu točku i sve datoteke dnevnika koje slijede iza nje
  - datoteke dnevnika koje sadrže zapise transakcija koje su bile aktivne u trenutku posljednje kontrolne točke i sve datoteke dnevnika koje slijede iza njih

# Koliko dugo čuvati dnevnik u dostupnoj stabilnoj memoriji

Primjer:



- t<sub>i</sub> je oznaka za zapis dnevnika koji pripada transakciji T<sub>i</sub>
- ct<sub>i</sub> je oznaka zapisa dnevnika <commit T<sub>i</sub>>
- CP je oznaka kontrolne točke
- uz pretpostavku da su sve prikazane datoteke dnevnika arhivirane
  - datoteka dnevnika 3. (i sve datoteke koje slijede) se mora čuvati jer datoteka dnevnika 3. sadrži posljednju kontrolnu točku
  - datoteka dnevnika 2. (i sve datoteke koje slijede) se mora čuvati jer sadrži zapise transakcije T<sub>4</sub> koja je bila aktivna tijekom posljednje kontrolne točke

# Problem dugih transakcija

---

- transakcije koje relativno dugo traju ili produciraju veliki broj zapisa dnevnika mogu uzrokovati stanje u kojem na raspolaganju nema slobodnih datoteka dnevnika
  - ⇒ blokiranje sustava
  - npr. ako transakcija  $T_4$  iz prethodnog primjera ne završi prije nego na red za punjenje dođe datoteka *log file 2*, sustav prestaje zaprimati nove transakcije

# Problem dugih transakcija

---

## IBM Informix - različiti načini sprječavanja blokiranja sustava

1. sustav se može konfigurirati tako da se u slučaju potrebe omogući automatsko ili ručno dodavanje datoteka dnevnika
  - potrebno je unaprijed predvidjeti prostor u stabilnoj memoriji
- ili
2. DBA definira granične vrijednosti ukupne popunjenoosti dnevnika (LTXHWM i LTXEHWM)
  - prvi znak za opasnost (*LTXHWM-Long Transaction High Water Mark*): npr. nakon što je 70% ukupne veličine dnevnika popunjeno, sustav prestaje prihvatići nove transakcije
  - drugi znak za opasnost (*LTXEHWM-Long Transaction Exclusive access High Water Mark*): npr. nakon što je 90% ukupne veličine dnevnika popunjeno, sustav počinje poništavati sve aktivne transakcije

# Kako ispravno odrediti veličinu i broj datoteka dnevnika

## IBM Informix

- LOGSIZE: veličina datoteke dnevnika
- LOGFILES: broj datoteka dnevnika
- pri određivanju ukupne veličine dnevnika trebalo bi uzeti u obzir:
  - tipično vrijeme trajanja transakcija (prevladavaju li konverzacijalne transakcije, *online* transakcije ili *batch* transakcije)
  - broj operacija koje transakcije obavljaju
  - vrstu operacija koje transakcije obavljaju (čitanje ili pisanje)
  - opterećenje sustava (može produljiti vrijeme odziva, odnosno trajanje transakcija)

$$\Rightarrow \text{ukupna veličina dnevnika} = \text{LOGSIZE} \times \text{LOGFILES}$$

*It is difficult to predict how much logical-log space your database server system requires until the system is fully in use (IBM Informix Dynamic Server Performance Guide )*

- pri određivanju veličine datoteke dnevnika treba uzeti u obzir:
  - veće datoteke dnevnika pružaju nešto bolje performanse
    - vrijedi li riskirati?
  - koliko transakcija smije biti izgubljeno u slučaju kvara medija na koji se pohranjuju datoteke dnevnika?
    - male datoteke dnevnika će biti brže arhivirane

# Obnova nakon pogreške medija

---

## Pogreška medija

- izgubljen je sadržaj baze podataka u postojanoj memoriji
- možda je izgubljen čak i dnevnik u *online* stabilnoj memoriji
- transakcije koje su bile u tijeku su prekinute (kao i kod pogreške sustava)

## Rješenje:

- periodička izrada redundantnih kopija (arhiva) sadržaja postojane memorije
- neprekidno arhiviranje stabilne *online* memorije (dnevnik iz *online* stabilne memorije čim je prije moguće kopirati u *offline*, po mogućnosti udaljenu, stabilnu memoriju)

# Obnova nakon pogreške medija

## Smještaj redundantnih kopija:

- nije važan broj izrađenih kopija, nego mjesto na kojem se te kopije čuvaju!
  - kopija se moraju čuvati na mediju s nezavisnim modalitetom kvarova (*independant failure mode*). Mediji imaju nezavisan modalitet kvarova ako niti jedan zasebni događaj (predviđen u okviru razumnih pretpostavki) neće uzrokovati istovremeni kvar na oba medija

događaj	primjer uređaja s nezavisnim modalitetom kvarova
kvar diska	drugi disk ili jedinica trake u istom računalu
požar u prostoriji	disk u računalu u susjednoj prostoriji
požar u zgradi	disk ili jedinica trake u računalu u susjednoj zgradi
katastrofalan potres	disk jedinica u drugom, dovoljno udaljenom gradu

# Arhiviranje baze podataka

---

- periodičko kopiranje cijelog sadržaja baze podataka (*database dump*, *database backup*) na uređaj s nezavisnim modalitetom kvarova
- u slučaju kvara medija, baza podataka se uz pomoć te kopije može vratiti u stanje u kojem je bila **u trenutku izrade posljednje arhive**

## Problemi i rješenja:

- potreba zaustavljanja sustava za vrijeme izrade arhive (*quiescent archiving*)
  - *non-quiescent archiving* (IBM Informix: *online backup*, Oracle 11g: *hot backup*)
- postupak arhiviranja *cijele* baze podataka je dugotrajan i zahtijeva puno prostora; obnova u posljednje konzistentno stanje pomoći svih dnevnika nastalih prije davno napravljene arhive može (pre)dugo trajati
  - inkrementalno arhiviranje

# Arhiviranje datoteke dnevnika

---

- za obnovu nakon pogreške medija potrebna je
  - posljednja arhiva
  - sve datoteke dnevnika koje su nastale nakon početka izrade posljednje arhive
  - što ako se neke datoteke dnevnika nisu stigle arhivirati prije pogreške medija?
    1. slučaj kada je *online* stabilna memorija ostala sačuvana - prije obnove provesti "*log salvage*"
    2. slučaj kada je i *online* stabilna memorija uništена
- preporuča se čuvati barem tri posljednje arhive i sve pripadne datoteke dnevnika
  - u slučaju gubitka posljednje arhive (npr. *tape read error*), moguća je obnova uz pomoć prethodne arhive i svih dnevnika nastalih nakon te arhive (vidjeti ilustraciju na sljedećoj stranici)

# Arhiviranje datoteka dnevnika

Primjer:



- obnova sustava je moguća pomoću:
  - arhiva 29.1. + datoteke dnevnika 2719 - 2720 ili
  - arhiva 22.1. + datoteke dnevnika 1253 - 2720 ili
  - arhiva 15.1. + datoteke dnevnika 115 - 2720

# Inkrementalno arhiviranje baze podataka

- arhiva razine 0 (*full backup*): arhiviranje **svih** blokova podataka
- **problem:** ako rijetko izradujemo arhivu - moguće je da će trebati obraditi veliki broj zapisa dnevnika nakon pogreške medija
- **problem:** ako često izrađujemo arhivu razine 0 smanjili bismo količinu zapisa dnevnika koje treba obraditi, ali doveli do problema utroška prostora i vremena potrebnog za pohranu arhive (performanse sustava)
- **rješenje:** česta izrada arhive samo onih blokova podataka koji su promijenjeni nakon posljednje arhive
  - arhiviranje na više razina
    - arhiva razine *i*: arhiviranje blokova podataka koji su promijenjeni nakon posljednje arhive razine *i - 1*

Inkrementalno arhiviranje NIJE zamjena za arhiviranje dnevnika! Inkrementalnim arhiviranjem se samo smanjuju broj zapisa dnevnika koje treba primijeniti tijekom obnove, a da se istovremeno izbjegne potreba za čestim izradama arhive razine 0.

# Inkrementalno arhiviranje baze podataka

---

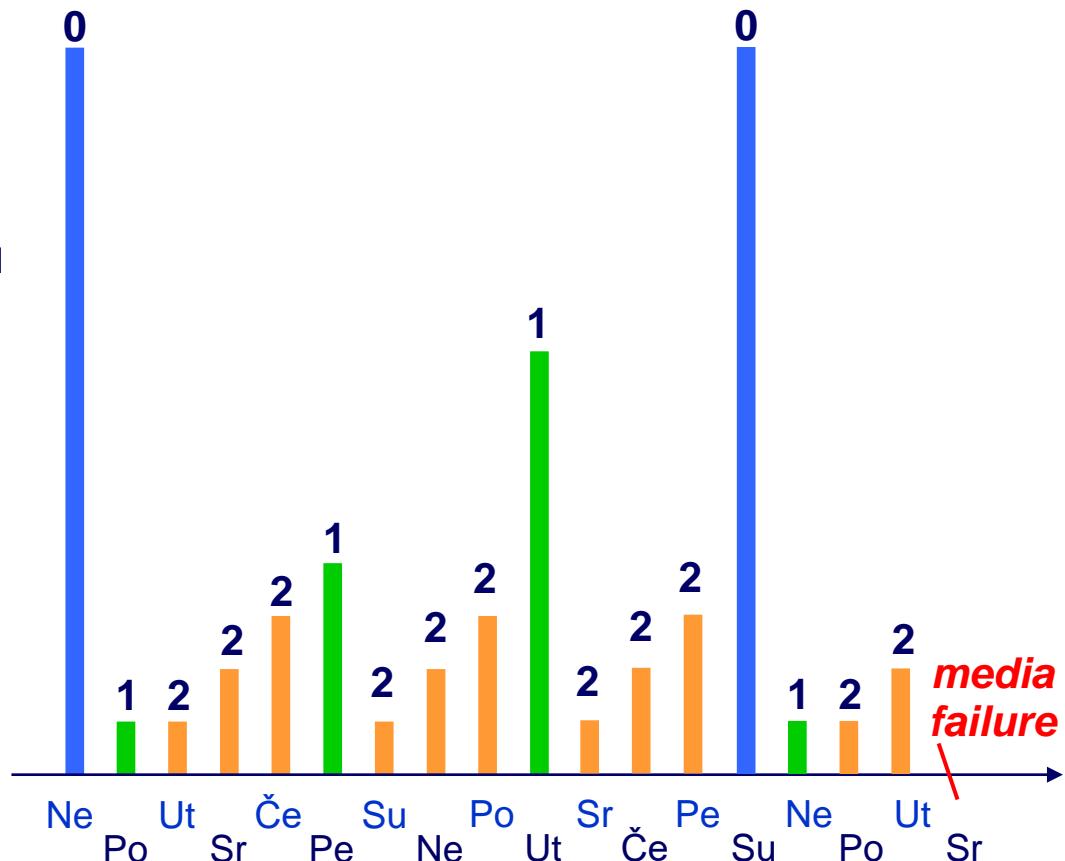
## IBM Informix

- arhiva razine 0: *full backup*
- arhiva razine 1: arhiviraju se samo blokovi podataka koji su promijenjeni nakon posljednje arhive razine 0
- arhiva razine 2: arhiviraju se samo blokovi podataka koji su promijenjeni nakon posljednje arhive razine 1

# Inkrementalno arhiviranje baze podataka

## Primjer: IBM Informix

- visinom stupca ilustrirana je veličina arhive. Boja stupca i broj na vrhu stupca označavaju razinu arhive
- slika je nacrtana s prepostavkom visoke volatilnosti baze podataka i jednolike distribucije količine blokova koji se mijenjaju: u svakom vremenskom intervalu promijenjen je jednak broj novih blokova



- obnova nakon pogreške medija u trenutku označenom na slici moguća je pomoću:
  - zadnje arhive razine 0 + zadnje arhive razine 1 + zadnje arhive razine 2 + datoteka dnevnika nastalih nakon početka izrade zadnje arhive razine 2

# Inkrementalno arhiviranje baze podataka

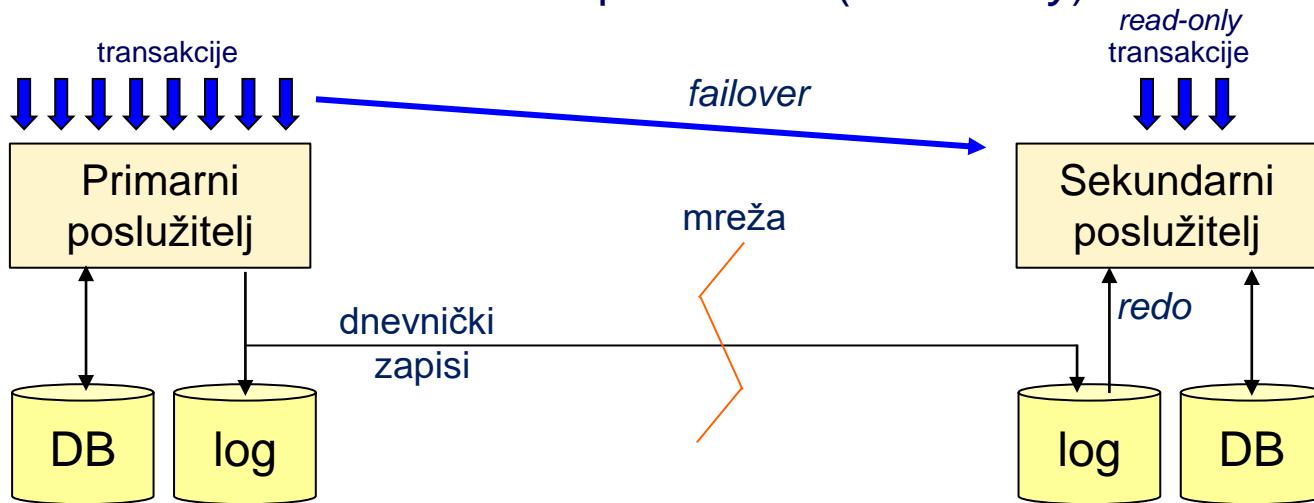
---

- pri definiranju sheme arhiviranja u obzir treba uzeti karakterističnu periodičku raspodjelu opterećenosti sustava (ako takva postoji). Mnogi sustavi (**ne svi**) su nedjeljom manje opterećeni u odnosu na druge dane u tjednu, te manje opterećeni noću u odnosu na druge dijelove dana. Ovdje su prikazani primjeri nekih od shema arhiviranja zasnovanih na toj pretpostavci:

- Shema 1:**
- arhiva razine 0: svake četvrte nedjelje
  - arhiva razine 1: svake nedjelje
  - arhiva razine 2: tijekom noćnih sati svakog dana osim nedjelje
  - arhiviranje datoteka dnevnika: kontinuirano
- Shema 2:**
- arhiva razine 0: svake nedjelje
  - arhiva razine 1: svake noći osim nedjelje
  - arhiviranje datoteka dnevnika: kontinuirano
- Shema 3:**
- arhiva razine 0: svake nedjelje
  - arhiviranje datoteka dnevnika: kontinuirano

# Postizanje vrlo visoke razine raspoloživosti

- arhitektura za vrlo visoku razinu raspoloživosti (*availability*)



- transakcije se obavljaju na primarnom poslužitelju (*primary server/site*)
- dnevnički zapisi se sinkrono ili asinkrono dostavljaju i u dnevnik sekundarnog poslužitelja
- na sekundarnom poslužitelju (*secondary server/site*) se odmah ili s odgodom obavlja *redo* potvrđenih transakcija
  - sekundarni poslužitelj može se koristiti kao *read-only server*
- u slučaju kvara primarnog poslužitelja, korisničke sjednice se preusmjeravaju na dotadašnji sekundarni poslužitelj (*failover*)
- *IBM Informix: HDR (High-availability Data Replication)*
- *SQL Server: Database Mirroring; Log Shipping*

# Osigurati svojstvo izdržljivosti transakcije!

---

- ne dopustiti gubitak efekata potvrđenih transakcija. Potrebno je:
  - spriječiti gubitak sadržaja baze podataka (kao posljedice pogreške medija)
    - arhivirati bazu podataka na medij s nezavisnim modalitetom kvarova
  - spriječiti gubitak sadržaja dnevnika (kao posljedice pogreške medija)
    - za pohranu datoteka dnevnika koristiti kvalitetnu *online* stabilnu memoriju, a arhivirati ih na medij s nezavisnim modalitetom kvarova
    - ne koristiti prevelike pojedinačne datoteke dnevnika
- OPREZ: magnetski diskovi, trake, DVD, SSD - retencija 2-10 godina, ovisno o mikroklimatskim uvjetima u kojima se čuvaju: temperatura, vlaga, svjetlost

# Literatura:

---

- H. Garcia-Molina, J. Ullman, J. Widom. Database system implementation. Prentice-Hall International, Inc. Upper Saddle River, New Jersey. 2000.
- R. Elmasri, S. Navathe. Fundamentals of database systems, 3rd ed. Addison-Wesley Publishing Company. Reading, Massachusetts. 2000.
- A. Silberschatz, H. Korth, S. Sudarshan. Database system concepts, 6th ed. McGraw-Hill. 2010.
- P. Bernstein, V. Hadzilacos, N. Goodman. Concurrency control and recovery in database systems. Addison-Wesley Publishing Company. Reading, Massachusetts. 1987.
- IBM Informix Backup and Restore Guide, Version 11.70, Informix Product Library, 2011.
- Oracle Database Backup and Recovery Advanced User's Guide 11g Release 2 (11.2)