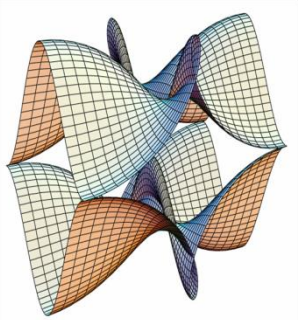
Sveučilište u Zagrebu
PMF – Matematički odsjek

BAZE PODATAKA
Predavanja 2019/2020



Poglavlje 6: Fizičko oblikovanje i implementacija baze podataka

Sastavio: Robert Manger
27.04.2020

Općenito o fizičkom oblikovanju i implementaciji (1)

- Poglavlje je posvećeno trećoj fazi oblikovanja baze podataka, a to je *fizičko oblikovanje*.
- Glavni cilj te faze je stvoriti *fizičku shemu* baze, dakle opis njezine fizičke građe.
  - Fizička shema zapravo je tekst sastavljen od naredbi u SQL-u ili nekom drugom jeziku.
  - Izvođenjem tih naredbi DBMS automatski stvara fizičku bazu.

Općenito o fizičkom oblikovanju i implementaciji (2)

- U ovom poglavlju također ćemo opisati načine na koji je baza podataka fizički realizirana.
- Pritom nas zanima i statički i dinamički aspekt:
  - fizička građa baze,
  - algoritmi koji omogućuju rad s pohranjenim podacima.
- Ova znanja projektantu baze nisu neophodna, ali su mu korisna.

Sadržaj Poglavlja 6

Općenito o fizičkoj građi

- Fizička baza podataka gradi se od datoteka i indeksa pohranjenih na disku.
- U ovom potpoglavlju
  - proučavamo elemente fizičke građe:
    - blokove,
    - zapise,
    - pokazivače,
  - objašnjavamo kako se ti elementi organiziraju u datoteke i indekse.
- To nam omogućuje da razumijemo način na koji DBMS fizičku shemu pretvara u fizičku bazu.

Elementi fizičke građe (1)

- Vanjska memorija podijeljena je u jednako velike *blokove*.
  - Veličina bloka je konstanta operacijskog sustava.
  - Svaki blok jednoznačno je zadan svojom *adresom*.
- Osnovna operacija s vanjskom memorijom je prijenos bloka iz vanjske memorije u glavnu ili obratno.
  - Blok je najmanja količina podataka koja se može prenijeti.
  - Vrijeme potrebno za prijenos bloka neusporedivo je veće od vremena potrebnog za bilo koju radnju u glavnoj memoriji.

Elementi fizičke građe (2)

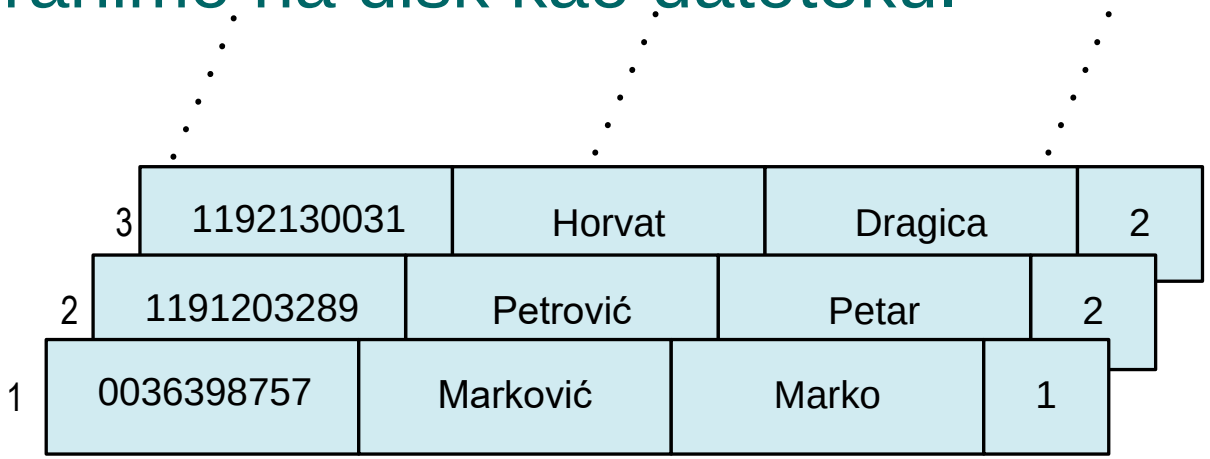
- *Datoteka* je konačni niz *zapisa* (*slogova*) istog tipa pohranjenih u vanjskoj memoriji.
  - Tip zapisa je uređena *n*-torka osnovnih podataka.
  - Svaki osnovni podatak opisan svojim imenom i tipom.
  - Sam zapis sastoji se od konkretnih vrijednosti osnovnih podataka.
  - Zapisi su fiksne duljine.
- Tipične operacije nad datotekom su:
  - ubacivanje novog zapisa,
  - promjena postojećeg zapisa,
  - izbacivanje zapisa,
  - pronalaženje zapisa gdje zadani osnovni podaci imaju zadane vrijednosti.

Elementi fizičke građe (3)

- Jedna datoteka obično služi za fizičko prikazivanje jedne relacije iz relacijske baze.
- Da bismo fizički prikazali relaciju

  STUDENT (JMBAG, PREZIME, IME, GODINA STUDIJA)

  svaku njezinu *n*-torku pretvaramo u zapis, te zapise poredamo u nekom redoslijedu, pa ih pohranimo na disk kao datoteku.



Elementi fizičke građe (4)

- Kod pretvorbe relacije STUDENT u datoteku morali smo uvesti brojne fizičke detalje:
  - točnu duljinu pojedinog podatka u byte-ovima,
  - redoslijed podataka unutar zapisa,
  - međusobni redoslijed zapisa ...
- U nekoj datoteci *kandidat za ključ* je osnovni podatak ili kombinacija podataka, čija vrijednost jednoznačno određuje zapis unutar datoteke.
  - Ukoliko ima više kandidata za ključ, tada odabiremo jednog od njih da bude *primarni* ključ.
  - Datoteka ne mora imati ključ, jer mogu postojati zapisi-duplikati.

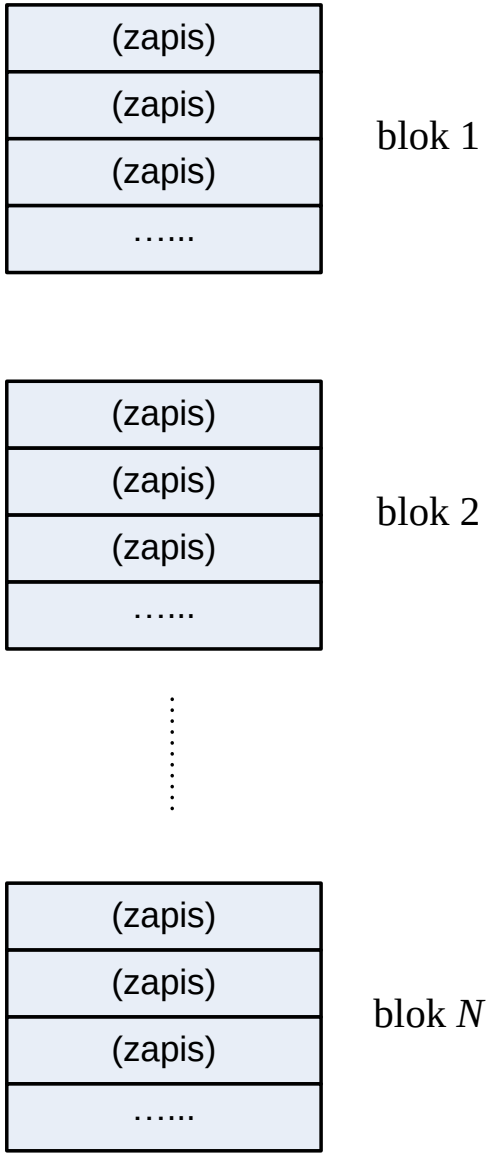
Elementi fizičke građe (5)

- Opisujemo kako se zapisi koji čine datoteku pohranjuju u vanjskoj memoriji.
  - Zapisi se moraju rasporediti po blokovima.
  - Više zapisa sprema se u jedan blok.
  - U jednom bloku smješten je cijeli broj zapisa.
  - Ni jedan zapis ne prelazi granicu između dva bloka.
  - Dio bloka možda ostaje neiskorišten.
  - Moguće je jednoznačno odrediti položaj zapisa na disku.

- *Adresa zapisa* je uređeni par adrese bloka i pomaka u byte-ovima unutar bloka.

Elementi fizičke građe (6)

- Cijela datoteka obično zauzima više blokova.
  - Položaj i redoslijed tih blokova određen je posebnim pravilima koja čine takozvanu *organizaciju* datoteke.
  - Blokovi se ne moraju nalaziti na uzastopnim adresama na disku.



Elementi fizičke građe (7)

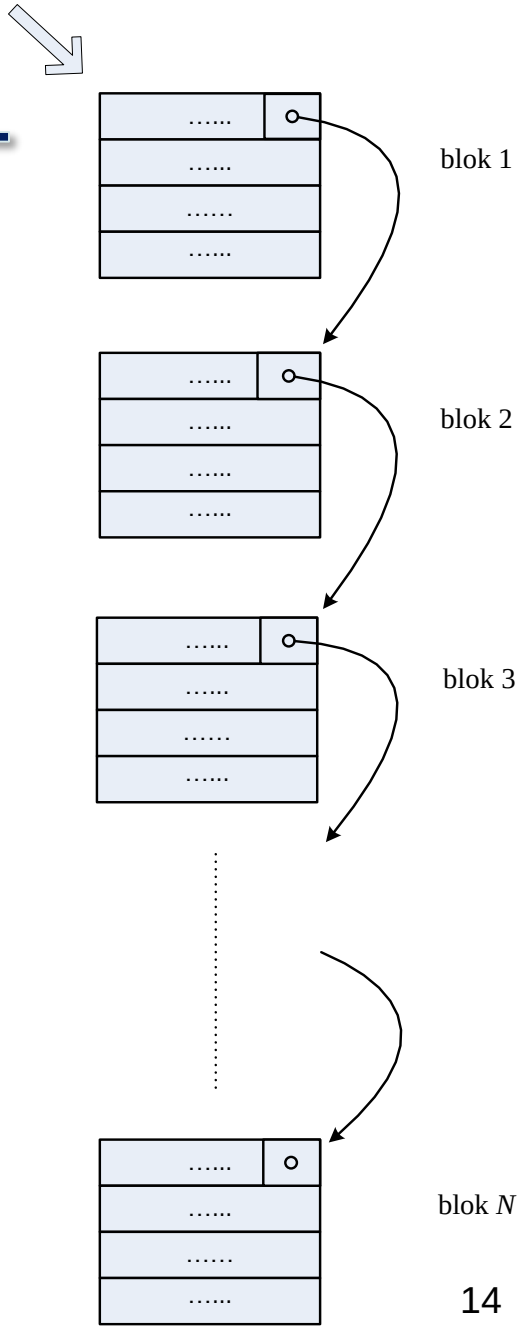
- *Pokazivač* (*pointer*) je podatak unutar zapisa ili bloka jedne datoteke koji pokazuje na neki drugi zapis ili blok u istoj ili drugoj datoteci.
- Pokazivač se realizira kao:
  - adresa zapisa ili bloka kojeg treba pokazati (*fizički pokazivač*).
  - vrijednost primarnog ključa zapisa kojeg treba pokazati (*logički pokazivač*).
- Pokazivači omogućuju:
  - uspostavljanje veza između zapisa ili blokova,
  - povezivanje dijelova datoteke u cjelinu,
  - pristup iz jednog dijela te cjeline u drugi.

Organizacija datoteke (1)

- Organizacija datoteke:
  - Skup pravila koja određuju sadržaj, raspored te način međusobnog povezivanja blokova iz iste datoteke.
- Opisat ćemo nekoliko najvažnijih organizacija.
  - Svaka od njih ima svoje prednosti i nedostatke u pogledu efikasnog obavljanja osnovnih operacija nad datotekom.

Organizacija datoteke (2)

- **Jednostavna datoteka.**
  - Zapisi su poredani u onoliko blokova koliko je potrebno.
  - Ti blokovi su povezani u vezanu listu.
  - Dakle svaki blok sadrži fizički pokazivač na idući blok.
  - Kao adresu cijele datoteke pamtimo adresu prvog bloka.



Organizacija datoteke (3)

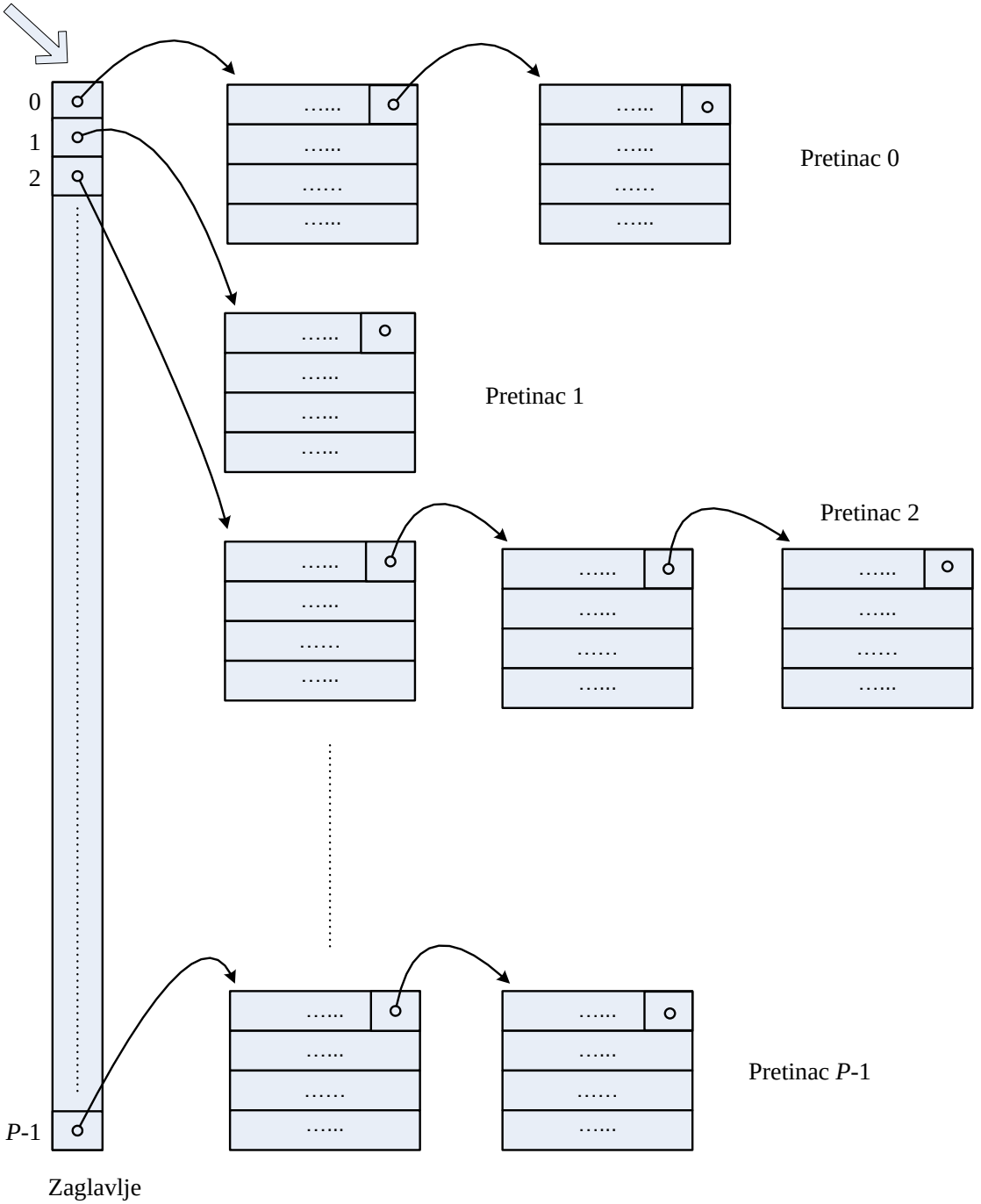
- Prednost jednostavne organizacije je da se kod nje lagano ubacuju, izbacuju i mijenjaju zapisi.
- Nedostatak jednostavne organizacije je da bilo kakvo traženje zahtijeva sekvencijalno čitanje blok po blok.
  - Jedno traženje u prosjeku zahtijeva čitanje pola datoteke, pa vrijeme traženja linearno raste s veličinom datoteke.

Organizacija datoteke (4)

- ***Hash* datoteka.**
  - Zapisi su raspoređeni u *P* *pretinaca* (*buckets*), označenih rednim brojevima 0, 1, 2, … , *P*-1.
  - Svaki pretinac građen je kao vezana lista blokova.
  - Zadana je *hash funkcija* *h*( ) – ona daje redni broj *h*(*k*) pretinca u kojeg treba spremiti zapis s vrijednošću ključa *k*.
  - Ista funkcija kasnije omogućuje i pronalaženje zapisa sa zadanom vrijednošću ključa.

Organizacija datoteke (5)

- Fizički pokazivači na početke pretinaca čine zaglavlje koje se smješta u prvi blok datoteke.
- Adresu zaglavlja pamtimo kao adresu datoteke.
- Zaglavlje se za vrijeme rada može držati u glavnoj memoriji.



Organizacija datoteke (6)

- Važno je da *h*( ) uniformno (jednoliko) distribuira vrijednosti ključa na pretince.
- Prednost *hash* organizacije je što ona omogućuje gotovo izravni pristup na osnovi ključa.
  - Da bismo pronašli zapis s vrijednošću ključa *k*, najprije računamo *h*(*k*), te pretražimo samo *h*(*k*)-ti pretinac.
  - Ako je *h*() uniformna, te ako je broj pretinaca *P* dobro odabran, tada ni jedan od pretinaca nije suviše velik.
  - Pristup zahtijeva svega nekoliko čitanja blokova.
- Nedostatak *hash* datoteke je da ona ne čuva sortirani redoslijed po ključu.
  - *Hash* funkcija razbacuje podatke na kvazi-slučajan način.
  - Teško je pronaći zapise gdje je vrijednost ključa u nekom intervalu.

Organizacija datoteke (7)

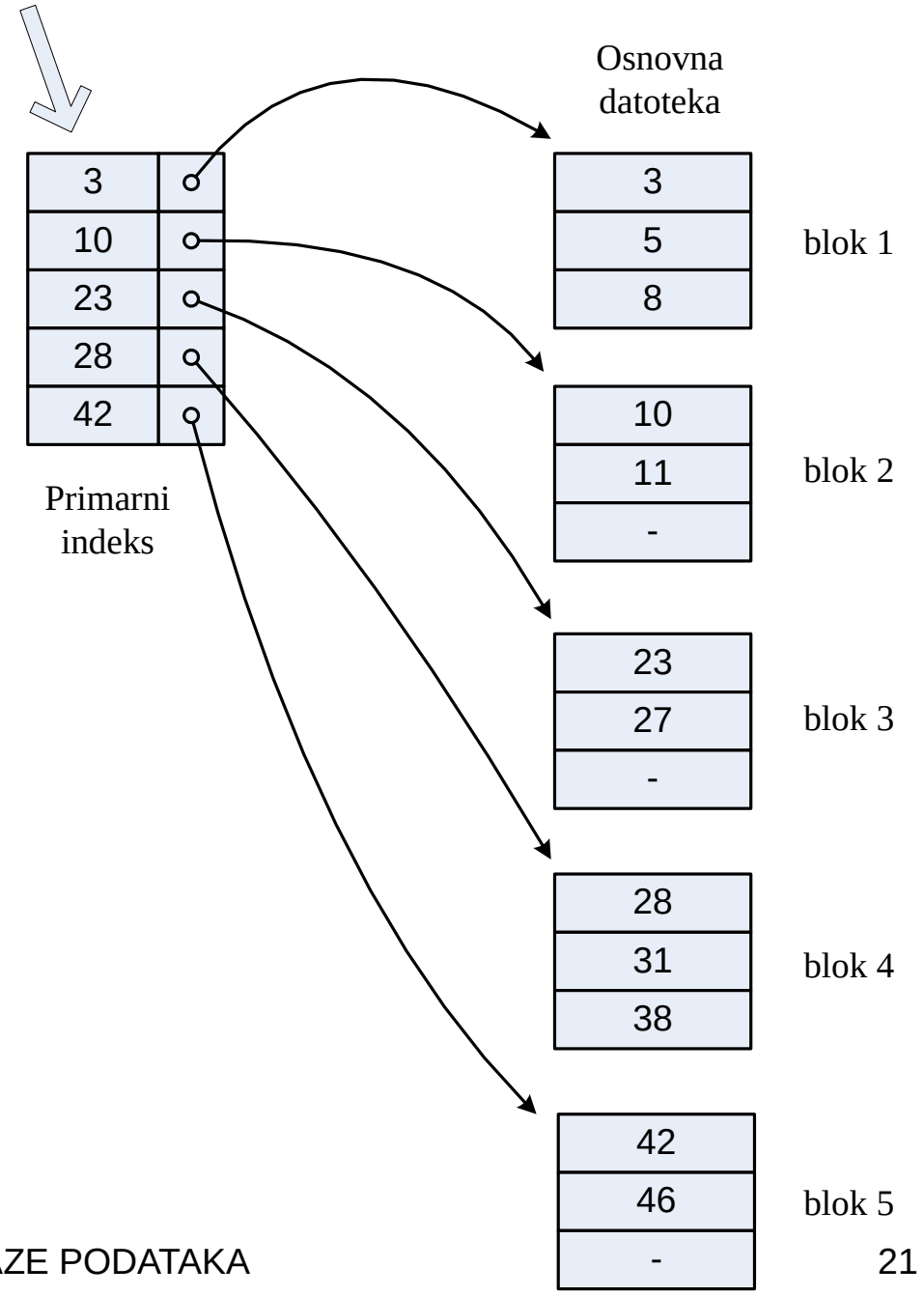
- *Indeks* je pomoćna datoteka koja olakšava traženje zapisa u osnovnoj datoteci.
- Indeks koji omogućuje traženje po primarnom ključu naziva se *primarni indeks*.
  - Zapisi su parovi (k, p), gdje je k vrijednost ključa, a p je fizički pokazivač na zapis u osnovnoj datoteci.
  - Za zadanu vrijednost k postoji najviše jedan par (k, p).
- Indeks koji omogućuje traženje po podatku koji nije ključ naziva se *sekundarni indeks*.
  - Zapisi su parovi oblika (v, p), gdje je v vrijednost podatka, a p je fizički ili logički pokazivač na zapis u osnovnoj datoteci.
  - Može postojati više parova s istim v, dakle (v, p_1), (v, p_2), (v, p_3), ….

Organizacija datoteke (7)

- **Indeks-sekvencijalna datoteka**
  - Sastoji se od jednostavno organizirane osnovne datoteke i od primarnog indeksa.
  - Ako je osnovna datoteka uzlazno sortirana po ključu, tada primarni indeks može biti *razrijeđen*.
  - Dakle dovoljno je da indeks ima adrese blokova i najmanju vrijednost ključa za svaki blok.
  - Kao adresu cijele datoteke pamtimo adresu indeksa.
- Prednost indeks-sekvencijalne organizacije je da ona omogućuje brz pristup na osnovi ključa, makar ipak malo sporiji nego *hash* datoteka.

Organizacija datoteke (8)

- Daljnja prednost je mogućnost čitanja osnovne datoteke sortirano po ključu.
- Nedostatak je da se znatno kompliciraju operacije ažuriranja podataka. Također, troši se dodatni prostor na disku.



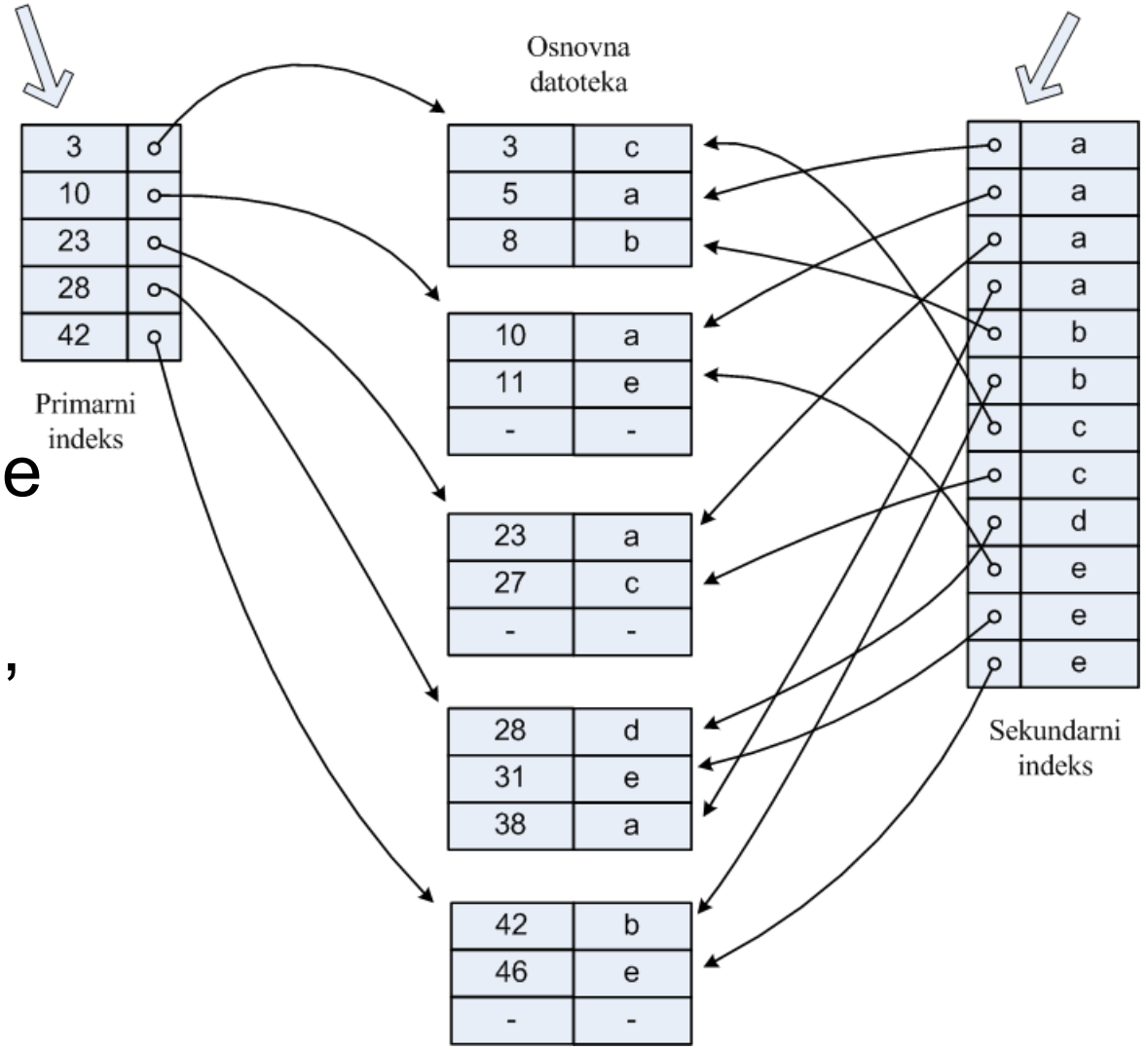
Organizacija datoteke (9)

- **Invertirana datoteka**
  - I dalje imamo osnovnu datoteku i primarni indeks, no dodan je barem jedan sekundarni indeks.
  - Sekundarni indeks omogućuje da se ista osnovna datoteka, osim po primarnom ključu, pretražuje i po nekom drugom podatku.
  - Sekundarni indeks je uvijek gust, dakle on sadrži pokazivač na svaki zapis iz osnovne datoteke.
  - Kao adresu cijele datoteke pamtimo ili adresu primarnog ili adresu sekundarnog indeksa.

- Prednost invertirane organizacije je brz pristup po više kriterija, te mogućnost sortiranog ispisa ili intervalnog pretraživanja po tim kriterijima.

Organizacija datoteke (10)

- Nedostatak invertirane organizacije je da se ažuriranje podataka još više komplicira, te se troši još više dodatnog prostora na disku.



Organizacija datoteke (11)

- **Hash datoteka s podijeljenom hash funkcijom.**
  - Poopćenje prije opisane *hash* datoteke.
  - Cilj koji se želi postići je mogućnost traženja ne samo po primarnom ključu nego i po drugim podacima.
  - Građa izgleda isto kao za običnu *hash* datoteku.
  - No *hash* funkcija definirana je općenitije, tako da osim ključa uzima kao argumente i ne-ključne podatke.

- Pretpostavimo da:
  - U datoteci želimo pronaći zapise gdje istovremeno podatak A_1 ima vrijednost v_1, podatak A_2 ima vrijednost v_2, ..., podatak A_r ima vrijednost v_r.

Organizacija datoteke (12)

  - Redni broj pretinca u *hash* tablici je niz od B bitova. Znači da je broj pretinaca $P = 2^B$.

- Postupamo na sljedeći način.
  - Podijelimo B bitova u skupine: b_1 bitova za podatak A_1, b_2 bitova za podatak A_2, ..., b_r bitova za podatak A_r.
  - Zadamo „male“ *hash* funkcije $h_i(v_i)$, i=1,2, ..., r, gdje h_i preslikava vrijednost v_i podatka A_i u niz od b_i bitova.
  - Redni broj pretinca za zapis u kojem je A_1=v_1, A_2=v_2, ..., A_r=v_r zadaje se spajanjem malih nizova bitova.
  - Dakle: $h(v_1,v_2, ..., v_r) = h_1(v_1) \mid h_2(v_2) \mid ... \mid h_r(v_r)$.

    Ovdje je | oznaka za „lijepljenje“ nizova bitova.

Organizacija datoteke (13)

- Promatramo zapise o studentima fakulteta. Želimo ih spremati u *hash* datoteku s 1024 (=2^{10}) pretinaca.
  - Podijelimo 10 bitova u rednom broju pretinca: 5 bitova za JMBAG, 3 za PREZIME, 2 za GODINU_STUDIJA.
  - Zadajemo male *hash* funkcije, gdje su v_1, v_2, v_3 vrijednosti za JMBAG, PREZIME i GODINU_STUDIJA:
    $h_1(v_1) = v_1$ % 32 (ostatak kod dijeljenja s 32) ,
    $h_2(v_2)$ = (broj znakova u v_2 različitih od bjeline) % 8,
    $h_3(v_3) = v_3$ % 4 .
- Tada redni broj pretinca za zapis
  (1192130031, Horvat, Dragica, 2)
  iznosi $(01111|110|10)_2 = (506)_{10}$.

Organizacija datoteke (14)

- Da bismo pronašli zapis (ili više njih) u kojem je $A_1=v_1$, $A_2=v_2$, ..., $A_r=v_r$, računamo redni broj pretinca $h(v_1,v_2, ...,v_r)$ i sekvencijalno pretražimo taj jedan pretinac.
- Ako u upitu nije fiksirana vrijednost podatka A_i, tada b_i bitova u rednom broju pretinca ostaje nepoznato, a broj pretinaca koje moramo pretražiti povećava se 2^{bi} puta.
- Ako tražimo sve studente na drugoj godini, tada su nam u rednom broju pretinca poznata zadnja 2 bita: 10, no prvih 8 bitova je nepoznato. Treba pretražiti 2^8=256 pretinaca, dakle 1/4 datoteke.

Organizacija datoteke (15)

- Prednost podijeljene *hash* funkcije u odnosu na invertiranu datoteku je da se ne troši dodatni prostor za indekse.
- Također, operacije ubacivanja, izbacivanja i promjene zapisa znatno su jednostavnije.
- Nedostatak je da traženje zapisa u kojem su specificirane vrijednosti samo nekih od podataka traje dulje nego kod invertirane organizacije.

- Organizacija pomoću podijeljene *hash* funkcije dobra je za datoteke koje nisu prevelike i čiji sadržaj se često mijenja.

Organizacija indeksa (1)

- Uobičajeni način fizičkog prikazivanja indeksa je pomoću B-stabla.
- Riječ je o hijerarhijskoj strukturi koja omogućuje da indeks efikasno obavlja svoje osnovne zadaće:
  - brzo pronalaženje zadane vrijednosti podatka,
  - čuvanje sortiranog redoslijeda svih vrijednosti.

- *B-stablo* reda m je m-narno stablo sa svojstvima:
  - korijen je ili list ili ima bar dvoje djece;
  - svaki čvor, izuzev korijena i listova, ima između $\lceil m/2 \rceil$ i m djece;
  - svi putovi od korijena do lista imaju istu duljinu.

Organizacija indeksa (2)

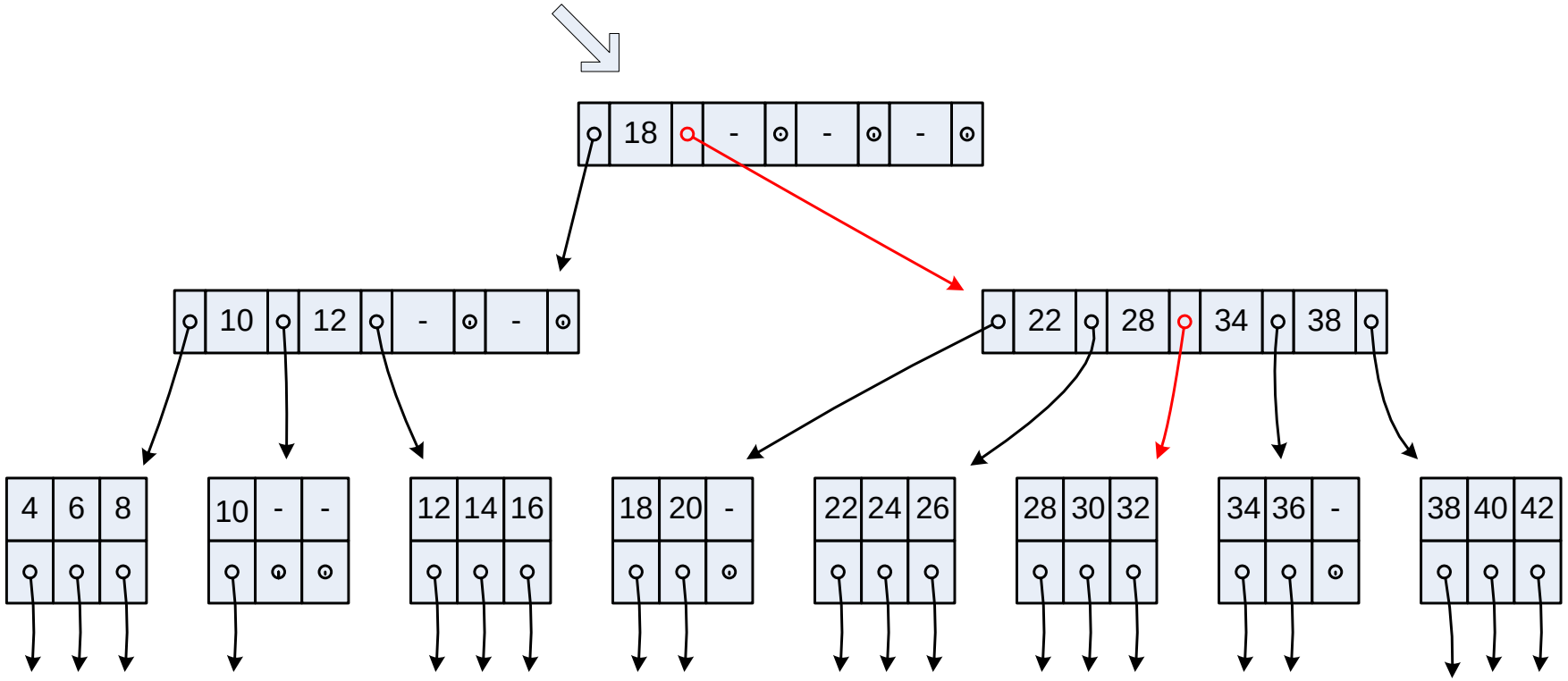
- Detaljno opisujemo prikaz gustog primarnog indeksa pomoću B-stabla.
  - Prikaz ostalih vrsta indeksa je vrlo sličan.
- Gusti primarni indeks prikazuje se kao B-stablo sagrađeno od blokova vanjske memorije, i to tako da jedan čvor bude jedan blok.
- Veza između roditelja i djeteta realizira se tako da u bloku-roditelju piše fizički pokazivač na blok-dijete.
- Sadržaj unutrašnjih čvorova i listova izgleda ovako:

Organizacija indeksa (3)

- Unutrašnji čvor ima sadržaj oblika (p_0, k_1, p_1, k_2, p_2, … , k_r, p_r), gdje je p_i pokazivač na i-to dijete dotičnog čvora ($0 \le i \le r$), k_i je vrijednost ključa ($1 \le i \le r$).
  - Vrijednosti ključa unutar čvora su sortirane: $k_1 \le k_2 \le \ldots \le k_r$.
  - Sve vrijednosti ključa u pod-stablu koje pokazuje p_0 su $< k_1$.
  - Za $1 \le i < r$, sve vrijednosti ključa u pod-stablu kojeg pokazuje p_i su u poluotvorenom intervalu $[k_i, k_{i+1})$.
  - Sve vrijednosti ključa u pod-stablu kojeg pokazuje p_r su $\ge k_r$.
- List sadrži parove oblika (k, p), gdje je k vrijednost ključa, a p je fizički pokazivač na pripadni zapis u osnovnoj datoteci.
  - Parovi unutar lista su uzlazno sortirani po k.
  - List ne mora biti sasvim popunjen.
  - Jednom zapisu osnovne datoteke odgovara točno jedan par (k, p) u listovima B-stabla.

Organizacija indeksa (4)

- Indeks u obliku B-stabla može shvatiti kao hijerarhija jednostavnijih indeksa.
- Gusti primarni indeks neke datoteke prikazan kao B-stablo reda 5.



Organizacija indeksa (5)

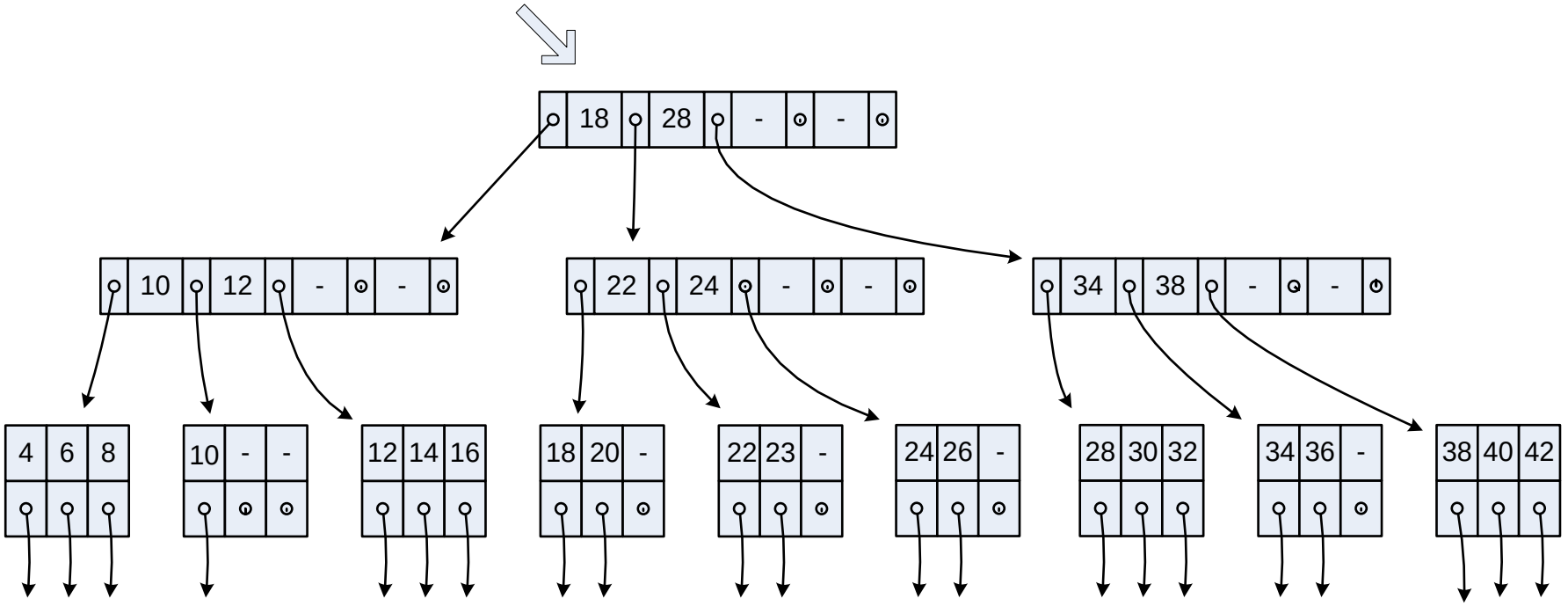
- U indeksu B-stablu moguće je vrlo brzo za zadanu vrijednost ključa *k* pronaći pokazivač *p* na odgovarajući zapis u osnovnoj datoteci.
  - Slijedimo put od korijena do lista koji bi morao sadržavati par (*k*, *p*).
    - To se radi tako da redom čitamo unutrašnje čvorove oblika (p_0, k_1, p_1, k_2, p_2, … , k_r, p_r), te usporedimo *k* s k_1, k_2, … , k_r.
    - Ako je $k_i \leq k < k_{i+1}$, dalje čitamo čvor kojeg pokazuje p_i.
    - Ako je $k < k_1$, dalje čitamo čvor s adresom p_0.
    - Ako je $k \geq k_r$, koristimo se adresom p_r.
    - Kad nas taj postupak konačno dovede u list, tražimo u njemu par sa zadanim *k*.
- U realnim situacijama B-stablo nema preveliku visinu, to jest sastoji se od svega 3-4 razine.

Organizacija indeksa (6)

- Za razliku od traženja, ubacivanje podataka u B-stablo je sporija i manje efikasna operacija.
  - Ubacivanje često zahtijeva da se neki od čvorova rascijepi na dva, te da se nakon toga izvrši promjena i u nadređenom čvoru.
  - Lančana reakcija promjena može doći sve do korijena, koji se također može rascijepiti čime se visina stabla povećava za 1.
- Izbacivanje podatka iz B-stabla odvija se analogno kao ubacivanje, no u obrnutom smjeru.
  - Prilikom izbacivanja može doći do sažimanja čvorova, te do smanjenja visine stabla.

Organizacija indeksa (7)

- B-stablo s prethodne slike nakon što je u njega ubačena nova vrijednost ključa 23.



Sadržaj Poglavlja 6

Općenito o pretvorbi u fizičku shemu

- Fizička shema baze je tekst sastavljen od naredbi u SQL-u ili nekom drugom jeziku.
  - Izvođenjem tih naredbi DBMS stvara fizičku građu baze.
- Fizička shema može se shvatiti kao opis fizičke građe.
  - Taj opis je samo implicitan jer iz njega ne možemo doslovno pročitati kako će datoteke biti organizirane.
- Projektant tu ima prilično ograničen utjecaj.
  - Većinu detalja automatski određuje DBMS pomoću svojih ugrađenih pravila.

Stvaranje početne fizičke sheme (1)

- Najvažnija SQL naredba koja se pojavljuje u fizičkoj shemi baze je naredba CREATE TABLE.
  - Njome se definira jedna relacija iz baze, dakle ime relacije, te imena i tipovi atributa.
  - Također je moguće zadati koji atributi čine primarni ključ relacije te smije li neki atribut imati neupisane vrijednosti ili ne smije.
- Početnu verziju fizičke sheme dobivamo tako da svaku relaciju iz relacijske sheme opišemo jednom naredbom CREATE TABLE.
  - Kod određivanja tipova atributa obično moramo napraviti neke kompromise budući da je popis tipova koje podržava DBMS ograničen.

Stvaranje početne fizičke sheme (2)

- Početna fizička shema za bazu podataka o fakultetu, na osnovi relacijske sheme iz Poglavlja 3. Sintaksa MySQL.

```
CREATE TABLE STUDENT
  (JMBAG NUMERIC(10) UNSIGNED NOT NULL,
   PREZIME CHAR(20),
   IME CHAR(20),
   GODINA_STUDIJA ENUM('1','2','3','4','5'),
   PRIMARY KEY(JMBAG));

CREATE TABLE PREDMET
  (SIFRA_PREDMETA NUMERIC(5) UNSIGNED NOT NULL,
   NASLOV CHAR(80),
   IME_ZAVODA CHAR(40),
   OIB_NASTAVNIKA NUMERIC(11) UNSIGNED,
   SEMESTAR ENUM('Z','L'),
   ECTS_BODOVI NUMERIC(2) UNSIGNED,
   PRIMARY KEY(SIFRA_PREDMETA));

CREATE TABLE NASTAVNIK
  (OIB NUMERIC(11) UNSIGNED NOT NULL,
   PREZIME CHAR(20),
   IME CHAR(20),
   IME_ZAVODA CHAR(40),
   BROJ_SOBE NUMERIC(3) UNSIGNED,
   PLACA NUMERIC(5) UNSIGNED,
   PRIMARY KEY(OIB));

CREATE TABLE ZAVOD
  (IME_ZAVODA CHAR(40) NOT NULL,
   OIB_PROCELNIKA NUMERIC(11) UNSIGNED,
   OPIS_DJELATNOSTI CHAR(160),
   PRIMARY KEY(IME_ZAVODA));

CREATE TABLE UPISAO
  (JMBAG NUMERIC(10) UNSIGNED NOT NULL,
   SIFRA_PREDMETA NUMERIC(5) UNSIGNED NOT NULL,
   DATUM_UPISA DATE,
   OCJENA ENUM('2','3','4','5'),
   PRIMARY KEY(JMBAG,SIFRA_PREDMETA));
```

Stvaranje početne fizičke sheme (3)

- Izvođenjem naredbi CREATE TABLE, MySQL će automatski stvoriti fizičku bazu gdje će svaka od relacija biti prikazana kao jedna datoteka.
  - Rabit će se jedna vrsta indeks-sekvencijalne organizacije koja se u MySQL-u zove MyISAM.
  - U datoteku će automatski biti ugrađen primarni indeks koji osigurava svojstvo primarnog ključa.
  - Indeks također ubrzava pretraživanje po ključu.

- Naredbe CREATE TABLE mogle su se napisati i bez navođenja primarnog ključa.
  - U tom slučaju MySQL bi se za prikaz relacije mogao koristiti i jednostavnom datotekom.

Stvaranje početne fizičke sheme (4)

  - Ne bi bilo ugrađenog indeksa, ne bi se garantirala jedinstvenost vrijednosti ključa.
  - Traženje po ključu zahtijevalo bi sekvencijalno čitanje.

- Neki DBMS-i, na primjer Oracle, omogućuju i prikaz relacije u obliku *hash* tablice.
  - Tada u odgovarajućoj naredbi CREATE TABLE moramo navesti posebnu opciju.
  - Odabirom *hash* tablice dodatno se ubrzava traženje po ključu, no usporavaju se sve operacije koje ovise o sortiranom redoslijedu po ključu.
  - Jedinstvenost vrijednosti ključa u *hash* tablici DBMS može garantirati provjerom sadržaja pretinca prilikom svakog upisa podataka.

Daljnje dotjerivanje fizičke sheme (1)

- Fizička shema s prethodne slike osigurava da će se u svim datotekama traženje po primarnom ključu odvijati relativno brzo.
- No traženje po drugim podacima bit će sporo jer će zahtijevati sekvencijalno čitanje datoteka.
- Pretraživanje po odabranim podacima koji nisu ključevi možemo ubrzati ako DBMS-u naredimo da sagradi odgovarajuće sekundarne indekse.
- U tu svrhu rabe se SQL naredbe CREATE INDEX.

Daljnje dotjerivanje fizičke sheme (2)

- Shemi s prethodne slike dodaje se indeks za PREZIME u STUDENT, te indeksi za JMBAG odnosno SIFRA_PREDMETA u UPISAO.
  - Indeks-sekvencijalna datoteka za STUDENT odnosno UPISAO nadograđuje se do invertirane organizacije.
  - Omogućeno je brzo pronalaženje studenta po prezimenu ili ispis svih studenata sortirano po prezimenu.
  - Brzo se pronalaze svi predmeti koje je upisao zadani student, te svi studenti koji su upisali zadani predmet.

```
CREATE INDEX SP_IND ON STUDENT (PREZIME);

CREATE INDEX UJ_IND ON UPISAO (JMBAG);

CREATE INDEX US_IND ON UPISAO (SIFRA_PREDMETA);
```

Daljnje dotjerivanje fizičke sheme (3)

- Uvođenjem sekundarnih indeksa poboljšavaju se performanse baze prilikom pretraživanja.
- No svaki sekundarni indeks predstavlja dodatni teret za DBMS budući da on zauzima prostor na disku i mora se ažurirati.
  - Zato projektant baze ne smije pretjerivati sa stvaranjem indeksa, već treba procijeniti koje su stvarne potrebe aplikacija.
  - Indeks je zaista potreban samo za one podatke po kojima se vrlo često pretražuje, ili ako se zahtijeva izuzetno velika brzina odziva.

Stvaranje i inicijalno punjenje baze (1)

- Fizička baza podataka nastaje izvođenjem naredbi iz fizičke sheme.
- Izvođenjem svih naredbi s prethodne dvije slike stvaraju se sve datoteke i indeksi koji čine fizičku bazu podataka o fakultetu.
- Detalji postupka u slučaju MySQL-a:

```
> CREATE DATABASE fakultet;
> USE fakultet;
> SOURCE CreateTables.txt;
```

Stvaranje i inicijalno punjenje baze (2)

- Prethodni redci su naredbe koje se izvode interaktivno unutar komandne ljuske mysql.
  - Cjelokupna fizička shema pohranjena kao jedna tekstualna datoteka s imenom CreateTables.txt.
  - Prvi redak stvara praznu bazu fakultet, dakle bazu u kojom nema ni jedne relacije.
  - Drugi redak otvara tu bazu.
  - Treći redak pokreće CreateTables.txt kao komandnu datoteku (*script file*).
- Nakon opisanog postupka, fakultetska baza imat će sve potrebne relacije, no te relacije će biti prazne, to jest u njima neće biti *n*-torki.

Stvaranje i inicijalno punjenje baze (3)

- Inicijalno punjenje baze moguće obaviti standardnim SQL naredbama za ažuriranje.
- Evo nekoliko primjera takvih naredbi:

```
INSERT INTO STUDENT VALUES (0036398757, 'Markovic', 'Marko',   '1');

INSERT INTO NASTAVNIK
  VALUES (13257600947, 'Cantor', 'Georg', 'Zavod za matematiku', 102,12000);

UPDATE PREDMET
  SET OIB_NASTAVNIKA = 67741205512  WHERE  NASLOV = 'Baze podataka';

DELETE FROM UPISAO WHERE  JMBAG = 1191203289;
```

Stvaranje i inicijalno punjenje baze (4)

- Većina DBMS-a nudi i dodatne mehanizme kojima se inicijalno punjenje baze može obaviti efikasnije.
- MySQL omogućuje da inicijalni sadržaj jedne relacije pripremimo u obliku tekstualne datoteke, pa ga onda jednom naredbom pretočimo u bazu.
- Pretpostavimo da datoteka StudentData.txt sadrži podatke za 7 studenata.
  - Svaki redak datoteke odgovara jednoj *n*-torki iz relacije STUDENT.
  - Vrijednosti atributa odvojene su znakovima tab i poredane su onako kako je određeno u odgovarajućoj naredbi CREATE TABLE.

Stvaranje i inicijalno punjenje baze (5)

0036398757	Marković	Marko	1
1191203289	Petrović	Petar	2
1192130031	Horvat	Dragica	2
1191205897	Janković	Marija	1
0165043021	Kolar	Ivan	3
0036448430	Grubišić	Katica	3
0246022858	Vuković	Janko	1

- Sljedeće naredbe izvode se iz komandne ljuske mysql – njima se sadržaj iz StudentData.txt pretvara u 7 *n*-torki relacije STUDENT.

```
> USE fakultet;
> LOAD DATA INFILE “StudentData.txt“ INTO TABLE STUDENT;
```

Stvaranje i inicijalno punjenje baze (6)

- Nakon inicijalnog punjenja, baza je spremna za rad.
  - U slučaju naše fakultetske baze podataka bilo bi moguće izvesti SQL upite iz Potpoglavlja 5.3.
- Također, pomoću raznih alata i programskih jezika mogli bismo dalje razvijati aplikacije.
  - Te aplikacije bi služile za daljnje ažuriranje podataka, izvještavanje, računanje statistika i slično.

Sadržaj Poglavlja 6

Općenito o izvrednjavanju upita

- Podrazumijevali smo da DBMS zna obavljati jednostavne radnje s relacijama (datotekama):
  - unos, promjena, brisanje ili pronalaženja *n*-torke (zapisa).
- No od implementacije baze očekuje se i više:
  - Ona također treba efikasno odgovarati na složene upite gdje se povezuju podaci iz raznih relacija.
  - DBMS treba imati na raspolaganju i algoritme koji omogućuju interpretaciju i izvrednjavanje složenih upita.
- U ovom potpoglavlju nastojimo prikazati način kako DBMS odgovara na upite (opći tijek, detalji).

Opći tijek izvrednjavanja upita (1)

- Upit se obično postavlja u neproceduralnom jeziku kao što je SQL.
- Pronalaženje odgovora odvija se u četiri faze.
  1. Prevođenje upita u relacijsku algebru.
  2. Viša razina optimizacije: algebarska transformacija.
  3. Niža razina optimizacije: odabir algoritma za izvrednjavanje pojedine algebarske operacije.
  4. Izvrednjavanje pojedine algebarske operacije odabranim algoritmom.
- Prevođenje iz SQL-a u relacijsku algebru utemeljeno je na ekvivalenciji SQL-a i relacijske algebre u smislu izražajnosti.

Opći tijek izvrednjavanja upita (2)

- U fazi optimizacije na višoj razini, polazni algebarski izraz koji je bio dobiven prevođenjem iz SQL-a nastoji se transformirati u ekvivalentni izraz koji je pogodniji za izvrednjavanje.
  - Pod ekvivalentnim izrazima smatramo one koji imaju jednaku vrijednost.
  - Izraz je pogodniji za izvrednjavanje ako se on može izračunati u kraćem vremenu ili uz manji utrošak memorije.
  - Transformacija se obavlja primjenom posebnih pravila o ekvivalenciji algebarskih izraza.

Opći tijek izvrednjavanja upita (3)

- Zadnje dvije faze u pronalaženju odgovora na upit ponavljaju se više puta, dakle jednom za svaku operaciju u transformiranom algebarskom izrazu, poštujući redoslijed zapisan u izrazu.
  - U algebarskom izrazu pojavljuju se unarne ili binarne operacije koje od relacija rade relacije.
  - S obzirom da su relacije fizički prikazane datotekama, algoritmi za izvrednjavanje algebarskih operacija zapravo su algoritmi koji čitaju jednu ili dvije ulazne datoteke te ispisuju izlaznu datoteku.
  - Složenost takvih algoritama mjeri se količinom podataka koje treba pročitati ili ispisati.

Opći tijek izvrednjavanja upita (4)

- Faza optimizacije na nižoj razini uvodi se zato jer za jednu algebarsku operaciju obično imamo na raspolaganju više algoritama koji je izvrednjavaju.
  - Svaki od tih algoritama u određenim okolnostima može se pokazati bržim ili sporijim od ostalih.
  - Optimizacija bi trebala osigurati da se od raspoloživih algoritama bira onaj koji je u danoj situaciji najbrži.
  - Odabir se zasniva na heurističkim pravilima koja procjenjuju brzinu algoritma preko raznih parametara:
    - veličine datoteka,
    - način na koje su one sortirane,
    - postojanje ili nepostojanje indeksa, ...

Opći tijek izvrednjavanja upita (5)

- Odgovaranje na upite u najvećoj mjeri se svodi na izvrednjavanje operacija iz relacijske algebre.
  - Sljedeća dva odjeljka zato detaljnije opisuju konkretne algoritme za izvrednjavanje algebarskih operacija.
  - Pritom nam je u središtu pažnje izvrednjavanje prirodnog spoja.
- Odgovaranje na upite nastoji se ubrzati primjenom dviju razina optimizacije.
  - Optimizacija upita je složeno područje.
  - Ipak, u zadnjem odjeljku navodimo kao ilustraciju nekoliko primjera pravila za optimizaciju.

Izvrednjavanje prirodnog spoja (1)

- Promatramo relacije $R_1(A,B)$ i $R_2(B,C)$ sa zajedničkim atributom B.
- Označimo sa $S(A,B,C)$ prirodni spoj od R_1 i R_2.
- Svaka od ovih triju relacija fizički se prikazuje jednom (istoimenom) datotekom:
  - n-torke se pretvaraju u zapise,
  - atributi se pretvaraju u istoimene osnovne podatke.
- Razmotrit ćemo nekoliko načina kako se pomoću datoteka R_1 i R_2 može generirati datoteka S.

Izvrednjavanje prirodnog spoja (2)

- **Algoritam ugniježđenih petlji.**
  - Doslovno se primjenjuje definicija prirodnog spoja.
  - Algoritam je opisan sljedećim pseudo-kodom.

```
inicijaliziraj praznu S;
učitaj prvi zapis iz R1;
dok ( nismo prešli kraj od R1 ) {
   učitaj prvi zapis iz R2;
   dok ( nismo prešli kraj od R2) {
            ako ( tekući zapisi iz R1 i R2 sadrže istu vrijednost za B )
                     stvori kombinirani zapis i ispiši ga u S;
            pokušaj učitati idući zapis iz R2;
   }
   pokušaj učitati idući zapis iz R1;
}
```

Izvrednjavanje prirodnog spoja (3)

- U algoritmu ugniježđenih petlji:
  - Datoteku R_1 čitamo samo jednom.
  - Za svaki zapis iz R_1 iznova čitamo cijelu datoteku R_2.
- Algoritam se može poboljšati tako da
  - U glavnu memoriju učitamo segment od što više blokova iz R_1.
  - Preinačimo petlje tako da uspoređujemo svaki zapis učitan iz R_2 sa svakim zapisom iz R_1 koji je trenutno u glavnoj memoriji.
  - Nakon što smo pročitali cijelu R_2 i obavili sva potrebna uspoređivanja, učitamo idući segment od R_1 u glavnu memoriju te ponavljamo postupak.

Izvrednjavanje prirodnog spoja (4)

- Nakon ovog poboljšanja:
  - R_1 se očigledno čita samo jednom,
  - R_2 se čita onoliko puta koliko ima segmenata u R_1.
- Poboljšana verzija osobito je dobra kad je jedna od datoteka dovoljno mala da stane u glavnu memoriju.
  - Tada se i R_1 i R_2 čitaju samo jednom.

- **Algoritam zasnovan na sortiranju i sažimanju.**
  - Datoteke R_1 i R_2 uzlazno su sortirane po zajedničkom podatku B.
  - U R_1 i u R_2 uočavamo skupine uzastopnih zapisa s istom vrijednošću za B.
  - Datoteka S koja sadrži prirodni spoj od R_1 i R_2 generira se sljedećim algoritmom koji podsjeća na sažimanje:

Izvrednjavanje prirodnog spoja (5)

inicijaliziraj praznu S;
učitaj prvu skupinu zapisa iz R_1;
učitaj prvu skupinu zapisa iz R_2;
dok (nismo prešli kraj ni od R_1 ni od R_2) {
    ako (tekuća skupina zapisa iz R_1 sadrži manju
    vrijednost za B nego tekuća skupina zapisa iz R_2)
        pokušaj učitati iduću skupinu zapisa iz R_1;
    inače ako (tekuća skupina zapisa iz R_2 sadrži
    manju vrijednost za B nego tekuća skupina zapisa iz R_1)
        pokušaj učitati iduću skupinu zapisa iz R_2;
    inače {
        svaki zapis iz tekuće skupine iz R_1 kombiniraj sa svakim zapisom
        iz tekuće skupine iz R_2 te sve generirane zapise ispiši u S;
        pokušaj učitati iduću skupinu zapisa iz R_1;
        pokušaj učitati iduću skupinu zapisa iz R_2;
    }
}

Izvrednjavanje prirodnog spoja (6)

- Pretpostavili smo da su skupine zapisa iz R_1 odnosno R_2 s istom vrijednošću za B dovoljno male tako da stanu u glavnu memoriju.
  - Pod ovakvim pretpostavkama i R_1 i R_2 se čitaju samo jednom.
- Ukoliko skupine ne stanu u glavnu memoriju, algoritam zasnovan na sortiranju i sažimanju treba preraditi.
  - Posebno treba učitavati segment po segment od svake skupine.
  - Segmenti jedne datoteke tada će se morati više puta učitavati.

Izvrednjavanje prirodnog spoja (7)

- Ako R_1 i R_2 nisu sortirane, tada ih najprije treba sortirati, pa tek onda računati prirodni spoj.
  - Da bismo sortirali veliku datoteku, dijelimo je na segmente koji stanu u glavnu memoriju, posebno sortiramo svaki segment i ispisujemo ga natrag u vanjsku memoriju.
  - Dalje sortirane segmente postepeno sažimljemo u sve veće i veće, sve dok na kraju ne dobijemo cijelu sortiranu datoteku.
  - Riječ je o prilično dugotrajnom postupku koji zahtijeva višestruko prepisivanje cijele datoteke.
- Sortiranje polaznih R_1 i R_2 trajat će znatno dulje nego generiranje S od već sortiranih R_1 i R_2.
  - Ipak, ako su R_1 i R_2 jako velike, cijeli postupak se isplati u odnosu na algoritam ugniježđenih petlji.

Izvrednjavanje prirodnog spoja (8)

- **Algoritam zasnovan na indeksu.**
  - Pretpostavimo da jedna od datoteka R_1 i R_2, na primjer R_2, ima sekundarni indeks za zajednički podatak B.
  - Tada se datoteka S, koja sadrži prirodni spoj od R_1 i R_2, može generirati na sljedeći način.

```
inicijaliziraj praznu S;
učitaj prvi zapis iz R1;
dok ( nismo prešli kraj od R1 ) {
    pomoću indeksa pronađi i učitaj sve zapise iz R2
    koji imaju istu vrijednost za B kao tekući zapis iz R1;
    tekući zapis iz R1 kombiniraj sa svakim od učitanih
    zapisa iz R2 te generirane zapise ispiši u S;
    pokušaj učitati idući zapis iz R1;
}
```

Izvrednjavanje prirodnog spoja (9)

- Algoritam zasnovan na indeksu:
  - jednom pročita cijelu R_1,
  - iz R_2 čita samo one zapise koji sudjeluju u prirodnom spoju.
- Ako i R_1 i R_2 imaju indeks za B, tada treba sekvencijalno čitati manju datoteku, a rabiti indeks veće datoteke.

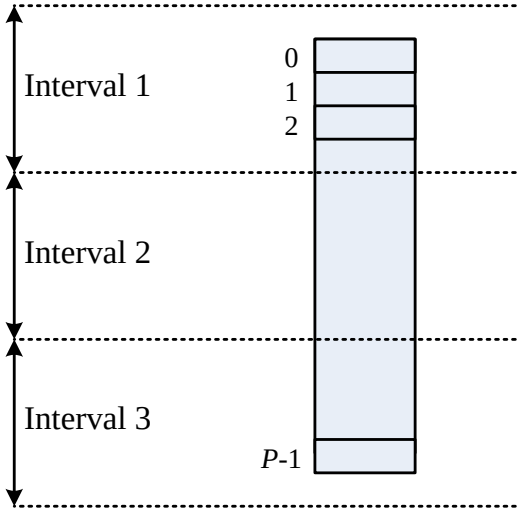
- **Algoritam zasnovan na *hash* funkciji i razvrst.**
  - Zadajemo hash funkciju h koja ovisi o podatku B.
  - Kombinacija zapisa iz datoteke R_1 sa zadanim zapisom iz datoteke R_2 pojavljuje se u prirodnom spoju S ako i samo ako oba zapisa imaju jednaku vrijednost za B.
  - Zato *hash* funkcija za oba zapisa daje istu vrijednost.
  - Razvrstavanjem zapisa iz R_1 i R_2 u skupine onih s bliskom vrijednošću h lakše ćemo odrediti koji od njih se mogu kombinirati.

Izvrednjavanje prirodnog spoja (10)

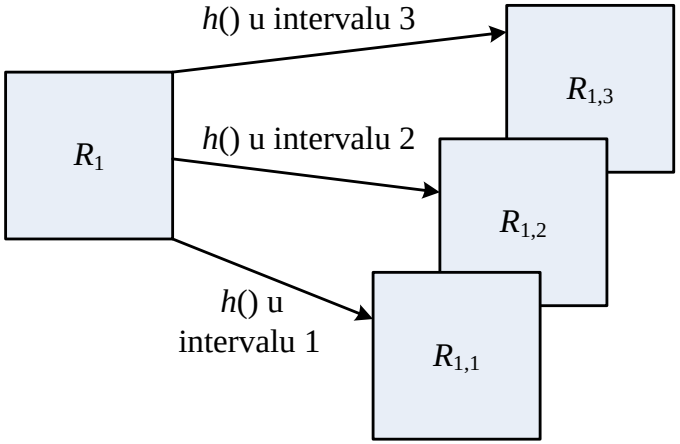
- Neka je R_1 manja od R_2. Algoritam ima pet koraka.
  1. Inicijaliziraj praznu datoteku *S*. Odaberi hash funkciju *h*. Podijeli ukupni raspon hash vrijednosti na *k* podjednakih intervala. Pritom je *k* odabran tako da 1/*k* od datoteke R_1 stane u glavnu memoriju.
  2. Čitaj sekvencijalno R_1 i razvrstaj njezine zapise u *k* skupina tako da jedna skupina sadrži sve zapise iz R_1 koje *h* preslikava u jedan od intervala. Ako je *h* zaista uniformna hash funkcija, tada su sve skupine podjednako velike - znači da jedna skupina stane u glavnu memoriju.
  3. Čitaj sekvencijalno R_2 i razvrstaj njezine zapise u *k* skupina (pomoćnih datoteka), slično kao što smo to napravili s datotekom R_1.
  4. Odaberi jedan od intervala za vrijednost *h*. Učitaj u glavnu memoriju odgovarajuću skupinu zapisa iz R_1. Sekvencijalno čitaj odgovarajuću skupinu zapisa iz R_2. Kombiniraj tekući zapis iz R_2 sa svim zapisima iz R_1 koji imaju jednaku vrijednost za *B*. Dobivene kombinacije ispiši u *S*.
  5. Ponovi korak 4, s time da odabereš novi interval za vrijednost od *h*. Ako smo već obradili svaki od *k* intervala, tada je algoritam završen.
- Svaka od datoteka R_1 i R_2 čita se točno dvaput.

Izvrednjavanje prirodnog spoja (11)

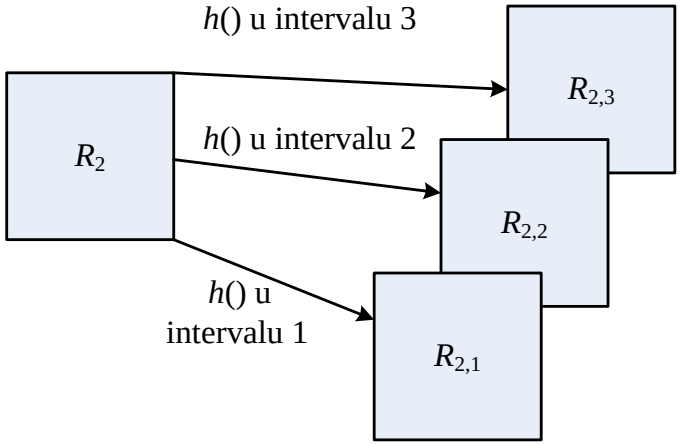
Korak 1:



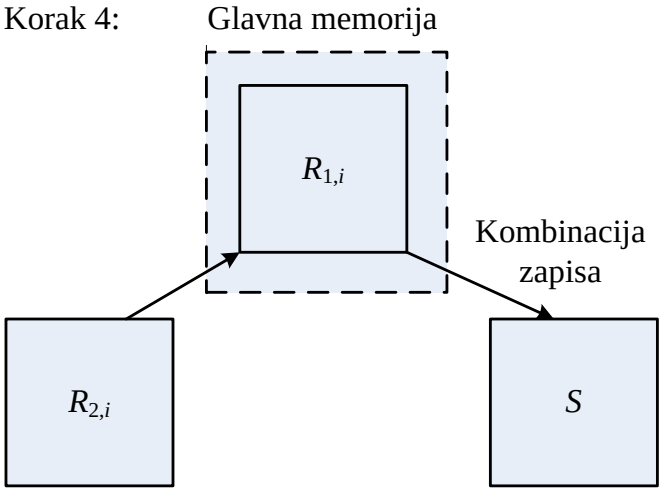
Korak 2:



Korak 3:



Korak 4:



Izvrednjavanje selekcije, proj … (1)

- Osim prirodnog spoja, najvažnije relacijske operacije su selekcija i projekcija.
  - Izložit ćemo ideje za izvrednjavanje tih dviju operacija.
  - Kratko ćemo napomenuti kako se implementiraju ostale operacije.
- **Izvrednjavanje selekcije.**
  - Zadana je relacija *R* i Booleov uvjet $\mathcal{B}$. *R* je fizički prikazana istoimenom datotekom.
  - Izvrednjavanje selekcije *R* where $\mathcal{B}$ ovisi o obliku uvjeta $\mathcal{B}$, no obično se svodi na traženje zapisa u datoteci *R* sa zadanom vrijednošću nekih podataka.
  - Dakle, obično se radi o pristupu na osnovi primarnog ključa ili o pristupu na osnovi ostalih podataka.

Izvrednjavanje selekcije, proj … (2)

- Naivni algoritam za selekciju bio bi: sekvencijalno čitanje cijele *R* i provjera svakog zapisa zadovoljava li on $\mathcal{E}$. Ako je *R* velika, to traje predugo.
- Ubrzavanje selekcije postiže se invertiranjem datoteka pomoću indeksa prikazanih B-stablima. Većina današnjih DBMS-a oslanja se na invertirane datoteke.
- Znatno rjeđe rabljeni način ubrzavanja selekcije je primjena *hash* organizacije.
- Neki oblici uvjeta $\mathcal{E}$ zahtijevaju da u *R* tražimo zapise gdje se vrijednost zadanog podatka kreće u zadanom intervalu. Indeks-B-stablo podržava ovakvo intervalno pretraživanje, *hash* organizacija ne podržava.

Izvrednjavanje selekcije, proj … (3)

- **Izvrednjavanje projekcije.**
  - Zadana je relacija *R* i njezin atribut *A*. *R* je fizički prikazana istoimenom datotekom.
  - Da bismo generirali datoteku koja odgovara projekciji *S* = *R*[*A*], očito treba pročitati cijelu datoteku *R* i izdvojiti sve vrijednosti podatka *A* koje se pojavljuju.
  - No ista vrijednost za *A* može se pojaviti više puta.
  - Osnovni problem implementiranja projekcije je: kako u *S* eliminirati zapise-duplikate?
  - Najjednostavniji algoritam za računanje projekcije zasnovan je na ugniježđenim petljama. Vanjska petlja čita datoteku *R*, a unutrašnja petlja prolazi trenutno stvorenim dijelom datoteke *S*.

Izvrednjavanje selekcije, proj … (4)

- Algoritam ugniježđenih petlji opisan je pseudo-kodom.

```
inicijaliziraj praznu S;
učitaj prvi zapis iz R;
dok ( nismo prešli kraj od R ) {
    duplikat = 0;
    pokušaj učitati prvi zapis iz S;
    dok ( nismo prešli kraj od S i duplikat==0 ) {
            ako ( tekući zapisi iz R i S sadrže istu vrijednost za A ) duplikat = 1;
            pokušaj učitati idući zapis iz S;
    }
    ako ( duplikat == 0 )
            prepiši A iz tekućeg zapisa iz R na kraj od S kao novi zapis;
    pokušaj učitati idući zapis iz R;
}
```

Izvrednjavanje selekcije, proj … (5)

- Ako je *R* velika, algoritam s ugniježđenim petljama zahtijeva previše vremena.
- Tada je bolje postupiti na sljedeći način: izdvojiti sve vrijednosti za *A* koje se pojavljuju u *R* te zatim sortirati niz izdvojenih vrijednosti. U sortiranom nizu duplikati će se pojavljivati jedan iza drugoga, pa ih je lako eliminirati jednim sekvencijalnim čitanjem.
- Još jedna ideja je: razvrstati izdvojene vrijednosti za *A* u skupine pomoću *hash* funkcije. Duplikati će se tada naći u istoj skupini, pa ih je opet lako eliminirati.

Izvrednjavanje selekcije, proj … (6)

- **Izvrednjavanje ostalih operacija.**
  - Kartezijev produkt dviju relacija R_1 i R_2 računa se ugniježđenom petljom. Bolji algoritam nije moguć.
  - Unija dviju relacija R_1 i R_2 dobiva se spajanjem datoteka. Pritom treba eliminirati zapise-duplikate. Opet pomaže sortiranje datoteke R_1 i R_2, ili uporaba *hash* funkcije.
  - Presjek dviju relacija specijalni je slučaj prirodnog spoja gdje su svi atributi zajednički. Zato algoritmi za računanje presjeka liče na one za računanje prirodnog spoja. Isto vrijedi i za skupovnu razliku.
  - Daljnji relacijski operatori mogu se izraziti pomoću već razmatranih, pa se oni ne implementiraju zasebno.

Optimizacija upita (1)

- Optimizacija upita provodi se na dvije razine.
  - Viša razina: transformacija algebarskog izraza u oblik koji je pogodniji za izvrednjavanje.
  - Niža razina: izbor algoritma za izvrednjavanje svake pojedine operacije unutar algebarskog izraza.
- Obje razine optimizacije provode se tako da DBMS primjenjuje odgovarajuća pravila.
- U nastavku navodimo 7 pravila za optimizaciju.
  - Prvih 6 pravila odnose na algebarsku transformaciju.
  - Zadnje pravilo odnosi se na izbor algoritma za izvrednjavanje.

Optimizacija upita (2)

- **Kombiniranje selekcija.**
  - Očito vrijedi ekvivalencija:

    (*R* where $\mathcal{B}_1$) where $\mathcal{B}_2$ = *R* where ($\mathcal{B}_1$ and $\mathcal{B}_2$).
  - Pravilo kaže da izraz s lijeve strane treba pretvoriti u oblik s desne strane.
  - Time smanjujemo potrebno vrijeme ukoliko se obje selekcije izvrednjavaju podjednako „sporo“ (dakle pregledom cijele relacije).
  - Ako se jedna relacija odvija brzo (npr. zbog indeksa) a druga sporo, tada se kombiniranje ne isplati.
  - Odluka što je bolje ovisi o fizičkoj građi baze.

Optimizacija upita (3)

- **Izvlačenje selekc ispred spoja ili Kart produkta.**
  - Ako uvjet $\mathcal{B}$ sadrži samo atribute od *R*, a ne one od *S*, tada treba primijeniti transformaciju:

    (*R* join *S*) where $\mathcal{B}$ = (*R* where $\mathcal{B}$) join *S*,

    (*R* times *S*) where $\mathcal{B}$ = (*R* where $\mathcal{B}$) times *S*,
  - Ovakva transformacija može znatno smanjiti broj *n*-torki koje ulaze u prirodni spoj ili Kartezijev produkt.
  - Ili ako $\mathcal{B}$ rastavimo na $\mathcal{B} = \mathcal{B}_R$ and $\mathcal{B}_S$ and $\mathcal{B}_C$ and $\mathcal{B}'$, gdje $\mathcal{B}_R$ sadrži samo atribute od *R*, $\mathcal{B}_S$ sadrži samo atribute od *S*, $\mathcal{B}_C$ sadrži zajedničke atribute od *R* i *S*, $\mathcal{B}'$ predstavlja ostatak od $\mathcal{B}$, tada vrijedi:

    (*R* join *S*) where $\mathcal{B}$ = ((*R* where ($\mathcal{B}_R$ and $\mathcal{B}_C$)) join (*S* where ($\mathcal{B}_S$ and $\mathcal{B}_C$))) where $\mathcal{B}'$
  - Slična ekvivalencija može se napisati i za times.

Optimizacija upita (4)

- Želimo naći JMBAG-ove svih studenata na drugoj godini studija koji su upisali neki predmet kod nastavnika Kleina i dobili ocjenu veću od 2.
  - Upit se može izraziti kao:

```
RESULT := ( (STUDENT join UPISAO join PREDMET) where
  ((GODINA_STUDIJA=2) and (OCJENA>2) and (OIB_NASTAVNIKA=44102179316)) )
  [JMBAG]
```

  - Primjenom transformacije dobivamo optimizirani izraz:

```
RESULT := ( (STUDENT where GODINA_STUDIJA=2) join
            (UPISAO where OCJENA>2) join
            (PREDMET where OIB_NASTAVNIKA=44102179316) )
            [JMBAG].
```

Optimizacija upita (5)

- **Izvlačenje selekcije ispred projekcije.**
  - Ako uvjet $\mathcal{Z}$ sadrži samo atribute X po kojima se obavlja projiciranje, tada treba primijeniti pravilo:

    $R[\,X\,]$ where $\mathcal{Z} = (R$ where $\mathcal{Z})\,[\,X\,]$.

  - Naime, projiciranje može dugo trajati zbog eliminacije „duplikata“. Zato je dobro najprije smanjiti broj n-torki selektiranjem.
  - To je posebno preporučljivo onda kad postoje fizička sredstva za brzu selekciju.

Optimizacija upita (5)

- **Kombiniranje projekcija.**
  - Ako su X, Y i Z atributi od relacije R, tada vrijedi:

$$((R[X,Y,Z]) [X,Y]) [X] = R[X] .$$

  - Pravilo kaže da je umjesto tri projekcije dovoljno primijeniti samo jednu.
  - U ovom jednostavnom slučaju, ta jednakost je očigledna. No u dugačkim i kompliciranim izrazima nije lako uočiti redundantno projiciranje.

Optimizacija upita (6)

- **Izvlačenje projekcije ispred prirodnog spoja.**
  - Ako X označava zajedničke atribute od relacija R i S, tada vrijedi pravilo:

    $$(R \text{ join } S)[X] = R[X] \text{ join } S[X] .$$

  - No pravilo ne vrijedi za proizvoljan skup atributa X. Neka su A_R atributi od R, A_S atributi od S, $A_C = A_R \cap A_S$ zajednički atributi od R i S. Tada vrijedi:

    $$(R \text{ join } S)[X] = (R[(X \cap A_R) \cup A_C] \text{ join } S[(X \cap A_S) \cup A_C]) [X] .$$

  - Smanjuje se broj n-torki koje ulaze u prirodni spoj.
  - Zahvat ne mora uvijek biti koristan, jer projekcija može spriječiti efikasnu implementaciju prirodnog spoja.
  - Odluka što je bolje opet ovisi o fizičkoj građi.

Optimizacija upita (7)

- **Optimizacija skupovnih operacija.**
  - Koji put su od koristi sljedeća pravila koja se odnose na skupovnu uniju ili razliku:

$$(R \text{ union } S) \text{ where } \mathcal{B} = (R \text{ where } \mathcal{B}) \text{ union } (S \text{ where } \mathcal{B}),$$
$$(R \text{ minus } S) \text{ where } \mathcal{B} = (R \text{ where } \mathcal{B}) \text{ minus } (S \text{ where } \mathcal{B}),$$
$$(R \text{ union } S)[X] = R[X] \text{ union } S[X],$$
$$(R \text{ minus } S)[X] = R[X] \text{ minus } S[X],$$
$$(R \text{ where } \mathcal{B}_1)[X] \text{ union } (R \text{ where } \mathcal{B}_2)[X] = (R \text{ where } (\mathcal{B}_1 \text{ or } \mathcal{B}_2))[X]$$

  - Predzadnja jednakost vrijedi pod pretpostavkom da X sadrži primarne atribute od R (a time i od S).
  - Transformacijama se nastoji smanjiti broj n-torki koje sudjeluju u operacijama union i minus.
  - Operacija intersect je specijalni slučaj od join, pa za nju vrijedi isto pravilo kao za join.

Optimizacija upita (8)

- **Optimalni izbor algoritma za izvrednjavanje prirodnog spoja.**
  - Treba izračunati prirodni spoj za relacije koje su pohranjene u datotekama R_1 i R_2.
  - Stoje nam na raspolaganju četiri algoritma za računanje prirodnog spoja koji su bili opisani u prethodnom odjeljku.
  - Tada odluku o izboru algoritma donosimo na sljedeći način.

Optimizacija upita (9)

- Ako su datoteke R_1 i R_2 već sortirane po zajedničkom podatku B, tada je najefikasniji algoritam zasnovan na sortiranju i sažimanju.
- Ako je jedna datoteka dovoljno mala da stane u glavnu memoriju, biramo algoritam ugniježđenih petlji.
- Ako je jedna datoteka znatno veća od druge i ima odgovarajući indeks, najbolji je algoritam zasnovan na indeksu.
- Za velike datoteke R_1 i R_2 bez indeksa, najbolji izbor je algoritam zasnovan na *hash* funkciji i razvrstavanju.
- Ovo pravilo zasnovano je na prije provedenoj analizi složenosti pojedinih algoritama.