

## KAZALO

1	UVOD.....	2
2	RELACIJSKI PODATKOVNI MODEL.....	5
3	MODEL PODATKOV ER.....	9
4	DISKI IN DATOTEKE.....	13
5	INDEKSI.....	18
6	OCENJEVANJE POIZVEDB.....	24
7	OPTIMIZACIJA POIZVEDB.....	29
8	KONTROLA SOČASNOSTI .....	32
9	OBNOVITEV PO ZRUŠITVI .....	36
10	LOGICAL DESIGN.....	40

[KNJIGA: Database Management Systems, 3rd Edition: Raghu Ramakrishnan, Johannes Gehrke](#)

*Prevod je lahko včasih napačen, zato priporočam, da si preberete v knjigi. Za slovnične napake se opravičujem. Če kaj ni prav popravite ali pa dodajte **IN MI NAPIŠITE**. Za napačne odgovore ali slabe ocene ne odgovarjam to je samo moja dobra volja.*

*LP Ž*

# **1 UVOD**

## **1) Kaj je SUPB?**

Sistem za Upravljanje s Podatkovnimi Bazami. Je programski sistem za shranjevanje podatkov in upravljanje z podatkovnimi bazami.

## **2) Datoteke vs. SUPB? Str. 8 -> Pdf. 43**

Težave pri tem, da bi imeli podatke shranjene v datotekah so naslednje:

- Nimamo dovolj velikega spomina, da lahko shranimo vse podatke. Datoteke moramo shraniti na kakšen disk ali trak, nato pa prenesti datoteke v glavni spomin, da jih lahko obdelujemo.
- Tudi če imamo dovolj velik glavni spomin z 32-bitnim sistemom lahko dostopamo samo do 4GB podatkov na enkrat. Moramo sprogramirati metodo za prepoznavanje vseh datotečnih elementov.
- Spisati moramo programe, da lahko odgovorimo vsako vprašanje, ki bi jih naj uporabnik zastavil, glede podatkov. Programi bodo zahtevni, zaradi velike količine podatkov.
- Podatke moramo zaščititi pred nedoslednimi spremembami, ki jih naredijo drugi uporabniki, ki hkrati dostopajo do podatkov. Če mora program obravnavati podatke s sočasnim dostopom in hkrati zaščiti ostale, to zelo prispeva k njihovi kompleksnosti.
- Zagotoviti moramo, da se podatki obnovijo v prvotno stanje, če se sistem zruši medtem, ko se podatki spreminjajo.
- Operacijski sistemi imajo samo geslo za zaščito. To ni dovolj prilagodljivo, da bi lahko uveljavili varnost in kontrolo dostopa, da bi lahko dovolili, da uporabniki dostopajo do svojih datotek, ki so v različnih podskupinah

SUPB, je programska oprema, ki je narejena zato, da so zgoraj naštete težave lahke za obravnavo. Ko shranimo v SUPB sistem podatke, lahko uporabimo SUPB lastnosti za njihovo obravnavo. Bolj ko naša baza raste bolj nepogrešljiv je SUPB sistem.

## **3) Kaj je podatkovni model?**

**Podatkovni model** je zbirka konceptualnih gradnikov za opis podatkov.

**Shema** je opis konkretnne zbirke podatkov z uporabo danega podatkovnega modela.

**Relacijski podatkovni model** je najbolj pogosto uporabljan model danes.

Osnovni koncepti:

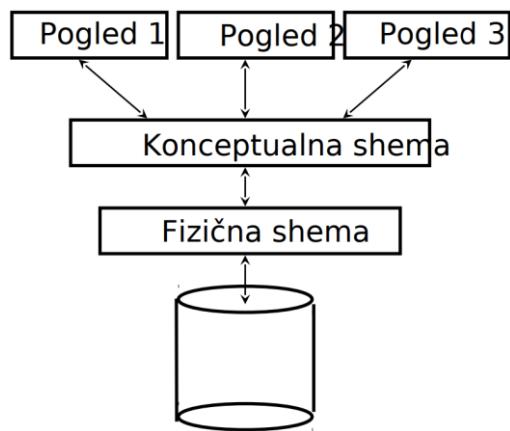
- Relacija, ki je v osnovi tabela s stolpci in vrsticami.
- Vsaka relacija ima shemo, ki opisuje stolpce in vrstice.

#### **4) Kaj je podatkovna neodvisnost? [Str. 15](#) -> [Pdf. 50](#)**

Podatkovna neodvisnost je to, da so programi izolirani od sprememb, kako so podatki strukturirani in shranjeni. Podatkovna neodvisnost je pridobljena skozi tri nivoje abstrakcije podatkov: konceptualno shemo, fizično shemo in zunanjo shemo. Predvsem vlogo pri podatkovni neodvisnosti imata konceptualna in zunanja shema, ki na tem področju prineseta veliko korist.

#### **5) Opišite ravni abstrakcije v SUPB. [Str. 12](#) -> [Pdf. 47](#)**

Obstajajo tri ravni abstrakcije: konceptualna, fizična in zunanja. Konceptualna shema definira logično strukturo. Fizična shema opisuje uporabljene datoteke in indekse. V "Pogledi" Zunanja shema opisuje kako uporabnik vidi podatke. Shema je definirana z uporabo DDL (Data definition language).



#### **6) Kaj je transakcija?**

Transakcija je osnovni koncept, ki je atomarna sekvenca akcij SUPB (branje / pisanje). Vsaka transakcija, ki se izvrši mora pustiti podatkovno bazo v konsistentnem stanju, če je podatkovna baza konsistentna, ko se je transakcija začela izvajati.

- Uporabniki lahko specificirajo enostavne integritetne omejitve nad podatki in podatkovna baza bo zagotovila veljavnost omejitev.
- SUPB zares ne razume pomena podatkov (npr., ne razume kako se računajo obresti na bančnem računu).
- Torej, zagotavljanje, da transakcija ohranja konsistentnost podatkovne baze je odgovornost uporabnika.

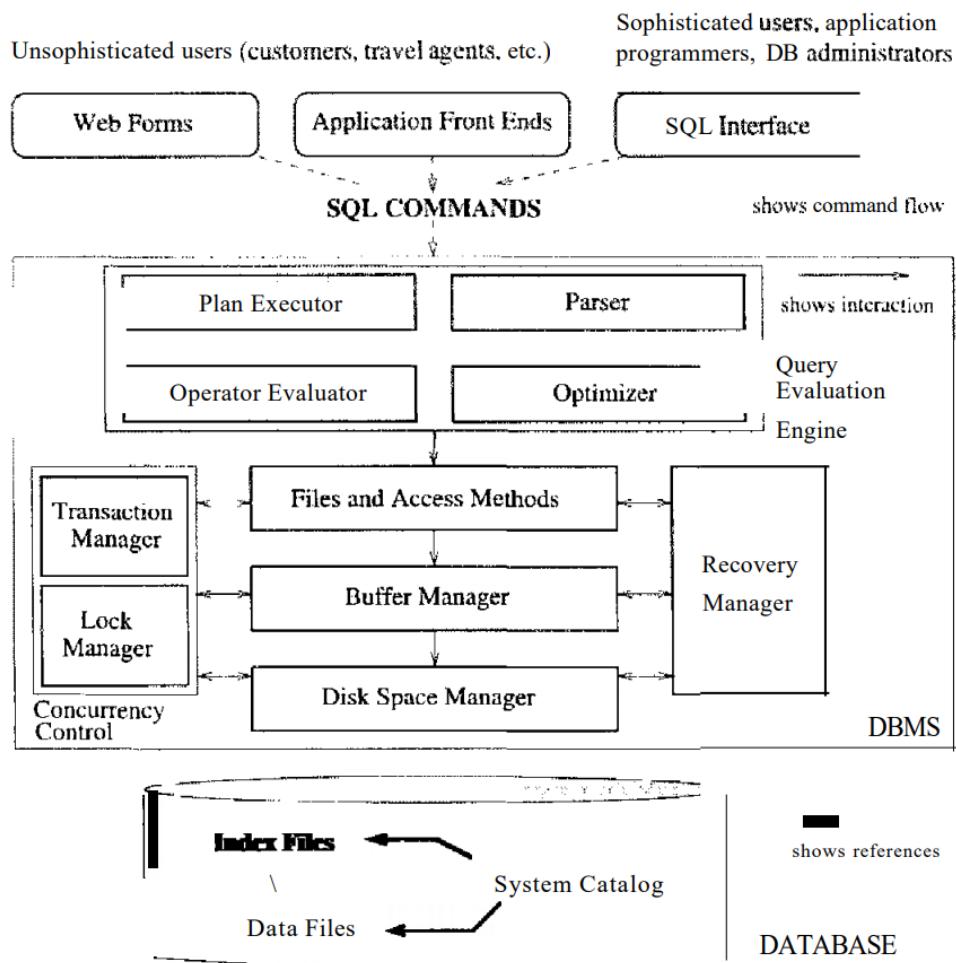
#### **7) Opiši strukturo SUPB. [Str. 19](#) -> [Pdf. 54](#)**

SUPB sprejme SQL komande, ki so generirane iz več različnih uporabniških vmesnikov, uporabnik izdeluje načrte za ocenjevanje poizvedb, izvaja te načrte proti bazi podatkov in vrne odgovore. Ko uporabnik izda poizvedbo, je razčlenjena poizvedba predstavljena optimizatorju poizvedb, ki uporablja informacije o tem, kako so shranjeni podatki, da izdela učinkovit načrt izvajanja za ocenjevanje poizvedbe.

Koda, ki izvaja operacije relacij, se nahaja na vrhu datoteke in dostopa metode. Ta plast podpira koncept datoteke, ki je v SUPB zbiranje strani ali zbirka zapisov. Datotečne kopice ali datoteke neurejenih strani, kot tudi indeksi. Poleg sledenja strani v datoteki, ta plast organizira informacije na strani.

Datoteka in koda načina dostopa se nahaja na vrhu upravljalnika medpomnilnika. Prenaša strani iz diska v glavni pomnilnik, kot je potrebno za odgovor na zahtevano branje. Najnižji nivo SUPB programske opreme se ukvarja z upravljanjem prostora na disku, kjer so shranjeni podatki.

SUPB podpira vzporednost in "crash recovery" tako, da previdno načrtuje in izvaja vse kar uporabnik vnese v bazo. Baza ima tudi protokol, da se zaklene in izvaja vzdrževalna dela dokler ne konča. Sledi tudi vsem zahtevam uporabnika in jih skrbno načrtuje kdaj jih bo lahko izvedla.



## **2 RELACIJSKI PODATKOVNI MODEL**

### **1) Kaj je relacijski model baze podatkov?**

Relacijski model predstavlja bazo podatkov kot zbirko "odnosov" oz. relacij. Relacija nič drugega kot tabela vrednosti. Vsaka vrsta v tabeli predstavlja zbirko povezanih podatkovnih vrednosti. Vrstice v tabeli označujejo subjekt ali razmerje v realnem svetu.

Ime tabele in imena stolpcev so v pomoč pri razlagi pomena vrednosti v vsaki vrstici. Podatki so predstavljeni kot niz odnosov. V relacijskem modelu so podatki shranjeni kot tabele. Vendar pa je fizično shranjevanje podatkov neodvisno od načina, kako so podatki logično organizirani.

### **2) Kaj so integritetne omejitve?**

Integritetne omejitve (IC) so pogoj, ki je podan v shemi baze podatkov in omejujejo podatke, ki jih je mogoče shraniti v bazo podatkov. Legalna instanca relacije, je takšna, ki zadovoljuje vse specificirane integritetne omejitve (SUPB ne bi smel dopuščati nelegalnih instanc). Če SUPB preverja integritetne omejitve, potem so shranjeni podatki bližje pomenu v realnem svetu (izogibanje napakam pri vnosu).

### **3) Opiši koncept primarnega ključa.**

Primarni ključ je poseben stolpec tabele relacijske baze podatkov (ali kombinacije stolpcev), ki je narejena da enkratno identificira vse zapise tabele. Glavne značilnosti primarnega ključa so, da mora vsebovati edinstveno vrednost za vsako vrstico podatkov in ne more vsebovati ničelnih vrednosti.

Primarni ključ je množica atributov relacije, če:

- Ne obstajata dva enaka zapisa z isto vrednostjo ključa
- To ne velja za nobeno podmnožico ključa
  - Če obstaja potem temu pravimo superključ?
  - Če obstaja več kot en ključ za relacijo potem izberemo enega, ki ga imenujemo primarni ključ.

### **4) Kaj je referenčna integriteta?**

Referenčna integriteta se nanaša na tuji ključ. Tuj ključ je množica atributov neke relacije, ki referencira zapise druge relacije. Izbrana množica atributov mora ustrezati primarnem ključu druge relacije. Neke vrste "logični kazalec".

Primer: sid je tuj ključ (Studenti):

- Vpis(sid: string, pid: string, ocena: string)
- Če je integritetna omejitev tujih ključev upoštevana v podatkovni bazi potem pravimo, da dosežemo referenčno integriteto; ni "visečih referenc".

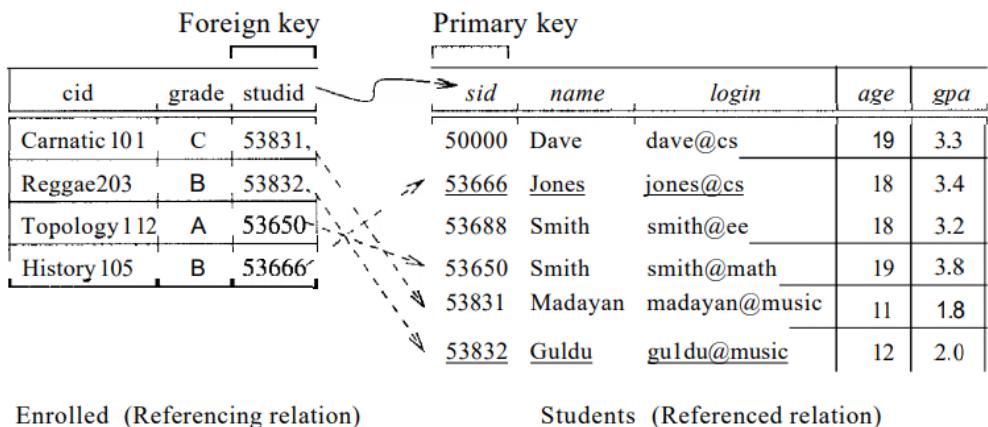


Figure 3.4 Referential Integrity

### 5) Opiši relacijsko algebro. Str. 102 -> Pdf. 137

Relacijska algebra je ena od dveh formalnih poizvedbenih jezikov, povezanih z relacijskim modelom. Poizvedbe v algebri so sestavljene iz zbirke operaterjev. Temeljna lastnost je, da vsak operater v algebri sprejema (en ali dva) primera relacij kot argumenta in vrne primer razmerja rezultata (Vsaka operacija vrne relacijo kot rezultat.) . Ta lastnost olajša sestavljanje operaterjev za tvorbo kompleksne poizvedbe - izraz relacijske algebre je rekurzivno definiran kot relacija, operater eno člene algebre, ki se uporablja za en sam izraz, ali operator binarne algebre za dva izraza. Osnovne operacije: Selekcija (  $\Pi$  ), Projekcija (  $\delta$  ), Produkt (  $X$  ), Unija (  $U$  ), Razlika (  $-$  ). Dodatne operacije (niso nujne, ampak so koristne): Presek, Stik, Deljenje, Preimenovanje.

### 6) Opišite relacijski račun. Str. 116 -> Pdf. 151

Relacijski račun je alternativa relacijski algebri. V nasprotju z relacijsko algebro, ki je proceduralni relacijski račun ni nepredelan ali deklarativen, kar nam omogoča, da opišemo niz odgovorov, ne da bi bili eksplisitni o tem, kako jih je treba izračunati. Uporabniki definirajo vprašanja tako, da zapišejo kaj želijo in kako naj sistem poišče rezultat.

- Dva jezika: **N-terični relacijski račun (TRC)** in **Domenski relacijski račun (DRC)**.
- Izrazi vsebujejo spremenljivke, konstante, primerjalne operacije, logične operacije in kvantifikatorje.
  - TRC: Spremenljivke so omejene na n-terice.
  - DRC: Spremenljivke so omejene na domene atributov.
  - TRC in DRC so podmnožice predikatnega računa.
- Izraze teh jezikov imenujemo formule. N-terico, ki je odgovor na vprašanje dobimo tako, da prostim spremenljivkam priredimo konstante tako, da je vrednost formule enaka true.

## 7) Predstavite osnovno sintakso poizvedbenega jezika SQL.

- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database
- CREATE DATABASE - creates a new database
- ALTER DATABASE - modifies a database
- CREATE TABLE - creates a new table
- ALTER TABLE - modifies a table
- DROP TABLE - deletes a table
- CREATE INDEX - creates an index (search key)
- DROP INDEX - deletes an index

### AND Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

### OR Syntax

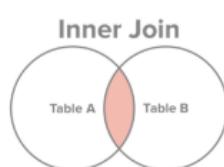
```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

### NOT Syntax

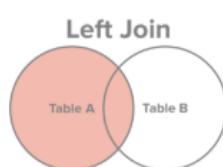
```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

## 8) Opišite koncept združevanja (JOIN) v SQL.,

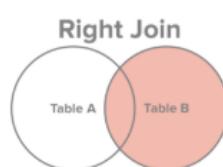
Poznamo štiri tipa SQL združevanja: notranji (inner), levi (left), desni (right) in poln (full). Lahko skombiniramo atribute iz več kot ene tabele da dobimo željen rezultat.



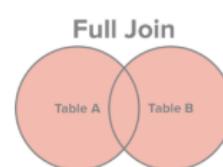
Select all records from Table A and Table B, where the join condition is met.



Select all records from Table A, along with records from Table B for which the join condition is met (if at all).



Select all records from Table B, along with records from Table A for which the join condition is met (if at all).



Select all records from Table A and Table B, regardless of whether the join condition is met or not.

### 9) Opišite agregacijske funkcije v SQL.

V bazi je upravljanje agregatne funkcije funkcija, pri kateri so vrednosti več vrstic združeni kot vhodni podatek na določenih merilih, da tvorijo eno samo vrednost bolj pomembnega pomena.

## Agregacijske operacije

- Pomembna razširitev relacijske algebре.

COUNT (\*)  
COUNT ( [DISTINCT] A)  
SUM ( [DISTINCT] A)  
AVG ( [DISTINCT] A)  
MAX (A)  
MIN (A)

*single column*

```
SELECT COUNT (*)
FROM Mornarji M
```

```
SELECT M.mime
FROM Mornarji M
WHERE M.ocena= (SELECT MAX(M2.ocena)
                  FROM Mornarji M2)
```

```
SELECT AVG (M.star)
FROM Mornarji M
WHERE M.ocena=10
```

```
SELECT COUNT (DISTINCT M.ocena)
FROM Mornarji M
WHERE M.mime='miha'
```

```
SELECT AVG (M.star)
FROM Mornarji M
WHERE M.ocena=10
```

### 10) Opišite poizvedbeni jezik QBE (Query by example).

QBE je metoda poizvedbe, uporabljena v večini sistemov baz podatkov, predvsem za relacijske podatkovne baze. Zasnovan je na relacijskem računu. Ima enostavno sintakso, je eleganten, prijazen do uporabnika in relacijsko kompleten. Je primeren za enostavna vprašanja in neroden za kompleksna.

QBE je jezik poizvedbe, ki se uporablja v relacijskih bazah podatkov, ki uporabnikom omogoča iskanje informacij v tabelah in poljih z zagotavljanjem enostavnega uporabniškega vmesnika, kjer bo uporabnik lahko vnesel primer podatkov, do katerih želi dostop. Jezik QBE je zgolj abstrakcija med uporabnikom in dejansko poizvedbo, ki jo bo prejel sistem baze podatkov. V ozadju se poizvedba uporabnika preoblikuje v obliko jezika za manipulacijo baze podatkov, kot je SQL, in ta stavek SQL bo izveden v ozadju.

### **3 MODEL PODATKOV ER**

#### **1) Opišite podatkovni model ER.**

ER model nam omogoča, da opišemo podatke, ki so vključeni v resnični svet v smislu objektov in njihovih razmerji, ter je široko uporabljen za razvoj začetne zasnove baze podatkov. Zagotavlja uporabne koncepte, ki nam omogočajo, da se iz neformalnega opisa, kaj uporabniki želijo do svoje baze podatkov, premakne na podrobnejši in natančnejši opis, ki ga je mogoče implementirati v SUPB.

ER diagram je sestavljen iz entitet, atributov in razmerji.

#### **2) Kaj je kardinalnost odnosa? (Mislim da je to števnost razmerja)**

Kardinalnost določa, koliko primerov entitete se nanaša na en primerek drugega subjekta. Pravilnost je tudi tesno povezana s kardinalnostjo. Medtem ko kardinalnost opredeljuje pojav razmerja, pravilo opisuje razmerje kot obvezno ali neobvezno. Z drugimi besedami, kardinalnost določa največje število razmerij.

Stopnja odnosa (kardinalnost) je število pojavov v eni enoti, ki so povezani s številom pojav v drugi. Obstajajo tri stopnje odnosa znane kot:

- 1 – 1 -> ena-proti-ena -> če sta obe E in F v razmerju R eno vrednostni
- N – N -> mnogo-proti-mnogo -> če sta obe E in F večvrednostni v razmerju R
- 1 – N (N - 1) -> ena-proti-mnogo -> če ima ena izmed entitetnih množic eno vrednostno vlogo in druga večvrednostno vlogo v razmerju R

#### **3) Kaj je šibka entiteta? Pdf. 70**

Doslej smo domnevali, da atributi, povezani z množico entitet, vključujejo ključ. Ta predpostavka ne velja vedno. Šibko entiteto je mogoče enoznačno identificirati samo z upoštevanjem nekaterih njegovih atributov v povezavi s primarnim ključem drugega subjekta, ki se imenuje identifikacijski lastnik.

Upoštevati je treba naslednje omejitve:

- Nizi lastniških entitet in nizi šibkih entitet morajo sodelovati v nizu odnosov ena-proti-mnogo (en lastnik je povezan z enim ali več šibkimi entitetami, vendar ima vsaka šibka entiteta enega samega lastnika). Ta nabor razmerij se imenuje množica identifikacijskih razmerij niza šibkih entitet.
- Niz šibkih entitet mora imeti popolno udeležbo v določenem razmerju identitete.

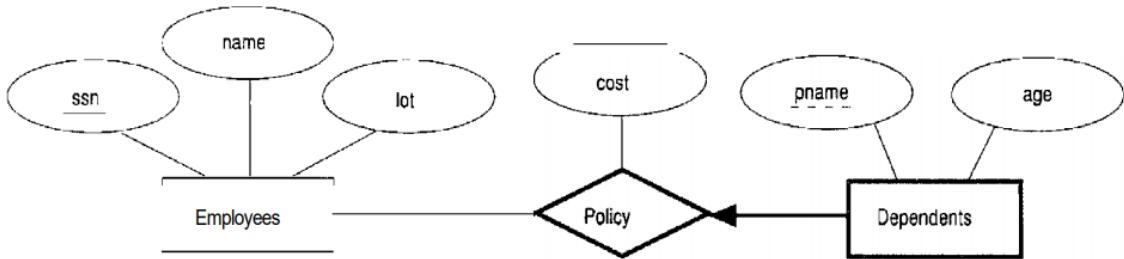
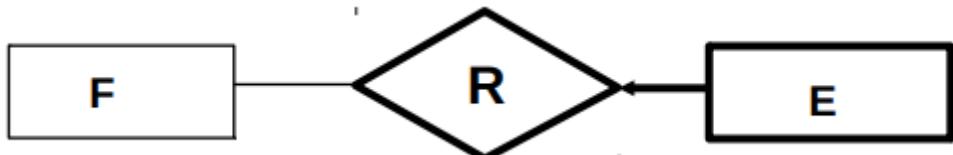
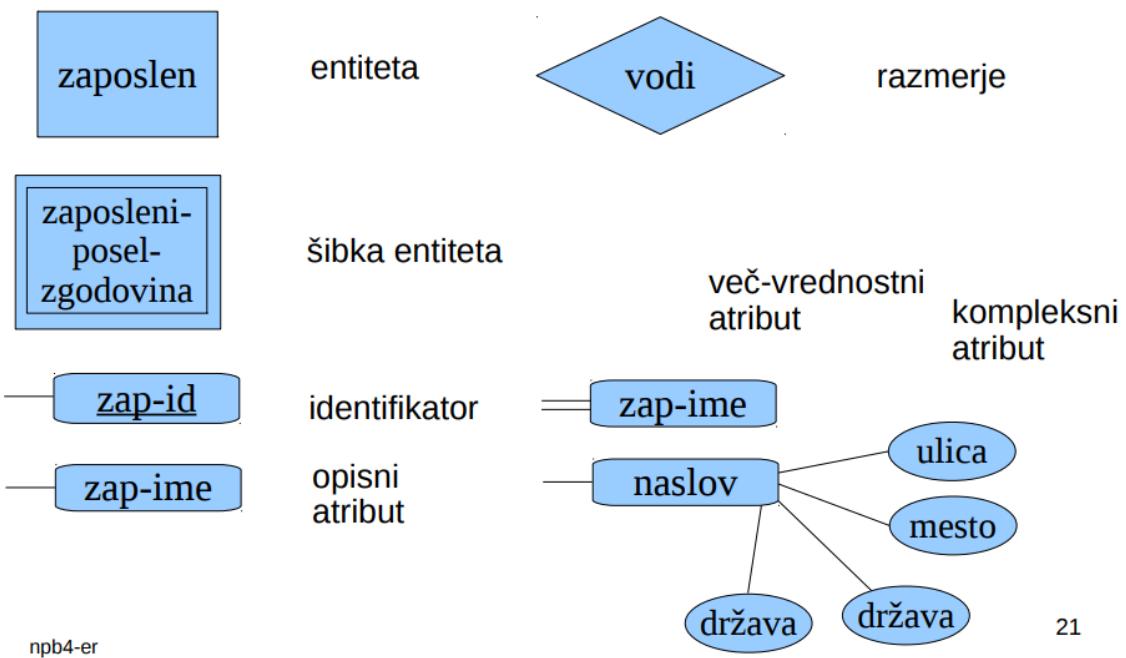


Figure 2.11 A Weak Entity Set



- Entitetna množica **E** je **šibka**, če je obstoj entitet iz entitetne množice E pogojen z obstojem entitet iz entitetne množice F.
  - Z drugimi besedami, entiteta iz množice E lahko obstaja samo, če obstaja pripadajoča entiteta iz množice F.
  - Identifikator entitetne množice E je tako relevanten samo v okviru povezave z F.

#### 4) Predstavite Chenov zapis ER modela.



21

**5) Predstavite konstrukte za modeliranje generalizacije / specializacije hierarhije entitet.**

## Generalizacija

- Prevod generalizacijske hierarhije sestavljen iz supertipa in podtipov lahko vodi do več SQL tabel
- Tabela izpeljana iz supertipa vsebuje ključ entitete supertipa in vse skupne atribute hierarhije
- Vsaka tabela izpeljana iz entitet podtipa vsebuje ključ entitete supertipa in samo atribute specifične za podtip
- Integriteto pri popravljanju je potrebno ohranljati tako, da vsi popravki, vstavljanja in brisanja spremenijo tako tabelo super tipa kot tudi pripadajočo tabelo podtipa
  - uporabiti je potrebno omejitev cascade za tuje ključe
  - popravek ključa supertipa zahteva popravek v hierarhiji
  - popravek opisnega atributa zahteva popravek samo enega zapisa
- Praktični sistemi velkokrat uporabijo klasifikacijski atribut v nadtipu s katerim lahko ločimo med instancami podtipov
- Če imamo več-nivojsko hierarhijo potem lahko uporabimo več klasifikacijskih atributov
- Pristop kombinira sistem tipov s klasifikacijsko hierarhijo definirano glede na vrednost izbranih atributov
- Transformacijska pravila so ista za primer prekrivanja kot tudi za disjunktne podtipe
- Drugi pristop je definicija ene same tabele, ki vsebuje vse atribute tako podtipa kot tudi nadtipa
  - celotna hierarhija v eni tabeli
  - po potrebi uporabimo null vrednosti
- Tretji pristop je ena tabela za en podtip vendar porinemo skupne atribute navzdol k tabelam podtipov
- Imamo prednosti in slabosti teh treh pristopov
- Programska orodja za načrtovanje navadno uporabljajo vse tri možnosti

**6) Kako modelirati združevanje entitet? Oz. Prevajanje ER diagramov z agregacijo (združevanjem). Pdf. 119**

Agregacija nam omogoča, da obravnavamo množice razmerij kot entitetne množice, ki lahko sodelujejo v drugih razmerjih.

Obstaja poseben primer, v katerem je mogoče ta prevod izboljšati s spuščanjem razmerja Sponzorji. Razmislite o odnosu sponzorjev. Ima atribute pid, did in since; in na splošno ga potrebujemo (poleg monitorjev) iz dveh razlogov:

- Zapisati moramo opisne atribute (v našem primeru, od takrat) razmerja Sponzorji.
- Vsako sponzorstvo nima monitorja, zato nekateri (pid, did) pari v razmerju sponzorjev morda ne bodo prikazani v razmerju Monitorji

Vendar, če sponzorji nimajo opisnih atributov in imajo popolno udeležbo v monitorjih, se lahko vsak možen primer razmerja sponzorjev dobi iz stolpcev (pid, did) monitorjev; Sponzorje lahko opustimo.

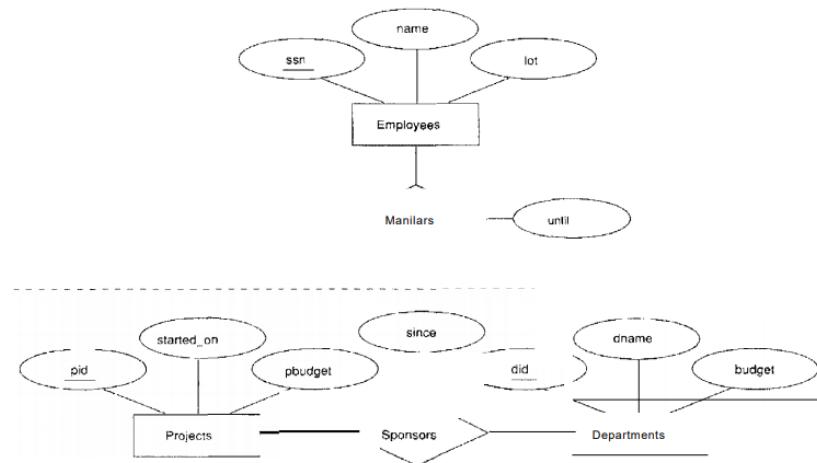


Figure 3.16 Aggregation

## Agregacija

- Prevod agregacijske hierarhije tudi vodi do ločenih tabel za ločene podtipe
- Agregacijo obravnavamo kot razmerje in uporabimo enaka pravila za prevajanje v tabele
- V primeru agregacije nimamo skupnih atributov in integritetnih omejitev, ki jih je potrebno vzdrževati
- Glavna funkcija agregacije je definicija abstrakcije, ki lahko pomaga pri integraciji schem
- V UML predstavlja agregacija relacijo kompozicije, ki ustreza šibkim entitetam

**7) Predstavite pravila za prevajanje modela ER v model relacijske baze podatkov.**

<http://osebje.famnit.upr.si/~savnik/predmeti/OPB/Prevod-SQL.pdf>

/\*mislim da je to to!\*/

1. Prevajanje entitet
2. Pretvorba razmerij
3. Rekurzivna razmerja
4. Ternarna razmerja
5. Generalizacija
6. Agregacija

## 4 DISKI IN DATOTEKE

### 1) Opišite arhitekturo in uporabo diskov. Str.307 -> Pdf. 341

Magnetni disk podpirajo direkten dostop do želene lokacije in so pogosto uporabljeni za aplikacije baze podatkov. SUPB zagotavlja nemoten dostop do podatkov na disku; programu ni treba skrbeti ali so podatki v glavnem pomnilniku ali na disku. **READ** je prenos med trdim diskom in RAM-om. **WRITE** pa prenos med RAM in diskom. Torej **READ** iz diska v RAM, **WRITE** pa obratno.

YT: [DELOVANJE DISKA](#)

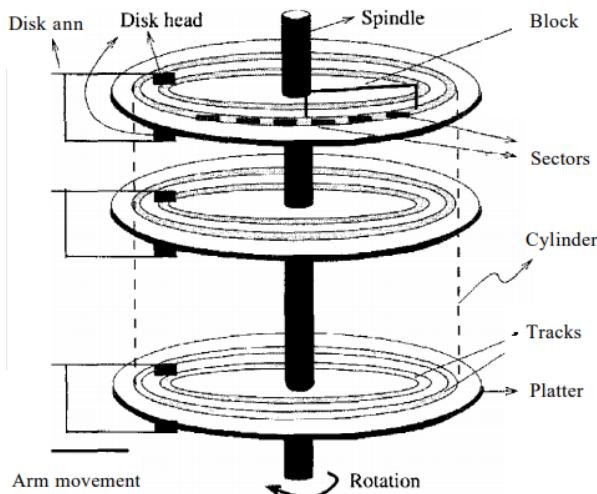


Figure 9.2 Structure of a Disk

### 2) Predstavite kako so SUPB podatki organizirani na diskih. (delal po prosojnicah)

SUPB shranjuje podatke na trde diske. To ima veliko posledic na zasnovo SUPB. Poznamo READ in WRITE operaciji, ki sta časovno potratni, zato se more njihova uporaba planirati pazljivo. Vsega se ne shrani v dinamičen spomin zaradi cene saj je RAM dražji od trtega diska in predvsem zaradi tega, ker dinamičen spomin ne ohranja podatkov. Želimo imeti shranjene podatke med večjimi sejami s SUPB. Podatki se shranjujejo in prenašajo v enotah, ki jih imenujemo diskovni bloki ali strani. Čas potreben za dostop do podatkov je odvisen od lokacije podatkov na disku. Seveda ima položaj podatkov na disku vpliv na hitrost delovanja SUPB.

Tipična hierarhija pomnilnikov:

- Dinamični spomin (RAM) za direktno delo s podatki.
- Disk za glavno podatkovno bazo.
- Trakovi za arhiviranje starejših verzij podatkov.

### 3) Predstavite glavna načela diskov v RAID - u. Str. 309 -> Pdf. 344

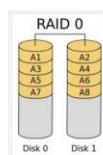
Diskovni nizi (disk array), ki izvajajo kombinacijo razpršenosti podatkov in redundance, se imenujejo redundantni nizi neodvisnih diskov ali na kratko RAID. Cilj RAID – a je povečati zmogljivost in zanesljivost.

Dve osnovni tehniki:

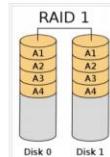
- Podatkovni pasovi (strips): Podatki so razdeljeni po diskih po pasovih; velikost pasa se imenuje "pasovna enota".
- Redundanca: Več diskov => več napak. Redundantna informacija omogoča rekonstrukcijo podatkov, če se disk pokvari.

## RAID Nivoji

- Nivo 0: Ni redundance.
  - Porazdeljeni podatki po pasovih
  - Ni redundance, paritete
  - Hitrost dostopa je glavni razlog

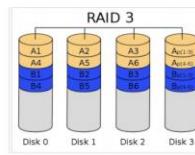


- Nivo 1: Zrcaljenje (dve identični kopiji).
  - Vsak disk ima zrcalno kopijo.
  - Paralelno branje; pisanje deluje nad dvemi diskami.
  - Pohitritev branja = št. zrcaljenih diskov X

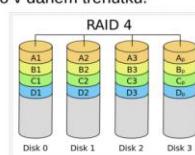


## RAID Nivoji

- Nivo 3: Pariteta "po bitih".
  - Pasovna enota: en zlog (byte)
  - En disk za paritet
  - Vsako pisanje/branje vključuje vse diske
  - Diskovno polje lahko procesira eno zahtevo v danem trenutku.

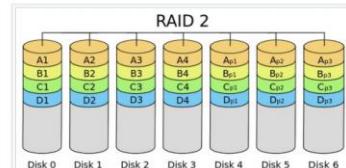


- Nivo 4: Pariteta "po blokih".
  - Pasovna enota: En diskovni blok
  - Eden disk za paritet
  - Paralelno branje za manjše zahteve, večje zahteve lahko uporabijo poln pretok.
  - Pisanje vključuje spremenjen blok in disk za preverjanje.



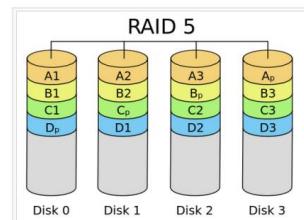
## RAID Nivoji

- Nivo 0+1: Pasovi in zrcaljenje.
  - Podatki se po bitih razporedijo po diskih.
  - Vzporedno branje; pisanje deluje nad dvemi diskami.
  - Maksimalen prenos = skupen pretok.
  - Redko se uporablja



## RAID Nivoji

- Nivo 5: Porazdeljena pariteta "po blokih"
  - Podobno RAID 4; paritetni blok je porazdeljen po večih diskih



## 4) Predstavite hierarhijo pomnilnika v SUPB. Str. 316 -> Pdf. 351

Najnižja raven programske opreme v SUPB arhitekturi se imenuje "space manager" in opravlja s prostorom na disku. Upravitelj prostora na disku podpira koncept strani kot enoto podatkov in zagotavlja ukaze za dodelitev ali razrešitev strani, ter branje ali pisanje strani. Velikost strani je izbrana tako, da je velikost disk bloka, strani pa so shranjene kot disk bloki, tako da je branje ali pisanje strani mogoče na enem diskusu I/O. Pogosto je koristno dodeliti zaporedje strani kot sosednje zaporedje blokov, da se podatki hranijo pogosto v zaporednem vrstnem redu. Ta zmogljivost je bistvena za izkoriščanje prednosti zaporednega dostopa do diskovnih blokov. Takšno zmožnost, če jo želimo, mora upravitelj prostora na diskusu zagotoviti na višjih nivojih SUPB. Upravitelj prostora na diskusu skriva podrobnosti o osnovni strojni opremi (in morda operacijskem sistemu) in omogoča višjim nivojem programske opreme, da podatke obravnavajo kot zbirko strani.

## 5) Opišite funkcijo vmesnega pomnilnika SUPB ("buffer pool"). Str. 318 -> Pdf. 353

Če želite razumeti vlogo upravitelja medpomnilnika, razmislite o preprostem primeru. Recimo, da baza podatkov vsebuje 1 milijon strani, vendar je na voljo samo 1000 strani glavnega pomnilnika za shranjevanje podatkov. Razmislite o poizvedbi, ki zahteva skeniranje celotne datoteke. Ker vseh podatkov ni mogoče vnesti v glavni pomnilnik naenkrat, mora SUPB vnesti strani v glavni pomnilnik tako, kot je potrebno, in se pri tem odločiti, katera obstoječa stran v glavnem pomnilniku bo zamenjana, da bo prostor za novo stran. Pravilnik, ki se uporablja za odločitev, katero stran naj nadomesti, se imenuje nadomestni program.

V smislu arhitekture SUPB je upravljalnik medpomnilnika programska plast, ki je odgovorna za vnos strani iz diska v glavni pomnilnik, kot je potrebno. Upravljalnik medpomnilnika upravlja razpoložljivi glavni pomnilnik tako, da ga razdeli v zbirko strani, ki jih skupaj imenujemo področje vmesnega pomnilnika. Glavne pomnilniške strani v področju medpomnilnika se imenujejo okvirji; primerno je, da jih smatramo kot režo, ki lahko vsebuje stran (ki je običajno na disku ali drugem pomnilniškem mediju).

Višje ravni kode SUPB lahko zapišemo brez skrbi, ali so podatkovne strani v spominu ali ne; vprašajo upravitelja medpomnilnika za stran in ga vnesejo v okvir v področju medpomnilnika, če še ni tam. Seveda mora tudi koda višjega nivoja, ki zahteva stran, sprostiti stran, ko ni več potrebna, tako da o tem obvesti upravitelja medpomnilnika, tako da se okvir, ki vsebuje stran, lahko ponovno uporabi. Koda višje ravni mora tudi obvestiti upravitelja medpomnilnika, če spremeni zahtevano stran; Upravljalnik medpomnilnika se nato prepriča, da se sprememba prenese na kopijo strani na disku.

Poleg samega vmesnega pomnilnika upravljalnik medpomnilnika vzdržuje tudi nekatere knjigovodske informacije in dve spremenljivki za vsak okvir v bazenu: *pinLcount* in *dirty*. Število primerov, v katerih je bila trenutna stran v določenem okviru zahtevana, vendar ne izpuščena - število trenutnih uporabnikov strani - se zabeleži v spremenljivki *pin\_count* za ta okvir. Boolova spremenljivka *dirty* označuje, ali je bila stran spremenjena, ker je bila prenesena v medpomnilnik iz diska.

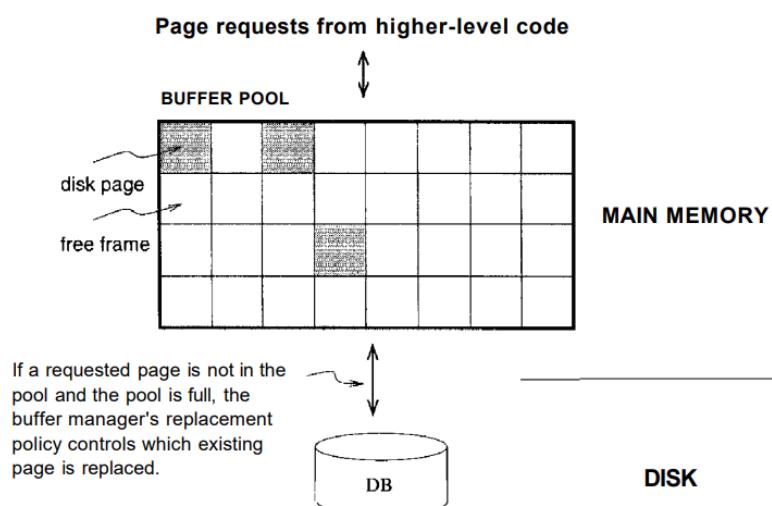
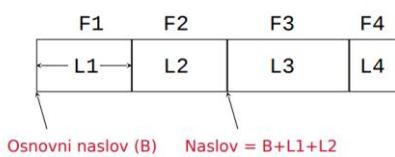


Figure 9.3 The Buffer Pool

## 6) Kakšne so možne izvedbe zapisov? Str. 326 -> Pdf. 361

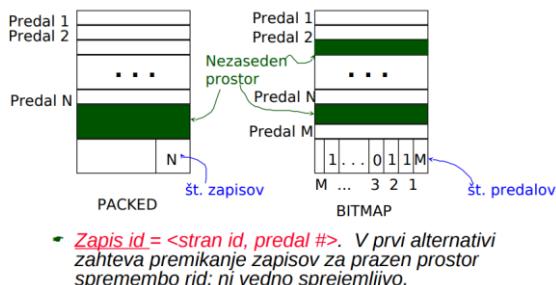
Zdaj pa pozornost usmerimo na način, na katerega so strani shranjene na disku in v glavni spomin, na način, na katerega se strani uporabljajo za shranjevanje zapisov in organizirane v logične zbirke ali datoteke. Višje ravni kode SUPB obravnavajo stran kot učinkovito zbirko zapisov, ne upoštevajo pa predstavitve in podrobnosti shranjevanja. Pravzaprav koncept zbirke zapisov ni omejen na vsebino ene same strani; datoteka lahko obsega več strani. V tem razdelku preučimo, kako lahko zbirko strani organiziramo kot datoteko. Stran lahko smatramo kot zbirko slotov, od katerih vsaka vsebuje zapis. Zapis se identificira z uporabo para (id strani, številke slotov); to je id zapisa (rid). Sedaj obravnavamo nekaj alternativnih pristopov za upravljanje slotov na strani. Glavni vidiki so, kako ti pristopi podpirajo operacije, kot so iskanje, vstavljanje ali brisanje zapisov na strani.

### Format zapisa: Fiksna dolžina



- Informacija o tipih polja je enaka za vse zapise v datoteki; shranjuje se v *sistemskem katalogu*.
- Poišči *i-to* polje ne zahteva pregled vseh zapisov.

### Format strani: Zapis fiksne dolžine



- Zapis id = <stran id, predal #>*. V prvi alternativi zahteva premikanje zapisov za prazen prostor spremembo rid; ni vedno sprejemljivo.

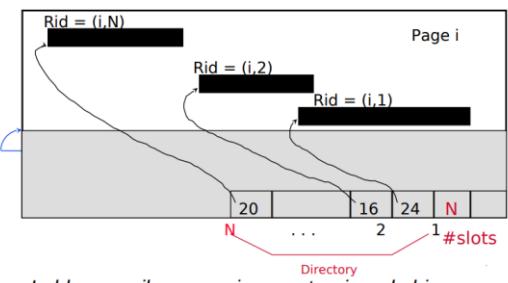
### Format zapisa: Variabilna dolžina

- Dva alternativna formata (# polj je fiksno):



- Druga alternativa ponuja direkten dostop do i-tega polj; učinkovito shranjevanje *null*; dir. ne zaseda veliko prostora.

### Format strani: Zapis variabilne dolžine



- Lahko premikamo zapis po strani ne da bi spremenili rid; privlačna predstavitev tudi za zapis s fiksno dolžino.

## 7) Kako je tabela shranjena v SUPB?

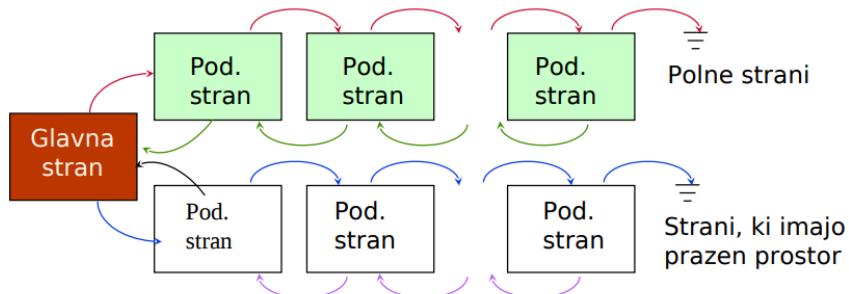
### Datoteke zapisov

- Stran ali blok je OK za I/O
- Višji nivoji SUPB delujejo z *zapisi* ter z *datotekami zapisov*
- DATOTEKA:** Zbirka strani; vsaka stran vsebuje množico zapisov. Operacije:
  - insert/delete/modify - zapis
  - read – zapis (uporaba *rid*)
  - pregled (scan) vseh zapisov (pogoji nad zapisi, ki naj jih sistem vrne)

# Neurejene datoteke

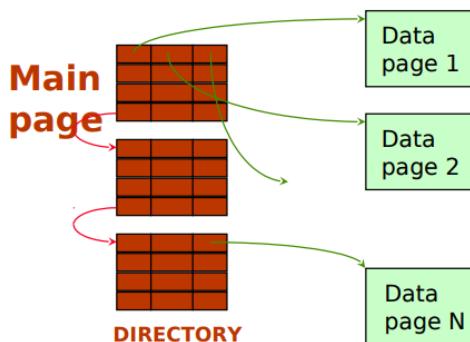
- Najenostavnejša datotečna struktura
  - Zapisi nimajo nobene urejenosti.
  - Tako kot se datoteka manjša in veča, tako se dodajajo in odvzemajo strani (bloki).
- Potrebno je shranjevati podatke o:
  - straneh datoteke
    - *neuporabljenem prostoru* na straneh
    - *zapisih* na strani
- Obstaja veliko alternativ za shranjevanje predstavljenih podatkov.

## Neurejena datoteka implementirana s seznamom



- ID glavne strani in sama datoteka morata biti shranjena nekje na disku.
- Vsaka stran vsebuje 2 `kazalca` in podatke.

## Neurejena datoteka z direktorijem strani



- Zapis pod. strani na glavni strani vsebuje lahko tudi št. prostih zlogov na pod. strani.
- Direktorij je zbirka strani; implementacija z seznamov je ena alternativa.
  - *Seznam glavnih strani je precej manjši kot št. vseh strani!*

## **5 INDEKSI**

### **1) Poimenujte in opišite nekaj zunanjih naprav za shranjevanje. Str. 274-> Pdf. 309**

- Diski so najpomembnejše zunanje naprave za shranjevanje podatkov. Omogočajo nam, da poiščemo katero koli stran na (več ali manj) fiksnih stroških na stran. Če pa preberemo več strani v vrstnem redu, v katerem so fizično shranjene, je lahko strošek veliko manjši od stroškov branja istih strani v naključnem vrstnem redu.
- Trakovi so naprave za zaporedni dostop in nas prisilijo, da preberemo podatke za eno stran za drugo. Večinoma se uporabljajo za arhiviranje podatkov, ki niso redno potrebni.
- Vsak zapis v datoteki ima enolični identifikator, imenovan ID zapisu, ali RID za krajše. RID ima lastnost, da lahko z uporabo RID prepoznamo naslov diska strani, ki vsebuje zapis.

### **2) Kaj je organizacija datotek? Kakšne alternative imamo? Str. 275 -> Pdf. 310**

Indeks je podatkovna struktura, ki organizira zapise podatkov na disku za optimizacijo nekaterih vrst pridobivanja operacij. Indeks nam omogoča učinkovito pridobivanje vseh zapisov, ki izpolnjujejo pogoje iskanja na poljih za iskanje po indeksu. Ustvarjamo lahko tudi dodatne indekse za dano zbirko podatkovnih zapisov, vsak z drugačnim iskalnim ključem, da pospešimo operacije iskanja, ki jih organizacija datotek, ki se uporablja za shranjevanje podatkovnih zapisov, ne podpirajo učinkovito.

Izraz vnos podatkov uporabljamo za sklicevanje na zapise, shranjene v indeksni datoteki. Vnos podatkov z vrednostjo ključa za iskanje k, označen kot k \*, vsebuje dovolj informacij za lociranje (enega ali več) podatkovnih zapisov z vrednostjo ključa za iskanje k. Lahko učinkovito iščemo indeks, da bi našli želene vnose podatkov, in jih nato uporabimo za pridobitev podatkovnih zapisov (če se ti razlikujejo od podatkovnih vnosov).

Obstajajo tri glavne možnosti za shranjevanje podatkov v indeksu:

1. Vnos podatkov k \* je dejanski podatkovni zapis (s tipko za iskanje K).
2. Vnos podatkov je par (k, rid), kjer je rid id zapisa podatkovnega zapisa z vrednostjo ključa za iskanje k.
3. Vnos podatkov je par (k, rid-list), kjer je rid-list seznam ID-jev zapisov podatkovnih zapisov z vrednostjo ključa za iskanje k.

### **Alternativne datotečne organizacije**

Obstaja več alternativ, vsaka je *idealna za določene situacije in ne tako dobra v drugih primerih:*

- **Neurejene datoteke:** Primerne, ko je tipični dostop branje celotne datoteke.
- **Sortirane datoteke:** Najboljše v primer, da iščemo zapise v določenem vrstnem redu ali želimo preiskati določen interval celotne množice zapisov.
- **Indeksi:** Podatkovna struktura, ki organizirajo zapise na osnovi dreves in razpršilnih funkcij.
  - Pohitrijo iskanje zapisa z dano vrednostjo iskalnega ključa.
  - Podobno sortiranim datotekam, pohitrijo iskanje podmnožice zapisov na osnovi določenih iskalnih ključev.
  - Popravljanje zapisov je mnogo hitrejše kot v primeru sortiranih datotek.

### **3) Kaj je indeks? Opišite pojme ključ za iskanje, vnos podatkov, vnos indeksa in podatkovni zapis. Str. 275 -> Pdf. 310 + poglej še prosojnice 5 do 8 Indeksi**

Indeks je podatkovna struktura, ki organizira zapise podatkov na disku za optimizacijo nekaterih vrst pridobivanja operacij. Indeks nam omogoča učinkovito pridobivanje vseh zapisov, ki izpolnjujejo pogoje iskanja na poljih za iskanje po indeksu. Ustvarjamo lahko tudi dodatne indekse za dano zbirkovo podatkovnih zapisov, vsak z drugačnim iskalnim ključem, da pospešimo operacije iskanja, ki jih organizacija datotek, ki se uporabljajo za shranjevanje podatkovnih zapisov, ne podpirajo učinkovito.

Izraz vnos podatkov uporabljamo za sklicevanje na zapise, shranjene v indeksni datoteki. Vnos podatkov z vrednostjo **ključa za iskanje k**, označen kot  $k^*$ , vsebuje dovolj informacij za lociranje (enega ali več) podatkovnih zapisov z vrednostjo ključa za iskanje  $k$ . Lahko učinkovito iščemo indeks, da bi našli želene vnose podatkov, in jih nato uporabimo za pridobitev podatkovnih zapisov (če se ti razlikujejo od podatkovnih vnosov).

1. Vnos podatkov  $k^*$  je dejanski podatkovni zapis (s tipko za iskanje  $K$ ).
2. Vnos podatkov je par  $(k, \text{rid})$ , kjer je  $\text{rid}$  id zapisa podatkovnega zapisa z vrednostjo ključa za iskanje  $k$ .
3. Vnos podatkov je par  $(k, \text{rid-list})$ , kjer je  $\text{rid-list}$  seznam ID-jev zapisov podatkovnih zapisov z vrednostjo ključa za iskanje  $k$ .

Seveda, če se indeks uporablja za shranjevanje dejanskih podatkovnih zapisov, alternativa (1), je vsak **vnos  $k^*$  podatkovni** zapis s ključem za iskanje  $k$ . Na takšen index lahko gledamo kot posebno datotečno organizacijo. Takšno indeksirano datotečno organizacijo lahko uporabite namesto, na primer, razvrščene datoteke ali neurejene datoteke zapisov.

Alternative (2) in (3), ki vsebujejo vnose podatkov, ki kažejo na podatkovne zapise, so neodvisne od organizacije datotek, ki se uporablja za indeksirano datoteko (to je datoteka, ki vsebuje podatkovne zapise). Alternativa (3) ponuja boljšo izkoriščenost prostora kot alternativa (2), vendar so vnoси podatkov spremenljivi po dolžini, odvisno od števila podatkovnih zapisov z dano vrednostjo ključa iskanja.

### **4) Katere so alternative za vnos podatkov $k^*$ ?**

### **5) Kaj je primarni/sekundarni index? Kaj je clustered / unclustered index? Str. 277 -> Pdf. 312**

Indeks na nizih polj, ki vključuje primarni ključ, se imenuje primarni indeks; drugi indeksi se imenujejo sekundarni indeksi.

Dva vnosa podatkov naj bi bila podvojena, če imata enako vrednost za polje za iskanje ključev, povezano z indeksom. Primarni indeks **NE** vsebuje dvojnikov, vendar lahko indeks na drugih (zbirkah) polj vsebuje dvojnice. Na splošno sekundarni indeks vsebuje

dvojnice. Če vemo, da ni dvojnikov, to pomeni, da vemo, da iskalni ključ vsebuje nekaj kandidatnih ključev, indeks imenujemo edinstven indeks.

Ko je datoteka organizirana tako, da je urejanje podatkovnih zapisov enako ali blizu vrstnemu redu vnosov podatkov v nekem indeksu, pravimo, da je indeks združen(clustered); v nasprotnem primeru je združen(clustered) indeks neklasificiran. Indeks, ki uporablja alternativo (1), je po definiciji združen v gruče(clustered). Indeks, ki uporablja alternativo (2) ali (3), je lahko združen(clustered) indeks samo, če so zapisi podatkov razvrščeni na polju za iskanje ključev. V nasprotnem primeru je vrstni red zapisa podatkov naključen, opredeljen zgolj po njihovem fizičnem vrstnem redu in ni razumnega načina, da se vnosi podatkov v indeksu uredijo v istem vrstnem redu.

V praksi se datoteke redko razvrščajo, ker je to predrago za vzdrževanje, ko so podatki posodobljeni. Tako je v praksi združen(clustered) indeks, ki uporablja alternativo (1), in indeksi, ki uporabljajo alternative (2) ali (3), niso združeni(unclustered). Včasih se indeks uporablja z alternativo (1) kot združena(clustered) datoteka, ker so vnosi podatkov dejanski zapisi podatkov, zato je indeks datoteka zapisov podatkov.

## 6) Predstavite indeks ISAM. Str. 341 -> Pdf. 376 POSNETEK NA: [YT](#)

Podatki indeksa ISAM so na listnih straneh drevesa in dodatne prelivne strani, vezane na stran z nekaj listi. Sistemi baz podatkov skrbno organizirajo postavitev strani, tako da meje strani tesno ustrezajo fizičnim značilnostim osnovne naprave za shranjevanje. Struktura ISAM je popolnoma statična (razen prelivnih strani, za katere se pričakuje, da jih bo malo) in olajša takšne optimizacije na nizki ravni. Vsako drevesno vozlišče je diskovna stran, vsi podatki pa se nahajajo na listnih straneh.

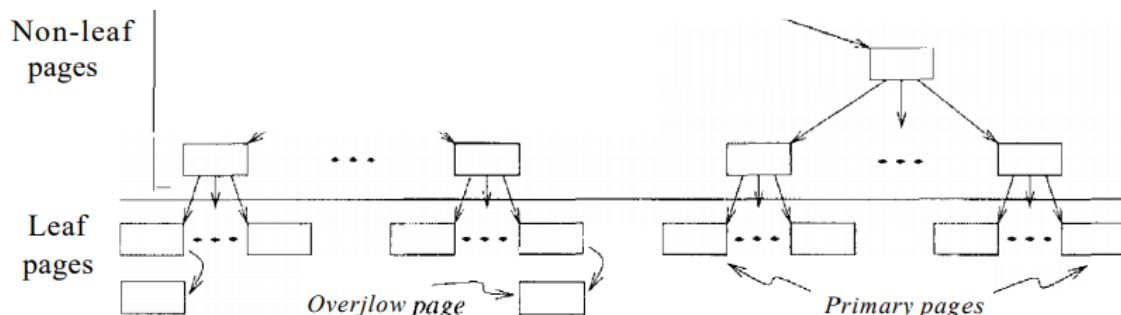


Figure 10.3 ISAM Index Structure

## ISAM

- Kreiranje datoteke:
  - Liste zasežemo sekvenčno sortirane po iskalnem ključu;
  - Zasežemo indeksne strani iterativno po nivojih
  - Prostor za prelivne strani
- Indeksni vpisi:
  - <vrednost iskalnega ključa, id strani>
  - Usmerjajo iskanje podatkovnih vpisov, ki so v listih.

## ISAM

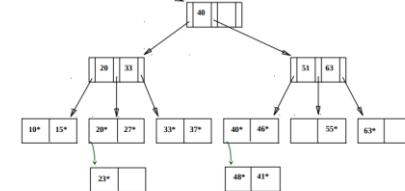
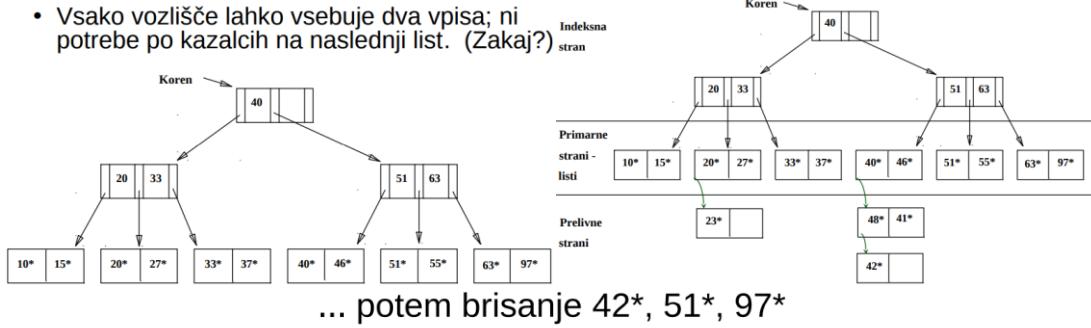
- Iskanje:
  - Začni v korenju
  - Uporabi primerjavo ključev za iskanje lista
  - Cena =  $\log_F N$
  - $F = \# \text{ vpisov}/\text{ind.stran}, N = \# \text{ listov}.$
- Vstavi:
  - Poisci list in zapisi pod. vpis
- Izbriši:
  - Poisci list in izbriši vpis iz lista; izbriši prelivno stran, če je prazna.

Pod. strani
Indeksne strani
Prelivne strani

## Primer ISAM drevesa

Po vstavljanju 23\*, 48\*, 41\*, 42\* ...

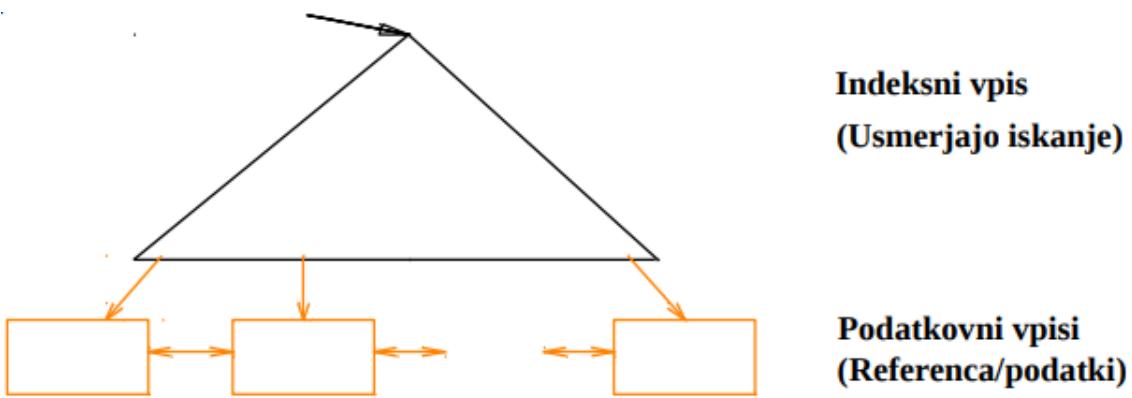
- Vsako vozlišče lahko vsebuje dva vpisa; ni potrebe po kazalcih na naslednjem listu. (Zakaj?)



► Vpis 51\* se pojavi v notranjih straneh in ne tudi v listih!

## 7) Predstavite indeks B+ drevesa. Str. 344 -> Pdf. 379

Statična struktura, kot je indeks ISAM, trpi zaradi težave, ki jo lahko razvijejo dolge overflow verige, ko se datoteka poveča, kar vodi do slabe učinkovitosti. Ta težava je spodbudila razvoj bolj prilagodljivih, dinamičnih struktur, ki se elegantno prilagajajo vstavkom in brisanjem. Struktura iskanja B + drevesa, ki je široko razširjena, je uravnoteženo drevo, v katerem notranja vozlišča usmerjajo iskanje, listna vozlišča pa vsebujejo vnose podatkov. Ker drevesna struktura raste in se dinamično skrči, ni mogoče dodeliti listnih strani zaporedoma kot v ISAM, kjer je bil set primarnih listnih strani statičen. Za učinkovito pridobivanje vseh listnih strani jih moramo povezati s kazalci strani. Z njihovo organizacijo v dvojno povezan seznam lahko preprosto prečkamo zaporedje listnih strani (včasih imenovanih zaporedje zaporedij) v obe smeri.



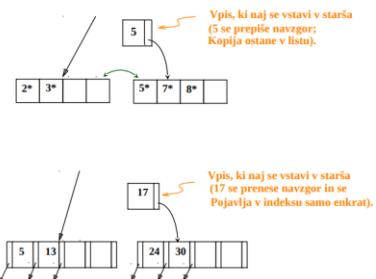
## 8) Opišite vstavljanje in brisanje operacije B+ drevesa. Str 348 -> Pdf. 383

### Vstavljanje pod. vpisa v B+ drevo

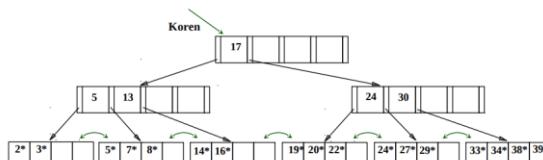
- Poišči list  $L$ .
- Vstavi pod. vpis v  $L$ .
  - Če ima  $L$  zadost prostora, je narejeno!
  - Sicer je potrebno razcepiti  $L$  (v  $L$  in novo vozlišče  $L2$ )
    - Enakomerno porazdeli vpise in prepisi srednji ključ gor.
    - Vstavi indeksni vpis, ki kaže na  $L2$ , v staršo od  $L$ .
- To se lahko zgodi rekurzivno
  - Razcep indeksnega vozlišča: enakomerno porazdeli vpise in prenesi srednji ključ gor. (Razlika z razcepom lista.)
- Razcepi "širijo" drevo; razcep korena zviša drevo.

### Vstavljanje 8\* v B+ drevo (primer)

- Minimalna zasedenost po razcepitvi je zajamčena v listih kot tudi v indeksnih straneh.
- Pogled razliko med prepisom in prenosom ključa navzgor.
- Kakšen je razlog za razliko?



### Primer B+ drevesa po vnosu 8\*

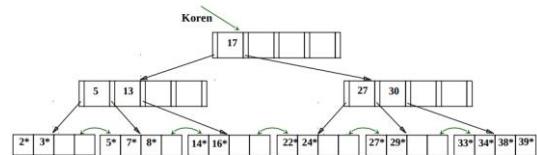


- Vidimo, da se je razcepil koren in se je povečala višina drevesa.
- V tem primeru bi se lahko izognili razcepu strani z porazdelitvijo vpisov; to se običajno ne uporablja v praksi.

### Brisanje pod. vpisa iz B+ drevesa

- Začni pri korenu in poišči list z vpisom.
- Izberi vpis.
  - Če je  $L$  vsaj polovico poln potem končaj!
  - Če vsebuje  $L$  d-1 vpisov,
    - Poskusimo porazdeliti vpise tako, da si jih sposodimo od sosedov.
    - Če porazdelitev ne uspe, zlij  $L$  in soseda.
- V primeru zlitja moramo zbrisati vpis (ki kaže na  $L$  ali na soseda) iz starša od  $L$ .
- Zlivanje se lahko nadaljuje vse do korena; v tem primeru se zniža višina drevesa.

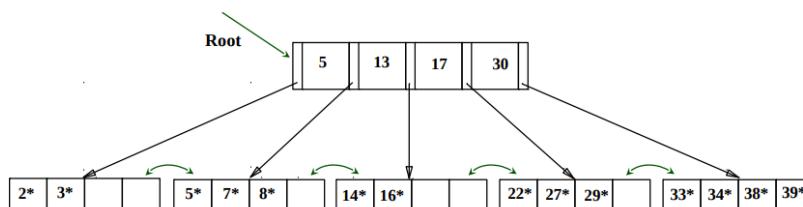
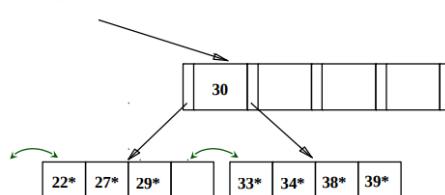
### Drevo po vstavljanju 8\* in brisanju 19\* in 20\* ...



- Brisanje 19\* je enostavno.
- Brisanje 20\* je narejeno s porazdelitvijo vpisov. Srednji ključ je prepisan navzgor.

### ... in brisanje 24\*

- Potrebno zlivanje.
- Zlitrje podatkovnih vpisov na desni ter indeksnih vpisov spodaj.

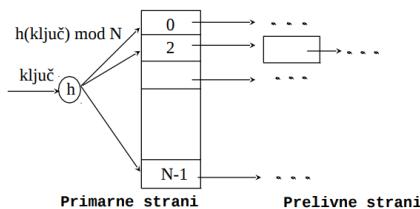


## 9) Opišite indeksi, ki temeljijo na razpršitvi. Katere so alternative? Str. 370 -> Pdf. 405

Razpršilni indeksi so dobri za iskanje z enačajem in se jih ne uporablja za iskanje področja. Poznamo statične in dinamične vrste indeksa.

### Statični razpršilni indeks

- # primarne strani so fiksne, zasežene sekvenčno ter niso nikoli sproščene.
- $h(k) \bmod N$  = skupek h kateremu pripada podatkovni vpis z ključem k. ( $N = \# \text{skupkov}$ )



### Razširljiv razpršilni indeks

- Dinamični razpršilni indeks
- *Situacija:* Skupek (primarna stran) se napolni.
  - Branje in pisanje strani skupka je potratno.
- *Ideja:* direktorij kazalcev na skupke
  - Lahko prestavljamo strani skupkov.
  - Število skupkov se podvoji s podvojitvijo direktorija.
  - Razcepimo samo skupek (stran), ki je napolnjen.
  - Direktorij je veliko manjši kot celotna datoteka zato se podvojitev izvrši hitro. *Ni prelivnih strani!*
  - Prilagoditi je potrebno še razpršilno funkcijo!

### Linear Hashing (Contd.)

- Izognemo se direktoriju z uporabo prelivnih strani in s cikličnim razceplanjem 0
  - Razceplanje potek v 'rundah'. Runda R se neha, ko so vsi začetni skupki ( $N_R$ ) razcepljeni.
    - Skupki 0 do *Next-1* so bili razcepljeni
    - Skupki *Next* do  $N_R$  morajo še biti razcepljeni
  - Trenutna runda je določena z Level.
  - *Iskanje:* Da bi poiskali skupek za  $r$ , poišči  $h_{\text{Level}}(r)$ :
    - Če  $h_{\text{Level}}(r)$  je v področju 'Next to  $N_R$ ', smo našli ...
    - Sicer, r lahko pripada skupku  $h_{\text{Level}}(r)$  ali skupku  $h_{\text{Level}}(r) + N_R$ ;
      - Aplicirati je potrebno  $h_{\text{Level}+1}(r)$ , da bi izvedeli
- **Insert:** Poišči skupek z uporabo  $h_{\text{Level}} / h_{\text{Level}+1}$ :
  - Če je skupek poln:
    - Dodaj prelivno strang in vstavi podatek
    - (Mogoče) Razcepi naslednji (*Next*) skupek in povečaj *Next*
  - Izberemo lahko katerikoli kriterij za proženje razcepa
  - Ker so skupki razcepljeni ciklično (round-robin) se ne morejo razviti dolge prelivne verige!
  - Podvojevanje direktorija v razširljivem razprševanju je podobno;
    - preklop med razpršilnimi funkcijami je *impliciten* v pregledu in uporabi # bitov

### LH je varianta EH

- Shemi sta precej podobni
  - Začni z EH indeksom, ki ima direktorijih z  $N$  elementi
  - Uporabi prelivne strani in razcepi skupke ciklično
  - Najprej razcepi skupek 0
    - "podvojevanje direktorija" v EH
      - Elementi  $<1, N+1>, <2, N+2>, \dots$
      - Ko se skupek razcepi, kreiraj element  $N+1$ , itd.
- Direktorij se podvojuje postopoma
  - Primarni skupki se kreirajo fizično zaporedno na disku
    - Alociramo jih sekvenčno
    - Ne potrebujemo direktorij

### Statični razpršilni indeks

- Skupki vsebujejo podatkovne vpise.
- Razpršilna fn se uporabi na *iskalnem ključu*. Vrednosti se morajo razprtiti na interval 0 ... N-1.
  - $h(k) = (a * k + b)$  običajno deluje dobro.
  - a in b so konstante; veliko je znano o tem kako umeriti  $h$ .
- Razvije se lahko dolg seznam prelivnih strani kar poslabša učinkovitost podatkovne strukture.
  - *Dinamični razpršilni indeks* reši problem.

### Linearni razpršilni indeks

- Dinamični razpršilni indeks
  - Alternativa razširljivem indeksu
- LH reši problem z dolgimi prelivnimi vrstami
  - Brez uporabe direktorija
  - Delo z duplikati
- *Ideja:* Uporabi družino razpršilnih funkcij  $h_0, h_1, h_2, \dots$ 
  - $h_i(\text{key}) = h(\text{key}) \bmod (2^i N)$ ;  $N = \text{initial } \# \text{ buckets}$
  - $h$  je neka razpršilna funkcija (zaloga vrednosti ni 0 do N-1)
  - Če  $N = 2^{\text{do}}$ , za nek  $d0$ ,  $h_i$  aplicira  $h$  in gleda zadnjih  $di$  bitov, kjer  $di = d0 + i$ .
  - $h_{i+1}$  podvoji zalogu vrednosti  $h_i$  (podobno podvojevanju direktorija)

### Linearno razprševanje

- *Insert:* Poišči skupek z uporabo  $h_{\text{Level}} / h_{\text{Level}+1}$ :
  - Če je skupek poln:
    - Dodaj prelivno strang in vstavi podatek
    - (Mogoče) Razcepi naslednji (*Next*) skupek in povečaj *Next*
- Izberemo lahko katerikoli kriterij za proženje razcepa
- Ker so skupki razcepljeni ciklično (round-robin) se ne morejo razviti dolge prelivne verige!
- Podvojevanje direktorija v razširljivem razprševanju je podobno;
  - preklop med razpršilnimi funkcijami je *impliciten* v pregledu in uporabi # bitov

## **6 OCENJEVANJE POIZVEDB**

### **1) Kaj je metoda dostopa? Kakšne metode dostopa poznate? Str. 398 -> Pdf. 433**

Dostopna pot je način zajemanja zapisov iz tabele in je sestavljen iz (1) skeniranja datotek ali (2) indeksa in ujemajočega izbirnega pogoja. Vsak relacijski operater sprejema eno ali več tabel kot vhodne podatke, metode dostopa, ki se uporabljajo za pridobivanje tuplov, pa znatno prispevajo k stroškom operaterja.

Razmislite o preprostem izboru, ki je konjunkcija pogojev form atr op vrednosti, kjer je op eden od primerjalnih operatorjev  $<$ ,  $\leq$ ,  $=$ ,  $\geq$ ,  $>$  ali  $\neq$ . Za take izbire se pravi, da so v konjunktivni normalni obliki (CNF), in vsak pogoj se imenuje konjunkt. Induktivno se indeks ujema z izbirnim pogojem, če je indeks mogoče uporabiti za pridobitev samo zapisov, ki izpolnjujejo pogoj.

### **Pogosto uporabljeni tehniki**

- Algoritmi za evaluacijo rel. operacij pogosto uporabljajo nekaj enostavnih idej:
  - **Indeksiranje:** Uporaba pogojev iz stavka WHERE za izbiro majhnega št. n-teric pri selekciji in stikih.
  - **Iteracija:** Včasih hitreje pregledamo vse n-terice čeprav je na razpolago indeks.
    - Včasih je koristno izvesti iteracijo po podatkovnih vpisih indeksa namesto na sami tabeli.
  - **Particije:** Velkokrat koristi razdeliti problem na več enakih delov – s tem zamenjamo izvajanje časovno potratnih operacij z podobnimi operacijami nad manjšim številom n-teric.

### **2) Opišite tehnike, ki se uporabljajo za vrednotenje relacijskih operacij. Str. 401 -> Pdf. 436**

Tehnike, ki se uporabljajo za vrednotenje relacijskih operacij so **selekcijska projekcija** in **stik.**

**Selekcija** je preprosto priklic korakov iz tabele. Če povzamemo, glede na izbiro oblike QRattr op vrednosti (R), če indeksa R.attr ni, moramo skenirati R. Verjetno je ceneje in bolj preprosto skenirati celotno tabelo (namesto da bi uporabili neklasificiran indeks), če je treba pridobiti več kot 5% oddelkov.

**Projekcija** zahteva od nas, da določena polja vnesemo, kar je enostavno. Dražji vidik operacije je zagotoviti, da se v rezultatu ne pojavi dvojniki. Na primer, če želimo le polja sid in bid ponudbe iz rezerv, bi lahko imeli dvojnice, če bi mornar rezerviral določeno ladjo več dni. Če dvojnikov ni treba odpraviti (npr. Ključna beseda DISTINCT ni vključena v člen SELECT), projekcija sestoji iz preprostega pridobivanja podmnožice polj iz vsake zaklepne tabele. To lahko dosežemo z enostavno ponovitvijo na tabeli ali indeksu, katerega ključ vsebuje vsa potrebna polja. Če moramo odstraniti dvojnice, moramo običajno uporabiti particoniranje. Recimo, da želimo pridobiti (sid, bid) s projektiranjem iz rezerve. Razdelimo lahko s (1) skeniranjem rezerv za pridobitev (sid, bid; parov in (2) razvrščanja teh parov z uporabo (sid, bid) kot ključa razvrščanja. Nato lahko pregledamo

razvrščene pare in preprosto zavрžemo dvojnice, ki so zdaj sosednji. Operacijo projekcije lahko optimizirate s kombiniranjem začetnega skeniranja rezervacij s skeniranjem v prvem prehodu razvrščanja. Podobno se lahko skeniranje razvrščenih parov združi z zadnjim prehodom razvrščanja.

**Stiki** so dragi in zelo pogosti. Zato so bili precej raziskani in sistemi običajno podpirajo več algoritmov za izvajanje stikov. Razmislite o pridružitvi rezerv in mornarjev z stičnim sklepanjem Reserves.sid = Sailors.sid. Recimo, da ima ena od tabel, recimo mornarji, indeks na stolpcu sid. Lahko skeniramo Rezerve in za vsako n-nico uporabimo indeks za Sailors za ujemanje tuples. Ta pristop se imenuje indeksne ugnezdenne zveze. Poznamo še: Stik z vgnezdeno zanko, Stik z drevesnim indeksom, Stik z vgnezdeno zanko po blokih, Stik z vgnezdeno zanko po vrstah, Stik z zlivanjem in Stik z razpršilnim indeksom.

**3) Predstavite splošni zunanji algoritem za razvrščanje (merge sort). Kakšna je kompleksnost zunanjega razvrščanja (merge sort)? Str. 458 -> Pdf. 493 [SPLETNA](#)**

Zunanje razvrščanje je izraz za razred algoritmov za razvrščanje, ki lahko obdelajo velike količine podatkov. Zunanje razvrščanje je potrebno, če se razvrščeni podatki ne uvrščajo v glavni pomnilnik računalniške naprave (običajno RAM) in se morajo nahajati v počasnejšem zunanjem pomnilniku (običajno trdi disk). Zunanje razvrščanje običajno uporablja hibridno strategijo sortiranja. V fazi razvrščanja se deli podatkov, ki so dovolj majhni, da se prilegajo v glavni pomnilnik, berejo, razvrščajo in izpisujejo v začasno datoteko. V fazi združevanja se razvrščene pod-datoteke združijo v eno večjo datoteko. Eden od primerov zunanjega razvrščanja je zunanji algoritem za razvrščanje, ki razvrsti dele, ki ustrezajo vsakemu pomnilniku RAM, in nato združi razvrščene dele. Najprej razdelimo datoteko na zagone, tako da je velikost teka dovolj majhna, da se prilega glavnemu pomnilniku. Nato razvrstite vsak zagon v glavnem pomnilniku z uporabo algoritma za razvrščanje sortiranja. Končno združite nastale tokove skupaj v zaporedno večje zapise, dokler datoteka ni razvrščena.

Kompleksnost zunanjega razvrščanja je kompleksna, saj če ne moremo shraniti vseh podatkov v pomnilnik jih lahko razčlenimo. Dolžina našega poteka je vezana na vašo razpoložljivo velikost vmesnega pomnilnika. -> Cena =  $2N * (\# \text{ prehodov})$

**4) Kako implementirati selekcijo? Str. 441 -> Pdf. 476**

V tem poglavju opisujemo različne algoritme za ocenjevanje izbirnega operaterja. To poizvedbo lahko ovrednotimo s skeniranjem celotnega razmerja, preverjanjem pogojev na vsakem krmilniku in dodajanjem tuple na rezultat, če je pogoj izpolnjen. Stroški tega pristopa so  $1000 l / Os$ , saj Rezerve vsebujejo 1000 strani. Če ima le nekaj tuples rname = 'Joe', je ta pristop drag, ker ne uporablja izbire za zmanjšanje števila nihanj, pridobljenih na kakršen koli način. Kako lahko izboljšamo ta pristop? Ključ je uporabiti informacije v izbirnem stanju in uporabiti indeks, če je na voljo ustrezen indeks. Na primer, indeks B + drevesa na rname bi se lahko uporabil za precej hitrejši odgovor na to poizvedbo, vendar indeks na ponudbi ne bi bil uporaben.

## Osnovni pristop

```
SELECT *
FROM Reserves R
WHERE R.rname='Joe'
```

Figure 14.1 Simple Selection Query

- **Algoritem:**
  - Poišči najbolj selektivne metode dostopa in z njimi poišči n-terice.
  - Ovrednoti preostale pogoje nad izbranimi n-tericami.
- **Selektivnost pogoja:**
  - Delež relacije, ki je rezultat selekcije z danim pogojem.
  - Čim bolj je selektiven pogoj manjši je rezultat selekcije.
- **Selektivnost metode dostopa:**
  - Metoda dostopa izraza za katerega ocenimo, da bo zahtevala najmanjše število prenosov blokov iz diska.
  - Selektivnost metode dostopa upošteva selektivnost pogoja ter samo metodo dostopa.

## Osnovni pristop

- **Komentarji:**
  - Izbera n-teric z najbolji selektivnimi potmi optimizira število izbranih n-teric
  - Preostali pogoji dodatno izberejo podmožico n-teric izbranih z najbolji selektivnimi izrazi.
  - Preostali pogoji ne vplivajo na št. strani prebranih iz diska.
- **Primer:**
  - Primer izraza: `dan<8/9/94 AND lid=5 AND mid=3`.
  - Lahko uporabimo B+ drevo za izraz "dan<8/9/94" – je najbolj selektiven.
  - Nato uporabimo pogoj "lid=5 and mid=3" za selekcijo iz izbranih z prejšnjim pogojem.
  - **Podobno:** lahko uporabimo razpršilni indeks na `<lid, mid>`; preveriti moramo še "dan<8/9/94".

## 5) Predstavite metode za implementacijo projekcije. Str. 447 -> Pdf. 482

### Projekcija

- **Primer:**

```
SELECT DISTINCT
R.mid, R.lid
FROM Rezervacije R
```
- **Enostavna rešitev brez izločitve duplikatov:**
  - Pregled vseh n-teric relacije in izbor atributov.
- **Najdražja operacija je odstranitev duplikatov.**
  - SUPB ne odstranjuje duplike, če ni specificirana ključna beseda DISTINCT v stavku SELECT.
- **Sortiranje:**
  - Sortiraj po `<mid, lid>` in odstrani duplike.
  - Optimizacija: odstrani nepotrebne podatke med sortiranjem.

### Projekcija

```
SELECT DISTINCT
R.mid, R.lid
FROM Rezervacije R
```

- ❖ **Uporaba zunanjega sortiranja:**
  - Spremeni sortiranje z zlivanjem tako, da se izločijo komponente zapisa.
    - Sortirne vrste so manjše kot vhodne n-terice; odvisno od velikosti polj.
  - Spremeni fazo združevanja tako, da se izločijo duplikati.
    - Število n-teric v rezultatu je manjše od vhoda (ni duplikatov)
  - **Cena:**
    - Cena sortiranja:  $O(M \log M)$
    - Preberi originalno relacijo velikost M in izpiši manjše n-terice.
    - Pri združevanju dobimo manjše število n-teric.

### Projekcija

- **Primer:**

```
SELECT DISTINCT
R.mid, R.lid
FROM Rezervacije R
```
- **Uporaba razpršilnega indeksa:**
  - S kreacijo razpršilnega indeksa nad `<mid, lid>` dobimo particije; funkcija `h1`.
  - Preberi vsako particijo v din. pomnilniku, sortiraj tabelo v din. pomnilniku ter eliminiraj duplike.
  - Če se particije prevelike rekurzivno ponovi postopek z razpršilno funkcijo `h2`.
- **Če že obstaja indeks z `R.mid` in `R.lid` v iskalnem ključu enostavno sortiramo podatkovne vpise.**
  - **Cena:** Branje in pisanje (projekcije) vseh n-teric.

## 6) Opišite stik z vgnezdeno zanko, stik z drevesnim indeksom in stik z vgnezdeno zanko po blokih. Str. 452 -> Pdf. 487

### Stik z vgnezdeno zanko

```
foreach tuple r in R do
    foreach tuple s in S do
        if ri == sj then add <r, s> to result
```

- Za vsako n-terico v zunanji relaciji pregledamo celotno notranje relacijo S.
- Delo z diskovnimi bloki:**
  - Za vsako stran R preberi vse strani relacije S in izpiši n-terice <r,s>, ki se ujemajo.
  - V primeru, da so podatkovni zapisi porazdeljeni po različnih straneh je ta pristop veliko hitrejši.

### Stik z indeksom

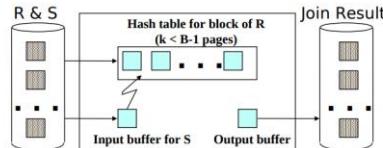
```
foreach tuple r in R do
    foreach tuple s in S where ri == sj do
        add <r, s> to result
```

- Če obstaja indeks na eni izmed relacij ga uporabimo za notranjo zanko.
  - Za vsako n-terico iz R z indeksom poiščemo n-terice, ki se ujemajo v S.
  - Cena:  $M + (M * p_R) * \text{cena iskanja n-teric iz S}$
- Cena za preverjanje vseake n-terice iz R je branje:
  - 1.2 strani v primeru razpršilnega indeksa,
  - 2-4 v primeru B+ drevesa.
  - Cena iskanja n-teric v S je v veliki meri odvisna od povezanosti S z relacijo – št. prebranih strani se precej zmanjša.
  - Povezan indeks: 1 V/I (tipično), nepovezan: do 1 V/I za eno S n-terico

### Vgnezdena zanka po vrstah

- Uporabi eno stran za vmesnik pri pregledu notranje relacije S, eno stran za izhodni vmesnik in uporabi preostale strani za vrsto blokov iz zunanje relacije R.

```
foreach vrsto B-2 blokov iz R
    foreach blok iz S
        vse pare <r, s> iz R vrste in S bloka, ki se ujemajo
        dodaj <r, s> k rezultatu
```



### Stik z vgnezdeno zanko

```
foreach tuple r in R do
    foreach tuple s in S do
        if ri == sj then add <r, s> to result
```

- Za vsako n-terico iz zunanje relacije pregledamo celotno notranje relacijo S.
  - Cena:  $M + p_R * M * N = 1000 + 100 * 1000 * 500 = 50.001.000 V/I$ .
- Vgnezdena zanka po blokih:**
  - Za vsako stran R preberi vsako stran S in izpiši spete pare n-teric <r, s>, kjer je r iz R-strani in s iz S-strani.
  - Cena:  $M + M * N = 1000 + 1000 * 500 = 501.000 V/I$ .
  - Če je manjša relacija zunanja, potem je cena = 500 +  $500 * 1000 = 500.500 V/I$ .

### Primeri stika z indeksom

- Razpršilni indeks (Alt. 2) na mid relacije Mornarji (notranja):**
  - Pregled Rezervacije: 1000 strani V/I,  $100 * 1000$  n-teric.
  - Za vsako n-terico Rezervacij: 1.2 V/I za branje podatkovnega vpisa + 1 V/I za izbrano n-terico Mornarjev.
  - Cena:  $1000 + 1000 * 100 * 2.2 = 221.000 V/I$ .
- Razpršilni indeks (Alt. 2) na mid relacije Rezervacije (notranja):**
  - Pregled Mornarjev: 500 strani V/I,  $80 * 500$  n-teric.
  - Za vsako n-terico Mornarjev: 1.2 V/I za iskanje indeksne strani s podatkovnimi vpisi + cena za branje zapisov Rezervacij .
  - Predpostavljamo enakomerno porazdelitev, 2.5 rezervacij na mornarja ( $100.000 / 40.000$ ).
  - Cena branja rezervacij je 1 ali 2.5 V/I odvisno od tega ali je indeks povezan.
  - Cena brez pod.zapisov S: 500 (pregled) +  $80 * 500 * 1.2$  (pod.vpisi) = 48.500 V/I.
  - Cena povezan:  $48.500 + 40.000$  (pod.zapisi) = 88.500 V/I.
  - Cena nepovezan:  $48.500 + 100.000$  (pod.zapisi) = 148.500 V/I.

### Primeri vgnezdene zanke po vrstah

- Cena:**
  - pregled zunanje rel. + #zunanjih vrst \* pregled notranje rel.
  - #zunanjih vrst = #strani rel. / velikost zunanje vrste
- Če vzamemo Rezervacije (R) za zunanjo relacijo; SUPB ima 100 strani izravnalnika:
  - Cena pregleda R je 1000 V/I; vsega skupaj 10 zun.vrst
  - Za vsako zun.vrst R pregledamo Mornarje (S);  $10 * 500$  V/I.
  - Cena =  $1000 + 10 * 500 = 6000 V/I$
  - Če imamo prostora za samo 90 strani R, moramo pregledati S 12 X.
- Če vzamemo 100-strani veliko zun.vrst S za Mornarje kot zunanja relacija:
  - Cena pregleda S je 500 V/I; skupaj 5 zun.vrst.
  - Za vsako zun.vrst S, pregledamo Rezervacije;  $5 * 1000$  V/I.
  - Cena =  $500 + 5 * 1000 = 5500 V/I$

## 7) Predstavi stik z zlivanjem (merge sort). Str. 458 -> Pdf. 493

Osnovna ideja algoritma stika z zlivanjem je, da razvrstite oba razmerja na atributu združitve in nato poiščete kvalifikacije T E Rand s E S tako, da v bistvu združite ta dva razmerja. Korak za razvrščanje združuje vse zapise z enako vrednostjo v stolpcu za povezavo in tako olajša identifikacijo particij ali skupin tuplov z isto vrednostjo v stolpcu za pridružitev. To delitev izkoriščamo tako, da primerjamo korake R v particiji s samo S tupli v isti particiji (namesto z vsemi S torki), s čimer se izognemo oštevilčenju navzkrižnega produkta Rand S.

## Stik z zlivanjem

- Algoritem:
  - Sortiraj R in S po atributih stika.
  - Pregled (scan) sortiranih tabel z zlivanjem.
  - Izpis parov n-teric, ki se ujemajo.
- Zlivanje:
  - "R-zapis = S-zapis" pomeni, da se zapisa ujemata v atributih stika.
  - Ponavljaj.
    - Beri R dokler R-zapis > S-zapis in se ustavi ko R-zapis = S-zapis
    - Beri S dokler S-zapis > R-zapis in se ustavi ko R-zapis = S-zapis
    - Izpiši vse pare  $\langle r, s \rangle$ , ki se ujemajo v vrednostih atributov stika

## Stik z zlivanjem

- Število pregledanih n-teric.
  - Ocena:  $|R| + |S|$
- Kompleksnost stika z zlivanjem.**
  - $2^*|R|^k (1+\log_{B-1}|R|/B) + 2^*|S|^k (1+\log_{B-1}|S|/B) + |R| + |S|$
- ✓ Z uporabo 35, 100 ali 300 strani vmesnika lahko tabele Rezervacije in Mornarji sortiramo z 2 prehodi;
- ✓ Celotna cena stika:  $4*1000+4*500+1000+500=7500$  V/I.

### 8) Opiši stik z razpršilnim indeksom. Str. 463 -> Pdf. 498

Algoritem združevanja razpršitve, tako kot algoritem združevanja sort-merge, identificira particije v Rand-u S v fazi particoniranja in v naslednji fazi sondiranja primerja tuples v particiji R samo s tuples v ustrezeni particiji S za preskušanje pogojev združevanja enakosti. Za razliko od razvrščanja razvrsti-združi, heš pridružitev uporablja razprševanje za prepoznavanje particij in ne za razvrščanje. Faza particoniranja hash-pridružitve je podobna particoniranju v projekciji na osnovi hashbata in je prikazana na sliki 14.3. Faza merjenja (včasih imenovana tudi ujemanje) je prikazana na sliki 14.11.

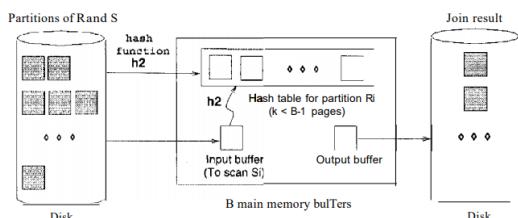


Figure 14.11 Probing Phase of Hash Join

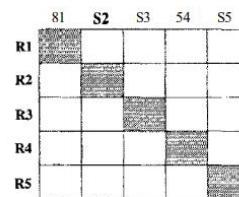
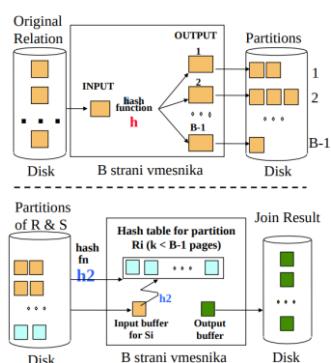


Figure 14.13 Hash Join Vs. Block Nested Loops for Large Relations

#### Stik z razpršilnim indeksom

- Porazdeli obe relaciji z razpršilno funkcijo  $h$ : n-terice particije i iz R se bodo ujeli z n-tericami particije i iz S.
- Preberi particijo R, jo porazdeli z  $h_2 (< > h_1)$ . Preglej ustrezeno particijo S in poišči n-terice, ki se ujemajo.



## Stik z razpršilnim indeksom

- B-2 > velikosti največje particije**, ki bo v spominu.
- Zgradimo razpršilno tabelo v spominu za pohitritev ujemanja n-teric (rabimo malce več spomina).
- Če razpršilna funkcija ne porazdeli n-teric enakomerno se lahko zgodi, da nekatere particije ne gredo v spomin.
  - Razprševanje lahko naredimo rekurzivno: R-particijo povežemo z ustreznim S-particijom.

## Cena stika z razpršilnim indeksom

- Faza izdelave particij:
  - branje+pisanje obeh rel. =  $2(M+N)$  V/I.
- V fazi ujemanja:
  - preberi obe rel. =  $M+N$  V/I.
- Naš primer:
  - Cena =  $3 * (1000+500) = 4500$  V/I.
- Stik z zlivanjem vs. Stik z razpršilnim indeksom:**
  - Pri minimalni količini spomina imata oba ceno  $3(M+N)$  V/I.
  - Stik z razpršilnim indeksom je superioren, če se velikosti relacij bistveno razlikujejo.
  - Stik z razpršilno funkcijo lahko zelo dobro paraleliziramo.
  - Stik z zlivanjem je manj občutljiv na "skewed" podatke; rezultat je sortiran.

## **7 OPTIMIZACIJA POIZVEDB**

### **1) Kako oceniti stroške poizvedbe? Str. 482 -> Pdf. 517**

Za vsak načrt moramo oceniti njegove stroške. Dva dela sta za ocenjevanje stroškov načrta ocenjevanja bloka poizvedbe:

1. Za vsako vozlišče v drevesu moramo oceniti stroške izvajanja ustreznih operacij. Na stroške bistveno vpliva, ali se uporablja cevovod ali če se ustvarijo začasni odnosi, s katerimi se izhod operaterja prenese na svoje matično podjetje.
2. Za vsako vozlišče v drevesu moramo oceniti velikost rezultata in razvrstiti ga. Ta rezultat je vhod za operacijo, ki ustreza nadrejenemu trenutnemu vozlišču, velikost in vrstni red sortiranja pa vplivata na oceno velikosti, stroškov in vrstnega reda za matično.

Ocene, ki jih uporablja SUPB za velikost rezultatov in stroške, so v najboljšem primeru približne dejanskim velikostim in stroškom. Nerealno je pričakovati, da bo optimizer našel najboljši načrt; bolj pomembno je, da se izognemo najhujšim načrtom in najdemo dober načrt.

### **2) Predstavite ekvivalence relacijske algebre. Str. 481 -> Pdf. 516**

#### **Prevajanje SQL v RA**

#### **Dekompozicija vprašanj v bloke**

- Dekompozicija vprašanj v bloke
- Blok vprašanja prevedemo v izraz RA
- Blok obravnavamo kot funkcijo

- Celoten SQL stavek se razdeli v bloke
- Zaključeni SQL bloki se obravnavajo kot procedure
- **Posledica:**
  - vgnezeni bloki se kličejo ob vsaki iteraciji v nadrejenem bloku

#### **Blok vprašanja prevedemo v izraz RA**

- **Vsak blok posebaj** se prevede v izraz RA
- Izraz RA se optimizira ne glede na nadrejene in podrejene bloke
- Podrejeni blok se izvaja kot **procedura**

### **3) Kako pridobiti vse enakovredne poizvedbene izraze za dano poizvedbo, izraženo v relacijski algebri? Str. 481 -> Pdf. 516**

Prvi korak pri optimizaciji bloka poizvedb je, da ga izrazimo kot izraz relacijske algebre. Za enotnost predpostavimo, da sta GROUP BY in HAVING tudi operaterji v razširjeni algebri, ki se uporablja za načrte, in da se lahko agregatne operacije pojavijo v seznamu argumentov operaterja projekcije. Pomen operaterjev bi moral biti jasen iz naše razprave o SQL. Vsak blok poizvedb SQL se lahko izrazi kot izraz razširjene algebre, ki ima to obliko. SELECT stavek ustreza operaterju projekcije, člen WHERE ustreza izbirnemu operaterju, člen FROM ustreza navzkrižnemu produktu relacij, preostale določbe pa se preslikajo na ustrezone operaterje na preprost način.

#### 4) Opišite oceno stroškov za posamezne načrte poizvedb.

### Ocena plana izvajanja

- Rezultat je vedno približek.
- Statistika shranjena v sistemskih katalogih.
- Statistika se uporabi za:
  - oceno hitrosti izvajanja plana poizvedbe, in
  - oceno velikosti vmesnih rezultatov.
- Pri oceni se uporabi cena CPU in I/O.
- V naših ocenah delamo samo s številom diskovnih blokov.

### Ocena velikosti rezultatov operacij

Poizvedba:

```
SELECT seznam-izbire
FROM seznam-relacij
WHERE term1 AND ... AND termk
```

- Maksimalno število n-teric:
  - Produkt kardinalnosti relacij v FROM stavku.
- Selektivnost izraza stavka WHERE:**
  - Selektivnost izraza je sestavljena iz selektivnosti vseh pogojev.
  - Selektivnost pogoja (SP) = delež relacije, ki je rezultat selekcije.
- Kardinalnost rezultata**
  - Max # n-teric \* produkt vseh SP.

### Ocena plana

- Za vsak plan je potrebno določiti ceno
- Potrebo je določiti ceno za vsako operacijo v drevesu
- Cena operacij je odvisna je od kardinalnosti vhodnih relacij
- Za vsak stik predpostavimo neodvisnost med predikati
- Cene posameznih operacij smo si ogledali na prejšnjem predavanju
  - Sekvenčni pregled, indeksni pregled, stik z zanko, stik z indeksom, itd.

### Ocene za prebrane bloke

- Uporaba indeksa I na primarnem ključu:
  - Visina(I)+1 za B+ drevo, cca. 1.2 za razpršilni indeks.
- Povezan indeks I, ki se ujema z enim ali več pogoji:
  - (NStrani(I)+NStrani(R)) \* produkt Si za pogoje selekcije.
- Nepovezan indeks, ki se ujema z enim ali več pogoji:
  - (NStrani(I)+NZapisov(R)) \* produkt Si za pogoje selekcije.
- Sekvenčni pregled tabele:
  - NStrani(R)

```
SELECT seznam-izbire
FROM seznam-relacij
WHERE pogoj1 AND...AND pogojk
```

### Ocena za plane nad večimi relacijami

- Blok poizvedbe:
- Maksimalno # n-teric v rezultatu je produkt kardinalnosti vseh vhodnih relacij iz FROM stavka.
- Selektivnost** povezana z vsakim **pogojem** odseva vpliv pogoja na velikost rezultata.
  - Kardinalnost rezultata = Max # n-teric \* produkt vseh S\_i.
- Multi-relacijski plani se gradijo pri stikih z dodajanjem ene same nove relacije naenkrat.
  - Cena metode stika + oceno kardinalnosti stika.

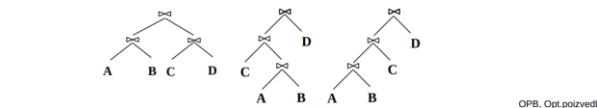
#### 5) Opišite levo usmerjene plane in ugnezdene poizvedbe? Str. 497 -> Pdf. 532

### Naštevanje levo-usmerjenih planov

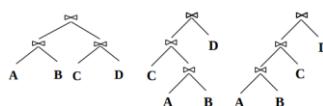
- Osnovna odločitev v System R:
  - Obračnavajo se samo levo-usmerjena drevesa.
  - Pogledali si bomo kako lahko učinkovito naštevamo levo-usmerjene plane z uporabo dinamičnega programiranja
- Levo-usmerjeni plani se razlikujejo samo v:
  - Vrstnem redu relacij
  - V metodah za dostop do relacij
  - Algoritmih za izvajanje stika

### Naštevanje levo-usmerjenih planov

- Poglejmo si bloka vprašanja:
  - Optimizator sistema R pregleda vse možne stike, kjer se  $\delta$  in  $\Pi$  spusti k relacijam
  - SELECT attribute list  
FROM relation list  
WHERE term<sub>1</sub>  $\wedge$  term<sub>2</sub>  $\wedge$  ...  $\wedge$  term<sub>n</sub>
- Lastnosti algoritma:
  - Navkljub omejitvi št. planov raste eksponentno s št. relacij.
  - Mogoče je generirati plane, ki realizirajo "cevovode".
  - Vmesne rezultate ni potrebno zapisati v začasne datoteke.



OPB. Opt.poizvedb



## Naštevanje levo-usmerjenih planov

- Naštevanje z N prehodi za N relacij:
  - **Prehod 1:** Poišči vse najboljše plane z eno samo relacijo.
  - **Prehod 2:** Poišči najboljši plan vseh stikov najboljših planov nad eno relacijo t.j. plan nad dvemi relacijami.
  - **Prehod N:** Poišči najboljši plan, ki združi najboljše plane nad N-1 relacijami z najboljšimi plani nad preostalo eno relacijo t.j. plan nad vsemi N relacijami.
- **Dinamično programiranje:**
  - Za vsako podmnožico sestavljeno iz K relacij je potrebno ohraniti samo **najcenejši plan!**
  - Plane za N relacij sestavljamo iz planov za N-1 relacij tako da dodajamo še en stik.

## Vgnezdene poizvedbe

- Vgnezden blok se optimizira neodvisno in zunanj n-terica se nad tem blokom preveri kot nad pogojem selekcije.
- Zunanji blok je optimiziran s ceno "klicanja" vgnezdenega bloka kot funkcijo.
- Implicitna urejenost vgnezdenih blokov omejuje pregled nekaterih dobrih strategij.
- *Poizvedbe brez gnezdenja se tipično bolje optimizirajo.*

```
SELECT M.mime
FROM Mornarji M
WHERE EXISTS
(SELECT *
FROM Rezervacije R
WHERE R.id=103
AND R.mid=M.mid)
```

Vgnezden blok za optim.:  
SELECT \*
FROM Rezervacije R
WHERE R.id=103
AND M.mid= zun.vred.

Ekvivalentna p. brez vgnezd.p.:  
SELECT M.Mime
FROM Mornarji M, Rez R
WHERE M.mid=R.mid
AND R.id=103

## 8 KONTROLA SOČASNOSTI

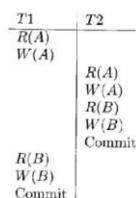
### 1) Kaj je transakcija? Str. 550 -> Pdf. 585

Transakcija je abstrakten pogled SUPB na uporabniški program. Je sekvenca ukazov za branje in pisanje. Uporabniki si pri delu s transakcijami lahko predstavljajo, da se vsaka transakcija izvaja posebej. Sočasnost doseže SUPB z izmenjavanjem akcij (branje/pisanje) večjih transakcij. Vsaka transakcija pusti podatkovno bazo v konsistentnem stanju, če je bila baza konsistentna ob začetku transakcije. SUPB zagotovi integritetne omejitve, ki so bile definirane v okviru tabel podatkovne baze. Razen integritetnih omejitev SUPB dejansko ne razume pomena podatkov.

### 2) Pojasnite morebitne nepravilnosti pri izvedbi transakcij.

#### Branje nepotrjenih podatkov

- Imamo dve transakciji
  - T1: iz računa A prenesemo 100eur na račun B
  - T2: dodamo 6% obresti računu A in B
- Recimo, da imamo sledečo ureditev akcij
  - 1. Odštejemo 100eur iz A in zapišemo A
  - 2. Prištejemo 6% obresti A in B
  - 3. Prištejemo 100eur B in zapišemo B
- **Branje nepotrjenih podatkov**
  - Stanje baze je lahko tudi nekonsistentno



#### Neponovljivo branje

- Drug problem, ki se lahko pojavi:
  - T2 spremeni vrednost A medtem, ko se T1 izvaja
  - Če T1 spet prebere A dobi drugo vrednost
  - To situacijo imenujemo **neponovljivo branje**
- Primer
  - A=1 predstavlja število knjig
  - T1 prebere A;
  - T2 prebere A in zmanjša za 1;
  - T1 poskuša zmanjšati A za 1 vendar je A že 0

#### Prepis nepotrjenih podatkov

- Transakcija T2 prepiše vrednost A med izvajanjem T1, po tem, ko transakcija T1 zapiše A
  - Tudi če T2 ne prebere vrednosti A (zapis T1) lahko pride do problemov
- Primer
  - Tone in Miha imata enako plačo
  - T1 popravi plačo obema na 2000 eur, T2 pa na 1000 eur.
  - Po izvajaju T1 in T2 je lahko plača 1000 ali 2000 eur
  - Če se akcije med sabo prepletajo so lahko plače različne

#### Anomalije pri prekrivajočih transakcijah

- Branje nepotrjenih podatkov (WR konflikt, "dirty reads"):

T1:	R(A), W(A),	R(B), W(B), Abort
T2:	R(A), W(A), C	

- Neponovljivo branje (RW konflikt):

T1:	R(A),	R(A), W(A), C
T2:	R(A), W(A), C	

#### Anomalije

- Prepis nepotrjenih podatkov (WW konflikt):

T1:	W(A),	W(B), C
T2:	W(A), W(B), C	

### 3) Kaj je konfliktni zaporedni prenos transakcije? Str. 525 -> Pdf. 560

Konfliktni zaporedni razpored nad nizom S zvezanih transakcij je urnik, katerega učinek na vsak skladen primerek baze podatkov je zagotovljen tako, da je enak kot pri celotnem serijskem načrtu nad S. To pomeni, da je primerek baze podatkov, ki izhaja iz izvajanja danega razporeda, enak kot primerek baze podatkov, ki se izvede z izvajanjem transakcij v nekaj zaporedjih.

Na primer, razpored prikazan na sliki 16.2 je serijski. Čeprav so dejanja T1 in T2 medsebojno prepletena, je rezultat tega razporeda enakovreden delovanju T1 (v celoti) in nato zagonu T2. Intuitivno T1-novo branje in zapisovanje B ni pod vplivom dejanj T2 na A, neto učinek pa je enak, če se ti "zamenjajo", da dobimo serijski razpored T1, T2.

$T1$	$T2$
$R(A)$	
$W(A)$	
	$R(A)$
	$W(A)$
$R(B)$	
$W(B)$	
	$R(B)$
	$W(B)$
	Commit
Commit	

Figure 16.2 A Serializable Schedule

### 4) Opišite striktno in ne striktno dve fazno zaklepanje. Str. 531 -> Pdf. 566

Striktno dve fazno zaklepanje ima dva pravila:

1. Če transakcija T želi prebrati (oziora spremeniti) objekt, najprej zahteva skupno (oziora izključno) zaklepanje objekta.

Seveda lahko transakcija, ki ima izključno zaklepanje, prebere tudi predmet; dodatna zaklepanje v skupni rabi ni potrebno. Transakcija, ki zahteva zaklepanje, je prekinjena, dokler ji SUPB ne dovoli zahtevane ključavnice. SUPB sledi ključavnicam, ki jih je podelil, in zagotavlja, da če transakcija vsebuje izključno zaklepanje za objekt, nobena druga transakcija ne vsebuje skupne ali izključne zaklepanja na istem predmetu.

2. Vse blokade, ki jih ima transakcija, se sprosti, ko je transakcija zaključena.

Zahteve za pridobitev in sprostitev ključavnic se lahko samodejno vnesejo v transakcije s strani SUPB; uporabnikom ni treba skrbeti za te podrobnosti.

Protokol zaklepanja dejansko dovoljuje le „varno“ prepletanje transakcij. Če dve transakciji dostopata do popolnoma neodvisnih delov baze podatkov, istočasno dobita ključavnice, ki jih potrebujejo, in se veselo nadaljujeta. Na drugem pasu, če dve transakciji dostopata do istega objekta in ga želita spremeniti, se njihova dejanja učinkovito zaporedoma serijsko ujemajo vsa dejanja ene od teh transakcij (tista, ki najprej zaklene skupni predmet) (ta zaklep se sprosti in) se lahko izvede druga transakcija.

Dvo-fazno zaklepanje sprosti zaklenjene objekte kadarkoli, vendar po sprostitvi istih objektov ni moč ponovno zakleniti. Ta protokol bi dopustil ureditev akcij.

### 5) Kaj se zgodi, ko je transakcija prekinjena? Str. 535 -> Pdf. 570

## Prekinitve transakcije

- Če je transakcija *Ti* **prekinjena** potem morajo biti vse njene akcije izničene.
  - Ne samo to; če *Tj* bere objekte, ki jih je napisala *Ti*, potem mora biti prekinjena tudi *Tj*!
- Večina sistemov se izogne **kaskadnim prekinitvam** s sprostitvijo zaklenjenih objektov tik pred potrditvijo.
  - *Ti* popravi objekt; *Tj* ga lahko prebere šele po potrditvi *Ti*.
- **SUPB zapisuje kompletен dnevnik vseh popravkov.**
  - Prekinjene transakcije je mogoče izničiti.
  - Isti mehanizem se uporablja za vzpostavitev konsistentnega stanja po sistemski napaki.
  - Faza širjenja in krčenja

### 6) Opišite metode za preprečevanje smrtnega objema. Str. 558 -> Pdf. 593

## Preprečevanje smrtnega objema

- Določi prioritete glede na časovne zaznamke. Ti želi zaklep, ki ga drži *Tj*. Dve možni politiki:
  - **Počakaj-Umri:** Če ima *Ti* višjo prioriteto potem *Ti* počaka na *Tj*; sicer *Ti* prekine izvajanje.
  - **Rani-Čakaj:** Če ima *Ti* višjo prioriteto potem *Tj* prekine izvajanje; sicer *Ti* počaka.
- Če se transakcija ponovno starta je potrebno zagotoviti, da ima originalno časovno oznako.

V shemi počakaj-umri transakcije z nižjo prednostjo ne morejo nikoli čakati na večje prednostne transakcije. V shemi rani-čakanj transakcije z višjo prioriteto nikoli ne čakajo na transakcije z nižjo prednostjo. V obeh primerih ni smrtnega objema.

Subtilna točka je, da moramo zagotoviti tudi, da nobena transakcija ni več prekinjena, ker nikoli nima dovolj visoke prioritete. (Upoštevajte, da v obeh shemah transakcija z višjo prioriteto ni nikoli prekinjena.) Ko je transakcija zavrnjena in ponovno zagnana, mora imeti isti časovni žig, kot je bil prvotno. Ponovna izdaja časovnih zapisov na ta način zagotavlja, da bo vsaka transakcija sčasoma postala najstarejša transakcija, torej tista, ki ima najvišjo prednost, in bo dobila vse ključavnice, ki jih zahteva.

### 7) Kaj je zaklepanje indeksa? Kako se izvaja?

Če obstaja indeks na ocenjevalnem polju, lahko *T1* ponovno dobi ključavnico na indeksni strani, pri čemer ugotovi, da je fizično zaklepanje izvedeno na ravni strani, ki vsebuje vnos podatkov z oceno = 1. Če takšnih vnosov ni, to pomeni, da ni nobene zapise s to bonitetno vrednostjo, stran, ki bi vsebovala vnos podatkov za rating = 1, je zaklenjena, da se prepreči vnos takega zapisa. Vsaka transakcija, ki poskuša vstaviti zapis z rating = -

1 v razmerje Sailors "11|USt", vstavi podatkovni vnos, ki kaže na nov zapis v to indeksno stran in je blokiran, dokler T1 ne izda svojih ključavnic. Ta tehnika se imenuje zaklepanje indeksa.



- Pri **sekvenčnem skenirjanju** je potrebno zagotoviti, da se ne sme dodajati zapisov tabeli
  - T1 mora zakleniti vse strani kot tudi datoteko/tabelo, da bi onemogočili dodajanje zapisov z *rating* = 1.
- Če imamo **indeks za atribut rating** z alternativo (2), potem bi T1 morala zakleniti indeksne strani z indeksnimi vpisi z *rating* = 1.
  - Če ni zapisov z *rating* = 1, potem mora T1 zakleniti indeksne strani kjer bi bili takšni podatki, če bi obstajali !

#### 8) Opišite optimistični nadzor sočasnosti. Str. 566 -> Pdf. 601

Protokoli za zaklepanje uporabljajo pesimistični pristop k konfliktom med transakcijami in za odpravljanje sporov uporabljajo bodisi prekinitve transakcije ali blokiranje. V sistemu s sorazmerno lahkim argumentom za podatkovne objekte je treba kljub temu plačati dodatne stroške za pridobivanje ključavnic in protokol zaklepanja.

Pri optimističnem nadzoru vzporednosti je osnovna predpostavka, da večina transakcij ni v nasprotju z drugimi transakcijami, ideja pa je, da je čim bolj dopustna pri omogočanju izvrševanja transakcij. Transakcije potekajo v treh fazah:

1. Branje: Transakcija se izvaja, branje vrednosti iz baze podatkov in pisanje v zasebni delovni prostor.
2. Validacija: Če se transakcija odloči, da želi komunicirati, SUPB preveri, ali bi transakcija lahko bila v nasprotju s katero koli drugo sočasno izvedeno transakcijo. Če pride do konflikta, se transakcija prekine; njegov zasebni delovni prostor se izbriše in se znova zažene.
3. Pisanje: Če preverjanje potrjuje, da ni možnih konfliktov, se spremembe podatkovnih objektov, ki jih je naredila transakcija v zasebnem delovnem prostoru, kopirajo v bazo podatkov.

Če je resnično malo konfliktov in če je vrednotenje mogoče učinkovito, bi moral ta pristop voditi k boljšemu delovanju kot zaklepanje. Če je veliko konfliktov, stroški ponavljajočih se transakcij (s tem zapravljanje, ki so jih naredili) močno prizadenejo izvedbo.

## **9 OBNOVITEV PO ZRUŠITVI**

### **1) Razložite lastnosti ACID relacij v SUPB. Str. 520 -> Pdf. 555**

SUPB mora zagotoviti štiri pomembne lastnosti transakcij za vzdrževanje podatkov ob sočasnem dostopu in sistemskih napakah:

1. Uporabniki bi morali imeti možnost, da izvajanje vsake transakcije obravnavajo kot atomsko: ali se vsa dejanja izvedejo ali pa jih ni. Uporabniki ne bi smeli skrbeti za učinek nepopolnih transakcij (recimo, ko pride do zrušitve sistema).
2. Vsaka transakcija, ki jo izvede sama po sebi brez sočasnega izvajanja drugih transakcij, mora ohraniti doslednost podatkovne zbirke. SUPB predpostavlja, da za vsako transakcijo velja skladnost. Zagotavljanje te lastnosti transakcije je odgovornost uporabnika.
3. Uporabniki naj bodo sposobni razumeti transakcijo, ne da bi upoštevali učinek drugih sočasno izvedenih transakcij, tudi če SUPB opravičuje delovanje več transakcij. Ta lastnost se včasih imenuje izolacija: Transakcije so izolirane ali zaščitene od učinkov sočasnega razporejanja drugih transakcij.
4. Ko SUPB obvesti uporabnika, da je bila transakcija uspešno zaključena, bi morali njeni učinki obstajati tudi, če se sistem zruši, preden se vse spremembe prikažejo na disku. Ta lastnost se imenuje trajnost.

Kratica ACID se včasih uporablja za sklicevanje na te štiri lastnosti transakcij: atomarnost, konsistentnost, izoliranost in ohranjanje.

### **Pregled: ACID lastnosti**

- ❖ **A tomicity:** (atomarnost)  
Izvršijo se vse akcije Xact ali nobena.
- ❖ **C onsistency:** (konsistentnost)  
Če sta Xact in celotna PB konsistentna, potem je po transakciji PB konsistentna.
- ❖ **I solation:** (izolacija)  
Izvajanje ene Xact je izolirano od izvajanja drugih transakcij.
- ❖ **D urability:** (ohranjanje)  
Če Xact potrdi, so vse spremembe stalne.
- **Upravljalnik obnavljanja** garantira atomarnost & ohranjanje.

## 2) Pojasnite kompromise "kraje" strani iz področja medpomnilnika in "prisilite" strani, da se shranijo na disk.

V zvezi s pisanjem predmetov se pojavita dve dodatni vprašanji:

1. Ali lahko spremembe v objektu O v področju medpomnilnika s transakcijo T zapišemo na disk, preden se T zažene? Takšni zapiski se izvedejo, ko druga transakcija želi vnesti stran in upravljalnik medpomnilnika se odloči zamenjati okvir, ki vsebuje O; Seveda pa mora biti ta stran odpeta s T. Če so taki napis dovoljeni, rečemo, da se uporablja pristop k ukradi. (Neformalno, druga transakcija "ukrade" okvir iz T).
2. Ko se zgodi transakcija, moramo zagotoviti, da so vse spremembe, ki jih je naredil za objekte v področju medpomnilnika, takoj prisiljeni na disk? Če je tako, rečemo, da se uporablja pristop sile.

Z vidika izvajanja upravljalnika obnovitve je najpreprosteje uporabiti upravitelja medpomnilnika s pristopom brez ukradene sile. Če uporabimo pristop brez ukrade, ni treba razveljaviti sprememb prekinjene transakcije (ker te spremembe niso bile zapisane na disk) in če je uporabljen pristop sile, ni treba ponavljati sprememb. prevzeta transakcija, če pride do poznejšega sesutja (ker so vse te spremembe zapisane na disk ob času izvršitve).

## Več o kraji in sili

- **KRAJA** (zakaj je doseganje atomarnosti težko)
  - *Ukrasti okvir F*: Trenutna stran v F (naj bo P) je zapisana na disk; neka Xact ima zaklep na P.
    - Kaj če Xact z zaklepom na P prekine izvajanje?
    - Moramo si zapomniti staro vrednost P ob kraji (za podporo UNDO pisanja na stran P).
- **BREZ SILE** (zakaj je doseganje ohranjanja težko)
  - Kaj če sistem pada preden se spremenjena stran zapiše na disk?
  - Zapiši čim manj je možno, na prizerno mesto, ob potrjevanju, da bi omogočili REDO sprememb.

## 3) Opišite protokol zapisovanja pred pisanjem (WAL). Str. 581 -> Pdf. 616

Zapisovanje naprej: Vsaka sprememba objekta baze podatkov se najprej zabeleži v dnevniku; zapis v dnevnik mora biti zapisan v stabilno shrambo, preden se sprememba objekta baze podatkov zapiše na disk.

### Dnevniško pisanje-vnaprej (WAL)

- WAL protokol
- Kontrolne točke
- Obnovitev po sistemski zrušitvi
  - Faza analize
  - REDO faza
  - UNDO faza
- Primer obnovitve
- Problemi pri obnovitvi

### Normalno izvajanje Xact

- Vrsta branj & pisanj, ki jim sledi potrditev ali prekinitev.
  - Predpostavili bomo, da je pisanje na disk atomično.
    - V praksi imamo več podrobnosti okoli atomičnosti pisanja.
- Striktno 2FZ.
- Delo izravnalnika:
  - STEAL, NO-FORCE + WAL dnevnik.

## WAL

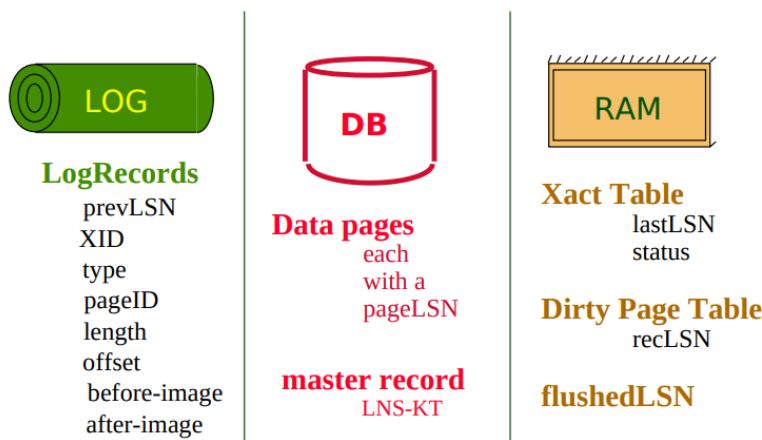
- Pred zapisovanjem strani na disk morajo biti vsi log zapisi, ki govorijo o spremembah na strani zapisani (in odplaknjeni) na disk
- WAL protokol zagotavlja, da se vsaka sprememba da obnoviti**
- Transakcija naredi spremembo
  - Zaradi NO-FORCE protokola se lahko zgodi, da spremembe po COMMIT niso zapisane na disk
  - Brez dnevnika nebi bilo mogočo obnoviti sprememb
- Potrjena (Committed) transakcija** = vsi dnevniški zapisi transakcije so bili zapisani na disk

## Kontrolna točka

- Periodično SUPB kreira **kontrolno točko** za minimizacijo časa porabljenega za obnovitev v primeru sistemskih zrušitev.
- Zapiši v dnevnik:**
  - `begin_checkpoint` zapis: Določa začetek KT.
  - `end_checkpoint` zapis: Vsebuje trenutno tabelo Xact in tabelo umazanih strani. To je **mehka kontrolna točka**:
    - Ostale Xacts nadaljujejo z delom; tabele so torej zanesljive samo v času `begin_checkpoint` zapisa.
  - Shrani LSN zapisa kontrolne točke na varno mesto (**master** zapis).

4) Opišite, kateri podatki so shranjeni in kje so shranjeni za izvajanje protokola WAL.

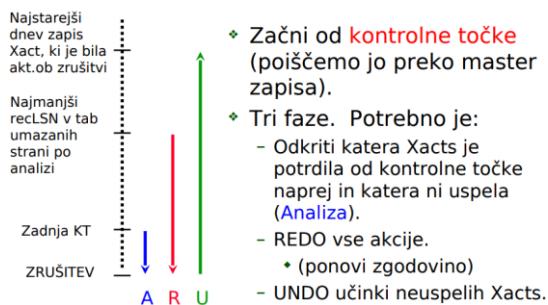
### Ptičja perspektiva: Kaj je shranjeno kje



5) Predstavite faze analize, ponovitve in razveljavitve obnove po zrušitvi.

### Obnovitev: faza analize

#### Obnovitev po zrušivi: celotna slika



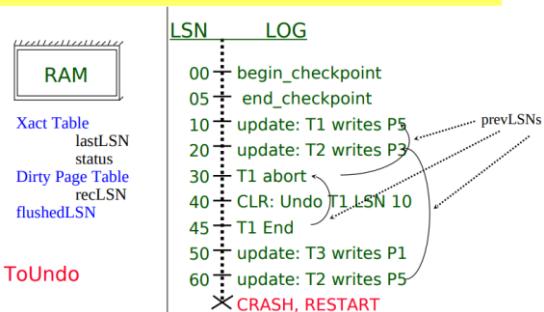
#### Naloge analize

- Določi točko kjer se začne REDO
- Identificira umazane strani v bazenu strani
- Identificiraj transakcije, ki so bile aktivne ob zrušitvi (jih bo potrebeno izničiti)
- Rekonstruiraj stanje ob kontrolni točki.
  - Uporabi `end_checkpoint` zapis.
- Preglej dnevnik naprej od kontrolne točke:
  - End** zapis: Odstrani Xact iz Xact tabele.
  - Ostali zapisi:** Dodaj Xact k Xact tabeli, postavi `lastLSN=LSN`, spremeni Xact status na **potrdi**.
  - Update** zapis: Če P ni v Tabeli umazanih strani,
    - Dodaj P K T.U.S., postavi `recLSN=LSN`.

## Obnovitev: Faza REDO

- Ponovimo zgodovino zato, da rekonstruiramo stanje ob zrušitvi:
  - Ponovno ovrednoti vse popravke (tudi za prekinjene Xact!), ponovno ovrednoti CLR.
- Pregled naprej od dnev zapisa, ki vsebuje najmanši **recLSN** v TUS.
  - Spremenjena stran ni v TUS ali
  - Spremenjena stran je v TUS, vendar **recLSN > LSN** ali **pageLSN** (v DB)  $\geq$  **LSN**.
- **REDO** akcije:
  - Ponovi akcijo v dnevniku.
  - Postavi **pageLSN** na **LSN**. Ni dodatnega zapisa v dnev!

## Primer obnovitve



## Obnovitev: Faza UNDO

**ToUndo** = { / | / -- lastLSN prekinjenih Xact}

### Repeat:

- Izberi največji LSN med ToUndo.
- Če je LSN **CLR** in velja **undonextLSN==NULL**
  - Zapiši **End** zapis za to Xact.
- Če je LSN **CLR**, in **undonextLSN != NULL**
  - Dodaj **undonextLSN** k **ToUndo**
- Sicer je LSN **popravek**. Izniči popravek, zapiši **CLR** in dodaj **prevLSN** k **ToUndo**.

Until **ToUndo** je prazen.

## Drugi problemi pri zrušitvi

- Kaj se zgodi, če se sistem zruši med analizo pri obnovi? Med fazo REDO?
- Kako omejiti količino dela med REDO?
  - Splakuj asinhrono v ozadju.
  - Pazi na "špice"!
- Kako omejiti količino dela pri UNDO?
  - Izogibaj se dolgo-trajajočim Xacts.

## 10 LOGICAL DESIGN

### 1) Zakaj se lahko v relacijskih podatkovnih bazah pojavi redundantnost?

#### Problem redundancy

- **Redundantnost** je vzrok za vrsto problemov relacijskih podatkovnih baz:
  - redundantni podatki, anomalije pri insert/delete/update.
- Integritetne omejitve -- funkcijeske **odvisnosti** -- uporabimo za ident. anomalij ter popravke shem.
- Osnovna tehnika: **dekompozicija**
  - Npr. zamenjava ABCD z AB in BCD, ali ACD in ABD.
- Dekompozicijo je potrebno uporabljati previdno:
  - Obstaja razlog za dekompozicijo relacije?
  - Obstaja kakšen problem, ki ga lahko povzroči dekompozicija?

### 2) Kaj je funkcijeska odvisnost? [VIR](#)

V teoriji relacijskih baz podatkov je **funkcionalna odvisnost omejitev med dvema nizoma atributov v relaciji** iz baze podatkov. Z drugimi besedami, funkcionalna odvisnost je med atributi v relaciji.

#### Funkcijeske odvisnosti

- Funkcijeska odvisnost  $X \rightarrow Y$  velja na relaciji R če, za vsako veljavno instanco r sheme R:
  - $t1 \in r, t2 \in r, \Pi_X(t1) = \Pi_X(t2) \rightarrow \Pi_Y(t1) = \Pi_Y(t2)$
  - Za dani n-terici relacije r velja: če se ujemata v vrednosti X, potem se morata ujemati tudi v vrednosti Y.
  - Morajo izvirati iz pomena aplikacije.
  - Za dano relacijo r1 nad shemo R, lahko preverimo ali velja  $FO f$ , ne moremo pa reči da f velja nad R!
- K je kandidatni ključ R; pomeni, da  $K \rightarrow R$ 
  - Seveda, za K ni potrebno, da je *minimalna množica*.

#### Zgled

- Relacija za zaposlene plačane na uro Zap\_Ura:
  - Zap\_Ura (dst, ime, delež, ocena, placilo, ure)
- **Notacija:** shemo relacije bomo označevali kar s seznamom atributov: **DIZOPU**
  - To je *množica* atributov {D,I,Z,O,P,U}.
  - Včasih se bomo sklicevali na atribute relacije kar z imenom relacije npr. Zap\_Ure za DIZOPU
- Nekatere f.odvisnosti v relaciji Zap\_Ura:
  - *dst je ključ:*  $D \rightarrow \text{DIZOPU}$
  - *ocena določa placilo:*  $O \rightarrow P$

#### Zgled

- Problemi zaradi O  $\rightarrow$  P:
  - **Anomalija pri popravljanju:** Lahko popravimo P v eni sami n-terici DIZOPU ?
  - **Anomalija pri vstavljanju:** Kaj v primeru, da vstavimo novo n-terico in ne poznamo urne postavka za dano oceno ?
  - **Anomalija pri brisanju:** Če zbrisemo vse zapise ocene 5 potem izgubimo tudi podatek o urni postavki za to oceno !

Postavke	O	P
8	10	
5	7	

Zap\_ura2

D	I	Z	O	P	U
123-22-3666	Ivan	48	8	10	40
231-31-5368	Tone	22	8	10	30
131-24-3650	Miha	35	5	7	30
434-26-3751	Janez	35	5	7	32
612-67-4134	Marjan	35	8	10	40

Bodo 2 manjši tabeli boljši ?

OPOB, Logično načrtovanje

### 3) Kako lahko razumemo funkcionalne odvisnosti? [VIR](#)

Glede na relacijo R naj bi niz atributov X v R funkcionalno določil drug niz atributov Y, tudi v R, (napisano  $X \rightarrow Y$ ), če in samo če je vsaka vrednost X v R povezana s točno enim Y vrednost v R; R naj bi torej zadovoljil funkcionalno odvisnost  $X \rightarrow Y$ . Enakovredno je projekcija  $\Pi_{\{X, Y\}} R$  funkcija, tj. funkcija X. S preprostimi besedami, če so vrednosti za attribute X znane (pravijo, da so x), se lahko vrednosti za attribute Y, ki ustrezajo x, določijo tako, da jih pogledamo v poljubni n-oki R, ki vsebuje x. Običajno se X imenuje niz determinant in Y odvisen niz. Funkcionalna odvisnost FD:  $X \rightarrow Y$  se imenuje trivialna, če je Y podmnožica X.

Z drugimi besedami, odvisnost FD:  $X \rightarrow Y$  pomeni, da se vrednosti Y določajo z vrednostmi X. Dve enoti, ki si delita iste vrednosti X, imata nujno enake vrednosti Y.

### 4) Kakšen je namen normalizacije odnosa? Kaj je normalna oblika? Str. 615 -> Pdf. 650 [WIKI](#)

Normalne oblike (Normal forms) so ravni normalizacije baze podatkov, ki določajo "dobroto (goodness)" tabele. Na splošno velja, da je tretja normalna oblika "dober" standard za relacijsko bazo podatkov.

Namen normalizacije je osvoboditi bazo podatkov iz anomalij posodobitev, vstavljanja in brisanja. Zagotavlja tudi, da ima nova vrednost v razmerju minimalni učinek na bazo podatkov in s tem minimalen učinek na aplikacije, ki uporabljajo bazo podatkov.

### 5) Predstavite normalno obliko razmerja Boyce-Codd. Str. 615 -> Pdf. 650 [WIKI](#)

Normalna oblika Boyce-Codd (ali BCNF ali 3.5NF) je normalna oblika, ki se uporablja pri normalizaciji baze podatkov. Je nekoliko močnejša različica tretje normalne oblike (3NF). BCNF so leta 1975 razvili Raymond F. Boyce in Edgar F. Codd za obravnavo določenih vrst anomalij, ki jih 3NF ni obravnaval, kot je bilo prvotno opredeljeno.

Če je relacijska shema v BCNF, je bila odstranjena vsa redundanca, ki temelji na funkcionalni odvisnosti, čeprav še vedno obstajajo druge vrste redundance. Relacijska shema R je v normalni obliki Boyce-Codd, če in samo če za vsako od njenih odvisnosti

$X \rightarrow Y$  velja vsaj eden od naslednjih pogojev:

1.  $X \rightarrow Y$  je trivialna funkcionalna odvisnost ( $Y \subseteq X$ )
2. X je superkey za shemo R

### 6) Predstavite 3. normalno obliko (3NF) relacije. Str. 617 -> Pdf. 652 [WIKI](#)

Tretja normalna oblika (3NF) je normalna oblika, ki se uporablja pri normalizaciji baze podatkov. 3NF je prvotno opredelil E.F. Codd leta 1971. Coddova definicija navaja, da je tabela v 3NF, če in samo če sta izpolnjena oba naslednja pogoja:

1. Razmerje R (tabela) je v drugi normalni oblik (2NF)
2. Vsak ne-glavni atribut R je neprehodno odvisen od vsakega ključa R.

Ne-glavni atribut R je atribut, ki ne pripada nobenemu kandidatnemu ključu R. Prehodna odvisnost je funkcionalna odvisnost, pri kateri  $X \rightarrow Z$  ( $X$  določa  $Z$ ) posredno, na podlagi  $X \rightarrow Y$  in  $Y \rightarrow Z$  (kjer ni tako, da je  $Y \rightarrow X$ ).

Opredelitev 3NF, ki je enakovredna Coddovim, vendar drugače izražena, je Carlo Zaniolo dal leta 1982. Ta definicija navaja, da je tabela v 3NF, če in samo če je za vsako od njegovih funkcionalnih odvisnosti  $X \rightarrow A$  vsaj ena od izpolnjuje naslednje pogoje:

1. X vsebuje A (to je, A je podmnožica X, kar pomeni, da je  $X \rightarrow A$  trivialna funkcionalna odvisnost)
2. X je super ključ
3. Vsak element A - X, nastavljena razlika med A in X, je glavni atribut (tj. Vsak atribut v A-X je vsebovan v nekem ključu kandidatu).

Zaniolova definicija daje jasen občutek za razliko med 3NF in strožjo normalno obliko Boyce – Codd (BCNF). BCNF preprosto odpravi tretjo alternativo ("Vsak element A-X, nastavljena razlika med A in X, je glavni atribut").

## 7) Kaj je dekompozicija razmerja brez izgube (lossless-join decomposition)? Str. 619 -> Pdf. 654 [WIKI](#)

### Dekompozicija relacijske sheme

- Dana je relacija R ( $A_1 \dots A_n$ ).
- **Dekompozicija** R pomeni zamenjavo R z dvemi ali več relacijami za katere velja:
  - Shema vsake relacije vsebuje podmnožico atributov R ter ni nobenih dodatnih atributov, in
  - Vsak atribut R se pojavi vsaj enkrat v neki novi relaciji.
- Intuitivno dekompozicija R pomeni shranjevanje instanc relacijskih shem, ki so bile izbrane med dekompozicijo namesto z instanco R.
- Primer: shemo **DIZOPU** razcepimo v **DIZOU** in **OP**.

### Dekompozicija z brezizgubnim stikom

- Dekompozicija R v X in Y je brezizgubna za množico FO F, če velja za vsako instanco r, ki zadošča F:
  - $\Pi_X(r) \bowtie \Pi_Y(r) = r$
- Vedno je res, da  $r \subseteq \Pi_X(r) \bowtie \Pi_Y(r)$ 
  - V splošnem obratno ne velja. Če velja potem je dekompozicija brezizgubna.
- Definicijo lahko razširimo na 3 ali več relacij.
- **Pomembno je, da so dekompozicije, ki se uporabljajo za odpravo redundancije, brezizgubne!**

### Problemi z dekompozicijo

- ❖ Trije potencialni problemi:
  - Nekatera vprašanja postanejo bolj zahtevna.
    - Vprašanja, ki potrebujejo stike.
    - Včasih iz instanc shem po dekompoziciji ni mogoče restavrirati originalno relacijo.
      - Informacija se izgubi pri dekompoziciji.
    - Preverjanje nekaterih odvisnosti zahteva stik večih instanc nastalih pri dekompoziciji.
  - **Tradeoff:** Problemi vs. Redundantnost.

### Dekompozicija z brezizgubnim stikom

S	P	D	S	P	P	D	S	P	D
s1	p1	d1	s1	p1	p1	d1	s1	p1	d1
s2	p2	d2	s2	p2	p2	d2	s2	p2	d2
s3	p1	d3	s3	p1	p1	d3	s3	p1	d3

Instance r       $\pi_{SP}(r)$        $\pi_{PD}(r)$        $\pi_{SP}(r) \bowtie \pi_{PD}(r)$

### Brezizgubni stik

- Dekompozicija R v X in Y je brezizgubna za F če in samo če zaprtje F vsebuje:
  - $X \cap Y \rightarrow X$ , ali
  - $X \cap Y \rightarrow Y$
- Dekompozicija R v UV in R-V je brezizgubna, če velja odvisnost  $U \rightarrow V$ .

OPB, Logično načrtovanje

A	B	C
1	2	3
4	5	6
7	2	8

S	P
s1	p1
s2	p2
s3	p1

P	D
p1	d1
p2	d2
p1	d3

A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

## 8) Kaj je odvisnost, ki ohranja dekompozicijo relacije? Str. 621 -> Pdf. 656 [VIR](#) -> preberi

### Ohranjanje f. odvisnosti

- Pogodba(*pid,bid,jid,oid,did,kol,vred*)
  - P je ključ:  $P \rightarrow PBJOKV$
  - $JD \rightarrow P$ ,  $P \rightarrow PBJOKV$  implicira  $JD \rightarrow PBJOKV$
  - $BO \rightarrow D$  implicira  $BJO \rightarrow JD$ 
    - $BO$  ni ključ =>  $BO \rightarrow D$  krši BCNO
  - $BJO \rightarrow JD$ ,  $JD \rightarrow PBJOKV$  implicira
  - $BJO \rightarrow PBJOKV$
- Dekompozicija:  $BOD + PBJOKV$ 
  - Zdaj sta tabeli v BCNO, dekompozicija je brezizgubna
  - Problem: v  $PBJOKV$  lahko enostavno zagotovimo  $JD \rightarrow P$
  - Zdaj je potrebno narediti stik za preverjanje  $JD \rightarrow P$
  - Dekompozicija ne ohranja odvisnosti

### Ohranjanje f. odvisnosti

- Ohranjanje f. odvisnosti pri dekompoziciji omogoča enostavno zagotavljanje odvisnosti
  - Preverjanje ob vstavljanju in spremnjanju
- Definicija dekompozicije z ohranjanjem odvisnosti
  - Na osnovi projekcije množice odvisnosti na podrelacije
  - Unija projekcij mora pokrivati vse odvisnosti

## 9) Predstavite algoritem za dekompozicijo relacije v BCNF. Str. 622 -> Pdf. 657

1. Recimo, da R ni v BCNF. Naj bo X c R, A, en atribut v R in  $X \rightarrow A$  FD, ki povzroči kršitev BCNF. Razstavimo R v  $R - A$  in  $XA$ .
2. Če  $R - A$  ali  $XA$  ni v BCNF, jih razčlenite še z rekurzivno uporabo tega algoritma.

$R - A$  označuje množico atributov, ki niso A v R, in  $XA$  označuje zvezo atributov v A in A. Ker  $X \rightarrow A$  krši BCNF, to ni trivialna odvisnost; nadalje, A je en atribut. Zato A ni v X; to pomeni, da je X n A prazen. Zato je vsaka razgradnja, izvedena v 1. koraku, združena brez izgube.

Niz odvisnosti, povezanih z  $R - A$  in  $XA$ , je projekcija F na njihove attribute. Če eden od novih odnosov ni v BCNF, ga razčlenimo še v koraku 2. Ker razgradnja privede do razmerij s strogo manj atributov, se ta proces zaključi, tako da imamo zbirko relacijskih shem, ki so vse v BCNF. Nadalje, povezovanje primerov (dveh ali več) relacij, dobljenih s tem algoritmom, daje natančno ustrezni primer prvotne relacije (t.j. dekompozicija v zbirko relacij, od katerih je vsaka v BCNF razgradnja združevanja brez izgub).

PROSOJNICE



## BCNO in ohranitev odvisnosti

- Včasih ne obstaja dekompozicija v BCNO
- Primer: SBD
  - $SB \rightarrow D$ ,  $D \rightarrow B$
  - Ne moremo razcepiti v BCNO in obdržati  $SB \rightarrow D$

## Dekompozicija v BCNO

- Recimo, da  $R$  ni v BCNO
  - $X \subset R$  in  $A \in R$
  - $X \rightarrow A$  krši BCNO v  $R$
  - Razcepi  $R$  v  $R-A$  in  $XA$
- Če  $R-A$  ali  $XA$  ni v BCNO ponovi prvi korak
- Lastnosti algoritma
  - Vsaka op. razcepi relacijo  $\Rightarrow$  alg. se zaključi
  - Brezizgubni stik (ker velja vsaj odvisnost  $X \rightarrow A$ )
  - Koraki alg. tvorijo drevo (rekonstrukcija)

## Minimalno pokritje množice FO

- Minimalno pokritje množice  $F$  je  $G$  za katero velja:
  - Vsaka FO je oblike  $X \rightarrow A$  kjer  $A \in R$
  - Zaprtje  $F^+ = G^+$
  - Če iz  $G$  odvzamemo kakšno odvisnost ali atribut iz odvisnosti potem  $F^+ \neq G^+$
- Minimalno pokritje  $F$  je minimalno v naslednjem smislu:
  - Vsaka odvisnost  $X \rightarrow A$  ima najmanjši možen  $X$
  - $F$  ne vsebuje redundantne odvisnosti

## Dekompozicija v 3NO + ohranitev odvisnosti

- $F$  je minimalno pokritje  $R$ ;  $R_1, R_2, \dots, R_n$  je brezizgubna dekompozicija  $R$ ;  $R_i$  so v 3NO;  $F_i$  so projekcije  $F$  na  $R_i$ 
  - Določi množico odvisnosti, ki niso bile ohranjene oz. niso v uniji  $F_i$
  - Za vsako odvisnost  $X \rightarrow A$  iz množice kreiraj relacijo  $XA$
- Odvisnosti so očitno ohranjene
- Optimizacija: združimo  $X \rightarrow A$  v  $X \rightarrow A_1, A_2, \dots, A_k$

## Dekompozicije

- Dekompozicije:
  - v BCNO
  - v 3NO
- Algoritmi za dekompozicije.
- BCNO in 3NO oblika se pogosto uporablja za načrtovanje in optimizacijo podatkovnih baz.

## Dekompozicija v 3NO

- Prejšnji algoritem bi lahko uporabili tudi za dekompozicijo z brez-izgubnim stikom v 3NO
- Ta pristop ne zagotavlja ohranitev odvisnosti
- Enostavna razširitev algoritma omogoča dekompozicijo z brez-izgubnim stikom in z ohranitvijo odvisnosti

## Algoritem za iskanje min.pokritja

- 1)Daj FO v standardno obliko
- 2)Minimiziraj leve strani FO
  - Preveri vsak atribut iz leve strani FO iz  $F$ ; če ostane  $F^+$  enak potem ga lahko odstranimo
- 3)Odstrani redundantne FO
  - Preveri, če  $F^+$  ostane enak v primeru, da odstranimo eno odvisnost
  - Vrstni red procesiranja FO vpliva na rezultat!
  - Lahko imamo množico rezultatov

## Povzetek

- Obstajajo tudi dekompozicije:
  - v BCNO, v 3NO in
  - dekompozicija, ki ohranja FO.
- Če je relacija v BCNO, potem nima redundanc, ki jih je mogoče odkriti z FO. Zagotavljanje, da so relacije v BCNO je dobra hevristika.
- Če relacija ni v BCNO poskušamo narediti dekompozicijo v relacije, ki so v BCNO.
  - Ohranitev FO.
  - Brezizgubna dekompozicija.
  - Dekompozicija v 3NO.
- Dekompozicija včasih poslabša performanse poizvedb nad podatkovno bazo.