Iztok Savnik, Matjaž Kljun
OSNOVE PODATKOVNIH BAZ: SKRIPTA
2010/2011

(Zbirka Izbrana poglavja iz računalništva in informatike, št. 2)

Urednica zbirke: Petruša Miholič

Izdala:
Univerza na Primorskem
Primorski inštitut za naravosloven in tehnične vede Koper
Fakulteta za matematiko, naravoslovje in informacijske tehnologije

Založila:
Knjižnica za tehniko, medicino in naravoslovje – TeMeNa

Koper, 2011

UNIVERZA NA PRIMORSKEM
UNIVERSITÀ DEL LITORALE
UNIVERSITY OF PRIMORSKA

Titov trg 4, SI – 6000 Koper
Tel.: + 386 5 611 75 00
Fax.: + 386 5 611 75 30
E-mail: info@upr.si
http://www.upr.si

CIP - Kataložni zapis o publikaciji
Narodna in univerzitetna knjižnica, Ljubljana

004.65(075.8)(0.034.2)

SAVNIK, Iztok
Osnove podatkovnih baz [Elektronski vir] : skripta : 2010/2011 / Iztok Savnik, Matjaž Kljun. - Koper : Knjižnica za tehniko, medicino in naravoslovje - TeMeNa, 2011. - (Zbirka Izbrana poglavja iz računalništva in informatike ; št. 2)

Način dostopa (URL): http://temena.famnit.upr.si/files/files/OPB_knjiga.PDF

ISBN 978-961-93076-0-1

1. Kljun, Matjaž

255836928

Osnove Podatkovnih Baz

--skripta--

2010/2011

Pripravila:
Iztok Savnik
Matjaž Kljun

Kazalo

OSNOVE PODATKOVNIH BAZ

Iztok Savnik, FAMNIT

Osnovni podatki

- *Predavatelj:* Iztok Savnik
- *Asistent*: Matjaž Kljun
- *Vaje*: Avditorne in laboratorijske
- *Govorilne ure*: Po predavanju
- *Izpit*: pisni + domače naloge + ustni

Literatura

- *Knjiga*
  - Raghu Ramakrishnan, Johannes Gehrke, *Database Management Systems, McGraw-Hill, 3rd ed., 2007.*
- *Prosojnice:*
  - *Prosojnice so pripravljene na osnovi prosojnic in ostalega materiala „Cow Book“: R.Ramakrishnan, http://pages.cs.wisc.edu/~dbbook/*
- *Dodatna literatura*
  - Abraham Silberschatz, Henry F. Korth, S. Sudarshan, *Database System Concepts,* 3rd ed., McGraw-Hill, 1997.

Ocenjevanje

- *Domače naloge – 40%*
- *Pisni izpit – 40%*
  - *Kolokvija*
  - *Pisni izpit*
  - *> 50% !*
- *Ustni izpit – 20%*
  - *Pregled snovi*
  - *Pregled delov ocene*

Kaj je SUPB?

- Sistem za Upravljanje s Podatkovnimi Bazami
- Velika, povezana zbirka podatkov.
- Modelira okolje v realnem svetu.
  - Entitete (e.g., osebe,studenti,predmeti...)
  - Razmerja (e.g., Tone je vpisan na PB)
- SUPB je programski sistem za shranjevanje podatkov in upravljanje s podatkovnimi bazami.

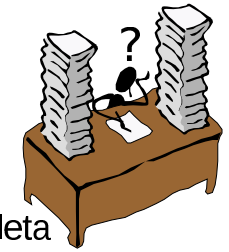
Datoteke vs. SUPB

- Aplikacije prenašajo velike količine podatkov med dinamičnim spominom in diskom (e.g., buffering, dostop na osnovi podatkovnih strani, 32-bit naslavljnje, itd.).
- Posebna koda za posamezne poizvedbe.
- Zaščita podatkov pred nekonsistenco, ki je lahko posledica večih hkratnih uporabnikov.
- Zaščita pred izpadom sistema.
- Varnost in kontrola dostopa.

Zakaj uporabljati SUPB?

- Podatkovna neodvisnost in učinkovit dostop do podatkov.
- Zmanjšan čas razvoja aplikacije.
- Podatkovna integriteta in varnost.
- Uniformno administriranje podatkov.
- Hkraten dostop, zaščita pred sistemskimi napakami.

Zakaj študij SUPB??



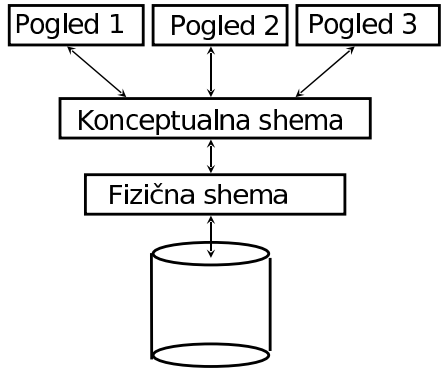
- Prehod iz računanja na informacije
  - "spodnja meja": preiskovanje datotečnega repozitorija, internet brez semantičnega spleta (kaos!)
  - "zgornja meja": iskanje na osnovi konceptualne mreže, znanstvene aplikacije, ...
- Podatkovne zbirke: narašča raznolikost in količina.
  - Digitalne knjižnice, interaktivni video, Genome projekt, ...
  - ... potreba po SUPB zelo narašča.
- SUPB pokrivajo večino računalništva
  - OS, diski, jeziki, teorija, AI, multimedia, logika

Podatkovni modeli

- Podatkovni model je zbirka konceptualnih gradnikov za opis podatkov.
- Shema je opis konkretne 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.

Nivoji abstrakcije

- Več pogledov, ena konceptualna shema in fizična shema.
  - Pogledi opisujejo kako uporabnik vidi podatke.
  - Konceptualna shema definira logično strukturo.
  - Fizična shema opisuje uporabljene datoteke in indekse.



- *Shema je definirana z uporabo DDL.*
- *Podatki se spreminjajo/poizvedujejo z DML.*

Primer: PB univerze

- Konceptualna shema:
  - *Študenti(sid: string, ime: string, login: string, starost: integer, povprečje:real)*
  - *Predmeti(pid: string, pime:string, točke:integer)*
  - *Vpis(sid:string, pid:string, ocena:string)*
- Fizična shema:
  - Relacije shranjene v neurejenih datotekah.
  - Indeks je definiran na prvem stolpcu relacije Študenti.
- Znanja shema (Pogled):
  - *Predmet_Info(pid:string,vpisani:integer)*

Podatkovna neodvisnost *

- Aplikacije se ne ukvarjajo s tem kako so podatki strukturirani in shranjeni.
- *Logična podatkovna neodvisnost*: Zaščita pred spremembami v logični strukturi podatkov.
- *Fizična podatkovna neodvisnost*: Zaščita pred spremembami fizične podatkovne stukture podatkov.

- *Ena od najbolj pomembnih prednosti uporabe SUPB !*

Kontrola sočasnega dostopa

- Sočasno izvajanje uporabniških programov je bistveno za dobre performanse SUPB.
  - Dostop do diska je pogost in relativno počasen, zato je pomembno da je procesor ves čas zaposlen na večih sočasnih programih uporabnikov.
- Prekrivajoče se akcije različnih uporabniških programov lahko vodijo v nekonsistentnost podatkov. Na primer, ček se je vpisal, ni pa se še izračunalo stanje računa.
- SUPB zagotavlja, da do podobnih problemov ne bo prišlo: uporabniki lahko mislijo, da delajo na eno-uporabniškem sistemu.

Transakcija: Primer programa SUPB

- Osnovni koncept je *transakcija*, ki je *atomarna* sekvenca akcij SUPB (branje/pisanje).
- Vsaka transakcija, ki se izvrši celotna mora pustiti PB v konsistentnem stanju če je PB 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 PB je odgovornost uporabnika !

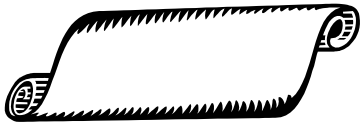
Razvrščanje sočasnih transakcij

- SUPB zagotavlja, da je izvajanje množice trans. $\{T_1, \ldots, T_n\}$ ekvivalentno nekem *zaporednem* izvajanju $T_1' \ldots T_n'$.
  - Pred pisanjem/branjem transakcija:
    1) zahteva zaklepanje vseh objektov, ki se jih dotika in
    2) počaka, da SUPB omogoči zaklepanje teh objektov. Vsi zaklenjeni objekti se sprostijo na koncu izvjanja transakcije.
    (Strikten 2-fazni protokol zaklepanja.)
- ❖ Zaporedno izvajanje:
  - Ideja: Če akcija T_i (pisanje X) vpliva na T_j (npr. bere X), bo ena izmed transakcij prva (npr. T_i) dobila in druga (T_j) bo morala počakati dokler prva ne konča; s tem je definirana urejenost transakcij.
- ❖ Smrtni objem:
  - Kaj če T_j že ima zaklenjen Y and T_i kasneje zahteva zaklepanje Y? (Smrtni objem - deadlock!) T_i ali T_j se ustavi in potem ponovno starta!

Zagotavljanje atomičnosti

- SUPB zagotavlja *atomičnost* (vse-ali-nič) četudi se zgodi sistemska napaka ali fizičen izpad sistema sredi transakcije.
- Ideja: Hranjenje dnevnika (*log*) (zgodovina) vseh akcij, ki jih izvaja SUPB medtem, ko izvaja množico transakcij:
  - Pred vsako spremembo v PB se vpiše ustrezen log zapis v log datoteko, pri tem pa zagotovimo, da se je zapis res fizično zapisal na disk. (*WAL protokol*; OS podpora često ni zadosti dobra)
  - Po sistemski napaki se vpliv delno izvršenih transakcij izniči tako, da se izničijo vse izvršene akcije transakcij. (WAL protokol zagotavlja, da ni zapisa v dnevniku, če ni sprememba že izvršena v PB!)

Dnevnik (Log)



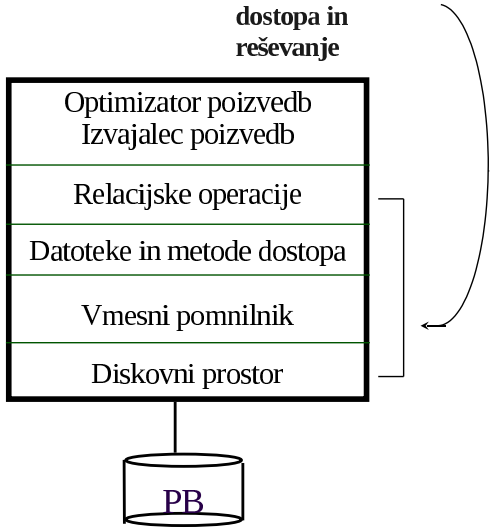
- Naslednje akcije se zapisujejo v dnevnik:
  - T_i *popravi objekt*: Stara in nova vrednost objekta.
    - Zapis dnevnika mora biti shranjen na disku pred podatkovno stranjo, ki vsebuje podatke objekta!
  - T_i *zaključi/prekine*: Zapis dnevnika shrani akcijo.
- Zapisi dnevnika so povezani med sabo z uporabo Xact ID; enostavno je vzpostaviti stanje pred Xact (npr., da se reši smrtnega objema).
- Dnevnik se pogosto podvoji zaradi varnosti in se praviloma arhivira na persistentnem mediju.
- Vse aktivnosti logiranja (aktivnosti za kontrolo sočasnega dostopa, zaklepanje/odklepanje, smrtni objemi,...) izvaja SUPB transparentno.

Baze osrečujejo sledeče ljudi ...

- Prodajalci SUPB.
- Razvijalci.
  - Načrtovalci informacijskih sistemov.
  - Programerji.
  - Webmaster.
- Končni uporabniki.
  - Uporabniki inf. Sistemov, vnašalci...
- Adminstrator SUPB (DBA)
  - Načrtovanje logiče/fizične sheme.
  - Varnost in autorizacija.
  - Dostopnost podatkov, vzpostavitev sistema po sistemski napaki.
  - Umerjanje SUPB po potrebi uporabnikov.

Struktura SUPB

- Tipični SUPB ima nivojsko arhitekturo.
- Slika ne prikazuje komponent za kontrolo hkratnega dostopa in recovery.
- To je samo ena izmed bolj pogostih arhitektur; vsak sistem ima svoje variacije.



Povzetek

- SUPB se uporablja za rokovanje (spreminjanje, poizvedovanje) velikih zbirk podatkov.
- Prednosti so ohranitev konsistentnega stanja PB po sistemskim napakam, hkraten dostop do SUPB, hiter razvoj aplikacij, podatkovna integriteta in zaščita.
- Nivoji abstrakcije nudijo podat. neodvisnost.
- SUPB tipično ima nivojsko arhitekturo.
- Adminstrator PB ima odgovorno službo in je dobro plačan ! ☺
- SUPB je eno od širših, bolj zanimivih področij CS

Relacijski podatkovni model

Iztok Savnik, FAMNIT

Zakaj študij relacijskega modela?

- Najširše uporabljan model.
  - Prodajalci: IBM, Informix, Microsoft, Oracle, Sybase, itd.
- "Legacy systems" v starih modelih.
  - Npr., IBM IMS
- Objektno-orientiran model
  - ObjectStore, Versant, Ontos
  - Združevanje: *objektno-relacijski model*
    - Informix Universal Server, UniSQL, O2, Oracle, DB2

Relacijska podatkovna baza

- *Relacijska PB:* *množica* *relacij.*
- *Relacija:* dva dela:
  - *Instanca* : *tabela*, ki ima vrstice in stolpce.
    #vrstic = *kardinalnost*, #stolpcev = *stopnja.*
  - *Shema* : določa ime relacije ter imena in tipe vseh stolpcev.
    - Studenti(*sid*: string, ime: string, *login*: string, starost: integer, po: real).
- Relacijo si lahko predstavljamo kot množico n-teric ali vrstic.

Primer relacije *študent*

sid	ime	login	star.	po
53666	Novak	novak@pef	18	7.2
53688	Kranjc	kranjc@ma	18	6.8
53650	Kranjc	kranjc@fi	19	9.3

- Kardinalnost = 3, stopnja = 5, vse vrstice različne
- Ali morajo biti vse vrstice v tabeli različne?

Relacijski povpraševalni jeziki

- Največja moč relacijskega modela:
  podpira enostaven povpraševalni jezik z veliko izrazno močjo.
- Vprašanja se lahko pišejo intuitivno, SUPB je odgovorna za učinkovito evaluacijo.
  - Razlog: natančna semantika relacijskih poizvedb.
  - Optimizator običajno temeljito preuredi poizvedbo in zagotavlja, da se odgovor ne spremni.

Povpraševalni jezik SQL

- Razvit na IBM (system R) leta 1970
- Potreba po standardu, ker ga uporablja veliko število proizvajalcev.
- Standardi:
  - SQL-86
  - SQL-89 (majhne spremembe)
  - SQL-92 (večje spremembe)
  - SQL-99 (večje spremembe)
  - SQL-07

Povpraševalni jezik SQL

- Vprašanje*: Poišči vse študente, ki so stari 18 let.*

```
SELECT *
FROM Studenti S
WHERE S.starost=18
```

sid	ime	login	star	po
53666	Novak	novak@pef	18	3.4
53688	Kranjc	kranjc@ma	18	3.2

- Vprašanje: *poišči imena in login študentov*.

```
SELECT S.ime,
S.login
FROM Studenti S
```


Povpraševanje nad večimi relacijami

- Kaj izračuna naslednje vprašanje?

```
SELECT S.ime, V.pid
FROM Studenti S, Vpis V
WHERE S.sid=V.sid AND V.ocena="10
```

Relaciji Studenti in Vpis:

sid	name	login	Star	po
53666	Novak	novak@pef	18	7.2
53688	Kranjc	kranjc@ma	18	6.8
53650	Kranjc	kranjc@fi	19	9.3

sid	pid	ocena
53831	Baze	5
53831	Matematika	7
53650	Geometrija	10
53666	Zgodovina	7

Dobimo:

S.ime	V.pid
Kranjc	Geometrija

Kreiranje relacij v SQL

- Kreiranje relacije *Studenti*. Tip se specificira za vsako polje. Pri vpisu podatkov v bazo, SPUB zahteva definiran tip podatka.

```
CREATE TABLE Studenti
        (sid: CHAR(20),
         ime: CHAR(20),
         login: CHAR(10),
         star: INTEGER,
         po: REAL)
```

- Tabela *Vpis* vsebuje podatke o predmetih, ki so jih vpisali študenti.

```
CREATE TABLE Vpis
        (sid: CHAR(20),
         pid: CHAR(20),
         ocena: CHAR(2))
```


Brisanje in spreminjanje relacij

```
DROP TABLE Studenti
```

- Zgornji ukaz izbriše relacijo Studenti -- shemo in zapise.

```
ALTER TABLE Studenti
        ADD COLUMN kraj: varchar(25)
```

❖ Shemo študenti spremenimo tako, da dodamo nov atribut *kraj* s katerim opišemo kraj stanovanja študenta.

Dodajanje in brisanje zapisov

- Vstavljanje enega zapisa:

```
INSERT INTO Studenti(sid, ime, login, star, po)
VALUES  (53688, 'Novak', 'novak@pef', 18, 3.2)
```

❖ Brišemo vse zapise, ki zadoščajo določenem pogoju, npr., ime="Novak":

```
DELETE
FROM Studenti S
WHERE S.ime = 'Novak'
```

Možne so variante ukazov z večjo izrazno močjo; več kasne

Integritetne omejitve (IC)

- IC: pogoj, ki mora biti izpolnjen za vsako n-terico relacije.
  - IC so določene pri definiciji sheme.
  - IC se preverjajo ob spreminjanju relacij.
- Legalna instanca relacije je takšna, ki zadovoljuje vse specificirane IC.
  - SPUB ne bi smel dopuščati nelegalnih instanc.
- Če SPUB preverja IC, potem so shranjeni podatki bližje pomenu v realnem svetu.
  - Izogibanje napakam pri vnosu!

Primarni ključ

- Množica atributov je ključ relacije če:
  1. Ne obstajata dva enaka zapisa z isto vrednostjo ključa.
  2. 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.
- Npr. *sid je* ključ za relacijo *Studenti.* (*ime*?) Množica {*sid, po*} je superključ.

Primarni in kandidatni ključ v SQL

- Običajno je na voljo več kandidatnih ključev (definiramo kot UNIQUE) izmed katerih izberemo en primarni ključ.

❖ Pravilno: "Za danega študenta in predmet imamo eno samo oceno."

❖ Nepravilno: "Študenti se lahko vpišejo na samo en predmet in dobijo eno samo oceno za ta predmet; ne smeta obstajati dva študenta, ki imata isto oceno."

❖ Če uporabimo IC nepravilno lahko onemogočimo vnos dejanskih podatkov iz realnega sveta!

```
CREATE TABLE Vpis
  (sid CHAR(20)
   pid  CHAR(20),
   ocena CHAR(2),
   PRIMARY KEY  (sid,pid) )
CREATE TABLE Vpis
  (sid CHAR(20)
   pid  CHAR(20),
   ocena CHAR(2),
   PRIMARY KEY  (sid),
   UNIQUE (pid, ocena) )
```


Tuji ključi in referenčna integriteta

- *Tuj ključ* : 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 omejitv tujih ključev upoštevana v PB potem pravimo, da dosežemo referenčno integriteto; ni "visečih referenc".
  - Naštej imena podatkovnih modelov z / brez referenčne integritete?
    - Link & HTML!

Tuji ključi v SQL

- Samo študenti, ki so v tabeli *Studenti* lahko nastopajo v relaciji *Vpis*.

```
CREATE TABLE Vpis
  (sid CHAR(20),  pid CHAR(20),  ocena CHAR(2),
   PRIMARY KEY  (sid,pid),
   FOREIGN KEY (sid) REFERENCES Studenti )
```

Vpis

sid	pid	grade
53666	Baze	5
53666	Matematika	7
53650	Geometrija	10
53666	Zgodovina	7

Studenti

sid	ime	login	Star	po
53666	Novak	novak@cs	18	7.2
53688	Kranjc	kranjc@ma	18	6.8
53650	Kranjc	kranjc@fi	19	9.3

Zagotavljanje referenče integritete

- Poglejmo si tabeli Studenti in Vpis; *sid* v tabeli Vpis je tuj ključ, ki referencira tabelo Studenti.
- Kaj naj se zgodi, če poskušamo dodati zapis, ki vsebuje neobstoječ sid v tabelo Vpis? *Zavrnitev zapisa!*
- Kaj naj se zgodi, če izbrišemo zapis tabele Studenti?
  - Pobrišejo se tudi pripadajoči zapisi v Vpis.
  - Ne dovoli se brisanje zapisa Studenti na katerega se referencirajo zapisi iz tabele Vpis.
  - Vrednosti zbrisanih *sid* se v tabeli Vpis postavi na privzeto vrednost.
  - SQL: Postavi vrednosti izbrisanih *sid* v Vpis na null vrednost, ki pomeni *neznano* ali *nenavedeno*.
- Podobno v primeru, da se ključ sid v tabeli Studenti spremeni.

Referenčna integriteta v SQL

- SQL/92 in SQL:1999 podpira vse 4 možnosti pri brisanju in popravljanju.
  - Privzeto je NO ACTION (*delete/update je zavrnjen*)
  - CASCADE (zbriši vse zapise, ki se sklicujejo na zbrisani zapis)
  - SET NULL / SET DEFAULT (postavi tuj ključ v zapisih, ki se referencirajo na izbrisani zapis)

```
CREATE TABLE Vpis
  (sid CHAR(20),
   pid CHAR(20),
   ocena CHAR(2),
   PRIMARY KEY (sid,pid),
   FOREIGN KEY (sid)
     REFERENCES Studenti
        ON DELETE CASCADE
        ON UPDATE SET DEFAULT )
```


Od kje so prišle IC omejitve?

- IC so osnovane na pomenu okolja, ki ga modeliramo z relacijskim modelom.
- Lahko preverimo podatkovno bazo ali je v skladu z integritetnimi omejitvami . Ne moremo pa NIKOLI sklepati, da je omejitev pravilna z opazovanjem modela (instance).
  - IC je izjava o vseh možnih instancah!
  - Iz primera vidimo, da ime ni ključ. Določitev, da sid je ključ je prepuščeno nam.
- Ključ in tuj ključ najbolj pogoste IC; bolj splošne IC tudi obstajajo.

Relacijski model: pregled

- Tabularna predstavitev podatkov.
- Enostavno in intuitivno, trenutno najbolj pogosto uporabljan model.
- Integritetne omejitve lahko vnesemo v podatkovno bazo na osnovi lastnosti modeliranega podatkovnega okolja. SUPB preveri veljavnost omejitev.
  - Pomembni omejitvi: primarni in tuj ključ.
  - Vedno so definirane domene za atribute.
- Na voljo je močan in dokaj naraven povpraševalni jezik.
- Pravila za prevajanje ER v relacijski model.

1. Naloga

Narišite relacijski podatkovni model baze podatkov, ki obsega podatke o poslovanju naftne družbe na ravni poslovanja njenih bencinskih servisov (črpalk). Pri tem poznamo naslednja dejstva:

- naftna družba ima v lasti številne bencinske servise, ki so razporejeni po celi državi,
- vsak bencinski servis ponuja prevzem različnih vrst goriva (po domače tankanje) na več točilnih mestih,
- vsako točilno mesto ponuja natanko eno vrsto goriva,
- naftna družba spremlja zaloge posameznih vrst goriva za posamezne bencinske črpalke,
- vsaka bencinska črpalka preko naročil, v katerih opredeli številko naročila, datum, vrsto goriva in zahtevano količino, posreduje svoje zahteve naftni družbi,
- naftna družba naročila posameznih becinskih črpalk izpolni z dobavo goriva preko cistern, ki jih ima v lasti,
- enkratna dobavo goriva na bencinski servis ni nujno, da v celoti izpolnjuje neko naročilo,
- zaposleni v naftni družbi se (za naš primer) delijo le na šoferje cistern in zaposlene na bencinskih črpalkah,
- vsaka bencinska črpalka ima tudi svojega vodjo.

Za izdelani model sestavite seznam SQL stavkov.

2. Naloga

Narišite relacijski podatkovni model baze podatkov, ki obsega podatke o letalskem potniškem prometu. Pri tem poznamo naslednja dejstva:

- v letalskem potniškem prometu sodeluje več letalskih družb s svojimi letali,
- vsako letalo pripada neki letaski družbi,
- piloti in drugo osebje (stevardese), ki sodelujejo pri posameznih poletih, so prav tako zaposleni pri letalski družbi,
- vsak polet uporablja letalo neke letalske družbe,
- potniki preko rezervacij lahko rezervirajo sedeže na posameznem poletu,
- za vsako rezervacijo poznamo tudi številko sedeža in potniški razred na letalu,
- za vsak polet spremljamo tako podatke o vzletu (letališče, datum in čas) ter podatke o pristanku (letališče, datum in čas),
- vsako letališče ima natančno opredeljene omejitve glede pristankov posameznih vrst (oz. tipov) letal.

Za izdelani model sestavite seznam SQL stavkov.

3. Naloga

Narišite relacijski podatkovni model baze podatkov, ki obsega podatke o poslovanju isposojevalnic motornih vozil. Pri tem poznamo naslednja dejstva:

- podjetje ima več poslovalnic po različnih državah
- vsaka poslovalnica ima več zaposlenih, od katerih vsak dela na točno eni poslovalnici
- zaposleni so razdeljeni v več kategorij (npr. mehanik)
- rezervacija vozila vsebuje podatke o vozilu, datumu prejema, datumu vrnitve, poslovalnicah prejema in vrnitve in osebi, ki je rezervirala vozilo
- vsak posameznik lahko rezervira več vozil,
- vozila so različnih tipov (eno vozilo je tipa tovornjak, eno avtomobil, …)
- vsako vozilo po končani rezervaciji pripada poslovalnici, kamor se je vrnilo, kar pomeni, da vsako vozilo pripada le eni poslovalnici
- o vozilu hranimo poleg modela, tipa in št. karoserije tudi št. prevoženih kilometrov, datum naslednjega tehničnega pregleda, datum naslednjega servisa in št. zavarovalne police

Za izdelani model sestavite seznam SQL stavkov.

4. Naloga

Opiši podatkovno okolje študentskega informacijskega sistema z relacijskim podatkovnim modelom. Predstavi naslednja dejstva:

- predmeti, ki jih študenti poslušajo so lahko bodisi obvezni ali izbirni.
- študenti so lahko vpisani na dodiplomski ali podiplomski stopnji
- za vsakega študenta vemo katero smer študija in kateri letnik posluša
- pri vsakem predmetu vemo, v katerem letniku, na kateri stopnji in na kateri smeri se izvaja, v katerem kvartalu se izvaja, kolikšno je število ur predavanj in vaj, kateri izvajalci ga izvajajo (predmet ima več izvajalcev in vsak izvajalec je lahko pri več predmetih)
- za vsakega zaposlenega vemo ali je pri predmetu predavatelj ali asistent
- študenti imajo v svojem indeksu predmete in za vsakega zapis ocene

Relacijska algebra

Iztok Savnik, FAMNIT

Relacijski povpraševalni jeziki

- *Povpravševalni jezik:* Omogoča urejanje podatkov in poizvedovanje po podatkih v podatkovni bazi.
- Relacijski model podpira enostavne PJ z veliko izrazno močjo:
  - Formalne osnove v logiki.
  - Omogoča optimizacijo poizvedb.
- Povpraševalni jezik != Programski jezik !
  - PJ ni "Računsko kompleten".
  - PJ ni namenjen za kompleksne izračune.
  - PJ podpira enostaven in učinkovit dostop do velikih zbirk podatkov.

Formalni relacijski povpraševalni jeziki

- Dva formalna (matematična) jezika tvorita osnovo za "realne" povpraševalne jezke (npr. SQL) in njihovo implementacijo:
  - *Relacijska algebra*: Bolj proceduralen jezik, uporaben za predstavitev plana izvajanja poizvedb.
  - *Relacijski račun*: Omogoča uporabnikom opisati kaj želijo in ne toliko kako to izračunati – deklarativni jezik.

Osnove

- Poizvedba je izvršena nad instancami relacij in rezultat poizvedbe je instanca neke relacije.
  - *Sheme vhodnih* relacij so fiksne.
  - *Shema rezultata* je tudi fiksna -- določi se s pravili gradnikov povpraševalnega jezika.
- Notacija osnovana na poziciji oz. imenih atributov:
  - Notacija osnovana na poziciji primernejša za programe; notacija osnovana na imenih je bolj berljiva.
  - Obe se uporabljata v SQL.

Primeri relacij

- Relaciji "Mornarji" in "Rezervacije" za naše primere.
- Uporabljali bomo notaciji osnovani na poziciji in imenih.
- Imena atributov v vmesnih in končnem rezutatu poizvedb se podedujejo od vhodnih relacij.

R1

mid	lid	dan
22	101	10/10/96
58	103	11/12/96

S1

mid	mime	ocena	star
22	novak	7	45.0
31	kranjc	8	55.5
58	petelin	10	35.0

S2

mid	mime	ocena	star
28	volk	9	35.0
31	kranjc	8	55.5
44	jauk	5	35.0
58	petelin	10	35.0

Relacijska algebra

- Osnovne operacije
  - *Selekcija* (σ) Izbere podmnožico n-teric iz relacije.
  - *Projekcija* (π) Izbere določene stolpce relacije.
  - *Produkt* ($\times$) Omogoča kombiniranje dveh relacij.
  - *Razlika* ($-$) N-terice iz prve in ne iz druge relacije.
  - *Unija* (ξ) N-terice iz obeh relacij.
- Dodatne operacije
  - Presek, *Stik*, Deljenje, Preimenvanje.
  - Niso nujne, so pa ZELO (!) koristne.
- Vsaka operacija vrne relacijo kot rezultat.
  - Operacije se lahko sestavljajo – funkcionalni jezik.

Projekcija

- Izbere atribute, ki so v *listi projekcije* iz relacije.
- *Shema* rezultata vsebuje samo atribute, ki so v listi projekcije z istimi imeni kot v vhodni relaciji.
- Projekcija mora odstraniti duplikate ! (Zakaj??)
  - Opomba: relani sistemi tipično ne odstranijo duplikate, če uporabnik tega ne zahteva. (Zakaj ne?)

mime	ocena
volk	9
kranjc	8
jauk	5
petelin	10

$$\pi_{mime,ocena}(S2)$$

star
35.0
55.5

$$\pi_{star}(S2)$$

Selekcija

- Izbere vrstice, ki zadoščajo *pogoju selekcije*.
- Ni duplikatov v rezulatu. (Zakaj?)
- *Shema* rezulata je identična shemi vhodnih relacij.
- *Relacija*, ki je rezultat je lahko vhodna relacija drugi relacijski operaciji! (*Kompozicija operacij.*)

mid	mime	ocena	star
28	volk	9	35.0
58	petelin	10	35.0

$$\sigma_{ocena>8}(S2)$$

mime	ocena
volk	9
petelin	10

$$\pi_{mime,ocena}(\sigma_{ocena>8}(S2))$$

Unija, Presek, Razlika

- Vse operacije so binarne in vhodni relaciji morata biti *unija-kompatibilni*:
  - Enako število atributov.
  - `Pripadajoča' polja imajo enake tipe.
- Kaj je *shema* rezultata?

mid	mime	ocena	star
22	novak	7	45.0
31	kranjc	8	55.5
58	petelin	10	35.0
44	jauk	5	35.0
28	volk	9	35.0

$S1 \cup S2$

mid	mime	ocena	star
22	novak	7	45.0

$S1 - S2$

mid	mime	ocena	star
31	kranjc	8	55.5
58	petelin	10	35.0

$S1 \cap S2$

Produkt

- *Kartezijski produkt*: vsaka vrstica S1 se poveže z vsako vrstico R1.
- *Shema rezultata* ima po en atribut za vsak atribut relacij S1 in R1; imena od operandov.
  - *Konflikt*: S1 in R1 h imata atribut *mid*.

(mid)	mime	ocena	star	(mid)	lid	day
22	novak	7	45.0	22	101	10/ 10/ 9
22	novak	7	45.0	58	103	11/ 12/ 9
31	kranjc	8	55.5	22	101	10/ 10/ 9
31	kranjc	8	55.5	58	103	11/ 12/ 9
58	petelin	10	35.0	22	101	10/ 10/ 9
58	petelin	10	35.0	58	103	11/ 12/ 9

- *Preimenovanje*: $\rho(C(1 \to mid1, 5 \to mid2), S1 \times R1)$

Stik (Join)

- *Stik s pogojem*: $R \bowtie_c S = \sigma_c (R \times S)$

(mid)	mime	ocena	star	(mid)	lid	dan
22	novak	7	45.0	58	103	11/ 12/ 96
31	kranjc	8	55.5	58	103	11/ 12/ 96

$S1 \bowtie_{S1.mid < R1.mid} R1$

- *Shema rezultata:* enako kot kartezijski produkt.
- Manj n-teric kot produkt; da se izračunati hitreje.
- Včasih ga imenujejo *theta-stik.*

Stiki

- *Equi-Stik*: Poseben primer stika, kjer je pogoj stika uporablja samo pogoj enačaj.

mid	mime	ocena	star	lid	dan
22	novak	7	45.0	101	10/ 10/ 96
58	petelin	10	35.0	103	11/ 12/ 96

$S1 \bowtie_{mid} R1$

- *Shema rezultata:* podobno kot kartezijski produkt; samo ena vrednost enačenih atributov je v rezultatu.
- *Naravni Stik*: Equi-Stik po vseh skupnih atributih.

Deljenje

- Ni osnovna operacija; uporabna za izražanje vprašanj kot na primer:
  - *Poišči vse mornarje, ki so rezervirali vse ladje.*
- Naj ima *A* dva atributa *x* in *y*; *B* pa samo en atribut *y*:
  - $A/B = \{\langle x\rangle \mid \exists \langle x, y\rangle \in A \ \ \forall \langle y\rangle \in B\}$
  - ***A/B* vsebuje vse n-terice *x* (mornarji) tako da za vsako n-terico *y* (ladja) v *B*, obstaja n-terica *xy* v *A*.**
  - *Ali*: Če množica vrednosti *y* (ladje) povezana z vrednostjo *x* (mornarji) v *A* vsebuje vse vrednosti *y* v *B*, potem je vrednost *x* v *A/B*.
- V splošnem sta *x* in *y* lahko poljubna seznama atributov; *y* je seznam atrributov v *B*, in *x* *y* je seznam atributov v *A*.

Primer deljenja A/B

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

B1

pno
p2
p4

B2

pno
p1
p2
p4

B3

sno
s1
s2
s3
s4

A/B1

sno
s1
s4

A/B2

sno
s1

A/B3

Izražanje A/B z osnovnimi operacijami

- Deljenje ni nujno potrebna operacija; uporabna bljižnica.
  - To je načeloma res tudi za stike, čeprav omogočajo stiki učinkovito implementacijo poizvedb.
- *Ideja*: *A/B* = izračunaj vse vrednosti *x*, ki niso izločeni z vrednostjo *y* v *B*.
  - *x* je izločena z *y* v primeru, da z dodajanjem vrednosti *y* iz *B* dobimo n-terico *xy*, ki ni v *A*.

Izločene vrednosti *x*: $\pi_x((\pi_x(A)\times B)-A)$

A/B: $\pi_x(A)$ – Izločene vrednosti

Poišči imena mornarjev, ki so rezervirali ladjo #103

- Rešitev 1: $\pi_{mime}((\sigma_{lid=103}\text{Rezervacije})\bowtie Mornarji)$

❖ Rešitev 2: $\rho(Temp١,\sigma_{lid=١٠٣}\text{Rezervacije})$

$\rho\ (Temp٢,\ Temp١ \bowtie Sailors)$

$\pi_{mime}(Temp٢)$

❖ Rešitev 3 $\pi_{mime}(\sigma_{lid=103}(\text{Rezervacije}\bowtie Mornarji)$

Poišči imena vseh mornarjev, ki so rezervirali rdečo ladjo.

- Informacije o barvah ladij so dostopne v relaciji Ladje; potrebujemo še en stik:

$$\pi_{mime}((\sigma_{barva=rdeca}Ladje)\bowtie Rezervacije\bowtie Mornarji)$$

  - ❖ Bolj učinkovita rešitev:

$$\pi_{mime}(\pi_{mid}((\pi_{lid}\sigma_{barva=rdeca}Ladje)\bowtie Rez)\bowtie Mornarji)$$

Optimizator poizvedb bi našel to rešitev ob dani prvi poizved

Poišči vse mornarje, ki so rezervirali rdečo ali zeleno ladjo.

- Identificiramo vse rdeče in zelene ladje in potem poiščemo mornarje, ki so rezervirali eno izmed izbranih ladij:

$$\rho(Temp,(\sigma_{barva=rdeca\vee barva=zelena}Ladje))$$

$$\pi_{mime}(Temp\bowtie Rezervacije\bowtie Mornarji)$$

❖ Temp se da definirati z unijo! (Kako?)

Kaj se zgodi, če je $\vee$ zamenjan z $\wedge$?

Poišči mornarje, ki so rezervirali rdečo in zeleno ladjo.

- Prejšnji način ne deluje.
- Poiščemo mornarje, ki so rezervirali rdeče ladje, mornarje, ki so rezervirali zelene ladje in potem naredimo presek.

$$\rho(Temp1,\pi_{mid}((\sigma_{barva=rdeca}Ladje)\bowtie Rezervacije))$$

$$\rho(Temp2,\pi_{sid}((\sigma_{barva=zelena}Ladje)\bowtie Rezervacije))$$

$$\pi_{mime}((Temp1\cap Temp2)\bowtie Mornarji)$$

Poišči imena mornarjev, ki so rezervirali vse ladje

- Uporaba deljenja; sheme vhodnih relacij morajo biti pazljivo izbrane:

$$\rho(Temp,(\pi_{mid,lid}Rezervacije)/(\pi_{lid}Ladje))$$

$$\pi_{mime}(Temp\bowtie Mornarji)$$

... mornarje, ki so rezervirali vse ladje 'Delfin':

..... $/\pi_{lid}(\sigma_{lime=delfin}Ladje)$

Ponovitev

- Relacijski model ima formalne povpraševalne jezike, ki so enostavni in imajo veliko izrazno moč.
- Relacijska algebra je proceduralen jezik; uporabna je interno predstavitev vprašanj.
- Veliko načinov za izražanje enega samega stavka; optimizator izbere tistega, ki poišče rezultat najhitreje.

1. Relacijska algebra - LETALSKI POTNIŠKI PROMET
Vzemimo naslednjo shemo:

```
LETALISCA
   _IdLE: integer,
    ime: varchar(30),
    kraj: varchar(50),
    drzava: varchar(50)
LETALO
   _IdTL: integer,
    tip: varchar(20),
    opis: varchar(250),
    OznakaProizv: varchar(20)
PRISTANE
   _IdLE: integer,
   _IdTL: integer,
    DatumOd: date
```

Atributi, ki predstavljajo ključ so podčrtani.
Relacija PRISTANE vsebuje podatke o tem, kateri tipi letal lahko pristanejo na posameznih letališčih in od kdaj.

Zapišite naslednje poizvedbe v relacijski algebri.

a) Poišči vsa imena letališč, na katerih lahko pristane letalo tipa B747.
b) Poišči vse IdLE letališč, na katerih lahko pristane letalo tipa B747 ali B748.
c) Poišči vse IdLE letališč, ki so ali v Avstriji ali pa na njih lahko pristane letalo tipa B747.
d) Poišči vse IdLE letališč, na katerih lahko pristane letalo tipa B747 in tipa B748.
e) Poišči vse IdLE letališč, na katerih lahko pristanejo vsi tipi letal.

2. Relacijska algebra – NAFTNA DRUŽBA
Vzemimo naslednjo shemo:

```
CISTERNE
   _IdC: integer,
    SerijskaSt: integer,
    kapaciteta: integer,
    DtServisa: date
NAROCILA
   _IdN: integer,
    St:integer,
    Datum: date,
    Kolicina: integer
DOBAVA
   _IdN: integer,
   _IdC: integer,
    Datum: date,
    Kolicina: integer
```

Atributi, ki predstavljajo ključ so podčrtani.
Relacija DOBAVA vsebuje podatke o opravljenih dobavah oz. prevozih cistern za izpolnitev posameznih naročil. Za izpolnitev enega naročila je lahko potrebno tudi več dobav oz. prevozov.

Zapišite naslednje poizvedbe v relacijski algebri.

a) Poišči vse številke naročil, za katere je pri dobavi goriva sodelovala cisterna s serijsko številko 123456.
b) Poišči vse IdN naročil, ki jih je izpolnila cisterna s serijsko številko 123456 ali 123457.
c) Poišči vse IdN naročil, ki ali zahtevajo količino 10.000 enot ali pa jih je izpolnila cisterna s serijsko številko 123456.
d) Poišči vse IdN naročil, ki jih je izpolnila cisterna s serijsko številko 123456 in cisterna s serijsko številko 123457.

3. Imamo naslednji shemo

```
Avto
  idA
  Model
  Barva
  StVrat
Pregled
  idO
  idA
Operater
  idO
  Ime
  DelDoba
```

a) Poišči imena vseh operaterjev ki so pregledali rdeče avte
b) Poišči imena mornarjv, ki so pregledali idA 2
c) Poišči imena vseh operaterjev ki so pregledali rdeče ALI zelene avte pri čemer uporabimo preimenovanje
d) Poišči imena vseh operaterjev ki so pregledali rdeče IN zelene avte pri čemer uporabimo preimenovanje
e) Poišči imena operaterjev ki so pregledali vse avtomobile
f) Poišči imena operaterjev ki so pregledali vse avtomobile modela Golf

SQL: Poizvedbe, Integritetne omejitve, Prožilci

Iztok Savnik, FAMNIT

Primeri relacij

- Instance ta bel *Rezervacije* in *Mornarji* bodo uporabljane v primerih.
- Če bi ključ tabele *Rezervacije* vseboval samo *mid* in *lid*, kakšen pomen bi imela relacija?

R1

mid	lid	dan
22	101	10/10/96
58	103	11/12/96

S1

mid	mime	ocena	star
22	novak	7	45.0
31	kranjc	8	55.5
58	petelin	10	35.0

S2

mid	mime	ocena	star
٢٨	volk	٩	٣٥.٠
٣١	kranjc	٨	٥٥.٥
٤٤	jauk	٥	٣٥.٠
٥٨	petelin	١٠	٣٥.٠

Osnovna SQL poizvedba

```
SELECT    [DISTINCT] seznam-izbire
FROM      seznam-relacij
WHERE     pogoj-izbire
```

- *seznam-relacij* Seznam relacij (lahko z uporabo spremenljivk).
- *seznam-izbire* Seznam atributov, ki so rezultat poizvedbe ali shema tabele -rezulata.
- *pogoj-izbire* Logični pogoj (Atr *op* const, Atr1 *op* Atr2, kjer je *op* eno izmed $<, >, =, \leq, \geq, \neq$) ; primerjave so povezane z logičnimi operacijami AND, OR in NOT.
- DISTINCT je opcijska ključna beseda, ki pove da naj se iz rezulatata odstranijo duplikati. Privzeto duplikati *niso* odstranjeni!

"Konceptualna" evaluacijska strategija

- Pomen SQL poizvedbe se lahko izrazi z naslednjo strategijo evaluacije:
  - Izračunaj Kartezijski produkt *seznama-relacij*.
  - Izloči tiste n-terice, ki ne izpolnjujejo *pogoja-izbire*.
  - Izloči atribute, ki niso v *seznama-izbire*.
  - Če je specificiran DISTINCT potem se izločijo duplikati.
- Opisana strategija je zelo verjetno najmanj učinkovit način evaluacije poizvedbe!
- Optimizator bo poiskal najbolj učinkovito strategijo, ki izračuna *isti odgovor*.

Primer konceptualne evaluacije

```
SELECT  S.mime
FROM    Mornarji M, Rezervacije R
WHERE   M.mid=R.mid AND R.lid=103
```

(mid)	mime	ocena	star	(mid)	lid	dan
22	novak	7	45.0	22	101	10/ 10/ 96
22	novak	7	45.0	58	103	11/ 12/ 96
31	kranjc	8	55.5	22	101	10/ 10/ 96
31	kranjc	8	55.5	58	103	11/ 12/ 96
58	petelin	10	35.0	22	101	10/ 10/ 96
58	petelin	10	35.0	58	103	11/ 12/ 96

Spremenljivke

- Potrebne so samo v primeru, da ista relacija uporablja dvakrat. Prejšnje vprašanje se lahko napiše tudi na sledeč način:

```
SELECT  M.mime
FROM    Mornarji M, Rezervacije R
WHERE   M.mid=R.mid AND lid=103
```

Dober stil: Vedno uporabljaj spremenljivke !

ALI

```
SELECT  mime
FROM    Mornarji, Rezervacije
WHERE
Mornarji.mid=Rezervacije.mid
AND lid=103
```

Poišči mornarje, ki so rezervirali vsaj eno ladjo

```
SELECT  M.mid
FROM    Mornarji M, Rezervacije R
WHERE   M.mid=R.mid
```

- Kaj se zgodi, če dodamo DISTINCT ?
- Kako vpliva zamenjava *M.mid* z *M.mime* na izvajanje SELECT stavka? Kaj če dodamo DISTINCT ?

Izrazi in nizi

```
SELECT  M.star, star1=M.star-5, 2*M.star AS star2
FROM    Mornarji M
WHERE   M.mime LIKE 'B_%N'
```

- Ilustracija uporabe aritmetičnih izrazov in ujemanja vzorcev pri nizih: *poišči trojice (starost mornarjev in dva polja opisana z izrazi) za vse mornarje katerih ime se začne z B, konča z N in vsebuje vsaj tri znake.*
- AS in = sta dva načina poimenovanja polj v rezultatu.
- LIKE se uporablja za primerjanje nizov. `_' pomeni katerikoli znak in `%' pomeni 0 ali več poljubnih znakov.

Poišči id-je mornarjev, ki so rezervirali rdečo ali zeleno ladjo

- UNION: Uporablja se za izračun unije dve *unija-kompatibilnih* množic n-teric (ki so rezultat SQL poizvedbe).
- Kaj dobimo, če zamenjamo OR z AND v prvi verziji poizvedbe?
- Uporaba EXCEPT (Kaj dobimo, če UNION zamenjamo z EXCEPT?)

```
SELECT M.mid
FROM Mornarji M, Ladje L, Rezervacije R
WHERE M.mid=R.mid AND R.lid=L.lid
AND (L.barva='rdeca' OR L.barva='zelena
```

```
SELECT M.mid
FROM Mornarji M, Ladje L,
Rezervacije R
WHERE M.mid=R.mid AND L.lid=R.lid
        AND L.barva='rdeca'
UNION
SELECT M.mid
FROM Mornarji M, Ladje L,
Rezervacije R
WHERE M.mid=R.mid AND L.lid=R.lid
        AND L.barva='zelena'
```


Poišči id-je mornarjev, ki so rezervirali rdečo in zeleno ladjo

- INTERSECT: lahko uporabljamo nad *unija-kompatibilnimi* množicami n-teric.
- Vključena je v SQL/92 standard; nekateri sistemi operacije ne podpirajo.

```
SELECT M.mid
FROM Mornarji M, Ladje L1, Rezervacije R1,
        Ladje L2, Rezervacije R2
WHERE M.mid=R1.mid AND R1.lid=L1.lid
  AND M.mid=R2.mid AND R2.lid=L2.lid
  AND (L1.barva='rdeca'
  AND L2.barva='zelena')
```

Ključ!

```
SELECT M.mid
FROM Mornarji M, Ladje L, Rezervacije R
WHERE M.mid=R.mid AND L.lid=R.bid
        AND L.barva='rdeca'
INTERSECT
SELECT M.mid
FROM Mornarji M, Ladje L, Rezervacije R
WHERE M.mid=R.mid AND L.lid=R.lid
        AND L.barva='zelena'
```


Vgnezdena vprašanja

Poišči imena mornarjev, ki so rezervirali ladjo #103:

```
SELECT M.mime
FROM Mornarji M
WHERE M.mid IN (SELECT R.mid
                FROM Rezervacije R
                WHERE R.lid=103)
```

- Zelo izrazna lastnost SQL: stavek WHERE lahko vsebuje SQL poizvedbo !
  - Kot tudi stavka FROM an HAVING.
- Mornarji, ki niso rezervirali ladje #103: NOT IN.
- Semantika vgnezdenih poizvedb:
  - Vgnezdene zanke: *Za vsakega mornarja, preveri pogoj poizvedbe, ki vsebuje vgnezdeno poizvedbo.*

Vgnezdena vprašanja (2)

Poišči imena mornarjev, so rezervirali ladjo #103:

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

- EXISTS: primerjava množice s prazno množico.
- Vgnezdeno vprašanje se izvede za vsakega mornarja.
- Če je uporabljen UNIQUE in je * zamenjana z *R.lid*, potem iščemo mornarje, ki imajo največ eno rezervacijo ladje #103. (UNIQUE preveri obstoj duplikatov; * pomeni vse atribute. Zakaj moramo zamenjati * z *R.lid*?)

Operacije za primerjanje množic

- Spoznali smo že IN, EXISTS in UNIQUE. Obstajajo še NOT IN, NOT EXISTS in NOT UNIQUE.
- Na voljo so še: *op* ANY, *op* ALL, *op* IN

>,<,=,≥,≤,≠

- Poišči mornarje kateri imajo oceno večjo od vseh mornarjev z imenom "miha":

```
SELECT  *
FROM  Mornarji M
WHERE  M.ocena > ANY  (SELECT  M2.ocena
                       FROM  Mornarji M2
                       WHERE M2.mime='miha')
```


Poizvedbe z INTERSECT : uporaba IN

Poišči id-je mornarjev, ki so rezervirali rdečo in zeleno ladjo:

```
SELECT  M.mid
FROM  Mornarji M, Ladje L, Rezervacije R
WHERE  M.mid=R.mid AND R.lid=L.lid AND L.barva='rdeca'
    AND M.mid IN  (SELECT  M2.mid
                   FROM  Mornarji M2, Ladje L2, Rezervacije R2
                   WHERE  M2.mid=R2.mid AND R2.lid=B2.lid
                      AND  L2.barva='zelena')
```

- Podobno, EXCEPT poizvedbe lahko prepišemo z uporabo NOT IN.
- Iskanje *imen* (ne *mid*) Mornarjev, ki so rezervirali rdečo in zeleno ladjo samo zamenjaj *M.mid* z *M.mime* v stavku SELECT. (Kaj je z INTERSECT poizvedbo?)

Deljenje v SQL

Poišči mornarje, ki so rezervirali vse ladje.

- Težja pot; brez EXCEPT:

(1)

```
SELECT  M.mime
FROM  Mornarji M
WHERE  NOT EXISTS
          ((SELECT  L.lid
            FROM  Ladje L)
           EXCEPT
            (SELECT  L.lid
             FROM  Rezervacije R
             WHERE R.mid=M.mid
                AND R.lid=L.lid))
```

(2)

```
SELECT  M.mime
FROM  Mornarji M
WHERE  NOT EXISTS  (SELECT  L.lid
                    FROM  Ladje L
                    WHERE  NOT EXISTS  (SELECT  R.lid
                                        FROM  Rezervacije R
                                        WHERE  R.lid=L.lid
                                           AND R.mid=M.mid)
```

Mornarji tako, da ...

ne obstaja ladja brez da nebi ...

obstajala rezervacija M ladje L

Agregacijske operacije

- Pomembna razširitev relacijske algebre.

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

single column

```
SELECT  COUNT (*)
FROM  Mornarji M

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

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

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

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

Poišči imena in starost najstarejšega mornarja (-ev)

- Prva poizvedba ni legalna! (Razlog malce kasneje, ko si bomo ogledali GROUP BY.)
- Tretja poizvedba je ekvivalentna drugi in je legalna v okviru SQL/92 standarda; ni podprta v nekaterih sistemih.

```
SELECT  M.mime, MAX (M.star)
FROM  Mornarji M
```

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

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


Motivacija za grupiranje

- V prejšnjih primerih so se agregacijske operacije izvajale nad celotnimi tabelami. Včasih potrebujemo kaj izračunati nad *skupinami* n-teric.
- Primer: P*oišči starost najmlajših mornarjev za vsako oceno.*
  - V splošnem ne vemo koliko ocen obstaja in kakšne so vrednosti ocene.
  - Recimo, da vemo da so vrednosti ocene od 1 do 10; lahko napišemo 10 vprašanj kot je naslednje: ☺

For $i = 1, 2, \ldots, 10$:

```
SELECT  MIN (M.star)
FROM  Mornarji M
WHERE  M.ocena = i
```


Poizvedbe z GROUP BY in HAVING

```
SELECT      [DISTINCT]  seznam-izbire
FROM        seznam-relacij
WHERE       pogoj-izbire
GROUP BY    seznam-skupine
HAVING      pogoj-skupine
```

- *seznam-izbire* vsebuje: (i) imena atributov (ii) izraze z agregacijskimi operacijami (npr., MIN (*M.star*)).
  - seznam-izbire (i): mora biti podmnožica *seznama-skupine*. Intuitivno, predstavlja vsaka n-terica rezultata skupino n-teric.
  - *Skupina* je množica n-teric, ki ima isto vrednost vseh atributov iz *seznama-skupine*.

"Konceptualna" evaluacija

- Najprej izračunamo kartezijski produkt relacij iz *seznam-relacij*.
- Izločimo n-terice, ki ne izpolnjijejo pogoja *pogoj-izbire*.
- Izločimo nepotrebne atribute.
- Preostale n-terice se razvrstijo v skupine glede na vrednost atributov iz *seznama-skupine*.
- Iz množice n-teric, ki predstavljajo skupine se izločijo tiste, ki ne zadoščajo *pogoju-skupine.* Izrazi v *pogoju-skupine* morajo imeti *eno vrednost za celotno skupino!*
- Atributi v *pogoju-skupine* so bodisi argumenti v agregacijskih funkcijah ali pa se pojavijo v *seznamu-skupine*.
- Ena n-terica se generira za eno od izbranih skupin.

Poišči starost najmlajšega mornarja, ki je star več kot 18, v skupinah, ki pripadajo ocenenam in vsebujejo vsaj dva takšna mornarja.

```
SELECT  M.ocena,  MIN (M.star)
          AS minstar
FROM  Mornarji M
WHERE  M.star >= 18
GROUP BY  M.ocena
HAVING  COUNT (*) > 1
```

Sailors instance:

mid	mime	ocena	star
22	novak	7	45.0
29	pevec	1	33.0
31	kranjc	8	55.5
32	andrej	8	25.5
58	petelin	10	35.0
64	tom	7	35.0
71	kos	10	16.0
74	tom	9	35.0
85	miha	3	25.5
95	janez	3	63.5
96	egon	3	25.5

Odgovor:

ocena	minstar
3	25.5
7	35.0
8	25.5

Poišči starost najmlajšega mornarja, ki je star več kot 18, v skupinah, ki pripadajo ocenam in vsebujejo vsaj dva takšna mornarja.

ocena	star
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

ocena	star
1	33.0
3	25.5
3	63.5
3	25.5
7	45.0
7	35.0
8	55.5
8	25.5
9	35.0
10	35.0

ocena	minstar
3	25.5
7	35.0
8	25.5

Poišči starost najmlajšega mornarja, ki je star več kot 18, v skupinah, ki pripadajo ocenenam, vsebujejo vsaj dva takšna mornarja in so vsi pod 60.

HAVING COUNT (*) > 1 AND EVERY (M.star <=60)

ocena	star
٧	۴۵.۰
١	٣٣.٠
٨	۵۵.۵
٨	٢۵.۵
١٠	٣۵.٠
٧	٣۵.٠
١٠	١۶.٠
٩	٣۵.٠
٣	٢۵.۵
٣	۶٣.۵
٣	٢۵.۵

ocena	star
١	٣٣.٠
٣	٢۵.۵
٣	۶٣.۵
٣	٢۵.۵
٧	۴۵.۰
٧	٣۵.٠
٨	۵۵.۵
٨	٢۵.۵
٩	٣۵.٠
١٠	٣۵.٠

ocena	minstar
٧	٣۵.٠
٨	٢۵.۵

Kaj je rezultat spremembe EVERY v ANY?

Poišči starost najmlajšega mornarja, ki je star več kot 18, v skupinah, ki pripadajo ocenenam in vsebujejo vsaj dva mornarja stara od 18 do 60.

```
SELECT  M.ocena,  MIN (M.star)
          AS minstar
FROM  Mornarji M
WHERE  M.star >= 18
   AND M.star <= 60
GROUP BY  M.ocena
HAVING  COUNT (*) > 1
```

Mornarji

mid	mime	ocena	star
22	novak	7	45.0
29	pevec	1	33.0
31	kranjc	8	55.5
32	andrej	8	25.5
58	petelin	10	35.0
64	tom	7	35.0
71	kos	10	16.0
74	tom	9	35.0
85	miha	3	25.5
95	janez	3	63.5
96	egon	3	25.5

Odgovor

ocena	minstar
3	25.5
7	35.0
8	25.5

Za vsako rdečo ladjo poišči število rezervacij.

```
SELECT  L.lid,  COUNT (*) AS mcnt
FROM  Mornarji M, Ladja L, Rezervacije R
WHERE  M.mid=R.mid AND R.lid=L.lid AND L.barva='rdeca'
GROUP BY  L.lid
```

- Grupiranje po stiku treh relacij.
- Kaj dobimo, če umaknemo *L.barva='rdeca'* ?
- Kaj dobimo, če damo ta pogoj v stavek HAVING ?
- Kaj se zgodi, če umaknemo relacijo *Mornarji* in stik z *mid* ?

Poišči vse ocene za katere je povprečna starost minimalna po vseh ocenah

❖ Agregacijske operacije se ne morejo gnezditi!

```
SELECT  M.ocena
FROM  Mornarji M
WHERE  M.star =
        (SELECT  MIN (AVG (M2.star))  FROM Mornarji M2)
```

❖Korektna rešitev z SQL/92):

```
SELECT  Temp.ocena, Temp.povprecje
FROM  (SELECT  M.ocena, AVG (M.star) AS povprecje
        FROM  Mornarji M
        GROUP BY  M.ocena) AS Temp
WHERE  Temp.povprecje = (SELECT  MIN (Temp.povprecje)
                         FROM  Temp)
```


Null vrednosti

- Vrednosti polj so včasih *neznane* (npr., ocena ni bila vnešena).
  - SQL nudi posebno vrednost *null* za takšne situacije.
- Prisotnost vrednosti *null* vnaša v jezik več posledic:
  - Posebne operacije so potrebne za preverjanje če je dana vrednost enaka *null*.
  - Je *ocena>8* true ali false, ko je ocena enaka *NULL*?
    Kaj se zgodi z AND, OR in NOT operacijami?
  - Logika s tremi vrednostmi (true, false in *unknown*).
  - Pomen gradnikov mora biti definiran zelo pazljivo. (npr., WHERE ne izloči n-terice za katere je pogoj true.)
  - Nove operacije (*zunanji stiki*) mogoče/potrebne.

Integritetne omejitve (Pregled)

- IC opisuje pogoje, ki jih mora izpolniti vsaka legalna instanca relacije.
  - insert/delete/update, ki kršijo IC niso dovoljeni.
  - Lahko se uporabljajo zato, da zagotavljajo pravilen pomen podatov (npr., *mid* je ključ), ali onemogočijo nekonsistentnost (npr., *mime* mora biti niz in mora biti < 200)
- *Tipi IC*: Omejitve domen, primarni ključ, tuj ključ, splošne omejitve.
  - *Omejitve domen*: Vrednosti polj morajo biti pravilnega tipa.

Splošne omejitve

```
CREATE TABLE   Mornarji
    ( mid  INTEGER,
    mime  CHAR(10),
    ocena  INTEGER,
    star  REAL,
    PRIMARY KEY  (mid),
    CHECK  ( ocena >= 1
        AND ocena <= 10 )
```

- Uporabne, ko so potrebne bolj splošne omejitve kot ključi.
- Poizvedbe uporabimo za izražanje omejitev.
- Omejitve lahko imenujemo.

```
CREATE TABLE  Rezervacije
    ( mid  INTEGER,
    lid  INTEGER,
    day  DATE,
    PRIMARY KEY  (mid,bid,day),
    CONSTRAINT  BrezJezer
    CHECK  (`Jezero' <>
            ( SELECT  L.lime
            FROM  Ladje L
            WHERE  L.lid=lid)))
```


Omejitve preko večih relacij

- Če je relacija *Mornarji* prazna potem je lahko v relaciji *Ladje* karkoli.
- ASSERTION je prava rešitev; ni povezana z nobeno tabelo.

```
CREATE TABLE  Mornarji M
    ( mid  INTEGER,
    mime  CHAR(10),
    ocena  INTEGER,
    star  REAL,
    PRIMARY KEY  (mid),
    CHECK
    ( (SELECT COUNT (M.mid) FROM Mornarji M)
    + (SELECT COUNT (L.lid) FROM Ladje L) < 100 )
```

Število ladij skupaj s številom mornarjev je < 100

```
CREATE ASSERTION  MaliKlub
CHECK
( (SELECT COUNT (M.mid) FROM Mornarji M)
+ (SELECT COUNT (L.lid) FROM Ladje L) < 100 )
```


Prožilci

- Prožilec (trigger): procedura, ki se štarta v primeru, da se zgodi specifična sprememba v podatkovni bazi.
- Trije deli:
  - *Dogodek* - ki aktivira prožilec.
  - *Pogoj* - pove ali naj se procedura sproži.
  - *Akcija* - pove kaj naj naredi prožilec.

Prožilec: Primer (SQL:1999)

```
CREATE TRIGGER ShraniMladeMornarje
   AFTER INSERT ON MORNARJI
REFERENCING NEW TABLE NoviMornarji
FOR EACH STATEMENT
   INSERT
    INTO MladiMornarji(mid, mime, star, ocena)
    SELECT mid, mime, star, ocena
    FROM NoviMornarji N
    WHERE N.star <= 18
```

Pregled

- SQL je bil pomemben pri sprejetju relacijskega podatkovnega modela kot osnovo večine realnih sistemov; jezik je bolj naraven kot starejši proceduralni jeziki.
- Veliko načinov za zapis istega vprašanja; optimizator bi moral poiskati najbolj učinkovit plan evaluacije.
  - V realnosti morajo uporabniki zadosti dobro poznati sistem in podatke, da v robnih primerih zagotovijo optimalno evaluacijo.

Pregled (2)

- SQL je relacijsko kompleten; precej močnejši jezik kot so relacijska algebra.
- NULL se uporablja za neznane vrednosti polj; komplikacije z NULL vrednostmi.
- SQL omogoča specifikacijo bogate množice integritetnih omejitev.
- Prožilci se odzivajo na spremembe v bazi.

1. SQL - LETALSKI POTNIŠKI PROMET

Vzemimo naslednjo shemo:

```
LETALISCA
  _IdLE: integer,
   ime: varchar(30),
   kraj: varchar(50),
   drzava: varchar(50)
LETALA
  _IdTL: integer,
   tip: varchar(20),
   opis: varchar(250),
   LetoProizv: integer,
   kapaciteta: integer
PRISTANE
  _IdLE: integer,
  _IdTL: integer,
   DatumOd: date
LET:
  _IdLet: integer,
   IdTL: integer,
   IdLE_vzleta: integer,
   IdLE_pristanka: integer,
   cas_vzleta: date,
   cas_pristanka: date,
   id_pilot
REZERVACIJE
  _IdR: integer,
   IdLet: integer,
   Cena: Double,
   Razred: integer,
   Ime: varchar(50),
   Priimek: varchar(50)
PILOT
  _id_pilot: integer,
   Ime: varchar(50),
   Priimek: varchar(50),
   DelovnaDoba: integer,
   StLetov: integer
```

Atributi, ki predstavljajo ključ so podčrtani. Relacija PRISTANE vsebuje podatke o tem, kateri tipi letal lahko pristanejo na posameznih letališčih in od kdaj. Relacija LET vsebuje podatke o letih posameznih letal. Relacija REZERVACIJE vsebuje podatke o rezervacijah za posamezne lete.

Zapišite naslednje poizvedbe v SQL poizvedovalnem jeziku.

a) Izpiši imena vseh letališč v ZDA.
b) Izpiši opis in IdTL vseh letal, ki so starejša od leta 2000.
c) Poišči vsa imena letališč, na katerih lahko pristane letalo tipa B747.
d) Poišči vse IdLE letališč, na katerih lahko pristane letalo tipa B747 ali B748.
e) Imena vseh letališč, na katerih je pristalo letalo tipa B747 ali B748.
f) Poišči vse IdLE letališč, ki so ali v Avstriji ali pa na njih lahko pristane letalo tipa B747.
g) Poišči vse IdLE letališč, na katerih lahko pristane letalo tipa B747 in tipa B748.
h) Zapišite SQL stavek za brisanje vseh zapisov v tabeli LETALA, za katere velja, da je kapaciteta letala manjša od 80.
i) Zapišite SQL stavek za spreminjanje podatka o ceni rezervacije v tabeli REZERVACIJE z novo vrednostjo, ki je za 30% višja od stare. Pri tem upoštevajte, da se spremenijo le cene tistih rezervacij, kjer je razred rezervacije 1.
j) Izpis Imena in Priimka pilotov v enem stolpcu.
k) Izpis začetnic vseh pilotov (predpostavljamo, da ime vsak le en priimek in le eno ime).
l) Izpis deleža števila letov pilota glede na 1 leta delovne dobe za vse pilote.
m) Poišči vse IdLE letališč, na katerih lahko pristanejo vsi tipi letal.

2. SQL – NAFTNA DRUŽBA

Vzemimo naslednjo shemo:

```
SERVIS
  _IdS: integer,
   Ime: varchar(50),
   Kraj: varchar(50)
ZALOGA
  _IdTG: integer,
  _IdS: integer,
   kolicina: integer
TIP_GORIVA
  _IdTG: integer,
   ime: varchar(50)
CISTERNE
  _IdC: integer,
   SerijskaSt: integer,
   kapaciteta: integer,
   LetoIzd: integer,
   DtServisa: date
NAROCILA
  _IdN: integer,
   IdS: integer,
   St:integer,
   Datum: date,
   Kolicina: integer
DOBAVA
  _IdN: integer,
  _IdC: integer,
   Datum: date,
   Kolicina: integer
   IdV: integer
VOZNIK
  _IdV: integer,
   Ime: varchar(50),
   Priimek: varchar(50)
```

Atributi, ki predstavljajo ključ so podčrtani.
Relacija DOBAVA vsebuje podatke o opravljenih dobavah oz. prevozih cistern za izpolnitev posameznih naročil. Za izpolnitev enega naročila je lahko potrebno tudi več dobav oz. prevozov. Dobavo izvede cisterna z voznikom. Servis hrani zalogo posameznega tipa goriv v tabeli ZALOGA.

Zapišite naslednje poizvedbe v SQL poizvedovalnem jeziku.

a) Imena vseh bencinskih servisov v Kopru.
b) IdC vseh cistern, ki so starejše od leta 2000.
c) Poišči vse številke naročil, za katere je pri dobavi goriva sodelovala cisterna s serijsko številko 123456.
d) Poišči vse IdN naročil, ki jih je izpolnila cisterna s serijsko številko 123456 ali 123457.
e) Poišči vse IdN naročil, ki ali zahtevajo količino 10.000 enot ali pa jih je izpolnila cisterna s serijsko številko 123456.
f) Poišči vse IdN naročil, ki jih je izpolnila cisterna s serijsko številko 123456 in cisterna s serijsko številko 123457.
g) Izbriši vse zapise v tabeli CISTERNE, za katere velja, da je kapaciteta cisterne manjša od 5000.
h) Zapišite SQL stavek za spreminjanje podatka o datumu zadnjega servisa cisterne v tabeli CISTERNE z novo vrednostjo 1.1.2005, za vse tiste cisterne, kjer je datum starejši od 1.1.2000.

i) Zapišite SQL stavke za vnos zapisa v eno izmed tabel v podatkovni bazi naftne družbe.
j) Izpis Imena in Priimka šoferjev v enem stolpcu.
k) Izpis začetnic vseh šoferjev (predpostavljamo, da ime vsak le en priimek in le eno ime).
l) Izpis zaloge po posameznih črpalkah tako, da je količina izražena v 1000 litrih (npr. v tabeli podatek 500 l, izpišemo 0,5 l).
m) Številke vseh naročil, katera so bila izpolnjena z dobavo cisterne s kapaciteto 16.000 ali 24.000.

1. LETALSKI PROMET

LETALISCA
_IdLE: integer,
ime: varchar(30),
kraj: varchar(50),
drzava: varchar(50)
LETALA
_IdTL: integer,
tip: varchar(20),
opis: varchar(250),
LetoProizv: integer,
kapaciteta: integer
IdLD: integer
PRISTANE
_IdLE: integer,
_IdTL: integer,
DatumOd: date
LET:
_IdLet: integer,
IdTL: integer,
IdLE_vzleta: integer,
IdLE_pristanka: integer,
cas_vzleta: date,
cas_pristanka: date,
id_pilot: integer
REZERVACIJE
_IdR: integer,
IdLet: integer,
Cena: Double,
Razred: integer,
Ime: varchar(50),
Priimek: varchar(50)
PILOT
_id_pilot: integer,
IdLD: integer,
Ime: varchar(50),
Priimek: varchar(50),
DelovnaDoba: integer,
StLetov: integer
STEVARDESA
_IdS: integer,
IdLD: integer
ime: varchar(50),
Priimek: varchar(50),
StLetov: integer
LETALSKA_DRUZBA
_IdLD:integer
ime
naslov

Navedena poizvedbena vprašanja predstavite z ustreznim SQL stavkom in pri tem uporabite zahtevane SQL ukaze:

a) SQL ukaz: DISTINCT
Poizvdbe:
- Vsi priimki pilotov

b) SQL ukaz: Uporabite MAX kjer je to potrebo
Poizvedbe:
- Imena in priimki vseh pilotov, ki so že pristali na enem izmed letališč v ZDA (USA).
- Številke letal, ki imajo najvišjo kapaciteto.
- Številke letal, ki imajo drugo najvišjo kapaciteto.

c) SQL ukaz: UNION ali INTERSECT ali EXCEPT
Poizvedbe:
- Imena vseh letališč, ki so v ZDA ali pa je na njih pristalo letalo tipa B747.
- Imena vseh letališč, ki so v ZDA in je na njih pristalo letalo tipa B747.
- IdP vseh pilotov, ki so pilotirali natanko 3 lete.

d) SQL ukaz: IN ali NOT IN
Poizvedbe:
- Imena letalskih družb, ki imajo v lasti letala s kapaciteto večjo od 300.
- Imena vseh letališč, na katerih so pristala letala s kapaciteto večjo od 300.

e) SQL ukaz: EXISTS ali ALL
Poizvedbe:
- Imena vseh letališč, na katerih so pristala letala s kapaciteto večjo od 300.
- Imena in priimki pilotov, ki so že pilotirali letala s kapaciteto večjo od 300.
- Imena in priimki pilotov, pri katerih je število ur letenja višje od števila ur letenja vsakega pilota z imenom William.

f) SQL ukaz: COUNT ali AVG ali SUM ali MAX ali MIN in druge agregacijske funkcije
Poizvedbe:
- Število opravljenih letov za posamezne stevardese letalske družbe Adria Airways.
- Imena in priimki stevardes, ki so opravile več kot 60 letov pri letalski družbi Adria Airways.
- Imena in priimki pilotov, ki so pilotirali samo letala s kapaciteto večjo od 300.

g) SQL ukaz: LEFT JOIN
Poizvedbe:
- Za letalsko družbo Adria Airways poiščite seznam vseh tipov letal, ki poleg tipa letala vsebuje še število tovrstnih letal (rezultat naj obsega vse tipe letal, tudi če takih letalska družba nima v lasti).

2. NAFTNA DRUŽBA
Navedena poizvedbena vprašanja predstavite z ustreznim SQL stavkom in pri tem uporabite zahtevane SQL ukaze:

SERVIS
_IdS: integer,
Ime: varchar(50),
Kraj: varchar(50)
ZALOGA
_IdTG: integer,
_IdS: integer,
kolicina: integer
TIP_GORIVA
_IdTG: integer,
ime: varchar(50)
TOCILNO_MESTO
_IdTM: integer,
IdTG: integer,
IdS: integer
CISTERNE
_IdC: integer,
SerijskaSt: integer,
kapaciteta: integer,
LetoIzd: integer,
DtServisa: date
NAROCILA
_IdN: integer,
IdS: integer,
St:integer,
Datum: date,

```
    Kolicina: integer
  DOBAVA
    _IdD: integer,
     IdN: integer,
     IdC: integer,
     Datum: date,
     Kolicina: integer
     IdV: integer
  VOZNIK
    _IdV: integer,
     Ime: varchar(50),
     Priimek: varchar(50),
     StKazni: integer
  ZAPOSLENIBC
    _IdZ: integer
     Ime: varchar(50),
     Priimek: varchar(50)
     IdS: integer
```

a) SQL ukaz: DISTINCT
   Poizvdbe:
   - Vsi priimki šoferjev.
   - Datumi vseh dobav (brez ponavljanj).

b) SQL ukaz: Uporabite MAX kjer je to potrebo
   Poizvedbe:
   - Imena in priimki vseh zaposlenih na kateremkoli bencinskem servisu v Kopru.
   - Imena in priimki šoferjev, ki imajo najvišje število kazni.
   - Imena in priimka šoferjev, ki imajo drugo najvišje število kazni.

c) SQL ukaz: UNION ali INTERSECT ali EXCEPT
   Poizvedbe:
   - Številke vseh naročil, ki zahtevajo količino višjo od 10.000 l ali pa so
     bila izpolnjena z dobavo cisterne, katere leto izdelave je enako 2000.
   - Številke vseh naročil, ki zahtevajo količino višjo od 10.000 l in so bila
     izpolnjena z dobavo cisterne, katere leto izdelave je enako 2000.
   - Imena in priimki vseh šoferjev, ki so sodelovali natanko v 3 dobavah.

d) SQL ukaz: IN ali NOT IN
   Poizvedbe:
   - Imena bencinskih servisov, ki imajo več kot 10.000 l zaloge dizelskega goriva.
   - Številke vseh naročil, za katere je bila opravljena dobava s cisterno, katere
     kapaciteta je bila večjo od 20.000.

e) SQL ukaz: EXISTS ali ALL
   Poizvedbe:
   - Številke vseh naročil, za katere je bila opravljena dobava s cisterno, katere
     kapaciteta je bila večjo od 20.000.
   - Imena in priimki šoferjev, ki so že vozili cisterne s kapaciteto večjo od 20.000.
   - Imena in priimki šoferjev, pri katerih je število kazni višje od števila
     kazni vsakega šoferja z imenom Kevin.

f) SQL ukaz: COUNT ali AVG ali SUM ali MAX ali MIN in druge agregacijske funkcije
   Poizvedbe:
   - Število točilnih mest po posameznih bencinskih servisih.
   - Imena in priimki šoferjev, ki so opravili več kot 500 dobav.
   - Imena in priimki soferjev, ki so sodelovali samo pri tistih dobavah, kjer so
     bile uporabljene cisterne s kapaciteto večjo od 20000.

Relacijski račun

Iztok Savnik, FAMNIT

Relacijski račun

- 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*.

Domenski relacijski račun

- *Vprašanje ima obliko*:

$$\{\langle x1,x2,...,xn\rangle \mid p(\langle x1,x2,...,xn\rangle)\}$$

❖ Odgovor vsebuje n-terice $\langle x1,x2,...,xn\rangle$ za katere vrne izraz $p(\langle x1,x2,...,xn\rangle)$ vrednost *true*.

❖ *Izraz* je rekurzivno definiran na osnovi enostavnih atomičnih izrazov (referenciranje n-teric v relacijah; primerjanje atributov), ki se lahko gradijo v bolj kompleksne izraze z *logičnimi operacijami*.

DRC Formule

- *Atomična formula:*
  - $\langle x1,x2,...,xn\rangle \in Rname$, ali X *op* Y, ali X *op* const
  - *op* je lahko $<,>,=,\leq,\geq,\neq$
- *Formula:*
  - *atomična formula*, ali
  - $\neg p, p \wedge q, p \vee q$, kjer so p in q formule, ali
  - $\exists X\,(p(X))$, kjer je sprem. X *prosta* v p(X), ali
  - $\forall X\,(p(X))$, kjer je sprem. X *prosta* v p(X)
- Uporaba kvantifikatorjev $\exists X$ in $\forall X$ *poveže* sprem. X.
  - Spremenljivka, ki ni povezana je prosta.

Proste in vezane spremenljivke

- Uporaba kvantifikatorjev $\exists X$ in $\forall X$ v formuli izvrši povezovanje X.
  - Spremenljivka, ki ni vezana je prosta.
- Poglejmo spet definicijo izraza (vprašanja):

$$\{\langle x1, x2, ..., xn\rangle \mid p(\langle x1, x2, ..., xn\rangle)\}$$

❖ Pomembna omejitev: spremenljivke x1, ..., xn ki se nahajajo na levo od `|' so edine spremenljivke, ki so lahko proste v formuli p(...).

Poišči vse mornarje, ki imajo oceno več kot 7

$$\{<I,N,T,A> \mid <I,N,T,A> \in Mornarji \wedge T > 7\}$$

- Pogoj $<I,N,T,A> \in Mornarji$ zagotavlja da so domenske spremenljivke *I, N, T* in *A* povezane z domenami atributov n-teric relacije *Mornarji*.
- Izraz $\langle I,N,T,A\rangle$ na levi strani `|' (beremo: "*tako da*") pravi, da je vsaka n-terica $\langle I,N,T,A\rangle$, ki zadošča T>7 v odgovoru.
- Spremeni izraz za odgovor na:
  - Poišči vse mornarje, ki so starejši od 18, imajo oceno pod 9, in jim je ime 'Janez'.

Poišči mornarje z oceno > 7, ki so rezervirali ladjo #103

$$\{<I,N,T,A> \mid <I,N,T,A> \in Mornarji \wedge T > 7 \wedge \exists Ir,Br,D(<Ir,Br,D> \in Rezervacije \wedge Ir = I \wedge Br = 103)\}$$

- Uporabili smo $\exists Ir, Br, D\ (\ldots)$ kot okrajšavo za $\exists Ir(\exists Br(\exists D(\ldots)))$
- Kvantifikator $\exists$ je bil uporabljen za povezovanje (Stik) n-teric iz relacije *Rezervacije* z n-tericami iz relacije *Mornarji*.

Poišči vse mornarje z oceno > 7, ki so rezervirali rdečo ladjo.

$$\{<I,N,T,A> \mid <I,N,T,A> \in Mornarji \wedge T > 7 \wedge \exists Ir,Br,D(<Ir,Br,D> \in Rezervacije \wedge Ir = I \wedge \exists B,BN,C(<B,BN,C> \in Ladje \wedge B = Br \wedge C = rdeca))\}$$

- Oklepaji kontrolirajo področje povezovanja kvantifikatorja .
- Izrazi delujejo kompleksno vendar uporabniški vmesnik (MS Access, QBE) na osnovi DRC je intuitiven.

Poišči mornarje, ki so rezervirali vse ladje.

$$\{<I,N,T,A> \mid <I,N,T,A> \in Mornarji \wedge$$
$$\forall B,BN,C\left(\neg\left(<B,BN,C> \in Ladje\right) \vee\right.$$
$$\left.\left(\exists Ir,Br,D\left(<Ir,Br,D> \in \mathrm{Rezervacije} \wedge I = Ir \wedge Br = B\right)\right)\right)\}$$

- Poišči vse mornarje *I* tako, da za vsako 3-terico $\langle B,BN,C \rangle$ velja, da bodisi ni v relaciji *Ladje* ali pa obstaja n-terica v relaciji *Rezervacije*, ki pokaže, da je dani mornar *I* rezerviral to ladjo.

Poišči mornarje, ki so rezervirali vse ladje (spet!)

$$\{<I,N,T,A> \mid <I,N,T,A> \in Mornarji \wedge$$
$$\forall <B,BN,C> \in Ladje$$
$$\left(\exists <Ir,Br,D> \in \mathrm{Rezervacije}\left(I = Ir \wedge Br = B\right)\right)\}$$

- Enostavna notacija, isto vprašanje. (Bolj čisto!)
- ... mornarje, ki so rezervirali vse rdeče ladje:

$$\{C \neq rdeca \vee \exists <Ir,Br,D> \in \mathrm{Rezervacije}\left(I = Ir \wedge Br = B\right)\}$$

.....

Varni izrazi & Izrazna moč

- Mogoče je zapisati korekten izraz relacijskega računa, ki ima neskončno število odgovorov. Pravimo, da takšni izrazi niso varni.
  - Primer: $\{S \mid \neg\left(S \in Mornarji\right)\}$
- Znano je, da lahko vsak izraz relacijske algebre prevedemo v varen izraz relacijskega računa (DRC/TRC); obratno je tudi res.
- *Relacijska kompletnost*: Povpraševalni jezik (npr. SQL) lahko izrazi vsako vprašanje,ki ga lahko izrazimo z relacijsko algebro ali računom.

Povzetek

- Relacijski račun je deklarativen (ne-proceduralen); uporabniki definirajo vprašanja tako, da zapišejo kaj želijo in kako naj sistem poišče rezultat.
- Algebra in varni izrazi relacijskega računa imajo enako izrazno moč; dobimo razred jezikov “relacijska kompletnost”.

Relacijski račun

1. Relacijski račun - LETALSKI POTNIŠKI PROMET
Vzemimo naslednjo shemo:

```
LETALISCA
  _IdLE: integer,
   ime: varchar(30),
   kraj: varchar(50),
   drzava: varchar(50)
TIP_LETALA
  _IdTL: integer,
   tip: varchar(20),
   opis: varchar(250),
   OznakaProizv: varchar(20)
PRISTANE
  _IdLE: integer,
  _IdTL: integer,
   DatumOd: date
```

Atributi, ki predstavljajo ključ so podčrtani. Relacija PRISTANE vsebuje podatke o tem, kateri tipi letal lahko pristanejo na posameznih letališčih in od kdaj.

Zapišite naslednje poizvedbe v relacijskem računu (DRC).

a) Poišči vsa imena letališč, na katerih lahko pristane letalo tipa B747.
b) Poišči vse IdLE letališč, na katerih lahko pristane letalo tipa B747 ali B748.
c) Poišči vse IdLE letališč, ki so ali v Avstriji ali pa na njih lahko pristane letalo tipa B747.
d) Poišči vse IdLE letališč, na katerih lahko pristane letalo tipa B747 in tipa B748.
e) Poišči vse IdLE letališč, na katerih lahko pristanejo vsi tipi letal.
f) Poišči vse IdLE letališč v Avstriji, ki na njih lahko pristane letalo tipa B747.

2. Relacijski račun – NAFTNA DRUŽBA
Vzemimo naslednjo shemo:

```
CISTERNE
   _IdC: integer,
    SerijskaSt: integer,
    kapaciteta: integer,
    DtServisa: date
NAROCILA
   _IdN: integer,
    St:integer,
    Datum: date,
    Kolicina: integer
DOBAVA
   _IdN: integer,
   _IdC: integer,
    Datum: date,
    Kolicina: integer
```

Atributi, ki predstavljajo ključ so podčrtani.
Relacija DOBAVA vsebuje podatke o opravljenih dobavah oz. prevozih cistern za izpolnitev posameznih naročil. Za izpolnitev enega naročila je lahko potrebno tudi več dobav oz. prevozov.

Zapišite naslednje poizvedbe v relacijskem računu (DRC).

a) Poišči vse številke naročil, za katere je pri dobavi goriva sodelovala cisterna s serijsko številko 123456.
b) Poišči vse IdN naročil, ki jih je izpolnila cisterna s serijsko številko 123456 ali 123457.
c) Poišči vse IdN naročil, ki ali zahtevajo količino 10.000 enot ali pa jih je izpolnila cisterna s serijsko številko 123456.
d) Poišči vse IdN naročil, ki jih je izpolnila cisterna s serijsko številko 123456 in cisterna s serijsko številko 123457.

3. Relacijski račun – SPLETNI FORUM
Vzemimo naslednjo shemo:

```
UPORABNIK
   _idu
    uporabniško_ime
    geslo
    enaslov
    vloga
UPORABNIK_VLOGA
   _idu
   _idv
VLOGA
   _idv
    naziv     (vzdrževalec,navaden uporabnik, ….)
FORUM
   _idf
    ime_forum
    idu       (vzdrževalec foruma)
OBJAVA
   _ido
    idf       (h kateremu forumu spada objava)
    idu       (uporabnik objave)
    datum
    naslov
    vsebina
    ido_o     (parent objava)
```

Atributi, ki predstavljajo ključ so podčrtani.

Zapišite naslednje poizvedbe v relacijskem računu (DRC).

a) Najdi vse uporabnike, ki niso administratorji
b) Poišči vse uporabnike, ki vnesli objavo v forum s številko 1
c) Izpiši vsa imena forumov pri katerih je vzdrževalec oseba z imenom Janez
d) Izpiši objave iz foruma z imenom "Splošno", ki so bile objavljene pred 1. 12. 2010

Query-by-Example (QBE)

Iztok Savnik, FAMNIT

QBE: Uvod

- "GUI" za poizvedovanje
  - Osnovan na domenskem relacijskem računu!
  - Narejen pred GUI.
  - Zelo primeren za enostavna vprašanja
  - Neroden za kompleksna vprašanja
- QBE - IBM trademark.
  - Vplival na veliko projektov
  - PC DB: Paradox, Access, …

Primeri tabel za QBE

- Uporabniki definirajo vprašanja s *primeri tabel*, ali *skeleti*.

Reserves	sid	bid	day

Boats	bid	bname	color

Sailors	sid	sname	rating	age

Osnove

- Izpiši vsa imena mornarjev

Sailors	sid	sname	rating	age
		P._N		P._A

❖ Izpiši vse mornarje z *rating* > 8 v naraščajočem vrstnem redu po (*rating, age*):

Sailors	sid	sname	rating	age
P.			AO(1). >8	AO(2).

❖ QBE vstavi unikatne nove spremenljivke v prazne stolpce. Isti stavek v DRR (brez urejenosti):

$$\{\langle I, N, T, A\rangle \mid \langle I, N, T, A\rangle \in Sailors \ \wedge \ T > 8\}$$

And/Or vprašanja

- Imena mornarjev, ki so mlajši od 30 ali starejši od 20:

Sailors	sid	sname	rating	age
		P.		< 30
		P.		> 20

❖ Imena mornarjev, ki so mlajši od 30 in starejši od 20:

Sailors	sid	sname	rating	age
	_Id	P.		< 30
	_Id	P.		> 20

❖ Imena mornarjev, ki so mlajši od 30 in imajo *rating* > 4:

Sailors	sid	sname	rating	age
	_Id	P.	> 4	< 30

Vprašanja s stikom

- Imena mornarjev, ki so rezervirali ladjo za dan 8/24/96 in so starejši od 25 (datumi in nizi imajo narekovaje):

Sailors	sid	sname	rating	age
	_Id	P._S		> ٢٥

Reserves	sid	bid	day
	_Id		'٨/٢٤/٩٦'

❖ Stik je definiran z ujemajočima spremeljivkama.

Duplikati

- *Ena vrstica vsebuje* P: duplikati niso izločeni privzeto; izločenje dosežemo z UNQ.

Sailors	sid	sname	rating	age
UNQ.		P.		< 30

❖ *Več vrstic z* P: duplikati so izločeni privzeto! lahko preprečimo izločenje duplikatov z ALL.

Sailors	sid	sname	rating	age
ALL.	_Id	P.		< ٣٠
	_Id	P.		> ٢٠

Stiki

- Barve ladij z imenom "Interlake", ki so jih rezervirali mornarji za dan 8/24/96 in so starejši od 25 :

Sailors	sid	sname	rating	age
	_Id			> 25

Reserves	sid	bid	day
	_Id	_B	'8/24/96'

Boats	bid	bname	color
	_B	'Interlake'	P.

Stik

- Imena in starost mornarjev, ki so rezervirali ladjo, ki jo je rezerviral mornar z si*d* = 22:

Sailors	sid	sname	rating	age
	_Id	P.		P.

Reserves	sid	bid	day
	22	_B	
	_Id	_B	

Neimenovani stolpci

- Uporabno, če želimo izpisati rezultate nekega izraza ali natiskati polja iz dveh relacij.
  - Nekateri QBE dovoljujejo P. v samo eni tabeli !

Sailors	sid	sname	rating	age		
	_Id	P.	_R	_A	P._D	P.(_R/_A)

Reserves	sid	bid	day
	_Id		_D

"Negativne tabele"

- Negacijo damo pod imenom relacije:

Sailors	sid	sname	rating	age
	_Id	P._S		

Reserves	sid	bid	day
¬	_Id	_B	

Agregacija

- QBE podpira AVG, COUNT, MIN, MAX, SUM
  - Nobena operacija ne eliminira duplikatov, razen COUNT
  - Imamo tudi AVG.UNQ. ...

Sailors	sid	sname	rating	age	
		G.	G.P.AO	_A	P.AVG._A

❖Stolpci z G. so *group-by* polja; vse n-terice iz skupine imajo isto vrednost izbranih atributov.
—Opcijska uporaba .AO uredi odgovore.
—Vsak stolpec z P. mora imeti G. ali agregacijsko operacijo.

Ločeni pogoji

- Ločene pogoje uporabimo za izražanje zvez med več relacijami oz. stolpci različnih relacij.
- Lahko izražamo pogoje za skupino n-teric podobno stavku HAVING v SQL:

Sailors	sid	sname	rating	age
			G.P.	_A

CONDITIONS
AVG._A > ۳۰

❖Izražanje pogojev z AND in OR:

Sailors	sid	sname	rating	age
		P.		_A

CONDITIONS
20 < _A AND _A < 30

Poišči mornarje, ki so rezervirali vse ladje

- Vprašanje z deljenjem
- Potrebovali bomo agregacijske operacije (in update).

Sailors	sid	sname	rating	age
	P.G._Id			

Reserves	sid	bid	day
	_Id	_B1	

CONDITIONS
COUNT._B1= COUNT._B2

Boats	bid	bname	color
	_B2		

❖Kako lahko spremenimo vprašanje, da izpiše imena mornarjev, ki so rezervirali vse ladje?

Vstavljanje zapisov

- Vstavljanje enega zapisa:

Sailors	sid	sname	rating	age
I.	74	Janice	7	14

❖Vstavljanje več zapisov (*rating* je *null* v spodnjih zapisih):

Sailors	sid	sname	rating	age
I.	_Id	_N		_A

CONDITIONS
_A > 18 OR
_N LIKE 'C%'

Students	sid	name	login	age
	_Id	_N		_A

Brisanje in update

- Pobriši vse rezervacije za mornarje, ki imajo *rating* < 4

Sailors	sid	sname	rating	age
	_Id		< 4	

Reserves	sid	bid	day
D.	_Id		

❖Povečaj za eno starost mornarja, ki ima *sid* = 74

Sailors	sid	sname	rating	age
	74			U._A+1

Omejitve pri popravljanju zapisov

- Ne moremo mešati I., D. in U. v enem primerku tabele
- I., D. in U. ne moremo kombinirati z P. in G.
- Ne moremo vstavljati, popravljati ali spreminjati n-teric z vrednostmi iz stolpcev iz ostalih n-teric iste tabele.

Sailors	sid	sname	rating	age
		john		_A
		joe		U._A+1

Naj popravimo starosti *vseh* Joe?
Starost *katerega* Johna naj uporabimo?

Uporaba pogledov

- Poišči mornarje, ki niso rezervirali neko ladjo _B:

Sailors	sid	sname	rating	age
	_Id	P._S		

BadSids	sid
I.	_Id

Boats	bid	bname	color
	_B		

Reserves	sid	bid	day
¬	_Id	_B	

❖Poišči mornarje, ki niso v neki množici:

Sailors	sid	sname	rating	age
	_Id	P._S		

BadSids	sid
¬	_Id

Poišči mornarje, ki so rezervirali vse ladje (Spet!)

- Hočemo poiskati mornarje _Id tako, da ne obstaja ladja _B, ki je nebi rezerviral _Id:

Sailors	sid	sname	rating	age
	_Id	P._S		

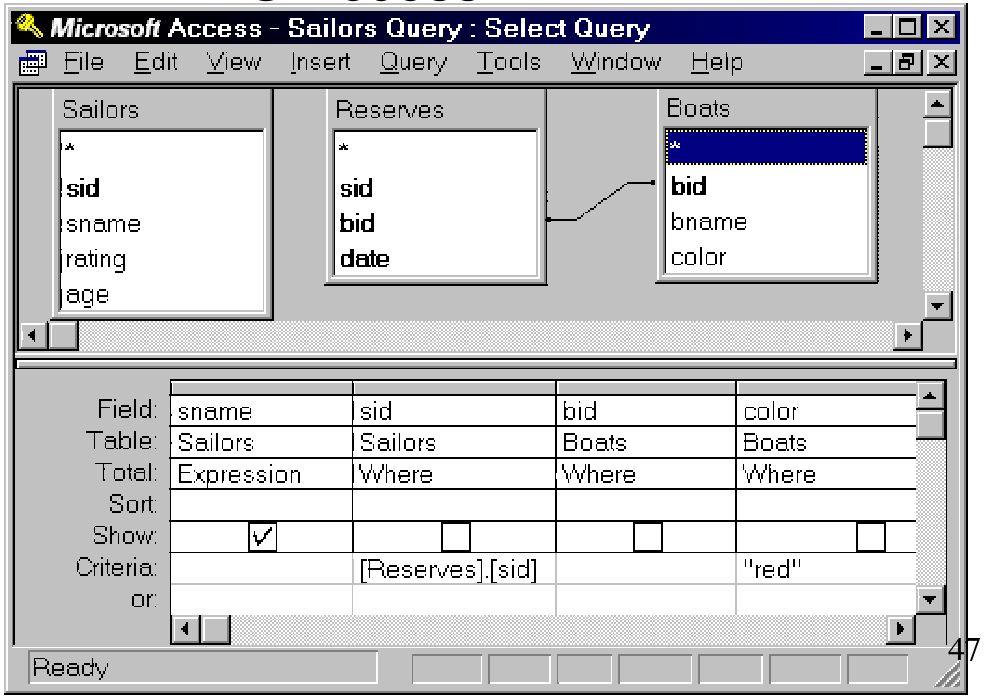
Boats	bid	bname	color
¬	_B		

Reserves	sid	bid	day
¬	_Id	_B	

❖Nelegalno vprašanje! Spremenljivka _B se ne pojavi v pozitivni vrstici.

❖V kakšnem vrstnem redu naj se obravnavajo negativne vrstice?

MS Access



Povzetek

- QBE je eleganten, prijazen do uporabnika
- Jezik osnovan na relacijskem računu
- Jezik je izrazen in relacijsko kompleten
- Enostavna vprašanja in enostavna sintaksa
- Vpliv na različna grafična orodja
  - Borland's Paradox
  - Microsoft's Access.

1. SQL in QBE na tablo in phpmyadmin

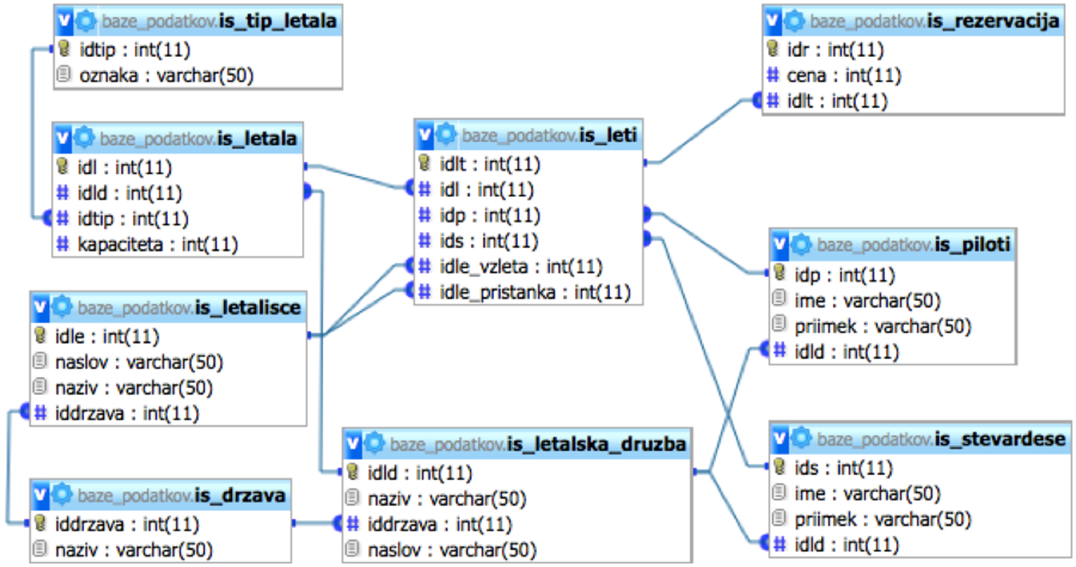
POGLEJ SLIKO ZA RELACIJSKI MODEL PODATKOVNE BAZE

2. Z uporabo QBEja na tabli in znotraj phpMyAdmin vmesnika, izvedi naslednje poizvedbe:

a) Izpiši vse vrstice pilotov, ki se jim priimek začne na B
b) Izpiši vse vrstice letališč, ki imajo predzadnjo črko imena (naziva) r in niso iz države Slovenija
c) Izpiši vsa letala (oznaka), ki imajo kapaciteto manjšo od 170
d) Izpiši imena in priimke vseh pilotov, ki so pristali v državi Italija
e) Najdi pilote ki so pilotirali vsa letala
f) Za vsakega pilota, izpiši ime, priimek in naziv letalske družne pri kateri je zaposlen
g) Izpiši letalske družbe, ki nimajo v lasti DC9

3. Zapiši naslednje poizvedbe v SQLu z uporabo ORDER BY, GROUP BY, HAVING, COUNT, JOIN, kjer je potrebno.

a) Izpiši imena in priimke ter število letov pilotov. Izpis naj bo urejen naraščajoče po priimkih
b) Za vsakega pilota (ime, priimek) izpiši vsa letala (opis) s katerimi so leteli, pri čemer za vsakega pilota izpiši vsak tip samo enkrat
c) Izpiši imena držav in števila letališč po državah padajoče (od Z proti A) brez duplikatov
d) Preštej število letal, ki imajo kapaciteto med 200 in 300
e) Izpiši ime države z največ letališč
f) Izpiši pilote (ime, priimek) in število letal, za vse pilote, ki so pilotirali več kot dva leta
g) Izpiši letala, ki so jih pilotirala dva pilota.
h) Za vsak let (idle vzleta, idle pristanka) izpiši stevardeso. Izpis leta naj se izpiše tudi če na njem ni bilo stevardese - uporabi LEFT JOIN



```
-- phpMyAdmin SQL Dump
-- version 3.2.2.1deb1
-- http://www.phpmyadmin.net
--
-- Host: localhost
-- Generation Time: Jan 28, 2010 at 02:19 PM
-- Server version: 5.1.37
-- PHP Version: 5.2.10-2ubuntu6

SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;

--
-- Database: `baze_podatkov`
--

-- --------------------------------------------------------

--
-- Table structure for table `is_drzava`
--

DROP TABLE IF EXISTS `is_drzava`;
CREATE TABLE IF NOT EXISTS `is_drzava` (
  `iddrzava` int(11) NOT NULL AUTO_INCREMENT,
  `naziv` varchar(50) NOT NULL,
  PRIMARY KEY (`iddrzava`)
) ENGINE=InnoDB  CHARACTER SET utf8 COLLATE utf8_bin AUTO_INCREMENT=3 ;

--
-- Dumping data for table `is_drzava`
--

INSERT INTO `is_drzava` (`iddrzava`, `naziv`) VALUES
(1, 'Slovenija'),
(2, 'Avstrija');

-- --------------------------------------------------------

--
-- Table structure for table `is_letalisce`
--

DROP TABLE IF EXISTS `is_letalisce`;
CREATE TABLE IF NOT EXISTS `is_letalisce` (
  `idle` int(11) NOT NULL AUTO_INCREMENT,
  `naslov` varchar(50) NOT NULL,
  `naziv` varchar(50) NOT NULL,
  `iddrzava` int(11) NOT NULL,
  PRIMARY KEY (`idle`),
  FOREIGN KEY (`iddrzava`) REFERENCES is_drzava(`iddrzava`) ON DELETE RESTRICT
) ENGINE=InnoDB  CHARACTER SET utf8 COLLATE utf8_bin AUTO_INCREMENT=3 ;

--
-- Dumping data for table `is_letalisce`
--

INSERT INTO `is_letalisce` (`idle`, `naslov`, `naziv`, `iddrzava`) VALUES
(1, 'Ljubljana', 'Ljubljana', 1),
(2, 'Dunaj', 'Dunaj', 2);

-- --------------------------------------------------------

--
-- Table structure for table `is_letalska_druzba`
--

DROP TABLE IF EXISTS `is_letalska_druzba`;
CREATE TABLE IF NOT EXISTS `is_letalska_druzba` (
  `idld` int(11) NOT NULL AUTO_INCREMENT,
  `naziv` varchar(50) NOT NULL,
  `iddrzava` int(11) NOT NULL,
  `naslov` varchar(50) NOT NULL,
  PRIMARY KEY (`idld`),
  FOREIGN KEY (`iddrzava`) REFERENCES is_drzava(`iddrzava`) ON DELETE RESTRICT
) ENGINE=InnoDB  CHARACTER SET utf8 COLLATE utf8_bin AUTO_INCREMENT=3 ;

--
-- Dumping data for table `is_letalska_druzba`
--

INSERT INTO `is_letalska_druzba` (`idld`, `naziv`, `iddrzava`, `naslov`) VALUES
(1, 'adria airways', 1, 'ljubljana'),
(2, 'lufthansa', 2, 'dunaj');

-- --------------------------------------------------------

--
-- Table structure for table `is_piloti`
--

DROP TABLE IF EXISTS `is_piloti`;
CREATE TABLE IF NOT EXISTS `is_piloti` (
  `idp` int(11) NOT NULL AUTO_INCREMENT,
  `ime` varchar(50) NOT NULL,
  `priimek` varchar(50) NOT NULL,
  `idld` int(11) NOT NULL,
  PRIMARY KEY (`idp`),
  FOREIGN KEY (`idld`) REFERENCES is_letalska_druzba(`idld`) ON DELETE RESTRICT
) ENGINE = InnoDB CHARACTER SET utf8 COLLATE utf8_bin AUTO_INCREMENT=3 ;

--
-- Dumping data for table `is_piloti`
--

INSERT INTO `is_piloti` (`idp`, `ime`, `priimek`, `idld`) VALUES
(1, 'Razem', 'Predjamski', 1),
(2, 'Janez', 'Kranjski', 2);

-- --------------------------------------------------------

--
-- Table structure for table `is_stevardese`
--

DROP TABLE IF EXISTS `is_stevardese`;
CREATE TABLE IF NOT EXISTS `is_stevardese` (
  `ids` int(11) NOT NULL AUTO_INCREMENT,
  `ime` varchar(50) NOT NULL,
  `priimek` varchar(50) NOT NULL,
  `idld` int(11) NOT NULL,
  PRIMARY KEY (`ids`),
  FOREIGN KEY (`idld`) REFERENCES is_letalska_druzba(`idld`) ON DELETE RESTRICT
) ENGINE = InnoDB CHARACTER SET utf8 COLLATE utf8_bin AUTO_INCREMENT=3 ;

--
-- Dumping data for table `is_stevardese`
```

```
--

INSERT INTO `is_stevardese` (`ids`, `ime`, `priimek`, `idld`) VALUES
(1, 'Brina', 'Sezase', 1),
(2, 'Ivy', 'Testisnu', 2);

-- --------------------------------------------------------

--
-- Table structure for table `is_tip_letala`
--

DROP TABLE IF EXISTS `is_tip_letala`;
CREATE TABLE IF NOT EXISTS `is_tip_letala` (
  `idtip` int(11) NOT NULL AUTO_INCREMENT,
  `oznaka` varchar(50) NOT NULL,
  PRIMARY KEY (`idtip`)
) ENGINE = InnoDB CHARACTER SET utf8 COLLATE utf8_bin AUTO_INCREMENT=5 ;

--
-- Dumping data for table `is_tip_letala`
--

INSERT INTO `is_tip_letala` (`idtip`, `oznaka`) VALUES
(1, 'A380'),
(2, 'DC9'),
(3, 'DC8'),
(4, 'Boeing 373');


-- --------------------------------------------------------

--
-- Table structure for table `is_letala`
--

DROP TABLE IF EXISTS `is_letala`;
CREATE TABLE IF NOT EXISTS `is_letala` (
  `idl` int(11) NOT NULL AUTO_INCREMENT,
  `idld` int(11) NOT NULL,
  `idtip` int(11) NOT NULL,
  `kapaciteta` int(11) NOT NULL,
  PRIMARY KEY (`idl`),
  FOREIGN KEY (idld) REFERENCES is_letalska_druzba(idld) ON DELETE RESTRICT,
  FOREIGN KEY (idtip) REFERENCES is_tip_letala(idtip) ON DELETE RESTRICT
) ENGINE = InnoDB CHARACTER SET utf8 COLLATE utf8_bin AUTO_INCREMENT=6 ;

--
-- Dumping data for table `is_letala`
--

INSERT INTO `is_letala` (`idl`, `idld`, `idtip`, `kapaciteta`) VALUES
(1, 1, 1, 150),
(2, 1, 1, 150),
(3, 1, 2, 250),
(4, 2, 3, 150),
(5, 1, 2, 250);

-- --------------------------------------------------------

--
-- Table structure for table `is_leti`
--

DROP TABLE IF EXISTS `is_leti`;
CREATE TABLE IF NOT EXISTS `is_leti` (
  `idlt` int(11) NOT NULL AUTO_INCREMENT,
  `idl` int(11) NOT NULL,
  `idp` int(11) NOT NULL,
  `ids` int(11) NOT NULL,
  `idle_vzleta` int(11) NOT NULL,
  `idle_pristanka` int(11) NOT NULL,
  PRIMARY KEY (`idlt`),
  FOREIGN KEY (`idl`) REFERENCES is_letala(`idl`) ON DELETE RESTRICT,
  FOREIGN KEY (`idp`) REFERENCES is_piloti(`idp`) ON DELETE RESTRICT,
  FOREIGN KEY (`ids`) REFERENCES is_stevardese(`ids`) ON DELETE RESTRICT,
  FOREIGN KEY (`idle_vzleta`) REFERENCES is_letalisce(`idle`) ON DELETE RESTRICT,
  FOREIGN KEY (`idle_pristanka`) REFERENCES is_letalisce(`idle`) ON DELETE RESTRICT
) ENGINE = InnoDB CHARACTER SET utf8 COLLATE utf8_bin AUTO_INCREMENT=5 ;

--
-- Dumping data for table `is_leti`
--

INSERT INTO `is_leti` (`idlt`, `idl`, `idp`, `ids`, `idle_vzleta`, `idle_pristanka`)
VALUES
(1, 3, 1, 1, 1, 2),
(2, 1, 2, 2, 1, 2),
(3, 2, 1, 1, 2, 1),
(4, 4, 2, 2, 2, 1);

-- --------------------------------------------------------

--
-- Table structure for table `is_rezervacija`
--

DROP TABLE IF EXISTS `is_rezervacija`;
CREATE TABLE IF NOT EXISTS `is_rezervacija` (
  `idr` int(11) NOT NULL AUTO_INCREMENT,
  `cena` int(11) NOT NULL,
  `idlt` int(11) NOT NULL,
  PRIMARY KEY (`idr`),
  FOREIGN KEY (`idlt`) REFERENCES is_leti(`idlt`) ON DELETE RESTRICT
) ENGINE = InnoDB CHARACTER SET utf8 COLLATE utf8_bin AUTO_INCREMENT=5 ;

--
-- Dumping data for table `is_rezervacija`
--

INSERT INTO `is_rezervacija` (`idr`, `cena`, `idlt`) VALUES
(1, 81000, 1),
(2, 70000, 2),
(3, 82000, 3),
(4, 83000, 4);
```

Diski in datoteke

Iztok Savnik, FAMNIT

Diski in datoteke

- SPUB shranjuje podatke na trdih diskih.
- To ima veliko posledic na zasnovo SUPB !
  - READ: prenos podatkov med trdim diskom in RAM.
  - WRITE: prenos med RAM na disk.
  - Obe vrsti operacij sta časovno potratni, zato se mora njihova uporaba planirati pazljivo!

Zakaj se vsega ne shrani v dinamični spomin?

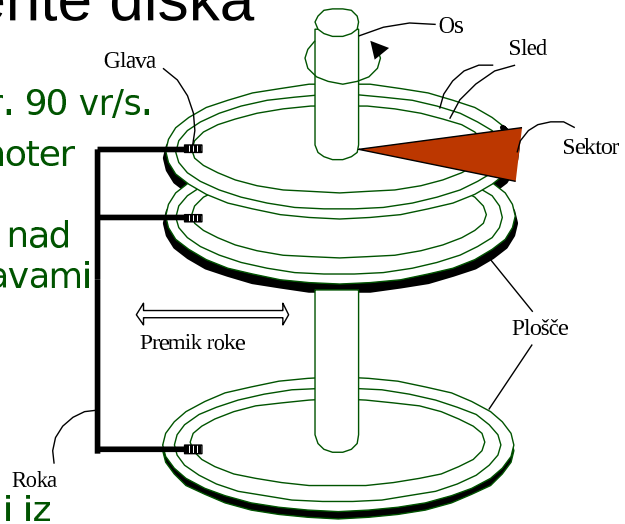
- *Prevelika cena*.
  - $100 ≈ 4GB RAM *ali* (skoraj) 1TB disk.
- *Dinamični spomin ne ohranja podatkov*. Želimo imeti shranjene podatke med večimi sejami s SUPB. (Očitno!)
- Tipična hierarhija pomnilnikov:
  - Dinamični spomin (RAM) za direktno delo s podatki.
  - Disk za glavno podatkovno bazo.
  - Trakovi za arhiviranje starejših verzij podatkov.

Diski

- Persistentno shranjevanje podatkov.
- Glavna prednost pred trakovi: *direkten dostop* vs. *sekvenčni*.
- 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.
  - Položaj podatkov na disku ima vpliv na hitrost delovanja SUPB!

Komponente diska

- Plošče se vrtijo npr. 90 vr/s.
- Roka se premika noter in ven s čim pozicioniramo glavo nad sledmi. Sledi pod glavami imenujemo *cilinder.*
- Branje/pisanje poteka samo preko ene glave v danem trenutku.
- *Bloki* so sestavljeni iz večih *sektorjev* (ki so fiksni).



Dostop do diskovne strani

- Čas dostopa (read/write) do bloka:
  - *čas iskanja* (premik roke/glave na željeno sled)
  - *rotacijska zakasnitev* (čakanje na blok pod glavo)
  - *čas prenosa* (prenos podatov iz/na površje plošče)
- Čas iskanja in rotacijska zakasnitev prevladujeta.
  - Čas iskanja se spreminja od 1 do 20msec
  - Rotacijska zakasnitev se spreminja od 0 to 10msec
  - Čas prenosa je tipično 1msec za 4KB stran
- Ključ za manjšo I/O ceno:
  - zmanjšati čas iskanja / rot. zakasnitev!
  - Hardware vs. software rešitev?

Primer: IBM Deskstar

The IBM Deskstar 14GPX.

- 3.5 inch, 14.4 GB
- Čas iskanja: 9.1 millisecond (msec)
- Ratacijska zakasnitev: 4.17 msec
- Preskok med sledmi: 2.2 msec
- Maksimalen iskalni čas: 15.5 msec
- 5 dvo-stranskih plošč, ki se vrtijo 7,200 vrt./min
- Plošča vsebuje 3.35 GB podatkov, gostota je 2.6 gigabit/inch^2.
- Hitrost prenosa je cca. 13 MB/s

Primerjava:

- Dostop do diska je cca. 10 msecs
- Dostop do glavnega spomina cca. 60 nanoseconds!

Primer: DDR

Standard	Clock	Cycle-time	I/O-Bus-clock	Data-transfers/s	VDDQ-volt	Module	Peak-transfer-rate
DDR-200	100 MHz	10 ns	100 MHz	200 Million	2.5±0.2 V	PC-1600	1600 MB/s
DDR-266	133 MHz	7.5 ns	133 MHz	266 Million	2.5±0.2 V	PC-2100	2100 MB/s
DDR-333	166 MHz	6 ns	166 MHz	333 Million	2.5±0.2 V	PC-2700	2700 MB/s
DDR-400	200 MHz	5 ns	200 MHz	400 Million	2.6±0.1 V	PC-3200	3200 MB/s

Ureditev blokov na disku

- Koncept: `*naslednja*' stran:
  - Bloki na isti sledi, ki sledijo
  - Bloki na istem cilindru, ki sledijo
  - Bloki na sosednem cilindru.
- Bloki ene datoteke naj bi bili organizirani sekvenčno na disku (glede na `naslednji'), da se minimizira čas iskanja in rotacijska zakasnitev.
- Sevenčno skeniranje: *branje vnaprej* tj. več sekvenčnih strani se prebere vnaprej; pridobitev na času!

RAID

- Diskovno polje (disk array).
  - Več diskov urejenih v polje, ki omogoča abstrakcijo: polje je videti kot en sam velik disk.
- Cilji: povečati performanse 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.
- Nivo 1: Zrcaljenje (dve identični kopiji).
  - Vsak disk ima zrcalno kopijo.
  - Paralelno branje; pisanje deluje nad dvemi diski.
  - Maksimalen prenos = prenos enega diska.
- Nivo 0+1: Pasovi in zrcaljenje.
  - Vzporedno branje; pisanje deluje nad dvemi diski.
  - Maksimalen prenos = skupen pretok.

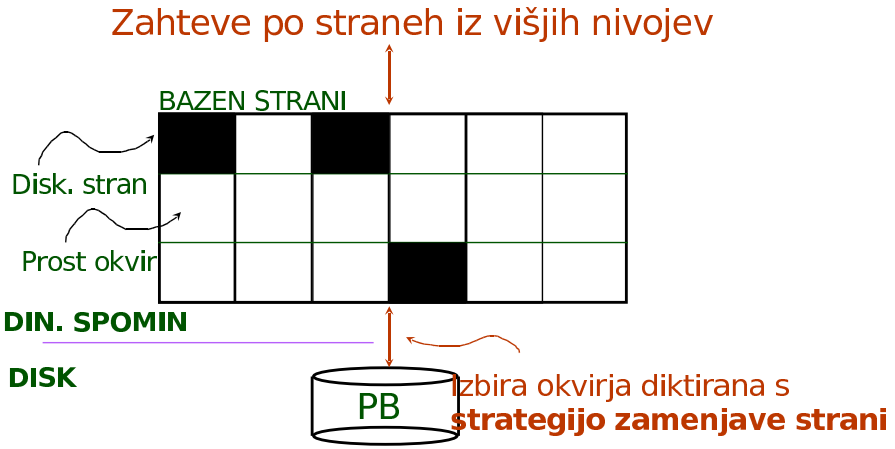
RAID Nivoji

- Nivo 3: Pariteta "po bitih".
  - Pasovna enota: En bit. En disk za preverjanje.
  - 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 preverjanje.
  - Paralelno branje za manjše zahteve, večje zahteve lahko uporabijo poln pretok.
  - Pisanje vključuje spremenjen blok in disk za preverjanje.
- Nivo 5: Porazdeljena pariteta "po blokih"
  - Podobno RAID 4; paritetni blok je porazdeljen po večih diskih

Delo z diskovnim prostorom

- Najnižji nivo SUPB dela z diskovnim pomnilniškim prostorom.
- Višji nivoji zahtevajo od tega nivoja:
  - alokacijo/de-alokacijo strani
  - branje /pisanje strani
- Zahteve po *sekvenci* strani mora biti izvršena tako, da sistem alocira strani v enem sekvenčnem prostoru na disku!
  - Višji nivoji ne potrebujejo vedeti kako je to narejeno oz. kako se dela s prostim prostorom.

Vmesni pomnilnik v SUPB



- *Pod. morajo biti v RAM da lahko SUPB dela z njimi!*
- Uporablja se tabela parov: ***<okvir#, stranid>***

Zahteva po strani ...

- Če iskana stran ni v bazenu:
  - Izberi okvir za *zamenjavo*
  - Če je okvir "umazan" potem se mora zapisati na disk
  - Preberi izbrano stran v izbrani okvir
- *Pripni* stran in vrni njen naslov.

 Če lahko predvidevamo zahteve (npr., sekv. pregled) potem lahko vnaprej preberemo več strani !

Več o delu z vmes. pomnilnikom

- Zahtevana stran mora biti *odpeta.*
- Vidno mora biti, če je stran spremenjena ali *"umazana"*:
  - *umazan* bit
- Stran v bazenu je lahko zahtevana večkrat,
  - Uporablja se *števec pripenjanj*. Stran je kandidat za zamenjavo če je *št. pripenjanj*= 0.
- Vzporednost in okrevanje:
  - dodatni I/O v primeru, da je stran izbrana za zamenjavo.
- Protokol *Dnevnika;*
  - *Pisanje-vnaprej* ; več kasneje.

Strategije za zamenjavo strani

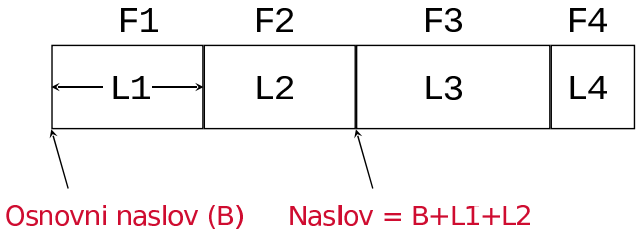
- Okvir izbran s *strategijo zamenjave:*
  - LRU, Ura (*Clock*), MRU, itd.
- Strategija ima lahko velik vpliv na # I/O operacij; odvisno od *vzorca dostopa*.
- *Sekvenčno prelivanje*:
  - Grda situacija povzročena z LRU + ponavljajoč sekvenčni pregled tabele.
  - # okvirjev < # strani datoteke vsaka zahteva povroči I/O. MRU je v tem primeru boljša (toda ne v vseh situacijah, seveda).

SUPB vs. OS Datotečni sistem

OS ureja diskovni prostor & vmesni pomnilnik: zakaj nebi OS za SUPB urejal ta opravila?

- Razlike v OS podpori: prenosljivost
- Nekatere omejitve, npr., datoteke se ne morajo raztezati preko velikosti diska.
- Delo z vmesnim pomnilnikom v SUPB zahteva zmožnost:
  - Pripeti stran v vmesni pomnilnik, prisiliti shranjevanje strani na disk (pomembno za SD & recovery),
  - Prilagoditev *strategije zamenjave strani in* branje strani vnaprej na osnovi obnašanja tipične operacije.

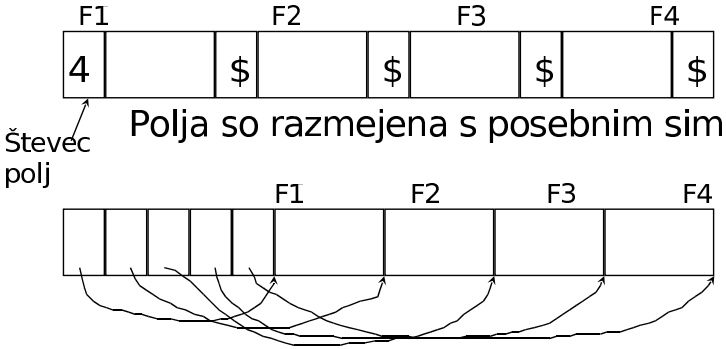
Format zapisa: Fiksna dolžina



- 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.

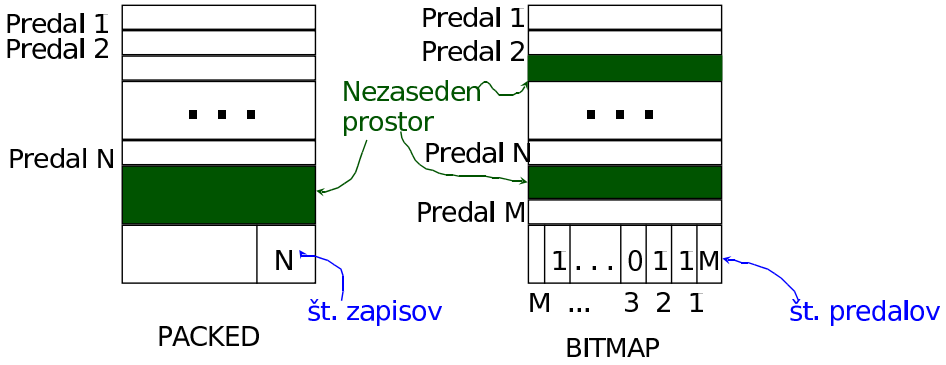
Format zapisa: Variabilna dolžina

- Dva alternativna formata (# polj je fiksno):



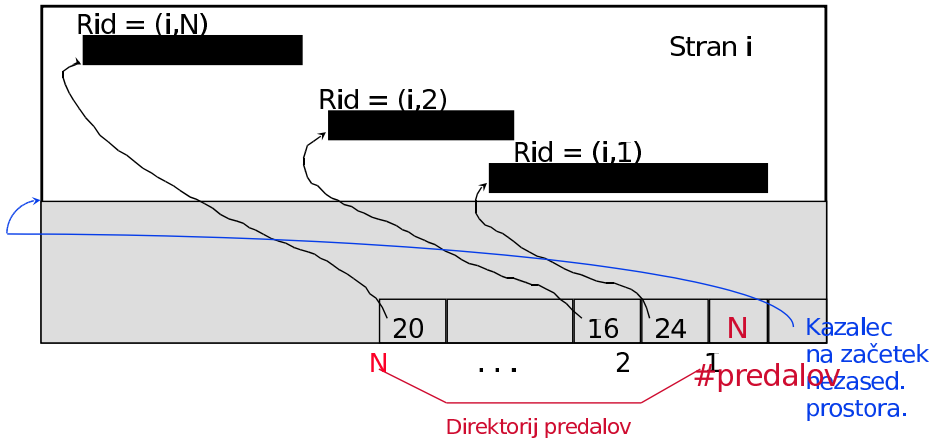
] Druga alternativa ponuja direkten dostop do i-tega polj; učinkovito shranjevanje *null*; dir. ne zaseda veliko prostora.

Format strani: Zapisi fiksne dolžine



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

Format strani: Zapisi variabilne dolžine



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

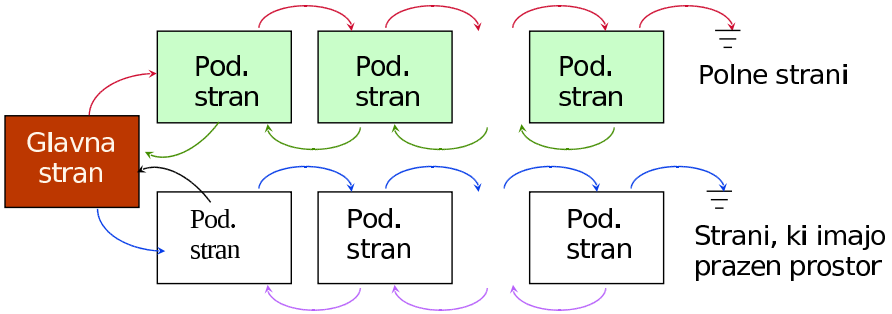
Datoteke zapisov

- 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

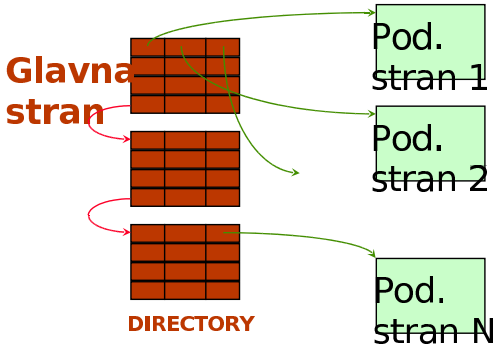
- Najenostavnejša datotečna struktura
  - Zapisi nimajo nobene urejenosti.
  - Tako kot se datoteka manjša in veča, tako se dodajajo in odvzemajo strani (bloki).
- Potrebno je shranjevati podatke o:
  - *straneh* datoteke
    - *neuporabljenem prostoru* na straneh
    - *zapisih* na strani
- Obstaja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

Sistemski katalogi

- Za vsak indeks:
  - struktura (npr., B+ drevo) in iskalni ključi
- Za vsako relacijo:
  - ime, ime datoteke, datotečna struktura
  - imena in tipi atributov
  - imena indeksov
  - integritetne omejitve
- Za vsak pogled:
  - ime pogleda in definicija
- Statistika, autorizacija, velikost bazena strani, itd.

 Katalogi so shranjeni kot relacije!

Attr_Cat(attr_name, rel_name, type, position)

attr_name	rel_name	type	position
attr_name	Attribute_Cat	string	1
rel_name	Attribute_Cat	string	2
type	Attribute_Cat	string	3
position	Attribute_Cat	integer	4
sid	Studenti	string	1
ime	Studenti	string	2
login	Studenti	string	3
star	Studenti	integer	4
ocena	Studenti	real	5
fid	Fakulteta	string	1
fime	Fakulteta	string	2
placa	Fakulteta	real	3

Povzetek

- Disk je poceni persistentni pomnilniški medij.
  - Direkten dostop, hitrost je odvisna od lokacije strani; pomembno je urediti podatke na disku sekvenčno; minimizirata se *čas iskanja* in *zakasnitev rotacije*.
- Vmesni pomn.: stran prenešena iz diska v RAM.
  - Stran ostane v RAM tako dolgo, dokler je ne sprosti tisti, ki jo je zahteval.
  - Zapisana je na disk, ko je izbrana za zamenjavo (po tem, ko je sproščena).
  - Izbira okvirja za zamenjavo se izvede na osnovi *strategije za zamenjavo.*
  - Poskuša se prenesti več strani za kasnejšo uporabo.

Povzetek

- SUPB vs. OS Datotečni sistem
  - SUPB potrebuje lastnosti, ki jih OS ne nudi. Na primer, eksplicitno zapisovanje strani na disk, kontroliranje vrstnega reda strani na disku, datoteke preko diskov, možnost vnaprejšnjega branja večje količine strani, prilagajanje dela vmesnega pomnilnika na vzorce dostopa, itd.
- Zapisi variabilne dolžine s specificiranimi odmiki polj v direktoriju nudi direkten dostop do i-tega polja in omogoča null vrednosti.
- Format strani s predali omogoča shranjevanje zapisov variabline dolžine in omogoča premikanje zapisov po strani.

Povzetek

- Nivo datoteke omogoča delo s stranmi datoteke in nudi abstrakcijo “zbirke zapisov”.
  - Strani s praznim prostorom se identificirajo z dvojno povezanim seznamom ali preko direktorija.
- Indeksi omogočajo učinkovit dostop do zapisov na osnovi vrednosti nekega atributa.
- Katalog shranjuje podatke o relacijah, indeksih in pogledih.

Indeksi

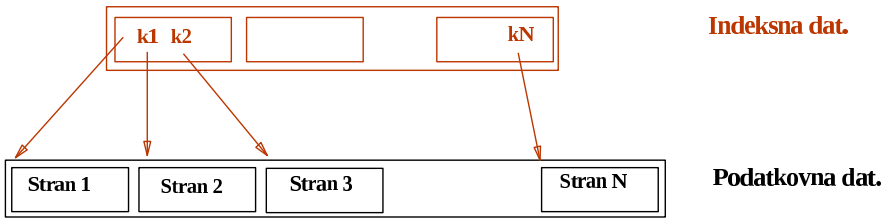
Iztok Savnik, FAMNIT

Drevesni indeksi

- *Tri alternative za podatkovne vpise* **k***:
  - Podatkovni zapisi z vrednostjo ključa **k**
  - <**k**, rid pod. zapisa z vrednostjo iskalnega ključa **k**>
  - <**k**, seznam rid pod. Zapisov z iskalnim ključem **k**>
- Izbira je ortogonalna na *indeksno tehniko* uporabljeno za iskanje podatkovnih vpisov **k***.
- Drevesna indeksna struktura podpira tako *iskanje področja* kot tudi *iskanje z enačajem*.
- *ISAM*: statična struktura; *B+ tree*: dinamična struktura, drevo je vedno uravnoteženo.

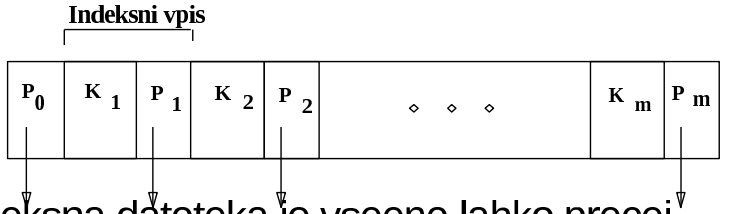
Iskanje področja

- ``*Poišči vse študente s povprečjem > 9.0*''
  - Če so podatki v urejenih datoteki naredimo binarno iskanje in nato pregled.
  - Cena binarnega iskanja je visoka.
- Ideja: Kreiraj ´´indeksno´´ datoteko.

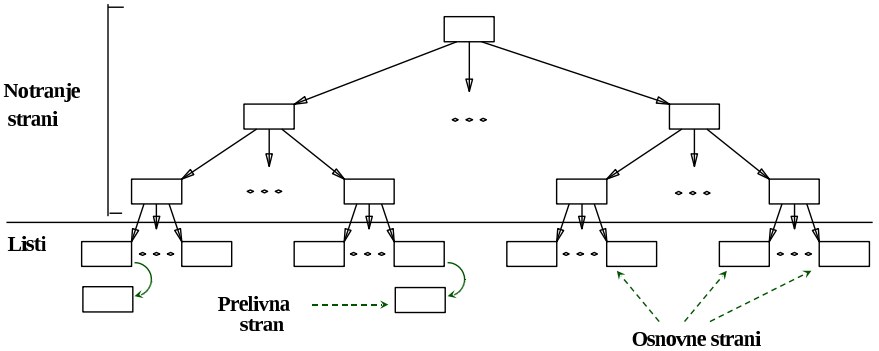


 Iščemo lahko preko manjše indeksne datoteke!

ISAM



- Indeksna datoteka je vseeno lahko precej velika. Idejo lahko ponovimo!



 Listi (strani) vsebujejo podatkovne vpise.

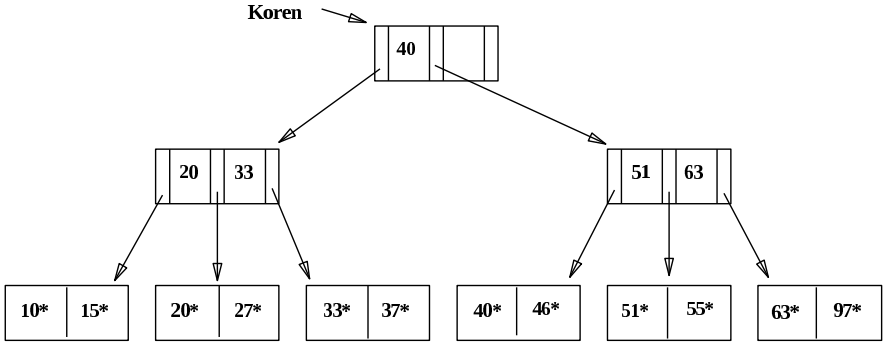
ISAM

- *Kreiranje datoteke*: Liste zasežemo sekvenčno sortirane po iskalnem ključu; zasežemo indeksne strani in potem še prostor za prelivne strani.
- *Indeksni vpisi*: <vrednost iskalnega ključa, id strani>; *usmerjajo iskanje podatkovnih vpisov*, ki so v listih.
- *Iskanje*: Začni v korenu; uporabi primerjavo ključev za iskanje lista. Cena =log $_F$ N ; F = # vpisov/ind.stran, N = # listov.
- *Vstavi*: Poišči list in zapiši pod. vpis.
- *Izbriši*: Poišči list in izbriši vpis iz lista; izbriši prelivno stran, če je prazna.

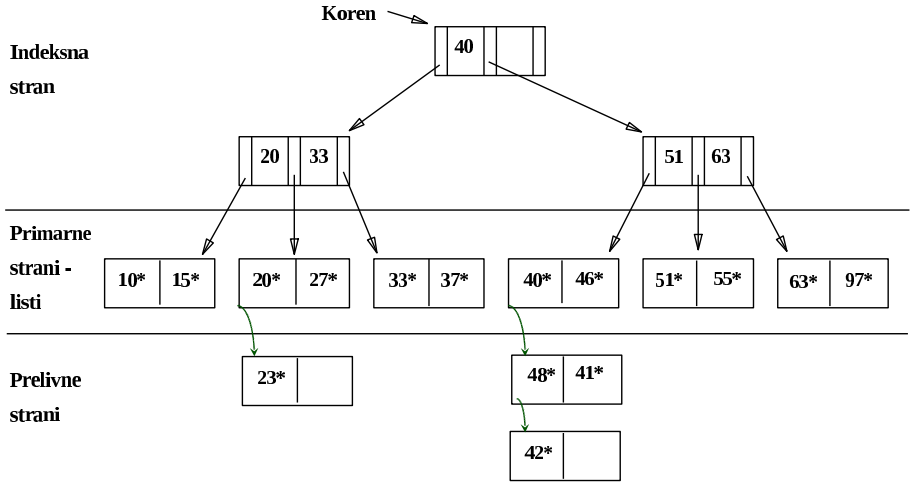
Statična struktura drevesa: *vstavljanje/brisanje samo v list*

Primer ISAM drevesa

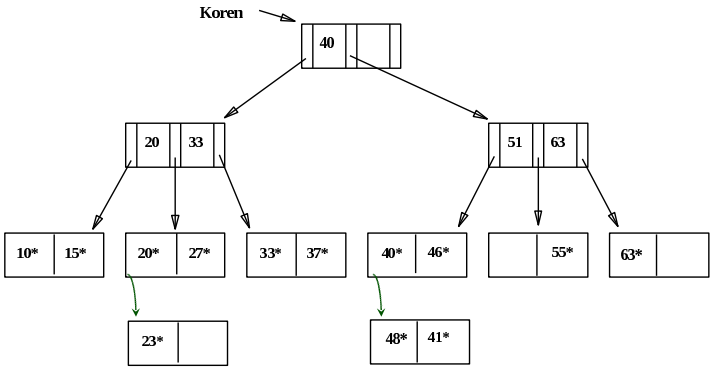
- Vsako vozlišče lahko vsebuje dva vpisa; ni potrebe po kazalcih na naslednji list. (Zakaj?)



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



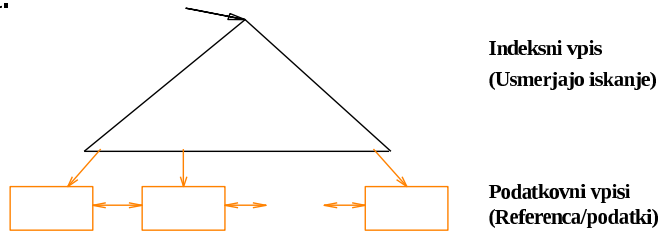
... potem brisanje 42*, 51*, 97*



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

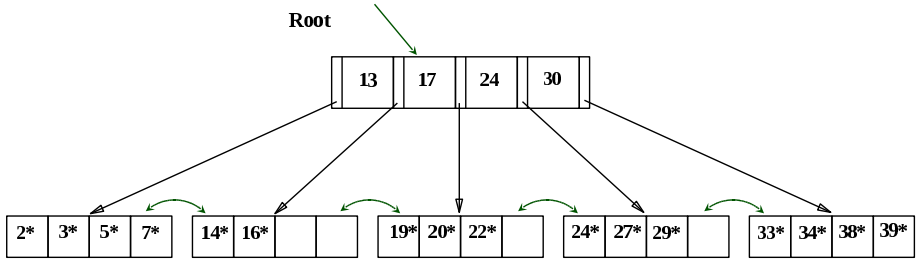
B+ Drevo: Najbolj razširjen indeks

- Kompleksnost op. Vstavi/Izbriši = $\log_F N$
  (F = fanout, N = # listov)
- Drevo je zelo uravnoteženo.
- Indeksne strani so zapolnjene vsaj 50%(razen korena). Vsako vozlišče vsebuje **d** <= *m* <= 2**d** vpisov.
  Parameter **d** imenujemo *stopnja* odrevesa.
- Učinkovito podpira iskanje po enakosti in iskanje področja.



Primer B+ Drevesa

- Iskanje se začne v korenu; primerjava ključev vodi do lista s pod. vpisom (kot pri ISAM).
- Išči 5*, 15*, vse pod. vpise >= 24* ...



- *Iz iskanja 15* vemo, da ključa ni v drevesu!*

B+ Drevesa v praksi

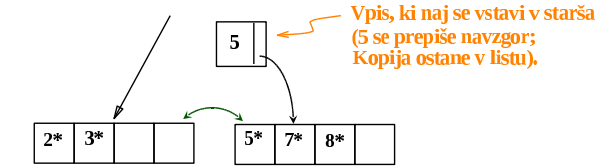
- Tipična stopnja: 100. Tipična zapolnjenost: 67%.
  - Povprečen fanout = 133
- Tipične kapacitete:
  - Višina 4: 133^4 = 312,900,700 zapisov
  - Višina 3: 133^3 = 2,352,637 zapisov
- Večinoma lahko shranimo višje nivoje drevesa v vmesni pomnilnik:
  - Nivo 1 = 1 stran = 8 Kb
  - Nivo 2 = 133 strani = 1 Mb
  - Nivo 3 = 17,689 strani = 133 MB

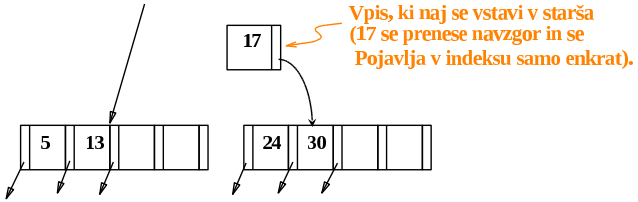
Vstavljanje pod. vpisa v B+ drevo

- Poišči list *L.*
- Vstavi pod. vpis v *L*.
  - Če ima *L* zadosti prostora, je narejeno!
  - Sicer je potrebno *razcepiti* *L (v L in novo vozlišče L2)*
    - Enakomerno porazdeli vpise in **prepiši** srednji ključ gor.
    - Vstavi indeksni vpis, ki kaže na *L2,* v starša 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.

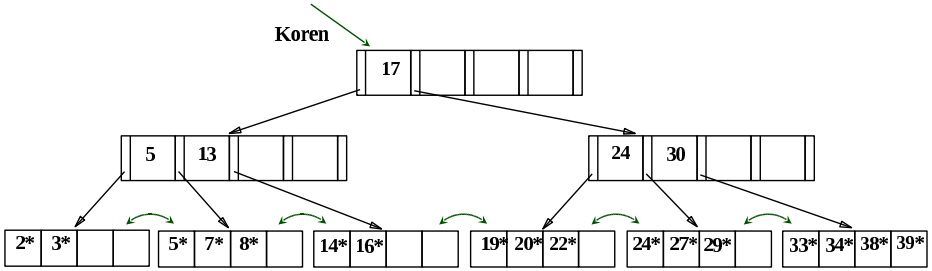
Vstavljanje 8* v B+ drevo (primer)

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





Primer B+ drevesa po vnosu 8*

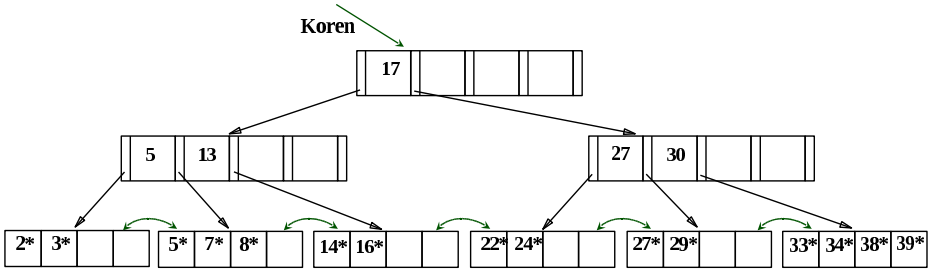


- Vidimo, da se je razcepil koren in se je povečala višina drevesa.
- V tem primeru bi se lahko izognili razcepu strani z porazdelitvijo vpisov; to se običajno ne uporablja v praksi.

Brisanje pod. vpisa iz B+ drevesa

- Začni pri korenu in poišči list z vpisom.
- Izbriši 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.

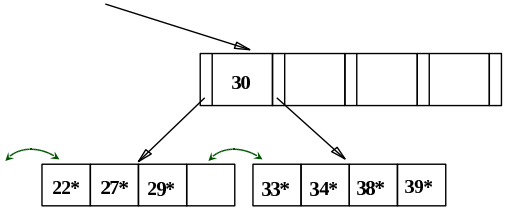
Drevo po vstavljanju 8* in brisanju 19* in 20* ...

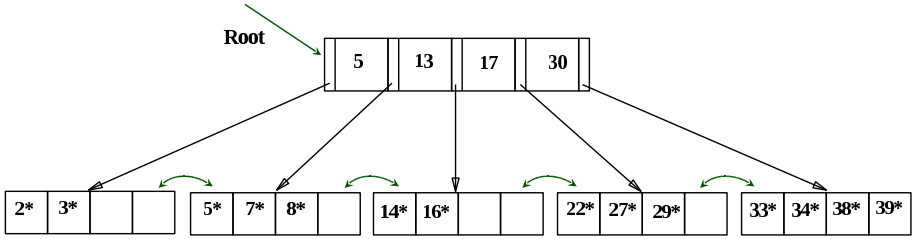


- Brisanje 19* je enostavno.
- Brisanje 20* je narejeno s porazdelitvijo vpisov. Srednji ključ je prepisan navzgor.

... in brisanje 24*

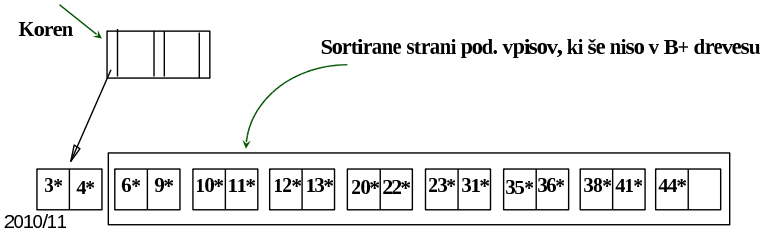
- Potrebno zlivanje.
- Zlitje podatkovnih vpisov na desni ter indeksnih vpisov spodaj.





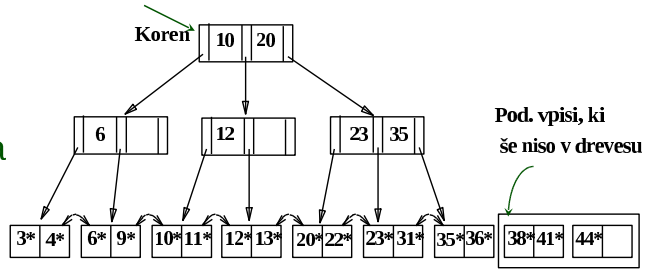
Polnjenje B+ drevesa

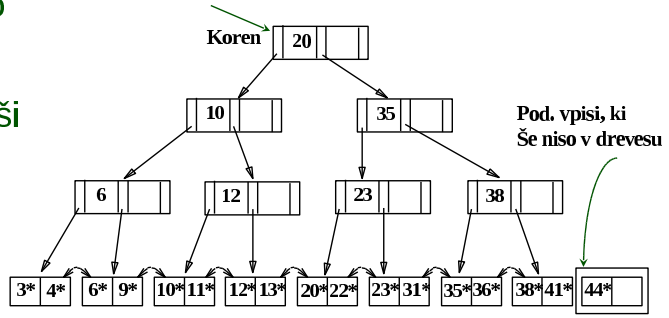
- Polnjenje B+ drevesa z vstavljanjem posameznega zapisa je lahko zamudno v primeru, da je podatkov veliko.
- *Polnjenje* lahko naredimo veliko hitreje.
- *Inicializacija*: Uredi podatkovne vpise, dodaj kazalec na prvo stran v novem korenu.



Polnjenje

- Indeksni vpisi, ki kažejo na liste so vedno vstavljeni na skrajno desno indeksno stran tik nad listi. Ko se ta napolni jo razcepimo – cepitev se lahko ponovi na višjih nivojih.
- Algoritem je hitrejši od vstavljanja zapisov.





Povzetek polnjenja B+ drevesa

- Možnost 1: iterativno vstavljanje.
  - Počasno.
  - Ne dobimo sekvenčen zapis indeksnih strani po nivojih.
- Možnost 2: *Polnjenje (Bulk Loading)*
  - Prednosti pri kontroli sočasnega dostopa.
  - Manj I/O operacij med gradnjo.
  - Listi so shranjeni sekvenčno.
  - Lahko kontroliramo “faktor zapolnjenosti” na straneh.

Stopnja drevesa

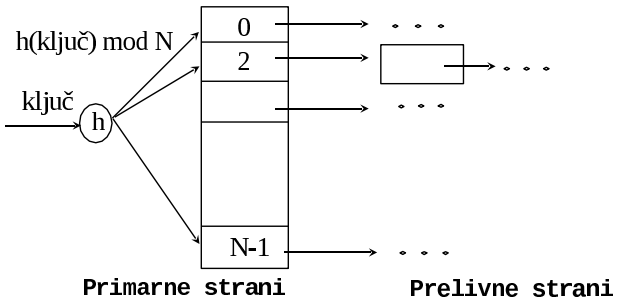
- *Stopnja* (**d**) **–** število ind. vpisov v vozliščih drevesa.
- V praksi se stopnja prevede v kriterij: *"vsaj polovica napolnjena"*.
  - Zapisi in iskalni ključi variabilne dolžine povzročijo variabilno število zapisov v listih.
  - Tudi v primeru polj fiksne dolžine lahko imamo vpise z variabilno dolžino v listih. (Možnost (3)).

Razpršilni indeksi

- *Razpršilni indeksi* so dobri za iskanje z enačajem. Ne uporabljajo se za iskanje področja.
- Obstajajo *statične* in *dinamične* vrste indeksa; podobno primerjavi ISAM vs. B+ drevo.
- Ogledali si bomo:
  - varianto statičnega indeksa
  - razširljiv razpršilni indeks (dinamičen)

Statični razpršilni indeks

- # primarne strani so fiksne, zasežene sekvenčno ter niso nikoli sproščene.
- **h**(*k*) mod N = skupek h kateremu pripada podatkovni vpis z ključem *k.* (N = # skupkov)



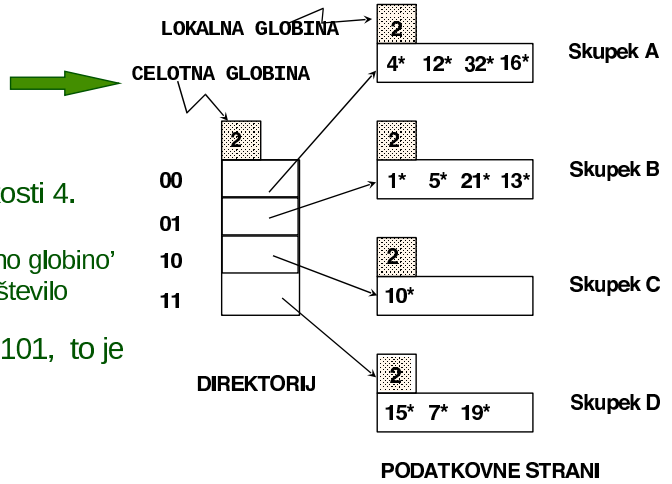
Statični razpršilni indeks

- Skupki vsebujejo *podatkovne vpise*.
- Razpršilna fn se uporabi na *iskalnem ključu.* Vrednosti se morajo *razpršiti* 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.

Razširljiv 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!

Primer

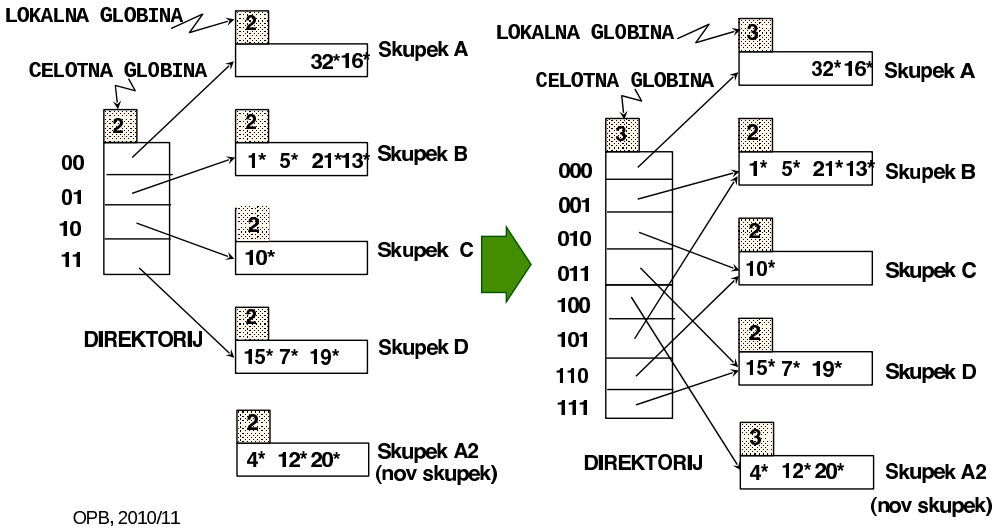


- Direktorij je polje velikosti 4.
- Iskanje skupka za *r:*
  - Vzemi zadnjo `celotno globino' # bitov od **h**(*r*) in to število uporabi kot indeks.
    - **h**(*r*) = 5 = binarno 101, to je skupek 01.

❖ Dodajanje:

- ❖ Če je skupek poln potem ga razcepi ter zaseži novo stran in porazdeli vpise.
  - ❖ *Če je potrebno* se podvoji direktorij.
  - ❖ Videli bomo, da razcep strani velikokrat ne zahteva podvojitev; to vidim tako, da primerjamo celotno in lokalno globino skupka.

Vstavi **h**(r)=20 (podvojitev)



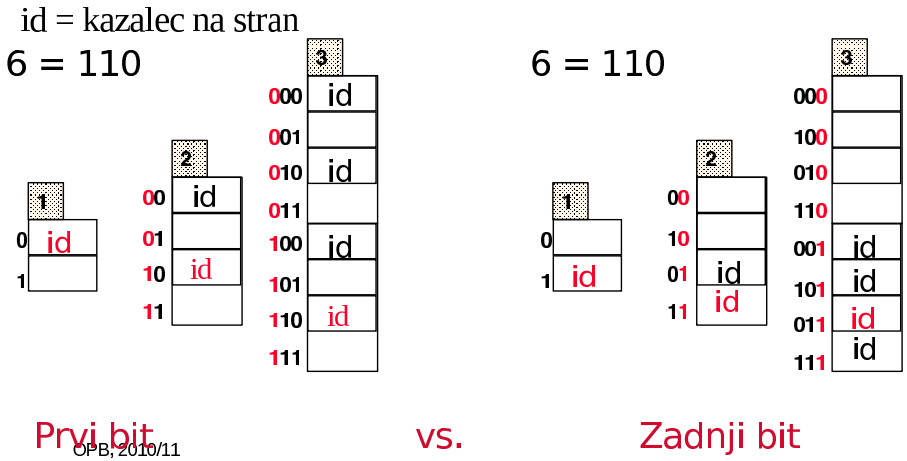
Komentarji

- 20 = binarno 10100. Zadnja **2** bita (00) nam povesta, da *r* pripada A ali A2; **3.** bit pove v kateri.
  - *Celotna globina direktorija*: Max # bitov potrebnih za skupka h kateremu pripada vpis.
  - *Lokalna globina skupka*: # bitov potrebnih za določitev ali vpis pripada k skupku.
- Kdaj razcep skupka povzroči podvojevanje direktorija?
  - Pred vstavljanjem: *lokalna globina* skupka = *celotna globina*.
  - Vstavljanje povzroči: *lokalna globina* postane > *celotne globine.*
- Podvojevanje direktorija.
  - Direktorij se podvoji s kopiranjem.
  - V direktorij se vpiše kazalec na novo stran.
  - Prvi bit se uporabi za določitev prave strani pri razcepu.

Podvojevanje direktorija

Zakaj uporabiti prvi/zadnji bit v direktoriju?

 Omogoča podvojitev direktorija s kopiranjem!



Komentarji

- Cena “iskanja z enačajem”:
  - Če direktorij gre v dinamični spomin potem so poizvedbe na osnovi enakosti izvedene z branjem enega bloka, sicer z dvemi.
- Tipične številke:
  - 100MB datoteka; 100 zlogov/zapis; 4K stran; 1,000,000 pod. vpisov; direktorij 25,000 elementov; velika verjetnost da bo cel direktorij v din. spominu.
- Problemi:
  - V primeru, da je porazdelitev vrednosti razpršilne funkcije ni enakomerna (skewed) lahko direktorij postane zelo velik.
  - Veliko vpisov z isto vrednostjo razpršilne funkcije povzroča probleme!
- **Brisanje** podatkovnega vpisa:
  - Če brisanje pod. vpisa izprazni skupek ga lahko združimo z njegovo “razcepljeno sliko”.
  - Če se prva polovica direktorija ujem z drugo jih lahko združimo.

Povzetek

- Drevesni indeksi so idealni za iskanje področij vendar dobri tudi za iskanje z enačajem.
- ISAM je statični indeks.
  - Spreminjajo se samo listi; potrebne so prelivne strani.
  - Prelivne strani lahko zelo poslabšajo učinkovitost strukture.
- B+ drevo je dinamična struktura.
  - Operaciji vnos/brisanje ohranjata uravnoteženost drevesa; cena operacij je $\log_F N$ cost.
  - Velik fanout (**F**) omogoča da višina drevesa redko presega 3 ali 4.
  - Drevo je skoraj vedno boljše od urejene datoteke.

Povzetek

- B+ drevo
  - Tipično so indeksne strani zapolnjene 67%.
  - B+ drevo je običajno boljše od ISAM.
  - Če so podatkovni vpisi enaki podatkovnim zapisom potem cepitve listov spremenijo identifikatorje zapisov!
- Polnjenje B+ drevesa je lahko veliko hitrejše kot vstavljanje posameznih zapisov v drevo.
- B+ drevo je najbolj razširjen indeks v sistemih za delo s podatkovnimi bazami.
  - Je tudi ena od najbolj optimiziranih komponent SUPB.

Povzetek

- Razpršilni indeksi: dobri za iskanje z enačajem in se jih ne uporablja za iskanje področja.
- Statični razpršilni indeksi lahko vodijo do dolgih prelivnih verig.
- Razširljiv razpršilni indeks nima prelivnih strani razen v primeru duplikatov.
  - *Duplikati zahtevajo uporabo prelivnih strani!*
- Ko se skupek zapolni se podvoji direktoriij.
  - Problemi v primeru neenakomerne porazdelitve vrednosti razpršilnih funkcij.

Datoteke in indeksi

Iztok Savnik, FAMNIT

Podatki na pomnilniških medijih

- Diski: Direkten dostop do blokov.
  - Branje večih zaporednih strani je cenejše kot v branje naključnih strani.
- Trakovi: Sekvenčni dostop do strani.
  - Cenejši medij kot diski; uporablja se za arhiviranje podatkov.
- Datotečna organizacija: Organizacija podatkov na pomnilniških medijih.
  - ID zapisa (rid) služi za identifikacijo fizične lokacije zapisa
  - Indeksi so podatkovne strukture, ki omogočajo iskanje ID-jev zapisov na osnovi podanih vrednosti atributov ključa indeksa.
- Arhitektura: Vmesni pomnilnik prenaša strani iz zunanjega pomnilniškega medija v bazen strani. Nivo datotek in indeksov dela z vmesnim pomnilnikov.

Alternativne datotečne organizacije

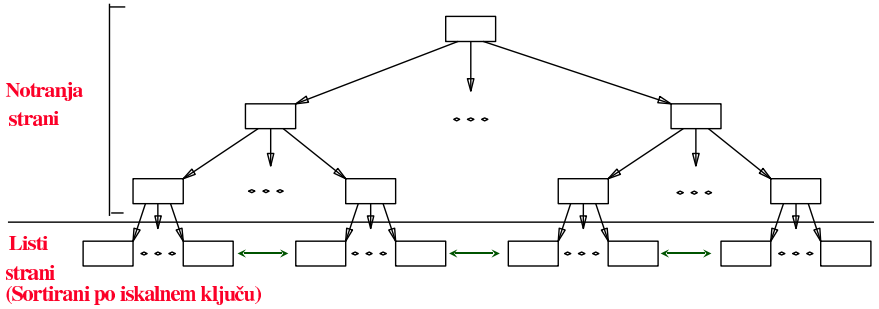
Obstaja več alternativ, *vsaka je idealna za določene situacije in ne tako dobra v drugih primerih*:

- Neurejene datoteke: Primerne, ko je tipični dostop branje celotne datoteke.
- Sortirane datoteke: BNajboljše v primer, da iščemo zapise v določenem vrstnem redu ali želimo preiskati določen interval celotne množice zapisov.
- Indeksi: Podatkovna strukture, 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.

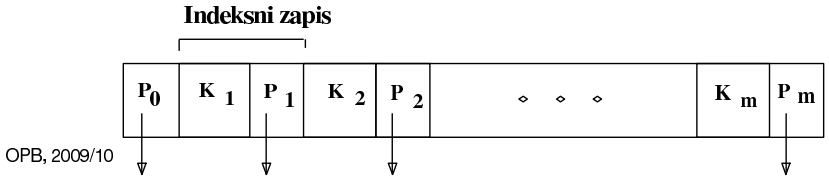
Indeksi

- *Indeks* pohitri iskanje zapisov na osnovi *polj iskalnih ključev* indeksa.
  - Vsaka podmnožica polj relacije je lahko iskalni ključ indeksa na dani relaciji.
  - *Iskalni ključ* ni enak *ključu* (minimalni množici polj, ki enolično določa zapis v relaciji).
- Indeks vsebuje množico *podatkovnih zapisov.*
- Indeks podpira učinkovito iskanje podatkovnih zapisov **k*** z dano vrednostjo ključa **k**.
  - Za dan ključ **k** lahko poiščemo podatkovni zapis **k*** z enim dostopom do diska. (Podrobnosti kmalu …)

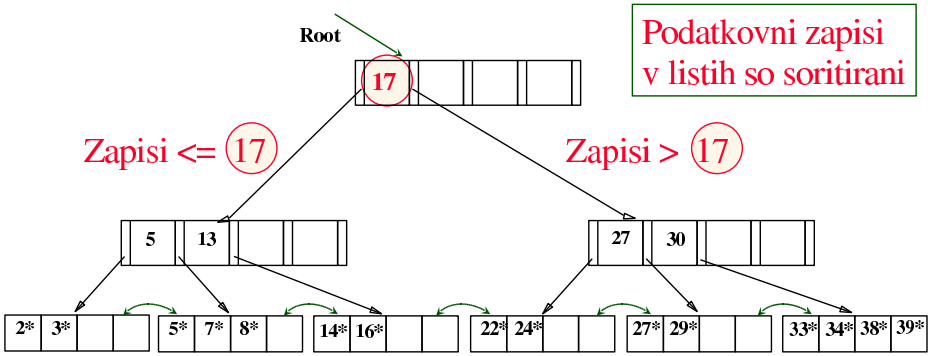
B+ Drevo



- Listi vsebujejo *podatkovne zapise*, ki so urejeni (prev & next)
- Notranje strani imajo *indeksne zapise;* samo za iskanje:



Primer B+ Drevo



- Poišči 28*? 29*? Vse > 15* in < 30*
- Insert/delete:
  - Poišči podatkovni zapis v listu in ga spremeni.
  - Včasih se pojavi potreba po restrukturiranju staršev…
  - … spreminjanje vozlišč "nad" spremenjenim zapisom (zbrisanim/vstavljenim).

Razpršilni indeksi

- Dobra rešitev za selekcijo z enakostjo.
- Indeks je zbirka *skupkov.*
  - Skupek = *primarna stran* plus nič ali več *prelivnih strani*.
  - Skupek vsebuje podatkovne vpise.
- *Razpršilna funkcija* **h**: **h**(*r*) = skupek k kateremu pripada podatkovni zapis *r*. **h** uporabi polja *iskalnega ključa* relacije *r.*
  - *Ni potrebe po "indeksnih zapisih" v tej shemi.*

Podatkovni vpis **k***

- Podatkovni vpis k* je lahko:
  - Podatkovni zapis z vrednostjo ključa **k.**
  - <**k**, rid>
  - <**k**, seznam rid>
- Izbira alternative za podatkovne vpise je ortogonalna izbiri indeksne tehnike uporabljene za iskanje podatkovnih vpisov z vrednostjo ključa **k**.
  - Primeri indeksnih tehnik: B+ drevesa, razpršilni indeksi.
  - Tipično indeks vsebuje dodatno informacijo, ki usmerja iskanje želenih podatkovnih zapisov.

Podatkovni vpis - možnosti

- Možnost 1:
  - V tem primeru je indeksna struktura *datotečna organizacija*.
    - Alternativa neurejeni datoteki ali sortirani datoteki. .
  - Samo en indeks nad dano zbirko podatkov lahko uporabi možnost 1.
    - Sicer imamo podvojene podatke, ki predstavljajo redundanco ter možnost nekonsistence.
  - V primeru, da podatkovni zapisi zasedejo veliko prostora je število strani, ki vsebuje podatke veliko.
    - Tipično je tudi količina dodatnih podatkov večja.

Možnosti za podatkovne vpise

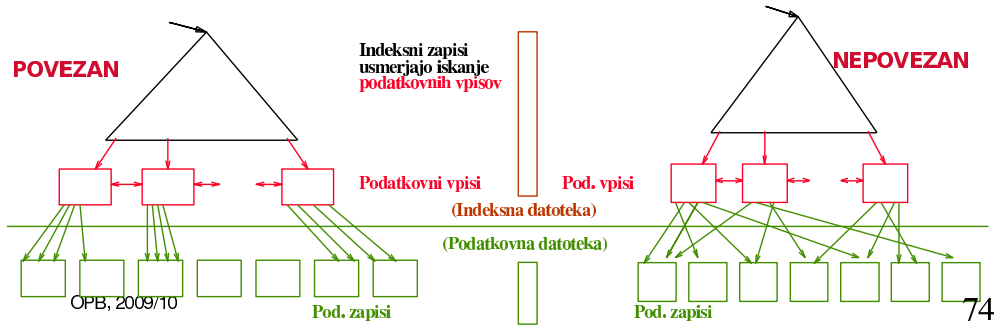
- Možnosti 2 in 3:
  - Podatkovni vpisi so tipično precej manjši kot podatkovni zapisi.
    - Boljše od možnosti 1 v primeru velikih podatkovnih zapisov, še posebej če so iskalni ključi majhni (del indeksne strukture, ki je namenjen iskanju je precej manjši kot pri alternativi 1.).
  - Možnost 3 je bolj optimalna glede prostora od možnosti 2, vendar so zapisi variabilne dolžine tudi v primeru, da so iskalni ključi fiksne dolžine.

Klasifikacija indeksov

- *Primarni* vs. *Sekundarni*:
  - Če iskalni ključ vsebuje primarni ključ potem imenujemo indeks *primarni*.
  - *Unique* indeks: Iskalni ključ je kandidatni ključ.
- *Povezan* vs. *Nepovezan*:
  - Če je urejenost podatkovnih zapisov enaka ali blizu urejenosti podatkovnim vpisom potem pravimo, da je indeks *povezan*.
  - Možnost 1 pogojuje povezan indeks; v praksi tudi obratno: povezan indeks pogojuje možnost 1 (sortirane datoteke so redke).
  - Datoteka je lahko povezana z indeksom po samo enem ključu.
  - Cena branja podatkovnega zapisa preko indeksa zelo varira v odvisnosti ali je indeks povezan ali ne!

Povezan vs. Nepovezan indeks

- Recimo, da je možnost 2 uporabljena za podatkovne vpise in so pod. zapisi shranjeni v neurejeni datoteki.
  - Gradnja povezanega indeksa: najprej se sortira neurejena datoteka (prazen prostore za bodoče zapise).
  - Možna je uporaba so prelivnih strani za vstavljanje zapisov. V tem primeru je urejenost podatkov "blizu" prave urejenosti.



Cenovni model za analizo

Poenostavitev: ignoriramo ceno CPU:

- **B:** Število podatkovnih strani.
- **R:** Število zapisov na strani.
- **D:** Povpr. čas branja ali pisanja diskovne strani.
- Merjenje števila branja/pisanja strani iz/na disk.
  - Ignoriramo branje vnaprej.
  - Tudi I/O cena je aproksimacija.
- Analiza:
  - Poenostavitve: CPU,prepostavke enakomerne porazdelitve, itd.

☛ *Zadosti dobro za prikaz splošnih trendov!*

Primerjava datotečnih organizacij

- Neurejena datoteka (vstavljanje na koncu)
- Sortirana datoteka, urejene po *iskalnem ključu*
- Povezano B+ drevo na osnovi iskalnega ključa, Možnost 1
- Neurejena datoteka z nepovezanim B + drevesom po iskalnem ključu
- Neurejena datoteka z nepovezanim razpršilnim indeksom po iskalnem ključu

Operacije

- Pregled: branje vseh zapisov iz diska
- Iskanje z enačajem
- Iskanje področja
- Vstavljanje zapisa
- Brisanje zapisa

Predpostavke v analizi

- Neurejene datoteke:
  - Iskanje z enačajem; samo en zapis se ujema.
- Sortirane datoteke:
  - Datoteke so stisnjene po brisanju zapisov.
- Indeksi:
  - Možnosti (2), (3): velikost pod. vpisa = 10% zapisa
  - Razpršilni indeks: ni prelivnih strani.
    - 80% zapolnjen ==> velikost datoteke = 1.25 količina pod.
  - Drevo: 67% zapolnjeno (tipično).
    - Velikost datoteke ≈ 1.5 količina podatkov

Predpostavke v analizi

- Pregled:
  - Listi drevesa so med sabo povezani v seznam.
  - Pregledujemo indeksne zapise in podatkovne zapise za nepovezan indeks.
- Iskanje področja:
  - Drevesne indekse uporabimo za omejitev množice prebranih podatkovnih zapisov.

Ocene operacij

	(a) Pregled	(b) Enakost	(c) Področje	(d) Insert	(e) Delete
(1) Neur. dat					
(2) Urejen. dat					
(3) Povezan drevesni ind					
(4) Nepovezan drevesni ind					
(5) Nepovezan razprš. ind					

☛*Ocene slonijo na precej predpostavkah!*

Ocene: Neurejena datoteka

- Pregled: **B*D**
  - Branje celotne datoteke
- Enakost: **0.5*B*D**
  - V povprečju pregledamo polovico datoteke
  - Če zapisa ni pregledamo celotno datoteko!
- Področje: **B*D**
  - Nujno je pregledati celotno datoteko
- Insert: **2*D**
  - Vstavljanje na konec datoteke; branje + pisanje ene strani
- Delete: **Iskanje+D**
  - Iskanje vsebuje branje strani

Ocene: Urejena datoteka

- Pregled: **B*D**
- Enakost: **D*log $_2$ B**
  - Binarno iskanje; ni všteto iskanje zapisa po eni strani
- Področje: **D*(log $_2$ B + #strani z iskan. zapisi)**
  - Binarno iskanje + branje rezultatov
- Insert: **Iskanje+ B*D**
  - Predpostavimo, da je najden zapis na sredini
  - Pri vstavljanju je potrebno prepisati polovico datoteke (2*1/2*B*D)
- Delete: **Iskanje+B*D**
  - Enako kot pri vstavljanju zapisa.

Ocene: Povezano B+ drevo

- Pregled: **1.5*B*D**
  - Velikost datoteke ≈ 1.5 količina podatkov.
- Enakost: **D*log $_F$ 1.5B**
  - F je število indeksnih zapisov v vozlišču drevesa.
  - Višina drevesa = log $_F$ 1.5*B strani
- Področje: **D*(log $_F$ 1.5B + #str z isk. zapisi)**
  - Enakost + branje iskanih strani
- Insert: **Iskanje+ D**
  - Iskanje vsebuje branje pod. strani + zapis podatkovne strani (D)
  - Ni všteto morebitno popravljanje drevesa.
- Delete: **Iskanje+D**

Ocene: Nepovezano B+ drevo

- Pregled: **B*D*(R+0.15)**
  - Ena stran za vsak pod. zapis = B*D*R
  - Pod. vpisi 10% ; drevo = 1.5 podatkov
  - Velikost drevesa = 0.1 * B*D * 1.5 = B*D*0.15
- Enakost: **D*(1+ log $_F$ 0.15*B)**
  - Višina drevesa = log $_F$ 1.5*B strani
  - … + ena podatkovna stran
- Področje: **D*(log $_F$ 0.15*B + #strani z isk. zapisi)**
  - Enakost + branje iskanih strani
- Insert: **Iskanje + 2*D**
  - 2*D = branje in pisanje pod. strani
- Delete: **Iskanje + 2*D**

Ocene: Razpršilni indeks

- Pregled: **B*D*(R+0.125)**
  - Pod. vpisi 10% ; r.datoteka 80% zapolnjena (1.25)
  - Velikost r.datoteke = 0.1 * BD * 1.25 = B*D*0.125
  - Ne upošteva se vmesni pomnilnik !
- Enakost: **2*D**
  - Podatkovni vpis + podatkovni zapis
- Področje: **D*(B*0.125 + #strani z isk. zapisi)**
  - Celotnen indeks + iskani zapisi
- Insert: **Iskanje+ 2*D**
  - 2D = branje + pisanje strani
- Delete: **Iskanje+ 2*D**

Ocene operacij

	(a) Pregled	(b) Enakost	(c) Področje	(d) Insert	(e) Delete
(1) Neureje. datoteka	**BD**	**0.5BD**	**BD**	**2D**	**Iskanje +D**
(2) Sortirana datoteka	**BD**	**Dlog$_2$B**	**D(log$_2$ B + #str z iskan. zapisi)**	**Iskanje +BD**	**Iskanje +BD**
(3) Povez. B+ indeks	**1.5BD**	**Dlog$_F$ 1.5B**	**D(log$_F$ 1.5B +#str z isk. zapisi)**	**Iskanje +D**	**Iskanje +D**
(4) Nepov. B+ indeks	**BD(R+0.15)**	**D(1+ log$_F$ 0.15B)**	**D(log$_F$ 0.15B +#str z isk. zapisi)**	**Iskanje +2D**	**Iskanje +2D**
(5) Razprš. Indeks	**BD(R+0.125)**	**2D**	**D(B*0.125+ #str z isk. z.)**	**Iskanje +2D**	**Iskanje +2D**

☛*Ocene slonijo na precej predpostavkah!*

Delovna obremenitev

- *"Delovna obremenitev"* vsebuje vse poizvedbe, ki jih dana aplikacija generira.
- Splošno:
  - Študij tipov UPDATE/INSERT/DELETE/UPDATE in atributov, ki sodelujejo.
- Za vsako selekcijo v *delovni obremenitvi:*
  - Katere relacije dosegajo?
  - Katere atribute izbere?
  - Kateri atributi sodelujejo pri pogojih za izbiranje in stike? Kako selektivni so pogoji?
- Za vsak update v delovni obremenitvi:
  - Kateri atributi sodelujejo v pogoju izbire ali stika? Kako selektivni so pogoji?

Izbor indeksov

- Katere indekse naj kreiramo?
  - Katere relacije naj imajo indekse?
  - Katera polja naj bodo iskalni ključi?
  - Naj zgradimo več indeksov na eni relaciji?
- Kakšen indeks izbrati?
  - Primarni – Sekundarni
  - Povezan – Nepovezan
  - Razpršilni - Drevo

Izbor indeksov

- Pristop:
  - Izberi najbolj pomembne poizvedbe.
  - Kateri indeksi lahko pohitrijo posamezno poizvedbo?
  - Ali izbrani indeksi pokrijejo vse poizvedbe?
- Ozadje:
  - Potrebno je poznavanje notranjih struktur SUPB.
  - Potrebno je poznavanje plana evaluacije poizvedbe.
- Vpliv indeksov na popravljanje tabel:
  - Trade-off: *Hitrost izvajanja vprašanj* VS *počasnost vnosov*.
- Indeksi potrebujejo precej diskovnega prostora.

Izbor indeksov

- Izbor atributov za iskalne ključe:
  - Atributi v pogoju izbire (where s.) so kandidati za iskalne ključe.
- Splošno:
  - Izberi indekse, ki pohitrijo čim večje število poizvedb.
  - Ker je lahko samo en indeks povezan z relacijo izberi povezanost indeksa na osnovi najpomembnejših vprašanj.

Smernice za izbor indeksov

- Izbiranje z enakostjo sugerira uporabo razpršilnih indeksov.
- Izbiranje po področju sugerira uporabo drevesnih indeksov.
- Povezani indeksi so dobri za iskanje po področju in iskanje z enakostjo v primeru velikega števila dvojnikov.
- Indeks z več atributi v iskalnem ključu je dobro izbrati pri velikem številu pogojev v iskalnih zapisih.
- Vrstni red atributov pri indeksu z več atributi je pomemben.
- Upoštevaj možnost evaluacije poizvedbe samo z indeksi.
- Povezanost indeksa s tabelo ni pomembna pri poizvedbah samo z indeksi.

- Več na predavanju 10 – Fizično načrtovanje relacijskih PB

Povzetek

- Obstaja veliko alternativnih dat. organizacij; vsaka je primerna za določene situacije.
- Indeks je zbirka podatkovnih referenc in struktura in/ali algoritem za hitro iskanje pod. referenc.
- Podatkovne reference so lahko: podatkovni zapisi, <ključ, rid> ali <ključ, rid-list>
- Tabela ima lahko več indeksov z različnimi iskalnimi ključi.

Povzetek

- Indeks je lahko: povezan/nepovezan, primarni/ sekundarni, razpršilni/drevesni.
- Pri pogostih selekcijah je priporočljivo zgraditi indeks ali datoteko urediti.
  - Razpršilni indeksi so dobri za iskanje z enakostjo.
  - Urejene datoteke in drevesni indeksi so dobre za iskanje po področju.
  - Datoteke so redko sortirane v praksi; B+ indeks je boljši.

Povzetek

- Poznavanje narave delovne obremenitve aplikacije je pomembno pri načrtovanju podatkovne baze.
  - Katere so pomembne poizvedbe?
  - Kateri atributi sodelujejo ?
- Indeksi morajo biti kreirani za pohitritev pomembnih poizvedb.
  - Indeksi upočasnijo dodajanje, brisanje in popravljanje zapisov.
  - Izberi indekse, ki izboljšajo izvajanje več poizvedb.
  - Zgradi indekse, ki podpirajo evaluacijo samo z indeksi.
  - Indeks povezan z datoteko je lahko en sam.
  - Urejenost polj v sestavljenem indeksu je pomembno.

Datoteke in indeksi

1. Koliko blokov/strani zasede datoteka z določeno organizacijo?

   B naj bo število strani v datoteki:

   - neurejena datoteka (kopica) s seznamom
   - neurejena datoteka (kopica) z direktorijem
   - urejena datoteka
   - datoteka s povezanim drevesnim indeksom
   - datoteka z nepovezanim drevesnim indeksom
   - datoteka s povezanim razpršilnim indeksom
   - datoteka z nepovezanim razpršilnim indeksom

2. Koliko blokov/strani je potrebno prebrati ob danem vprašanju?

   Predpostavimo, da samo 1 zapis ustreza enakosti:

datotečna organizacija	enakost (=)	področje (<,>)
neurejena datoteka		
urejena datoteka		
B+ drevo povezano		
B+ drevo nepovezano		
razpršilni indeks povezan		
razpršilni indeks nepovezan		

3. Oceni porabljen čas pri dani akciji

   Branje vseh zapisov
   Iskanje enakosti
   Iskanje območja
   Vnašanje
   Brisanje

   za naslednje datotečne organizacije

   Neurejena Datoteka (Kopica),
   Sortirana (urejena) datoteka,
   Povezana datoteka B+,
   Kopica z nepovezanim B+,
   Kopica z nepovezanim razpršilnim indeksom

   Predpostavke
   B je število strani v datoteki (vse strani polne)
   R je število zapisov na strani
   D je čas branja in pisanja strani iz diska in nazaj na disk
     = 15 ms
   C je čas procesiranja zapisa na strani (npr. primerjanje zapisa z vrednostjo)
     = 100 nano sekund
   H je čas za izvedbo razpršilen (Hash) funkcije nad zapisom
     = 100 nano sekund
   F je povprečno število listov na vozlišče v drevesu
     = 100

Pregled evaluacije poizvedb

Iztok Savnik, FAMNIT

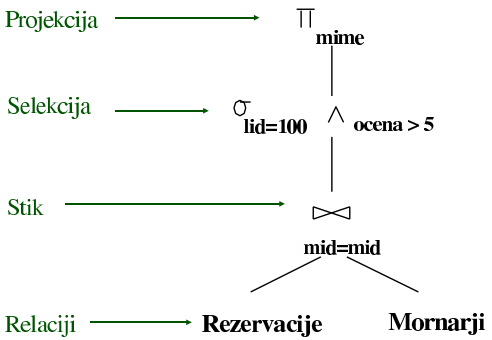
Vsebina

- Pregled evaluacije vprašanj v relacijskem SUPB
- Podatki, podatki o podatkih in sist. katalogi
- SQL vprašanja prevedemo v razširjeno RA
- Razširjena RA vsebuje oznake algoritmov
- Iskanje izraza RA, ki se najhitreje evaluira = optimizacija poizvedb
- Algoritmi za evaluacijo relacijskih vprašanj
- Tipični relacijski optimizator poizvedb

Program v SUPB

- *Kaj je program v SUPB?*
  - Drevo operacij relacijske algebre.
  - Izberemo lahko algoritem za vsako posamezno operacijo.
- Drevo iteratorjev.
  - Vsaka operacija je implementirana z iteratorjem.
  - Vmesnik iteratorja: open(), next(), close()
  - Klic operacije next() sproži evaluacijo operacije next() na otrocih operacije; nad rezultati izvede svojo kodo.
- Dobimo drevesno strukturiran cevovod po katerem se pretakajo n-terice.

Primer drevesa operacij



Evaluacija drevesa operacij

- Kako evaluiramo drevo RA operacij?
  - Logična in fizična optimizacija. Nivoji se zelo prepletajo!
  - Implementacija rel.optimozatorja = 40-50 človek/let
  - Evaluator dreves RA
- ❖ Logična optimizacija
  - Iščemo izraz relacijske algebre, ki se najhitreje izvede.
  - Da bi ocenili izraz relacijske algebre je potrebno določiti fizično implementacijo drevesa RA.
- ❖ Fizična optimizacija.
  - Iščemo optimalne algoritme za izvajanje RA operacij.
  - Fizična optimizacija se prepleta z logično optimizacijo.
- ❖ Evaluacija drevesa RA
  - Algoritmi za izvajanje relacijskih operacij.

Uvod v algoritme RA

- Ogledali si bomo algoritme za izvajanje relacijskih operacij
- Nobeden algoritem ni superioren – vsak se uporablja za določene vrste relacij
- Fizična optimizacija je izbor konkretnih metod dostopa in algoritmov za RA operacije

Pogosto uporabljene tehnike

- 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: Velikokrat koristi razdeliti problem na več enakih delov – s tem zamenjamo izvajanje časovno potratnih operacij z podobnimi operacijami nad manjšim številom n-teric.

* *Bodimo pozorni na te tehnike pri opisu izvajanja operacij!*

Statistike in katalogi

- Podatki o relacijah in indeksih v podatkovni bazi.
- ***Katalogi*** običajno vsebujejo:
  - # n-teric za vsako relacijo.
  - # strani za vsako relacijo.
  - # št. različnih vrednosti za ključ indeksa.
  - # št. strani za vsak indeks.
  - višina indeksa, najnižja/najvišja vrednost ključa za vsak drevesni indeks.
- Katalogi se ažurirajo periodično.
  - Sprotno ažuriranje je preveč drago.
  - Ker je veliko približkov je majhna nekonsistenca OK.
- Včasih so shranjene bolj podrobne vrednosti.
  - Npr. število posamezni vrednosti za atribut relacije.

Metode dostopa

- Metoda dostopa je alg. za iskanje n-teric relacije:
  - Pregled datoteke (file scan).
  - Dostop z indeksom, ki se ujema z atributi v pogoju izbire.
- Drevesni indeks:
  - se ujema z atributi v pogoju izbire v primeru da so atributi prefiks iskalnega ključa.
  - Primer: drevesni indeks nad <*a, b, c*> se ujema z iskalnim pogojem *a=5 AND "b=3"* in z *"a=5 AND b>6"*, toda ne tudi z *"b=3"*.
- Razpršilni indeks:
  - se ujema z atributi v pogoju oblike *atribut=vrednost* za vsak atribut iz iskalnega ključa indeksa.
  - Primer: razpršilni indeks nad <*a, b, c*> se ujema s pogojem *"a=5 AND b=3 AND c=5"*; ne ujema pa se s pogoji *"b=3"*, *"a=5 AND b=3"*, ali *"a>5 AND b=3 AND c=5"*.

Selektivnost metode dostopa

- Koliko strani prebere dana metoda dostopa?
- Najbolj selektivna metoda dostopa izbere najmanj n-teric
- Selektivnost metode dostopa je odvisna od osnovnih konjunkcij in selektivnosti pogojev
- Vsaka konjunkcija je filter nad tabelo
- Redukcijski faktor metode dostopa = delež n-teric, ki so izbrane z metodo dostopa

Selektivnost metode dostopa

- Primer 1:
  - Imamo razpršilni indeks H na <mime,lid,mid> za tabelo Mornarji
  - Imamo pogoj „mime='Tone' and lid=10 and mid=19"
  - Katalog: Nkeys(H), Npages(Mornarji)
  - Selektivnost: Npages(Mornarji) * 1/Nkeys(H)

Selektivnost metode dostopa

- Primer 2:
  - Redukcijski faktor a področja
  - Pogoj: dan > 12/12/09
  - Dano je B+ drevo T na atributu dan.
  - Redukcijski faktor je
    (High(T) - vrednost)/(High(T)-Low(T))

Algoritmi za implementacijo RA

- Ogledali si bomo osnovne algoritme za relacijske operacije
- Upoštevamo samo V/I ceno v št. prebranih blokov

Selekcija

- Prevajanje pogoja izbire v KNO
- Iščemo najbolj selektivne pogoje
- Najmanjše število prenosov I/O blokov
- Uporaba indeksov pri metodah dostopa
- Uporaba samih indeksov pri selekciji

Projekcija

- Pregled vseh n-teric relacije in izbor atributov
- Najdražja operacija je odstranitev duplikatov
- Uporaba sortiranja
- Uporaba razpršilnih funkcij

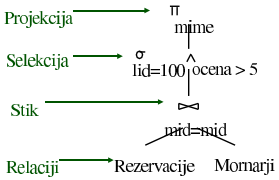
Stik

- Stik z vgnezdeno zanko
- Stik z drevesnim indeksom
- Stik z vgnezdeno zanko po blokih
- Stik z zlivanjem
- Stik z razpršilnim indeksom

Uvod v optimizacijo poizvedb

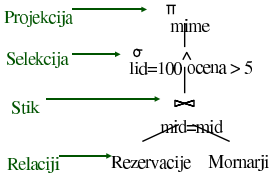
- Ogledali si bomo optimizator System R
  - 1972 (IBM).
  - Najbolj razširjen algoritem za optimizacijo.
  - Deluje dobro za < 10 stikov.
- Prostor planov izvajanja
  - Levo-v-globino, grmičevje
- Ekvivalence relacijske algebre
- Optimizacijski algoritmi
- Ocena plana izvajanja

Kaj optimiziramo?



- Opisni parametri
  - Pretok
    - Št. pretočenih n-teric naj bo čim manjše. Po nepotrebnem naj nebi brali n-terice. Optimizacija poizvedb -- naslednje predavanje!
  - Velikost n-teric
    - N-terice naj bodo čim manjše. Navzgor ne prenašamo atributov, ki jih ne potrebujemo v višjem delu drevesa.
  - Izvajanje iteratorja
    - Optimalen algoritem za izvajanje operacij.Stik z indeksom, zlivanjem, …
- Kvantitativni parametri
  - Število prebranih blokov iz diska!
  - Zanemarimo vrsto parametrov.

Optimizacija poizvedb



- Kako optimiziramo izvajanje?
  - Preiskovanje prostora rešitev – ekvivalentnih izrazov.
  - Uporaba ekvivalenc RA.
  - Izračun ocene plana je zelo pomemben.
  - Pri izračunu je potrebno vedeti kako se bo izvršila konkretna operacija.
  - Ocena plana = št. prebranih/napisanih blokov.
- Relacijski optimizator
  - Konreten primer relacijskega optimizatorja si bomo ogledali na naslednjem predavanju.
  - Optimizator System-R (IBM).
  - Uporaba dinamičnega programiranja.
- Evaluacija poizvedb:
  - Izbor optimalnega algoritma za izvajanje posameznih operacij?
  - Fiični nivo optimizacije.
  - Pregled algoritmov za izvajanje operacij!

Cena planov izvajanja

- Za vsak plan je potrebno narediti oceno.
- ❖ Potrebno je narediti oceno hitrosti izvajanja vsake operacije v drevesu.
  - Odvisna je od kardinalnosti vhodnih relacij.
  - Pogledali smo si že ocene za nekatere operacije, npr. pregled tabele, indeksni dostop, itd.
- ❖ Potrebno je narediti oceno velikosti rezultata vsake posamezne operacije drevesa!
  - Uporabi podatke o vhodnih relacijah.
  - Za selekcije in stike predpostavi neodvisnost pogojev.

Ocena velikosti rezultatov operacij

- 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.

Povzetek

- Obstaja veliko različnih alternativnih algoritmov za evaluacijo relacijskih operacij.
- Poizvedba se ovrednoti s prevedbo v drevo in evaluacijo operacij drevesa.
- Uporabnik mora razumeti optimizacijo poizvedb, da bi lahko razumel vpliv načrtovalskih odločitev na izvajanje poizvedb aplikacije.
  - Relacijski SUPB vsebujejo ukaze za izpis plana poizvedb.
- Dva dela optimizacije vprašanj:
  - Pregledati je potrebno vse alternativne plane izvajanja poizvedbe.
    - Iskalni prostor je potrebno omejiti zaradi velikega števila ekvivalentnih poizvedb.
  - Potrebno je narediti oceno izvajanja vsakega plana poizvedbe.
    - Oceniti je potrebno velikost rezultatov ter hitrost izvajanja operacij.
    - *Ključni elementi*: Statistike, indeksi, implementacije operacij.

Evaluacija relacijskih operacij

Iztok Savnik, FAMNIT

Implementacija RA

- Ogledali si bomo implementacijo:
  - Selekcija (σ)
  - Projekcija (π)
  - Stik (⋈)
  - Razlika (−)
  - Unija (∪)
  - Agregacijske op. (SUM, MIN , itd.) in GROUP-BY
- Operacije vračajo relacije ⇒ lahko jih sestavljamo!

Shema za primere

Mornarji (*sid*: integer, *sname*: string, *rating*: integer, *age*: real)
Rezervacije (*sid*: integer, *bid*: integer, *day*: dates, *rname*: string)

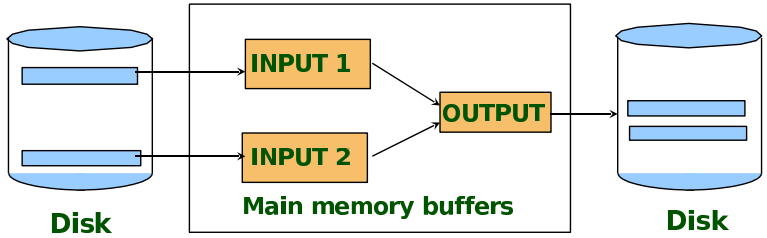
- Podobno kot stara shema; dodan *rname*
- Rezervacije:
  - Vsaka n-terica ima 40 zlogov, 100 n-teric na stran, 1000 strani.
- Mornarji:
  - Vsaka n-terica ima 50 zlogov, 80 n-teric na stran, 500 strani.

Zunanje sortiranje z zlivanjem

- Ne potrebujemo veliko pomnilnika za sortiranje velikih datotek
- Vmesni pomnilnik (izravnalnik) SUPB uporabimo za pohitritev algoritma
- Učinkovit algoritem:
  - N strani datoteke, B strani izravnalnika:
  - Cena: $2*N*(1+\log_{B-1}(N/B))$

Dvosmerno sortiranje: 3 izravnalniki

- Prehod 1: Preberi stran, jo sortiraj, jo napiši.
  - Uporabimo samo eno stran izravnalnika
- Prehodi 2, 3, …, itd.:
  - Uporabio tri strani izravnalnika.



Dvosmerno zunanje sortiranje z zlivanjem

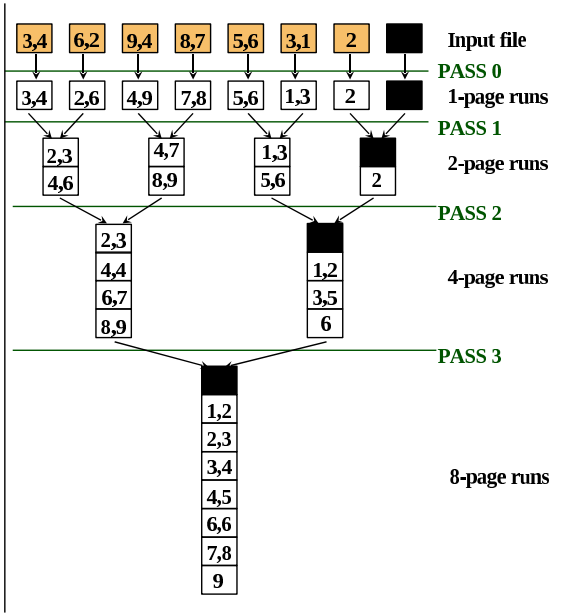
- V vsakem prehodu preberemo + zapišemo vsako stran datoteke
- N strani datoteke => število prehodov

$$=\lceil \log_2 N \rceil + 1$$

- Celotna cena:

$$2N\left(\lceil \log_2 N \rceil + 1\right)$$

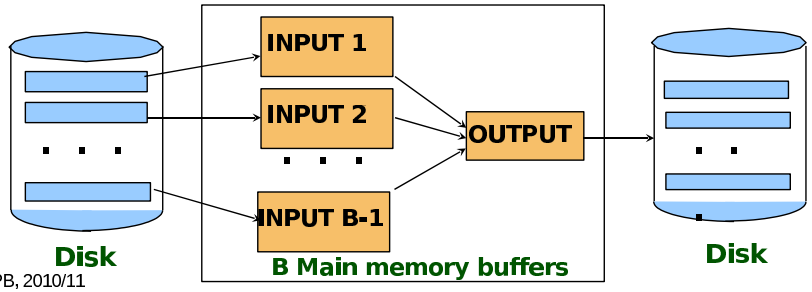
- *Ideja:*
  - *Deli in vladaj:*
  - sortiraj pod-datoteke in združi tokova



Splošno zunanje sortiranje z zlivanjem

 Imamo več kot 3 strani izravnalnika. Kako jih lahko uporabimo?

- Za sortiranje datoteke z N stranmi in pri uporabi *B* strani izravnalnika:
  - Prehod 0: uporabi *B* strani izravnalnika. Naredi $\lceil N/B \rceil$ urejenih vrst dolžine *B* strani.
  - Prehod 2, …, itd.: zlij *B-1* vrst.



Cena zunanjega sortiranja z zlivanjem

- Število prehodov: $1+\lceil \log_{B-1} \lceil N/B \rceil \rceil$
- Cena = 2N * (# prehodov)
- Npr., 5 strani izravnalnika, sortiramo dat. velikosti 108 strani:
  - Prehod 0: $\lceil 108/5 \rceil$ = 22 sortiranih vrst po 5 strani (zadnja vrsta ima samo 3 strani)
  - Prehod 1: $\lceil 22/4 \rceil$ = 6 sortiranih vrst po 20 strani (zadnja vrsta ima 8 strani)
  - Prehod 2: 2 sortirani vrsti, 80 strani in 28 strani
  - Prehod 3: Sortirana datoteka velikosti 108 strani

Selekcija

(dan<8/9/94 AND mime='Janez') OR lid=5 OR mid=3

- Pogoj izbire je najprej preveden v *konjuktivno normalno obliko* (KNO):
  (dan<8/9/94 OR lid=5 OR mid=3) AND (mime='Janez' OR lid=5 OR mid=3)
- Obravnavamo samo primere brez OR; knjiga predstavlja tudi splošne primere.

Osnovni pristop

- 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:
  - Izbira n-teric z najbolj selektivnimi potmi optimizira število izbranih n-teric
  - Preostali pogoji dodatno izberejo podmnožico n-teric izbranih z najbolj 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*".

Uporaba indeksov za selekcijo

- Cena je odvisna od #izbranih zapisov in od povezanosti indeksa:
  - Cena iskanja izbranih podatkovnih vpisov (tipično majhno)
  - Cena branja zapisov (je lahko velika brez povezanosti indeksa).
  - Primer: če predpostavimo enakomerno porazdelitev potem 10% zapisov ustreza pogoju (100 strani, 10000 zapisov). Z povezanim indeksom je cena nekaj več kot 100 V/I; če pa ni povezan pa do 10000 V/I!
- *Pomembna izboljšava za nepovezane indekse*:
  1. Poišči izbrane podatkovne vpise
  2. Sortiranj rid-je podatkovnih zapisov, ki se bodo brali
  3. Preberi rid-je po vrsti. To zagotavlja, da je vsaka podatkovna stran prebrana samo enkrat.
  - # takšnih strani je verjetno večje kot pri povezanem indeksu

Presek rid-jev

- Drugi pristop: če imamo na razpolago dva ali več indeksov z alternativama (2) ali (3) za podatkovne vpise:
  - Preberi množice rid-jev podatkovnh zapisov z uporabo primernega indeksa.
  - Potem naredi *presek* poiskanih množic rid-jev.
  - Preberi zapise in uporabi preostale pogoje izbire.
  - Poglejmo si pogoj *day<8/9/94 AND bid=5 AND sid=3.*
    - Če imamo B+ drevo na atributu *day* in razpršilni indeks na *sid*, oba uporabljata alternativo (2).
    - Poiščemo lahko:
      - rid-je zapisov, ki zadoščajo *day<8/9/94,*
      - *rid-je zapisov, ki zadoščajo sid=3*,
      - naredimo presek in preverimo *bid=5.*

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 odstranijo duplikate, če ni specificirana ključna beseda DISTINCT v stavku SELECT.
- Sortiranje:
  - Sortiraj po <mid, lid> in odstrani duplikate.
  - Optimizacija: odstrani nepotrebne podatke med sortiranjem.

Projekcija

```
SELECT   DISTINCT
         R.sid, R.bid
FROM     Reserves 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: $M*\log_2 M$
    - 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 duplikate.
  - 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.

Diskusija

- Uporaba sortiranja je standardna rešitev; boljše obravnavanje “skew” in sortiran rezultat
- Če indeks vsebuje vse potrebne atribute potem lahko uporabljamo samo indeks
  - Apliciranje projekcije samo na podatkovnih vpisih.
- Če urejen (npr. drevo) indeks vsebuje vse potrebne atribute že v predponi iskalnega ključa je izvajanje še hitrejše:
  - Preberemo podatkovne vpise (indeksni pregled) po vrsti, spustimo neželjene atribute in primerjamo sosedne vpise za izločitev duplikatov.

Implementacija stika

- Imamo več algoritmov:
  1. Stik z vgnezdeno zanko
  2. Stik z drevesnim indeksom
  3. Stik z vgnezdeno zanko po blokih
  4. Stik z zlivanjem
  5. Stik z razpršilnim indeksom

Stik po enakosti enega atributa

```
SELECT  *
FROM    Reserves R1, Sailors S1
WHERE   R1.sid=S1.sid
```

- Algebra: R ⋈ S.
  - Optimizirat je potrebno previdno! R x S je velika relacija.
- Predpostavke:
  - M strani R, p_R n-teric na stran
  - N strani v S, p_S n-teric na stran
  - R = Rezervacije, S = Mornarji
- Bolj kompliciranе stike bomo obravnavali kasneje
- *Metrika cene*: # V/I blokov, ostalo ignoriramo

Stik z vgnezdeno zanko

```
foreach tuple r in R do
        foreach tuple s in S do
                if r_i == s_j  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 vgnezdeno zanko

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

- Za vsako n-terico iz zunanje relacije pregledamo celotno notranjo 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.

Stik z indeksom

```
foreach tuple r in R do
    foreach tuple s in S where r_i == s_j 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: $M + ((M*p_R) *$ cena iskanja n-teric iz S)
- Cena za preverjanje vsake 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

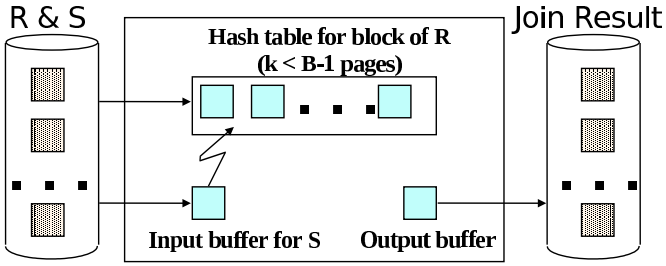
Primeri stika z indeksom

- Razpršilni indeks (Alt. 2) na *sid* 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 *sid* 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.

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
```



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.vrsto 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 velik zun.vrsto za Mornarje kot zunanja relacija:
  - Cena pregleda S je 500 V/I; skupaj 5 zun.vrst.
  - Za vsako zun.vrsto S, pregledamo Rezervacije; 5*1000 V/I.
  - Cena = 500 + 5*1000 = 5500 V/I

Primer uporabe Sortiraj-Zlij

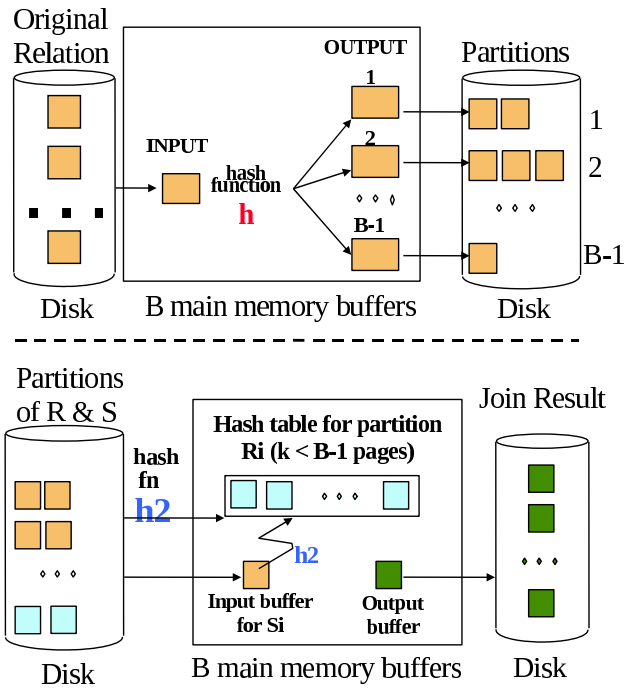
sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

- Cena: M log M + N log N + (M+N)
  - Cena pregleda, M+N, lahko je M*N (ni preveč verjetno!)
- Z uporabo 35, 100 ali 300 strani vmesnika lahko tabele Reserves in Sailors sortiramo z 2 prehodi;
- Celotna cena stika: 4*1000+4*500+1000+500=7500 V/I.

Stik z razpršilnim indeksom

- Porazdeli obe relaciji z razpršilno funkcijo **h**: n-terice particije i iz R se bodo ujele z n-tericami particije i iz S.
- Preberi particijo R, jo porazdeli z **h2 (<> h!)**. Preglej ustrezno 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-partiticijo povežemo z ustrezno S-partiticijo.

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 bistvno razlikujejo.
  - Stik z razpršilno funkcijo lahko zelo dobro paraleliziramo.
  - Stik z zlivanjem je manj občutljiv na "skewed" podatke; rezultat je sortiran.

Splošni pogoji stika

- Enakost večih atributov (npr., *R.sid=S.sid* AND *R.rname=S.sname*):
  - Za vgnez.zanko z indeksom kreiraj indeks na *<sid, sname>* (če je S notranja rel.); ali uporabljaj obstoječe indekse na *sid* ali *sname*.
  - Za zlivanje in razpršilni stik sortiraj/porazdeli s kombinacijo dveh atributov stika.
- Pogoji neenakosti (npr., *R.rname < S.sname*):
  - Za vgnezdeno zanko z indeksom uporabi (povezano!) B+ drevo.
  - Stik z zlivanjem in razpršilni stik ni uporaben.
  - Vgnezdena zanka po blokih bo zelo verjetno dala najboljše rezultate.

Operacije nad množicami

- Presek in kartezijski produkt sta posebna primera stika.
- Unija in razlika se implementirata podobno.
- Unija s sortiranjem:
  - Sortiramo obe relaciji (po vseh atributih).
  - Pregled sortiranih relacij in zlivanje
  - *Alternativa*: Zlij vrste iz prehoda 0 iz obeh ralacij.
- Unija z razpršilnim indeksom:
  - Porazdeli R in S z uporabo razpršilne funkcije *h*.
  - Za vsako S-particijo izgradi razpršilno tabelo v spominu (z uporabo h2), preglej ustrezno R-particijo in dodaj n-terice v tabelo medtem, ko se izloča duplikate.

Agregacijske operacije(AVG,MIN,itd.)

- Brez grupiranja
  - V splošnem zahteva pregled relacije
  - Pregled je možen samo z indeksom, če iskalni ključ vsebuje vse atribute v SELECT ali WHERE stavku.
- Z grupiranjem:
  - Sortiraj po group-by atributih, potem preglej relacijo in izračunaj agregacuijske funkcije za vsako skupino.
  - Podoben pristop je osnovan na porazdelitvi (razpršitvi) po group-by atributih.
  - Z drevesnim indeksom, ki vsebuje atribute v SELECT, WHERE in GROUP BY stavkih lahko naredimo pregled samo na osnovi indeksa
    - Če group-by atributi tvorijo predpono iskalnega ključa, potem lahko preberemo vpise/zapise v vrstnem redu določenem z group-by.

Vpliv izravnalnika

- Če se več operacij izvaja vzporedno potem je ocenjevanje št. prostih strani v izravnalniku samo predvidevanje.
- Ponavljajoč vzorec dostopa mora biti usklajen s strategijo zamenjave strani v izravnalniku.
  - Notranja relacija se pregleduje ciklično pri enostavni vgnezdeni zanki.
  - Če imamo zadosti strani, da hranimo notranjo relacijo potem zamenjalna strategija ne igra posebne vloge.
  - Sicer pa je najboljša strategija MRU; LRU je najslabša (sekvenčno prelivanje).
    - Ali je zamenjalna strategija pomembna pri vgnezdeni zanki po blokih?
    - Kaj pa pri vgnezdeni zanki z indeksom? Sortiranje-zlivanje stik?

Pregled

- Relacijski SUPB:
  - *Poizvedbe so sestavljene iz nekaj osnovnih operacij*
  - Implementacija teh operacij je skrbno uglašena
  - Podrobnosti pri implementaciji so pomembne!
- Obstaja več različnih alternativnih implementacij za vsako operacijo; ne obstaja superiorna tehnika za večino operatcij.
- Potrebno je pregledati alternative pri implementaciji vsake operacije v poizvedbi
  - Izberemo najboljšo strategijo na osnovi statističnih podatkov o tabelah.
  - Izbor ene operacije je del celotnega dela optimizacije celotne poizvedbe.

```
PILOTI                                  LETI
   IdP             int(11),                IdLT          int(11),
   Priimek         varchar(50),            IdP           int(11),
   Ime             varchar(50),            IdL           int(11),
   Emso            varchar(13),            IdLEVzleta    int(11),
   Naslov          varchar(250),           DtVzleta      date,
   DtZaposlitve    date,                   CasVzleta     time,
   LetaDelIzkusenj varchar(20),            IdLEPristanka int(11),
   StLetov         int(11),                DtPristanka   date,
   StUrLetenja     int(11),                CasPristanka  time
   IdLD            int(11)


REZERVACIJE
   IdR      int(11),
   Stevilka int(11),
   Cena     float,
   Priimek  varchar(50),
   Ime      varchar(50),
   Naslov   varchar(250),
   Telefon  varchar(20),
   StSedeza int(11),
   Razred   int(11),
   IdLT     int(11)
```

Pri tem upoštevaj naslednje informacije:

PILOTI: 10 zapisov na stran, vsak zapis predstavlja enega samega pilota,
40 strani,
relacija hrani 100 različnih vrednosti za StUrLetenja in
piloti so enakomerno porazdeljeni med te vrednosti,
pri tem pa je od takih vrednosti, katerih vrednost je nad 500, 70 različnih.

LETI: 20 zapisov na stran, 150 strani

REZERVACIJE: 50 zapisov na stran,
30000 strani,
različnih cen rezervacij je 150 in takih, ki so večje od 300 je 70

Za vsako izvajanje poizvedbe imamo na voljo 10.000 strani v vmesnem pomnilniku (SUBP).

1. Za dani poizvedbi sestavi drevo operacij in oceni plan njihovega izvajanja (koliko strani se bere iz diska):

   a) SELECT Priimek, Ime FROM PILOTI
   b) SELECT Priimek, Ime FROM PILOTI WHERE StUrLetenja>500

2. Za koliko se ocena plana izvajanja za drugo poizvedbo iz prejšnjega primera spremeni, če pri svojem izvajanju uporablja drevesni indeks višine 2 na polju StUrLetenja.

3. Napiši odgovore na naslednja vprašanja s povpraševalnim jezikom SQL:

   a) Poišči vse IdL (identifikator letal), pri katerih je cena rezervacije njihovega leta presegala 300 €.
   b) Poišči vse IdL (identifikator letal), pri katerih je cena rezervacije njihovega leta presegala 300 € in je let pilotiral pilot z IdP=50.

4. Za koliko se ocena plana izvajanja za drugo poizvedbo iz prejšnjega primera spremeni, če pri svojem izvajanju uporablja:

- 2-nivojski drevesni indeks na polju IdP v relaciji LETI
- razpršilni indeks na polju Cena in
- razpršilni indeks na polju IdLT v relaciji REZERVACIJE

5. Za dano shemo ocenimo plan izvajanja naslednje poizvedbe:

```
SELECT U.Ime
FROM Ucitelj U, Predava P
WHERE U.Id = P.UciteljId AND
      U. OddelekId = 'RIN' AND
      P.Semester = 'Zimski 2010'
```

Shema:

```
UCITELJ(Id, ImeName, OddelekId)
PREDAVA(UciteljId, PredmetSifra, Semester)
```

Pri tem upoštevamo naslednje informacije:

UCITELJ: 200 strani, 1000 zapisov,
50 različnih vrednosti za oddelek,
drevesni indeks na polju OddelekId,
razpršilni indeks na polju Id

PREDAVA: 1000 strani, 10.000 zapisov,
4 različne vrednosti za semester,
drevesni indeks na polju Semester,
razpršilni indeks na polju UciteljId,
predavanja so enakomkerno porazdeljena med ucitelji (10 predavanj / ucitelja)

Na voljo imamo 52 prostih strani v vmesnem pomnilniku.

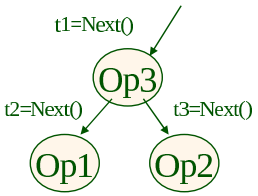
Optimizacija poizvedb

Iztok Savnik, FAMNIT

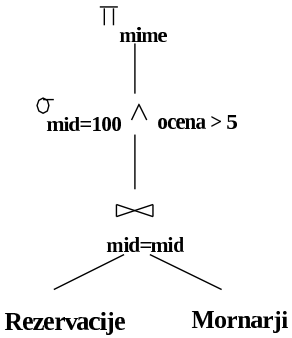
Osnove optimizacije poizvedb

- Osnovni pojmi:
  - Plan izvajanja poizvedbe.
    - Prevajanje SQL v RA
    - Ocenitvena funkcija.
    - Algoritmi za izvajanje operacij RA.
    - Prostor rešitev.
    - Algoritem za iskanje optimalnega plana.
- System R:
  - Najbolj razširjen.
  - Dela v redu za < 15 stikov.

Plan izvajanja poizvedbe



- Implementacija operacije:
  - Iterator: popularna + objektna tehnika.
    - Implementacija iteratorja: `open(),next(),close()`.
    - Metoda `next()` vrne eno n-terico ali `eof`.
- Drevo iteratorjev:
  - Vsaka operacija je lahko implementirana z množico algoritmov.
  - Plan izvajanja vsebuje tudi algoritem za izvedbo operacije.



Prevajanje SQL v RA

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

Dekompozicija vprašanj v bloke

- Celoten SQL stavek se razdeli v bloke
- Zaključeni SQL bloki se obravnavajo kot procedure
- Posledica:
  - vgnezdeni 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

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 plana

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

Ocena za plane nad eno relacijo

- 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)*

Shema za zgled

Mornarji (*mid*: integer, *mime*: string, *ocena*: integer, *star*: real)
Rezervacije (*mid*: integer, *lid*: integer, *dan*: dates, *rime*: string)

- Podobno kot stara shema; *rime* dodana.
- Rezervacije:
  - Velikost zapisa = 40 zlogov, 100 zapisov na stran, 1000 strani.
- Mornarji:
  - Velikost zapisa = 50 zlogov, 80 zapisov na stran, 500 strani.

Primer

```
SELECT M.mid
FROM Mornarji M
WHERE M.ocena=8
```

- Imamo indeks na atributu *ocena*:
  - (1/NKljucev(I)) * NZapisov(R) = (1/10) * 40000 = 4000 zapisov.
  - Povezan indeks:
    (1/NKljučev(I)) * (NStrani(I)+NStrani(R)) = (1/10) * (50+500) = 55 prebranih strani.
  - Nepovezan indeks:
    (1/NKeys(I)) * (NStrani(I)+ NZapisov(R)) = (1/10) * (50+40000) = 4005 prebranih strani.
- Imamo indeks na *sid*:
  - Prebrati je potrebno vse n-terice/strani.
  - Povezan indeks = 50+500 strani; nepovezan indeks = 50+40000 strani.
  - Slabo !
- Sekvenčni pregled:
  - Vse strani tabele = 500 strani.

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.

Ocene selektivnosti pogojev

- *Ocene posameznih tipov pogojev:*
  - Predpostavka, da so pogoji med seboj neodvisni.
  - Pogoj *atr=vrednost* ima SP = *1 / Nključev(I); I* je indeks na *atr.*
  - Pogoj *atr1=atr2* ime SP = *1 / MAX(Nključev(I1),Nključev(I2))*
  - pogoj *atr>vrednost* ime SP = *(High(I)-value)/(High(I)-Low(I))*
- Selektivnost pogoja odseva vpliv pogoja na zmanjšanje velikosti rezultata.
- *Kardinalnost rezultata*
  - Max # n-teric * produkt vseh SP.

Ocena za plane nad večimi relacijami

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

- 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.

Definicija problema

- Dana je poizvedba *q* nad podatkovno bazo *B*.
- Poišči najcenejši plan poizvedbe.
  - Algoritem za optimizacijo.
    - System R uporabi dinamično programiranje.
  - Imamo *ocenitveno funkcijo*, ki izračuna *oceno plana*.
- Ocenitvena funkcija upošteva:
  - "Pretok" po drevesu iteratorjev.
  - Število prebranih blokov iz diska.
  - CPU uporabljen za izračun operacij (mi ne upoštevamo).
  - Velikosti vmesnih rezultatov in končnega rezultata.

Prostor rešitev

- Prostor ekvivalentnih poizvedb.
- Lastnosti relacijske algebre omogočajo transformacije poizvedbe.
  - Transformirana poizvedba vrne isti rezultat.
  - Transformirana poizvedba povzroči spremembo plana.
  - Ponovno je potrebno določiti algoritme za impl. operacij.
  - Iščemo izraz, ki se najhitreje izvede in porabi najmanj prostora.
- Praksa:
  - Izogibamo se kartezijskim produktom.
  - Izogibajmo se najslabšim rešitvam.

Iskanje optimalnega plana

- Prostor je prevelik zato ga je potrebno omejiti.
- Kateri del prostora pregledamo?
  - Odvisno od algoritma za preiskovanje.
  - System R: uporablja le levo-usmerjene plane.
  - Obravnavamo samo prostor *levo-usmerjenih* planov.
  - Levo-usmerjeni plani omogočajo izvedbo *cevovoda.*
  - Ni potrebno kreirati vmesnih začasnih tabel.

Bloki poizvedbe

- SQL se prevede v množico *blokov poizvedbe.*
- Optimizirajo se bloki poizvedbe.
- Vgnezdeni bloki se obravnavajo kot klici podprogramov, ki se izvajajo enkrat za vsako n-terico.
- Poenostavljen pogled, drži v večini primerov.

```
SELECT  M.mime
FROM    Mornarji M
WHERE   M.star IN
     (SELECT  MAX (M2.star)
      FROM    Mornarji M2
      GROUP BY  M2.ocena)
```

Zunanji blok

Notranji blok

❖ Za vsak blok obravnavamo naslednje plane:
- Vse metode dostopa za vsako relacijo iz FROM stavka.
- Vse *levo-usmerjena drevesa stikov.*
  - ✓ Vse permutacije relacij povezane s stiki v levo-usmerjeno drevo.
  - ✓ Pogoje stikov lahko izračunamo sproti iz pogojev v WHERE stavku.

Ekvivalence operacij RA

- Omogočajo:
  - Naštevanje ekvivalentnih zapisov izrazov RA.
  - Različni vrstni red stikov.
  - Spuščanje selekcije in projekcije proti listom.
- Zapis izraza RA določa drevo operacij – program!
- Z pregledom vseh ekvivalentnih izrazov RA naštejemo vsa drevesa operacij.
  - Vse načine evaluacije dobimo z dodatno izbiro vseh možnih metod dostopa.
- Oglejmo si ekvivalence za operacije RA.

Ekvivalence operacij RA

- *Selekcija:* $\sigma_{c1\wedge\ldots\wedge cn}(R) \equiv \sigma_{c1}(\ldots \sigma_{cn}(R))$ (*Razcep*)

  $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$ (*Komutativnost*)

❖ *Projekcija:* $\pi_{a1}(R) \equiv \pi_{a1}(\ldots(\pi_{an}(R)))$ (*Razcep*)

❖ *Stik:* $R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T$ (*Asociativnost*)

$(R \bowtie S) \equiv (S \bowtie R)$ (*Komutativnost*)

 Dokaži: $R \bowtie (S \bowtie T) \equiv (T \bowtie R) \bowtie S$

Ekvivalence operacij RA

- Spuščanje selekcije/projekcije proti listom.

  $\delta(R \bowtie S) \equiv \delta(R) \bowtie S$, če selekcija izbira samo atribute R.

  $\Pi(R \bowtie S) \equiv \Pi(R) \bowtie S$, če projekcija ohrani atribute stika.
- Projekcija:
  - Projekcija je komutativna s selekcijo, ki uporablja atribute projekcije.
- Selekcija, ki vsebuje primerjavo atributov obeh argumentov kartezijskega produkta se prevede v stik.

Optimizacijska metoda

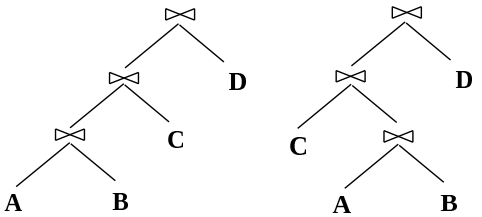
- Poenostavitev problema
  - Sami STIKI + metode dostopa
- Prostor alternativnih planov
  - Uporabljamo ocene planov
  - Uporabljamo algebraična pravila za iskanje ekvivalentnih poizvedb
  - Algoritem za pregledovanje prostora
  - Optimizator običajno pregleda del prostora
  - Izbere plan za katerega imamo najmanjšo oceno

Tipični relacijski optimizator

- Algoritem osnovan na dinamičnem programiranju
  - Optmalne plane gradimo iz optimalnih rešitev podproblemov
- Uporaba samo levo-usmerjenih planov
  - Nimamo materializacije
  - Omejimo prostor rešitev
  - Iskanje permutacij oz. uporaba dinamičnega prog.
  - Rešitve so sub-optimalne!
- Optimizator Sistema R

Poenostavitev problema

- Imamo samo dve vrsti vozlišč
- Vozlišče z metodo dostopa
  - Dostop do zapisov
  - MD z indeksom, MD s sortiranjem, MD z iteracijo, ...
  - Vsebuje (tudi) selekcijo in projekcijo
- Vozlišče s stikom
  - Različni algoritmi za izvajanje stika
  - Stik z vgnezdeno zanko, indeksom, zlivanjem, ...
  - Vsebuje (tudi) selekcijo in projekcijo



Naštevanje alternativnih planov

- Uporaba ekvivalenčnih pravil nad izrazi
  - Dobimo logično ekvivalentne izraze
  - Različna implementacija in cena
  - Samo nad levo-usmerjenimi plani!
- Dva osnovna primera
  - Plan nad eno relacijo ⟶ optimalna metoda dostopa
  - Plan nad večimi relacijami ⟶ permutacija stikov

Poizvedbe nad eno relacijo

- Kombinacije selekcije, projekcije in agregacijskih operacij
  - Ni stikov!
- Pregledajo se vse možne metode dostopa
  - Metode dostopa smo si ogledali na prejšnjih predavanjih.
  - Metode brez indeksov
  - Metode z indeksi
- Izberemo metodo z najnižjo ceno

Zgled

- Vprašanje:
  Za vsako oceno večjo od 5, izpiši oceno in število mornarjev starih 20 let, ki imajo takšno oceno, če sta za dano oceno vsaj dva takšna mornarja z različnim imenom.

```
select S.rating, count(*)
from Sailors S
where S.rating > 5 and A.age=20
groupby S.rating
having count distinct(S.sname >2);
```


Zgled

- Prepišemo vprašanje v razširjeno relacijsko algebro.
- Druga projekcija ima dodan sname, ker rabimo sname za having.

$$\Pi_{S.rating,\ count(*)}(\text{having}_{countdistinct(S.sname)\ >2}(\text{groupby}_{S.rating}(\Pi_{S.rating,S.sname}(\delta_{S.rating\ >\ 5\ and\ A.age=20}(Sailors)))))$$

Zgled

Zdaj lahko pregledamo možne plane izvajanja, ki bi jih optimizator pregledal.

Najbolj pomembna odločitev je izbor metode dostopa do Sailors.

$$\Pi_{S.rating,\ count(*)}(\ having_{countdistinct(S.sname)\ >2}(\ groupby_{S.rating}(\ \Pi_{S.rating,S.sname}(\ \delta_{S.rating\ >\ 5\ and\ A.age=20}(Sailors)))))$$

Ugotoviti je potrebno tudi cene preostalih operacij.

- Sortiranje, grupiranje, having, ...

Metode dostopa brez indeksov

- Osnovni pristop je pregled vseh zapisov tabele Sailors in sprotno filtriranje zapisov (selekcija + projekcija)
- Izraz RA: $\Pi_{S.rating,S.sname}(\delta_{S.rating\ >\ 5\ and\ A.age=20}(Sailors))$
- Rezultat selekcije in projekcije se nato uredi v skladu z atributi operacije group-by.
- Zapis rezultata se generira iz ene skupine mornarjev, ki ustreza pogojem operacije having.
  - Agregacijske funkcije se izvršijo nad skupinami.

Plani izvajanja brez indeksov

- V ceni upoštevamo:
  - Pregled Sailors
  - Izpis n-teric (rezultat selekcije in projekcije)
  - Sortiranje za implementacijo groupby
- Operacija having se izvrši on-the-fly
- Pregled Sailors = 500 V/I
- Zapis vmesnega rezultata = 20 V/I
- Sortiranje za group-by = 60 V/I
- Skupaj = 580 V/I

Plani izvajanja z indeksi

- 1. Dostop z enim indeksom
  - Če je na razpolago več indeksov
  - Vsak indeks predstavlja eno metodo dostopa
  - Izberemo indeks, ki zahteva najmanj V/I
- 2. Dostop z večimi indeksi
  - Če več indeksov tipa (2) in (3) ustreza pogoju izbire
  - Preberemo rezultate večih indeksov in naredimo presek, sortiramo izbrane id-je za optimalen dostop
  - Projekcije in dodatne selekcije lahko sledijo po branju zapisa (kot tudi grupiranje in izbiranje grup)

Plani izvajanja z indeksi

- 3. Dostop s sortiranim indeksom
  - Če seznam atributov za grupiranje ustreza sortiranem indeksu
  - Urejenost indeksa uporabimo za grupiranje
  - Vse selekcije lahko naredimo, po tem ko preberemo n-terice preko indeksa

Plani izvajanja z indeksi

- 4. Dostop samo z indeksom
  - Če imamo indeks, ki vsebuje vse atribute potrebne v vprašanju potem lahko uporabimo samo indeks
  - Vsakemu podatkovnemu vpisu ustreza podatkovni zapis, ki ga ni potrebno prebrati
  - Naredimo selekcijo, projekcijo in sortiramo končne zapise za grupiranje
  - Dostop samo z indeksom funkcionira tudi če atributi izbire niso vsebovani v atributih indeksa
  - Uporabimo dostop samo z indeksom kot običajno metodo dostopa, filtriramo n-terice in dostopamo samo do izbranih podatkovnih zapisov

Zgled

```
select S.rating, count(*)
from Sailors S
where S.rating > 5 and A.age=20
groupby S.rating
having count distinct(S.sname >2);
```

- Poglejmo spet prejšnji primer
- Predpostavimo da imamo nasl. indekse:
  - B+ drevo na rating
  - Razpršilni ind. na age
  - B+ indeks na <rating,sname,age>
- Vsi indeksi uporabljajo varianto (2)

Zgled

- 1. možnost:
  - razpršilni indeks na za pogoj S.age=20
  - Cena branja indeksnih zapisov + pod.zapisov Sailors
  - Cena odvisna od povezanosti indeksa
  - Filtriramo s pogojem S.rating>5
  - Projeciramo stolpce, ki jih potrebujemo v SELECT, GROUP BY in HAVING.
  - Zapišemo začasno relacijo
  - Začasno relacijo sortiramo po S.rating za GROUP BY
  - Eliminiramo skupine, ki ne zadoščajo HAVING

Zgled

- 2. možnost:
  - preberemo rid-je zapisov, ki zadoščajo pogoju S.rating>5
  - Preberemo rid-je zapisov S.age=20
  - Sortiramo obe množici in preberemo ustrezne zapise Sailors
  - Projeciramo atributa rating in sname ter zapišemo vmesno relacijo
  - ...

Zgled

- 3. možnost:
  - Preberemo zapise Sailors z S.rating>5 v vrstnem redu določenim z rating oz. B+ drevesom.
  - Naredimo selekcijo on-the-fly
  - Izračunamo lahko agregacijske funkcije za GROUP BY in HAVING, ker so n-terice urejene po rating.
  - ...

Zgled

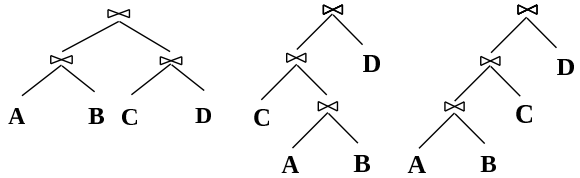
- 4. možnost:
  - Preberemo zapise <rating,sname,age> za katere velja rating>5
  - Zapise sortiramo po rating (in sname in age, kar ni pomembno za to vprašanje)
  - Izberemo age=20 on-the-fly
  - Izračunamo agregacijske funkcije za GROUP BY in HAVING
  - Ne preberemo pod.zapisov Sailors!
  - Pomembno za nepovezane indekse

Poizvedbe nad večimi relacijami

- Gledamo plane nad večimi relacijami
- Takšna vprašanja so lahko zelo zahtevna!
- Ne glede na to kakšen plan izberemo lahko ocenimo velikosti rezultatov
- Izbor vrstnega reda izvajanja stikov
  - Vpliva na kreiranje vmesnih rezultatov zelo različnih velikosti
  - Plani imajo lahko zelo različne cene!
  - Glej primere iz prejšnjega predavanja

Naštevanje levo-usmerjenih planov

- Osnovna odločitev v System R:
  - Obravnavajo 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

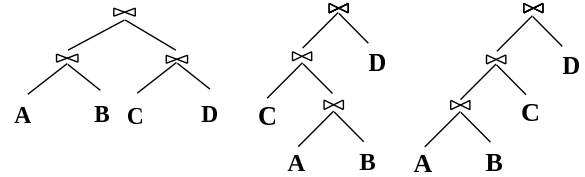


Naštevanje levo-usmerjenih planov

- Poglejmo si bloka vprašanja:
  - Optimizator sistema R pregleda vse možne stike, kjer se δ in Π spusti k relacijam

SELECT attribute list
FROM relation list
WHERE $term_1 \wedge term_2 \wedge \ldots \wedge term_n$

- 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.



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 sestavjamo iz planov za N-1 relacij tako da dodajamo se en stik.

Prehod 1:

- Naštevamo vse plane z eno relacijo iz FROM dela SELECT stavka
- Plan nad eno relacijo je delen levo-usmerjen plan
- Za vsako relacij A identificiramo pogoje izbire, ki se nanašajo samo na atribute v A. To so selekcije, ki jih je potrebno narediti pred vsemi stiki.
- Identificirati je potrebno tudi atribute A, ki niso potrebni (niso v SELECT, WHERE, …) in jih lahko izločimo po branju podatkovnega zapisa.
- Za dane pogoje je potrebno poiskati optimalno metodo dostopa.
- Poiščemo lahko najcenejše plane za vsako urejenost rezultata.
- Urejenost lahko izkoristimo kasneje npr. za GROUP BY operacijo.

Prehod 2:

- Naštevamo vse plane z dvemi relacijami.
  - Prej izračunane optimalne metode dostopa do ene relacije uporabimo kot zunanjo relacijo.
  - Vse ostale relacije uporabimo za notranje relacije.
- Recimo, da je A zunanja relacija in B notranja.Pregledamo WHERE stavek in identificiramo:
  - Selekcije nad atributi, ki se tičejo samo B se lahko izvršijo pred stikom.
  - Selekcije nad atributi, ki definirajo stik.
  - Selekcije nad atributi drugih relacij, ki se lahko izvršijo po stiku.
- Posledice:
  - Prvi dve skupini selekcij upoštevamo pri izbiri metode dostopa do B
  - Identificiramo atribute B, ki jih lahko zavržemo pred izvajanjem stika

Prehod 2:

- Še par komentarjev:
  - Generiranje n-teric z zunajnjim planom je definirano za implementacijo cevovoda.
  - Nekatere metode stika ne zahtevajo materializacije rezultata, druge zahtevajo.
  - Metoda stika lahko zahteva dol. urejenost vhoda. Npr. sortiranje z zlivanjem. Če obstoječa metoda dostopa do notranje relacije ne da željene urejenosti je treba prišteti ceno sortiranja.

Prehod 3:

- Generiramo vse plane s tremi relacijami.
- Zdaj vzamemo plane iz prehoda 2 in jih damo kot zunanjo relacijo.
- Spet dodajamo preostalo notranjo relacijo.
- Itd.

Preostali prehodi:

- Grupiranje v bloku se izvrši na koncu.
- Včasih je potrebno poskrbeti za pravilno urejenost izhoda stikov, da bi lahko izvajali grupiranje.

Kompleksnost algoritma

- Imamo *n* relacij
- 1. prehod:
  - Za vsako relacijo ⟶ n relacij
- 2. prehod:
  - Imamo optimalne plane za dostop do ene relacije
  - Za vsako podmnožico z 1 relacijo priključimo še vse preostale relacije
  - (n 1) * (n-1)
- 3. prehod:
  - Imamo optimalne plane za dostop do vseh možnih parov relacij
  - Za vse podmnožice z dvemi relacijami priključimo še preostale relacije
  - (n 2) * (n-2)
- i-ti prehod:
  - Imamo optimalne plane za dostop do vseh možnih podmnožic z i-1 relacijami
  - Za vsako pomnožico z i-1 relacijami priključi še preostalo relacijo
  - (n i-1) * (n-i)
- Skupno:
  - n+(n 1)*(n-1)+...+(n i-1)*(n-i)+...
  - >> 2^n

Zgled 1

$\Pi_{sname}(\bowtie_{sid=sid}(\delta_{bid=100}(Reserves), \delta_{rating>5}(Sailors)))$

- Poglejmo si primer optimizacije vprašanja z enim stikom
- Predpostavljamo da imamo naslednje indekse:
  1) B+ indeks na rating tabele Sailors
  2) Razpršilni indeks na sid tabele Sailors
  3) B+ indeks na bid tabele Reserves
- Predpostavimo, da lahko naredimo sekvenčni pregled na Sailors in Reserves

Zgled 1: Prehod 1

- Pregledamo tri možne metode dostopa do Sailors
  - B+ indeks se ujema s selekcijo $\delta_{rating>5}$ in zmanjša ceno dostopa do zapisov Sailors
  - Uporaba razpršilnega indeksa ali sekvenčnega pregleda je zelo verjetno dražja
  - Plan torej dostopa do Sailors preko B+ indeksa in vrača zapise urejene po rating

Zgled 1: Prehod 1

- Pregledamo dve možne metode dostopa do Reserves
  - B+ indeks na bid se ujema s selekcijo $\delta_{bid=100}$ in zmanjša ceno dostopa do zapisov Reserves
  - Sekvenčni pregled bo zelo verjetno slabša metoda
  - Dostop preko B+ indeksa je torej plan za Reserves

Zgled 1: Prehod 2

- Pogledamo najprej stik med Reserves in Sailors.
  - Potrebujemo samo zapise Sailors za katere velja $\delta_{rating>5}$ in $\delta_{sid=value}$
  - $\delta_{sid=value}$ se ujema z razpršilnim indeksom na sid tabele Sailors
  - $\delta_{rating>5}$ se ujema z B+ indeksom na rating
  - Dostop z enakostjo ima večjo selektivnost.
  - Uporabimo razpršilni indeks na sid tabele Sailors
  - Pogledamo tudi alternativne algoritme; stik z zlivanjem?
    - Urejenost po sid – ni uporabno
    - Imamo Π_{sname} - sid odpade

Zgled 1: Prehod 2

- Stik med Sailors in Reserves.
  - Sailors izberemo kot zunanjo relacijo.
  - Potrebujemo samo zapise Reserves za katere velja $\delta_{bid=100}$ in $\delta_{sid=value}$, kjer je vrednost določena z zunanjo n-terico.
  - Spet pogledamo vse metode za stik

Mornarji (mid: integer, mime: string, ocena: integer, star: real)
Rezervacije (mid: integer, lid: integer, dan: dates, rime: string)

Ocena

- Mornarji, Zapis = 50 zlogov, 80 zapisov na stran, 500 strani.
- Rezervacije, Zapis = 40 zlogov, 100 zapisov na stran, 1000 strani.
- |Rid| = 16, |int| = 4, Podatkovni vpis = 20 zlogov
- Velikost indeksov
  - B+ indeks na rating tabele Sailors
    - Št.zapisov = 80 * 500 = 40000 zapisov => 200 strani pod.vpisov
    - Višina 2, 1 ind.stran, 100% zapolnjenost, Skupno = 201 strani
  - Razpršilni indeks na sid tabele Sailors
    - Št.zapisov = 80 * 500 = 40000 zapisov => 200 strani pod.vpisov
    - 80% zapolnjenost, Skupno = 250 strani
  - B+ indeks na bid tabele Reserves
    - Št.zapisov = 100 * 1000 = 100000 zapisov => 500 strani pod.vpisov
    - Višina = 3, 4 ind.strani, 67% zapolnjenost, Skupno = 756 strani

Ocena: Prehod 1

- Sailors
  - B+ indeks na rating ($\delta_{rating>5}$)
  - Selektivnost = 1/2, preberemo 101 ind.strani + (sortirani rid-ji) 300 strani
  - Skupno = 101 + 300 = 401 strani
  - Sekvenčni pregled = 500 strani
- Reserves
  - B+ indeks na bid ($\delta_{bid=100}$)
  - 500 ladij, 100000/500 = 200 zapisov
  - Cena = (ind.strani) 2 + (listi ind.) 2 + (pod.zapisi) 200
  - Skupno = 204 strani

Ocena: Prehod 2

- OptMD(Sailors) $\bowtie_{sid=sid}$ Rel(Reserves)
  - 20.000 zapisov se ujema z $\delta_{rating>5}$(Sailors)
  - Za vsak zapis je potreben dostop do Reserves
  - Ne obstaja indeks za sid na Reserves => vgn.zanka
  - Sort. z zlivanjem je tudi slabše od opt.:
    - Zač.tabela Sailors = 20000*30=150 strani
    - Sort(Zač.tabela Sailors) = 2*3*150 = 900 strani
    - Sort(Zač.tabela Reserves) = 2*1*2 = 4 strani
    - Zlivanje = 900 + 4 + 150 + 2 = 1056 strani
- OptMD(Reserves) $\bowtie_{sid=sid}$ Rel(Sailors)
  - 200 zapisov ustreza $\delta_{bid=100}$(Reserves)
  - Imamo indeks za sid na Sailors
  - Cena = 204 + (razpr.indeks) 200 * 1.2 = 444 strani
  - Stik z zlivanjem je dražji (glej zgoraj)

Zgled 2

```
select S.sid, count(*)
from Boats B, Reserves R,  Sailors S
where R.sid=S.sid and B.bid=R.bid
and B.color='red'
group by S.sid;
```

- Vprašanje poišče število rdečih ladij rezerviranih po posameznih mornarjih.

$$\Pi_{sid,\ count(*)}(\ groupby_{S.sid}(\ \bowtie_{sid=sid}(\ \bowtie_{bid=bid}(\ \delta_{color='red'}(Boats),\ Reserves),\ Sailors)))$$

Zgled 2

```
select S.sid, count(*)
from Boats B, Reserves R,  Sailors S
where R.sid=S.sid and B.bid=R.bid
and B.color='red'
group by S.sid;
```

- Predpostavimo, da imamo naslednje indekse:
  - Reserves:
    - B+ drevo na sid
    - povezano B+ drevo na bid
  - Sailors:
    - B+ indeks ana sid
    - Razpršilni indeks na sid
  - Boats:
    - B+ indeks na color
    - Razpršilni indeks na color

Zgled 2: Prehod 1

- Iščemo najboljši plan za dostop do vseh treh relacij
- Za Reserves in Sailors je očitno najcenejši plan sekvenčni dostop
  - Nimamo selekcij na Reserves in Sailors
- Najboljši dostop do Boats je očitno razpršilni indeks na bid, ki se ujema z $\delta_{color='red'}$
- B+ indeks na color se tudi ujema s pogojem
  - Zaradi urejenosti rezultata ga obdržimo med možnimi plani

Zgled 2: Prehod 2

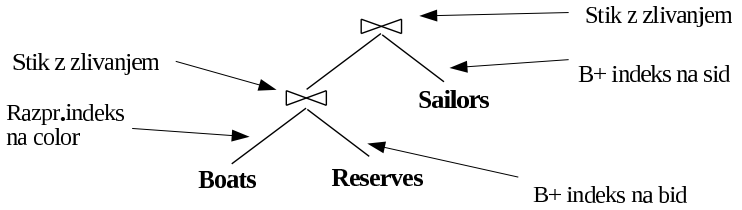
- Za vsak plan določen pri prehodu 1:
  - Uporabi plan za zunanjo relacijo
  - Poglej vse možne notranje relacije
- Oceniti je potrebno vse naslednje stike:
  - Pregled Reserves ⋈ Boats
  - Pregled Reserves ⋈ Sailors
  - Pregled Sailors ⋈ Boats
  - Pregled Sailors ⋈ Reserves
  - B+ dostop do Boats ⋈ Sailors
  - Razp.indeks do Boats ⋈ Sailors
  - B+ dostop do Boats ⋈ Reserves
  - Razpr.indeks do Boats ⋈ Reserves

Zgled 2: Prehod 2

- Za vsak stik je potrebno oceniti vse algoritme za stik.
  - Za vsak algoritem pogledamo vse možne metode dostopa do notranje relacije
- Na primer (Boats ⋈ Reserves):
  - Zapršilni indeks do Boats (color) ⋈(vgn.zanka z indeksom) B+ indeks za bid na Reserves
    - To bo verjetno zelo dober plan
    - Ne obstaja razpr.indeks za bid na Reserves
  - Razpršilni indeks do Boats (color) ⋈(stik z zlivanjem) B+ indeks za bid na Reserves
    - Rezultat je sortiran po bid
    - Plan zadržimo zaradi urejenosti rezultata (če ni cenejšega)
    - Prejšnji plan (vgn.zanka z indeksom) nebi zadržali, če bi bil ta cenejši!

Zgled 2: Prehod 3

- Vsak plan, ki je bil narejen v prehodu 2
  - Dodamo preostalo relacijo kot notranjo relacijo.
  - Pregledamo spet vse možne algoritme in za vsak algoritem vse možne metode dostopa do notranje relacije.
- Poglejmo si primer, ko dodamo relacijo Sailors k optimalnem planu za množico relacij {Boats,Reserves}.



Vgnezdene poizvedbe

- Vgnezden blok se optimizira neodvisno in zunanja 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.lid=103
 AND R.mid=M.mid)
```

Vgnezden blok za optim.:

```
SELECT  *
FROM  Rezervacije R
WHERE  R.lid=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.lid=103
```


Povzetek

- Optimizacija poizvedb je pomembno opravilo relacijskega SUPB.
- Potrebno je poznati optimizacijo, da bi lahko razumeli vpliv načrtovanja podatkovne baze na izvajanje delovne obremenitve aplikacije.
- Dva dela optimizacije poizvedb:
  - Naštevanje alternativnih planov.
    - Iskalni prostor je potrebno zmanjšati: levo-usmerjeni plani.
  - Oceniti je potrebno vsak plan.
    - Velikost rezultata + cena plana vsakega vozlišča.
    - *Ključni parametri*: statistike, indeksi, implementacije operacij RA.

Povzetek

- Plani nad eno relacijo:
  - Pregledamo vse metode dostopa.
  - *Teme*: Selekcije, ki se ujemajo z indeksi; ali ima ključ indeksa vse potrebna polja; ali indeks vrne n-terice v pričakovanem vrstnem redu.
- Plani nad več relacijami:
  - Najprej naštejemo vse plane nad eno relacijo.
    - Selekcije in projekcije se izvedejo čim hitreje je mogoče.
  - Za vsak plan z *i* relacijami vzamemo najboljši plan nad *i-1* relacijami in ga povežemo z stikom z naslednjo "notranjo" relacijo oz. najboljšim planom nad eno relacijo.
  - Za vsako podmnožico relacij ohranimo samo najboljši plan.

OPB, 2010/11
R.Ramakrishnan, DBMS

1. Naloga

Za podatkovno bazo letalske družbe (glej skico pri vajah) zapiši odgovore na naslednja vprašanja s povpraševalnim jezikom SQL. Pri tem poskusite odgovor optimizirati ali s spreminjanjem SQL stavkov ali s smiselno uporabo indeksov. Narišite še drevo poizvedb in poskušajte različne SQL stavke pognati v phpMyAdmin z ukazom EXPLAIN. Za vsak odgovor si oglejte razlago njegovega SQL stavka, ki jo vrne mySql (phpMyAdmin)

a. Poišči imena in priimke pilotov, ki so pilotirali vsaj eno letalo s kapaciteto večjo od 200 sedežev.

b. Poišči vse letalske družbe, ki imajo v lasti več kot 20 letal.

c. Poišči vse stevardese, ki so letela pod vodstvom pilota IdP=50.

d. Poišči naziv letalske družbe, ki ima trenutno največje število letal.

e. Poišči rezervacije (IdR), katerih cena je višja od 300 in se nanašajo na letala, katerih oznaka tipa letal je 'DC9' ali 'DC8'.

f. Poišči vse rezervacije (IdR), katerih cena je višja od 300 in se nanašajo na letala
   - pri katerih oznaka tipa letal je 'DC9' ali 'DC8' in
   - in ki so jih pilotirali piloti zaposleni pri letalski družbi v Sloveniji.

g. Poišči nazive vseh letališč, na katerih so že pristale stevardese, zaposlene pri letalskih družba iz Slovenije.

2. Naloga

Za podatlnovno bazo naftne družbe (glej Vaje02) zapiši odgovore na naslednja vprašanja s povpraševalnim jezkom SQL. Pri tem poskusi odogovor optimizirati ali s spreminjanjem samega SQL stavka poizvedbe ali s smiselno uporabo indeksov. Narišite še drevo poizvedb in poskušajte različne SQL stavke pognati v phpMyAdmin z ukazom EXPLAIN. Za vsak odgovor si oglejte razlago njegovega SQL stavka, ki jo vrne mySql (phpMyAdmin).

a. Poišči imena in priimke šoferjev, ki so že vozii vsaj eno cistemo s kapacitao višjo od 10.000 litrov.

b. Poišči vse benzinske črpalke, ki imajo več kot 10 točilnih mest.

c. Poišči vse zaposlene, ki trenutno delajo na bencinskii čpalkah z zalogo dizelskega goriva višjo od 5000 litrov.

d. Poišči naziv bencinske črpalke, ki imajo trenutno najvišje število točilnih mest.

e. Poišči vse dobave goriva (IdD), pri katerih je količina višja od 1.000 litrov in vkljlčujejo cisterne proizvajalcev 'MAN' ali 'Iveco'.

f. Poišči vse dobave goriva (IdD), pri katerih je količina višja od 1.000 litrov in vključujejo cisterne:
   - proizvajalcev 'MAN' ali 'Iveco' in
   - ki so jih vozii kaznovani šoferji (tj. šoferji s številom kazni > 0)

g. Poišči nazive vseh bencinskii črpalk, h katerim so naročeno gorivo pripeljali samo nekaznovani šoferji (tj. šoferji s številom kazni > 0).

Pregled upravljanja transakcij

Iztok Savnik, FAMNIT

Uvod

- Sočasno izvajanje uporabniških programov je ključno za dobre performanse SUPB.
  - Dostop do diska je pogost in relativno zamuden zato je pomembno, da je CPU konstantno zaposlen.
- Uporabniški program lahko izvrši več operacij nad podatki, vendar lahko dela le nad podatki, ki se preberejo/napišejo v bazo.
- Transakcija je abstrakten pogled SUPB na uporabniški program.
  - Sekvenca ukazov za branje in pisanje.

Sočasnost v SUPB

- Uporabniki si pri delu s transakcijami lahko predstavljajo, da se vsaka transakcija izvaja posebaj.
  - Sočasnost doseže SUPB z izmenjevanjem akcij (branje/pisanje) večih 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.

Sočasnost v SUPB

Problemi:

- Prekrivajoče transakcije.
  - Akcije ene transakcije lahko vplivajo na izvajanje druge transakcije.
- Sistemske napake.
  - Po sistemski napaki je potrebno zagotoviti konsistentno stanje podatkovne baze.

Atomičnost transakcij

- Stanje transakcij:
  - Transakcija je lahko potrjena po izvršitvi akcij.
  - Transakcija je lahko prekinjena (lahko jo prekini SUPB) po izvajanju neke akcije.
- Zelo pomembna lastnost, ki jo zagotavlja SUPB je atomičnost transakcije.
- Uporabnik lahko vidi transakcijo kot samostojno akcijo, ki se bodisi izvrši v celoti ali pa se ne izvrši.
  - SUPB zapisuje vse akcije tako, da jih je mogoče izničiti pri prekinjeni transakciji.

Lastnosti transakcij

- Konsistentnost
  - Če se izvaja sama mora ohraniti konsistentnost podatkovne baze.
- Izolacija
  - Transakcije so zaščitene pred vplivi drugih transakcij, ki se izvajajo hkrati.
- Trajnost
  - Po izvršitvi potrditve transakcije se učinki ohranijo tudi v primeru sistemske napake.

Primer

- Poglejmo si dve transakciji (*Xact*):

```
T1:   BEGIN  A=A+100,   B=B-100   END
T2:   BEGIN  A=1.06*A,  B=1.06*B  END
```

- Prva transakcija prenese $100 iz računa B na račun A.
- Druga transakcija doda vsakemu računu 6% obresti.
- Ni garancije, da se bo T1 izvajala pred T2 ali obratno.
- Poglejmo si kaj se lahko zgodi.

Primer

- Recimo, da se akcije prekrivajo na sledeč način (razporeditev)

```
T1:   A=A+100,                B=B-100
T2:             A=1.06*A,           B=1.06*B
```

- To je v redu. Kaj pa:

```
T1:   A=A+100,                          B=B-100
T2:             A=1.06*A, B=1.06*B
```

- Pogled SUPB na razporeditev:

```
T1:   R(A), W(A),                       R(B), W(B)
T2:             R(A), W(A), R(B), W(B)
```

Razporejanje transakcij

- Zaporedna razporeditev: Razporeditev s katero se akcije dveh različnih transakcij ne prekrivajo.
- Ekvivalentna razporeditev: Za vsako stanje podatkovne baze je rezultat izvajanja (na množici objektov SUPB) prve razporeditve identičen rezultatu izvajanja druge razporeditve.
- Serializibilna razporeditev: Razporeditev akcij, ki je ekvivalentna neki zaporedni razporeditvi.
- Če transakcija zagotovi konsistenco potem tudi serializibilna razporeditev zagotovi konsistentnost podatkovne baze.

Anomalije

- Prepis nepotrjenih podatkov (WW konflikt):

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


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
```


Kontrola sočasnosti z zaklepanjem

- *Dosledno dvo-fazno zaklepanje (Dosledno 2FZ)*:
  - Vsaka Xact mora pridobiti S (*deljen*) zaklep na objektu pred branjem in X (*ekskluzivni*) zaklep na objektu pred pisanjem.
  - Vsi zaklepi transakcije so sproščeni na koncu transakcije.
- ❖ *Nedosledno 2FZ*: Sprosti zaklenjene objekte kadarkoli, vendar po sprostitvi istih objektov ni moč ponovno zakleniti.
- Če Xact X zaklene nek objekt potem ne more nobena druga transakcija zakleniti istega objekta (S in X).

Kontrola sočasnosti z zaklepanjem

- Dosledno 2FZ:
  - Dovoli samo serializibilne razporeditve.
  - Poenostavi prekinitev transakcije.
- Nedosledeno 2FZ:
  - Zagotovi samo serializibilne razporeditve.
  - Zahteva bolj kompleksno proceduro za prekinitev transakcije.

Prekinitev 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 kompleten dnevnik vseh popravkov.
  - Prekinjene transakcije je mogoče izničiti.
  - Isti mehanizem se uporablja za vzpostavitev konsistentnega stanja po sistemski napaki.

Dnevnik (Log)

- Naslednje akcije se zapisujejo v dnevnik:
  - *Ti zapiše objekt*: stara vrednost in nova vrednost.
    - Zapis dnevnika se mora zapisati na disk pred podatki!
  - *Ti potrdi/prekine*: zapis dnevnika, ki zabeleži akcijo.
- Zapisi dnevnika so med seboj povezani z Xact ID.
  - Enostavno je izničiti specifično Xact.
- Dnevnik se pogosto podvoji in arhivira na stabilni medij
- Vse aktivnosti nad dnevnikom (kot tudi aktivnosti sočasnega izvajanja npr. zaklepanje, preprečevanje smrtnega objema, itd.) izvaja SUPB transparentno.

Vzpostavitev konsistentnega stanja po sistemski napaki

- Algoritem *Aries* ima tri faze:
- *1. Analiza*:
  - Pregled dnevnikov od zadnje fiksne točke.
  - Algoritem pregleduje naprej po dnevniku.
  - Identifikacija vseh Xacts, ki so bile aktivne v času sistemske napake.
  - Identifikacija vseh “umazanih” strani v času sistemske napake.

Algoritem Aries

- *2. ponovitev akcij (Redo)*:
  - Ponovi vse popravke “umazanih” strani.
  - Zagotovi, da so vsi popravki zapisani v dnevniku dejansko narejeni.
- *3. restavracija starega stanja (Undo)*:
  - Vsi popravki, ki so bili aktivni v času sistemske napake se izničijo; restavrira se staro stanje.
  - Algoritem pregleduje nazaj po dnevniku.
  - Paziti je potrebno na sistemske napake med procesom vzpostavljanja konsistentnega stanja (recovery).

Pregled

- Med najpomembnejšimi funkcijami SUPB:
  - Kontrola sočasnega izvajanja.
  - Vzpostavitev konsistentnega stanja po sistemski napaki.
- Uporabniki ne vedo za sočasno izvajanje.
  - Sistem sam vstavi kodo za zaklepanje/odklepanje objektov.
  - Sistem razporedi akcije različnih transakcij tako, da zagotovi rezultat ekvivalenten izvajanju zaporedne razporeditve transakcij (ena za drugo).

Pregled

- Dnevnik se uporablja za vzpostavitev začetnega stanja po:
  - prekinitvi transakcije ali
  - Za vzpostavitev konsistentnega stanja po sistemski napaki. .

❖ *Konsistentno stanje*: Vidni so samo učinki potrjenih Xact.

Transakcije

PREMALO NALOG, DODAJ ŠE 16.3 (str 220), 16.6 isolation-levels and two access-modes,

1. Transakcija T1 ima naslednjo razporeditev nad objektoma A in B baze podatkov:

R(A), W(A), R(B), W(B)

a. Podajte primer transakcije T2, ki bi, pognana vzporedno s transakcijo T1 lahko privedla do konflikta. Predpostavimo, da nimamo nobene kotrole vzporednosti.

b. Kako bi uporaba striktnega 2 faznega zaklepanja preprečila konfliktno situacijo iz prejšnjega primera. Ponovno napišite izvedbo.

2. Imamo naslednji transakciji s pripadajočima razporedoma

T1: R(A) R(B) W(A)
T2: R(A) R(B) W(A) W(B)

a. Podajte razpored transakcij T1 in T2, ki privede do WR konflikta.

b. Podajte razpored transakcij T1 in T2, ki privede do RW konflikta.

c. Podajte razpored transakcij T1 in T2, ki privede do WW konflikta.

d. Za vsako od podanih razporedov, pokažite kako bi strikten 2FZ to preprečil.

3. Za naslednje razporede 3eh transakcij povejte ali med njimi prihaja do konfliktov.

T1	R(A) W(A)		C
T2		W(A) C	
T3		R(A) C	

a. Če ja, jih opišite in razložite kako bi z 2 faznim zaklepanjem rešili te konflikte.

b. Kam bi morali potrditev T3 da bi med T2 in T3 prišlo do RW konflikta?

4. Imejmo naslednji dve transakciji:

T1: R(A), R(B), if A = 0 then B:=B+1, W(B)
T2: R(B), R(A), if B = 0 then A:=A+1, W(A)

Konsistentnost baze je zagotovljena z A = 0 ∨ B = 0 z začetnimi vrednostimi A = B = 0.

a. Prikaži, da po obeh možnih zaporednih izvajanjih baza ohrani konsistentnost.

b. Podajte primer, ko bi s sočasnim izvajanjem prišlo do nekonsistentnosti baze

c. A obstaja primer sočasnega izvajanja, ki bi ohranil konsistenčnost?

5. Kakšni so izhodi vseh select stavkov?

	T1	T2
a.	INSERT INTO Kvadrati VALUES (4);	
b.		SELECT * FROM Kvadrati;
c.		INSERT INTO Kvadrati VALUES (9);
d.		INSERT INTO Kvadrati VALUES (16);
e.		SELECT * FROM Kvadrati;
f.	COMMIT;	
g.		SELECT * FROM Kvadrati;
h.	SELECT * FROM Kvadrati;	
i.		COMMIT;
j.	SELECT * FROM Kvadrati;	

Kako bi rešili vse zaplete?

Kontrola vzporednosti

Iztok Savnik, FAMNIT

2FZ, zaporedna uredljivost in obnovljivost

- Zaporedni razporedi
- Graf odvisnosti
- Dvo-fazno zaklepanje
- Zaporedna uredljivost po oknih

Uredljivost konfliktov

- Dva razporeda sta konfliktno ekvivalentna, če:
  - Vsebujta enake akcije na istih transakcijah
  - Vsak par konfliktnih akcij je urejen na isti način
- Razpored S je zaporedno uredljiv po konfliktih, če je S konfliktno ekvivalenten nekem zaporednem razporedu.

Uredljiv razpored, ki ni zaporedno uredljiv po konfliktih.

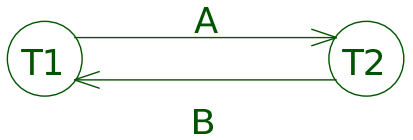
```
T1: R(A)       W(A)
T2:      W(A)
T3:             W(A)
```

Kasneje bomo videli, da je uredljiv po oknih.

Primer

- Razpored, ki ni uredljiv po konfliktih:

```
T1:    R(A), W(A),                    R(B), W(B)
T2:                 R(A), W(A), R(B), W(B)
```



Graf odvisnosti

- Cikel v grafu odkrije problem. T1 je odvisna od T2, in obratno.

Graf odvisnosti

- Graf odvisnosti:
  - Eno vozlišče za Xact.
  - Povezava od *Ti do Tj, če akciji Ti sledi konfliktna akcija Tj.*
- Izrek: Razporeditev je zaporedno uredljiva po konfliktih, če je graf odvisnosti acikličen.

Pregled: Strikten 2FZ

- Protokol striktnega 2 faznega zaklepanja (Strikten 2FZ):
  - Vsaka Xact mora zakleniti S (deljeno) objekt pred branjem in mora zakleniti X (ekskluzivno) objekt pred pisanjem.
  - Vsi zaklepi, ki jih držijo transakcije se sprostijo, ko se transakcije zaključijo.
  - Če neka Xact drži X zaklep na nekem objektu, potem nobena druga transakcija ne more dobiti zaklepa (S ali X) na tem objektu.
- Striktni 2FZ protokol dovoljuje samo razporede katerih graf odvisnosti je acikličen.

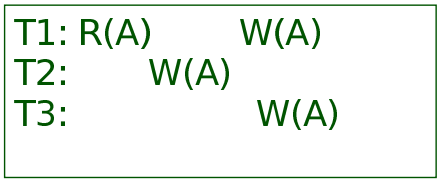
Dvo-fazno zaklepanje (2FZ)

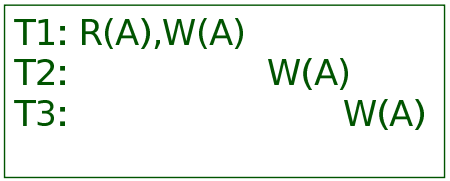
- Protokol dvo-faznega zaklepanja:
  - Vsaka Xact mora pridobiti S zaklep objekta pred branjem, in X zaklep pred pisanjem.
  - Transakcija ne more zahtevati dodatnih zaklepov po tem, ko sprosti zaklepe.
  - Če Xact drži X zaklep objekta potem nobena druga Xact ne more zakleniti objekta.

❖ Faza "širjenja" in "krčenja"

Zaporedna uredljivost po oknih

- Razporeda S1 in S2 sta "ekvivalentna po oknih" če:
  - Če Ti prebere začetno vrednost A v S1, potem Ti prebere začetno vrednost A v tudi S2
  - Če Ti prebere vrednost A, ki jo napiše Tj v S1, potem Ti prebere vrednost A, ki jo napiše Tj tudi v S2
  - Če Ti napiše končno vrednost A v S1, potem Ti napiše končno vrednost A tudi v S2





Upravljanje zaklepanja

- Zahteve po zaklepanju in odklepanju izvršuje upravitelj zaklepanja.
- Zapis tabele zaklepanja:
  - Št. transakcij, ki trenutno drži zaklep.
  - Tip zaklepa (S ali X)
  - Kazalec na vrsto zahtev po zaklepanju
- Zaklepanje in odklepanje mora biti atomična operacija.
- Popravek zaklepanja: transakcije, ki držijo deljen zaklep lahko spremenijo zaklep v ekskluziven.

Smrtni objemi

- Smrtni objem:
  - Cikel transakcij, ki čakajo druga drugo na odklep virov.
- S smrtnim objemom delamo na dva načina:
  - Preprečevanja smrtnega objema
  - Detekcija smrtnega objema

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.

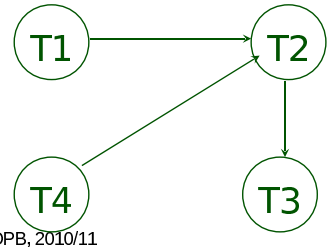
Odkrivanje smrtnega objema

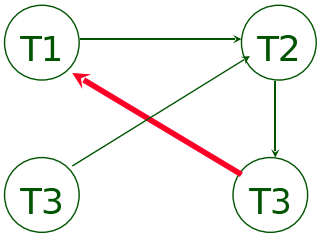
- Kreiraj čakalni graf:
  - Vozlišča so transakcije
  - Obstaja povezava od Ti do Tj, če Ti čaka na Tj za sproščanje zaklepa.
- Periodično preveri ali ima čakalni graf cikle.

Odkrivanje smrtnega objema

Primer:

T1: S(A), R(A), S(B)
T2: X(B),W(B) X(C)
T3: S(C), R(C) X(A)
T4: X(B)





Tehnike zaklepanja

- Fantomski problem
- Zaklepanje B+ dreves
- Večnivojsko zaklepanje

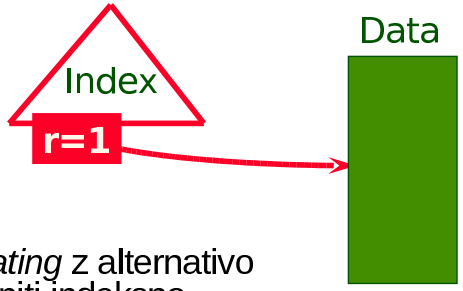
Dinamične podatkovne baze

- Če sprostimo predpostavko, da je PB fiksna kolekcija objektov potem tudi striktno 2FZ ne zagotovi serializabilnosti:
  - T1 zaklene vse strani, ki vsebujejo zapise mornarjev z *rating* = 1 in poišče najstarejšega mornarja (npr., *age* = 71).
  - Naprej, T2 vstavi novega mornarja; *rating* = 1, *age* = 96.
  - T2 zbriše tudi najstarejšega mornarja z rating = 2 (*age* = 80) ter naredi commit.
  - T1 zdaj zaklene vse strani, ki vsebujejo zapise mornarjev z *rating* = 2 in poišče najstarejšega (npr., *age* = 63).
- Nekonsistentno stanje PB, če je T1 “korektna”!

Problem

- T1 implicitno predpostavlja, da je zaklenila celotno množico mornarjev za katere velja *rating* = 1.
  - Predpostavka velja samo, če se ne dodaja novih zapisov mornarjev medtem, ko se T1 izvaja!
  - Potrebujemo nek mehanizem, ki to zagotovi. (Zaklepanje indeksov in predikatov.)
- Primer pokaže, da zaporedna uredljivost po konfliktih garantira uredljivost samo, če je množica objektov fiksna!

Zaklepanje indeksa



- 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!
- Če ni primernega indeksa potem mora T1 zakleniti vse strani kot tudi zakleniti datoteko/tabelo, da bi onemogočili dodajanje zapisov z *rating* = 1.

Zaklepanje predikatov

- Zaklep vseh zapisov, ki ustrezajo pogoju logičnega predikata, npr. *age > 2*salary*.
- Zaklepanje indeksa je poseben primer zaklepanja predikata.
- Indeks podpira učinkovito implementacijo zaklepanja predikata.
  - Kaj je predikat v primeru mornarjev?
- V splošnem vsebuje zaklepanje predikatov veliko dodatnega dela zaradi zaklepanja.

Zaklepanje v B+ drevesih

- Kako lahko učinkovito zaklenemo določen list drevesa?
  - Mmg, ne zamenjuj tega z večnivojskim zaklepanjem!
- Ena rešitev: ignoriraj strukturo drevesa; zakleni strani med potovanjem po drevesu; uporabljaj 2FZ.
- Učinkovitost tega algoritma je slaba!
  - Koren drevesa in višji nivoji drevesa postanejo ozko grlo, ker se vsak dostop začne pri korenu.

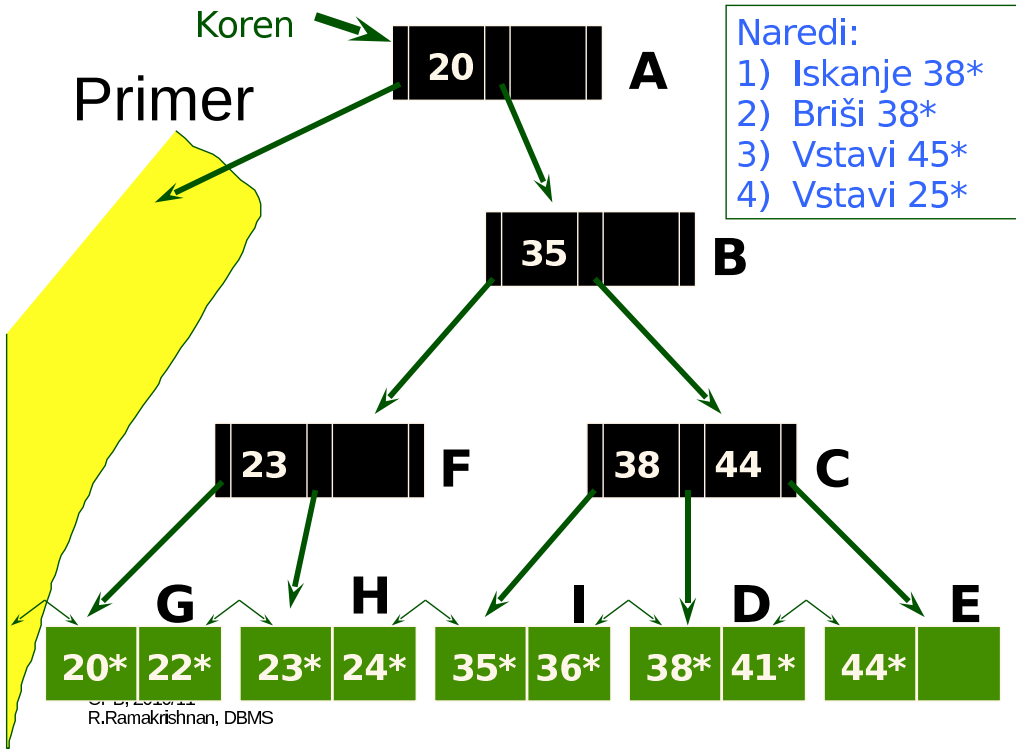
Dve koristni opazki

1) Višji nivoji drevesa samo usmerjajo iskanje listov (strani).
2) Pri vstavljanju morajo biti vsa vozlišča od korena do spremenjenega vozlišča zaklenjena na X način, samo v primeru, da lahko razcep prodre navzgor od spremenjenega lista.

- Opazki lahko uporabimo pri zasnovi učinkovitih protokolov za zaklepanje, ki zagotavljajo zaporedno uredljivost *navkljub kršitvi 2PZ.*

Enostaven algoritem za zaklepanje dreves

- Iskanje: začni pri korenu in na poti do lista vedno zakleni otroka z S zaklepom in odkleni starša.
- Vstavi/Briši: Začni pri korenu in potuj navzdol tako, da pridobiš X zaklepe po potrebi. Ko je otrok zaklenjen preveri, če je varen:
  - Če je otrok varen sprosti vse zaklepe na starših.
- Varno vozlišče: Vozlišče, ki se spremeni spremembe ne bo širilo navzgor.
  - Vstavljanje: Vozlišče ni polno.
  - Brisanje: Vozlišče ni pol prazno.

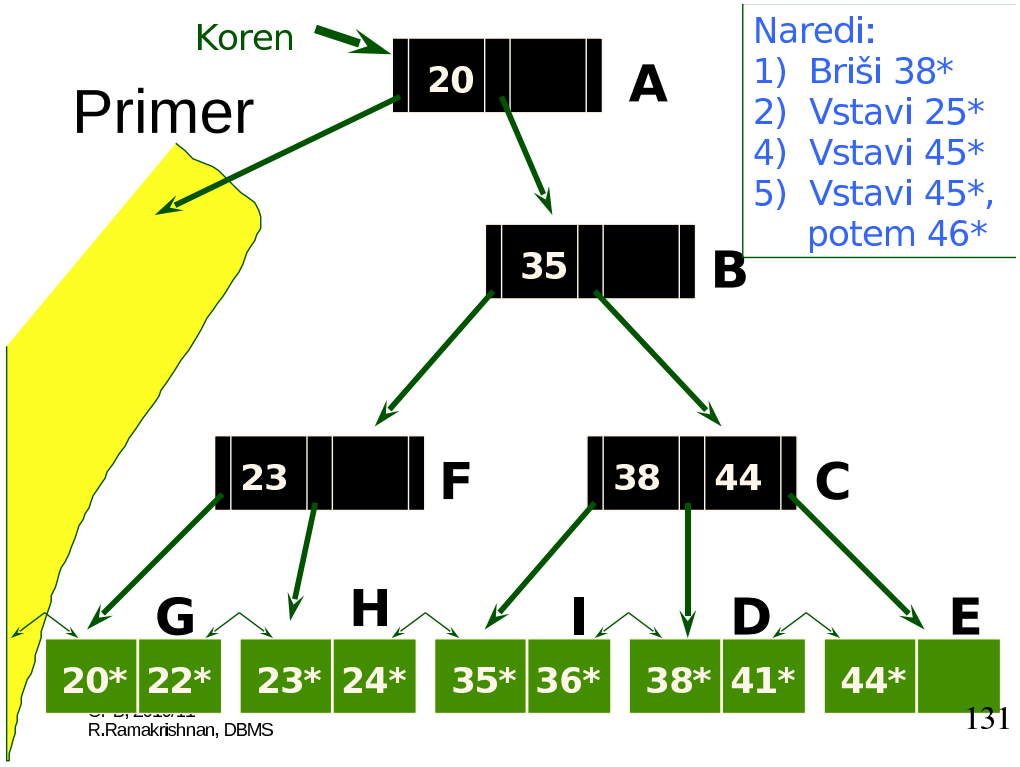


OPB, 2010/11
R.Ramakrishnan, DBMS

Boljši algoritem za zaklepanje drevesa

- Iskanje: Kot prej.
- Vstavljanje/Brisanje:
  - Postavi zaklepe kot v primeru iskanja, pride do lista in postavi X zaklep na listu.
  - Če list ni varen, sprosti vse zaklepe in ponovno poženi Xact s prejšnjim protokolom za Vstavljanje/Brisanje.
- Spekulira, da se bo spremenil samo list
  - Če ne so S zaklepi pri prvem prehodu odvečni.
  - V praksi dobimo boljši algoritem od prejšnjega.

OPB, 2010/11
R.Ramakrishnan, DBMS



OPB, 2010/11
R.Ramakrishnan, DBMS

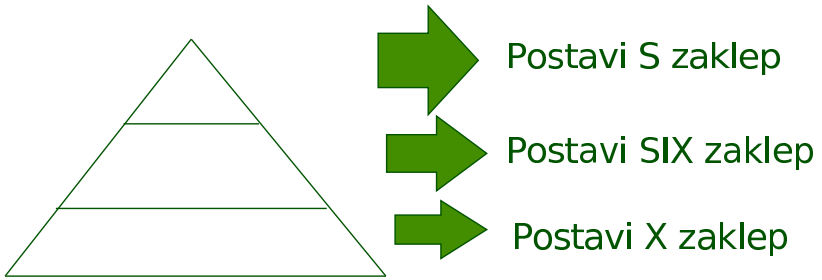
Še boljši algoritem

- Iskanje: Kot prej.
- Vstavljanje/Brisanje:
  - Uporabi orginalen protokol za vstavljanje/brisanje toda postavi IX zaklepe namesto X zaklepov.
  - Ko je list zaklenjen, prevedi vse IX zaklepe v X zaklepe od zgoraj-navzdol:
    - Začnemo v vozlišču najbljižjem korenu.
    - Protokol od-zgoraj-navzdol zmanjša možnosti za smrtni objem.

(Primerjaj uporabo IX zaklepov z uporabo v večnivojskem zaklepanju.)

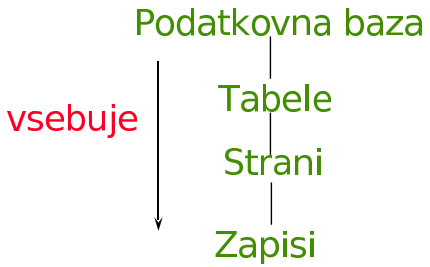
Hibriden algoritem

- Verjetnost, da res potrebujemo zaklep tipa X se manjša s pomikanjem višje po drevesu.
- Hibriden pristop:



Večnivojsko zaklepanje

- Težko se je odločiti kateri nivo zaklepati (zapisi vs. strani vs. tabele).
- Nebi se smeli odločati o tem!
- Podatkovni "kontejnerji" so vgnezdeni:



Rešitev: Nova stanja zaklepanja, protokol

- Dovolimo Xact zaklepati vsak nivo
  - Uporaba protokola, ki vsebuje "namere" zaklepanja:

- Pred zaklepom objekta mora Xact postaviti "namero" zaklepa na vseh naslednikih.
- Pri odklepanju gremo od specifičnega proti splošnem (t.j., od-spodaj-navzgor).
- SIX način: Kot S & IX skupaj.

Večnivojski protokol zaklepanja

- Vsaka Xact začne iz korena hierarhije.
- Da bi dobili S ali IS zaklep vozlišča moramo imeti IS ali IX na starših.
  - Kaj če Xact drži SIX na staršu? S na staršu?
- Da bi dobili X ali IX ali SIX zaklep vozlišča moramo imeti IX ali SIX na starših.
- Moramo sproščati zaklepe od spodaj navzgor.

Protokol je korekten ker je ekvivalenten protokolu, ki dela direktno z listi.

Primeri

- T1 pregleda R in popravi nekaj n-teric:
  - T1 dobi SIX zaklep na R, potem lahko dobi S zaklepe na n-tericah R in občasno spremeni stanje v X zaklep n-teric.
- T2 uporablja indeks za branje dela R:
  - T2 dobi IS zaklep na R in s tem dobi S zaklep na n-tericah R.
- T3 prebere celoten R:
  - T3 dobi S na R.
  - ALI, T3 bi se lahko obnašala kot T2; uporaba lock escalation da bi se odločila katere n-terice.

Kontrola vzporednosti brez zaklepanja

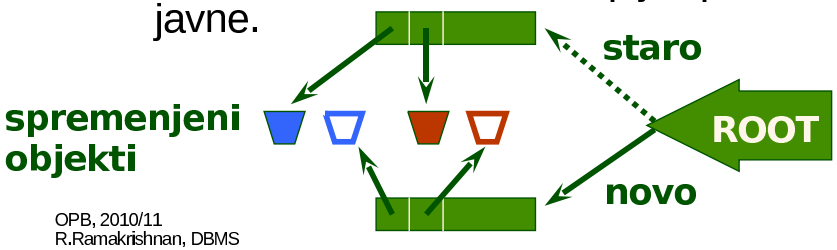
- Optimistična kontrola zaklepanja
- Kontrola vzporednosti s časovnimi značkami (Timestamp CC)
- Multiverzijska kontrola vzporednosti s časovnimi značkami (Multiversion Timestamp CC)

Optimistična kontrola vzporednosti

- Zaklepanje je konzervativni pristop, ki prepreči konflikte. Slabe lastnosti:
  - Dodatna koda za zaklepanje.
  - Preverjanje/reševanje smrtnega objema.
  - Zasičenost z zaklepi za vse pogosto uporabljene objekte.
- Če so konflikti redki lahko pridobimo na vzporednosti tako, da ne zaklepamo.
  - Namesto tega preverimo konflikte pred izvajanjem Xacts potrjevanja.

Kung-Robinson model

- Xacts ima tri faze:
  - READ: Xacts prebere iz pod.baze vendar naredi spremembe na lokalni kopiji objektov.
  - VALIDATE: Preveri konflikte.
  - WRITE: Naredi lokalne kopije sprememb javne.



Validacija

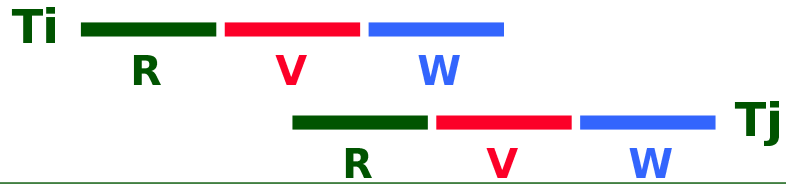
- Preveri pogoje, ki so zadostni za to, da se ne pojavi konflikt.
- Vsaka Xact dobi numerični id.
  - Uporabi časovni žig.
- Xact id-ji se priredijo na koncu faze BERI, tik pred validacijo. (zakaj potem?)
- ReadSet(Ti): Množica objektov, ki jih prebere Xact Ti.
- WriteSet(Ti): Množica objektov, ki jih spremeni Ti.

Test 1

- Za vse i in j tako da Ti < Tj, preveri če se Ti zaključi preden Tj začne.

Test 2

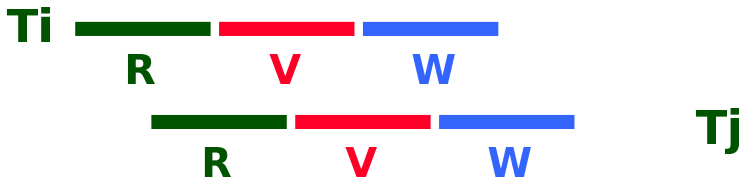
- Za vse i in j tako da Ti < Tj, preveri:
  - Ti zaključi preden Tj začne s pisalno fazo +
  - WriteSet(Ti) ∩ ReadSet(Tj) je prazna.



Ali Tj prebere “umazane” podatke?
Ali Ti prepiše zapise Tj?

Test 3

- Za vse i in j tako da Ti < Tj, preveri da:
  - Ti zaključi bralno fazo pred Tj **+**
  - WriteSet(Ti) ∩ ReadSet(Tj) je prazna **+**
  - WriteSet(Ti) ∩ WriteSet(Tj) je prazna.



Ali Tj prebere “umazane” podatke?
Ali Ti prepiše zapise Tj?

Aplikacija Testov 1 & 2: Zaporedna validacija

- Validacija Xact T:

```
valid = true;
// S = set of Xacts that committed after Begin(T)
< foreach Ts in S do {
  if ReadSet(T) does intersect WriteSet(Ts)
      then valid = false;
  }
  if valid then { install updates; // Write phase
                  Commit T } >
            else Restart T
```

end of critical section

Zaporedna validacija: komentarji

- Aplicira Test 2, kjer T igra vlogo Tj in vsaka Xact v Ts ustreza Ti.
- Prirejanje Xact id, validacija, in pisalna faza so znotraj kritične sekcije!
  - Nič drugega se ne izvaja vzporedno.
  - Če je pisalna faza dolga potem je to problem.
- Optimizacija (samo) za bralne Xacts:
  - Ne potrebujemo kritične sekcije (ker ni pisalne faze).

Zaporedna validacija

- Večnivojska zaporedna validacija:
  - Validiraj na vseh nivojih, na vsakem nivoju validiranja T nad podmnožico Xacts, ki so potrjeni (committed) po Begin(T).
  - Samo zadnja faza mora biti znotraj kritične sekcije.
- Stradanje:
  - Poženi stradajočo Xact v kritični sekciji (!!)
- Prostor za WriteSets:
  - Za validacijo Tj, moramo imeti WriteSets za vse Ti kjer Ti < Tj in Ti je aktivna, ko Tj začne.
  - Lahko je več takšnih Xacts in lahko zmanjka prostora.
  - Validacija Tj ne uspe, če zahteva manjkajočo WriteSet.
  - Ni problema, če so Xact ids prirejeni na začetku bralne faze.

Problemi optimistične kontrole vzporednosti

- Mora zabeležiti vse aktivnosti branja/pisanja v ReadSet in WriteSet za posamezno Xact.
  - Mora se kreirati in sprostiti množice po potrebi.
- Mora preveriti konflikte pred validacijo in mora izvajati validirana pisanja ``globalno''.
  - Kritična sekcija lahko zmanjša vzporednost!
  - Pisanje "globalno" lahko razdrobi skupke strani po tabelah in indeksih.
- Optimistična KV ponovno požene Xacts, ki se ne validira.
  - Delo narejeno do takrat je izgubljeno; potrebuje še čiščenje.

``Optimističen'' 2FZ

- Lahko bi naredili tudi sledeče:
  - Postavi S zaklepe kot običajno.
  - Naredi spremembe na privatnih kopijah objektov.
  - Pridobi vse X zaklepe na koncu Xact, naredi pisanja globalna in potem sprosti zaklepe.
- Za razliko od Optimistične KV v primeru Kung-Robinson, so rezultati te sheme blokirane Xact, ki čakajo na zaklep.
  - Ni validacije, ni ponovnega izvajanja transakcij (modulo smrtni objemi).

Kontrola vzporednosti s časovnimi žigi

- **Ideja:** Priredi vsakemu objektu bralni časovni žig (RTS) in pisalni časovni žig (WTS), priredi vsaki Xact časovni žig (TS) ko se začne:
  - Če je akcija ai od Xact Ti v konfliktu z akcijo aj od Xact Tj, in TS(Ti) < TS(Tj), potem se mora ai zgoditi pred aj. Sicer, ponovno poženi Xact.

Ko Xact T želi prebrati objekt O...

- Če TS(T) < WTS(O), potem je kršena urejenost časovnih žigov T nasproti pisalcu O.
  - Prekini T in jo ponovno zaženi z večjim TS. (sicer spet nebi uspela)
  - Primerjaj uporabo TS v 2PZ za izogibanje smrtnim objemom.
- Če TS(T) > WTS(O):
  - Dovoli T prebrati O.
  - Resetiraj RTS(O) na max(RTS(O), TS(T))
- Spremembe RTS(O) ob branju se morajo zapisati na disk!
  - To in ponovni zagoni so dodatno delo.

Ko Xact T želi zapisati objekt O...

- Če TS(T) < RTS(O), potem to krši urejenost časovnih žigov T nasproti pisalcu O; prekini in ponovno zaženi T.
- Če TS(T) < WTS(O), krši urejenost časovnih žigov T nasproti pisalcu O.
  - **Thomas-ovo pisalno pravilo: Varno lahko ignoriramo prestara pisanja; ni potrebe po ponovnem zagonu T!**
  - **Pisanju T sledijo druga pisanja, brez vmesnih branj.**
  - **Dovoljuje zaporedne ureditve ne pa tudi ureditve po konfliktih.**
- Sicer, dovoli T zapisati O.

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

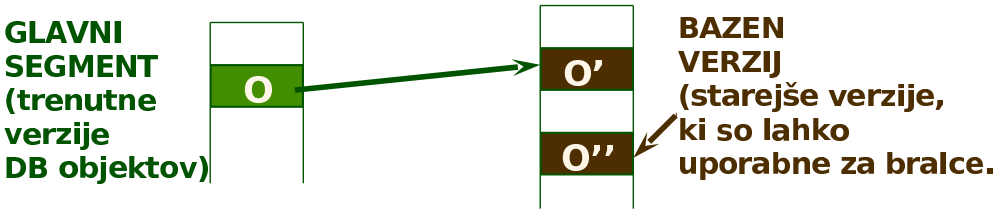
Kontrola vzporednosti in obnovljivost

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

- Na žalost so možne neobnovljive razporeditve:
- Kontrola vzporednosti s časovnimi žigi se da prilagoditi tako da dovoli samo obnovljive razporeditve:
  - Shrani vse zapise dokler pisalec ne potrdi (popravi tudi WTS(O) ko je pisanje dovoljeno.)
  - Blokiraj bralce T (kjer TS(T) > WTS(O)) dokler pisalec O ne potrdi.
- Podobno pisalcem, ki držijo X zaklep do potrditve, vendar še vedno ni zares 2FZ.

Multi-verzijska kontrola vzporednosti s časovnimi žigi

- **Ideja:** Naj pisalci naredijo novo kopijo medtem ko bralci berejo staro:

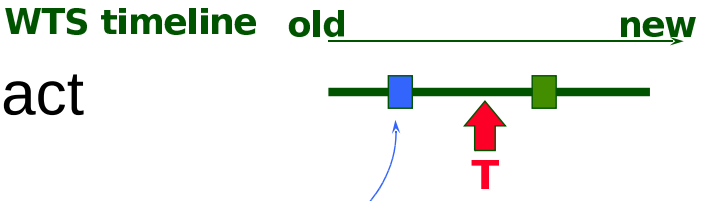


- Bralci lahko vedno delajo.
  - Lahko so blokirani dokler pisalci ne potrdijo pisanje.

Multi-verzijska kontrola vzporednosti (2)

- Vsaka verzija objekta ima TS pisalca kot WTS in TS Xact bralca, ki je zadnji prebral to verzijo kot RTS.
- Verzije so povezane nazaj; opustimo lahko verzije, ki so "prestare".
- Vsaka Xact se klasificira kot Bralec ali Pisalec.
  - Pisalec lahko napiše kakšen objekt; Bralec ne bo nikoli.
  - Xact pove ali je Bralec, ko se začne izvajat.

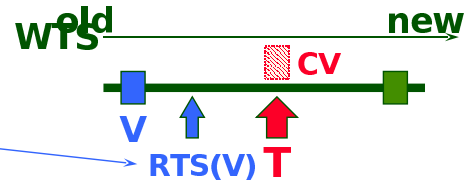
Bralec Xact



- Za vsak objekt, ki bo prebran:
  - Poišči **najnovejšo verzijo** z WTS < TS(T).
  - Začni s trenutno verzijo in se premikaj nazaj po seznamu.
- Če predpostavljamo, da najdemo neko verzijo vsakega objekta od začetka časa, potem bralčeve Xact ni potrebno nikoli ponovno zagnati.
  - Lahko blokira dokler pisalec prave verzije ne potrdi.

Pisalec Xact

- Za branje objekta uporabi protokol bralca.
- Za zapisovanje objekta:
  - Poišči najnovejšo verzijo V z WTS < TS(T).
  - Če RTS(V) < TS(T), potem T naredi kopijo CV verzije V, s kazalcem na V in WTS(CV) = TS(T), RTS(CV) = TS(T).
  - Vse zapisano je hranjeno dokler ni potrjeno; ostale Xacts lahko vidijo vrednosti TS vendar ne more prebrati verzije CV.
  - Sicer, zavrni pisanje.



Podpora transakcijam v SQL-92

- Vsaka transakcija ima način dostopa, predvideno velikost, in nivo izolacije.

Isolation Level	Dirty Read	Unrepeatable Read	Phantom Problem
Read Uncommitted	Maybe	Maybe	Maybe
Read Committed	No	Maybe	Maybe
Repeatable Reads	No	No	Maybe
Serializable	No	No	No

1. Transakcija T1 in T2 imata naslednji razporeditvi nad objektoma A in B baze podatkov:

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

a. Kako bi razpored transakcij sočasno izvajali, da pridemo do smrtnega objema?

b. Kako bi rešili smrtni objem?

c. Kako bi ga rešili z Rani-Čakaj?

d. Kako bi ga rešili z Počakaj-Umri?

2. Imamo naslednje razporede:

S1=W2(x), W1(x), R3(x), R1(x), W2(y), R3(y), R3(z), R2(x)
S2=R3(z), R3(y), W2(y), R2(z), W1(x), R3(x), W2(x), R1(x)
S3=R3(z), W2(x), W2(y), R1(x), R3(x), R2(z), R3(y), W1(x)
S4=R2(z), W2(x), R3(z), W1(x), W2(y), R1(x), R3(x), R3(y)

a. Kateri od zgornjih razporedov je konfliktno ekvivalenten?

b. Kateri od zgornjih razporedov je zaporedno uredljiv po konfliktih (conflict serializable)?

3. Imamo naslednji dve transakciji:

T1: R(A); R(B); if A = 0 then B=B+1; W(B).
T2: R(B); R(A); if B = 0 then A=A+1; W(A).

a. Dodajte zaklepe in sprostitve z dofaznim zaklepanjem 2ZF.

b. Ali lahko pri izvajanju teh dveh transakcij privede do smrtnega objema?

c. Kako bi rešili smrtni objem s konzervativnim 2FZ?

d. Kako bi rešili smrtni objem z Počakaj-Umri?

4. Imamo naslednja razporeda:

S1: T1:R(X), T2:W(X), T2:W(Y), T3:W(Y), T1:W(Y), T1:Commit, T2:Commit, T3:Commit
S2: T1:R(X), T2:W(Y), T2:W(X), T3:W(Y), T1:W(Y), T1:Commit, T2:Commit, T3:Commit

Za vsakega od spodnjih mohanizmov za kontrolo vzporednosti in razporeda ugotovi, kako mehanizm izvede razpored.

a. Strikten 2FZ s časovnimi zaznamki za preprečevanje smtrnih objemov

b. Strikten 2FZ z detekcijo zaznave smtrnih objemov (prikažite waits-for graph in case of deadlock.)

c. Konzervativen 2FZ (and Strict, i.e., with locks held until end-of-transaction) 2PL.

d. Optimistična kontrola vzporednosti.

e. Kontrola vzporednosti s časovnimi žigi

f. Multi-verzijska kontrola vzporednosti s časovnimi žigi

Obnovitev po zrušitvi

Iztok Savnik, FAMNIT

Pregled: ACID lastnosti

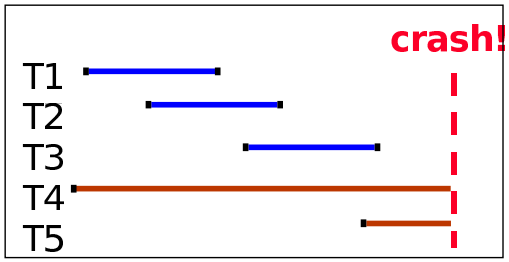
- **A**tomicity: (atomarnost)
  Izvršijo se vse akcije Xact ali nobena.
- **C**onsistency: (konsistentnost)
  Če sta Xact in celotna PB konsistentna, potem je po transakciji PB konsistentna.
- **I**solation: (izolacija)
  Izvajanje ene Xact je izolirano od izvajanja drugih transakcij.
- **D**urability: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

- **Upravljalnik obnavljanja** garantira atomarnost & ohranjanje.

Motivacija

- Atomarnost:
  - Transakcija se lahko prekine ("Rollback").
- Ohranjanje:
  - Kaj če neha delati SUPB? (Razlogi?)

- Pričakovano obnašanje po ponovnem zagonu sistema:
  - T1, T2 & T3 bi morale biti persistentne.
  - T4 & T5 bi morale biti prekinjene (izničene).



Predpostavke

- Kontrola vzporednosti se izvaja.
  - Striktni 2FZ, konkretno.
- Popravki se dogajajo „na mestu".
  - Podatki so prepisani (izbrisani) na disk.

- Enostavna shema garantira atomarnost in ohranjanje.

Delo z izravnalnim bazenom strani

- Zagotovi vsa pisanja na disk?
  - Slab odzivni čas.
  - Omogoča ohranjanje podatkov.
- Ukradi okvirje izravnalnega bazena nepotrjenim Xacts?
  - Če ne, slab pretok.
  - Če da, kako zagotoviti atomarnost?

	No Steal	Steal
Force	Trivial	
No Force		Desired

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 pade preden se spremenjena stran zapiše na disk?
  - Zapiši čim manj je možno, na primerno mesto, ob potrjevanju, da bi omogočili REDO sprememb.

Osnovna ideja: Dnevniki

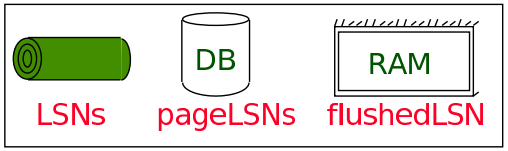
- Zapiši REDO in UNDO podatke za vsak popravek v dnevnik.
  - Sekvenčno pisanje v dnevnik (gre na ločen disk).
  - Minimalno podatkov (diff) zapisano v dnevnik; več popravkov gre na eno stran.
- Dnevnik: Urejen seznam REDO/UNDO akcij.
  - Zapis dnevnika vsebuje:
    <XID, pageID, offset, length, old data, new data>
  - in dodatne kontrolne podatke (do katerih pridemo kmalu).

Dnevnik: Pisanje-vnaprej (WAL)

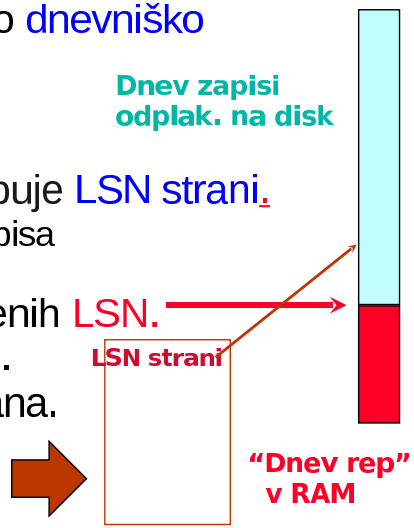
- Dnevnik s protokolom Pisanje-vnaprej:
  - #1 Mora shraniti dnevniški zapis preden se pripadajoča stran zapiše na disk.
  - #2 Mora zapisati vse dnevniške zapise za Xact pred potrditvijo.
- #1 garantira atomarnost.
- #2 garantira ohranitev.

- Kako natančno je dnevnik (in obnovitev) narejen?
  - Študirali bomo ARIES algoritme.

WAL & dnevnik



- Vsak dnev zapis ima unikatno dnevniško sekvenčno številko.
  - Log Sequence Number (LSN).
  - LSN vedno narašča.
- Vsaka podatkovna stran vsebuje LSN strani.
  - LSN zadnjega dnevniškega zapisa popravka dane strani.
- Sistem hrani stanje odplaknjenih LSN.
  - Maks LSN odplaknjen do sedaj.
- WAL: Preden je stran zapisana.
  - page LSN ≤ flushed LSN



Dnevniški zapisi

Polja dnev zapisa:

- prevLSN
- XID
- type
- samo za update zapise:
  - pageID
  - length
  - offset
  - before-image
  - after-image

Možni tipi dnev zapisov:

- Update
- Commit
- Abort
- End
  - konec potrjevanja ali prekinitve
- Kompenzacijski dnev zapisi (CLRs)
  - za UNDO akcije

Strukture povezane z dnevnikom

- Transakcijska tabela
  - En vpis za eno aktivno Xact.
  - Vsebuje XID, status (running/commited/aborted) in lastLSN.
- Tabela umazanih strani:
  - En vpis za eno umazano stran v izravnalnem bazenu.
  - Vsebuje recLSN -- LSN dnev zapisa, ki je prvi povzročil, da je stran umazana.

Dnevniško pisanje-vnaprej (WAL)

- 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.
  - Prepostavili bomo, da je pisanje na disk atomično.
    - V praksi imamo več podrobnosti okoli atomičnosti pisanja.
- Strikten 2FZ.
- Delo izravnalnika: STEAL, NO-FORCE + WAL dnevnik.

Kontrolna točka

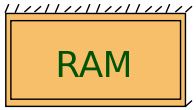
- Periodično SUPB kreira kontrolno točko za minimizacijo časa porabljenega za obnovitev v primeru sistemske zrušitve.
- Zapiši v dnevnik:
  - begin_checkpoint zapis: Določa začetek KT.
  - end_checkpoint zapis: Vsebuje trenutno tabelo *Xact* in tabelo umazanih strani. To je `mehka kontrolna točka':
    - 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).

Ptičja perspektiva: Kaj je shranjeno kje

LogRecords
prevLSN
XID
type
pageID
length
offset
before-image
after-image

Data pages
each
with a
pageLSN

master record



Xact Table
lastLSN
status

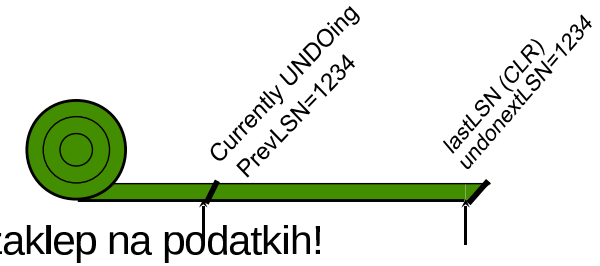
Dirty Page Table
recLSN

flushedLSN

Enostavna prekinitev transakcije

- Za zdaj si poglejmo samo eksplicitno prekinitev Xact.
  - Ni zrušitve sistema.
- Želimo "odvrteti nazaj" dnevnik v obratnem vrstnem redu z izničenjem (UNDO) sprememb.
  - Dobi lastLSN transakcije Xact iz tabele Xact.
  - Lahko sledimo dnev zapisom z uporabo polja prevLSN.
  - Pred začetkom vračanja z UNDO, zapiši dnev zapis prekinitve
    - Za reševanje iz zrušitve med vračanjem.

Prekini (2)



- UNDO zahteva zaklep na podatkih!
  - Ni problem!
- Preden restavriramo staro vrednost strani, se napiše CLR:
  - Dnevnik se lahko piše med UNDO!!
  - CLR ima eno dodatno polje: undonextLSN
    - Kaže na naslednji LSN za undo.
    - prevLSN zapisa, ki ga trenutno restavriramo.
  - CLR ni nikoli restavriran
    - Lahko je ponovno izveden pri ponavljanju zgodovine (atomarnost).
- Na koncu UNDO zapiši “end” dnevniški zapis.

Potrditev transakcije

- Zapiši potrdi zapis v dnevnik.
- Vsi dnev zapisi vse do lastLSN Xact so odplaknjeni.
  - Garantira flushedLSN ≥ lastLSN.
  - Odplakovanje dnevniških strani je sekvenčno, sinhrono zapisovanje na disk.
  - Veliko dnevniških zapisov na stran.
- Commit() se vrne.
- Zapiši end zapis v dnevnik.

Obnovitev po zrušivi: celotna slika

- Začni od kontrolne točke (poiščemo jo preko master zapisa).
- Tri faze. Potrebno je:
  - Odkriti katera Xacts je potrdila od kontrolne točke naprej in katera ni uspela (Analiza).
  - REDO vse akcije.
    - (ponovi zgodovino)
  - UNDO učinki neuspelih Xacts.

Obnovitev: faza analize

- 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.
- Za vsako CLR ali LSN dnev zapisa popravka, naredi REDO akcije razen v primeru:
  - Spremenjena stran ni v TUS ali
  - Spremenjena stran je v TUS, vendar recLSN > LSN ali
  - pageLSN (v DB) ≥ LSN.
- REDO akcije:
  - Ponovi akcijo v dnevniku.
  - Postavi pageLSN na LSN. Ni dodatnega zapisa v dnev!

Obnovitev: Faza UNDO

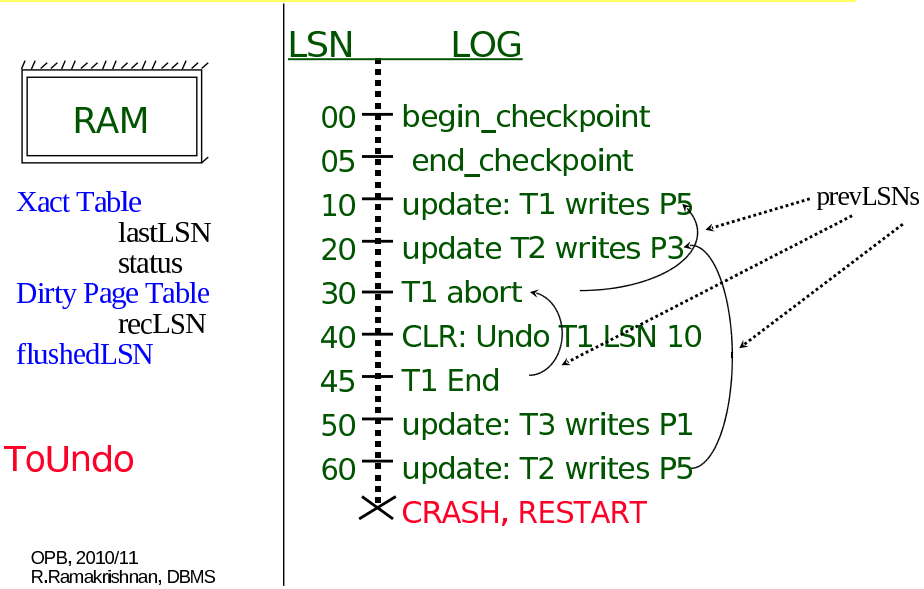
ToUndo={ / | / -- lastLSN prekinjenih Xact}

Repeat:

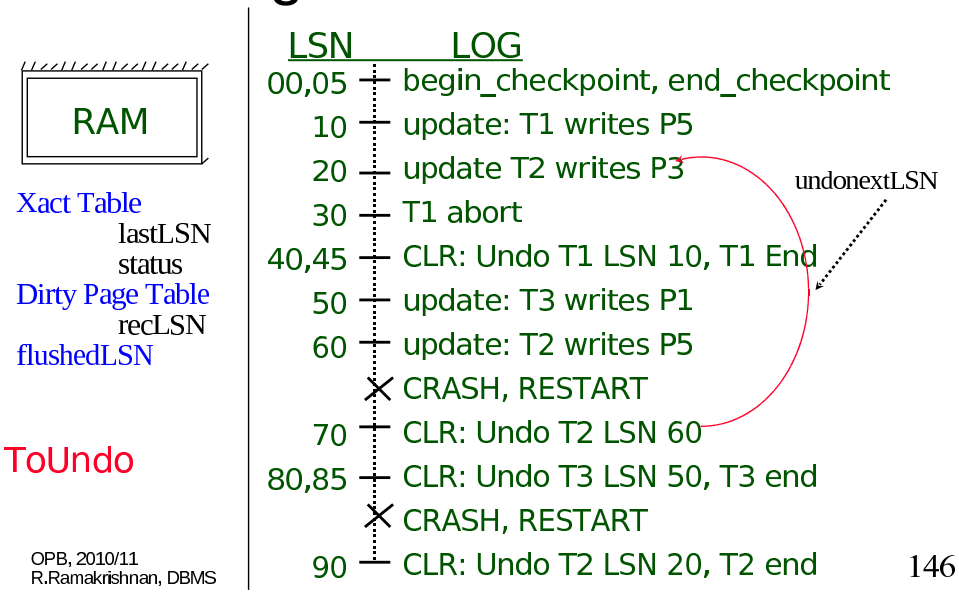
- Izberi največji LSN med ToUndo.
- Če je LSN CLR in velja undonextLSN==NULL
  - Zapiši End zapis za to Xact.
- Če je LSN CLR, in undonextLSN != NULL
  - Dodaj undonextLSN k ToUndo
- Sicer je LSN popravek. Izniči popravek, zapiši CLR in dodaj prevLSN k ToUndo.

Until ToUndo je prazen.

Primer obnovitve



Primer: Zrušitev med ponovnim zagonom



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.

Ponavljanje

Naloge iz starih kolokvijev in izpitov.

Najpomembnejše tabele v informacijskem sistemu Banke so naslednje.
Stranka(sid,ime,priimek,kraj,naslov,telefon);
Racun(rid,sid,znesek,valuta);
Polog(rid,znesek,ime,priimek,naslov,datum);
Dvig(rid,znesek,ime,priimek,naslov,datum);

Dani so še naslednji podatki.
1 stran na disku = 8K
|Stranka| = 100.000 zapisov, 100 zlogov, 80 zapisov/stran, 1250 strani
|Racun| = 120.000 zapisov, 80 zlogov, 100 zapisov/stran, 1200 strani
|Polog| = 1.000.000 zapisov, 120 zlogov, 66 zapisov/stran, 15151 strani
|Dvig| = 1.500.000 zlogov, 120 zlogov, 66 zapisov/stran, 22727 strani

Vse dodatne predpostavke uporabljene v nalogah napišite !

1. Naloga (25%) Napiši odgovore na naslednja vprašanja v QBE.
a) Izpiši vse pologe v znesku 1.000EU ali več ter za stranke z imenom Tone.
b) Izpiši številke vseh računov na katere je bilo položeno v zadnjem mesecu več kot 100.000 EU.

2. Naloga (25%) Koliko blokov prebere naslednje vprašanje, če ne obstaja nobeden indeks? Predpostavi, da selekcija Select [znesek>1000] (Polog) izbere 50% zapisov

Join(Select [znesek>1000] (Polog), Racun, Polog.rid=Racun.rid)

3. naloga (25%)
Kakšen je optimalen plan izvajanja poizvedbe, če ima sistem razpršilne indekse in B+ drevesa. Predstavi celoten postopek in uporabljene algoritme za izvajanje operacij relacijske algebre.

Join(Select [znesek>1000] (Polog), Racun, Polog.rid=Racun.rid)

1.
Na neki fakulteti imajo napreden sistem za beleženje oddajanja nalog študentov. Shema baze je sledeča:

student(ids:int, ime: varchar, priimek: varchar, letnik: int)
oddaja(ids:int, idn: int, datum: date, sprejeto: boolean)
naloga(idn:int, naslov: varchar, predmet: varchar, tocke: int)

a) Z uporabo relacijske algebre napiši poizvedbo za imena vseh študentov 3. letnika, ki so oddali nalogo pri predmetu "Zgodovina" in jim jo je sistem zavrnil (sprejeto ima vrednost false).

b) Z uporabo domensko relacijskega računa napiši poizvedbo za imena študentov, ki so oddali nalogo za več kot 5 točk.

2.
Uporabi shemo prejšnje naloge in napiši SQL stavke za naslednje poizvedbe:
a) Nalogam pri predmetu "Matematika", ki imajo 8 točk, popravi število točk na 10.
b) Zbriši oddaje nalog študentov 1. letnika, ki niso bile sprejete.

c) Izpiši ime in priimek študenta, ki je oddal največ nalog z maksimalnim številom točk.

3.
Podatkovna baza predstavljena v prvi nalogi ima strani velike 4000 zlogov.

|student| = 5.000 zapisov, 110 zlogov, 35 zapisov/stran, 145 strani
|oddaja| = 50.000 zapisov, 30 zlogov, 130 zapisov/stran, 385 strani
|naloga| = 500 zapisov, 110 zlogov, 35 zapisov/stran, 5 strani

Koliko strani prebere naslednje vprašanje?

select[letnik=4](join[student.ids=oddaja.ids](student, oddaja))

Predpostavimo, da ima relacija oddaja razpršilni indeks za atribut ids.

4.
Poišči optimalen plan izvedbe vprašanja:

```
select[letnik=4]( join[oddaja.idn=naloga.idn](
                                join[student.ids=oddaja.ids]( student, oddaja ),
                                naloga ))
```

Utemelji izbiro optimalnega plana.
Opiši plan izvedbe, uporabljene algoritme za izvajanje operacij relacijske algebre in vse predpostavke.

Najpomembnejše tabele v informacijskem sistemu knjižnice so naslednje.

Knjige(kid,avtor,naslov,zalozba,leto);
Clan(cid,ime,priimek,naslov,telefon);
Izposoja(iid,cid,kid,zid,datum);
Zaposleni(zid,ime,priimek,naslov,telefon)

Dani so še naslednji podatki.

1 stran na disku = 8K

|Knjige| = 1.000.000 zapisov, 320 zlogov, 25 zapisov/stran, 40000 strani
|Clan| = 10.000 zapisov, 200 zlogov, 40 zapisov/stran, 250 strani
|Izposoja| = 300.000 zapisov, 40 zlogov, 200 zapisov/stran, 1500 strani
|Zaposleni| = 100 zapisov, 200 zlogov, 40 zapisov/stran, 3 strani

Vse dodatne predpostavke uporabljene v nalogah napišite !

1.
Zapiši SQL stavke za naslednje poizvedbe:

a)izpiši imena članov, ki so si izposodili knjigo, ki jo je napisal avtor z drugo črko imena 'k' pri zaposlenem, ki živi v Kopru (9%)

b)izpiši ime in priimek člana, ki ima izposojenih največ knjig (9%)

c)izpiši avtorja, ki je izdal največ knjig, izmed knjig ki si jih je izposodil Matija (7%)

2.
Z uporabo relacijske algebre zapiši prvo poizvedbo, z domensko relacijskih računom pa drugo poizvedbo:

a)izpiši založbe knjig, ki si jih je izposodil član iz Ljubljane pri zaposlenem
  s telefonsko številko 127434 (12%)

b)izpiši člane, ki so si izposodili vse knjige avtorja Zelen (13%)

3.
Dano je vprašanje zapisano v relacijski algebri:

```
join( select[avtor='Zelen'](Knjige),
      select[datum=2008](Izposoja) )
```

a)Koliko blokov prebere vprašanje? Predpostavi, da ni zgrajen noben indeks.

b)Poišči ekvivalenten izraz v relacijski algebri, ki prebere najmanj blokov. Lahko
  uporabiš poljuben indeks oz. metodo dostopa do relacij. Obrazloži!

SQL v aplikacijski kodi

Iztok Savnik, FAMNIT

Pregled

- SQL v aplikacijski kodi
- Vgnezden SQL
- Kursorji
- Dinamični SQL
- JDBC
- SQLJ
- Shranjene procedure

SQL v aplikacijski kodi

SQL ukazi se lahko prožijo iz programa napisanega v splošnem programskem jeziku (C++ | Java).

- SQL stavki lahko uporabljajo spremenljivke jezika gostitelja.
- Vsebujejo tudi stavke za priklop na podatkovno bazo.

Obstajata dva glavna pristopa k integraciji:

- Vgnezden SQL v gostiteljskem programskem jeziku (Embedded SQL, SQLJ)
- Kreacija posebnega API za klicanje SQL ukazov (JDBC)

SQL v aplikacijski kodi (nadal.)

Problem impedance:

- SQL relacije so (multi-) množice zapisov, ki nimajo določenega maksimalnega št. zapisov.
- Običajno nimamo takšne podatkovne strukture v klasičnih proceduralnih programskih jezikih (imamo sicer knjižnice, npr. STL).
- SQL podpira mehanizem kurzorjev za delo z relacijami.

Vgnezden SQL

Pristop:

Vgnezdi SQL stavke v gostiteljski jezik.

Pred-procesor prevede SQL stavke v posebne API klice.

Potem uporabimo običajen prevajalnik za prevajanje kode .

Gradniki jezika:

Priključitev na SPUB:

EXEC SQL CONNECT

Deklaracija vrednosti:

EXEC SQL BEGIN (END) DECLARE SECTION

Stavki:

EXEC SQL Statement;

Vgnezden SQL: Spremenljivke

```
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20];
long c_sid;
short c_rating;
float c_age;
EXEC SQL END DECLARE SECTION
```

Two special “error” variables:

SQLCODE (long, is negative if an error has occurred)

SQLSTATE (char[6], predefined codes for common errors)

Kurzorji

Definiramo lahko kurzor na relaciji ali poizvedbi (ki generira relacijo).

Lahko odpremo kurzor, preberemo n-terico in potem premaknemo kurzor naprej za eno mesto. Potem spet preberemo n-terico...

Vrstni red n-teric v relaciji, ki jo beremo je določen z SQL vprašanjem.

Lahko tudi spremenimo ali izbrišemo zapis na katerega kaže kurzor.

Primer: Kurzor, ki poišče imena mornarjev, ki so rezervirali rdečo ladjo, v abecednem vrstnem redu

```
EXEC SQL DECLARE sinfo CURSOR FOR
    SELECT S.sname
    FROM Sailors S, Boats B, Reserves R
    WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='red'
    ORDER BY S.sname
```

SQL vgnezden v C: Primer

```
char SQLSTATE[6];
 EXEC SQL BEGIN DECLARE SECTION
 char c_sname[20]; short c_minrating; float c_age;
 EXEC SQL END DECLARE SECTION
 c_minrating = random();
 EXEC SQL DECLARE sinfo CURSOR FOR
   SELECT S.sname, S.age        FROM Sailors S
   WHERE S.rating > :c_minrating
   ORDER BY S.sname;
 do {
   EXEC SQL FETCH sinfo INTO :c_sname, :c_age;
   printf("%s is %d years old\n", c_sname, c_age);
 } while (SQLSTATE != '02000');
 EXEC SQL CLOSE sinfo;
```


Dinamični SQL

SQL vprašanja so znana v času prevajanja.

Dinamični SQL omogoča konstrukcijo SQL stavkov on-the-fly.

Primer:

```
char c_sqlstring[]=
  {"DELETE FROM Sailors WHERE raiting>5"};
EXEC SQL PREPARE readytogo FROM :c_sqlstring;
EXEC SQL EXECUTE readytogo;
```


API alternativa

Namesto da bi spreminjali prevajalnik raje dodamo knjižnico za delo z podatkovno bazo (API).

- Poseben standardiziran vmesnik: procedure/objekti.
- Podamo SQL niz iz jezika SQL stežniku in predstavi rezultate vprašanja v obliki prijazni do jezika.
- Sun JDBC: Java API
- Gonilnik neodvisen od SUPB ujame klice in jih prevede v DBMS-specifično kodo; podatkovna baza je lahko nekje v mreži.

JDBC Arhitektura

Štiri arhitekturne komponente:

- Aplikacija: inicira in zaključi povezavo (connection), pošlje SQL stavke stežniku.
- Upravljalnik gonilnika: nalaganje JDBC gonilnika
- Gonilnik: poveže se na podatkovni vir, prenaša zahteve ter vrača in prevaja rezultate poizvedb ter kode napak.
- Podatkovni vir: preprocesira SQL stavke

JDBC Arhitektura (nadal.)

Štiri vrste gonilnika:

Direktno prevajanje v domač API pod.vira, gonilnik ni v Javi:

Prevede SQL ukaze v domač API od podatkovnega vira. Potrebuje OS-specifičeno binarno kodo na vsakem klientu.

Prevajanje v domač API preko Java gonilnika:

Prevede JDBC klice direktno v omrežni protokol, ki ga uporablja SUPB. Potrebuje Java gonilnik za specifičen SUPB na vsakem klientut.

Most (Bridge):

Prevede SQL ukaze v tuj API. Primer: JDBC-ODBC most. Koda za ODBC in JDBC gonilnik morata biti na klientu.

Omrežni most (Network bridge):

Pošilja ukaze prek omrežja vmesni opremi (middleware), ki komunicira s podatkovnim virom. Potrebuje majhen JDBC gonilnik na vsakem klentu.

JDBC razredi in vmesniki

Koraki pri pošiljanju poizvedbe nad podatkovno bazo:

- Naloži JDBC gonilnik
- Povezovanje s podatkovnim virom
- Izvajaj SQL stavke

Upravljanje JDBC gonilnika

- Vse gonilnike upravlja razred DriverManager
- Nalaganje JDBC gonilnika:

  v Java kodi:

  Class.forName("oracle/jdbc.driver.Oracledriver");

  Ob startu Java aplikacije:

  Djdbc.drivers=<oracle/jdbc.driver

Povezave z JDBC

S podatkovnim virom delamo preko seje (session). Vsaka povezava identificira logično sejo:

- JDBC URL:

  jdbc:<subprotocol>:<otherParameters>

Primer:

```
String url="jdbc:oracle:www.bookstore.com:3083";
Connection con;
try{
   con = DriverManager.getConnection(url,usedId,password);
} catch SQLException excpt { ...}
```

Vmesnik razreda Connection

- Postavi izolacijski nivo trenutne povazave s podatkovno bazo.

public int getTransactionIsolation()
void setTransactionIsolation(int level)

- Specificira če so transakcije v dani povezavi samo-bralne.

public boolean getReadOnly()
void setReadOnly(boolean b)

- Če je postavljen autocommit potem se vsak SQL stavek smatra transakcija.
- Sicer se transakcija zaključi s commit(), ter prekine z rollback().

public boolean getAutoCommit() and void setAutoCommit(boolean b)

- Preveri ali je povezava še odprta.

public boolean isClosed()

Izvajanje SQL stavkov

Imamo tri različne načine za izvajanje SQL stavkov:

Statement (statični in dinamični SQL stavki)
PreparedStatement (semi-statični SQL stavki)
CallableStatment (shranjene procedure)

Razred PreparedStatement:

Vnaprej prevedeni, parametrizirani SQL stavki:
Struktura je fiksna.
Vrednosti parametrov so določene v času izvajanja.

Izvajanje SQL stavkov (nadal.)

```
String sql="INSERT INTO Sailors VALUES(?,?,?,?)";
PreparedStatment pstmt=con.prepareStatement(sql);
pstmt.clearParameters();
pstmt.setInt(1,sid);
pstmt.setString(2,sname);
pstmt.setInt(3, rating);
pstmt.setFloat(4,age);

// we know that no rows are returned, thus we use
   executeUpdate()
int numRows = pstmt.executeUpdate();
```


ResultSets

- PreparedStatement.executeUpdate vrne število spremenjenih zapisov.
- PreparedStatement.executeQuery vrne podatke, ki so shranjeni v objektu ResultSet (ki je kurzor)

```
ResultSet rs=pstmt.executeQuery(sql);
// rs is now a cursor
While (rs.next()) {
  // process the data
}
```

ResultSets (nadaljevanje)

ResultSet je zelo izrazen kurzor:

- previous(): premakne se eno vrstico nazaj
- absolute(int num): premakne na vrstico z določeno številko
- relative (int num): premakne se naprej ali nazaj
- first() in last()

Ujemanje Java in SQL tipov

SQL Type	Java class	ResultSet get method
BIT	Boolean	getBoolean()
CHAR	String	getString()
VARCHAR	String	getString()
DOUBLE	Double	getDouble()
FLOAT	Double	getDouble()
INTEGER	Integer	getInt()
REAL	Double	getFloat()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.TimeStamp	getTimestamp()

JDBC: Izjeme in opozorila

Večino java.sql vrže SQLException če se pojavi napaka.

SQLWarning je podrazred SQLException;
ni tako resna napaka (obstoj moramo testirat eksplicitno)

Opozorila in izjeme (nadal.)

```
try {
  stmt=con.createStatement();
  warning=con.getWarnings();
  while(warning != null) {
    // handle SQLWarnings;
    warning = warning.getNextWarning():
  }
  con.clearWarnings();
  stmt.executeUpdate(queryString);
  warning = con.getWarnings();
  ...
} //end try
catch( SQLException SQLe) {
  // handle the exception
}
```

Meta podatkovna baza

Objekt DatabaseMetaData vsebuje podatke o sistemu za delo s podatkovno bazo in katalog.

```
DatabaseMetaData md = con.getMetaData();
// print information about the driver:
System.out.println(
  “Name:” + md.getDriverName() +
  “version: ” + md.getDriverVersion());
```


Meta podatkovna baza (nadal.)

```
DatabaseMetaData md=con.getMetaData();
ResultSet trs=md.getTables(null,null,null,null);
String tableName;
While(trs.next()) {
  tableName = trs.getString(“TABLE_NAME”);
  System.out.println(“Table: “ + tableName);
  //print all attributes
  ResultSet crs = md.getColumns(null,null,tableName, null);
  while (crs.next()) {
     System.out.println(crs.getString(“COLUMN_NAME” + “, “);
  }
}
```


(ne-) Kompleten primer

```
Connection con = // connect
    DriverManager.getConnection(url, "login", ”pass");
Statement stmt = con.createStatement(); // set up stmt
String query = "SELECT name, rating FROM Sailors";
ResultSet rs = stmt.executeQuery(query);
try { // handle exceptions
   // loop through result tuples
   while (rs.next()) {
      String s = rs.getString(“name");
      Int n = rs.getFloat(“rating");
      System.out.println(s + " " + n);
   }
} catch(SQLException ex) {
   System.out.println(ex.getMessage ()
      + ex.getSQLState () + ex.getErrorCode ());
}
```


SQLJ

- Komplementira JDBC s (semi-)statičnim modelom vprašanja:
- Prevajalnik preveri sintakso, preveri tipe, konsistenco poizvedbe s shemo, ...
- Vsi argumenti so vedno povezani z isto spremenljivko:

```
#sql = {
   SELECT name, rating INTO :name, :rating
   FROM Books WHERE sid = :sid;
```

- Primerjaj z JDBC:

```
sid=rs.getInt(1);
if (sid==1) {sname=rs.getString(2);}
else { sname2=rs.getString(2);}
```

- SQLJ (del SQL standarda) proti vgnezdenem SQL (specifičen)

SQLJ koda

```
Int sid; String name; Int rating;
// named iterator
#sql iterator Sailors(Int sid, String name, Int rating);
Sailors sailors;
// assume that the application sets rating
#sailors = {
    SELECT sid, sname INTO :sid, :name
    FROM Sailors WHERE rating = :rating
};
// retrieve results
while (sailors.next()) {
    System.out.println(sailors.sid + " " + sailors.sname));
}
sailors.close();
```


Shranjene procedure

Kaj so shranjene procedure:

Program, ki se izvaja preko enega samega SQL stavka

Izvaja se kot proces na strežniku

Prednosti:

Lahko vgradimo aplikacijsko logiko in ostanemo blizu podatkov.

Možna ponovna uporaba aplikacijske logike, različni uporabniki.

Izognemo se vračanju zapis-po-zapis preko kursorja.

SQLJ iteratorji

Dva tipa iteratorjev ("kurzorjev"):

- Imenovan iterator

  Potrebuje ime in tip spemenljivke; omogoča branje stolpcev po imenu.

  Poglej primer na prejšnji prosojnici.

- Pozicijski iterator

  Potrebuje samo tip spremenljivke; nato uporablja konstrukt FETCH .. INTO:

```
#sql iterator Sailors(Int, String, Int);
Sailors sailors;
#sailors = ...
while (true) {
    #sql {FETCH :sailors INTO :sid, :name} ;
    if (sailors.endFetch()) { break; }
    // process the sailor
 }
```


Shranjene procedure: Primeri

```
CREATE PROCEDURE ShowNumReservations
  SELECT S.sid, S.sname, COUNT(*)
  FROM Sailors S, Reserves R
  WHERE S.sid = R.sid
  GROUP BY S.sid, S.sname
```

Shranjene procedure imajo lahko parametre:

Tri različne načini: IN, OUT, INOUT

```
CREATE PROCEDURE IncreaseRating(
  IN sailor_sid INTEGER, IN increase INTEGER)
UPDATE Sailors
  SET rating = rating + increase
  WHERE sid = sailor_sid
```

Shranjene procedure: Primeri

Shranjene procedure so lahko napisane v drugih programskih jezikih, ne nujno v SQL:

```
CREATE PROCEDURE TopSailors( IN num INTEGER)
LANGUAGE JAVA
EXTERNAL NAME “file:///c:/storedProcs/rank.jar”
```


Klicanje shranjenih procedur

```
EXEC SQL BEGIN DECLARE SECTION
Int sid;
Int rating;
EXEC SQL END DECLARE SECTION
// now increase the rating of this sailor
EXEC CALL IncreaseRating(:sid,:rating);
```


Klicanje shranjenih procedur

JDBC:

```
CallableStatement cstmt=
  con.prepareCall(“{call
  ShowSailors}});
ResultSet rs =
  cstmt.executeQuery();
while (rs.next()) {
  ...
}
```

SQLJ:

```
#sql iterator
   ShowSailors(...);
ShowSailors showsailors;
#sql showsailors={CALL
   ShowSailors};
while (showsailors.next()) {
   …
}
```


SQL/PSM

- Večina DBMS omogoča uporabnikom pisanje shranjenih procedur v visoko-nivojskem programskem jeziku (blizu SQL).
- SQL/PSM - Persistent Stored Modules, SQL/Foundations

Deklariraj shranjeno proceduro:

```
CREATE PROCEDURE name(p1, p2, ..., pn)
  local variable declarations
  procedure code;
```

Deklariraj funkcijo:

```
CREATE FUNCTION name (p1, ..., pn) RETURNS
  sqlDataType
  local variable declarations
  function code;
```

Osnovni SQL/PSM gradniki

```
CREATE FUNCTION rateSailor
    (IN sailorId INTEGER)
    RETURNS INTEGER
DECLARE rating INTEGER
DECLARE numRes INTEGER
SET numRes = (SELECT COUNT(*)
              FROM Reserves R
              WHERE R.sid = sailorId)
IF (numRes > 10) THEN rating =1;
ELSE rating = 0;
END IF;
RETURN rating;
```


Osnovni SQL/PSM gradniki

- Lokalne spremenljivke (DECLARE)
- RETURN vrednosti za FUNCTION
- Prirejanje vrednosti spremenljivkam SET
- Vejitve in zanke:
  IF (condition) THEN statements;
  ELSEIF (condition) statements;
  ... ELSE statements; END IF;
  LOOP statements; END LOOP
- Poizvedbe so deli izrazov.
- Lahko uporabljamo kurzorje naravno brez “EXEC SQL”

Povzetek

- Vgnezdeni SQL dovoljuje izvajanje parametričnih statičnih poizvedb v gotiteljskem jeziku.
- Dinamični SQL dovoljuje izvajanje ad-hoc poizvedb iz gostiteljskega jezika.
- Mehanizem kurzorjev omogoča branje enega zapisa naenkrat in premosti impedančni problem med gostiteljem in SQL.
- API kot npr. JDBC implementira plast abstrakcije med aplikacijo in DBMS.

Povzetek (nadal.)

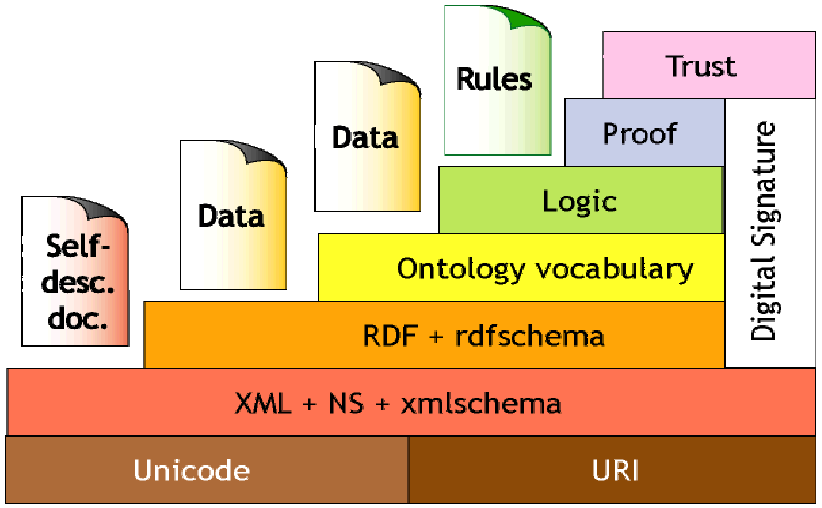
- SQLJ: Statični model, poizvedbe se preverijo v času prevajanja.
- Shranjene procedure izvajajo aplikacijsko logiko direktno na strežniku.
- SQL/PSM standard za pisanje shranjenih procedur.

```
/*

 Program pozenes z:
 java -classpath .:./mysql-connector-java-5.1.7-bin.jar Sqltest

 */

import java.sql.*;

public class Sqltest {


    public static void main(String args[]){

         System.out.println("lalalalal...");
        try {
          Statement stmt;
          Class.forName("com.mysql.jdbc.Driver");
          String url = "jdbc:mysql://localhost/opb_lpp";
          Connection con = DriverManager.getConnection(url,"opb", "opb0809");

          System.out.println("URL: " + url);
          System.out.println("Connection: " + con);

          stmt = con.createStatement();

          //izvedi SQL ukaz, ki ne vrne nicesar
          //stmt.executeUpdate("INSERT INTO PILOTI(ime, priimek, idld)
          //VALUES('janez','cigara', 5)");

          //shrani rezultate SELECT stavka v objekt rs
          ResultSet rs = stmt.executeQuery("SELECT idtl, opis FROM TIP_LETALA");

          //premaknem kurzor na polozaj 24
          rs.absolute(24);

          //izpisi rezultate
          while(rs.next()){
              int idtl= rs.getInt("idtl");
              String opis = rs.getString("opis");
              System.out.println("\tidtl= "+idtl+" \topis= "+opis);
          }

          //izpisi 4. vrstico
          if(rs.absolute(4)){
           int idtl= rs.getInt("idtl");
              String opis = rs.getString("opis");
              System.out.println("\tidtl= "+idtl+" \topis= "+opis);
          }//end if

          //izpisi prvo vrstico
          if(rs.first()){
           int idtl= rs.getInt("idtl");
              String opis = rs.getString("opis");
              System.out.println("\tidtl= "+idtl+" \topis= "+opis);
          }//end if

          //izpisi zadnjo vrstico
          if(rs.last()){
           int idtl= rs.getInt("idtl");
              String opis = rs.getString("opis");
              System.out.println("\tidtl= "+idtl+" \topis= "+opis);
          }//end if

            con.close();

          }catch( Exception e ) {
              e.printStackTrace();
          }

    }
}
```

Semantične tehnologije in spletne aplikacije

Iztok Savnik

Stolp semantičnega spleta



Sintaksa in semantika

- Sintaksa: struktura podatkov
- Semantika: pomen podatkov
- Dva pogoja potrebna za skupno delo:
  - Skupna sintaksa: programi lahko razčlenjujejo podatke
  - Skupen način razumevanja pomena: programi lahko uporabljajo podatke.

XML

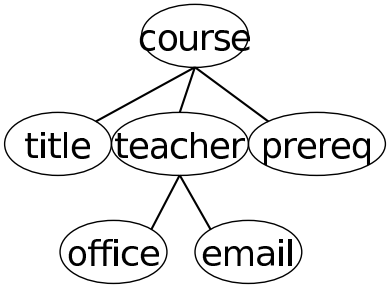
- XML: eXtensible Mark-up Language
- XML dokumenti so napisani z uporabniško definiranimi značkami
- Oznake so uporabljene za izražanje "pomena" delov podatkov

XML

- XML: dokument = označeno drevo
- vozlišče = oznaka + atributi/vrednosti + vsebina

```
<course date="...">
  <title>...</title>
  <teacher>
    <office>...</office>
    <email>...</email>
  </teacher>
  <room>...</room>
  <prereq>...</prereq>
</course>
```

=



XML

- XML Schema = slovnica za opis dreves in podatkovnih tipov
- Lahko uporabljamo XML za predstavitev semantike?

XML in pomen

```
<Predator>
     …
</Predator>
```

- Predator: srednje višine, velika vztrajnost, letalna naprava.
- Predator: tisti, ki terorizira, razbija in uničuje še posebno za neko korist.
- Predator: organizem, ki živi na osnovi ulova drugih organizmov.
- ...

Omejitve pri opisu pomena

- XML ne opisuje:
  1. Besednjak specifičen za neko domeno
  2. Ontološki primitivi za modeliranje podatkov
- Zahteva dogovor glede 1 in 2
- Uporabna rešitev za lokalne projekte:
  - Agenti v manjšem stabilnem okolju
  - Strani na manjšem in stabilnem intranet
- Ni primerno za predstavitev spletnih virov

RDF

- RDF je podatkovni model
  - Model je neodvisen od domene, aplikacije in se lahko internacionalizira
  - Model lahko vidimo kot usmerjen, označen graf ali kot objekten model (objekti/atributi/vrednosti)
- RDF podatkovni model je abstrakten, konceptualni nivo neodvisen od XML
  - XML je lahko sintaksa za RDF in ne del RDF
  - RDF podatki se lahko sploh ne pojavijo v XML obliki

Asociativni model

- Splošen model
- RDF – Resource Description Language
  - RDF je podatkovni model !
  - Opisi temeljijo na trojicah: asociacije
- Lahko opišemo kompleksne vrednosti
  - Kolekcije, vreče, sestavljene objekte, …

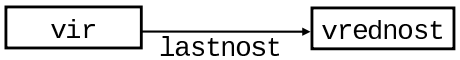
RDF model

RDF model = množica RDF trojic

trojica = izraz (stavek)

`(subjekt, predikiat, objekt)`

- subjekt = vir
- predikat = lastnost (vira)
- objekt = vrednost (lastnosti)

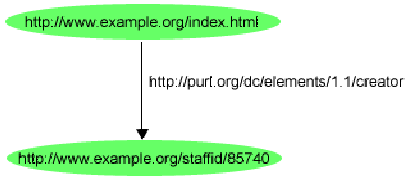


`http://www.example.org/index.html` has a **creator** whose value is **`John Smith`**

RDF model

- Subjekt: `http://www.example.org/index.html`
- Predikat: `http://purl.org/dc/elements/1.1/creator`
- Objekt: `http://www.example.org/staffid/85740`

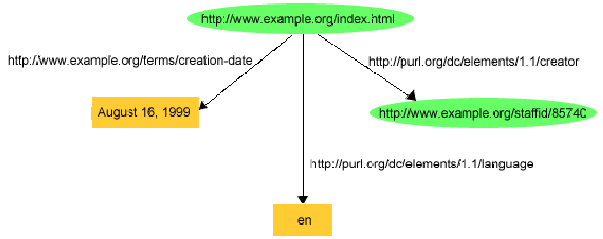
- Vsaka trojica ustreza eni povezavi v grafu



Primer RDF opisa

```
<http://www.example.org/index.html> <http://purl.org/dc/elements/1.1/creator> <http://www.example.org/staffid/85740> .
<http://www.example.org/index.html> <http://www.example.org/terms/creation-date> "August 16, 1999" .
<http://www.example.org/index.html> <http://purl.org/dc/elements/1.1/language> "en" .
```

- Uporabljamo polne poti !
- Koncepti so ovali
- Literali so pravokotniki



RDF sintaksa

RDF model = graf z označenimi povezavami
= množica trojic

- Grafična notacija (graf)
- Notacija (neformalna) na osnovi trojic, npr.:
  `(subject, predicate, object)`
- Formalna sintaksa:
  - N3 notacija
  - Konkretna sintaksa: RDF/XML

Prostori imen

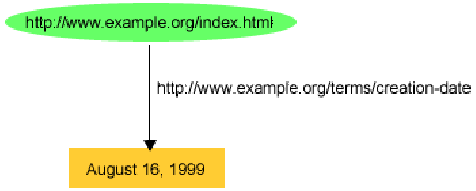
- Uporabljali bomo okrajšana imena URL naslovov
- Imena so definirana na določenih URL-jih
- Imena, ki jih uporabljamo so pripone
- Qname prefiksi:

```
prefix rdf:, namespace URI: http://www.w3.org/1999/02/22-rdf-syntax-ns#
prefix rdfs:, namespace URI: http://www.w3.org/2000/01/rdf-schema#
prefix dc:, namespace URI: http://purl.org/dc/elements/1.1/
prefix owl:, namespace URI: http://www.w3.org/2002/07/owl#
prefix ex:, namespace URI: http://www.example.org/ (or http://www.example.com/)
prefix xsd:, namespace URI: http://www.w3.org/2001/XMLSchema#
```

- Poglejmo si zdaj okrajšan zapis primerov

En stavek

```
ex:index.html exterms:creation-date "August 16, 1999" .
```




```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:exterms="http://www.example.org/terms/">

<rdf:Description rdf:about="http://www.example.org/index.html">
     <exterms:creation-date>August 16, 1999</exterms:creation-date>
</rdf:Description>

</rdf:RDF>
```

Dva stavka

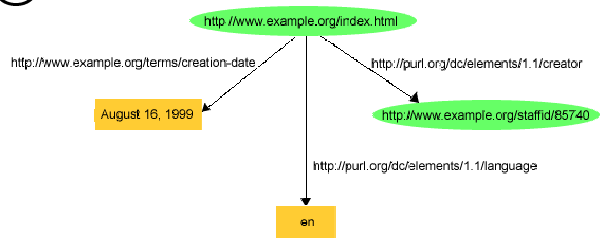
```
ex:index.html exterms:creation-date "August 16, 1999" .
ex:index.html dc:language "en" .
```

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:exterms="http://www.example.org/terms/">

<rdf:Description rdf:about="http://www.example.org/index.html">
<exterms:creation-date>August 16, 1999</exterms:creation-date>
</rdf:Description>

<rdf:Description rdf:about="http://www.example.org/index.html">
<dc:language>en</dc:language>
</rdf:Description>

</rdf:RDF>
```



```
ex:index.html dc:creator exstaff:85740 .
ex:index.html exterms:creation-date "August 16, 1999" .
ex:index.html dc:language "en" .
```

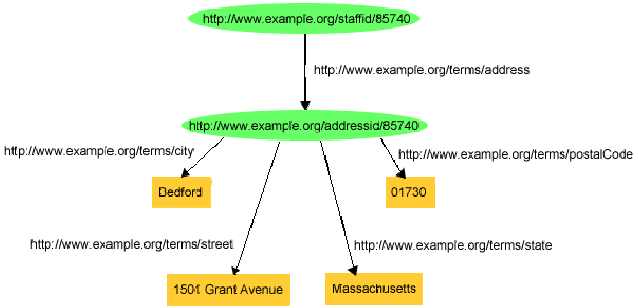
Trije stavki ☺




```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                xmlns:dc="http://purl.org/dc/elements/1.1/"
                xmlns:exterms="http://www.example.org/terms/">

<rdf:Description rdf:about="http://www.example.org/index.html">
   <exterms:creation-date>August 16, 1999</exterms:creation-date>
   <dc:language>en</dc:language>
   <dc:creator rdf:resource="http://www.example.org/staffid/85740"/>
</rdf:Description>

</rdf:RDF>
```

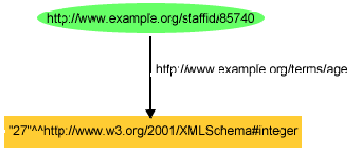

Strukturirane vrednosti

```
exstaff:85740 exterms:address exaddressid:85740 .
exaddressid:85740 exterms:street "1501 Grant Avenue" .
exaddressid:85740 exterms:city "Bedford" .
exaddressid:85740 exterms:state "Massachusetts" .
exaddressid:85740 exterms:postalCode "01730" .
```



Tip skalarja

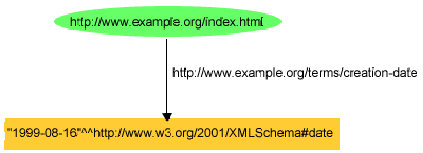
- Skalarju definiramo tip
- 27 pomeni celo število in ne znaka “2” in “7”




```
exstaff:85740 exterms:age "27"^^xsd:integer .

<http://www.example.org/staffid/85740> <http://www.example.org/terms/age>
   "27"^^<http://www.w3.org/2001/XMLSchema#integer> .
```

- RDF nima vgrajenih tipov
- Tipi so definirani izven RDF: datatype URI
- XML Schema enostavni tipi
- `xsd:integer, xsd:float, xsd:double, xsd:boolean, xsd:boolean, xsd: date, …`

Tip skalarja




```
ex:index.html exterms:creation-date "1999-08-16"^^xsd:date .
```

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:exterms="http://www.example.org/terms/">

<rdf:Description rdf:about="http://www.example.org/index.html">
   <exterms:creation-date rdf:datatype=
   "http://www.w3.org/2001/XMLSchema#date">1999-08-16 </exterms:creation-
   date>
</rdf:Description>

</rdf:RDF>
```


Uporaba ID

- Do zdaj smo uporabljali `rdf:about` za specifikacijo objekta, ki ga opisujemo
  - Opisovanje virov
- Včasih želimo opisati objekt, ki ga ni moč opisati z URI referenco
  - Primer: katalog objektov na določenem naslovu
- Podobno XML ID: unikaten znotraj osnovnega URI

Primer

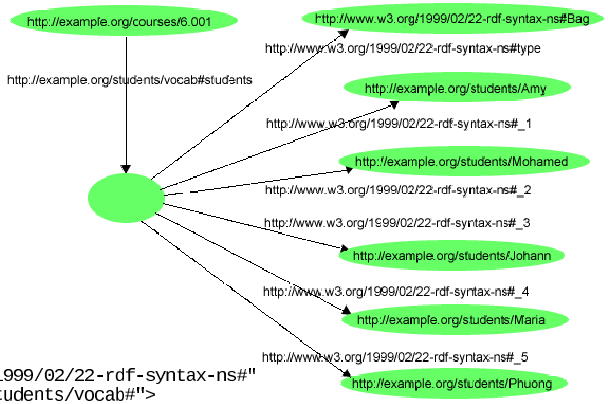
- Kolekcija objektov na naslovu: `http://www.example.com/2007/04/products`

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:exterms="http://www.example.com/terms/">

<rdf:Description rdf:ID="item10245">
    <exterms:model rdf:datatype="&xsd;string">Overnighter</exterms:model>
    <exterms:sleeps rdf:datatype="&xsd;integer">2</exterms:sleeps>
    <exterms:weight rdf:datatype="&xsd;decimal">2.4</exterms:weight>
    <exterms:packedSize rdf:datatype="&xsd;integer">784</exterms:packedSize>
 </rdf:Description>

...other product descriptions...

</rdf:RDF>
```


Kontejnerji

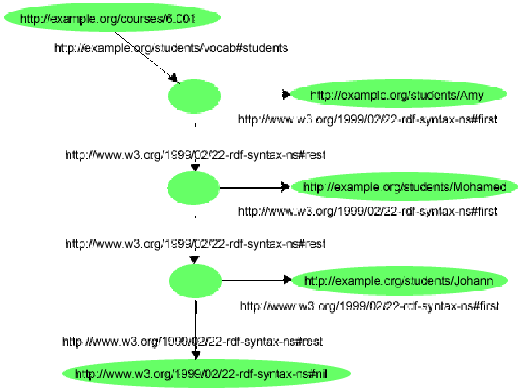
- Kontejnerji omogočajo grupiranje virov (ali besed)
- Napišemo lahko izjave o kontejnerju (kot celota) ali individualno o njegovih članih
- Poznamo različne tipe kontejnerjev
  - Vreča (rdf:Bag) – neurejena kolekcija
  - Zaporedje (rdf:Seq) – urejena kolekcija (= “sekvenca”)
  - Alternative (rdf:Alt) – predstavlja alternative
- Možno je tudi kreirati kolekcije na osnovi vzorcev URI
- Dvojniki so dovoljeni (ni mehanizma za zagotavljanje unikatnosti vrednosti)

Vreča




```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:s="http://example.org/students/vocab#">

<rdf:Description rdf:about="http://example.org/courses/6.001">
    <s:students>
     <rdf:Bag>
             <rdf:li rdf:resource="http://example.org/students/Amy"/>
             <rdf:li rdf:resource="http://example.org/students/Mohamed"/>
             <rdf:li rdf:resource="http://example.org/students/Johann"/>
                  <rdf:li rdf:resource="http://example.org/students/Maria"/>
                  <rdf:li rdf:resource="http://example.org/students/Phuong"/>
     </rdf:Bag>
    </s:students>
</rdf:Description>

</rdf:RDF>
```


Kolekcija




```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:s="http://example.org/students/vocab#">

<rdf:Description rdf:about="http://example.org/courses/6.001">
   <s:students rdf:parseType="Collection">
    <rdf:Description rdf:about="http://example.org/students/Amy"/>
    <rdf:Description rdf:about="http://example.org/students/Mohamed"/>
    <rdf:Description rdf:about="http://example.org/students/Johann"/>
   </s:students>
</rdf:Description>

</rdf:RDF>
```


RDF shema

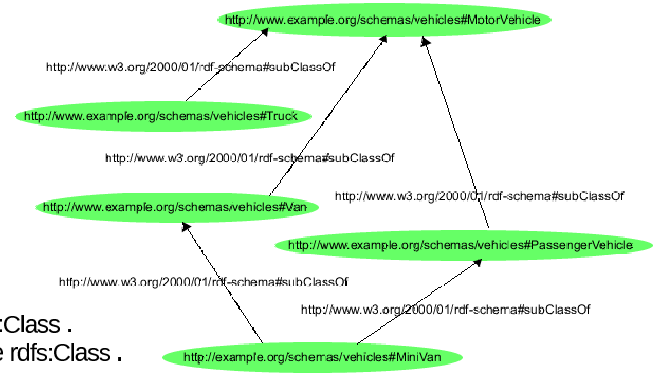
RDFS = RDF shema

- Definira majhen slovar za RDF:
  - `rdfs:class, rdfs:subClassOf, rdfs:type`
  - `rdfs:property, rdfs:subPropertyOf`
  - `rdfs:domain, rdfs:range`
- Ustreza množici RDF predikatov:

  ⇒meta-nivo

  ⇒poseben vnaprej definiran pomen

Razredi



ex:MotorVehicle rdf:type rdfs:Class .
ex:PassengerVehicle rdf:type rdfs:Class .
ex:Van rdf:type rdfs:Class .
ex:Truck rdf:type rdfs:Class .
ex:MiniVan rdf:type rdfs:Class .

ex:PassengerVehicle rdfs:subClassOf ex:MotorVehicle .
ex:Van rdfs:subClassOf ex:MotorVehicle .
ex:Truck rdfs:subClassOf ex:MotorVehicle .

ex:MiniVan rdfs:subClassOf ex:Van .
ex:MiniVan rdfs:subClassOf ex:PassengerVehicle .

XML/RDF hierarchija vozil

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd http://www.w3.org/2001/XMLSchema#">]>

<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xml:base="http://example.org/schemas/vehicles">

<rdf:Description rdf:ID="MotorVehicle">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>

<rdf:Description rdf:ID="PassengerVehicle">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>

<rdf:Description rdf:ID="Truck">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>

<rdf:Description rdf:ID="Van">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>

<rdf:Description rdf:ID="MiniVan">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="#Van"/>
    <rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
</rdf:Description>

</rdf:RDF>
```


RDFS lastnosti

Slovar za lastnosti:

- RDF razred:
  - `rdfs:Property` – definicija lastnosti
- RDF schema lastnosti:
  - `rdfs:domain` – definicija prve komponente lastnosti
  - `rdfs:range` – definicija druge komponente lastnosti
  - `rdfs:subPropertyOf` - izraža ISA relacijo med lastnostmi

Opisovanje lastnosti

- Opisujemo lastnosti razredov podobno kot razrede v programskih jezikih
- Vse lastnosti so definirane kot primerki `rdf:Property`
- Ni omejitev glede številnosti lastnosti!
- Pri definiciji zaloge vrednosti lahko uporabljamo tudi osnovne tipe npr. `xsd:integer`

Primer

- Lastnost `ex:author` ima domeno `ex:Book` in zalogo vrednosti razred `ex:Person`

```
ex:Book rdf:type rdfs:Class .
ex:Person rdf:type rdfs:Class .
ex:author rdf:type rdf:Property .
ex:author rdfs:domain ex:Book .
ex:author rdfs:range ex:Person .
```

- Lastnost ima lahko več kot eno zalogo vrednosti:

```
ex:hasMother rdfs:range ex:Female .
ex:hasMother rdfs:range ex:Person .
```

Lastnosti vozil

- Avto je registrirala oseba:

```
<rdf:Property rdf:ID="registeredTo">
   <rdfs:domain rdf:resource="#MotorVehicle"/>
   <rdfs:range rdf:resource="#Person"/>
</rdf:Property>
```

- Razdalja med sedeži:

```
<rdf:Property rdf:ID="rearSeatLegRoom">
   <rdfs:domain rdf:resource="#PassengerVehicle"/>
   <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>
```


Pod-lastnosti

- Primarni voznik je podlastnost voznika:

```
ex:driver rdf:type rdf:Property .
ex:primaryDriver rdf:type rdf:Property .
ex:primaryDriver rdfs:subPropertyOf ex:driver .

<rdf:Property rdf:ID="driver">
   <rdfs:domain rdf:resource="#MotorVehicle"/>
</rdf:Property>

<rdf:Property rdf:ID="primaryDriver">
   <rdfs:subPropertyOf rdf:resource="#driver"/>
</rdf:Property>
```

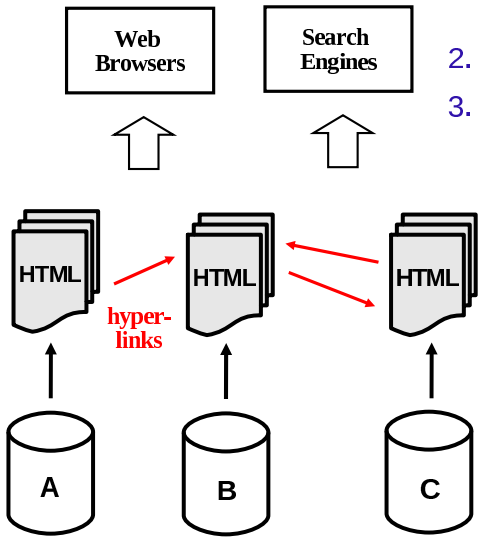

Linked data

1. Od spleta dokukemntov do spleta podatkov
   - Spletni API-ji in Linked Data
2. Linked Data implementacija na spletu
   - Kateri podatki obstajajo?
3. Aplikacije
   - Kaj se dogaja s podatki?
4. Naslednji koraki
   - Kaj manjka?

Klasični splet

En sam globalen inform. prostor

1. URL za:
   - Globalni unikatni IDs
   - Poizvedovalni mehanizmi
2. HTML kot skupna oblika vsebine
3. Hyper-povezave



Problem in rešitev

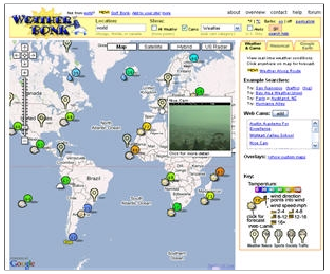
Problem

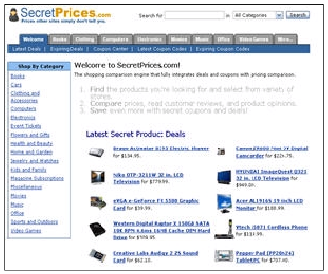
Ker je vsebina spleta zelo šibko strukturirana, aplikacije težko implementirajo pametne operacije.

Rešitev

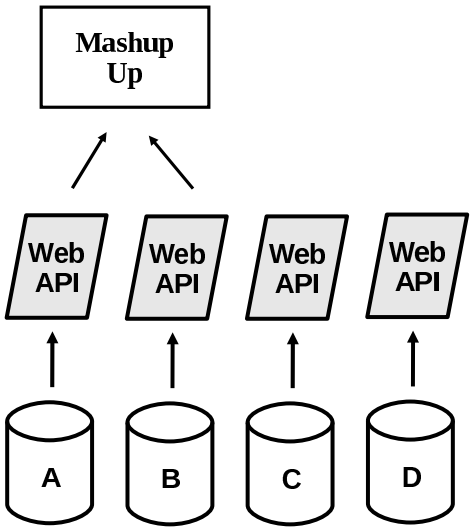
Povečaj strukturo vsebine spleta.

Spletni API-ji in prepletene storitve





Spletni API-ji in mashup-i



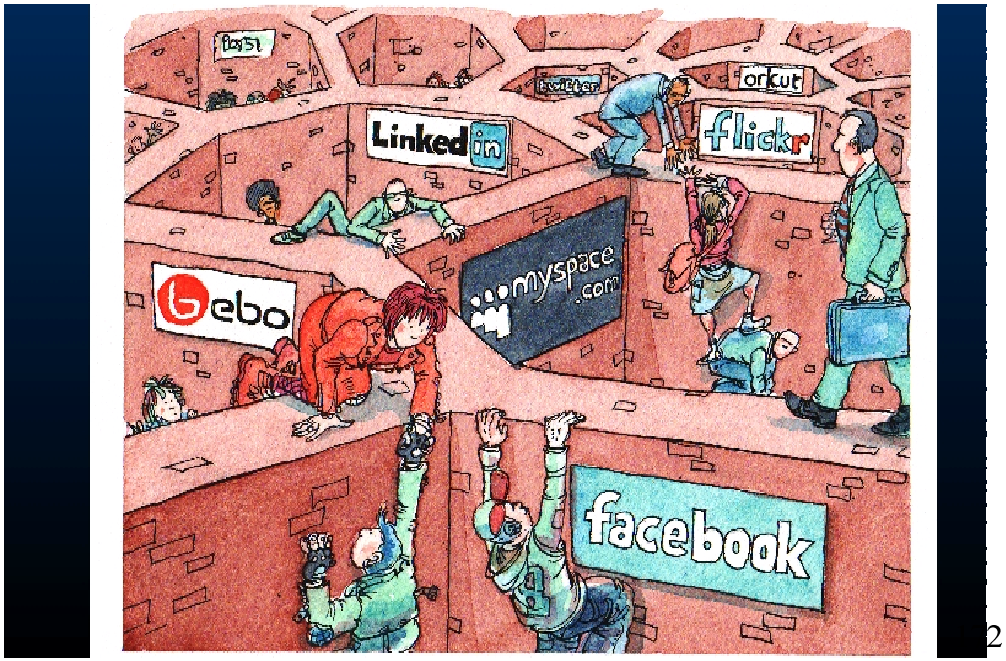
Slabe lastnosti:

0) API-ji nudijo privatne vmesnike.

1) Mashup-i temeljijo na fiksni množici podatkovnih virov.

2) Ne moremo definirati povezav med podatkovnimi objekti.

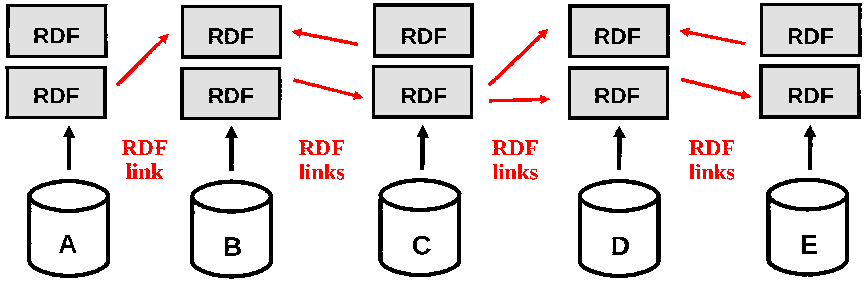
Spletni API-ji razdelijo splet na vrtove z ograjami



Povezani podatki

Uporabi tehnologije semantičnega spleta za:

- Publiciranje strukturiranih podatkov na splet,
- Definicijo povezav med podatki iz enega spletnega mesta s podatki v drugih spletnih mestih.



Principi povezanih podatkov

1. Uporabi URI-je kot imena za objekte
2. Uporabi HTTP URI, da lahko ljudje dostopajo do teh imen
3. Ko nekdo dostopa do URI zagotovi uporabne RDF podatke
4. Vključi RDF stavke drugih URI, da lahko ljudje odkrijejo sorodne objekte.

Tim Berners-Lee 2007

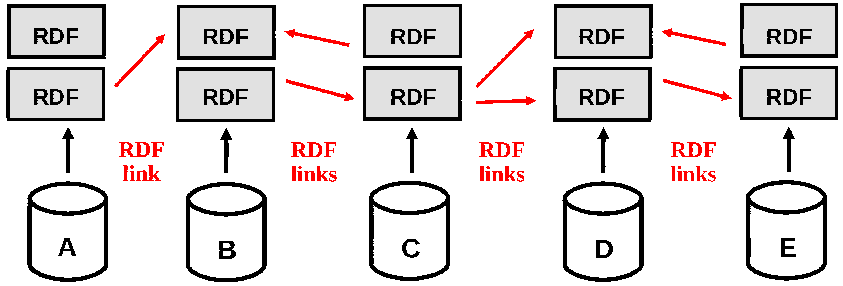
http://www.w3.org/DesignIssues/LinkedData.html

Lastnosti spleta povezanih podatkov

- Kdorkoli lahko publicira podatke na spletu povezanih podatkov
- Entitete so povezane s povezavami
  - Kreacija globalnega podatkovnega grafa, ki povezuje podatkovne vire in omogoča odkrivanje novih virov.
- Podatki so samo-opisni
  - Če aplikacija dobi podatke, ki so predstavljeni z nepoznanim besednjakom, mora aplikacija identificirati URI-je, ki identificirajo slovarje z definicijami RDFS in OWL izrazov.
- Splet podatkov je odprt
  - To pomeni, da lahko aplikacije odkrivajo nove podatkovne vire v času izvajanja.

Implementacija povezanih podatkov na spletu

- **Je to realno?**

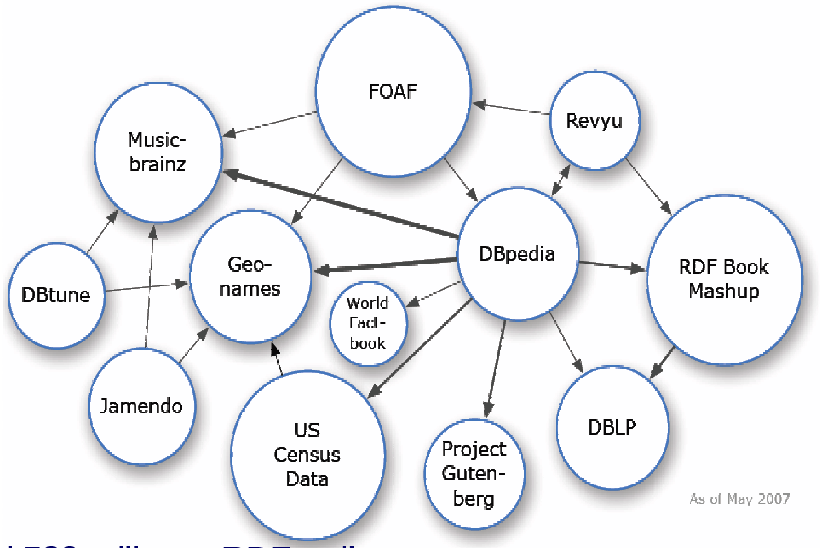


W3C Linking Open Data

LINKINGOPENDATA
W3C SWEO Community Project

- **Skupnost:**
  - Publiciranje obstoječih odprtih podatkovnih zbirk na spletu.
  - Povezovanje stvari med različnimi podatkovnimi viri

LOD zbirke na spletu: Maj 2007



- Nad 500 miljonov RDF trojic
- Okoli 120,000 RDF povezav med pod.viri

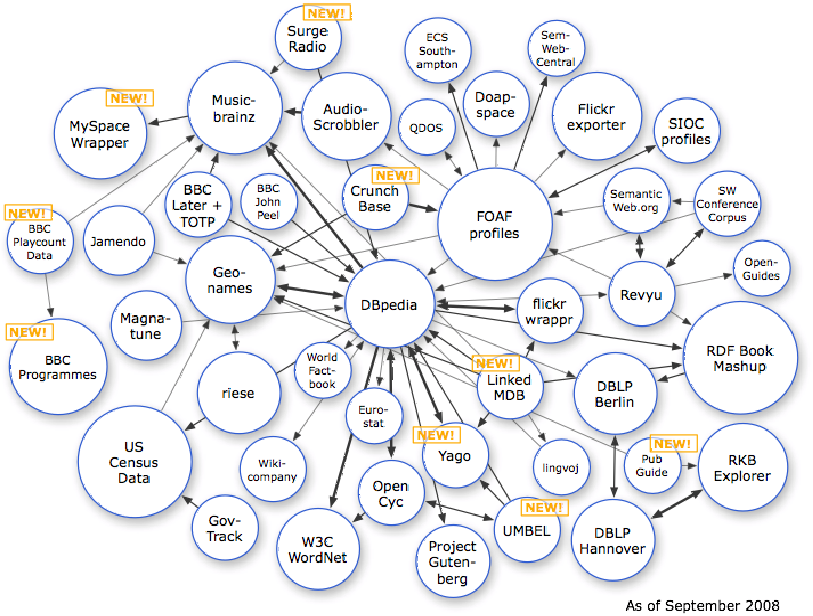
Primer RDF povezave

- RDF povezave iz DBpedia na druge podatkovne vire

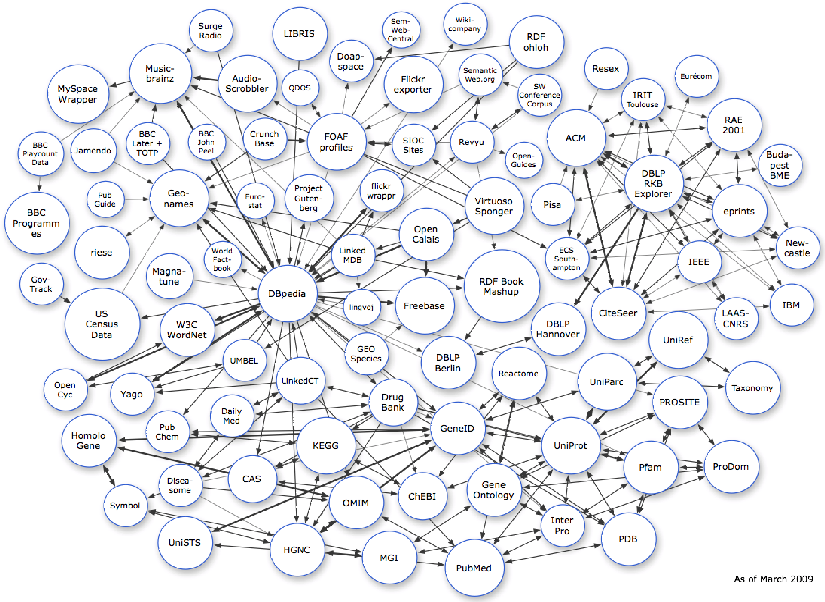
```
<http://dbpedia.org/resource/Berlin> owl:sameAs
<http://sws.geonames.org/2950159> .
```

```
<http://dbpedia.org/resource/Tim_Berners-Lee>
owl:sameAs <http://www4.wiwiss.fu-
berlin.de/dblp/resource/person/100007> .
```

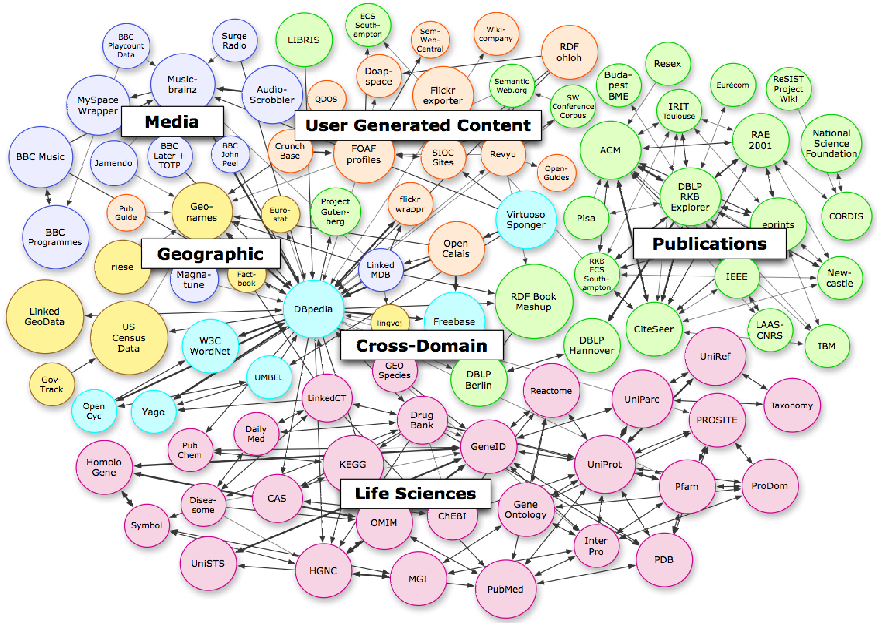
LOD zbirke na spletu: September 2008



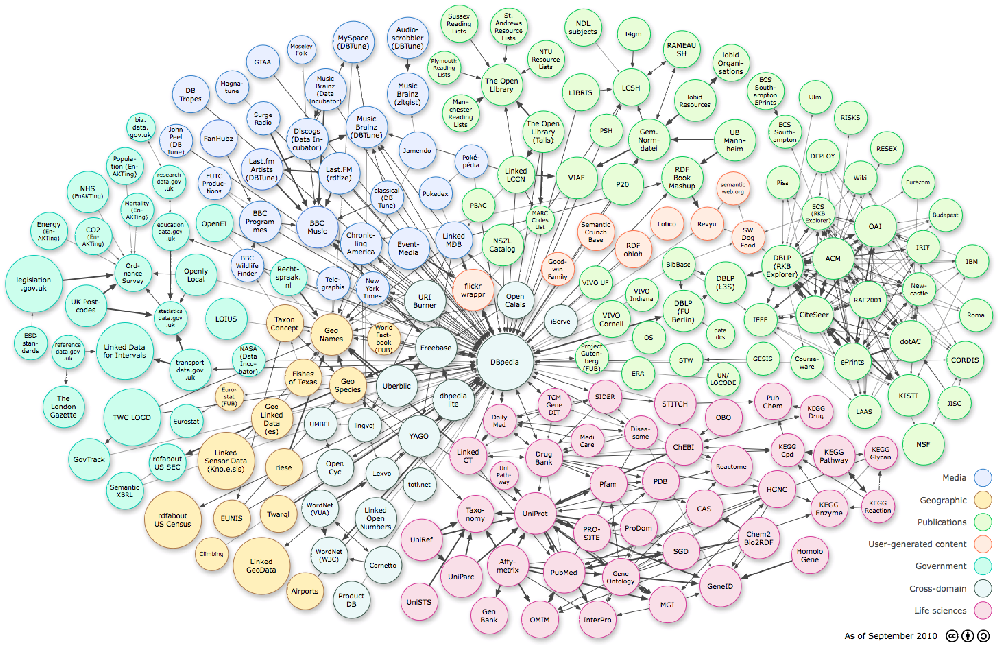
LOD zbirke na spletu: Marec 2009



LOD zbirke na spletu: Julij 2009



LOD zbirke na spletu: Sept 2010



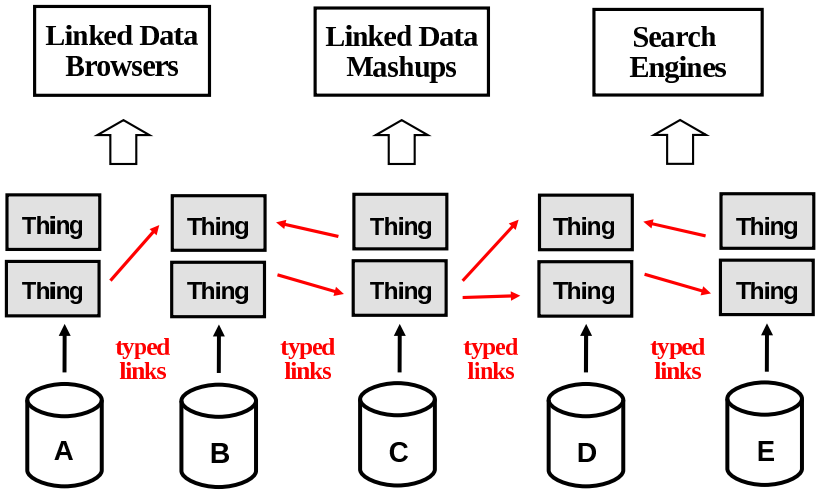
LOD statistika: Julij 2009

Domain	***No of Triples***	***% of Cloud***	***No of Links***	***% of Links***
Media	698.000.000	10,4%	1.238.000	0,8%
Publications	212.000.000	3,2%	4.922.000	3,3%
Life Sciences	2.429.000.000	36,1%	133.199.000	89,4%
Geographic Data	3.097.000.000	46,0%	4.038.000	2,7%
User Generate Content	76.000.000	1,1%	1.559.000	1,0%
Cross-Domain	214.000.000	3,2%	3.992.000	2,7%
Total	***6.726.000.000***		***148.948.000***	

+ 2 billion triples from Data.gov.

Aplikacije

- Kaj lahko naredimo s tem?

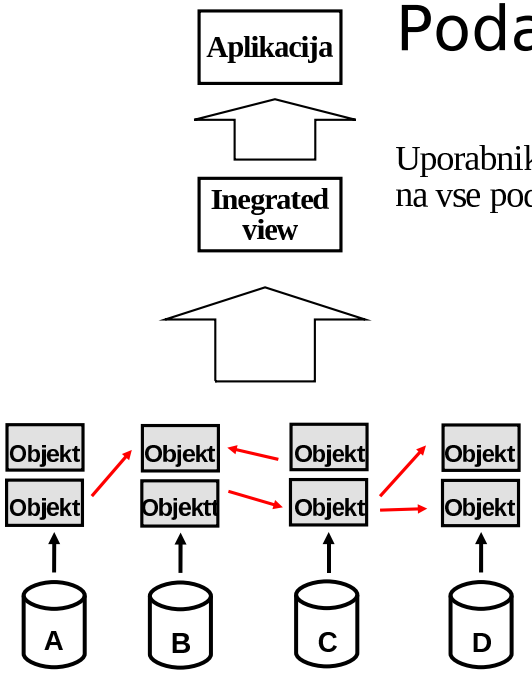


Brskalnik za povezane podatke

- Tabulator Browser (MIT, USA)
- Marbles (FU Berlin, DE)
- OpenLink RDF Browser (OpenLink, UK)
- Zitgist RDF Browser (Zitgist, USA)
- Humboldt (HP Labs, UK)
- Disco Hyperdata Browser (FU Berlin, DE)
- Fenfire (DERI, Irland)

Povezani podatki v prepletenih storitvah (meshups)

Aplikacije s specifično domeno, ki uporabljajo povezane podatke iz spleta



Podatkovna fuzija

Uporabnik želi imeti integriran pogled na vse podatke, ki so dostopni za objekt.

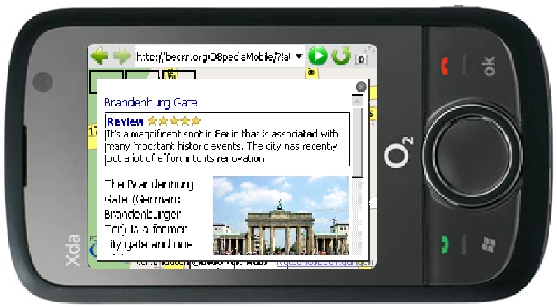
Znani problemi:

Preslikave shem
Reševanje nekonsistentnosti

DBpedia Mobile

Geospatial
vhodna točka
na splet
podatkov

Uporablja:
DBpedia,
Revyu in Flickr
podatke



Podatkovni splet in iskalniki

- Falcons (IWS, China)
  - http://iws.seu.edu.cn/services/falcons/objectsearch/index.jsp
- Sig.ma (DERI, Ireland)
  - http://www.deri.ie/
- Swoogle (UMBC, USA)
- VisiNav (DERI, Ireland)
- Watson (Open University, UK)