Podatkovne baze I in Osnove podatkovnih baz

dr. Rok Rupnik

Splošne informacije

- Priporočena literatura
  - [1] Raghu Ramakrishnan, Johannes Gehrke (2003). Database Management Systems, Third Edition, McGraw-Hill
  - [2] Thomas M. Connolly, Carolyn E. Begg (2005). Database Systems, A Practical Approach to Design, Implementation and Management, Fourth Edition, Addison-Wesley
  - [3] Ramez Elmasri, Shamkant B. Navathe (2003). Fundamentals of Database Systems, Fourth Edition, Addison-Wesley
  - [4] Tomaž Mohorič (2002). Podatkovne baze, Založba Bi-TIM.
  - [5] Peter Rob, Carlos Coronel (2005). Database Systems: Design, Implementation and Management, Sixth Edition, Addison Wesley.

Citiranje: glej [4,15-20] = glej v knjigi T. Mohorič, strani od 15 do 20.

List za študente

- Seminarske naloge
- Izpitni režim
- Kontaktni mail: rok.rupnik@fri.uni-lj.si

Poglavje 1

Uvod v podatkovne baze

Povzeto po [2]

Primeri uporabe podatkovne baze

- **Nakupovanje v trgovini**
- **Plačilo s plačilno/kreditno kartico**
- **Rezervacija počitniškega aranžmaja pri agenciji**
- **Dejavnosti v knjižnici**
- **Področje zavarovalništva**
- **Izposoja filmov (DVD ali VHS)**
- **Spletne trgovine in spletne strani**
- **Študijska informatika**

Opozorilo

- Podatkovno bazo in podatke v njen uporabljamo bodisi preko aplikacij, bodisi preko posebnih orodij
- Pri našem predmetu se osredotočamo na podatkovno bazo

Datotečni sistemi

- **Včasih so aplikacije uporabljale datotečne sisteme**
- **Vsaka aplikacija je uporabljala svoje podatke**
- **Praviloma je ena datoteka predstavljala eno tabelo**

Arhitektura aplikacij, ki uporabljajo datotečne sisteme

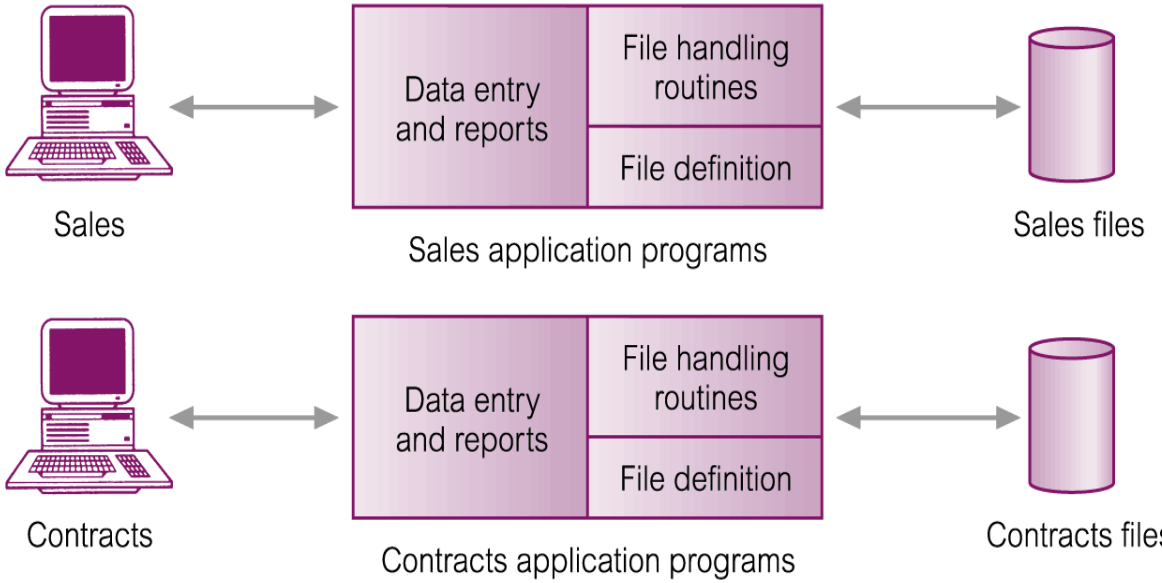


Figure 1.5 File-based processing.

Sales Files

PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo)
PrivateOwner (ownerNo, fName, lName, address, telNo)
Client (clientNo, fName, lName, address, telNo, prefType, maxRent)

Contracts Files

Lease (leaseNo, propertyNo, clientNo, rent, paymentMethod, deposit, paid, rentStart, rentFinish, duration)
PropertyForRent (propertyNo, street, city, postcode, rent)
Client (clientNo, fName, lName, address, telNo)

Omejitve datotečnih sistemov..

- **Ločevanje in izolacija podatkov**
  - **Vsaka aplikacija upravlja s svojimi podatki**
  - **Neka aplikacija uporablja lahko le svoje podatke oz. uporabniki ene aplikacije ne poznajo razpoložljivih podatkov in njihovega pomena v drugih aplikacijah**
- **Dupliciranje podatkov**
  - **En in isti podatek je potrebno vnesti in vzdrževati preko več aplikacij**
  - **Odvečna poraba prostora na disku ter potencialna možnost za nekonsistentnost**
  - **Različni formati podatkov**

Omejitve datotečnih sistemov

- **Podatkovna odvisnost**
  - **Struktura je definirana v aplikaciji, sprememba v strukturi podatkov zahteva spremembo v aplikaciji**
- **Nekompatibilni formati datotek**
  - **Programi, napisani v različnih programskih jezikih ne morejo enostavno dostopati do datotek drugih sistemov**

Pristop uporabe podatkovnih baz

- **Se je pojavil:**
  - **Zaradi težav oz. neučinkovitosti zaradi dejstva, ker so bile podatkovne strukture definirane v aplikacijah, namesto da bi bile definirane in shranjene ločeno od aplikacij**
  - **Zaradi dejstva, da datotečni sistem ne omogoča nadzora nad dostopom in upravljanja s podatki drugače, kot preko aplikacij**
- **Posledica:**
  - **Pojavili so se SUPB**

Podatkovna baza

- **PB je zbirka (lahko) medsebojno logično povezanih podatkov (in opisov podatkov), ki zadovoljujejo informacijske potrebe organizacije in njenih poslovnih procesov**
- **Sistemski katalog (meta podatki) vsebuje opise podatkovnih struktur in omogoča fizično podatkovno neodvisnost**
- **Logična povezanost podatkov je določena s konceptualnim podatkovnim modelom (entitetami, razmerji med njimi in atributi)**

Sistem za upravljanje podatkovnih baz (SUPB)..

- **Skupek programske opreme, ki omogoča kreiranje, vzdrževanje in nadzor nad dostopom do podatkov v podatkovni bazi:**
  - **Kreiranje podatkovnih struktur je omogočeno preko DDL (*Data Definition Language*)**
  - **Vzdrževanje podatkov (Create, Insert, Update, Delete) pa preko DML (*Data Manipulation Language*)**
  - **Povpraševalni jeziki (*query language*)**

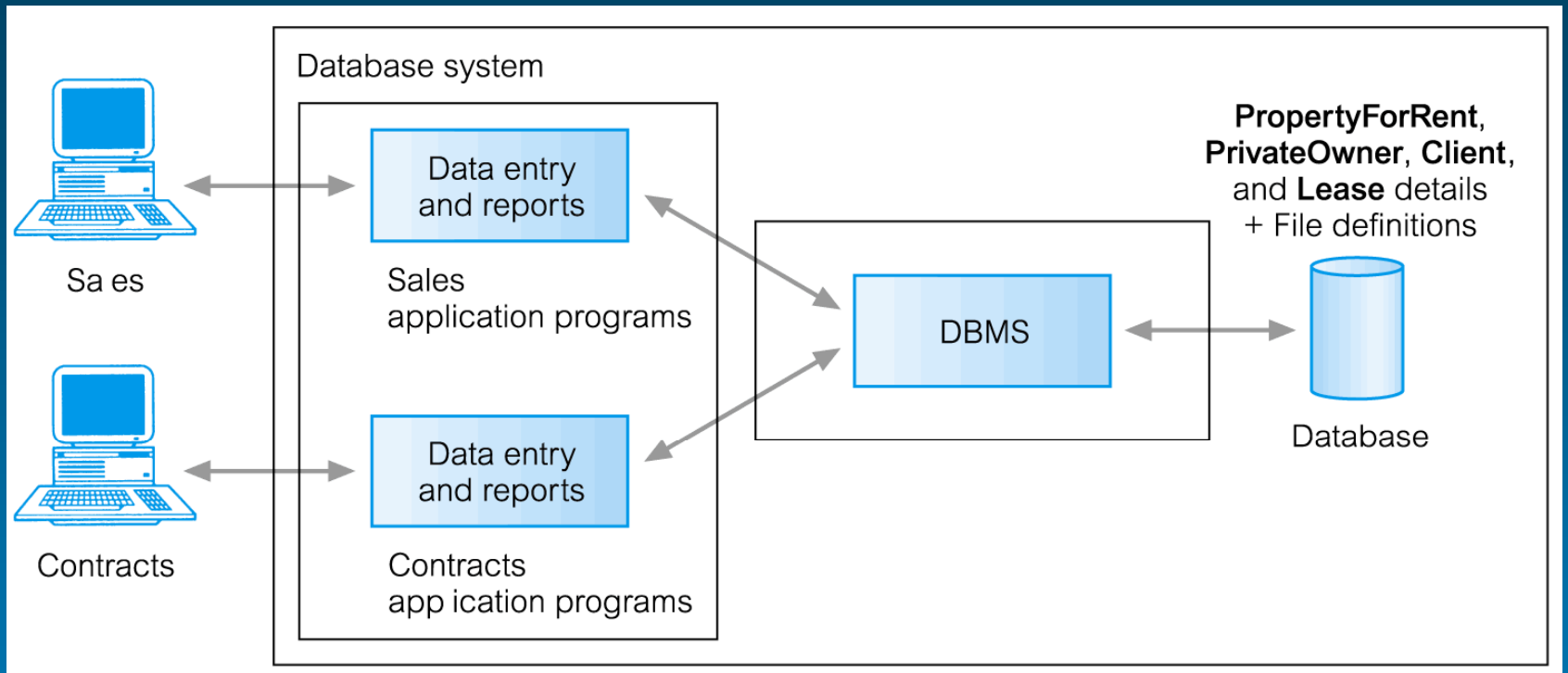
Sistem za upravljanje podatkovnih baz (SUPB)..

- **DDL:**
  - **Omogoča definiranje podatkovnih struktur, podatkovnih tipov ter integritetnih omejitev**
  - **Vse specifikacije oz. definicije so shranjene v podatkovni bazi (podatkovni slovar, sistemski katalog). Kreiranje tabele pomeni poseg v sistemski katalog**

Sistem za upravljanje podatkovnih baz (SUPB)..

- **Nadzor nad dostopom do podatkov obsega več področij:**
  - **Sistem varnosti:** dostop do podatkov v skladu z avtorizacijo
  - **Sistem nadzora integritete:** zagotavlja integriteto (smiselno vsebino, konsistenco) podatkov
  - **Sistem nadzora sočasnega dostopa**
  - **Sistem obnove podatkovne baze (*recovery*)**
  - **Sistemski katalog (*data dictionary*)**
- **Aplikacija (aplikacijski sistem, aplikativni sistem): program, ki v okviru informacijskega sistema delno ali v celoti podpira enega ali več poslovnih procesov in za shranjevanje podatkov uporablja podatkovno bazo**

Sistem za upravljanje podatkovnih baz (SUPB)



PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo)
PrivateOwner (ownerNo, fName, lName, address, telNo)
Client (clientNo, fName, lName, address, telNo, prefType, maxRent)
Lease (leaseNo, propertyNo, clientNo, paymentMethod, deposit, paid, rentStart, rentF nish)

Zgodovina SUPB

- **Prva generacija**
  - **Hierarhične in mrežne PB**
- **Druga generacija**
  - **Relacijske PB**
- **Tretja generacija**
  - **Objektno-relacijske PB**
  - **Objektne PB**

Zgodovina shranjevanja podatkov..

- zgodnja 60':Charles Bachman iz General Electric-a je razvil prvi splošno – namenski SUPB (Integrated Data Store).
  - Predstavljal je osnovo za mrežni podatkovni model.
  - Imel velik vpliv na SUPB-je v 60' letih.
- pozna 60': IBM razvije Information Management System (IMS), ki se uporablja še danes.
  - Predstavljal je osnovo za hierarhični podatkovni model.
  - American Airlines in IBM razvijeta sistem SABRE za letalske rezervacije, ki je več uporabnikom omogočal dostop do istih podatkov preko mreže.
  - ZANIMIVOST: isti SABRE se še danes uporablja za spletni servis Travelocity.

Zgodovina shranjevanja podatkov..

- 70': Edgar Codd predlaga relacijski podatkovni mode (IBM).
  - Razvije se obilo relacijskih PB.
  - Podatkovne baze postanejo akademsko področje.
  - Relacijske podatkovne baze postanejo standard za upravljanje s podatki.
- 80': Relacijski model si še utrdi položaj, kot SUPB.
  - SQL postane standardni jezik za izvajanje poizvedb v relacijski PB.
  - SQL je bil standardiziran v poznih 80' – SQL-92.
  - Standard prevzamejo American National Standard Institute (ANSI) in International Standards Organization (ISO).
  - Pojavi se programiranje transakcij.

Zgodovina shranjevanja podatkov..

- pozna 80’ in 90’: veliko napredka na podpodročjih PB.
  - Veliko raziskav se opravi na področju poizvedovalnih jezikov in bogatejših podatkovnih modelov.
  - Velik poudarek na kompleksnih analizah podatkov iz vseh področij organizacijskih sistemov.
  - Veliko proizvajalcev SUPB-jev (IBM – DB2, Orale 8, Informix UDS) razširi svoje sisteme s podporo novim podatkovnim tipom: slike, text in s podporo kompleksnejšim poizvedbam.
  - Razvijejo se podatkovna skladišča, ki združujejo (povzemajo) podatke iz večih PB in omogočajo izvajanje specializiranih analiz (iskanje zakonitosti v podatkih).

Zgodovina shranjevanja podatkov..

- Pojavi se zlivanje različnih paketov za enterprise resource planning (ERP) in management resource planning (MRP).
- Primeri takih paketov:
  - Baan,
  - Oracle,
  - PeopleSoft,
  - SAP in
  - Siebel.
- Vsi omenjeni paketi omogočajo: upravljanje s premoženjem, planiranje človeških virov in finančne analize.
- Podatki so v teh paketih shranjeni v relacijskih PB, praviloma podpirajo uporabo vseh pomembnejših SUPB.

Zgodovina shranjevanja podatkov

- Naslednjo stopnjo razvoja predstavlja vstop SUPB-jev v svet Interneta.
  - Prva generacija spletih strani je shranjevala podatke v datotekah OS.
  - Uporaba PB za shranjevanje podatkov, ki so dostopni preko Interneta, postaja vsakdanja.
    - Poizvedbe se generira preko spletnih form, odgovore pa se nazaj posreduje v obliki jezika HTML, za lažji prikaz v spletnem brskalniku.
  - Vsi proizvajalci dodajajo svojim SUPB-jem možnosti za čim lažjo uporabo v spletu.
- State of the Art:
  - multimediske podatkovne baze,
  - PB za interaktivni video,
  - digitalne knjižnice,...
- Raziskovanje in obvladovanje področja se nedvomno izplača!!!

Poglavje 2

Relacijska algebra in relacijski račun

Povzeto po [2]

O relacijskih poizvedovalnih jezikih

- **Relacijska algebra** in **relacijski račun** sta formalna jezika povezana z relacijskim modelom
- Neformalno je relacijska algebra visoko-nivojski **postopkovni jezik**, relacijski račun pa nepostopkovni ali **deklarativni jezik**
- Formalno sta ekvivalentna
- Vsak jezik, s katerim lahko pridobimo relacije, ki jih je moč pridobiti z relacijskim računom, je **relacijsko popoln** (relationally complete)

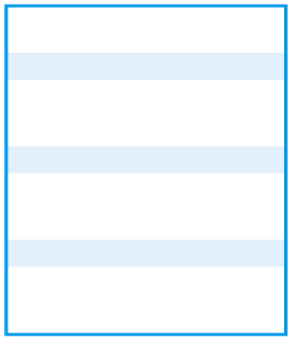
Relacijska algebra..

- Operacije relacijske algebre se izvedejo na eni ali več relacij, z namenom, da bi pridobili novo relacijo. Pri tem se osnovna relacija ne spremeni
- Tako operandi kot tudi rezultat so relacije → izhod ene operacije je lahko vhod v drugo
- Omogoča **gnezdenje izrazov** – tako kot velja za aritmetične izraze. Tej lastnosti jezika pravimo **zaprtje** (closure)

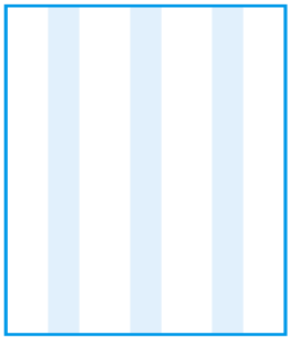
Relacijska algebra

- Relacijska algebra vsebuje pet **osnovnih operacij**:
  - Selekcija,
  - Projekcija,
  - Kartezijski produkt,
  - Unija in
  - Razlika.
- Z uporabo teh operacij se izvede večina poizvedb
- Možne so tudi **izpeljane operacije**:
  - Stik,
  - Presek in
  - Količnik (razlika).

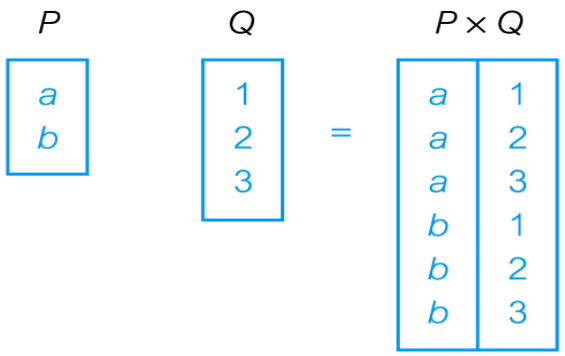
Operacije relacijske algebre..



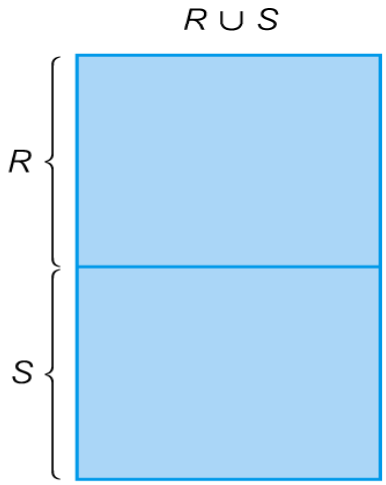
(a) Selection



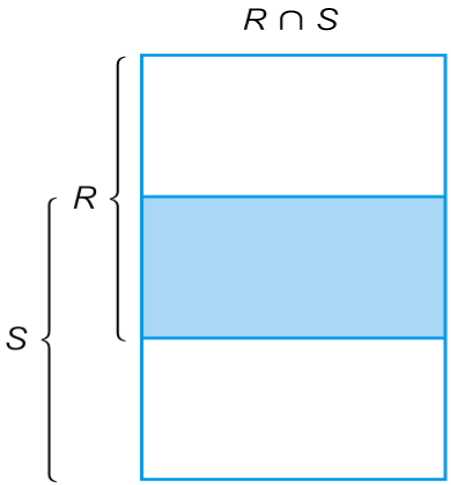
(b) Projection



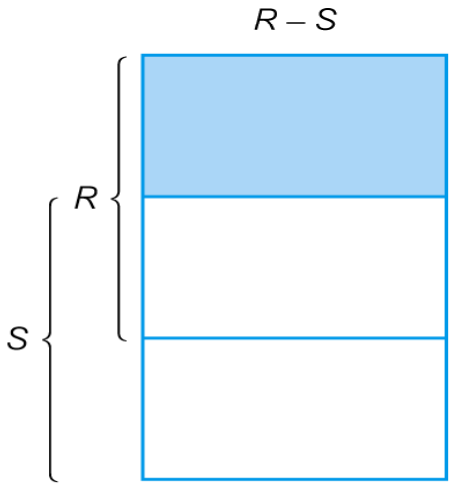
(c) Cartesian product



(d) Union

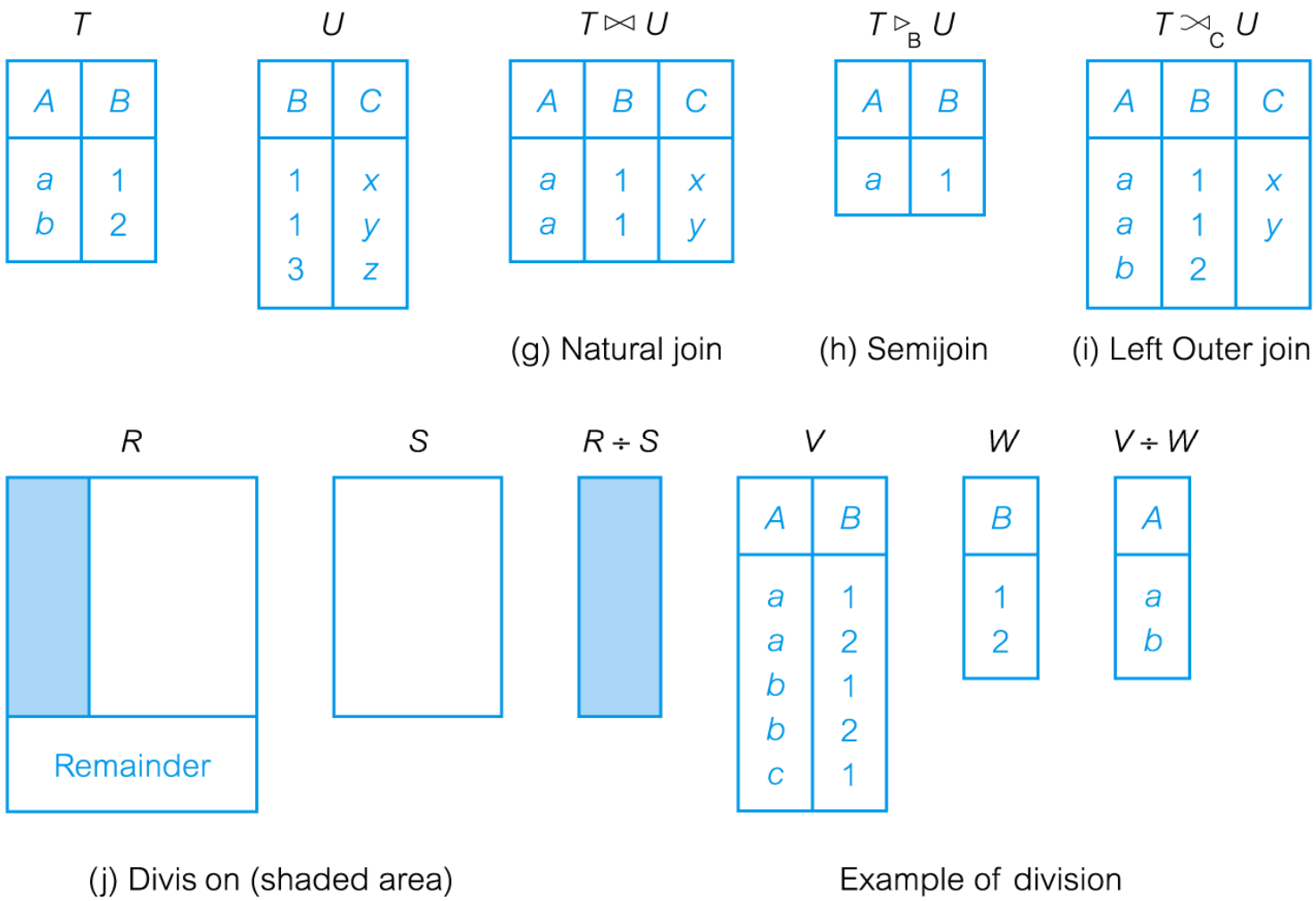


(e) Intersection



(f) Set difference

Operacije relacijske algebre



Selekcija

- $\sigma_{predikat}$ (R)
- Deluje na enojni relaciji R
- Vrne relacijo, ki vsebuje samo tiste n-terice (vrstice) iz relacije R, ki zadoščajo pogoju, ki ga določa **predikat**

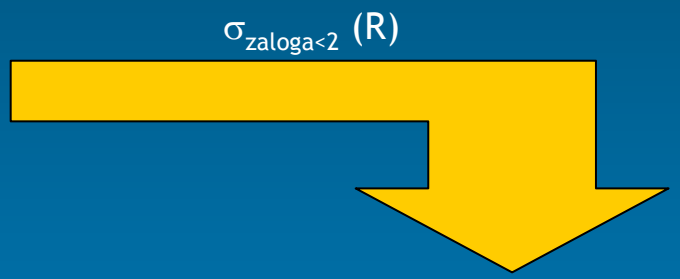
Primer selekcije

- Izpiši vse artikle z zalogo manjšo od 2

$\sigma_{zaloga<2}$ (R)

R=ARTIKEL

Šifra	Naziv	Zaloga
A10	Telovadni copati Nike	10
A12	Trenerka Bali	4
BC80	Moška jakna QuickSilver	1
X12	Ženska jakna QuickSilver	0



Šifra	Naziv	Zaloga
BC80	Moška jakna QuickSilver	1
X12	Ženska jakna QuickSilver	0

Projekcija

- $\Pi_{s1, \ldots, sn}(R)$
- Deluje na enojni relaciji R
- Vrne relacijo, ki vsebuje samo tiste atribute (stolpce), ki so določeni s predikatom
- Operacija eliminira duplikate (rezultat je relacija, ki po definiciji ne vsebuje duplikatov)

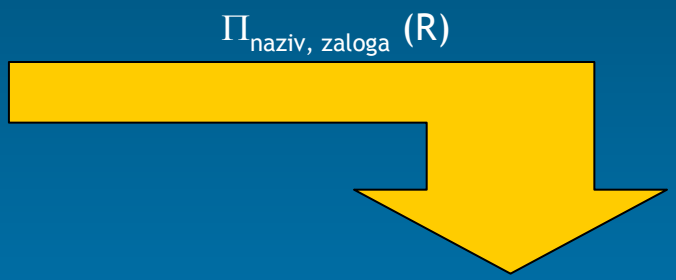
Primer projekcije

- Izpiši nazive in zalogo vseh artiklov

$\Pi_{naziv,\ zaloga}\ (R)$

R=ARTIKEL

Šifra	Naziv	Zaloga
A10	Telovadni copati Nike	10
A12	Trenerka Bali	4
BC80	Moška jakna QuickSilver	1
X12	Ženska jakna QuickSilver	0



Naziv	Zaloga
Telovadni copati Nike	10
Trenerka Bali	4
Moška jakna QuickSilver	1
Ženska jakna QuickSilver	0

Unija

- $R \cup S$
- Unija dveh relacij R in S je relacija, ki vsebuje vse n-terice (vrstice) relacije R in relacije S
- Operacija eliminira duplikate

- R in S se morata ujemati po atributih.
  - R(ime: text, EMSO: int) $\cup$ S (priimek: text, starost: int)
- Če ima relacija R I n-teric in relacija S J n-teric, potem njuna unija predstavlja združitev v eno relacijo z največ I+J n-teric

Primer unije

- Izpiši vsa mesta, kjer se nahajajo skladišča ali stranke

$\Pi_{kraj}(R) \cup \Pi_{kraj}(S)$

R=ARTIKEL

Šifra	Naziv	Kraj
A10	Telovadni copati Nike	LJ
A12	Trenerka Bali	MB
BC80	Moška jakna QuickSilver	LJ
X12	Ženska jakna QuickSilver	GO

S=STRANKA

Šifra	Naziv	Kraj
S1	Novak Janez	LJ
S2	Krašna Miha	CE
S3	Bele Simon	PO
S4	Šuc Vilma	GO

$\Pi_{kraj}(R) \cup \Pi_{kraj}(S)$

Kraj
LJ
CE
PO
GO
MB

Razlika

- R – S
- Razlika med relacijama R in S (R-S) vrne relacijo, ki vsebuje samo tiste n-terice (vrstice), ki so v R in jih ni v S
- R in S se morata ujemati po atributih
  - ~~R(ime: text, EMSO: int) S (priimek: text, starost: int)~~

Primer razlike

- Izpiši vsa mesta, kjer se nahajajo skladišča ne pa tudi stranke

Π_{kraj} (R) – Π_{kraj} (S)

R=ARTIKEL

Šifra	Naziv	Kraj
A10	Telovadni copati Nike	LJ
A12	Trenerka Bali	MB
BC80	Moška jakna QuickSilver	LJ
X12	Ženska jakna QuickSilver	GO

S=STRANKA

Šifra	Naziv	Kraj
S1	Novak Janez	LJ
S2	Krašna Miha	CE
S3	Bele Simon	PO
S4	Šuc Vilma	GO

Π_{kraj} (R) – Π_{kraj} (S)

Kraj
MB

Presek

- $R \cap S$
- Presek med relacijama R in S ($R \cap S$) vrne relacijo, ki vsebuje tiste n-terice (vrstice), ki se nahajajo v obeh relacijah

- R in S se morata ujemati po atributih
  - ~~R(ime: text, EMSO: int) S (priimek: text, starost: int)~~
- Presek lahko izpeljemo iz osnovnih operacij:

  $R \cap S = R - (R - S)$

Primer preseka

- Izpiši vsa mesta, kjer se nahajajo tako skladišča kot stranke

$\Pi_{kraj}(R) \cap \Pi_{kraj}(S)$

R=ARTIKEL

Šifra	Naziv	Kraj
A10	Telovadni copati Nike	LJ
A12	Trenerka Bali	MB
BC80	Moška jakna QuickSilver	LJ
X12	Ženska jakna QuickSilver	GO

S=STRANKA

Šifra	Naziv	Kraj
S1	Novak Janez	LJ
S2	Krašna Miha	CE
S3	Bele Simon	PO
S4	Šuc Vilma	GO

$\Pi_{kraj}(R) \cap \Pi_{kraj}(S)$

Kraj
LJ
GO

Kartezijski produkt

- $R \times S$
- Kartezijski produkt relacij R in S vrne vse možne kombinacije med n-tericami (vrsticami) relacije R in n-tericami (vrsticami) relacije S

Primer kartezijskega produkta

- Izpiši šifre, nazive in količino artiklov, ki se pojavljajo na računih

$(\Pi_{\text{šifra, naziv}}(R)) \times (\Pi_{\text{šifra artikla, količina}}(S))$

R=ARTIKEL

Šifra	Naziv	Zaloga
A10	Telovadni copati Nike	10
A12	Trenerka Bali	4
BC80	Moška jakna QuickSilver	1
X12	Ženska jakna QuickSilver	0

S=RAČUN

Račun	Šifra artikla	Količina
15/05	A10	1
15/05	X12	1

Šifra	Naziv	Šifra artikla	Količina
A10	Telovadni copati Nike	A10	1
A10	Telovadni copati Nike	X12	1
A12	Trenerka Bali	A10	1
A12	Trenerka Bali	X12	1
BC80	Moška jakna QuickSilver	A10	1
BC80	Moška jakna QuickSilver	X12	1
X12	Ženska jakna QuickSilver	A10	1
X12	Ženska jakna QuickSilver	X12	1

Kartezijski produkt s selekcijo

- S selekcijo lahko **omejimo kartezijski produkt**
- Izpiši šifre, nazive in količino artiklov, ki se pojavljajo na računih, kjer je šifra artikla na računu enaka šifri artikla v artiklu

$\sigma_{\text{R.šifra = S.šifra artikla}} ((\Pi_{\text{šifra, naziv}} (\text{R})) \times (\Pi_{\text{šifra artikla, količina}} (\text{S})))$

Primer kartezijskega produkta s selekcijo

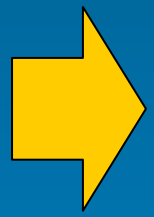
- $\sigma_{R.\text{šifra} = S.\text{šifra artikla}} ((\Pi_{\text{šifra, naziv}} (R)) \times (\Pi_{\text{šifra artikla, količina}} (S)))$

R=ARTIKEL

Šifra	Naziv	Zaloga
A10	Telovadni copati Nike	10
A12	Trenerka Bali	4
BC80	Moška jakna QuickSilver	1
X12	Ženska jakna QuickSilver	0

S=RAČUN

Račun	Šifra artikla	Količina
15/05	A10	1
15/05	X12	1



Šifra	Naziv	Šifra artikla	Količina
A10	Telovadni copati Nike	A10	1
X12	Ženska jakna QuickSilver	X12	1

Stične operacije..

- Kartezijski produkt s selekcijo združimo v eno operacijo, ki jo imenujemo **stik**

- Stik je **ena najtežjih operacij** s stališča implementacije v relacijskih SUPB; eden ključnih krivcev za probleme z učinkovitostjo

Stične operacije

- Obstaja več vrst stičnih operacij:
  - Stik Theta (Theta join)
  - Stik Equijoin (Equijoin) je poseben primer stika Theta
  - Naravni stik (Natural join)
  - Odprti stik (Outer join)

Stik Theta (θ stik)..

- $R \bowtie_F S$
- Stik Theta med relacijama R in S vrne n-terice (vrstice), ki zadoščajo predikatu F kartezijskega produkta R in S
- Predikat F je oblike R.ai θ S.bi, kjer je θ aritmetična operacija ($<$, $\leq$, $>$, $\geq$, $=$, $\neq$).

Stik Theta (θ stik)

- Theta stik lahko izpeljemo s pomočjo selekcije in kartezijskega produkta:
- $R \bowtie_F S = \sigma_F(R \times S)$

- Stopnja Theta stika med R in S je seštevek stopenj operandov relacij R in S. Če predikat F vsebuje zgolj enakost (=), gre za stik tipa **Equijoin**

Primer stika Equijoin

- Izpiši šifre, nazive in količino artiklov, ki se pojavljajo na računih, kjer je šifra artikla na računu enaka šifri artikla v artiklu

$(\Pi_{\text{šifra, naziv}}\ (R)) \bowtie_{R.\text{šifra} = S.\text{šifra artikla}} (\Pi_{\text{šifra artikla, količina}}\ (S))$

R=ARTIKEL

Šifra	Naziv	Zaloga
A10	Telovadni copati Nike	10
A12	Trenerka Bali	4
BC80	Moška jakna QuickSilver	1
X12	Ženska jakna QuickSilver	0

S=RAČUN

Račun	Šifra artikla	Količina
15/05	A10	1
15/05	X12	1

Šifra	Naziv	Šifra artikla	Količina
A10	Telovadni copati Nike	A10	1
X12	Ženska jakna QuickSilver	X12	1

Naravni stik

- $R \bowtie S$
- Naravni stik relacij R in S je posebna vrsta stika Equijoin-a prek enakosti skupnih atributov relacij R in S
- Pri vsakem stiku se vzame le en primerek skupnega atributa

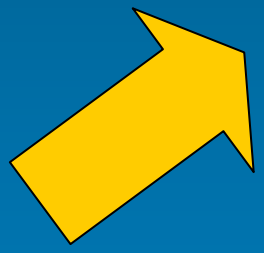
Primer naravnega stika

- Izpiši šifre, nazive in količino artiklov, ki se pojavljajo na računih, kjer je šifra artikla na računu enaka šifri artikla v artiklu

$(\Pi_{\text{šifra, naziv}}(R)) \bowtie (\Pi_{\text{šifra, količina}}(S))$

R=ARTIKEL

Šifra	Naziv	Zaloga
A10	Telovadni copati Nike	10
A12	Trenerka Bali	4
BC80	Moška jakna QuickSilver	1
X12	Ženska jakna QuickSilver	0



S=RAČUN

Račun	Šifra	Količina
15/05	A10	1
15/05	X12	1

Šifra	Naziv	Količina
A10	Telovadni copati Nike	1
X12	Ženska jakna QuickSilver	1

Zunanji stik

- Zunanji stik nam omogoča, da prikažemo n-terice (vrstice), ki nimajo vrednosti v stičnem atributu (stolpcu)
- $R \bowtie S$
- Obstajata **levo odprti** in **desno odprti stik**
- Levo odprti stik med relacijama R in S je stik, kjer so n-terice relacije R, ki nimajo para v S z enakim stičnim atributom, vključene v rezultat

Primer zunanjega stika

- Izpiši osebe in njihova začasna prebivališča

$(\Pi_{ID,\ Priimek\ in\ ime,\ PTT}\ (R)) \overset{\leftarrow}{\bowtie} S$

R=OSEBA

ID	Priimek in ime	PTT
1	Kante Janez	5270
2	Tratnik Jože	5000
3	Mali Mihael	
4	Brecelj Jana	1000

S=KRAJ

PTT	Naziv
1000	Ljubljana
5000	Nova Gorica
5270	Ajdovščina

$(\Pi_{ID,\ Priimek\ in\ ime,\ PTT}(R)) \overset{\leftarrow}{\bowtie} S$

Priimek in ime	PTT	Naziv zač. preb.
Kante Janez	5270	Ajdovščina
Tratnik Jože	5000	Nova Gorica
Mali Mihael		
Brecelj Jana	1000	Ljubljana

Poglavje 3

SQL in QBE

Povzeto po [2]

Uvod v SQL..

- SQL je transformacijsko usmerjen jezik, ki ga sestavljata dve skupini ukazov:
  - Skupina ukazov DDL (Data Definition Language) za opredelitev strukture podatkovne baze in
  - Skupina ukazov DML (Data Manipulation Language) za poizvedovanje in ažuriranje podatkov.

- SQL do izdaje SQL:1999 ne vključuje ukazov kontrolnega toka. Kontrolni tok je bil potrebno obvladati s programskim jezikom ali interaktivno z odločitvami uporabnikov.

Uvod v SQL

- Lastnosti SQL:
  - Enostaven;
  - Nepostopkoven (kaj in ne kako);
  - Uporaben v okviru številnih vlog: skrbniki PB, vodstvo, razvijalci informacijskih rešitev, končni uporabniki;
  - Obstaja ISO standard za SQL;
  - SQL de-facto in tudi uradno standardni jezik za delo z relacijskimi podatkovnimi bazami.

Zgodovina SQL..

- 1974, D. Chamberlin (IBM San Jose Laboratory) definira jezik, ki ga poimenuje ‘Structured English Query Language’ (SEQUEL).
- 1976, Prenovljena različica, SEQUEL/2. Ime se sčasoma spremeni v SQL.
- IBM izdela prototipni SUPB System R, ki temelji na SEQUEL/2.
- Korenine SQL so v SQUARE (Specifying Queries as Relational Expressions) segajo v čas pred projektom System R.

Zgodovina SQL

- Pozna 70-a, pojavi se prvi komercialni SUPB, ki temelji na SQL.
- 1987, ANSI in ISO izdata prvi standard za SQL.
- 1989, ISO objavi dodatek, s katerim opredeli mehanizme za doseganje integritete podatkov (Integrity Enhancement Feature).
- 1992, prva večja revizija ISO standarda (SQL2 ali SQL/92).
- 1999, izide SQL:1999, podpira tudi manipulacijo z objekti.
- Pozno 2003, izide SQL:2003.

Pomembnost jezika SQL..

- SQL do sedaj edini široko sprejet standardni podatkovni jezik.
- za SQL postane del aplikacijskih arhitektur (npr. v okviru IBM-ove arhitekture - Systems Application Architecture (SAA).
- Strateška odločitev več pomembnih združb
  - Konzorcij X/OPEN za UNIX
  - Federal Information Processing Standard (FIPS) - standard kateremu morajo ustrezati vsi SUPB-ji prodani državnim organom v ZDA.
  - ...

Pomembnost jezika SQL

- SQL uporabljen tudi v drugih standardih
  - ISO Information Resource Dictionary System (IRDS)
  - Remote Data Access (RDA),...
- Interes v akademskih krogih daje jeziku teoretično osnovo in tehnike za implementacijo
  - Optimizacija poizvedb
  - Distribucija podatkov
  - Varnost podatkov
- Pojavljajo se specializirane implementacije SQL, npr. za OLAP

Pisanje SQL stavkov..

- SQL stavki so sestavljeni iz rezerviranih in uporabniško definiranih besed.
- Rezervirane besede so natančno določene, napisane morajo biti pravilno, ne smejo se lomiti med vrstice.
- Uporabniško definirane besede označujejo razne podatkovne objekte, kot so npr. relacije, stolpci, pogledi,…

Pisanje SQL stavkov..

- Večina komponent SQL stavkov je neodvisna od velikosti pisave; izjema so tekstovni podatki.
- Da dosežemo boljšo berljivost, pišemo SQL stavke v več vrsticah in z zamiki:
  - Vsak sklop SQL stavka se začne v novi vrstici
  - Sklopi so levo poravnani
  - Če ima sklop več delov, je vsak v svoji vrstici in poravnan z začetkom sklopa

Pisanje SQL stavkov..

- Za opis sintakse SQL stavkov bomo uporabljali razširjeno BNF notacijo:
  - REZERVIRANE BESEDE z velikimi črkami,
  - uporabniško definirane besede z malimi črkami,
  - Znak | za izbiro med alternativami,
  - {Obvezni elementi} v zavitih oklepajih,
  - [Opcijski elementi] v oglatih oklepajih,
  - Znak … za opcijske ponovitve (0 ali več).

BNF = Backus Naur Form

Pisanje SQL stavkov

- Podatkovne vrednosti predstavljajo konstante v SQL Stavkih.

- Vse ne-numerične vrednosti so zapisane v enojnih navednicah

  'Ljubljana'

- Vse numerične vrednosti brez navednic

  225.990

Stavki skupine SQL DML

- DML skupina zajema SQL stavke za manipulacijo s podatki
  - SELECT → Izbira
  - INSERT → Dodajanje
  - DELETE → Brisanje
  - UPDATE → Spreminjanje

- Sintaksa SELECT stavka najbolj kompleksna

SELECT stavek..

```
SELECT [DISTINCT | ALL]
        {* | [columnExpression [AS newName]] [,...] }
FROM        TableName [alias] [, ...]
[WHERE      condition]
[GROUP BY   columnList]  [HAVING      condition]
[ORDER BY   columnList]
```

SELECT stavek

- FROM Določa tabele za poizvedbo
- WHERE Filtrira vrstice
- GROUP BY Združuje vrstice po vrednostih izbranih stolpcev
- HAVING Filtrira skupine glede na določene pogoje
- SELECT Določa stolpce, ki naj se pojavijo v izhodni relaciji
- ORDER BY Določa vrstni red vrstic na izhodu

Vrstni red sklopov ni možno spreminjati!
Obvezna sta samo SELECT in FROM sklopa!

Primeri

- Za primere bomo uporabljali shemo PB o hotelih

  Hotel (hotelNo, hotelName, address)
  Room (roomNo, hotelNo, type, price)
  Booking (hotelNo, guestNo, dateFrom, dateTo, roomNo, comments)
  Guest (guestNo, guestName, guestAddress)

- Izpiši vse podatke hotelih

```
SELECT hotelNo, hotelName, address
FROM Hotel
```

ali krajše

```
SELECT * FROM Hotel
```

Uporaba DISTINCT

- Izpiši oznake hotelov in sob, ki so bile kdaj koli rezervirane

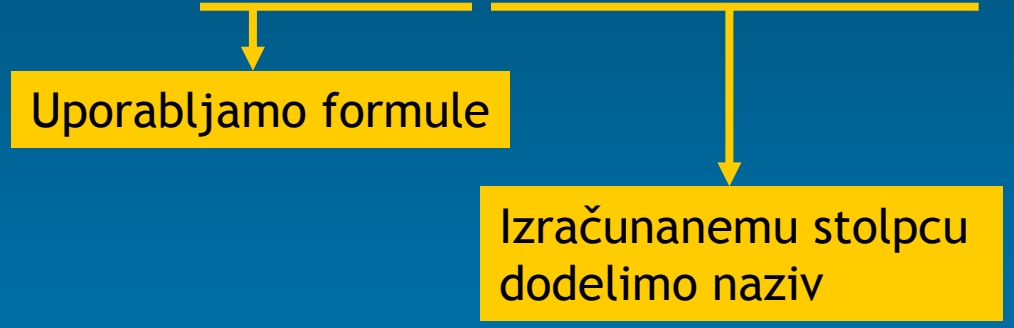
```
SELECT DISTINCT hotelNo, roomNo
FROM Booking
```

Ukaz DISTINCT eliminira dvojnike

Izračunana polja

- Izpiši ceno sob za deset dnevni najem

```
SELECT roomID, type, price*10 AS CenaNajema
FROM Room
```



Iskalni kriteriji

- Izpiši oznake hotelov, ki imajo tro-posteljne sobe (type = 3) in ceno manjšo kot 22.000 SIT

```
SELECT hotelNo
FROM Room
WHERE type = 3 AND price < 22000
```

Iskanje z uporabo BETWEEN

- Izpiši vse sobe s ceno med 10.000 in 15.000 SIT

```
SELECT roomNo
FROM Room
WHERE price BETWEEN 10000 AND 15000
```

BETWEEN vključuje spodnjo in zgornjo mejo!

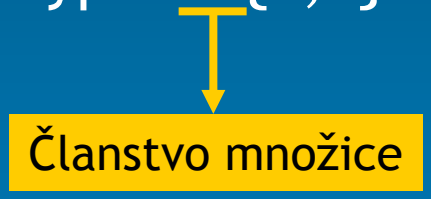
Uporabimo lahko tudi negacijo NOT BETWEEN

BETWEEN ne doda veliko SQL moči, možno izraziti posredno

Iskanje po članstvu množice

- Izpiši oznake hotelov in številke dvo ali tro-posteljnih sob

```
SELECT hotelNo, roomNo
FROM Room
WHERE type IN {2,3}
```



Uporabimo lahko tudi negacijo NOT IN

IN ne doda veliko SQL moči; koristno pri večjih množicah

Iskanje z vzorcem

- Izpiši vse goste, ki živijo kjerkoli v Ljubljani (v polju guestAddress je tudi string ‘Ljubljana’)

```
SELECT guestNo, guestName, guestAddress
FROM Guest
WHERE guestAddress LIKE ‘%Ljubljana%’
```

Iskanje z vzorcem

SQL ima dva posebna znaka za iskanje z vzorcem:

Znak % nadomešča katerikoli niz znakov

Znak _ nadomešča katerikoli znak

Iskanje z NULL vrednostjo v pogoju

- Izpiši vse rezervacije brez podanih komentarjev

```
SELECT *

FROM Booking

WHERE comments IS NULL
```

Uporabljamo lahko tudi negacijo **IS NOT NULL**

Sortiranje vrstic v izhodni relaciji

- Izpiši vse podatke o sobah, urejene po tipu sobe od največje do najmanjše in znotraj tipa po cenah od najmanjše do največje

```
SELECT *

FROM Room

ORDER BY type DESC, price ASC
```

ASC – ascending → naraščujoče
DESC – descending → padajoče

Agregiranje podatkov..

- ISO standard definira pet agregarnih operacij
  - COUNT vrne število vrednosti v določenem stolpcu
  - SUM vrne seštevek vrednosti v določenem stolpcu
  - AVG vrne povprečje vrednosti v določenem stolpcu
  - MIN vrne najmanjšo vrednost v določenem stolpcu
  - MAX vrne največjo vrednost v določenem stolpcu

- Vse operacije delujejo na enem stolpcu in vračajo eno samo vrednost.

Agregiranje podatkov..

- COUNT, MIN in MAX se uporabljajo za numerične in ne-numerične vrednosti, SUM in AVG zahtevata numerične vrednosti.

- Vse operacije razen COUNT(*) najprej odstranijo vrstice z NULL vrednostjo v stolpcu, po katerem agregiramo.

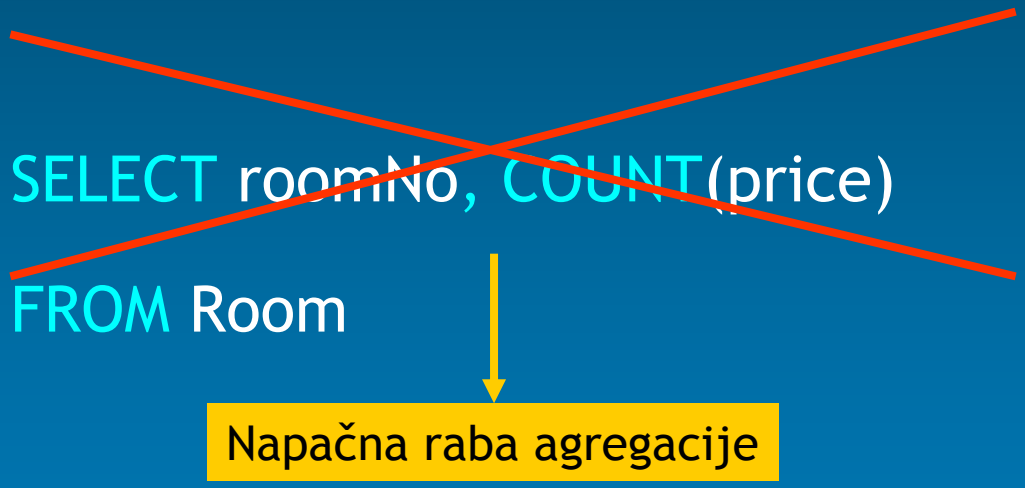
- COUNT(*) prešteje vse vrstice, ne glede na NULL vrednosti ali duplikate.

Agregiranje podatkov..

- Če se želimo znebiti duplikatov, uporabimo DISTINCT pred imenom stolpca.
- DISTINCT nima učinka na MIN/MAX, lahko pa vpliva na SUM/AVG.
- Agregarne operacije lahko uporabimo le v SELECT ali HAVING sklopu

Agregiranje podatkov

- Če SELECT sklop vsebuje agregarno operacijo, mora obstajati tudi GROUP BY sklop, sicer ni moč dodeliti agregirane vrednosti.



Uporaba COUNT in DISTINCT

- V koliko različnih hotelih obstajajo rezervacije za prvi teden v mesecu januarju 2005

```
SELECT COUNT (DISTINCT hotelNo) AS numH
FROM Booking
WHERE     dateFrom = '1.1.2005' AND
          dateTo = '7.1.2005'
```

Uporaba več agregatov istočasno

- Izpiši povprečno, minimalno in maksimalno ceno dvoposteljne sobe

```
SELECT AVG(price), MIN(price), MAX(price)
FROM Room
WHERE type = 2
```

Združevanje podatkov..

- Sklop GROUP BY uporabimo za združevanje podatkov v skupine.

- SELECT in GROUP BY sta tesno povezana
  - vsak element v SELECT seznamu, mora imeti samo eno vrednost za vse elemente v skupini,
  - SELECT sklop lahko vsebuje le:
    - imena stolpcev
    - agregarne operacije
    - konstante ali
    - izraze, ki so sestavljeni iz kombinacije naštetih elementov.

Združevanje podatkov

- Vsi stolpci, ki so navedeni v SELECT sklopu, se morajo nahajati tudi v GROUP BY sklopu, razen tistih, ki nastopajo samo v agregarnih operacijah.

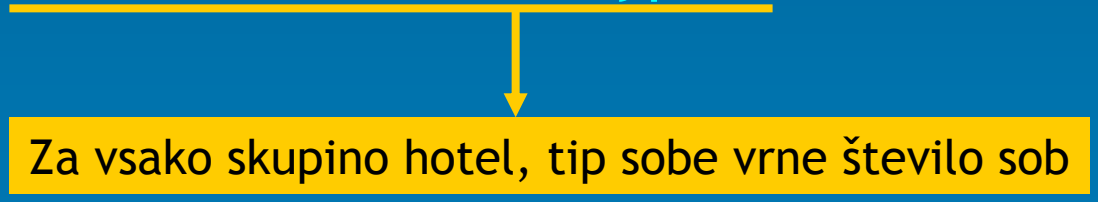
- Če uporabljamo WHERE sklop v kombinaciji z GROUP BY, se WHERE upošteva najprej, združevanje pa se izvede na preostalih vrsticah.

- ISO standard jemlje NULL vrednosti kot enake, ko gre za združevanje.

Primer združevanja

- Izpiši število enoposteljnih, dvoposteljnih in troposteljnih sob v vsakem hotelu

```
SELECT hotelNo, type, COUNT(roomID)
FROM Hotel
GROUP BY hotelNo, type
```



Omejitev skupin

- HAVING sklop je namenjen uporabi v kombinaciji z GROUP BY kot omejitev skupin, ki se lahko pojavijo v rezultatu.

- Deluje podobno kot WHERE
  - WHERE filtrira posamezne vrstice
  - HAVING filtrira skupine.

- Stolpci, ki so navedeni v HAVING sklopu, morajo biti tudi v SELECT sklopu ali v agregatih.

Uporaba sklopa HAVING

- Izpiši število enoposteljnih, dvoposteljnih in troposteljnih sob v vsakem hotelu ter njihovo povprečno ceno. Upoštevaj samo tiste primere, ko je število sob večje od 4.

```
SELECT hotelNo, type, COUNT(roomNo), AVG(price)
FROM Hotel
GROUP BY hotelNo, type
HAVING COUNT(roomNo) > 4
```

Gnezdenje poizvedb

- Nekateri SQL stavki imajo lahko vgnezdene SELECT stavke.

- Vgnezdeni SELECT stavki se lahko uporabijo v WHERE ali HAVING sklopih drugega SELECT stavka (subselect).

- Vgnezdeni SELECT stavki se lahko pojavijo tudi v INSERT, UPDATE in DELETE stavkih.

Primer vgnezdenega SELECT stavka

- Izpiši nazive hotelov, ki imajo vsaj 10 troposteljnih sob

```
SELECT hotelName
FROM Hotel
WHERE hotelNo IN
    ( SELECT hotelNo
      FROM Room
      GROUP BY hotelNo
      HAVING COUNT(hotelNo) > 9)
```

Ali je stavek pravilen?

Manjka WHERE pogoj v vgnezdenem SELECT stavku:

WHERE type = 3

Pravila gnezdenja SELECT stavkov..

- Vgnezdeni SELECT stavki ne smejo uporabljati ORDER BY sklopa.

- SELECT sklop vgnezdenega SELECT stavka lahko zajema samo en stolpec, razen v primeru uporabe ukaza EXISTS.

- Imena stolpcev v vgnezdenem SELECT stavku se privzeto nanašajo na tabele iz vgnezdenega ali zunanjega SELECT stavka (uporaba *alias*-ov)

Pravila gnezdenja SELECT stavkov

- Ko je vgnezden SELECT stavek operand v primerjavi, se mora nahajati na desni strani enačbe¸ oz. enačaja.

- Vgnezdeni SELECT stavek ne more biti operand v izrazu.

Uporaba ANY in ALL

- V vgnezedenih SELECT stavkih, ki vračajo en sam stolpec, lahko uporabljamo operatorja ANY in ALL.
- Z uporabo ALL bo pogoj izpolnjen samo, če bo veljal za vse vrednosti, ki ji vrača poizvedba.
- Z uporabo ANY, bo pogoj izpolnjen, če bo veljal za vsaj eno od vrednosti, ki ji poizvedba vrača.
- Če je rezultat poizvedbe prazen, bo ALL vrnil true, ANY pa false.
- Namesto ANY lahko uporabljamo tudi SOME.

Primer uporabe ANY

- Izpiši številke sob ter pripadajočih hotelov, katerih cena je večja kot cena vsaj ene sobe v hotelu ‘Hilton New York’ (hotelNo = ‘HIL’).

```
SELECT hotelNo, roomNo
FROM Room
WHERE price > ANY
        (SELECT price
        FROM Room
        WHERE hotelNo = ‘HIL’)
```

Primer uporaba ALL

- Izpiši številke sob ter pripadajočih hotelov, katerih cena je večja kot cena katerekoli sobe v hotelu ‘Hilton New York’ (hotelNo = ‘HIL’).

```
SELECT hotelNo, roomNo
FROM Room
WHERE price > ALL
            (SELECT price
            FROM Room
            WHERE hotelNo = ‘HIL’)
```

Poizvedbe po več tabelah..

- Poizvedbe po več tabelah lahko izvajamo z uporabo vgnezdenih SELECT stavkov
- Omejitev: stolpci v rezultatu so lahko le iz ene tabele
- V poizvedbah, ki vračajo stolpce različnih tabel, moramo uporabljati stik.
- Stik izvedemo tako, da v sklopu FROM navedemo tabele, v sklopu WHERE pa določimo stolpce za stik.

Poizvedbe po več tabelah

- Za tabele v razdelku FROM lahko uvedemo sinonime (alias).
- Sintaksa:

```
SELECT H.hotelNo, R.roomNo
FROM Hotel H, Room R
...
```

- Sinonimi so potrebni za ločevanje med istoimenskimi stolpci različnih tabel.

Primer poizvedbe po dveh tabelah

- Izpiši številke, tip in cene sob v hotelu z nazivom ‘Hilton New York’

```
SELECT R.roomNo, R.type, R.price
FROM Room R, Hotel H
WHERE     R.hotelNo = H.hotelNo AND
          H.hotelName = ‘Hilton New York’
```

Primer poizvedbe po več tabelah

- Izpiši imena in naslove gostov, ki imajo za termin od 1.1.2005 do 6.1.2005 rezervacije v hotelu ‘Hilton New York’

```
SELECT G.guestName, G.guestAddress
FROM Gost G, Booking B, Hotel H
WHERE     B.guestNo = G.guestNo AND
          B.hotelNo = H.hotelNo AND
          B.fromDate = ‘1.1.2005’ AND
          B.toDate = ‘6.1.2005’ AND
          H.hotelName = ‘Hilton New York’
```

Alternativni načini stika več tabel

- SQL omogoča alternativne načine stika med več tabelami:
  - FROM Room R JOIN Hotel H ON R.hotelNo = H.hotelNo
  - FROM Room JOIN Hotel USING hotelNo
  - FROM Room NATURAL JOIN Hotel
- Zgornji zapisi nadomestijo sklopa FROM in WHERE
- V prvem primeru rezultat vsebuje dva identična stolpca hotelNo.

Računanje stika..

- Stik predstavlja podmnožico kartezijskega produkta
- Če pri navedbi dveh tabel A in B v FROM sklopu ne navedemo stika v WHERE sklopu, dobimo kartezijski produkt med A in B.

- ISO standard nudi posebno obliko zapisa za kartezijski produkt:

```
SELECT [DISTINCT | ALL] {* | columnList}
FROM Table1 CROSS JOIN Table2
```

Računanje stika

- V splošnem je postopek za generiranje rezultata SELECT stavka s stikom naslednji:
  - Sestavi kartezijski produkt tabel, ki so naštete v sklopu FROM
  - Če obstaja WHERE sklop, upoštevaj vse pogoje in iz kartezijskega produkta izberi samo tiste vrstice, ki pogojem ustrezajo (selekcija)
  - Iz izbranih vrstic kartezijskega produkta izberi samo tiste stolpce, ki ustrezajo naboru v SELECT sklopu (projekcija)
  - Če je bil uporabljen DISTINCT operator, eliminiraj dvojnike (del projekcije)
  - Če obstaja ORDER BY sklop, ustrezno razvrsti vrstice

Zunanji stik..

- S pomočjo zunanjega stika dobimo v rezultat tudi vrstice, ki nimajo stične vrednosti v drugi tabeli.

R=OSEBA

ID	Priimek in ime	PTT
1	Kante Janez	5270
2	Tratnik Jože	5000
3	Mali Mihael	
4	Brecelj Jana	1000

S=KRAJ

PTT	Naziv
1000	Ljubljana
5000	Nova Gorica
5270	Ajdovščina

$(\Pi_{ID,\ Priimek\ in\ ime,\ PTT}(R)) \rtimes S$

Priimek in ime	PTT	Naziv zač. preb.
Kante Janez	5270	Ajdovščina
Tratnik Jože	5000	Nova Gorica
Mali Mihael		
Brecelj Jana	1000	Ljubljana

Zunanji stik

- Za zapis SELECT stavka, ki vsebuje zunanji stik med dvema tabelama, uporabimo naslednjo sintakso:

```
SELECT DISTINCT H.hotelName
FROM Hotel H LEFT JOIN
        Booking B ON H.hotelNo = B. hotelNo
```

Kaj izpiše zgornja poizvedba?

Polni zunanji stik

- SQL omogoča tudi izvedbo polnega zunanjega stika (Full Outer Join)
- Polni zunanji stik med tabelama A in B kot rezultat vrne tudi tiste vrstice, ki v tabeli A ali B nimajo stičnega para.

- Sintaksa:

```
SELECT DISTINCT G.guestName, H.hotelName
FROM Hotel H FULL JOIN Guest G
        ON H.address = G.guestAddress
```

Uporaba EXISTS in NOT EXISTS

- EXISTS in NOT EXISTS lahko uporabljamo le v vgnezdenih poizvedbah.
- Vračajo logičen rezultat true/false.
  - True dobimo, če obstaja vsaj ena vrstica v tabeli, ki je rezultat vgnezdene poizvedbe.
  - False dobimo, če vgnezdena poizvedba vrača prazno množico.
- NOT EXISTS je negacija EXISTS.

Uporaba EXISTS in NOT EXISTS

- (NOT) EXISTS preveri samo, če v rezultatu vgnezdene poizvedbe (ne) obstajajo vrstice

- Število stolpcev v SELECT sklopu vgnezdene poizvedbe je zato irelevantno
- Navadno uporabimo sintakso:

  (SELECT * ...)

Primer uporabe EXISTS

- Izpiši vse goste, ki so kdaj koli imeli rezervacije v hotelu Hilton New York.

```
SELECT guestName, guestAddress
FROM Guest G
WHERE EXISTS
    (SELECT *
     FROM Booking B, Hotel H
     WHERE B.guestNo = G.guestNo AND
         B.hotelNo = H.hotelNo AND
         H.hotelName = 'Hilton New York')
```

Kaj dobimo, če ta pogoj izpustimo?

Namesto EXISTS lahko uporabimo stik

```
SELECT DISTINCT G.guestName, G.guestAddress
FROM Guest G, Booking B, Hotel H
WHERE B.guestNo = G.guestNo AND
        B.hotelNo = H.hotelNo AND
        H.hotelName = 'Hilton New York')
```

Uporaba operacij nad množicami..

- Rezultate dveh ali več poizvedb lahko združujemo z ukazi:
  - Union (unija),
  - Intersection (Presek)
  - Difference (EXCEPT) (Razlika)

- Da lahko izvajamo naštete operacije, morata tabeli A in B biti skladni (domene atributov morajo biti enake).

Primer unije

- Izpiši vse mesta, kjer je bodisi lociran kakšen gost hotelske verige ali kakšen hotel.

```
(SELECT address
FROM Hotel)
UNION
(SELECT guestAddress
FROM Guest)
```

Kako bo naziv stolpca, ki ga poizvedba vrne?

Primer preseka

- Izpiši vse mesta, kjer je lociran kakšen gost hotelske verige in obenem kakšen hotel.

```
(SELECT address
FROM Hotel)
INTERSECTION
(SELECT guestAddress
FROM Guest)
```

Uporaba EXCEPT

- Izpiši vse mesta, kjer je lociran kakšen gost hotelske verige in obenem v tem kraju ni nobenega hotela.

```
(SELECT guestAddress
FROM Guest)
EXCEPT
(SELECT address
FROM Hotel)
```

INSERT stavek..

INSERT INTO TableName [(columnList)]

VALUES (dataValueList)

- Seznam columnList ni obvezen; če ga spustimo, interpreter pričakuje vrednosti za vse stolpce tabele, v vrstnem redu, kot so bili kreirani.
- Pri vnosu moramo vpisati najmanj vse obvezne vrednosti (not null), razen za stolpce, pri katerih je bila ob kreiranju določena privzeta vrednost (DEFAULT).

INSERT stavek

- Seznam dataValueList mora ustrezati seznamu columnList:
  - Število elementov v seznamih mora biti enako;
  - Vrednost, ki se nanaša na nek stolpec, mora biti v seznamu dataValueList na istem mestu, kot je stolpec v seznamu columnList;
  - Podatkovni tip vrednosti, ki se nanaša na nek stolpec, mora biti enak kot podatkovni tip stolpca.

Primeri INSERT stavkov..

- Vnos nove vrstice v tabelo Booking

```
INSERT INTO Booking
VALUES (21, 109, '12.12.2005', '17.12.2005', 109,
'soba za nekadilce');
```

- Shema relacije Booking:

  Booking (hotelNo, guestNo, dateFrom, dateTo, roomNo, comments)

Primeri INSERT stavkov..

- Vnos nove vrstice v tabelo Booking – vnos samo obveznih vrednosti

```
INSERT INTO Booking
VALUES (21, 109, '12.12.2005', '17.12.2005', 109,
null);
```

stolpec comments je neobvezen

- ali

```
INSERT INTO Booking (hotelNo, guestNo, dateFrom,
dateTo, roomNo)
VALUES (21, 109, '12.12.2005', '17.12.2005', 109);
```

Primeri INSERT stavkov..

- Vnos več vrstic iz ene ali več drugih tabel...

```
INSERT INTO TableName [ (columnList) ]
SELECT ...
```

Primeri INSERT stavkov..

- Predpostavimo, da imamo tabelo HotelRez, ki za vsak hotel pove oznako hotela, naziv hotela in število trenutno odprtih rezervacij

  HotelRez(hotelNo, hotelName, NumRez)

- S pomočjo tabel Hotel in Booking napolnimo tabelo HotelRez

Primeri INSERT stavkov

```
INSERT INTO HotelRez
        (SELECT H.hotelNo, H.hotelName, COUNT(*)
        FROM Hotel H, Booking B
        WHERE H.hotelNo = B.hotelNo AND
                B.dateFrom >= date()
        GROUP BY H.hotelNo, H.hotelName)
        UNION
        (SELECT hotelNo, hotelName, 0
        FROM Hotel
        WHERE hotelNo NOT IN
                (SELECT DISTINCT hotelNo
                FROM Booking));
```

Kaj se zgodi, če spodnji del SQL stavka spustimo?

UPDATE stavek..

UPDATE TableName

SET columnName1 = dataValue1

[, columnName2 = dataValue2...]

[WHERE searchCondition]

- TableName se lahko nanaša na ime osnovne tabele ali ime pogleda.
- Sklop SET določa nazive enega ali več stolpcev ter nove vrednosti teh stolpcev.

UPDATE stavek

- WHERE sklop je neobvezen:
  - Če ga spustimo, se v imenovane stolpce vpišejo nove vrednosti za vse vrstice v tabeli;
  - Če WHERE sklop določimo, se spremembe zgodijo zgolj za vrstice, ki ustrezajo WHERE pogojem.

- Nove podatkovne vrednosti morajo ustrezati podatkovnemu tipu stolpca.

Primeri UPDATE stavkov

- Vse dvoposteljne sobe v hotelu z oznako 201 povišaj za 5%
- V WHERE imamo lahko vgnezdene stavke

```
UPDATE Room
SET price = price * 1,05
WHERE type = 2 AND hotelNo = 201
```

DELETE stavek

DELETE FROM TableName

[WHERE searchCondition]

- TableName se lahko nanaša na ime osnovne tabele ali ime pogleda.
- WHERE sklop ni obvezen. Če ga spustimo, zbrišemo vse vrstice v tabeli. Tabela ostane.

Primeri DELETE stavkov

- Izbriši vse potekle rezervacije, ki se nanašajo na hotel Slon.

```
DELETE FROM Booking
WHERE     FromDate < '25.11.2005' AND
          hotelNo IN (SELECT hotelNo
                      FROM Hotel
                      WHERE hotelName = 'Hotel Slon'
                      )
```

Stavki skupine SQL DDL..

- DDL skupina zajema SQL stavke za manipulacijo s strukturo podatkovne baze.

Podatkovni tipi v SQL standardu

Table 6.1 ISO SQL data types.

Data type	Declarations			
boolean	BOOLEAN			
character	CHAR	VARCHAR		
bit	BIT	BIT VARYING		
exact numeric	NUMERIC	DECIMAL	INTEGER	SMALLINT
approximate numeric	FLOAT	REAL	DOUBLE PRECISION	
datetime	DATE	TIME	TIMESTAMP	
interval	INTERVAL			
large objects	CHARACTER LARGE OBJECT		BINARY LARGE OBJECT	

Vir: [2,XX]

Integrity Enhancement Feature..

- Poznamo več vrst omejitev:
  - Obveznost podatkov
  - Omejitve domene (Domain constraints)
  - Pravila za celovitost podatkov (Integrity constraints)
    - Celovitost entitet (Entity Integrity)
    - Celovitost povezav (Referential Integrity)
  - Številnost (Multyplicity)
  - Splošne omejitve (General constraints)

Integrity Enhancement Feature..

- Obveznost podatkov

  hotelNo Numeric(3) NOT NULL

- Omejitve domene

  CHECK

  spol CHAR NOT NULL

  CHECK (spol IN (‘M’, ‘Ž’))

Integrity Enhancement Feature..

- CREATE DOMAIN

  CREATE DOMAIN DomainName [AS] dataType
  [DEFAULT defaultOption]
  [CHECK (searchCondition)]

  Primer:

  CREATE DOMAIN Tspol AS CHAR
  CHECK (VALUE IN (‘M’, ‘Ž’));

  Spol Tspol NOT NULL

Integrity Enhancement Feature..

- searchCondition lahko vsebuje iskalno tabelo (lookup table):

```
CREATE DOMAIN guestNo AS CHAR(4)
CHECK (VALUE IN (SELECT guestNo FROM Guest));
```

- Domeno lahko ukinemo z uporabo stavka DROP DOMAIN:

```
DROP DOMAIN DomainName
[RESTRICT | CASCADE]
```

IEF - Celovitost entitet

- Primarni ključ tabele mora vsebovati enolično neprazno vrednost v vsaki vrstici tabele.
- ISO standard podpira tuje ključe s sklopom FOREIGN KEY v okviru CREATE in ALTER TABLE stavkov.

  PRIMARY KEY(hotelNo, roomNo)

- Vsaka tabela ima lahko največ en primarni ključ. Enoličnost neosnovnih stolpcev lahko zagotavljamo z uporabo UNIQUE

  UNIQUE(priimek)

IEF - Celovitost povezav..

- FK (tuji ključ) je stolpec ali množica stolpcev, ki povezujejo vsako vrstico tabele A z vrstico referenčne tabele B, kjer se ujemajo vrednosti A.FK = B.PK.
- Celovitost povezav zagotavlja, da če ima FK neko vrednost, potem se ta vrednost nahaja v primarnem ključu povezane tabele.
- ISO standard omogoča definicijo tujih ključev s sklopom FOREIGN KEY v CREATE in ALTER TABLE

  FOREIGN KEY(hotelNo) REFERENCES Hotel

IEF - Celovitost povezav..

- Vsak INSERT/UPDATE stavek, ki skuša kreirati FK vrednost v tabeli, brez da bi ta vrednost obstajala kot PK v povezani tabeli, je zavrnjen.

IEF – Celovitost povezav

- Določimo z uporabo ON UPDATE, ON DELETE

  ON UPDATE SET NULL

- Primeri:

  FOREIGN KEY (hotelNo) REFERENCES Hotel

  ON DELETE SET NULL

  FOREIGN KEY (guestNo) REFERENCES Guest

  ON UPDATE CASCADE

Kreiranje podatkovnih objektov..

- SQL DDL omogoča kreiranje in brisanje podatkovnih objektov, kot so: shema, domena, tabela, pogled in indeks.
- Glavni SQL DDL stavki so:

CREATE/ALTER DOMAIN	DROP DOMAIN
CREATE/ALTER TABLE	DROP TABLE
CREATE VIEW	DROP VIEW

- Mnogi SUPB-ji omogočajo tudi

CREATE INDEX	DROP INDEX

Kreiranje podatkovnih objektov..

- Relacije in drugi podatkovni objekti obstajajo v nekem okolju.
- Vsako okolje vsebuje enega ali več katalogov, vsak katalog pa množico shem.
- Shema je poimenovana kolekcija povezanih podatkovnih objektov.
- Objekti v shemi so lahko tabele, pogledi, domene, trditve, dodelitve, pretvorbe in znakovni nizi. Vsi objekti imajo istega lastnika.

Kreiranje tabele..

```
CREATE TABLE TableName
{(colName dataType [NOT NULL] [UNIQUE]
[DEFAULT defaultOption]
[CHECK searchCondition] [,...]}
[PRIMARY KEY (listOfColumns),]
{[UNIQUE (listOfColumns),] [...,]}
{[FOREIGN KEY (listOfFKColumns)
    REFERENCES ParentTableName [(listOfCKColumns)],
    [ON UPDATE referentialAction]
    [ON DELETE referentialAction ]] [,...]}
    {[CHECK (searchCondition)] [,...] })
```

Primer kreiranja tabele..

Najprej kreiramo domene

```
CREATE DOMAIN hotelNumber AS NUMERIC(3)
CHECK (VALUE IN (SELECT hotelNo FROM Hotel));

CREATE DOMAIN guestNumber AS NUMERIC(3)
CHECK (VALUE IN (SELECT guestNo FROM Guest));

CREATE DOMAIN rezervDate AS DATE;
CHECK(VALUE BETWEEN '1.1.1995' AND '1.1.2200');

CREATE DOMAIN roomNumber AS INTEGER;
CHECK(VALUE BETWEEN 100 AND 545);

CREATE DOMAIN comments AS VARCHAR(100);
```

Primer kreiranja tabele..

potem kreiramo tabelo

```
CREATE TABLE Booking (
    hotelNo      hotelNumber   NOT NULL,
                 CONSTRAINT PrevecRezervacij…
    guestNo      guestNumber   NOT NULL,
    dateFrom     rezervDate    NOT NULL      DEFAULT date(),
    dateTo       rezervDate    NOT NULL,
    roomNo       roomNumber    NOT NULL,
    comments     comments,
    PRIMARY KEY (hotelNo),
    FOREIGN KEY (guestNo) REFERENCES Guest
    ON DELETE SET NULL ON UPDATE CASCADE …);
```

ALTER TABLE stavek..

- S stavkom ALTER TABLE lahko:
  - Dodajamo ali ukinjamo stolpce v tabeli;
  - Dodajamo ali ukinemo omejitve tabele;
  - Za stolpce v tabeli določamo ali ukinjamo privzete vrednosti;
  - Spreminjamo podatkovne tipe stolpcev v tabeli;

Primeri ALTER TABLE stavkov..

- Spremeni tabelo Booking tako, da ukineš privzeto vrednost stolpca fromDate.

```
ALTER TABLE Booking
    ALTER fromDate DROP DEFAULT;
```

Primeri ALTER TABLE stavkov..

- Spremeni tabelo Booking tako, da ukineš omejitev, da nobena soba nobenega hotela ne sme imeti več kot eno rezervacijo na isti dan. V tabelo Gost dodaj stolpec Spol.

```
ALTER TABLE Booking
        DROP CONSTRAINT prevecRezervacij;

ALTER TABLE Gost
        ADD Spol NOT NULL DEFAULT = 'M';
```

Stavek DROP TABLE

- S pomočjo stavka DROP TABLE ukinemo tabelo. Obenem se zbrišejo vsi zapisi tabele.

  DROP TABLE TableName [RESTRICT | CASCADE]

- Restrict: Ukaz se ne izvede, če obstajajo objekti, ki so vezani na tabelo, ki jo brišemo.
- Cascade: kaskadno se brišejo vsi vezani objekti.
- Primer:

  DROP TABLE Gost RESTRICT;

QBE - Query-By-Example..

- Vizualen način dostopa do podatkov podatkovne baze z uporabo vzorcev poizvedb.
- Z vzorcem povemo, kakšni podatki nas iz podatkovne baze zanimajo.
- QBE originalno razvil IBM v 70' letih. Včasih zelo popularen. Večina SUPB ga nudi.
- Poizvedbo, ki jo uporabnik napiše v QBE, se v ozadju prevede v SQL poizvedbo.

QBE - Query-By-Example..

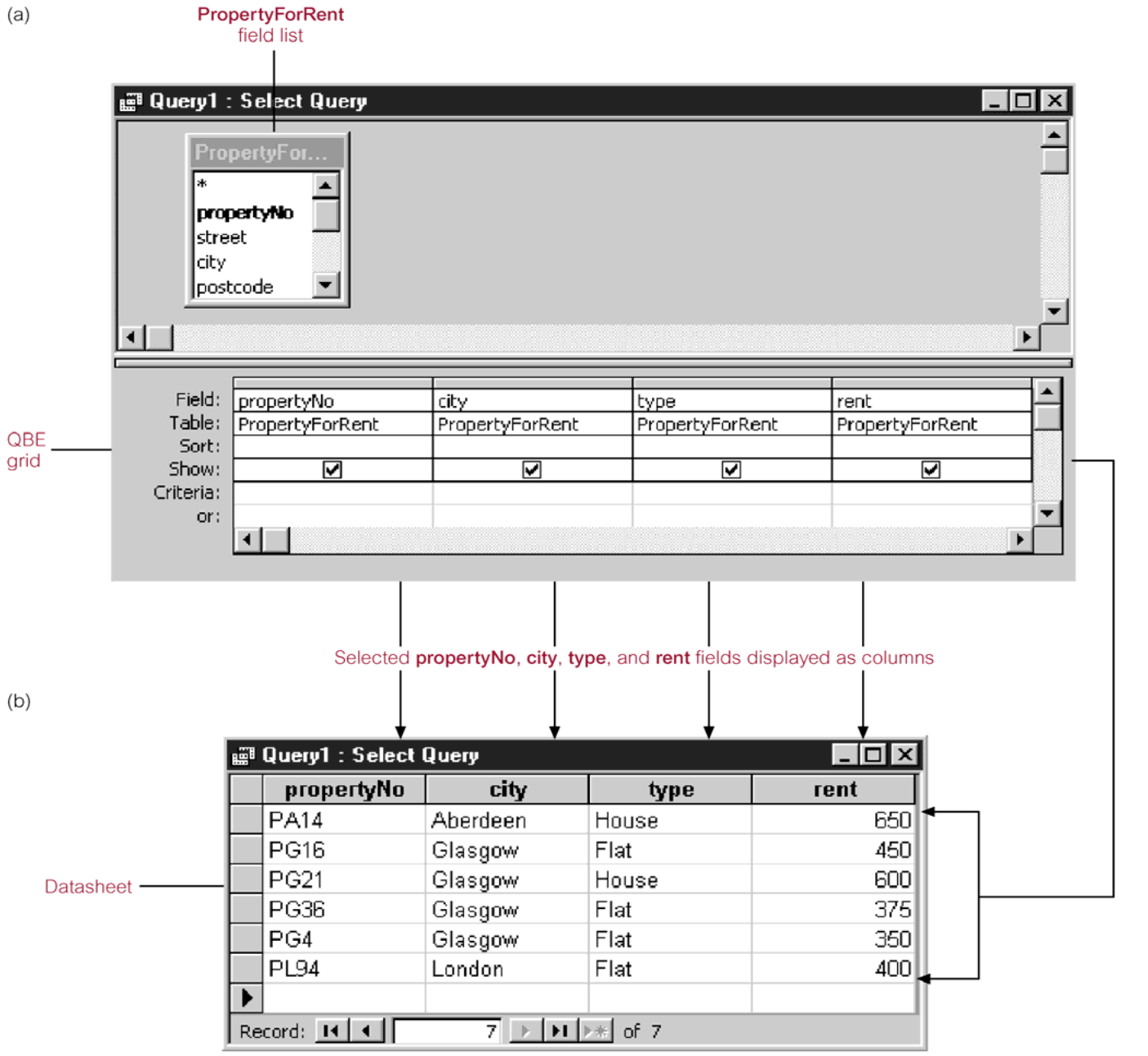
- QBE uporabnikom omogoča:
  - Poizvedovati po podatkih ene ali več tabel.
  - Določiti stolpce, ki jih želimo imeti v odgovoru (projekcija).
  - Določiti kriterije za izbiro vrstic (selekcija).
  - Izvajati izračune nad podatki v tabelah.
  - Dodajati in brisati zapise.
  - Spreminjati vrednosti v poljih.
  - Kreirati nove tabele in stolpce.

QBE - Query-By-Example..

- Primeri iz Accessa

Primeri iz

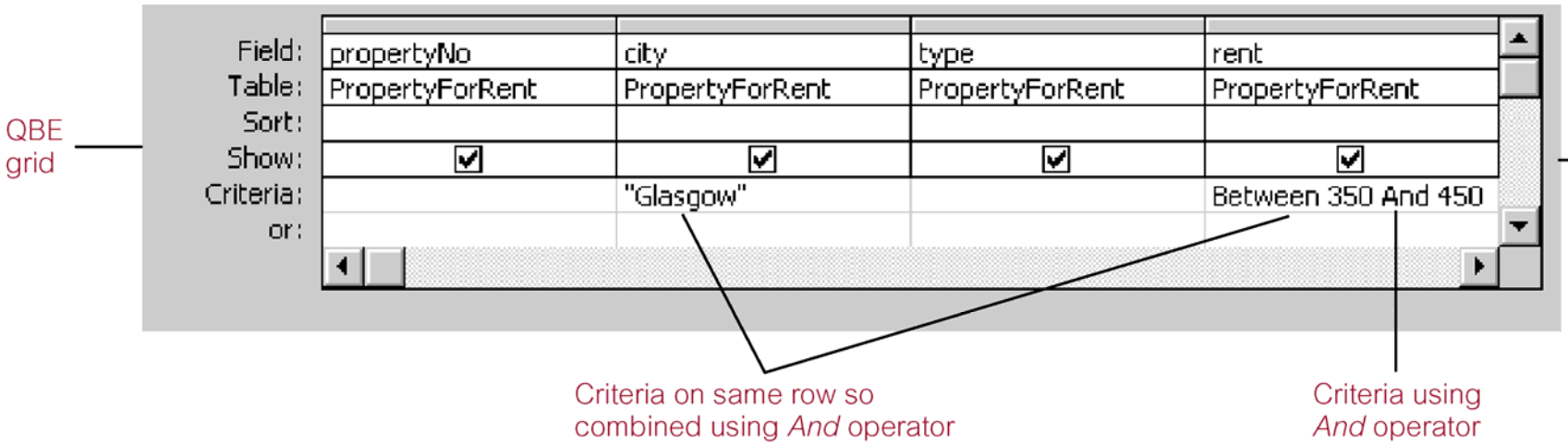
- Poizvedba



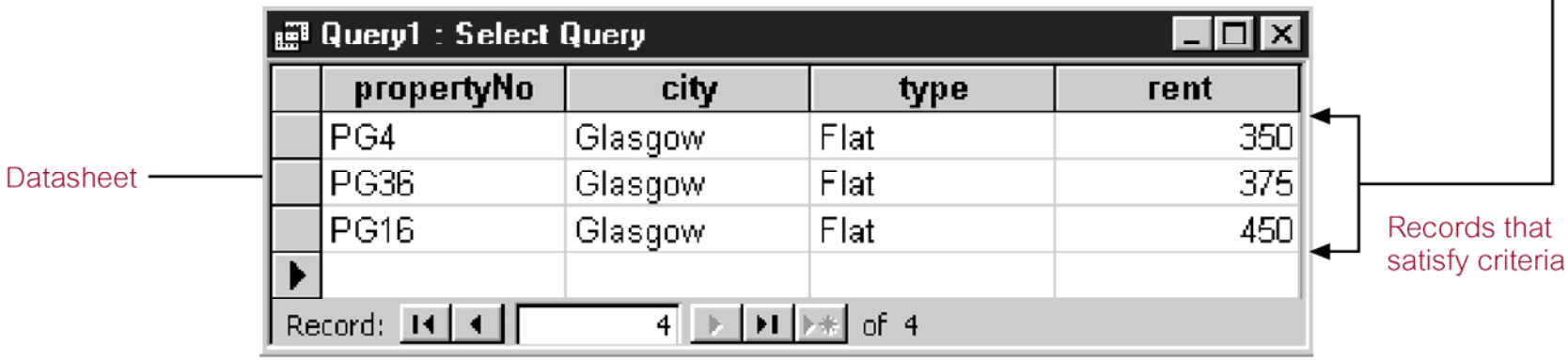
SELECT PropertyForRent.propertyNo, PropertyForRent.city, PropertyForRent.type, PropertyForRent.rent
FROM PropertyForRent;

Primeri iz Access-a..

(a)



(b)

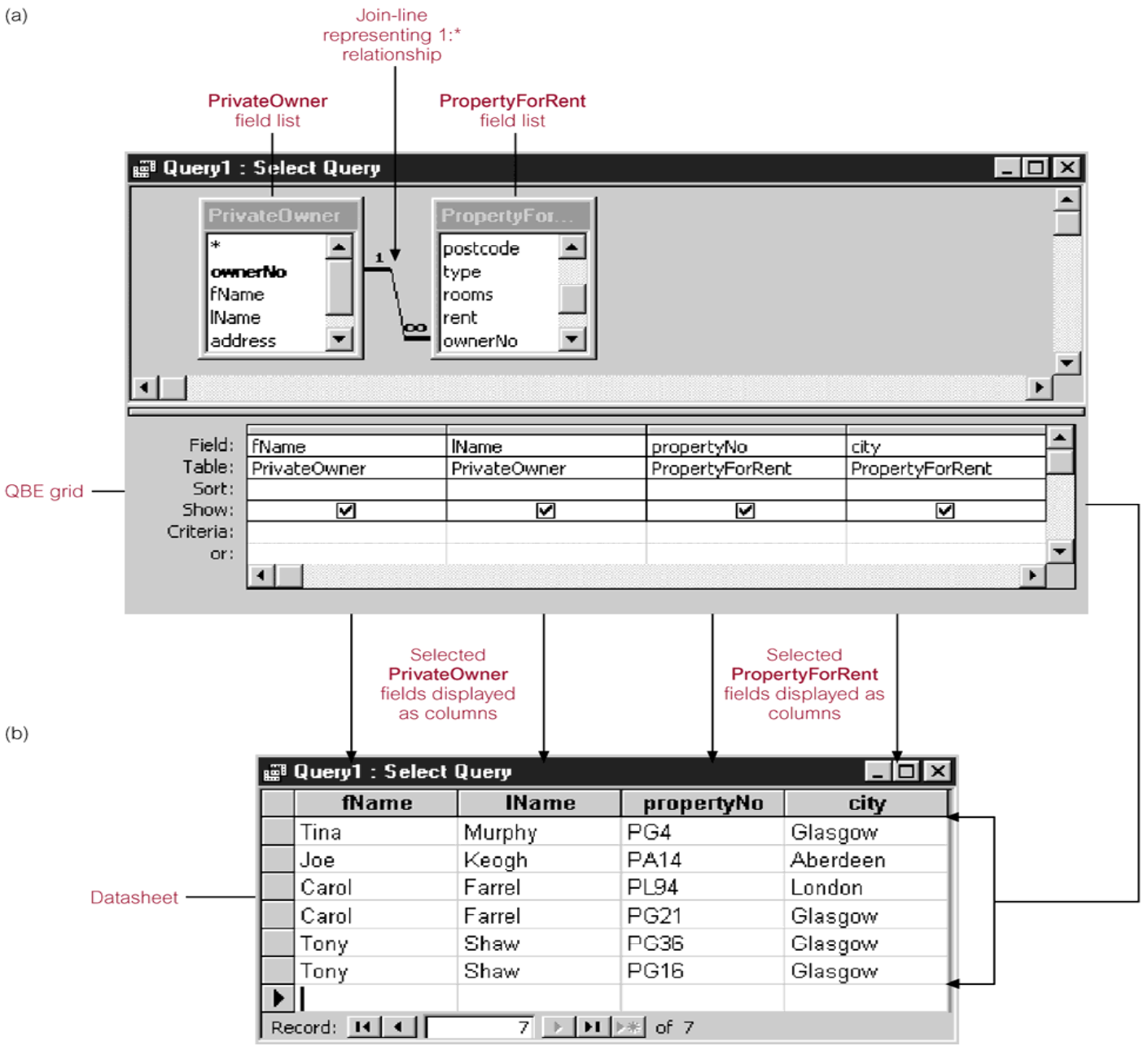


(c)

```
SELECT PropertyForRent.propertyNo, PropertyForRent.city, PropertyForRent.type, PropertyForRent.rent
FROM PropertyForRent
WHERE (((PropertyForRent.city)="Glasgow") AND ((PropertyForRent.rent) Between 350 And 450));
```

Primeri iz

- Poizvedb
- Stik tabe

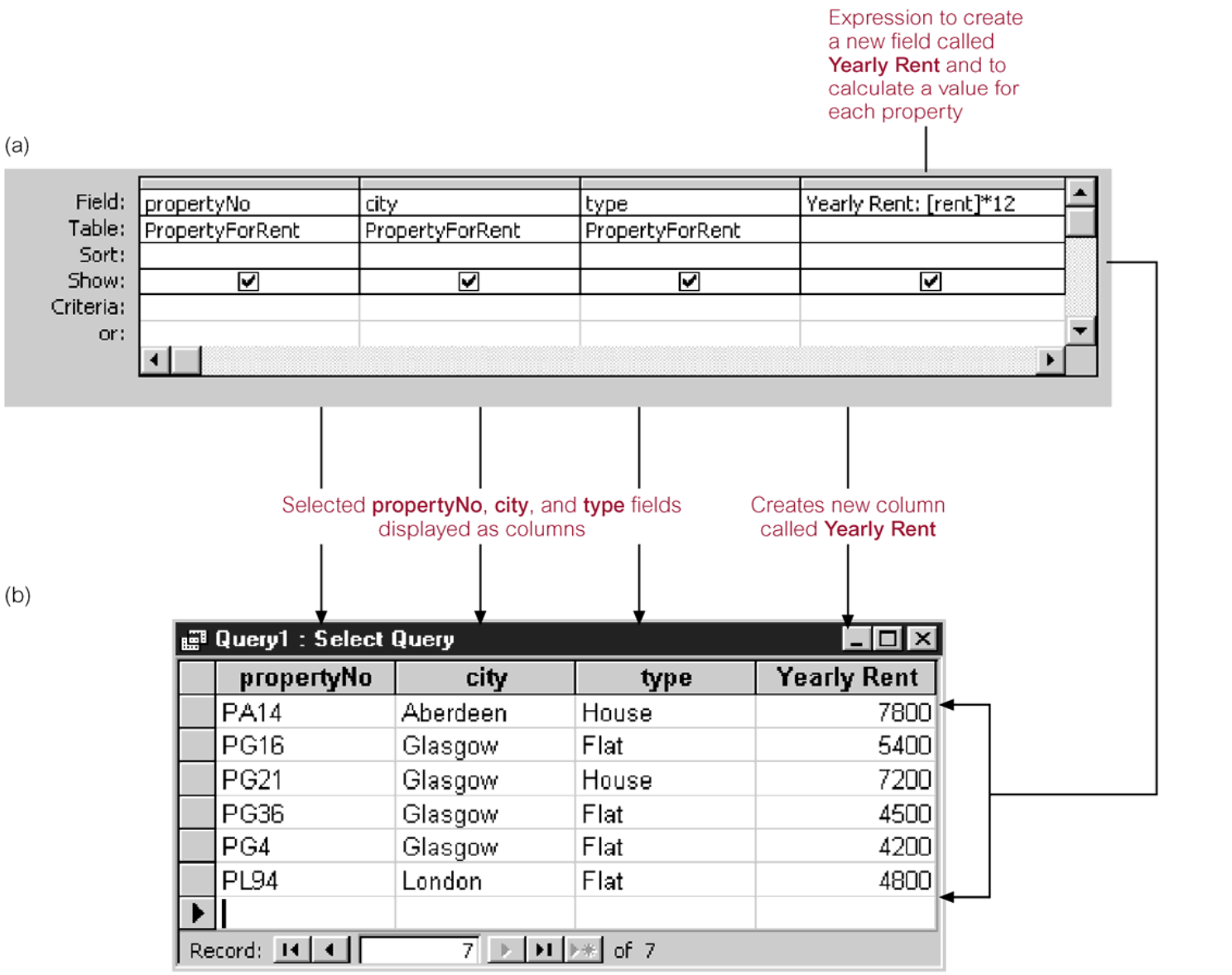


(c)

```
SELECT PrivateOwner.fName, PrivateOwner.lName, PropertyForRent.propertyNo, PropertyForRent.city
FROM PrivateOwner INNER JOIN PropertyForRent ON PrivateOwner.ownerNo =
PropertyForRent.ownerNo;
```

Primeri

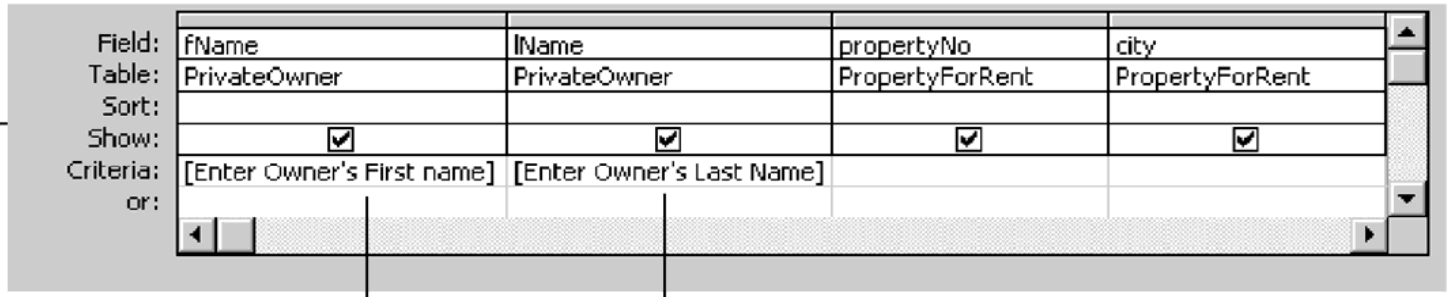
- Poizve
- Uporal



Prim

- Poi
- Upd

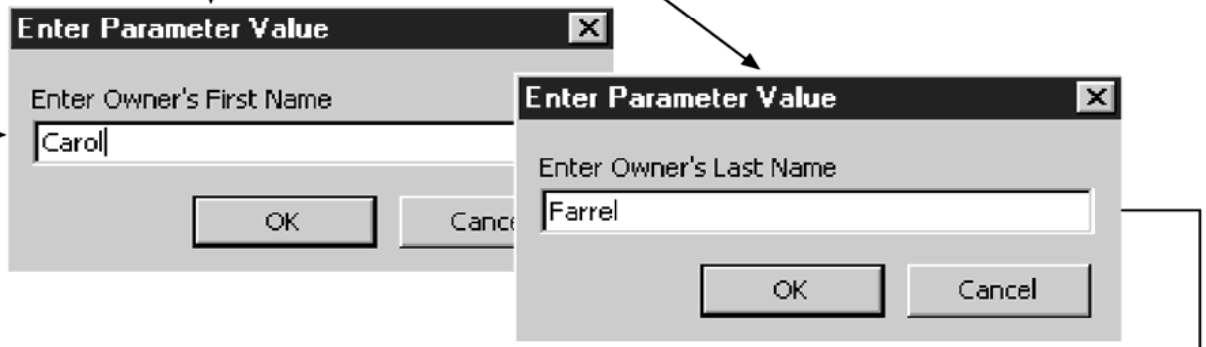
(a)



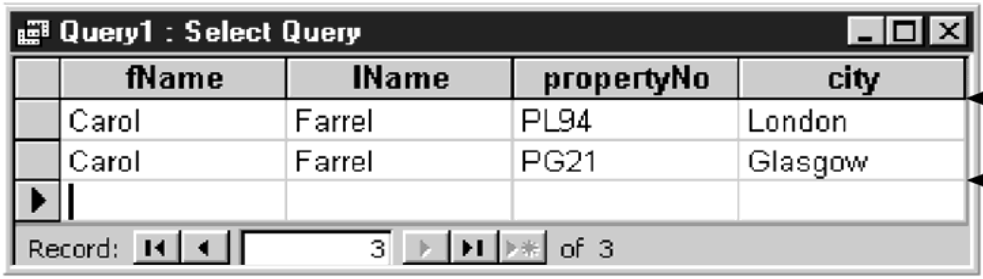
Expression to create prompt for **fName** field

Expression to create prompt for **lName** field

(b)



(c)



Records that satisfy criteria

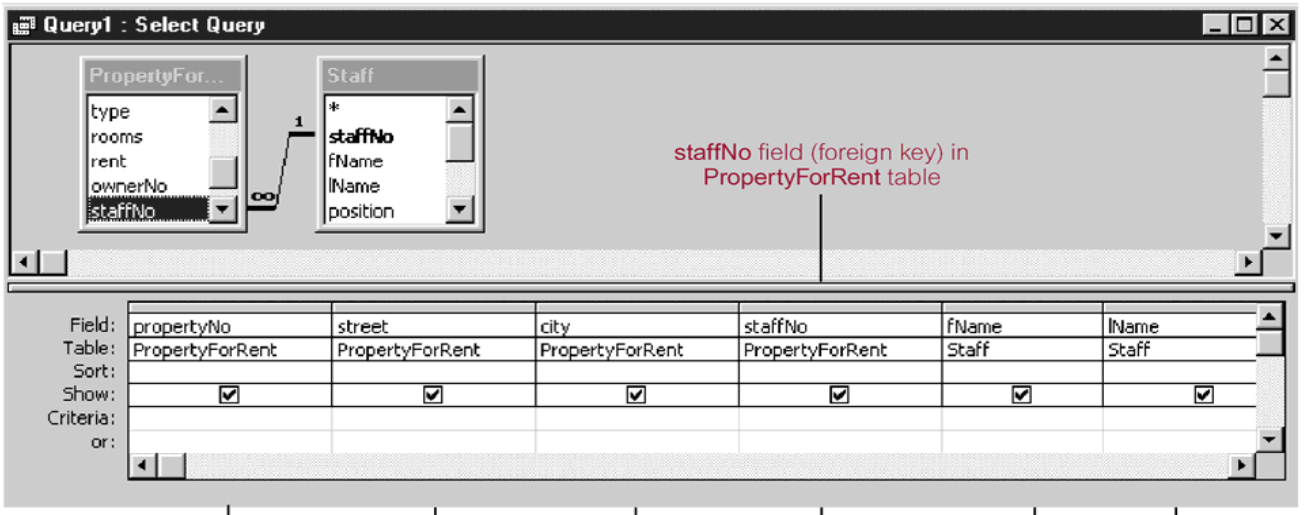
(d)

```
SELECT PrivateOwner.fName, PrivateOwner.lName, PropertyForRent.propertyNo, PropertyForRent.city
FROM PrivateOwner INNER JOIN PropertyForRent ON PrivateOwner.ownerNo = PropertyForRent.ownerNo
WHERE (((PrivateOwner.fName)=[Enter Owner's First Name]) AND ((PrivateOwner.lName)=[Enter
Owner's Last Name]));
```

Primeri iz A

- Poizvedba S
- Uporaba "Au

(a)



Selected **PropertyForRent** fields displayed as columns

Selected **Staff** fields displayed as columns

(b)

Query1 : Select Query

propertyNo	street	city	staffNo	fName	lName
PA14	16 Holhead	Aberdeen	SA9	Mary	Howe
PG16	5 Novar Drive	Glasgow	SG14	David	Ford
PG36	2 Manor Road	Glasgow	SG37	Ann	Beech
PG21	18 Dale Road	Glasgow	SG37	Ann	Beech
PL94	6 Argyll Street	London	SL41	Julie	Lee
PG97	Muir Drive	Aberdeen	SA9	Mary	Howe
*					

Record: 6 of 6

User enters values for new property

User enters **staffNo** 'SA9'

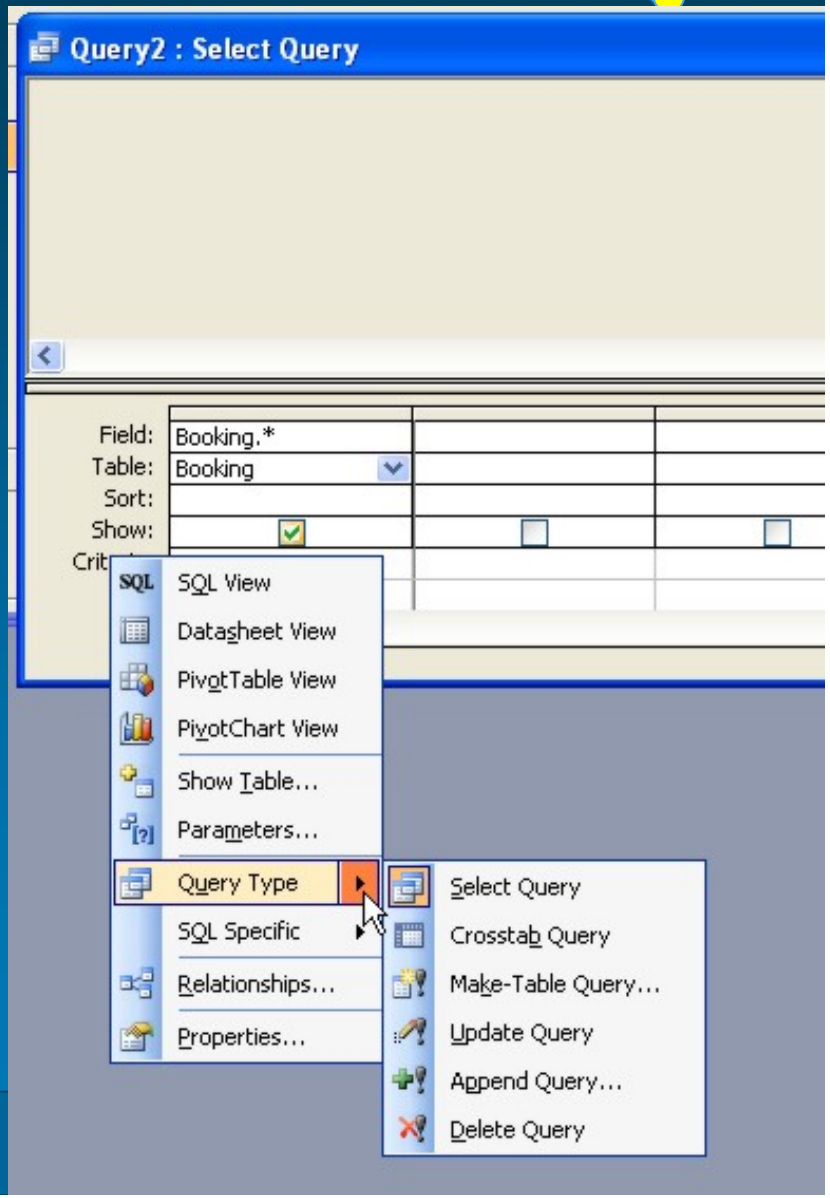
Microsoft Access automatically fills in the **fName** and **lName** columns with the values associated with **staffNo** 'SA9'

(c)

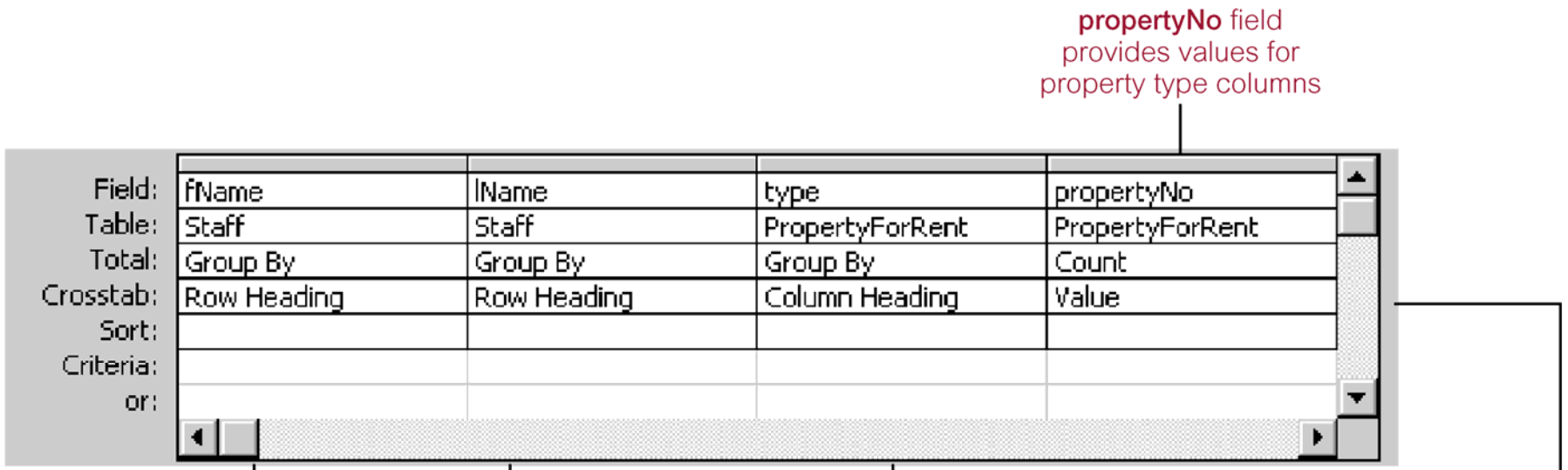
```
SELECT PropertyForRent.propertyNo, PropertyForRent.street, PropertyForRent.city,
PropertyForRent.staffNo, Staff.fName, Staff.lName
FROM Staff INNER JOIN PropertyForRent ON Staff.staffNo = PropertyForRent.staffNo;
```

Primeri iz Access-a..

- Posebne vrste SQL stavkov



(a)



(b)

Query1 : Crosstab Query

	fName	lName	Bungalow	Cottage	Flat	Mid-Terrace	Semi-Detached
▶	Ann	Beech	43	4	45	26	33
	David	Ford	7	2	14		42
	Mary	Howe	45	4	31	2	7

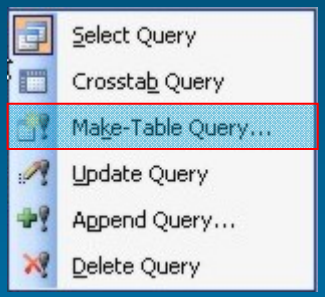
Record: 1 of 3

(c)

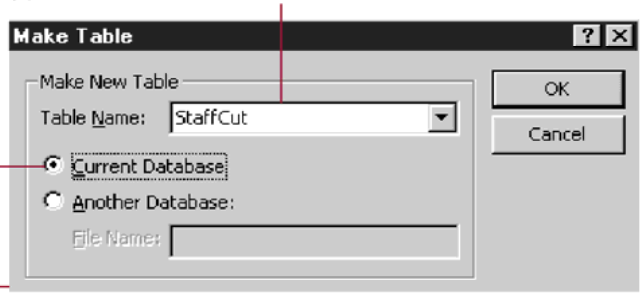
```
TRANSFORM Count(PropertyForRent.propertyNo) AS CountOfpropertyNo
SELECT Staff.fName, Staff.lName
FROM Staff INNER JOIN PropertyForRent ON Staff.staffNo = PropertyForRent.staffNo
GROUP BY Staff.fName, Staff.lName
PIVOT PropertyForRent.type;
```

Primeri iz Acc

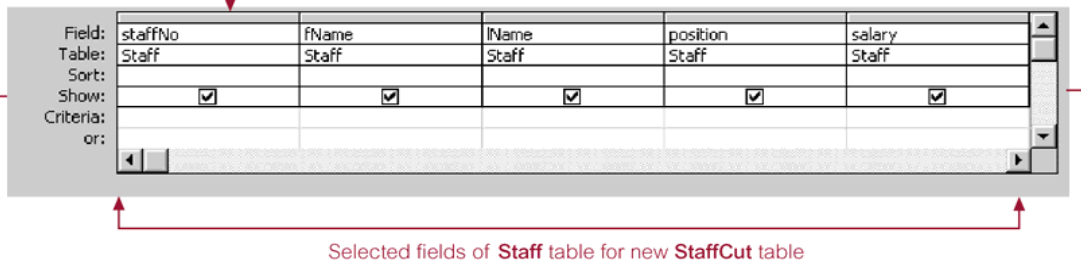
- Kreiranje nove



(a)

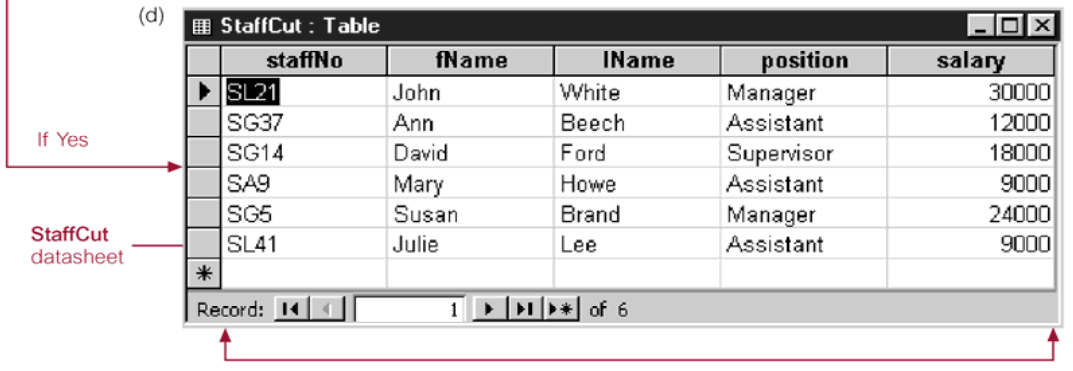


(b)



(c)

(d)

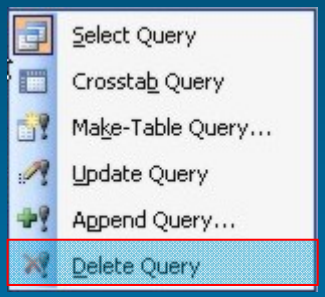


(e)

```
SELECT Staff.staffNo, Staff.fName, Staff.lName, Staff.position, Staff.salary INTO StaffCut
FROM Staff;
```

Primeri iz

- Brisanje z



(a)

Target table for deletion

Delete QBE grid

Field:	PropertyForRent.*	city
Table:	PropertyForRent	PropertyForRent
Delete:	From	Where
Criteria:		"Glasgow"
or:		

Criterion for properties in Glasgow

(b)

Microsoft Access

You are about to delete 4 row(s) from the specified table.

Once you click Yes, you can't use the Undo command to reverse the changes.
Are you sure you want to delete the selected records?

Yes No

If Yes

(c)

PropertyForRent : Table

propertyNo	street	city	postcode
PA14	16 Holhead	Aberdeen	AB7 5SU
PL94	6 Argyll Street	London	NW2

Record: 1 of 2

PropertyForRent datasheet minus deleted records (properties in Glasgow)

Deletion cascades to related table if referential integrity is set and cascade deletes are allowed

Viewing : Table

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-May-01	too small
CR62	PA14	14-May-01	no dining room

Record: 1 of 2

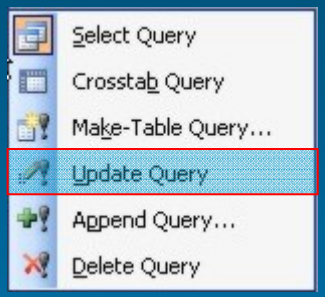
Viewing datasheet minus deleted records (viewings of properties) in Glasgow)

(d)

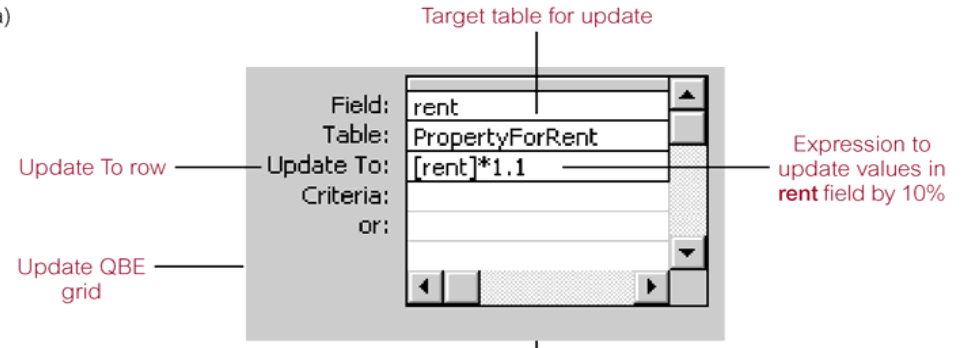
```
DELETE PropertyForRent.*, PropertyForRent.city
FROM PropertyForRent
WHERE (((PropertyForRent.city)="Glasgow"));
```

Primeri iz Access-a

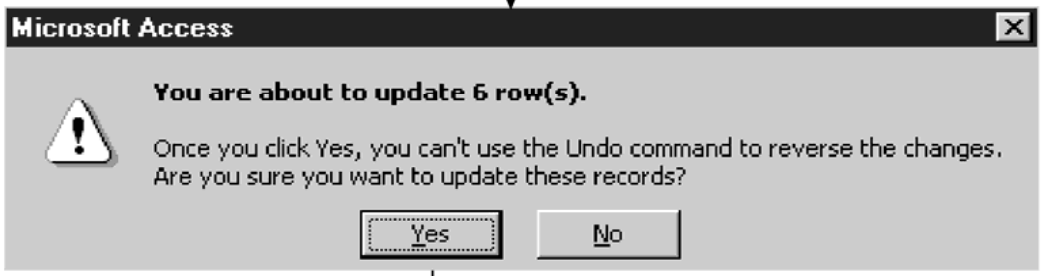
- Spreminjanje zapiso



(a)



(b)



If Yes

(c)

Query1 : Select Query

propertyNo	street	city	rent
PA14	16 Holhead	Aberdeen	715
PG16	5 Novar Drive	Glasgow	495
PG21	18 Dale Road	Glasgow	660
PG36	2 Manor Road	Glasgow	412
PG4	6 Lawrence Street	Glasgow	385
PL94	6 Argyll Street	London	440

Record: 1 of 6

Values updated in **rent** column by 10%

(d)

```
UPDATE PropertyForRent SET PropertyForRent.rent = [rent]*1.1;
```

Poglavje 4

Nivoji abstrakcije, podatkovna neodvisnost in Osnove SUPB

Povzeto po [2]

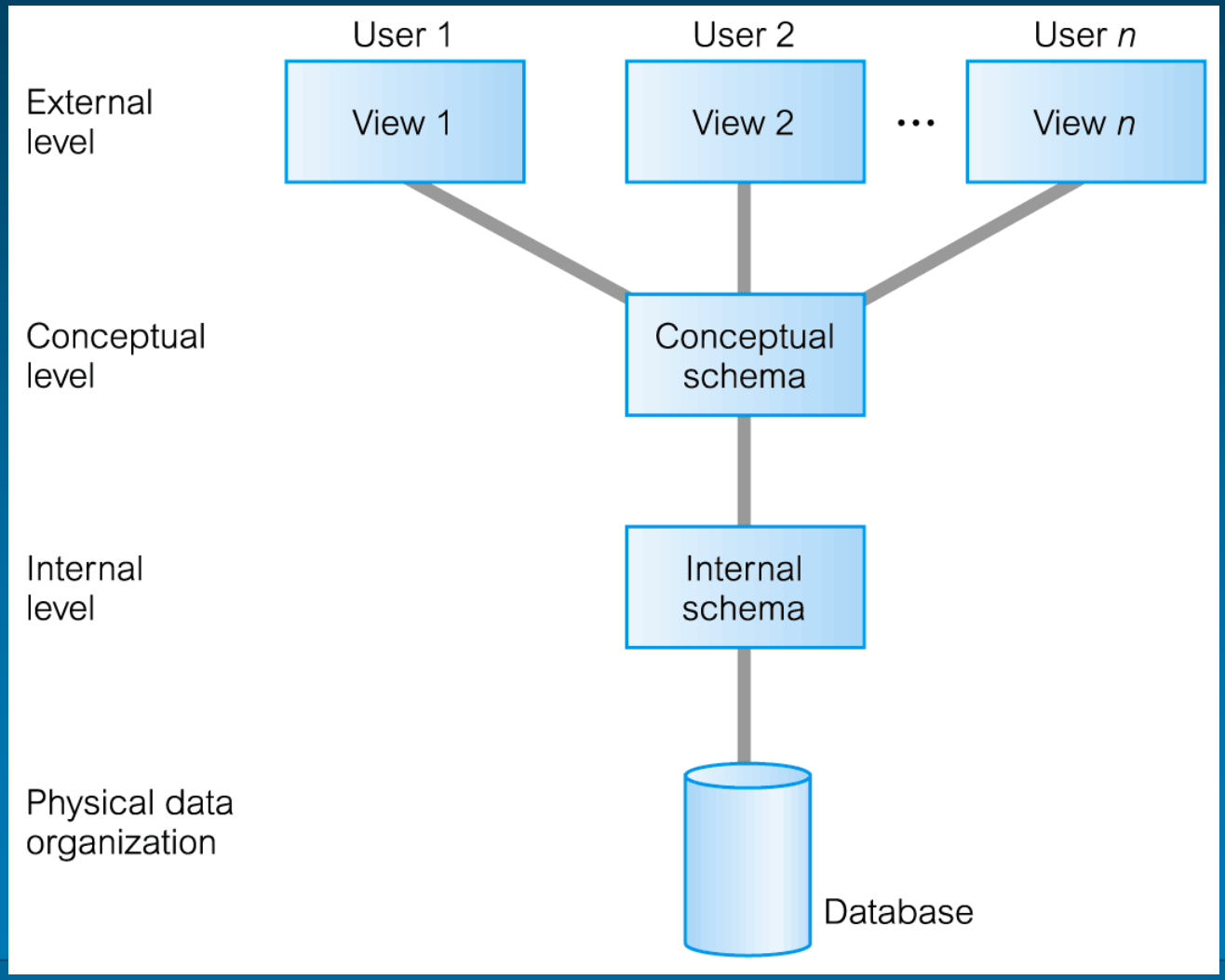
Zakaj več nivojev abstrakcije?..

- **Vsi uporabniki morajo imeti možnost dostopa do istih podatkov, a vsak ob različnem pogledu na iste podatke**
- **Spremembe pogleda na podatke enega uporabnika, ne smejo vplivati na poglede ostalih uporabnikov**
- **Uporabnikom naj ne bo potrebno poznati podrobnosti fizičnega nivoja, niti podrobnosti o tem, kako in kje so shranjeni podatki ter kako se z njimi manipulira**

Zakaj več nivojev abstrakcije?

- **DBA mora imeti možnost spremeniti fizične parametre podatkovnih struktur ne da bi to kaj vplivalo na poglede uporabnikov**

Nivoji abstrakcije..



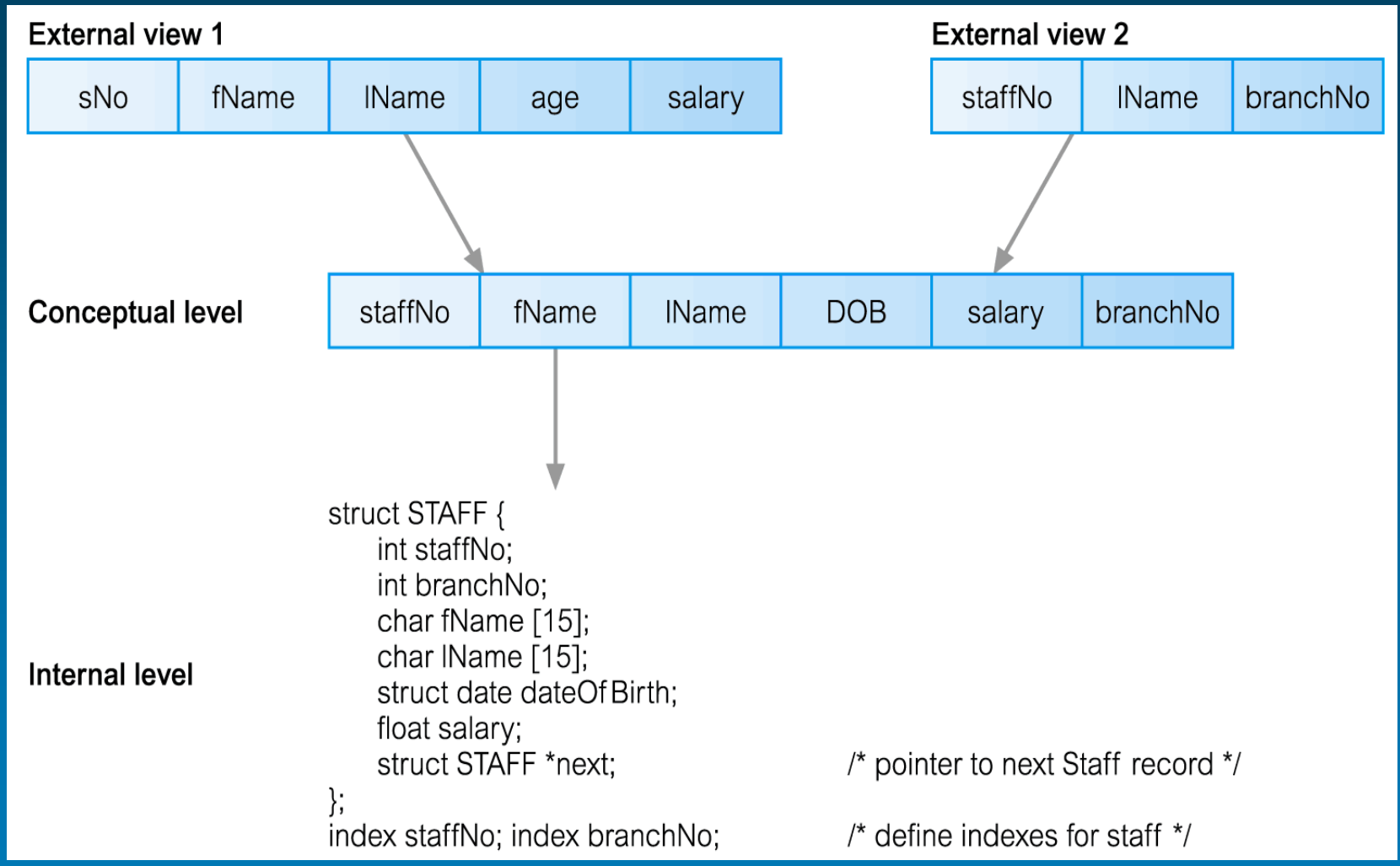
Nivoji abstrakcije..

- **Zunanja shema – zunanji nivo (External level)**
  - **Uporabnikov pogled na podatke**
  - **Predstavlja tisti del PB, ki je relevanten za posameznega uporabnika**
  - **Različni pogledi imajo lahko različne predstavitve istega podatka (EMŠO, datum rojstva)**
- **Konceptualna shema – konceptualni nivo (Conceptual level)**
  - **Združen pogled na podatke**
  - **Opisuje kateri podatki so shranjeni v PB in razmerja med njimi**
  - **Ne vsebuje podrobnosti o fizičnih parametrih (atribut in podatkovni tip, brez fizičnih podrobnosti)**
  - **Opisuje integritetne omejitve**

Nivoji abstrakcije..

- **Fizična shema - fizični nivo (Internal level)**
  - **Fizična predstavitev in organizacija podatkov v PB**
  - **Opisuje, kako in kje (v fizičnem smislu) so podatki shranjeni v PB**
- **Vsaka podatkovna baza ima le eno konceptualno in fizično shemo**

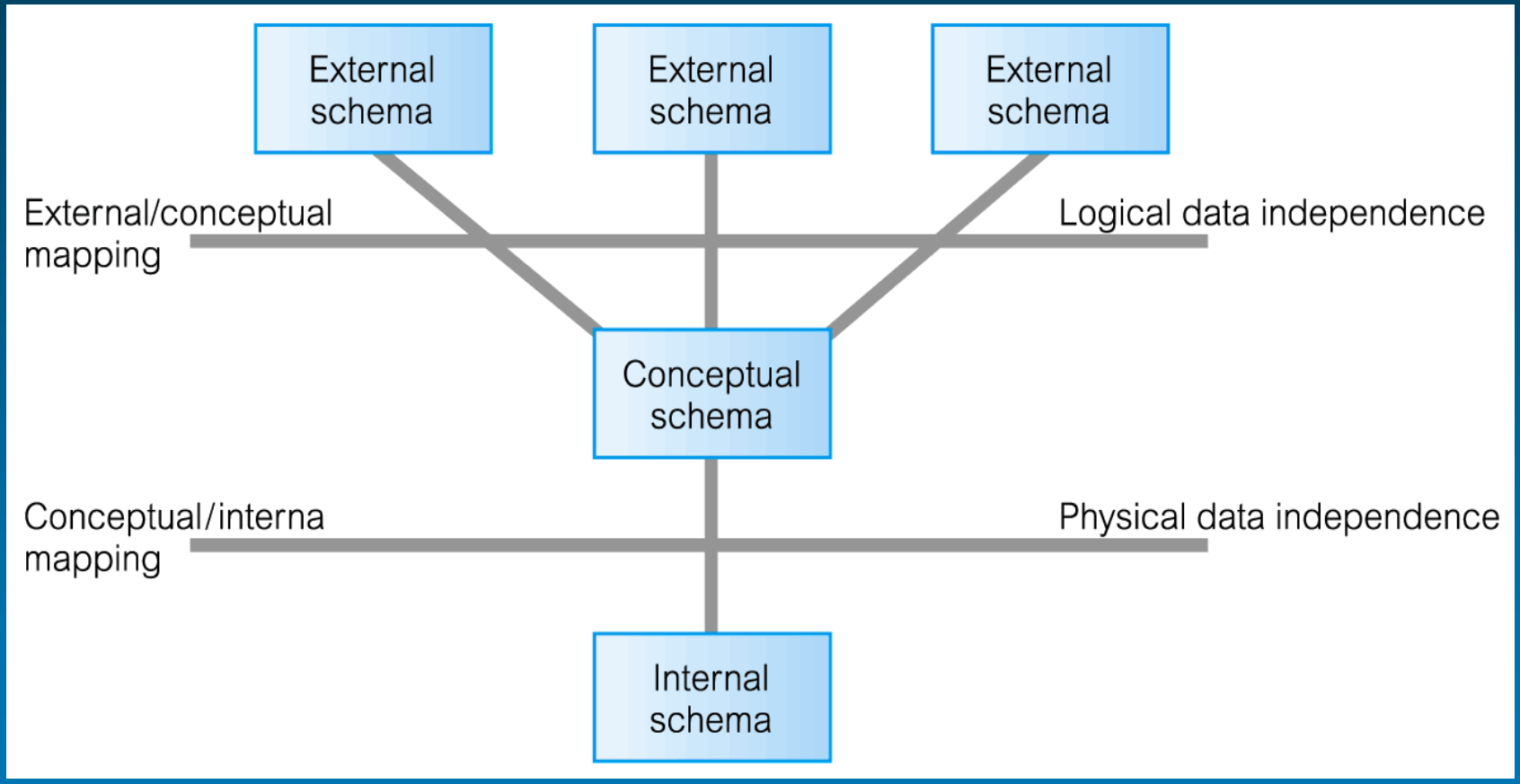
Nivoji abstrakcije



Podatkovna neodvisnost..

- **Zagotavlja, da višji nivo ostane nespremenjen kljub spremembi na nižjem nivoju**
- **Logična podatkovna neodvisnost**
  - **Zagotavlja, da ostanejo ob spremembah v konceptualni shemi zunanje sheme nespremenjene**
  - **Omogoča, da ostanejo aplikacije (njihova programska koda) kljub (morebitnim) spremembam v konceptualni shemi nespremenjene**

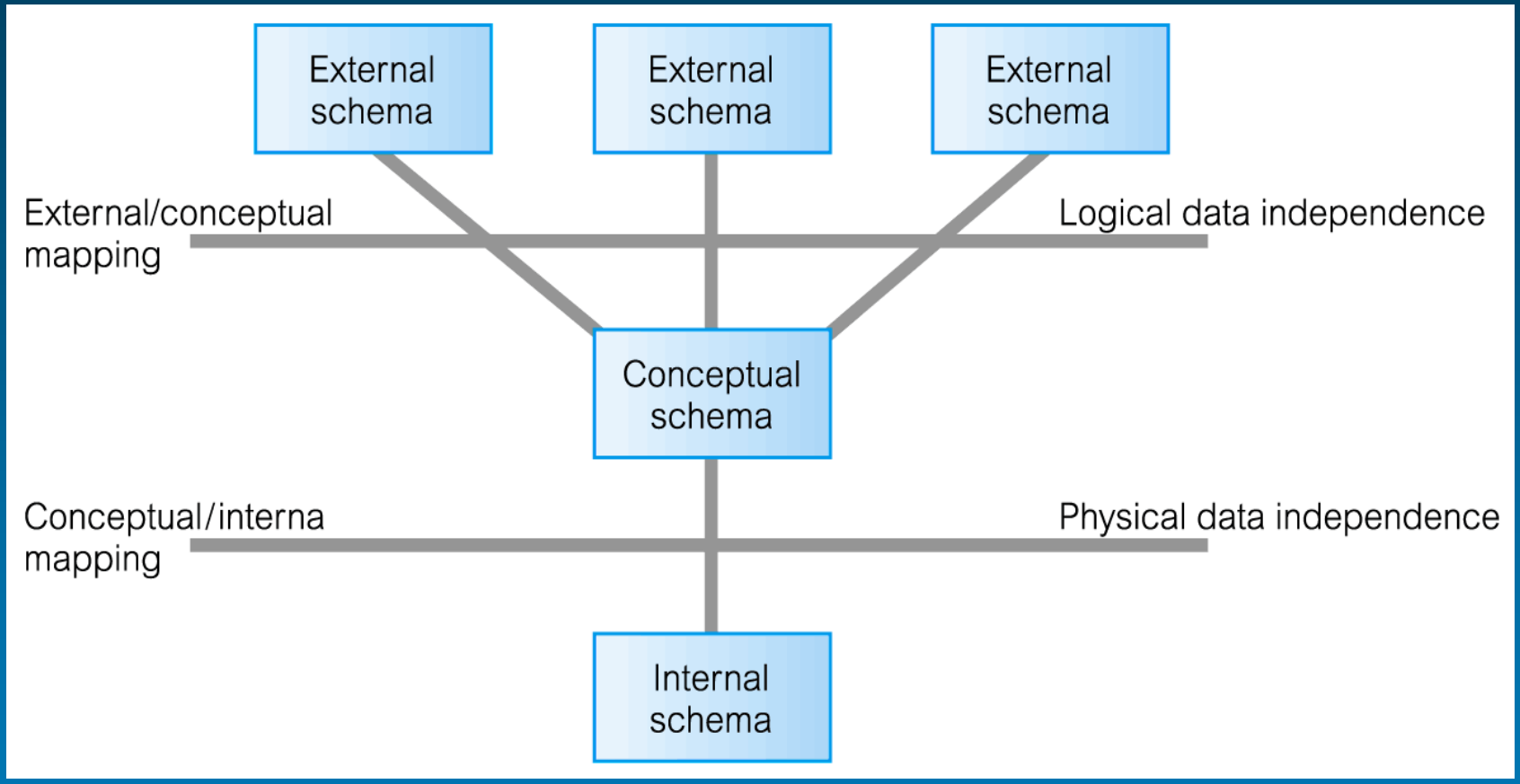
Podatkovna neodvisnost..



Podatkovna neodvisnost..

- **Fizična podatkovna neodvisnost**
  - **Zagotavlja, da ob spremembi v fizični shemi ostane konceptualna shema nespremenjena (zamenjava podatkovnega strežnika, …)**

Podatkovna neodvisnost..



Podatkovna neodvisnost..

- Zelo pomembna lastnost SUPB je, da omogoča podatkovno neodvisnost - programi so neodvisni od načina shranjevanja in strukturiranja podatkov v PB.
- Podatkovno neodvisnost dosežemo z uporabo tri-nivojske abstrakcije podatkov:
  - Če se spremeni konceptualna shema, lahko zunanjo shemo priredimo tako, da pogledi ostanejo nespremenjeni → logična podatkovna neodvisnost.
  - Podobno konceptualna shema ločuje uporabnike od sprememb, ki se naredijo na fizični PB → fizična podatkovna neodvisnost.

Podatkovna neodvisnost

- Konceptualna shema skrije podrobnosti o tem, kako so podatki dejansko shranjeni na disku, o strukturi datotek in o indeksih.
- Dokler ostaja konceptualna shema nespremenjena, spremembe na fizičnem nivoju ne vplivajo na programe, ki podatke uporabljajo.
- Lahko pa spremembe vplivajo na učinkovitost.

DDL in DML..

- **DDL (ponovitev):**
  - **Omogoča definiranje podatkovnih struktur, podatkovnih tipov ter integritetnih omejitev**
  - **Vse specifikacije oz. definicije so shranjene v podatkovni bazi (podatkovni slovar ali sistemski katalog). Govorimo o metapodatkih**
  - **Kreiranje tabele pomeni poseg v sistemski katalog**
  - **Praviloma se v okviru SUPB uporablja isti DDL za vse tri sheme (ali vsaj za zunanjo in konceptualno)**

DDL in DML..

- **DML**
  - Omogoča manipuliranje s podatki (Insert, Update, Delete)
- **Jeziki četrte generacije (4GLs)**
  - So pretežno ne-proceduralni: v večini primerov lahko specificiramo, *kaj* je potrebno narediti in ne tudi *kako*
  - SQL, QBE
- **Poleg 4GLs na učinkovitost razvoja vplivajo tudi:**
  - Generatorji form
  - Generatorji izpisov
  - Generatorji grafikonov
  - Generatorji aplikacij

DDL in DML

- **Proceduralni DML**
  - Omogoča, da specificiramo, kako pridobiti in spremeniti podatke
- **Ne-proceduralni (deklarativni) DML**
  - Omogoča, da za potrebe operacije podamo lastnost podatkov, ki nas zanimajo za potrebe pridobitve podatkov ali za potrebe njihove transformacije
  - SQL (Structured Query Language) – Poizvedovalni jezik

Poizvedovanje v PB..

- **Enostavnost pridobivanja informacij iz PB je ključna prednost SUPB za uporabnike**
- **Relacijske PB omogočajo uporabnikom postavljati enostavna vprašanja (poizvedbe), s katerimi pridobivajo podatke/informacije**
- **Poizvedbe se podaja v jeziku, ki je prirejen za opisovanje poizvedb - poizvedovalni jezik**
- **Relacijski model podpira zelo močne poizvedovalne jezike**
- **Ena pomembnih nalog SUPB je optimizacija poizvedb, tako da se te čim hitreje izvedejo**

Poizvedovanje v PB

- Učinkovitost poizvedb je močno odvisna od načina, kako so podatki shranjeni v fizični obliki ter indeksirani
- SUPB omogoča uporabnikom izvajati poizvedbe ter kreirati in posodabljati vrednosti s pomočjo DML

Upravljanje s transakcijami

- **Transakcija** predstavlja skupek ažuriranj, ki jih izvede transakcijski program
- Z vidika SUPB predstavlja transakcija osnovno enoto spremembe → transakcija se mora izvesti v celoti ali sploh ne
- Dve pomembni nalogi SUPB pri izvajanju transakcij:
  - **Zagotavljanje sočasnosti** pri izvajanju transakcij in
  - **Obnavljanje PB** po transakcijskih in sistemskih nesrečah (razveljavljanje, ponavljanje transakcij...)

Relacijski podatkovni model

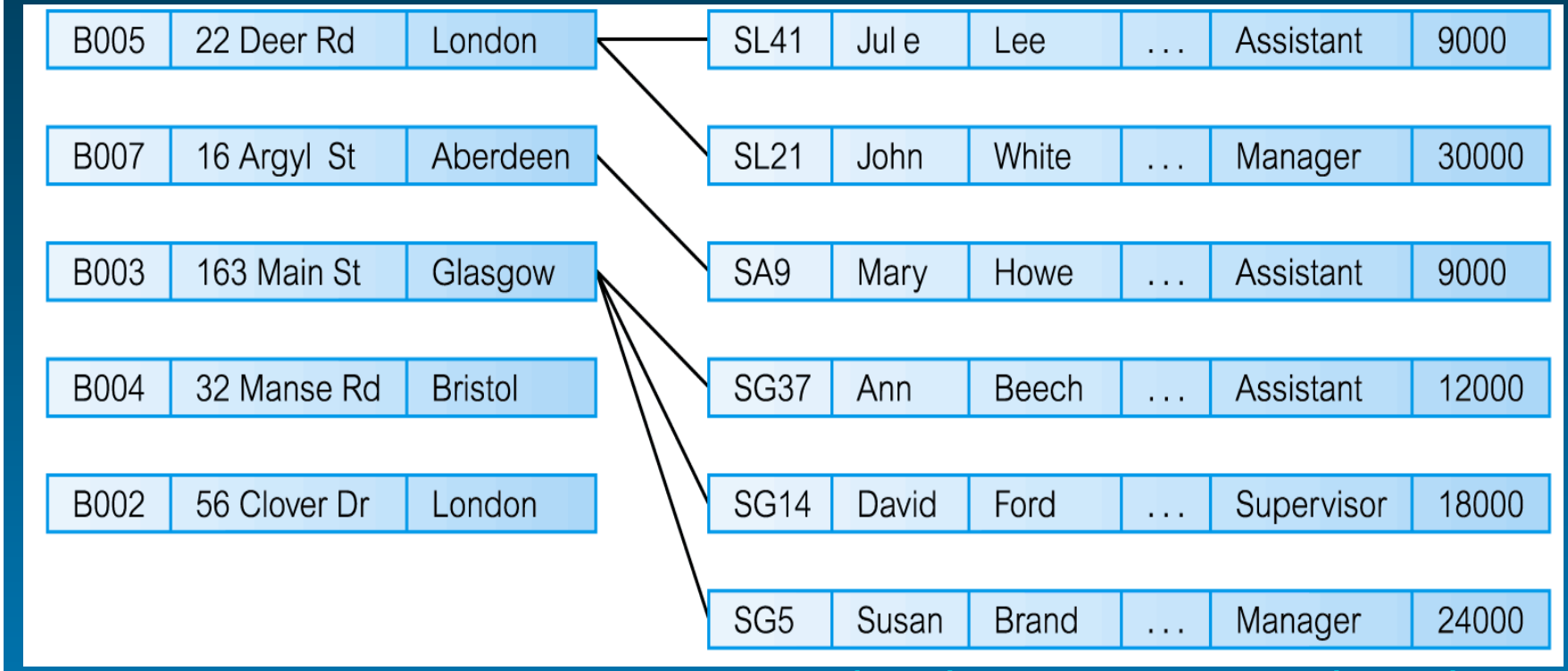
Branch

branchNo	street	city	postCode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Staff

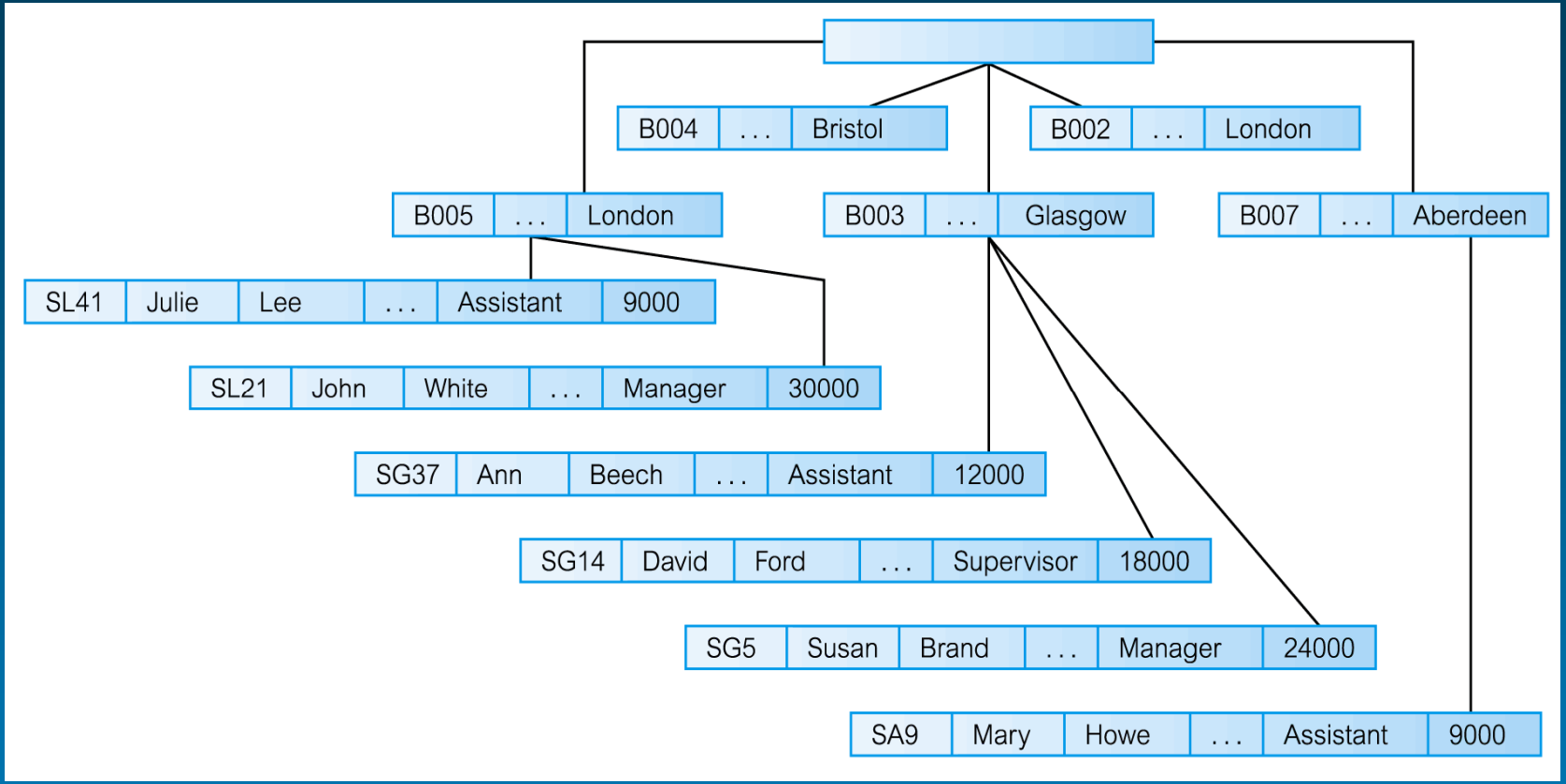
staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Mrežni podatkovni model



- Zapis je v razmerju s setom (drugih zapisov) oz. set lastnikov in set članov sta v razmerju
- Glej [4, 159-178]

Hierarhični podatkovni model



- Struktura Gozd; v njej so zapisi povezani v razmerja dreves
- Glej [4, 181-206]

Splošno o podatkovnih bazah (PB)

- Stanje danes:
  - Organizacije odvisne od zmožnosti pridobivanja natančnih in pravočasnih podatkov...
  - Podatki predstavljajo konkurenčno prednost.
  - Brez zmožnosti za upravljanje z velikimi količinami podatkov in zmožnosti za hitro iskanje ustreznih podatkov postanejo podatki breme za organizacijo.
  - Paradoks: zaradi preveč informacij potrebujemo še več informacij
  - Potrebujemo ustrezne mehanizme za upravljanje s podatki in učinkovito iskanje po njih → Podatkovne baze.
- Definicija: Podatkovna baza je mehanizirana, večuporabniška, formalno definirana in centralno nadzorovana zbirka podatkov (glej [4,15]).

Sistemi za upravljanje s PB (SUPB)

- Sistem za upravljanje s podatkovno bazo – SUPB je programska oprema za obvladovanje velikih količin podatkov.
- Alternativa – shranjevanje v aplikaciji lastni obliki; problemi: neprenosljivost idr.
- Obstaja veliko vrst SUPB. Omejili se bomo predvsem na relacijske ter omenili objektne.
- Primeri SUPB: Oracle, Sybase, DB2, MS SQL, Ingres, Postgres, MySQL, ObjectStore, Jasmine, Objectivity/DB, Versant Object Database,...

Datotečni sistemi in SUPB..

- Omejitve datotečnih sistemov:
  - Ločevanje in izolacija podatkov
    - Vsaka aplikacija* obvladuje svoje podatke
    - V okviru ene aplikacije lahko dostopamo le do podatkov te aplikacije (ne vemo za razpoložljive podatke v drugih aplikacijah)
  - Podvajanje podatkov
    - Podvajanje vnosa – iste podatke vnašamo večkrat in na več mest
    - Odvečna poraba prostora na disku ter potencialna možnost za neskladnost

*Definicija: Aplikacija je računalniški program, ki v okviru IS delno ali v celoti podpira enega ali več poslovnih procesov in za shranjevanje podatkov uporablja podatkovno bazo.
Sinonimi: informacijska rešitev, program, aplikativni sistem…

Datotečni sistemi in SUPB..

- Omejitve datotečnih sistemov (nadaljevanje):
  - Podatkovna odvisnost
    - Struktura podatkov je definirana v aplikaciji; sprememba v strukturi podatkov zahteva spremembo v aplikaciji
  - Neskladnost med oblikami datotek
    - Aplikacije, napisane v različnih programskih jezikih, ne morejo enostavno dostopati do datotek drugih sistemov

Datotečni sistemi in SUPB..

- Primer:
  - Imamo podjetje, ki ima veliko količino podatkov (okoli 500 GB) o zaposlenih, oddelkih, izdelkih, prodaji itd.
  - Do podatkov sočasno dostopa več uporabnikov.
  - Odgovore potrebujemo hitro, spreminjanje podatkov mora ohranjati njihovo konsistentnost.
  - Dostop do nekaterih podatkov mora biti omejen.
- Ali lahko uporabimo shranjevanje v datotečni sistem?

Datotečni sistemi in SUPB..

- Problemi pri shranjevanju v datoteke OS:
  - Potrebujemo 500 GB pomnilnika, sicer moramo podatke hraniti na diskih in jih po potrebi prenašati v pomnilnik.
  - Tudi če imamo 500 GB pomnilnika, ga neposredno lahko z 32 biti naslovimo le 4 GB. Potrebujemo mehanizme za dostop do ostalih podatkov.
  - Za vsako poizvedbo moramo izdelati program, ki obvlada iskanje po velikih količinah podatkov.
  - Za zagotovitev skladnosti podatkov potrebujemo mehanizme, ki močno povečajo kompleksnost programov.
  - Zagotoviti moramo obnavljanje podatkov v primeru ‘nesreč’.
  - OS zagotavlja le varnost z uporabniškimi imeni in gesli. Ne zadošča za opredeljevanje pravic uporabnikov nad različnimi podatki.

Datotečni sistemi in SUPB..

- Zaradi težav oziroma neučinkovitosti shranjevanja podatkov neposredno v datoteke, se pojavijo Sistemi za upravljanje s podatkovnimi bazami – SUPB.
- Definicija: SUPB je skupek programske opreme, ki omogoča kreiranje, vzdrževanje in nadzor nad dostopom do podatkov v PB.

Datotečni sistemi in SUPB..

- SUPB uporablja različne mehanizme za upravljanje s podatki:
  - Kreiranje podatkovnih struktur je omogočeno z jezikom DDL - Data Definition Language.
    - Omogoča definiranje podatkovnih struktur in tipov ter integritetnih omejitev
    - Vse specifikacije so shranjene v PB (podatkovni slovar, sistemski katalog). Kreiranje tabele pomeni poseg v sistemski katalog.
  - Vzdrževanje podatkov (Create, Insert, Update, Delete) izvajamo z uporabo jezika DML - Data Manipulation Language.
  - Za izvajanje povpraševanja obstajajo povpraševalni jeziki (query language)

Datotečni sistemi in SUPB..

- SUPB zagotavlja nadzor nad dostopom do podatkov:
  - Varnost: dostop do podatkov v skladu z avtorizacijo
  - Skladnost: zagotavlja skladnost podatkov
  - Sočasni dostop: zagotavlja in nadzira sočasni dostop
  - Obnova: zagotavlja mehanizme za obnovo podatkov

Datotečni sistemi in SUPB..

- Uporaba SUPB prinaša naslednje prednosti:
  - Podatkovna neodvisnost: Programi so neodvisni od predstavitve podatkov in načina shranjevanja podatkov. SUPB zagotavlja abstrakcijo podatkov in ločuje programe od podrobnosti predstavitve podatkov.
  - Učinkovit dostop do podatkov: SUPB zagotavlja tehnike za učinkovito hranjenje in dostop do podatkov.
  - Varnost in integriteta podatkov: Če se do podatkov dostopa preko SUPB, se lahko uporabi omejitve, ki zagotavljajo skladnost podatkov.

Datotečni sistemi in SUPB..

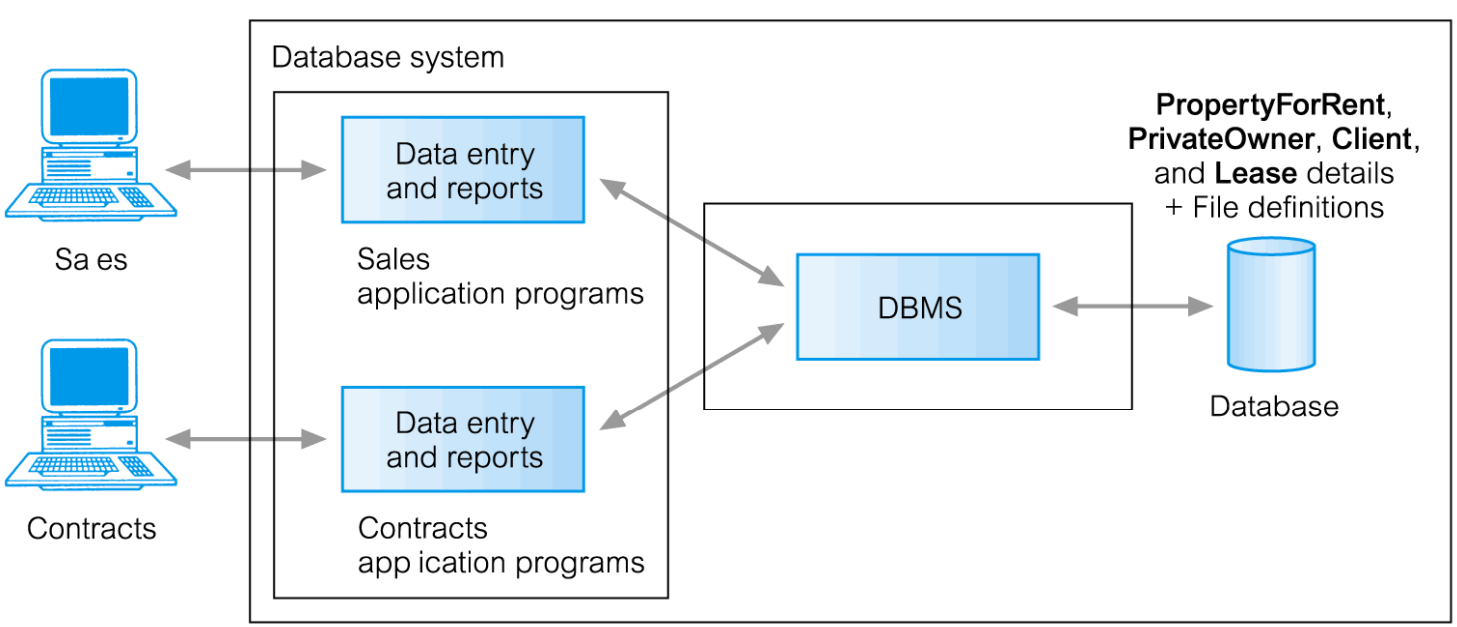
- Prednosti uporabe SUPB (nadaljevanje):
  - Administracija podatkov: Če so podatki shranjeni centralno, je upravljanje s podatki lažje.
  - Sočasen dostop do podatkov in obnavljanje PB: SUPB razporeja sočasne dostope tako, da izgleda, kot da do podatkov dostopa en uporabnik. Omogoča tudi obnavljanje PB po nesrečah.
  - Skrajša čas razvoja programov: SUPB podpira številne mehanizme za dostop do podatkov.

Datotečni sistemi in SUPB..

- Zaključek: SUPB prinaša veliko prednosti v primerjavi s hranjenjem podatkov neposredno v datotekah.
- Obstajajo (redke) izjeme, ko uporaba SUPB ni primerna. Npr.:
  - Za specializirane aplikacije (npr.: v realnem času) klasični SUPB niso primerni. Za te aplikacije se raje izdela namensko kodo za rokovanje s podatki.
  - Če SUPB ne podpira dela s podatki na način, ki ga zahteva program.

Datotečni sistemi in SUPB..

- Shema SUPB



PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo)
PrivateOwner (ownerNo, fName, lName, address, telNo)
Client (clientNo, fName, lName, address, telNo, prefType, maxRent)
Lease (leaseNo, propertyNo, clientNo, paymentMethod, deposit, paid, rentStart, rentF nish)

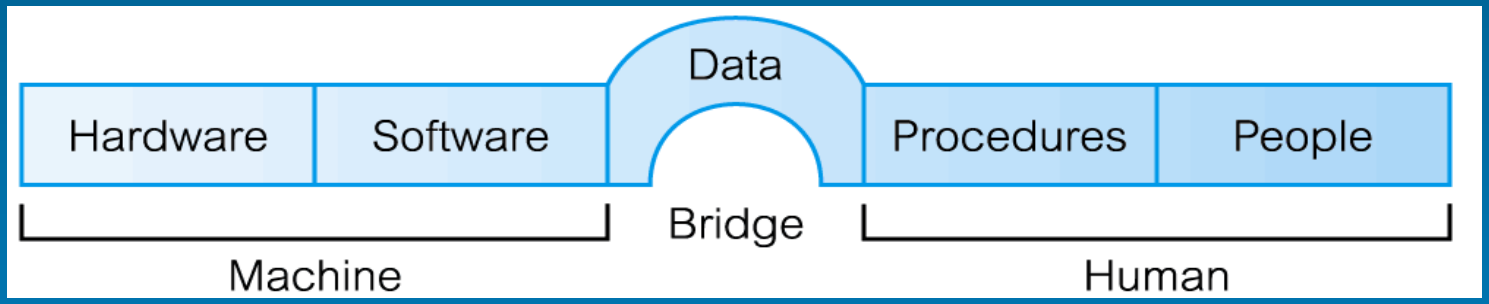
Vir: [2,XX]

Datotečni sistemi in SUPB..

- Razmerje med SUPB in PB:
  - En SUPB upravlja z eno ali več instanc PB na enem podatkovnem strežniku
  - Na enem podatkovnem strežniku je lahko nameščenih več SUPB

Datotečni sistemi in SUPB..

- Komponente SUPB
  - Strojna oprema
  - Programska oprema
  - Podatki
  - Postopki
  - Ljudje



Vir: [2,XX]

Datotečni sistemi in SUPB..

- Komponente SUPB
  - Strojna oprema:
    - Podatkovni strežnik
    - Pomembna parametra za strežnik: dovolj pomnilnika, hitri pomnilnik in diskovni prostor
    - Ostala strojna oprema
  - Programska oprema:
    - SUPB, operacijski sistem, omrežna programska oprema
    - Različna sistemska oprema (razvojna orodja, orodja za dostop do podatkov, orodja za upravljanje s PB in SUPB)
    - Specializirane (poslovne) aplikacije

Datotečni sistemi in SUPB

- Komponente SUPB (nadaljevanje):
  - Postopki
    - Načini prijave
    - Uporaba posameznih orodij
    - Zagon (startup) in zaustavitev (shutdown) podatkovne baze
    - Izdelava varnostnih kopij
    - Obvladovanje nesreč/okvar
  - Ljudje, njihove vloge
    - Skrbnik podatkov
    - Skrbnik podatkovne baze (DBA – Database Administrator)
    - Analitik, načrtovalec PB
    - Razvijalci aplikacij
    - Končni uporabniki (izkušeni, neizkušeni)

Funkcije SUPB..

- **Hranjenje, pridobivanje (branje) in ažuriranje podatkov**
- **Razpoložljivost (vsem uporabnikom, v skladu s privilegiji) sistemskega kataloga (podatkovnega slovarja):**
  - **Struktura podatkov (tabel, indeksov, pogledov, ..) in ustrezni opisi (atributi, podatkovni tipi, …)**
  - **Integritetne omejitve**
  - **Vloge, uporabniki ter njihovi privilegiji (nivoji privilegijev) za uporabo podatkov in ostalih objektov**
  - **Opisi zunanje, konceptualne in fizične sheme ter preslikave med njimi**
  - **Statistike uporabe podatkov**

Funkcije SUPB..

- **Pomen koncepta sistemskega kataloga:**
  - Podatki o podatkih so shranjeni centralno
  - Centralno definiran pomen podatkov omogoča uporabnikom lažjo uporabo in interpretacijo podatkov
  - Redundanco je lažje odkriti, ker gre za centralizacijo
  - Spremembe, ki se izvajajo na strukturi podatkov so zabeležene
  - Lažja vzpostavitev sistema varnosti in zaupnosti
  - Omogoča zagotavljanje integritete
- **Obvladovanje (nadzor in upravljanje) delovanja transakcij**
- **Nadzor nad sočasno uporabo podatkov oz. sočasnim izvajanjem transakcij**
- **Obnavljanje stanja PB**

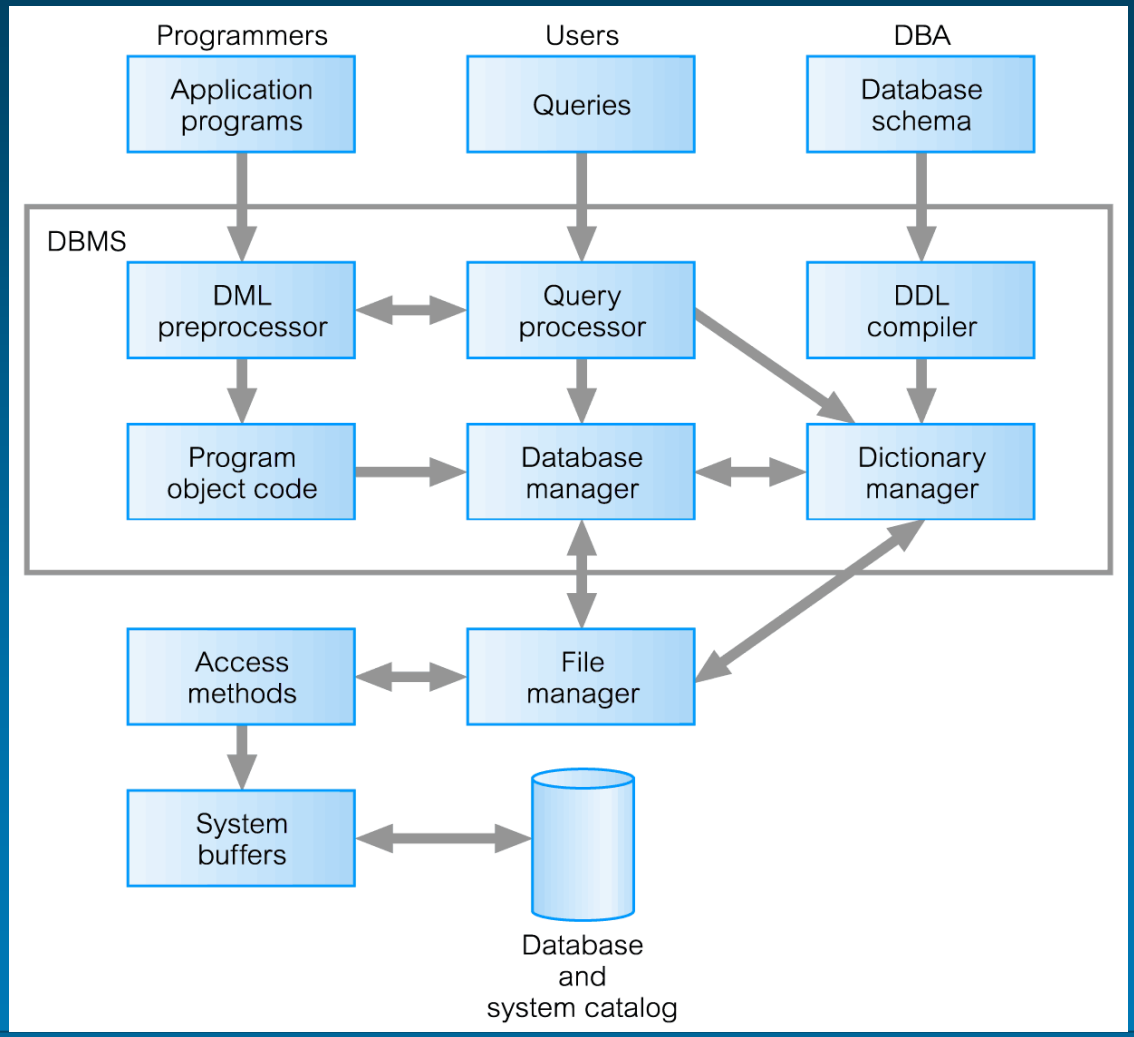
Funkcije SUPB..

- Nadzor nad zaupnostjo podatkov, avtorizacija uporabe podatkov
- Komunikacijske storitve in protokoli, ki omogočajo povezovanje s PB in uporabo podatkov
- Zagotavljanje integritete podatkov
- Zagotavljanje podatkovne neodvisnosti

Funkcije SUPB

- **Zagotavljanje različnih storitev za DBA preko različnih orodij:**
  - Izdelava varnostnih kopij in uvoz podatkov iz varnostnih kopij
  - Monitoring delovanja PB
  - Izdelovanje statistik uporabe PB
  - …

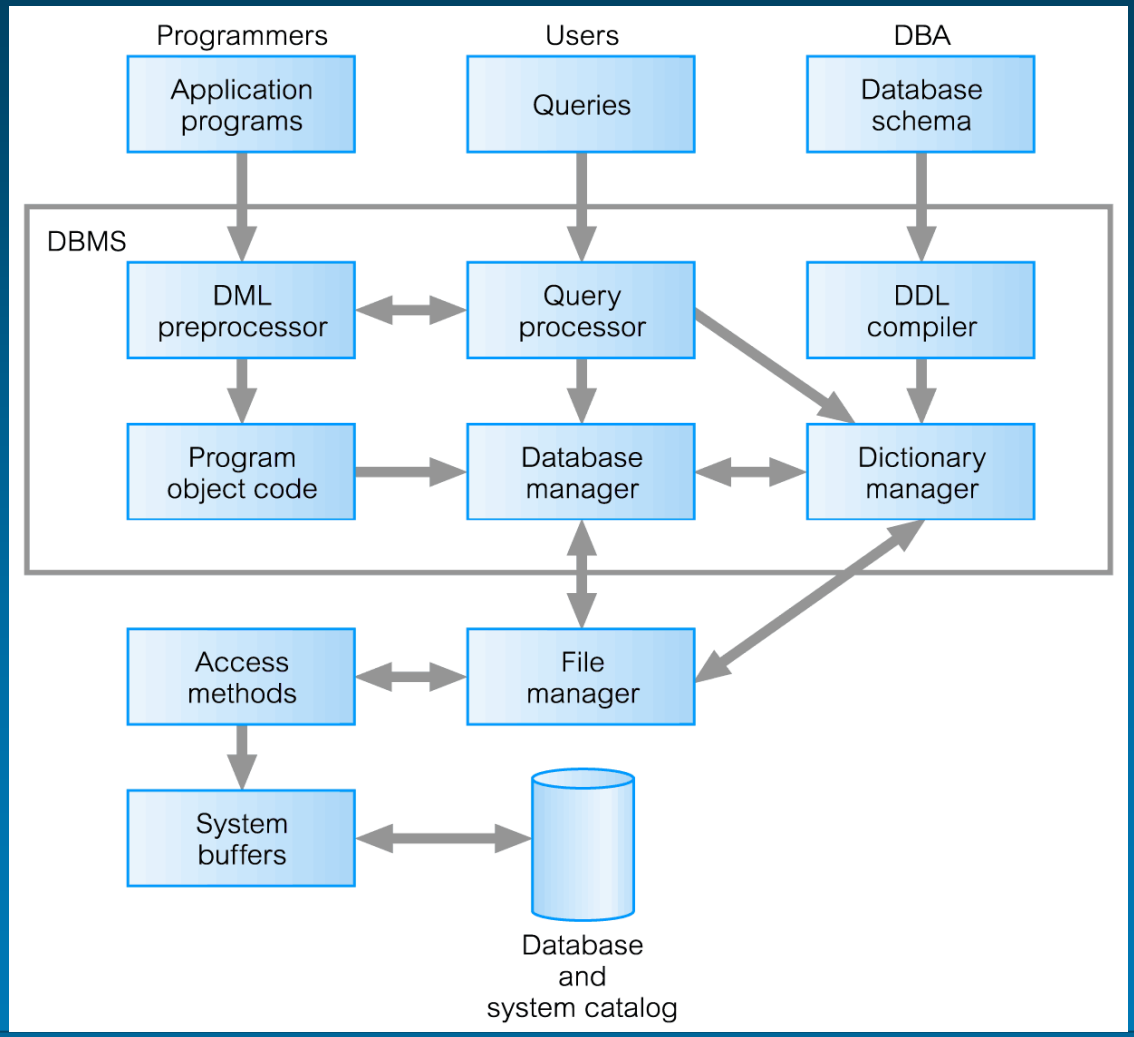
Struktura SUPB..



Struktura SUPB..

- **Query processor:** transformira poizvedbe v skupino ukazov nižje nivojskih jezikov in jih posreduje *Database Managerju*. Vsebuje tudi *Query Optimizer*
- **Database manager:**
  - prejema zahteve od Query processorja (poizvedbe uporabnikov in aplikacij) ter zahteve, ki so posledica ostalih DML ukazov
  - Preverja zunanje in konceptualne sheme, da ugotovi, kateri zapisi konceptualnega nivoja “ustrezajo” zahtevi
  - Preverja avtorizacijo
  - Posreduje zahteve naprej File Managerju

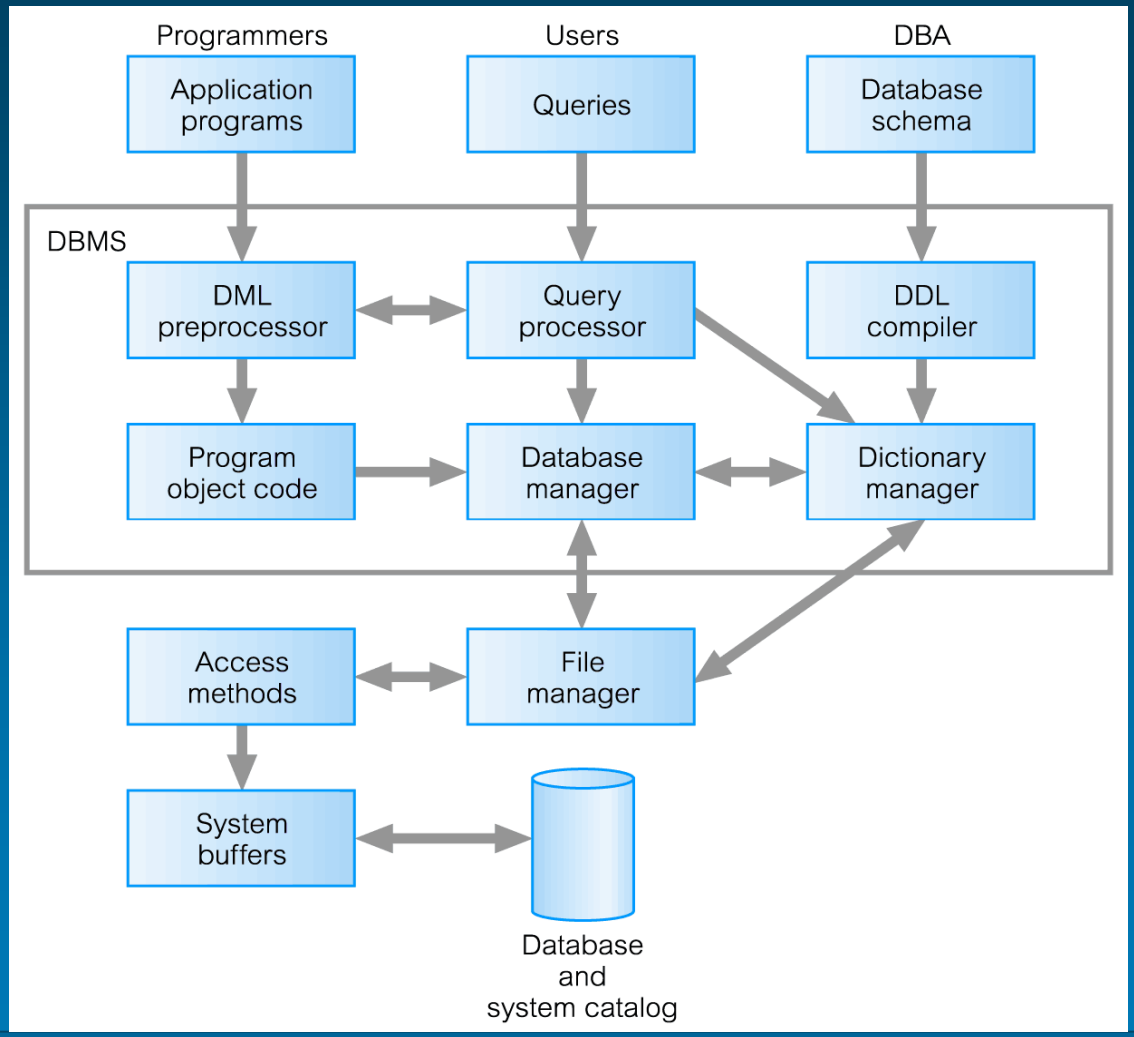
Struktura SUPB..



Struktura SUPB..

- **File manager:** upravlja s podatkovnimi datotekami oz. fizičnimi enotami, kjer se nahajajo podatki. Učinkovitost njegovega delovanja povečajo *System buffers*
- **DML preprocessor:** pretvarja DML ukaze bodisi v poizvedbe, bodisi v standardne funkcijske klice, ki se nahajajo prevedeni v *Object code*
- **DDL compiler:** na podlagi prejetega DDL ukaza izdela vse ukaze za poseg v eno ali več tabel v Data Dictionary in jih posreduje *Dictionary Managerju*

Struktura SUPB..

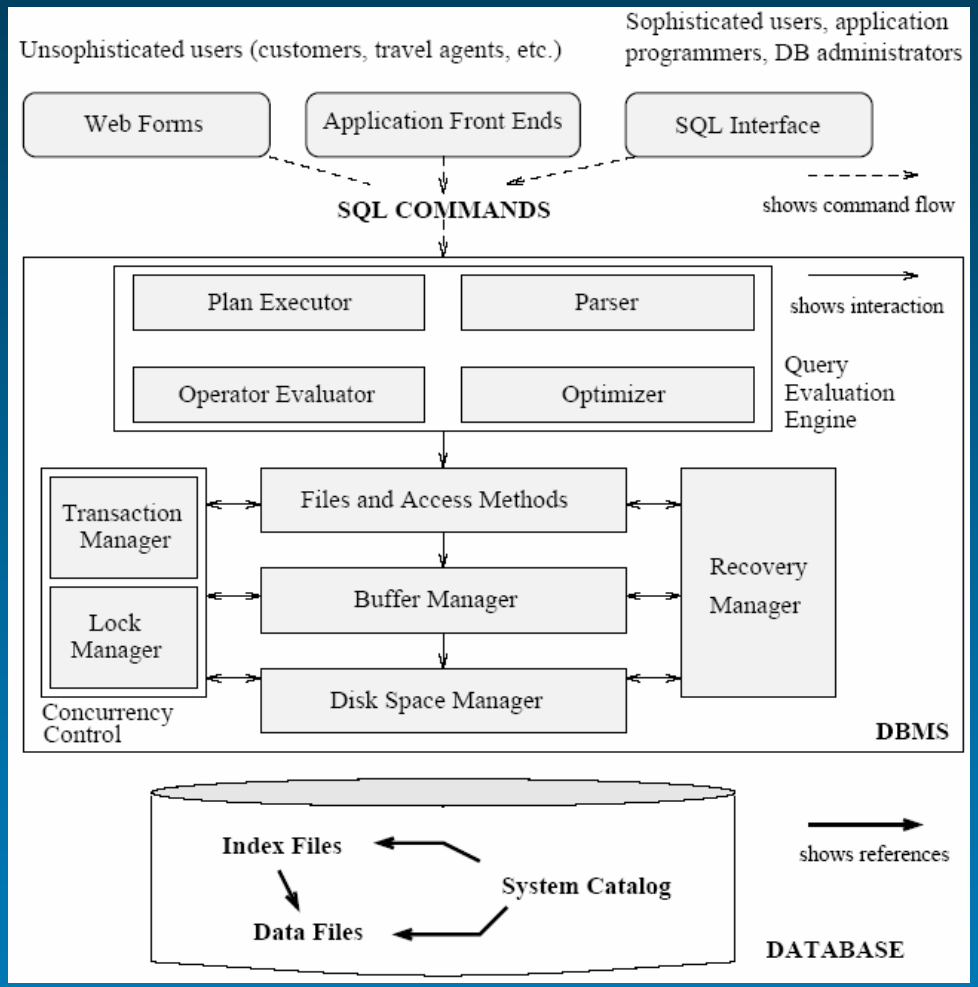


Struktura SUPB

- **Dictionary manager:**
  - Omogoča dostop do sistemskega kataloga
  - Upravlja s sistemskim katalogom

- Predstavljena je bila tipično, splošna struktura
- Posamezni SUPB imajo lahko nekoliko drugačno zasnovo, vsekakor pa lahko za iste komponente/koncepte uporabljajo druge izraze
- Oracle, IBM DB2 in MS SQL bomo spoznali preko vabljenih predavanj

Tipična zgradba SUPB – drug pogled..

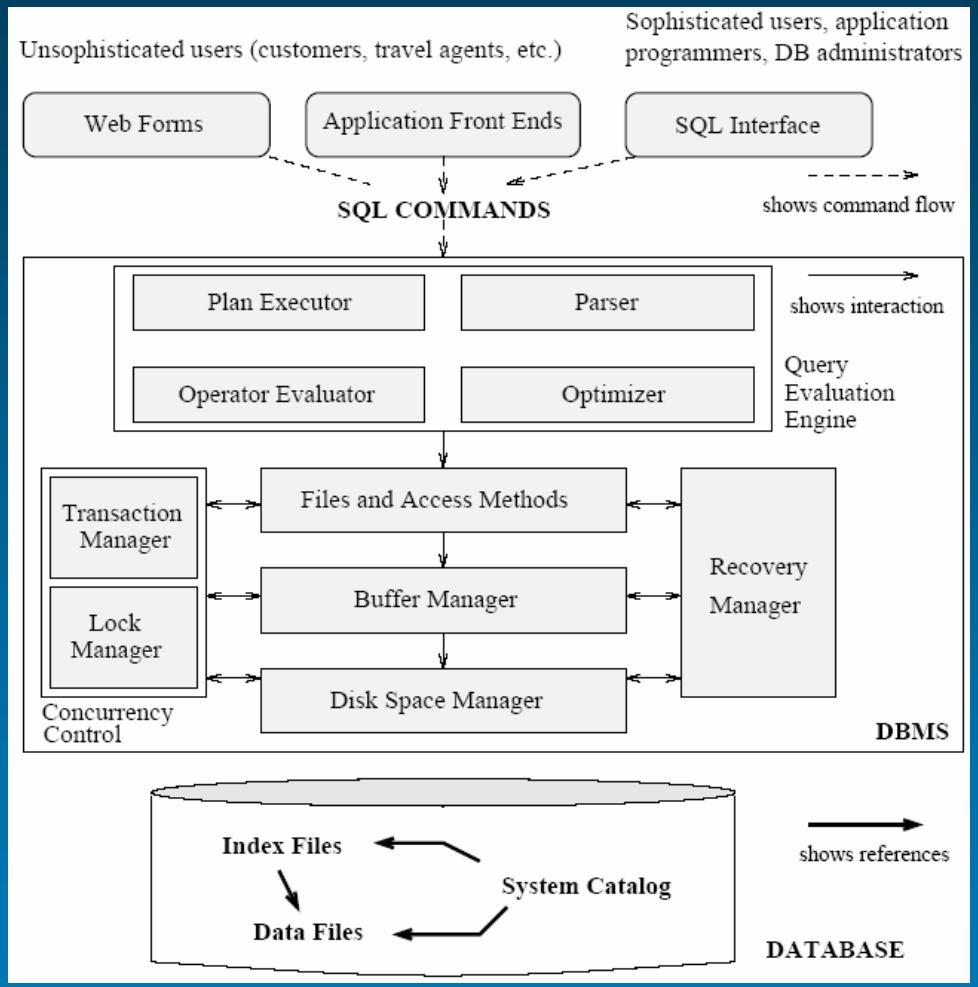


Vir: [1,18]

Tipična zgradba SUPB - drug pogled..

- Funkcije posameznih enot SUPB:
  - Stroj za evaluacijo poizvedb (Query Evaluation Engine)
    - Sintaktični analizator (Parser): Sintaktično analizira poizvedbo, ki jo SUPB-ju posreduje aplikacija.
    - Optimizator (Optimizer): Na podlagi informacij o tem, kako so podatki shranjeni, izdela učinkovit plan za izvajanje poizvedbe. Plan izvajanja predstavlja načrt za izvedbo poizvedbe in je ponavadi predstavljen kot drevo relacijskih operatorjev.
    - Evaluator operatorjev (Operator Evaluator): Na osnovi plana izvajanja analizira poizvedbo in ugotovi, če lahko operatorje nadomesti z bolj učinkovitimi operatorji
    - Izvajalec plana (Plan Executor): Izvede poizvedbo po navodilih plana poizvedbe.

Tipična zgradba SUPB – drug pogled..

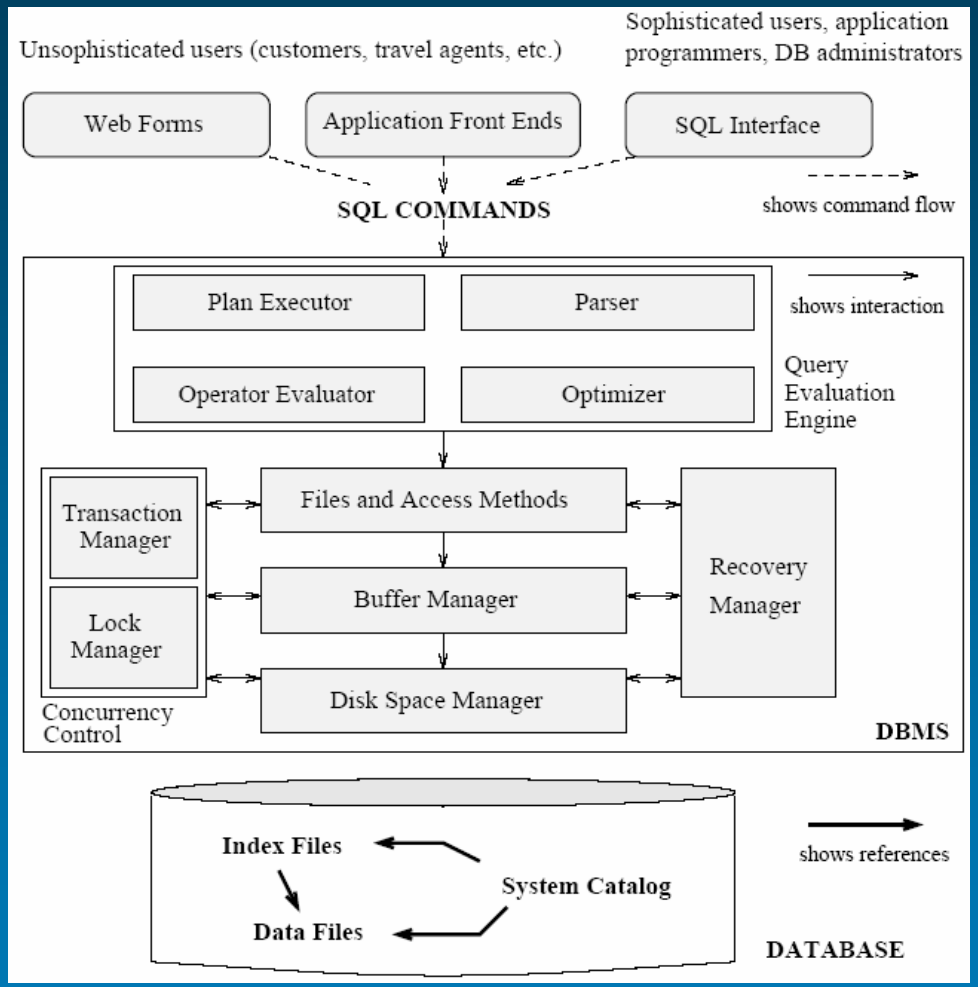


Vir: [1,18]

Tipična zgradba SUPB – drug pogled..

- Funkcije posameznih enot SUPB (nadaljevanje):
  - Datoteke in metode dostopa (Files and Access Methods): enota, ki omogoča delo z datotekami.
  - Upravljalec pomnilnika (Buffer Manager): Prenaša strani iz diska v pomnilnik glede na bralne potrebe.
  - Upravljalec prostora na disku (Disk Space Manager): Najnižji nivo SUPB je zadolžen za upravljanje z diskom. Vse operacije višjih plasti se tukaj prevedejo v nizko-nivojske ukaze za delo z diskom.

Tipična zgradba SUPB – drug pogled..

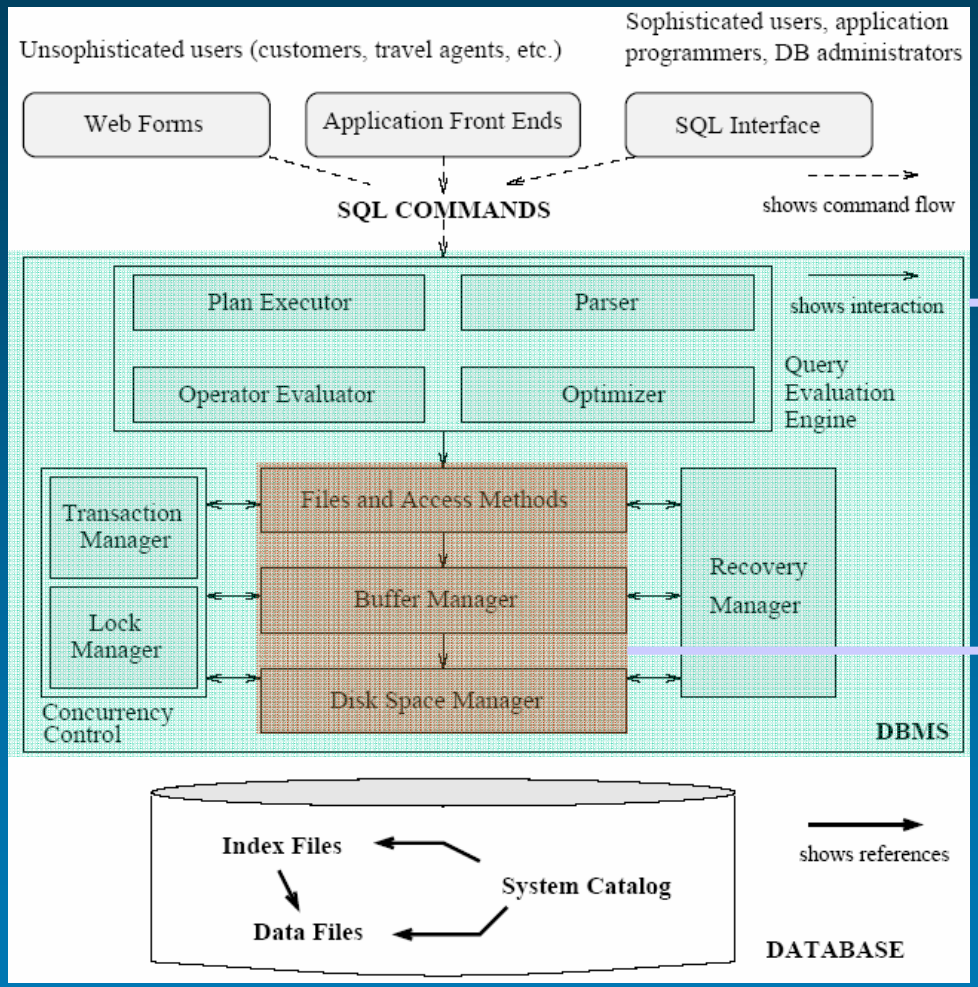


Vir: [1,18]

Tipična zgradba SUPB - drug pogled..

- Funkcije posameznih enot SUPB (nadaljevanje):
  - Enota za nadzor sočasnosti (Concurrency Control):
    - Upravljalec transakcij (Transaction Manager): Zagotavlja zaseganje podatkov z uporabo določenih protokolov in skrbi za razporejanje izvajanja transakcij.
    - Upravljalec zaklepanja (Lock Manager): Vzdržuje informacije o zahtevanih in odobrenih zaseženjih podatkov.
  - Upravljalec reševanja podatkov (Recovery Manager): Vzdržuje dnevnik in skrbi za obnavljanje sistema v zadnje skladno stanje pred nesrečo.

Struktura SUPB – drug pogled



Osnovne komponente SUPB

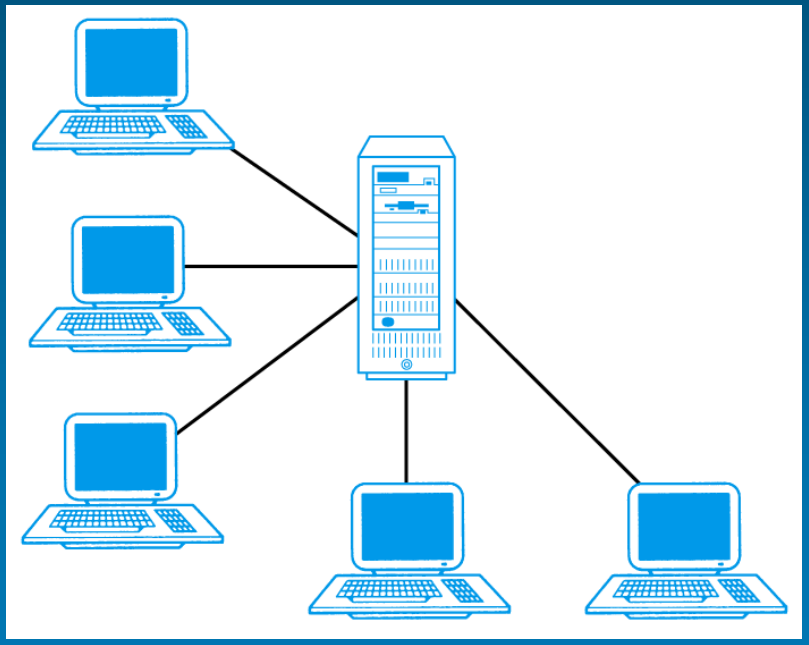
Centralne komponente - sodelujejo z vsemi ostalimi komponentami

Arhitekture (uporabe SUPB)

- **Terminalski dostop**
- **Client-Server oz. Odjemalec-strežnik**
- **Tri-nivojska arhitektura**

Terminalski dostop

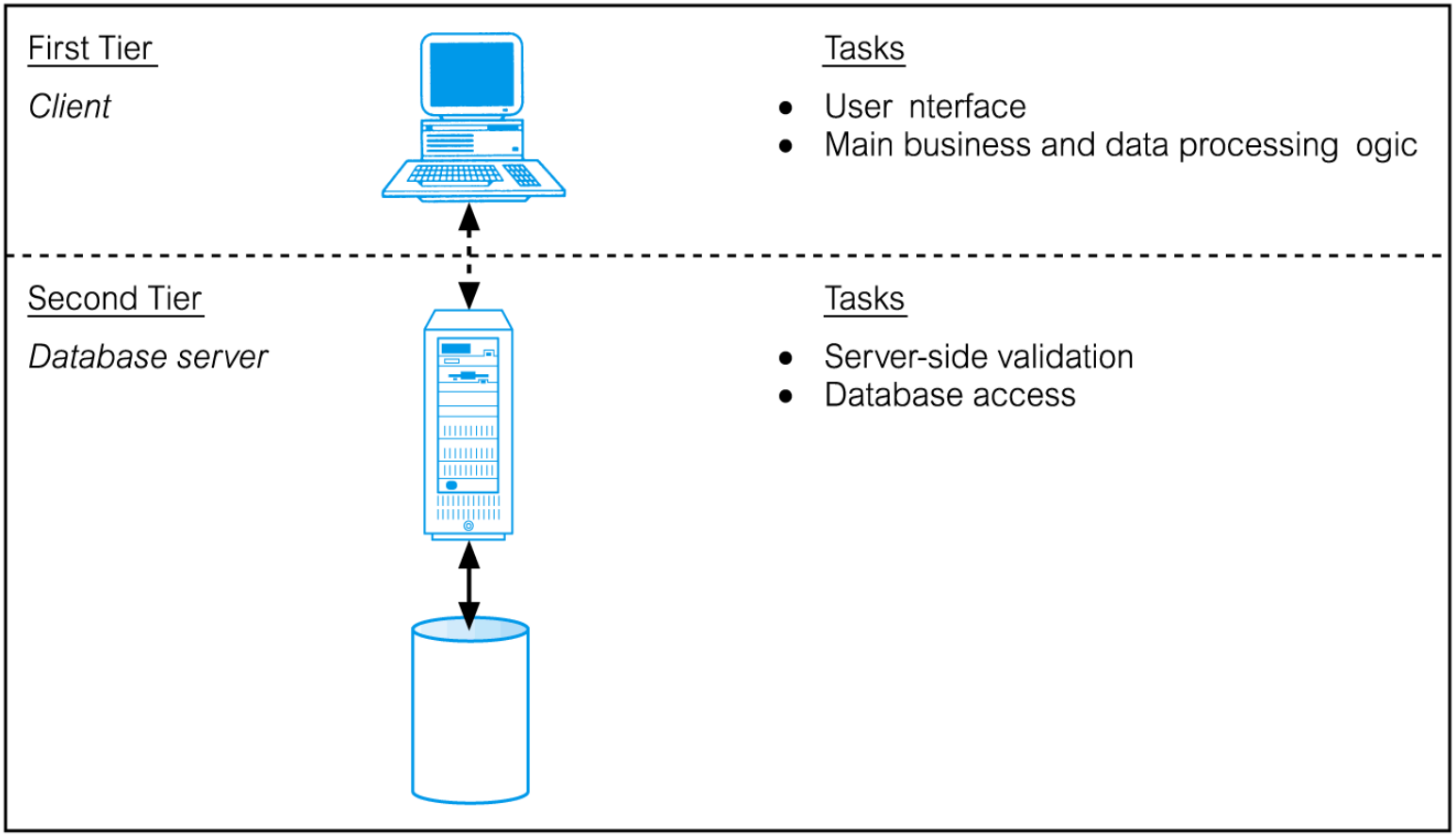
- **Mainframe, na katerega je priključenih več terminalov**



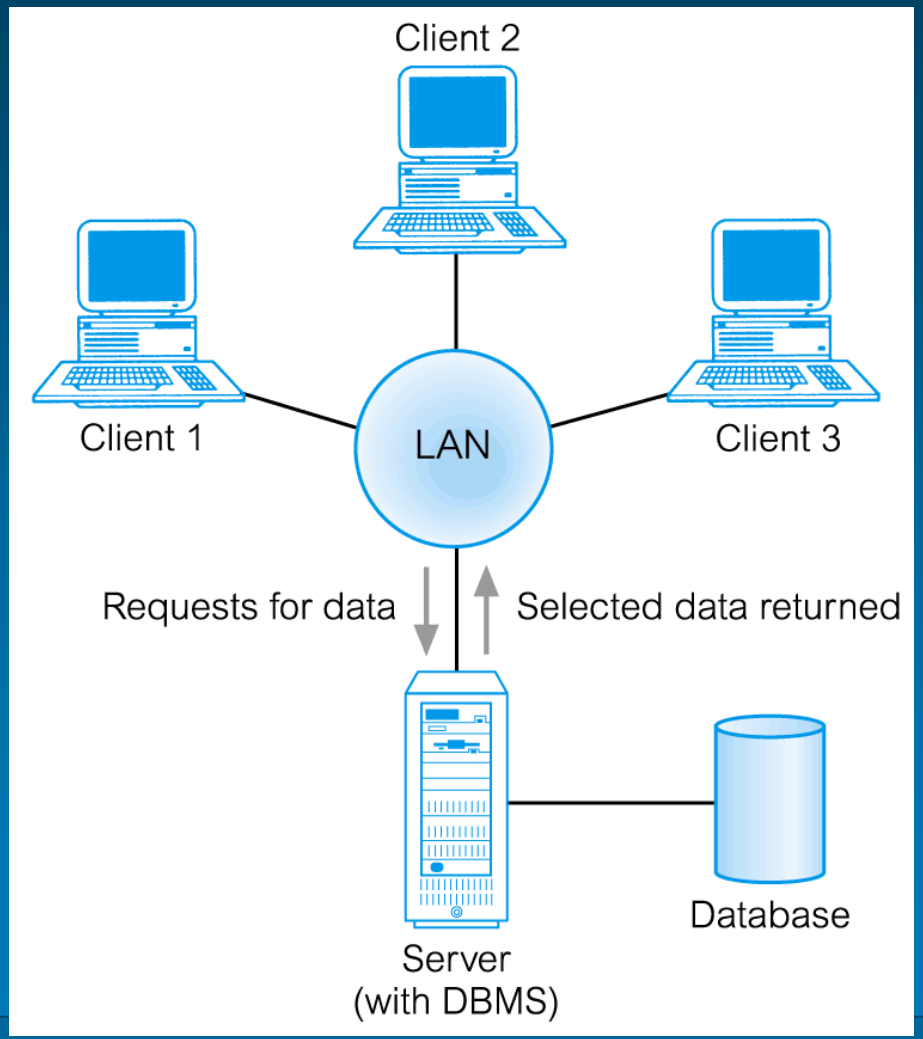
Client-server oz. Odjelamec-strežnik..

- **Odjemalec je zadolžen za izvajanje aplikacije: uporabniški vmesnik in poslovna logika**
- **Na strežniku se nahaja SUPB ter ena ali več (instanc) podatkovnih baz**

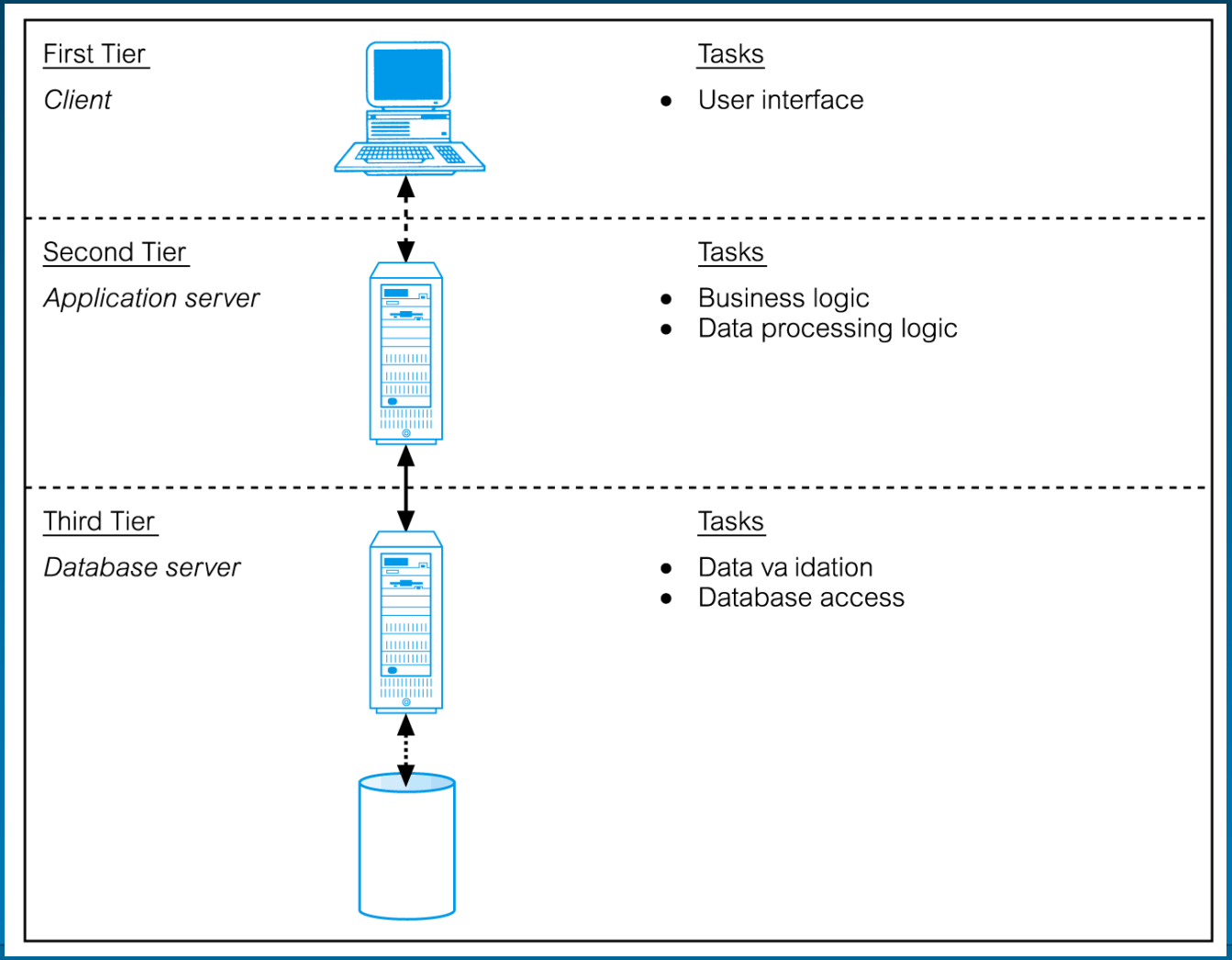
Client-server oz. Odjelamec-strežnik..



Client-server oz. Odjelamec-strežnik



Tri-nivojska arhitektura..



Tri-nivojska arhitektura..

- **Problemi arhitekture Odjemalec-strežnik:**
  - Zahteva relativno zmogljive odjemalce
  - Zahteva relativno velik napor za administracijo odjemalca

Tri-nivojska arhitektura

- **Prednosti tri-nivojske arhitekture:**
  - Zahteva manj zmogljive odjemalce
  - Centralizirana administracija aplikacije
  - Ustreza konceptu svetovnega spleta

Podatek in informacija..

- **Nekaj dejstev o podatku in informaciji**
  - **Podatek je predstavitev informacije na *formaliziran način*, ki je primeren za komunikacijo, interpretacijo ali obdelavo (s strani človeka ali stroja). Predstavimo ga lahko s pomočjo simbolov ali analognih veličin, ki ji je pripisan, ali se ji lahko pripiše nek pomen**
  - **Informacija je znanje, ki se nanaša na objekte, kot so dejstva, dogodki, stvari, procesi ali ideje, vključno s koncepti, ki imajo v okviru nekega konteksta določen pomen (ISO)**

Podatek in informacija..

- **Borje Langefors – *informacijska enačba***
  - **Informacija je *novo spoznanje*, ki ga človek doda svojemu poznavanju sveta. Odnos med informacijo, podatki, časom in interpretatorjevim znanjem predstavlja informacijska enačba:**

$$I = i(D, S, t)$$

I – informacija, ki jo posredujejo podatki
i – informacijska funkcija
D – podatki
S – prejemnikovo znanje
t – čas, ki je na voljo prejemniku za interpretacijo podatkov

Podatek in informacija

- **Langefors zaključuje, da:**
  - Podatki niso informacija
  - Podatki ne vsebujejo informacije
  - Podatki posredujejo informacijo prejemniku, katerega znanje je konsistentno z izbrano predstavitvijo podatkov in modelom sveta, na katerega se nanašajo
  - Informacija je pomen, ki ga prejemnik glede na svoje znanje v določenem času pripiše podatkom
  - Če je količina podatkov tako velika, da se jih v času, ki je na voljo za ukrepanje na njihovi osnovi, ne da interpretirati, se lahko zgodi, da s podatki ni posredovana nobena informacija

Poglavje 5

Prikaz in demonstracija ORACLE SUPB

Poglavje 6

Diski in datoteke

Povzeto po [1]

Uvod

- Podatki SUPB se hranijo na diskih (in trakovih)
- Pri upravljanju s podatki pa se uporablja tudi glavni pomnilnik

Hierarhija pomnilnika..

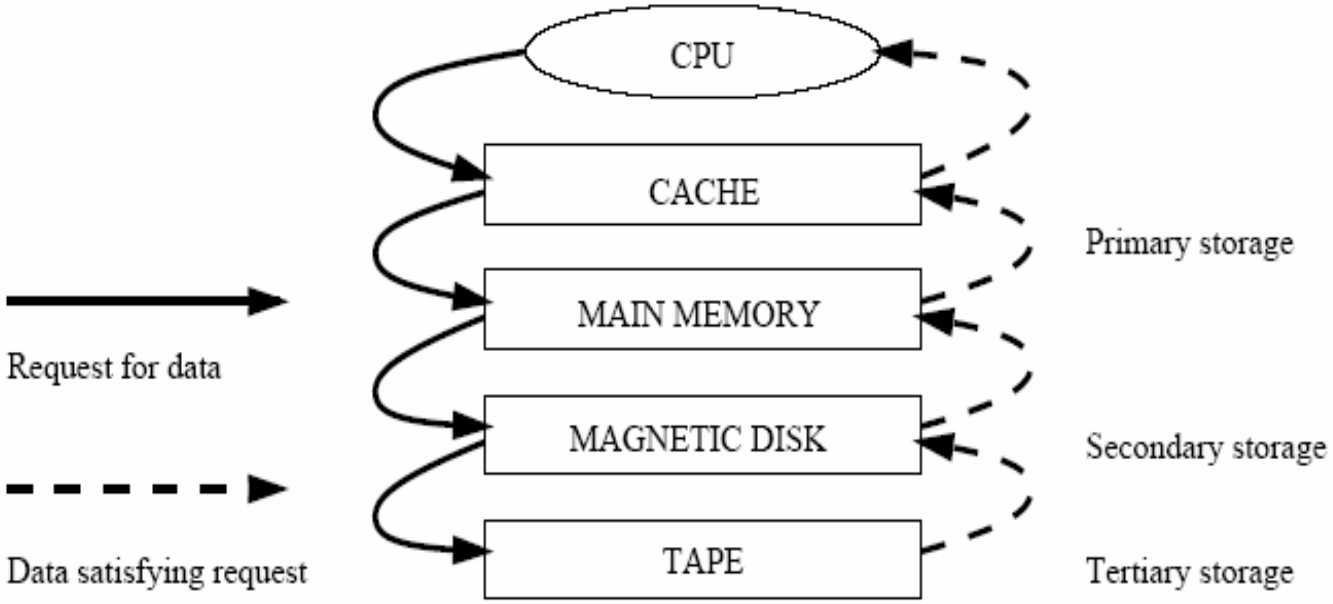


Figure 7.1 The Memory Hierarchy

Hierarhija pomnilnika..

- Dostop do primarnega pomnilnika je zelo hiter
- Stroški so za določeno količino glavnega pomnilnika so približno **100 krat večji** kot stroški za enako količino sekundarnega pomnilnika (disk)
- Počasnejše enote sekundarnega pomnilnika igrajo pomembno vlogo, saj je podatkov običajno zelo veliko
- Ker ne moremo vseh podatkov shraniti v glavni pomnilnik, jih shranjujemo na sekundarni pomnilnik

Hierarhija pomnilnika

- Obstajajo tudi **drugi razlogi** za shranjevanje podatkov na sekundarnem pomnilniku:
  - 32 bitni naslovni prostor omogoča naslavljanje samo 4Gb podatkov....
  - podatki morajo biti obstojni, v primarnem pomnilniku podatki običajno niso obstojni.
- Stanje glede kapacitete in cene pomnilnikov se spreminja iz dneva v dan, a okvirno razmerje 100:1 se ne spreminja

Magnetni disk..

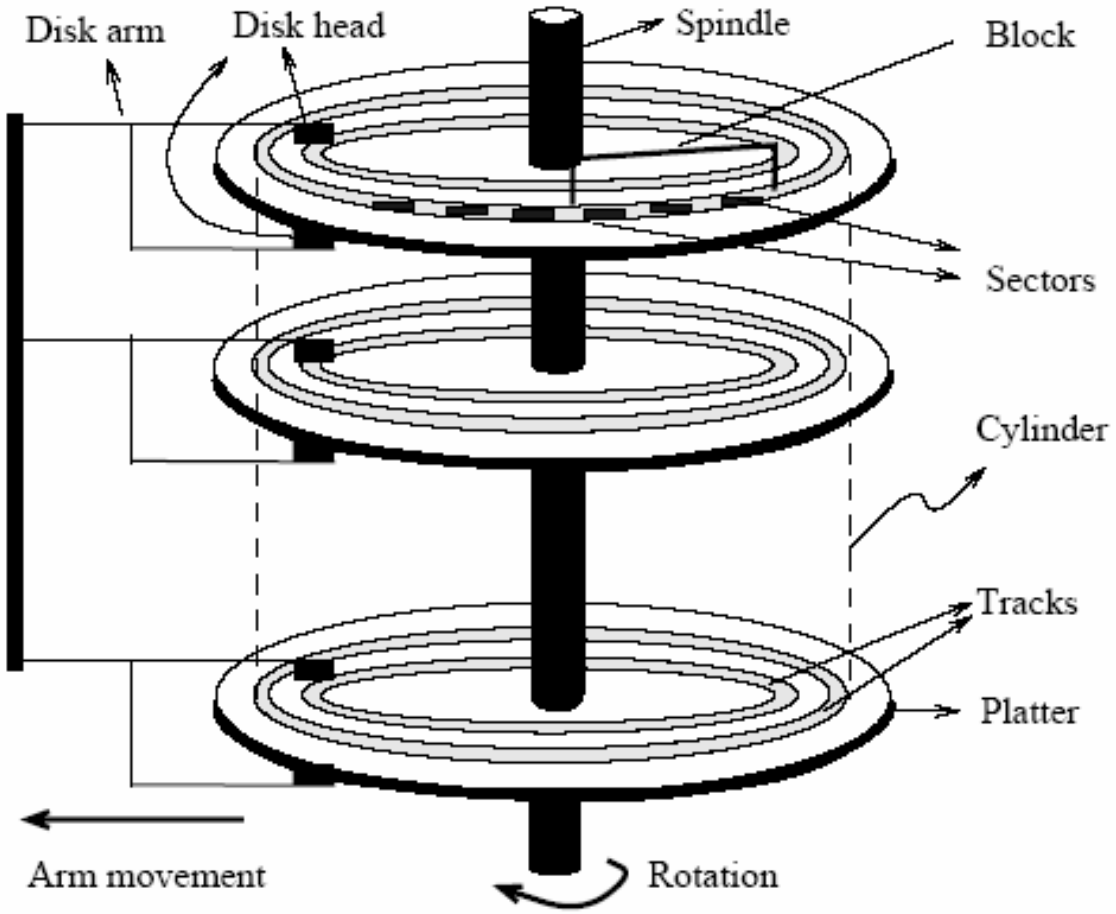


Figure 7.2 Structure of a Disk

Magnetni disk..

- Magnetni disk omogoča **neposreden dostop** do želene lokacije na njem (kaj pa magnetni trak?)
- SUPB omogoča **transparenten dostop** do podatkov, ki se nahajajo na disku
- Podatki na disku so shranjeni v enotah, ki se imenujejo BLOK-i
- BLOK predstavlja **zaporedje nizov** (byte) in je najmanjša enota, ki se jo lahko bere iz ali piše na disk

Magnetni disk..

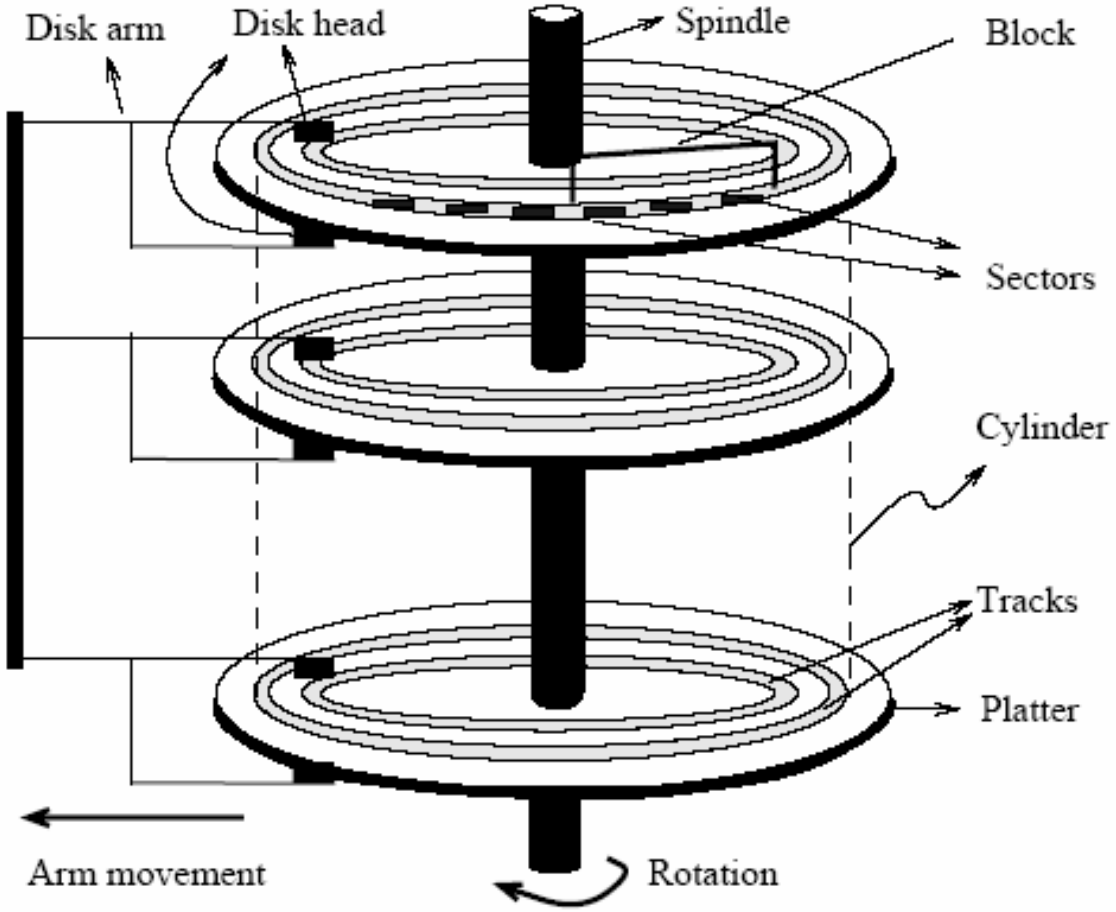


Figure 7.2 Structure of a Disk

Magnetni disk..

- BLOKI so organizirani v **koncentrične kroge** – SLEDI (track)
- SLEDI se nahajajo na eni ali obeh straneh magnetne plošče
- Množica vseh SLEDI, ki so **enako oddaljene** od središča, se imenuje CILINDER

Magnetni disk..

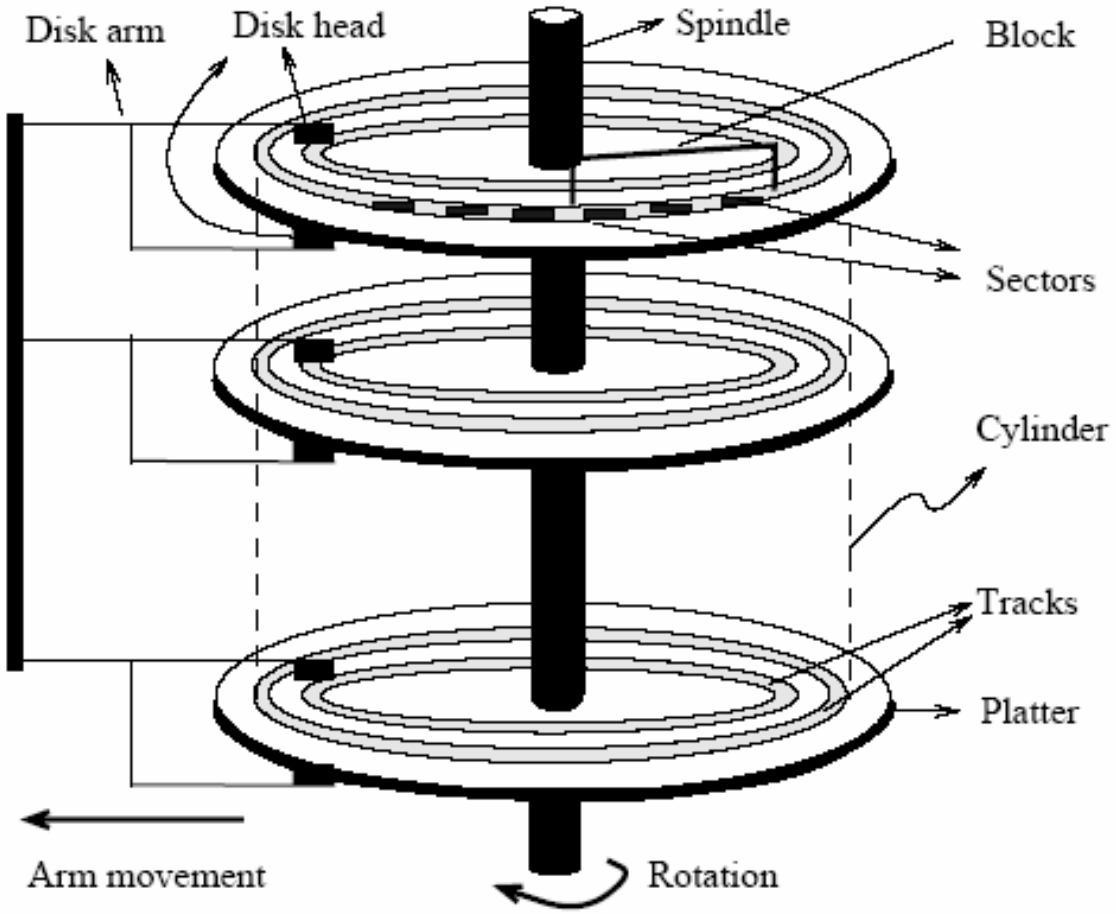


Figure 7.2 Structure of a Disk

Magnetni disk..

- Vsaka SLED **je razdeljena** na ODSEKE (sektorje) Velikost ODSEKA je določena z diskom in je ni mogoče spreminjati
- Velikost BLOKA se določi pri **formatiranju diska**. Njegova velikost je mnogokratnik velikosti ODSEKA
- V okviru diska je več glav, ki se premikajo hkrati. Za vsako ploščo ena glava (ali dve, če je vsebina na obeh straneh)

Magnetni disk..

- Disk je z računalnikom povezan preko krmilnika. Krmilnik **izvaja ukaze** za branje in pisanje na disk in zagotavlja pravilnost izvajanja teh ukazov

Magnetni disk..

- Čas, ki je **potreben za dostop** do določene lokacije na disku (**povprečni dostopni čas**) je sestavljen iz več komponent:
  - **iskalni čas** (premik glave na ustrezno sled)
  - **rotacijska zakasnitev** (čakalni čas, da se ustrezen blok na sledi zavrti do glave). Povprečno znaša polovico časa rotacije in je manjši od iskalnega časa.
  - **čas prenosa** (dejanski prenos bloka – branje ali pisanje).
- Primerjava: dostopni čas RAM-a (10ns) in diska (10ms).

Magnetni disk..

- Vpliv strukture diska na performanse:
  - podatki se morajo pred uporabo prenesti v glavni pomnilnik,
  - **najmanjša enota podatkov**, ki se bere ali piše na disk je blok. Če se potrebuje **samo en zapis** iz bloka, se prenese celotni blok.
  - čas za pisanje ali branje podatkov variira, odvisno od položaja podatkov na disku:
    - *dostopni čas=iskalni čas + rotacijska zakasnitev + čas prenosa*

Magnetni disk

- ZAKLJUČEK: Čas, ki ga SUPB porabi za obdelavo podatkov je močno odvisen od **lokacije podatkov na disku oz. razpršenosti podatkov po disku**. Čas, potreben za premikanje blokov iz diska oz. na disk je tako ponavadi večji kot čas potreben za obdelavo določenega podatka
- **Podatke je potrebno zato dobro razporediti po disku!!!.**

RAID – Uvod..

- Disk predstavlja **potencialno ozko grlo** za performanşe in vpliva na zanesljivost delovanja sistema
- Čeprav se performanse diskov zvišujejo, performanse CPU-jev rastejo veliko hitreje: perf. CPU 50% na leto, perf. diskov 10% na leto
- Diski vsebujejo mehanske elemente, zaradi česar je **verjetnost za napake** večja kot pri notranjem pomnilniku

RAID – Uvod..

- Če disk odpove, potem to v kontekstu podatkovnih baz pomeni **izgubo podatkov**
- REŠITEV: polje diskov (disk array), s katerim povečamo tako performanse kot zanesljivost delovanja

RAID – Uvod

- **Performanse se poveča** z razstavljanjem podatkov (**data striping**): podatke se distribuira po več diskih (občutek enega zelo hitrega diska)
- **Zanesljivost povečujemo** z redundandnostjo. Redundantne informacije so organizirane tako, da v primeru napake na disku omogočajo restavriranje podatkov na pokvarjenem disku
- Diskovna polja, ki implementirajo razstavljanje podatkov in/ali redundantnost se imenujejo "**Redundant Arrays of Independent Disks**" – RAID
- Poznamo več RAID organizacij, ki predstavljajo **različne kompromise** (trade-offs) med performansami in zanesljivostjo

RAID – Razstavljanje podatkov..

- RAID – Data striping se kaže uporabniku kot zelo **velik disk**
- Podatki se razdelijo na enake particije (**striping unit**), ki se distribuirajo **na več diskov**
- Enote se po diskih distribuirajo po "**round robin**" algoritmu. Dva logično sosedni particiji tako nista na istem disku
- Če polje vključuje D diskov, se particija i zapiše na "i mod D" disk

RAID – Razstavljanje podatkov

- Primer:

Če je “**striping unit**” = blok in če so I/O operacije dolge po več blokov, potem se zahteva procesa paralelno na več diskih in tako povečamo pasovno širino prenosa podatkov tolikokrat, kolikor diskov imamo v polju

RAID – Redundantnost..

- S tem ko se z več diski povečajo performanse sistema, pa se **zmanjša** celotna zanesljivost sistema
- Predpostavimo: MTTF (**mean-time-to-failure**) znaša 50.000 ur (5,7 let). Pri 100 diskih v polju znaša MTTF 50.000/100=500ur (21 dni)
- Zanesljivost diskovnega polja se lahko poveča z **redundantnimi podatki**

RAID – Redundantnost..

- Redundantnost lahko **neizmerno poveča** MTTF
- Določiti oz. odločiti se je potrebno:
  - **kje bodo shranjeni redundantni podatki**:
    - na manjšem številu kontrolnih diskov ali
    - bodo porazdeljeni po vseh diskih.
  - **kako izračunati redundantne podatke**:
    - večina diskovnih polj shranjuje informacijo o pariteti (paritetna shema uporablja dodaten – redundanten kontrolni disk za obnovo (recovery) po nesreči)

RAID – Redundantnost..

- Če dodamo prejšnjemu polju 100-ih diskov 10 diskov z **redundantnimi podatki**, naraste MTTF na več kot 250 let!!!
- Velik MTTF pomeni **manjšo verjetnost** za napako

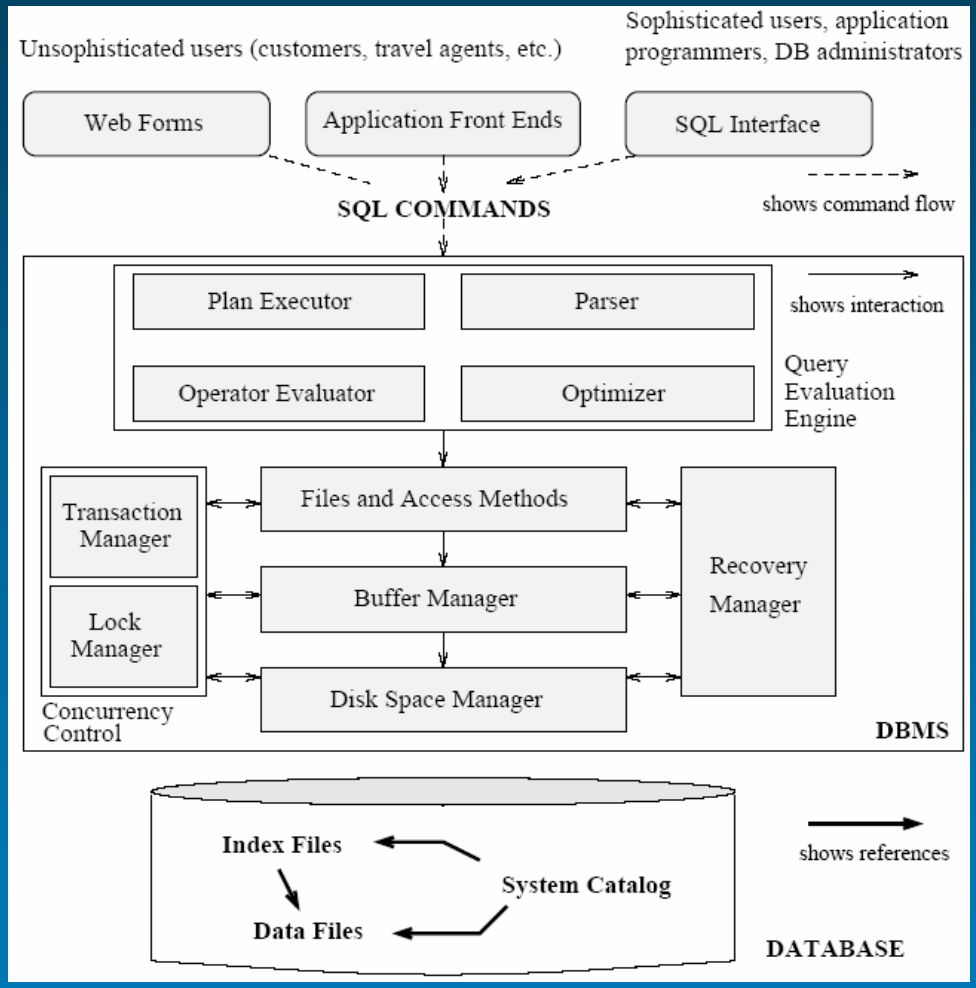
RAID – Stopnje redundance..

- Za primer predpostavimo, da imamo podatke, ki jih lahko spravimo na 4 diske. Od izbrane stopnje RAID, je odvisno dodatno število diskov (od 0 do 4).
- RAID 0: Nonredundant (**data striping**)
  - Uporablja data striping za povečanje pasovne širine
  - Ne vzdržuje nobene redundantne informacije
  - PROBLEM: MTTF pada linearno s številom diskov v polju
  - PREDNOSTI: najboljše performanse pisanja na disk, saj ni potrebno vzdrževati nobenih redundantnih podatkov
  - Efektivno uporabljeni prostor znaša vedno 100% prostora na disku. V našem primeru rabimo za svoje podatke 4 diske

RAID – Stopnje redundance

- RAID 1: Mirrored
  - Najdražja rešitev za polje diskov, ker se vzdržujeta dve kopiji podatkov na dveh diskih
  - Vsako pisanje bloka na disk vključuje pisanje na dva diska
  - Pisanje se ne izvede hkrati, ampak eno za drugo (zaradi primera nesreče med pisanjem)
  - Branje lahko vključuje paralelno branje dveh različnih blokov iz dveh diskov. Branje se lahko dodeli na disk, ki ima najmanjši dostopni čas
  - V našem primeru rabimo 4 diske + 4 diske za mirroring
  - efektivna uporaba prostora tako znaša 50%

Upravljanje prostora na disku..



Upravljanje prostora na disku..

- Za upravljanje z diskom skrbi **najnižji nivo** v SUPB arhitekturi – **Disk Space Manager**
- **Disk Space Manager** **podpira koncept** "strani" in podpira ukaze za dodeljevanje in sproščanje prostora na disku ter branje in pisanje strani

Upravljanje prostora na disku..

- Velikost strani je **enaka velikosti bloka** na disku, tako da se strani shranjujejo kot bloki na disku. Branje ali pisanje strani se tako lahko izvede v okviru ene I/O operacije
- Disk Space Manager **mora skriti podrobnosti** strojne opreme (in tudi operacijskega sistema) in omogočiti višjim plastem programske opreme (SUPB), da obravnava podatke kot zbirko strani

Upravljanje prostora na disku..

- Disk Space Manager **mora vzdrževati** stanje zasedenih in prostih blokov na disku (2 načina):
  - **seznam prostih blokov** (kazalec na prvi blok seznama se shrani na znano lokacijo na disku),
  - **vzdrževanje bitne mape** (za vsak blok je v bitni mapi bit, ki označuje, ali je blok zaseden ali ne)

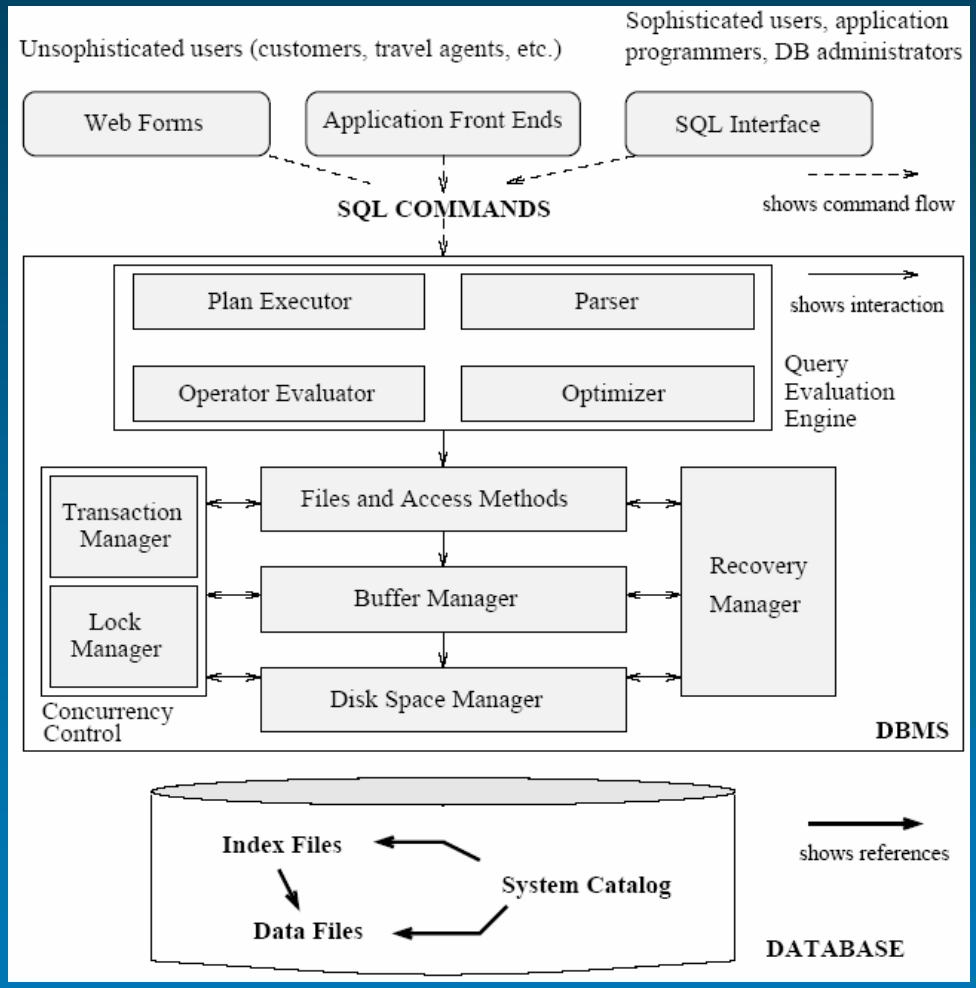
Upravljanje prostora na disku..

- Uporaba datotečnega sistema za upravljanje s prostorom:
  - Disk Space Manager lahko uporablja datoteke operacijskega sistema. Posledično se celotna PB se nahaja v eni ali več datotekah
  - V tem primeru je zadolžen za upravljanje prostora v teh datotekah

Upravljanje prostora na disku

- Nekateri SUPB **ne uporabljajo** datotečnega sistema, ampak svoj lastni sistem za upravljanje z diskom (ali pa obe možnosti):
  - Praktični razlog za to je, da bazo lahko uporabimo na več platformah (proizvajalec lažje portira SUPB na različne OS)
  - Tehnični razlog pa, da se pri 32 bitnem naslavljanju pojavi omejitev v velikosti datoteke

Buffer Manager..



Buffer Manager..

- Za razumevanje vloge in pomena Buffer managerja je nazoren naslednji PRIMER:
  - Kaj če hočemo izvesti poizvedbo nad PB, ki ima 1.000.000 strani, v pomnilnik pa jih lahko spravimo le 1.000?
- Buffer Manager je programska plast, ki skrbi za **prenašanje ustreznih strani** v pomnilnik
- Buffer Manager **upravlja z razpoložljivim pomnilnikom** (buffer pool)

Buffer Manager..

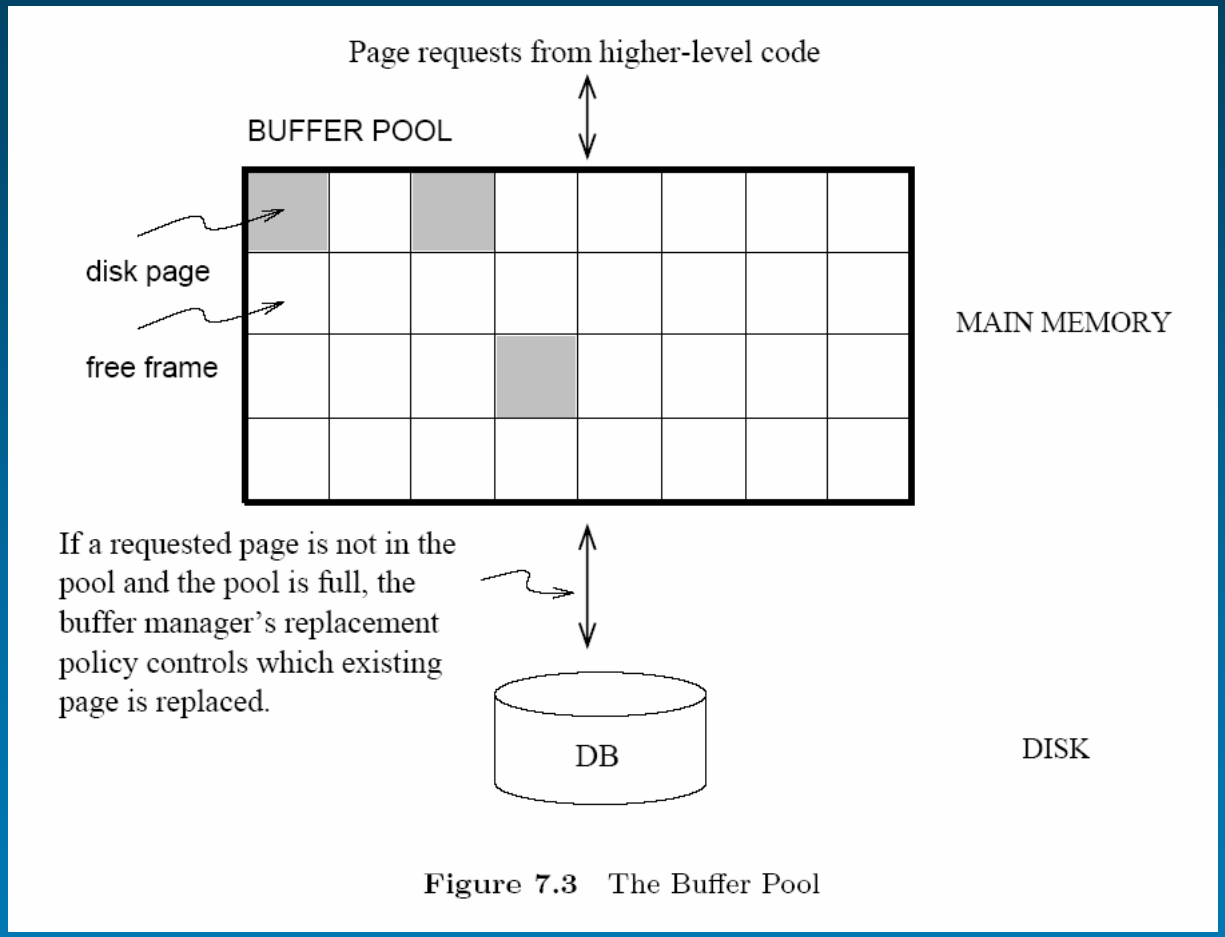


Figure 7.3 The Buffer Pool

Buffer Manager..

- Buffer Manager **zagotavlja** višjim plastem SUPB-ja strani, ki jih te rabijo za svoje delo
- Buffer Manager prenese v buffer pool tisto stran, katero je višja plast zahtevala za delo
- SUPB **mora obvestiti** Buffer Manager o tem, da je "sprostila" stran, ki je ne rabi več, ali pa da je stran ažurirala. Buffer manager je odgovoren za prenos na disk
- Pri odločanju o tem katere strani se bodo v pomnilniku zamenjale, se uporablja določena strategija (**replacement strategy**)

Buffer Manager..

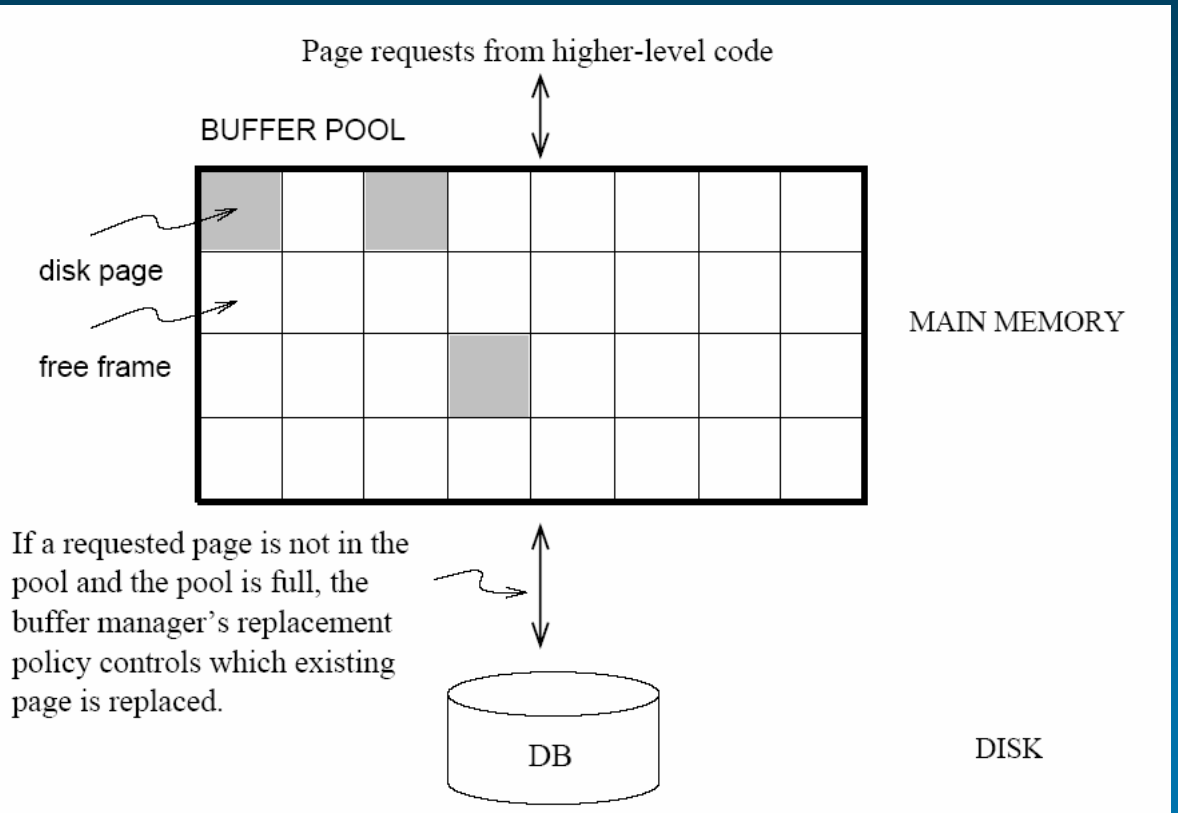


Figure 7.3 The Buffer Pool

Buffer Manager..

- Za vsak okvir v buffer pool-u se hrani 2 spremenljivki:
  - **pin_count**: kolikokrat je bila stran v okvirju zahtevana (+1) in kolikokrat sproščena (-1): število trenutnih uporabnikov strani
  - **dirty**: boolean vrednost, ki označuje, ali je bila stran spremenjena ali ne
- Na začetku je za vsak okvir pin_count=0 in dirty=off

Buffer Manager..

- Ko se pojavi zahteva po **določeni strani**, Buffer Manager izvede naslednje:
  - če se stran nahaja v kakšnem od okvirjev, vrne pomnilniški naslov okvirja in poveča pin_count za 1,
  - drugače pa izvede naslednje:
    - *izbere okvir za zamenjavo (z uporabo strategije za zamenjavo) če je dirty bit okvirja, ki bo zamenjan "on", se stran prepiše na disk*
    - *stran se prenese iz diska v okvir, ki je določen za zamenjavo in poveča pin_count*

Buffer Manager..

- Če se zahtevana stran **ne nahaja** v buffer pool-u, in če so vsi okvirji zasedeni, se za zamenjavo izbere okvir, katerega pin_count=0. V primeru več takih okvirjev se izmed njih izbere okvir po določeni uporabljeni strategiji
- Če je dirty bit postavljen, potem je potrebno stran ob zamenjavi **zapisati nazaj na disk**, drugače pa se njena vsebina lahko prepiše z vsebino nove strani

Buffer Manager..

- Če v buffer pool-u ni nobene strani katera bi imela pin_count=0 in hkrati iskane strani ni v buffer pool-u, potem mora Buffer Manager počakati, da se sprosti zaseženje katere od strani in potem se lahko stran zamenja:
  - V praksi to pomeni, da je transakcija, ki zahteva tako stran lahko enostavno razveljavljena.

Buffer Manager

- Strategija zamenjevanja strani v buffer pool (buffer replacement policy). Najbolj uporabljena strategija je LRU – least recently used
- Buffer manager vodi seznam kazalcev na okvirje, ki imajo pin_count enak 0. Nov sproščeni okvir (sprosti se stran, ki je v okvirju) doda na seznam na koncu

Buffer Manager : Predpomnilnik OS..

- Rečemo lahko, da obstaja podobnost med navideznim pomnilnikom operacijskega sistema in upravljanjem s pomnilnikom pri SUPB
- Cilj obeh je zagotoviti dostop do več podatkov, kot pa jih lahko spravimo v pomnilnik. Ideja je, da se strani iz diska prenašajo v pomnilnik po potrebi in pri tem nadomeščajo strani, ki se jih v pomnilniku ne rabi več

Buffer Manager : Predpomnilnik OS..

- Zakaj ne uporabimo navideznega pomnilnika OS za potrebe SUPB?
  1. SUPB lahko bolj natančno predvidi zaporedje, v katerem se bo dostopalo do strani, kot pa tipičen OS
  2. SUPB rabi več nadzora nad stranmi, ki se zapisujejo na disk, kot pa ga omogoča tipičen OS

Buffer Manager : Predpomnilnik OS

- **Še bolj pomembno**: buffer manager uporablja strategijo “prefetching of pages”, ki omogoča predvidevanje več naslednjih zahtev in v skladu s tem se ustrezne strani prenesejo v pomnilnik, preden so dejansko zahtevane

Poglavje 7

Relacijski model

Povzeto po [2]

O relacijskem podatkovnem modelu

- Pojavi se leta 1970, predlaga ga Edgar Codd.
- Pomeni revolucijo, nadomesti starejše modele.
- Model PB, ki temelji na relacijskem modelu, je predstavljena z množico relacij, kjer je vsaka relacija tabela z vrsticami in stolpci.
- Je zelo enostaven za razumevanje:
  - Tudi neizkušeni uporabniki lahko razumejo vsebino podatkovne baze;
  - Na voljo so enostavni vendar močni jeziki za poizvedovanje po vsebini PB.

Terminologija pri relacijskem modelu..

- Pri relacijskem modelu uporabljamo določeno terminologijo:
  - Relacija
  - Atribut
  - Domena
  - n-terica
  - Stopnja relacija
  - Številnost relacije
  - Relacijska shema
  - Relacijska PB

Terminologija pri relacijskem modelu..

- Relacijo si lahko predstavljamo kot dvodimenzionalno tabelo s stolpci in vrsticami.
  - Velja za logično strukturo podatkovne baze in ne za fizično.
  - Vrstica relacije predstavlja objekt, osebo, dogodek, pravilo – ima nek pomen

Ime	Starost (v letih)	Teža (v kg)
Tine	15	50
Meta	20	45
Jure	40	80
Ana	5	10

Terminologija pri relacijskem modelu..

- Atribut je poimenovani stolpec relacije.
  - Predstavlja lastnost tega (objekt, osebo, dogodek, pravilo), kar predstavlja relacijo
  - Atributu določimo podatkovni tip in dolžino

Ime	Starost (v letih)	Teža (v kg)
Tine	15	50
Meta	20	45
Jure	40	80
Ana	5	10

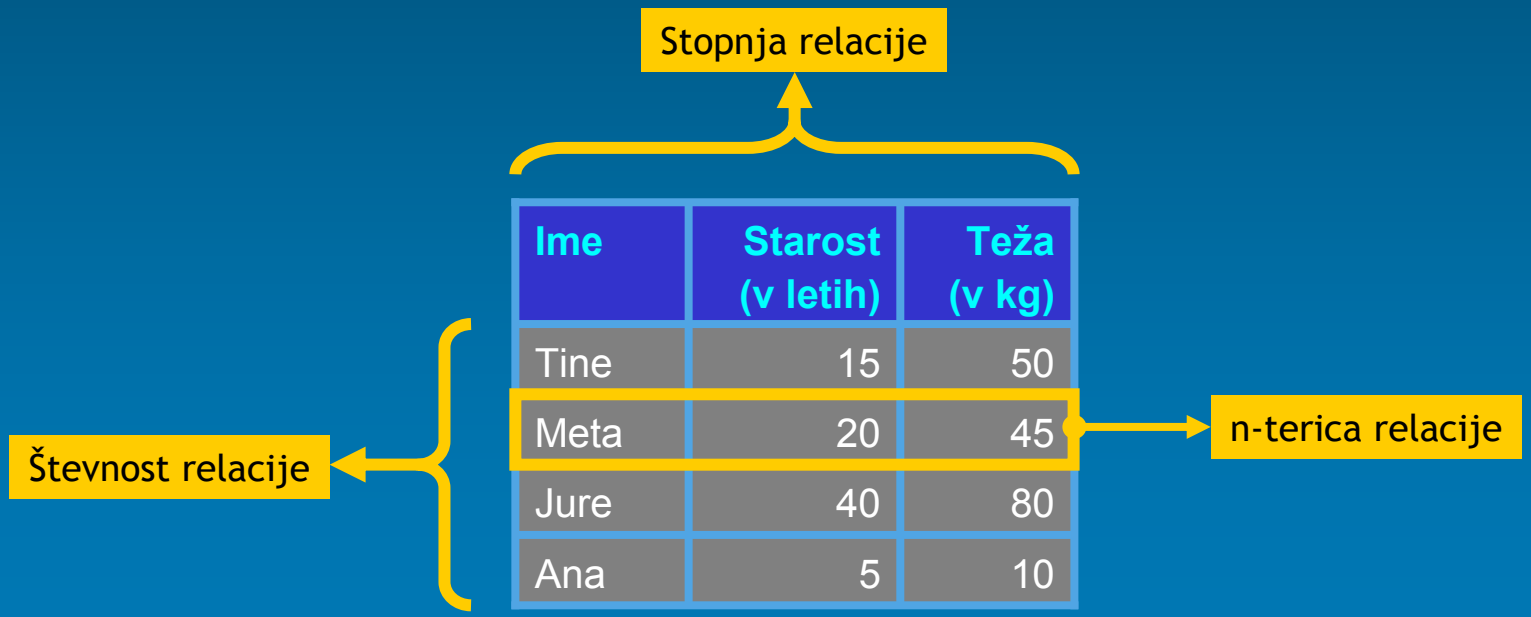
Terminologija pri relacijskem modelu..

- Domena določa poleg podatkovnega tipa in dolžine tudi množico dovoljenih vrednosti enega ali več atributov, ki so vključeni v to domeno.
- Primeri domen:

Attribute	Domain Name	Meaning	Domain Definition
branchNo	BranchNumbers	The set of all possible branch numbers	character: size 4, range B001–B999
street	StreetNames	The set of all street names in Britain	character: size 25
city	CityNames	The set of all city names in Britain	character: size 15
postcode	Postcodes	The set of all postcodes in Britain	character: size 8
sex	Sex	The sex of a person	character: size 1, value M or F
DOB	DatesOfBirth	Possible values of staff birth dates	date, range from 1-Jan-20, format dd-mmm-yy
salary	Salaries	Possible values of staff salaries	monetary: 7 digits, range 6000.00–40000.00

Terminologija pri relacijskem modelu..

- N-terica je ena vrstica v relaciji.
- Številnost relacije je število n-teric relacije.
- Stopnja relacije je število atributov v relaciji.



Terminologija pri relacijskem modelu

- Relacijska podatkovna baza je množica normaliziranih relacij z enoličnimi imeni.
- Kaj so normalizirane relacije se bomo učili v nadaljevanju.

Matematična definicija relacije..

- Vzemimo dve množici D1 in D2, kjer D1 = {2, 4} in D2 = {1, 3, 5}.
- Kartezijski produkt D1 x D2 je množica vseh urejenih parov, kjer prvi element pripada množici D1, drugi množici D2: D1 x D2 = {(2, 1), (2, 3), (2, 5), (4, 1), (4, 3), (4, 5)}

Matematična definicija relacije..

- Vsaka podmnožica kartezijskega produkta D1 x D2 je relacija: $R \subset (D1 \times D2)$.
- Primer:

  $R = \{(2, 1), (4, 1)\}$
- Kateri pari so v relaciji lahko določimo s pogoji: na primer vsi pari, kjer je drugi element enak 1:

  $R = \{(x, y) \mid x \in D1, y \in D2 \text{ in } y = 1\}$
- ali kjer je prvi element dvakrat večji od drugega:

  $S = \{(x, y) \mid x \in D1, y \in D2 \text{ in } x = 2y\}$

Matematična definicija relacije..

- Vzemimo tri množice D1, D2, D3 s kartezijskim produktom D1 x D2 x D3; Na primer:

  D1 = {1, 3} D2 = {2, 4} D3 = {5, 6}

  D1 x D2 x D3 = {(1,2,5), (1,2,6), (1,4,5), (1,4,6), (3,2,5), (3,2,6), (3,4,5), (3,4,6)}

- Vsaka podmnožica teh urejenih trojic je relacija!

Matematična definicija relacije..

- Kartezijski produkt n množic (D1, D2, ..., Dn) je:

  D1 x D2 x . . . x Dn = {(d1, d2, . . . , dn) | d1 ∈D1, d2 ∈ D2, . . . , dn ∈ Dn}

- Navadno pišemo kot $\prod_{i=1}^{n} D_i$

- Vsaka podmnožica n-teric iz tega kartezijskega produkta je relacija teh n množic.

Relacijska shema..

- Vsaki relaciji pripada relacijska shema.
- Relacijsko shemo sestavlja oznaka sheme R ter lista oznak atributov Ai s pripadajočimi oznakami domen Di:

  R (A1: D1, A2: D2, ..., An: Dn)

- Relacijska shema predstavlja semantiko ali pomen relacije.

Relacijska shema..

- Relacijske sheme so del konceptualnih shem. Razlagajo pomen relacij.
- Glede na skromno izrazno možnost nudijo informacijo le poznavalcem podatkovne baze, ki znajo relacijske sheme pravilno interpretirati.
- Vsaki relaciji pripada natanko ena relacijska shema; neka relacijska shema pa lahko pojasnjuje tudi več relacij:

Sh: {r} → {R} Sh(r) = R

Relacijska shema

Ime	Starost (v letih)	Teža (v kg)
Tine	15	50
Meta	20	45
Jure	40	80
Ana	5	10

Relacijska shema

Relacija, predstavljena kot tabela

Sh(r) = Oseba(Ime: I, Starost: C, Teža: C)

Relacijska shema

Domena, ki obsega imena: I ≡ {Tine, Meta, Jure, Ana}
Domena, ki obsega interval celih števil: C ≡ 1, 2,... 200

Domene atributov relacije

Lastnosti relacij..

- Ime relacije je enolično. V logični enoti (odvisno od SUPB, praviloma gre za shemo) podatkovne baze ni dveh relacij z enakim imenom
- Vsaka celica tabele (polje), ki predstavlja relacijo, vsebuje natančno eno atomarno vrednost
- Vsak atribut relacije ima enolično ime. V isti relaciji ni dveh atributov, ki bi imela isto ime
- Vrednosti nekega atributa so vse iz iste domene

Lastnosti relacij

- Vsaka n-terica relacije je enolična → v relaciji ni dveh enakih n-teric.
- Vrstni red atributov v relaciji je nepomemben.
- Vrstni red n-teric v relaciji je nepomemben.

Primeri..

Ime	Starost (v letih), teža (v kg)
Tine	S15_T50
Meta	S20_T45
Jure	S40_T80
Ana	S5_T10

Celice ne vsebujejo atomarnih vrednosti

Zakonci	Leto poroke (celo število)
Tine, Meta	1995
Ana, Jure	1980

Celice vsebujejo več vrednosti

Primeri

- Dvojni pomen posameznih atributov ni priporočljiv:

Tip	Ime/Priimek
1	Jana
2	Novak
1	Tina
1	Kaja

Če je tip=‘1’ potem ime, sicer priimek

Funkcionalne odvisnosti..

- Relacija je model nekega stanja v poslovni domeni → njena vsebina ne more biti poljubna.
- Realne omejitve ne omogočajo, da bi bili odnosi v svetu kakršnikoli; možna so le določena stanja.
- Odvisnosti so sredstvo, s katerim lahko v relacijskem modelu povemo, katere vrednosti relacij so veljavne in katere sploh ne morejo obstajati.

Funkcionalne odvisnosti..

- Poznamo več vrst odvisnosti:
  - Funkcionalne odvisnosti (functional dependency)
  - Večvrednostne odvisnosti (multivalued dependency)
  - Stične odvisnosti (join dependency)
- Obravnavali bomo funkcionalne odvisnosti

Funkcionalne odvisnosti..

- Predpostavimo, da obstaja relacijska shema R z množico atributov, katere podmnožici sta X in Y.

- V relacijski shemi R velja X → Y (X funkcionalno določa Y oziroma Y je funkcionalno odvisen od X), če v nobeni relaciji, ki pripada shemi R, ne obstajata dve n-terici, ki bi se ujemali v vrednostih atributov X in se ne bi ujemali v vrednostih atributov Y.

Funkcionalne odvisnosti

- Množico funkcionalnih odvisnosti, ki veljajo med atributi funkcionalne sheme R in v vseh njenih relacijah, označimo s F

$X \rightarrow Y \in F \Leftrightarrow \forall\, r\, (\, Sh(r) = R \Rightarrow \forall\, t, \forall\, u\, (t \in r \text{ in } u \in r \text{ in } t.X = u.X \Rightarrow t.Y = u.Y\,)$

kjer

t.X, u.X, t.Y in u.Y označujejo vrednosti atributov X oziroma Y v n-tericah t oziroma u.

Primeri funkcionalnih odvisnosti

- Imamo relacijo s shemo

  Izpit(VpŠt, Priimek, Ime, ŠifraPredmeta, Datum izpita, OcenaPisno, OcenaUstno)

- z naslednjim pomenom:

  Študent z vpisno številko VpŠt ter priimkom Priimek in imenom Ime je na DatumIzpita opravljal izpit iz predmeta s šifro ŠifraPredmeta. Dobil je oceno OcenaPisno in OcenaUstno.

- Funkcionalne odvisnosti relacijske sheme Izpit so:

  F ≡ { VpŠt → (Priimek, Ime), (VpŠt, ŠifraPredmeta, DatumIzpita) → (OcenaPisno, OcenaUstno) }

Ključi relacije..

- Ker je relacija množica n-teric, so v njej vse n-terice ločene med seboj (ne pozabimo: v relaciji ni dveh enakih n-teric).
- Za sklicevanje na posamezno n-terico ni potrebno poznati vseh vrednosti atributov n-terice, če v shemi nastopajo funkcionalne odvisnosti.
- Množici atributov, ki določajo vsako n-terico, pravimo ključ relacije oziroma ključ relacijske sheme.

Ključi relacije..

- Predpostavimo, da obstaja relacijska shema z atributi A_1 A_2 ... A_n katere podmnožica je množica atributov X.

- Atributi X so ključ relacijske sheme oziroma pripadajočih relacij, če sta izpolnjena naslednja dva pogoja:

(1) $X \rightarrow A_1 A_2 \ldots A_n$

(2) ne obstaja X', ki bi bila prava podmnožica od X in ki bi tudi funkcionalno določala $A_1 A_2 \ldots A_n$

Ključi relacije..

- Poznamo več vrst ključev:
  - Kandidat za ključ (a key candidate)
  - Primarni ključ (primary key)
  - Tuji ključ (foreign key)

- Kandidat za ključ je vsaka podmnožica atributov relacije, ki relacijo enolično določa.

Ključi relacije

- Primarni ključ je tisti kandidat za ključ, ki ga (med vsemi kandidati za ključi) izberemo za shranjevanje relacij v fizični podatkovni bazi.
- Tuji ključ je množica atributov, v okviru ene relacije, ki je enaka primarnemu ključu neke druge ali iste relacije.

Primeri ključev

ARTIKEL

Šifra	Naziv	Zaloga
A10	Telovadni copati Nike	10
A12	Trenerka Bali	4
BC80	Moška jakna QuickSilver	1
X12	Ženska jakna QuickSilver	0

Primarni ključ v tabeli Artikel

POSTAVKA

Račun	Šifra artikla	Količina
15/05	A10	1
15/05	X12	1

Primarni ključ v tabeli Postavka

Tuji ključ v tabeli Postavka → kaže na primarni ključ v tabeli Artikel

Omejitve nad podatki..

- Podatkovni model v širšem smislu je skupek konceptov za opisovanje strukture podatkov, mehanizmov za obdelavo podatkov, povezav med njimi in omejitev nad podatki.
- Sestoji se iz treh delov:
  - Komponenta za strukturo podatkov: zajema pravila, po katerih je možno kreirati podatkovno bazo;
  - Komponenta za obdelavo: definira tipe operacij, ki so dovoljene nad podatki → vključuje operacije za ažuriranje in iskanje podatkov v bazi ter za spreminjanje strukture;
  - Komponenta za omejitve: množica pravil oziroma omejitev, ki skrbijo za celovitost podatkov.

Omejitve nad podatki

- Poznamo več vrst omejitev:
  - Omejitve domene (Domain constraints)
  - Pravila oz. omejitve za celovitost podatkov (Integrity constraints):
    - Omejitve entitet (Entity Integrity)
    - Omejitve povezav (Referential Integrity)
  - Splošne omejitve (General constraints)

Oznaka “Null”

- Oznaka “Null”:
  - Predstavlja vrednost atributa, ki je trenutno neznana ali irelevantna za n-terico.
  - Gre za nepopolne podatke ali podatke pri izjemnih primerih.
  - Predstavlja odsotnost podatka. Ni enako kot 0 ali prazen znak, kar je dejansko vrednost.

- Oznaka “Null” je problematična pri implementaciji, saj je relacijski model osnovan na predikatnem računu prvega reda (Boolean logic), kjer sta edini možni vrednosti true in false.

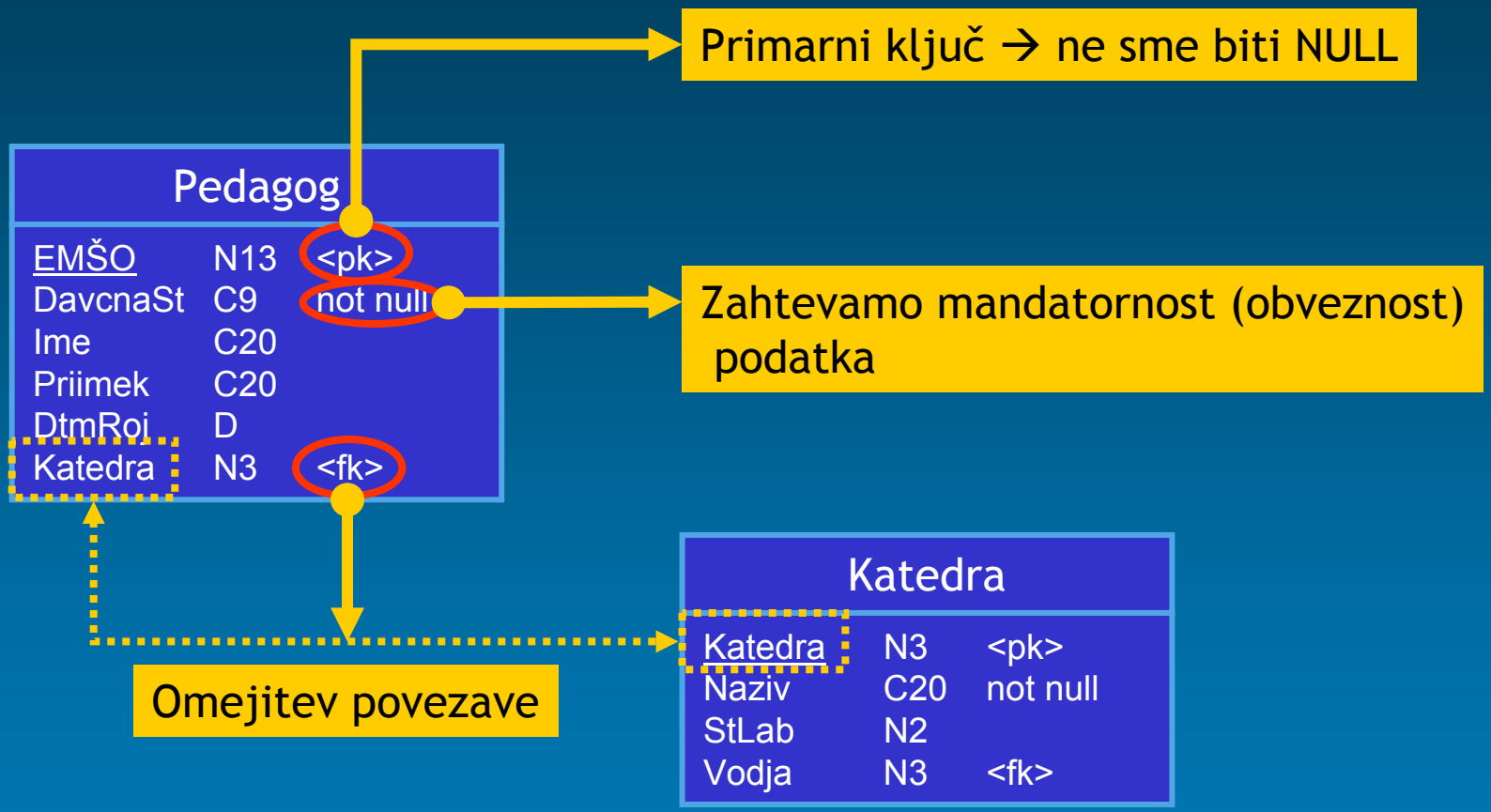
Omejitve entitet in povezav

- Omejitev entitete
  - V osnovni relaciji ne sme biti noben atribut, ki je del ključa, enak Null.
  - Primarni ključ kot integritetna omejitev
  - Ostale omejitve vezane na kombinacijo vrednosti posameznih atributov v n-terici
- Omejitve povezav
  - Če v relaciji obstajajo tuji ključi, potem morajo:
    - (a) njihove vrednosti ustrezati tistim, ki so v obliki ključa zapisane v eni izmed n-teric neke druge ali iste relacije
    - (b) ali pa mora biti tuji ključ v celoti enak Null.
  - Tuji ključ kot integritetna omejitev

Splošne omejitve

- Splošne omejitve
  - Dodatna pravila, ki jih določi uporabnik ali skrbnik podatkovne baze, ki definirajo ali omejujejo nek vidik poslovne domene, za katero je narejena podatkovna baza.

Primeri omejitev



Pogledi..

- Osnovna relacija (base relation)
  - Poimenovana relacija, ki ustreza nekemu entitetnemu tipu v konceptualnem modelu, katere n-terice so fizično shranjene v podatkovni bazi.
  - Predstavlja konceptualno shemo
- Pogled (view)
  - Rezultat ene ali več operacij nad osnovnimi relacijami z namenom pridobitve nove relacije.
  - Predstavlja zunanjo shemo

Pogledi

- Pogled je navidezna relacija, ki ne obstaja v relacijski bazi, temveč se dinamično kreira takrat, ko nekdo po njej povprašuje.
- Vsebina pogleda je definirana kot poizvedba nad eno ali več osnovnimi relacijami.
- Pogledi so dinamični → spremembe nad osnovnimi relacijami, katerih atributi so zajeti tudi v pogledu, so v pogledu takoj vidne.

Primer pogleda

ARTIKEL

Šifra	Naziv	Zaloga
A10	Telovadni copati Nike	10
A12	Trenerka Bali	4
BC80	Moška jakna QuickSilver	1
X12	Ženska jakna QuickSilver	0

RAČUN

Račun	Šifra artikla	Količina
15/05	A10	1
15/05	X12	1
16/05	A10	3
17/05	A10	1

```
SELECT A.sifra, A.naziv, sum(R.kolicina) AS Prodanih
FROM artikel A, racun R
WHERE A.sifra = R.sifra GROUP BY A.sifra, A.naziv
```

Šifra	Naziv	Prodanih
A10	Telovadni copati Nike	5
X12	Ženska jakna QuickSilver	1
...		

Namen uporabe pogledov

- Predstavljajo odličen mehanizem za zagotavljanje varnosti → skrivajo posamezne dele konceptualne sheme pred določenimi uporabniki.
- Uporabnikom dajejo možnost, da do podatkov dostopajo na prilagojen način → isti podatki so lahko s strani različnih uporabnikov v istem času vidni na različne načine.
- Poenostavljajo kompleksne operacije nad osnovnimi relacijami.
- Omogočajo zunanjo shemo (nivoji abstrakcije)

Spreminjanje vsebine pogledov..

- Vse spremembe nad osnovnimi relacijami morajo biti takoj vidne tudi v pogledih nad temi relacijami.

- Če spremenimo podatke v pogledu, se morajo spremembe poznati tudi v osnovnih relacijah, na katere se te spremembe nanašajo.

Spreminjanje vsebine pogledov..

- V pogledih niso možne vse spremembe. Veljajo naslednje omejitve:
  - Nad pogledom so možne spremembe, če pogled zajema eno samo osnovno relacijo ter vključuje atribute, ki so kandidat za ključ relacije.
  - Če pogled zajema več relacij, spremembe niso možne (izjeme).
  - Če je pogled pridobljen z agregacijo ali grupiranjem n-teric, spremembe niso možne.

Poglavje 8

Organizacija datotek in indeksi

Povzeto po [1]

Uvod..

- Potrebno je določiti **stroškovni model**, ki opredeljuje stroške dostopanja do podatkov
- Analizirali bomo tri osnovne datotečne organizacije:
  - neurejeno datoteko,
  - urejeno datoteko,
  - razpršeno datoteko.

Uvod

- Namen analize je pravilna izbira datotečne organizacije, glede na potrebe.
- Rešitev za hitrejši dostop do zapisov je indeksiranje

Stroškovni model..

- Stroškovni model omogoča oceniti stroške (v smislu izvajalnega časa) I/O operacij nad stranmi v PB
- Za analizo bomo uporabljali naslednje parametre:
  - **B**: število podatkovnih strani,
  - **R**: število zapisov na stran,
  - **D**: (povprečen) čas, potreben za branje ali pisanje strani,
  - **C**: čas za procesiranje zapisa,
  - **H**: čas, ki je potreben za uporabo hash funkcije nad zapisom (pri razpršenih datotekah).
- Tipične vrednosti znašajo: D=15 ms, C in H=100 ns

Stroškovni model..

- Poenostavitve:
  - Ker so I/O operacije časovno obsežnejše, stroške procesiranja lahko zanemarimo (C = H = 0),
  - natančen model bi bil preveč zapleten, zato se zaradi lažjega razumevanja osredotočimo le na število strani, ki se jih prebere ali zapiše na disk in nam služijo kot število I/O operacij.
- Kot stroškovno metriko torej uporabljamo število I/O operacij, ki prenašajo (branje in pisanje) strani

Stroškovni model

- Poenostavitve lahko uporabimo, saj se izkaže, da predstavljeni model kljub poenostavitvi odraža dejanske trende pri uporabi različnih datotečnih organizacij

Primerjava datotečnih organizacij..

- Primerjamo stroške preprostih operacij za tri vrste datotek:
  - neurejeno datoteko (**heap**): zapisi si vrstijo naključno,
  - urejeno datoteko (**sorted**): vrstni red zapisov je določen glede na *iskalni ključ*),
  - razpršeno datoteko (**hashed**): zapisi so grupirani v skupine na podlagi razpršilne funkcije (hash function).
- Cilj je poudariti, kako pomembna je pravilna izbira datotečne organizacije

Primerjava datotečnih organizacij..

- Operacije:
  - **skeniranje**: preberi vse zapise v datoteki. Vse strani iz datoteke je potrebno prenesti v Buffer pool
  - **iskanje s pogojem ekvivalence**: poišči vse zapise, ki ustrezajo določenemu pogoju. Strani, ki vsebujejo ustrezne zapise je potrebno prenesti z diska in na teh straneh je potrebno locirati želene zapise

Primerjava datotečnih organizacij..

- Operacije:
  - **iskanje z definiranjem območja**: poišče vse zapise iz določenega območja. Ustrezne strani je potrebno presesti z diska v Buffer Pool
  - **vstavljanje**: Vstavi dan zapis v datoteko. Potrebno je identificirati stran datoteke, v katero se bo zapis vstavil, prebrati to stran z diska, jo ažurirati in zapisati nazaj na disk. Od datotečne organizacije je odvisno ali bo potrebno poleg želene strani, z diska brati in nanj pisati še dodatne strani
  - **brisanje**: Brisanje zapisa s podanim rid (record id). Potrebno je identificirati stran z danim zapisom, jo prebrati z diska, jo ažurirati in zapisati nazaj na disk

Primerjava datotečnih organizacij..

- Ko zanemarimo v izračunih vrednosti za C in H, potem dobimo naslednjo primerjalno tabelo z ocenami za posamezno datotečno organizacijo

Primerjava datotečnih organizacij..

- Značilnosti:
  - **neurejena datoteka**: hitro skeniranje, vstavljanje in brisanje; počasno iskanje
  - **urejena datoteka**: omogoča hitro iskanje območja; srednja hitrost pri iskanju; počasno vstavljanje in brisanje
  - **razpršena datoteka**: hitro vstavljanje in brisanje zapisov; zelo hitro je iskanje na osnovi enakosti; nobene podpore izboru območja; skeniranje celotne datoteke je nekoliko počasnejše

File Type	*Scan*	*Equality Search*	*Range Search*	*Insert*	*Delete*
Heap	BD	$0.5BD$	BD	$2D$	$Search + D$
Sorted	BD	$Dlog_2B$	$Dlog_2B$+# *matches*	$Search + BD$	$Search + BD$
Hashed	$1.25BD$	D	$1.25BD$	$2D$	$Search + D$

Figure 8.1 A Comparison of I/O Costs

Primerjava datotečnih organizacij

- Zaključek:
  - Nobena organizacija ni dobra za vse operacije.
  - Neurejena datoteka je zelo primerna za skeniranje celotne datoteke
  - Razpršena datoteka je najprimernejša pri uporabi iskanja s pogojem ekvivalence
  - Urejena datoteka je najprimernejša pri iskanju intervalov

Indeksi - uvod..

- Včasih hočemo najti vse zapise, ki imajo določeno vrednost v določenem polju
- INDEKS je zunanja podatkovna struktura, ki pomaga najti zapise, ki ustrezajo določenemu pogoju
- INDEKS lahko bistveno pospeši iskanje

Indeksi - uvod..

- Vsak indeks vsebuje **iskalni ključ**, ki je sestavljen iz **enega ali več polj** zapisov, ki se nahajajo v datoteki. Iskalni ključ je lahko katerikoli podmnožica polj zapisov v datoteki
- **Primer**: Indeks nad tabelo ŠTUDENT ima lahko indekse nad naslednjimi polji oz. iskalnimi ključi:
  - Indeks1: <Vpisna stevilka>
  - Indeks2: <ime, priimek>
  - Indeks3: <kraj, ime, priimek>

Indeksi - uvod..

- Datoteki, ki jo indeksiramo, pravimo tudi **indeksirana datoteka**

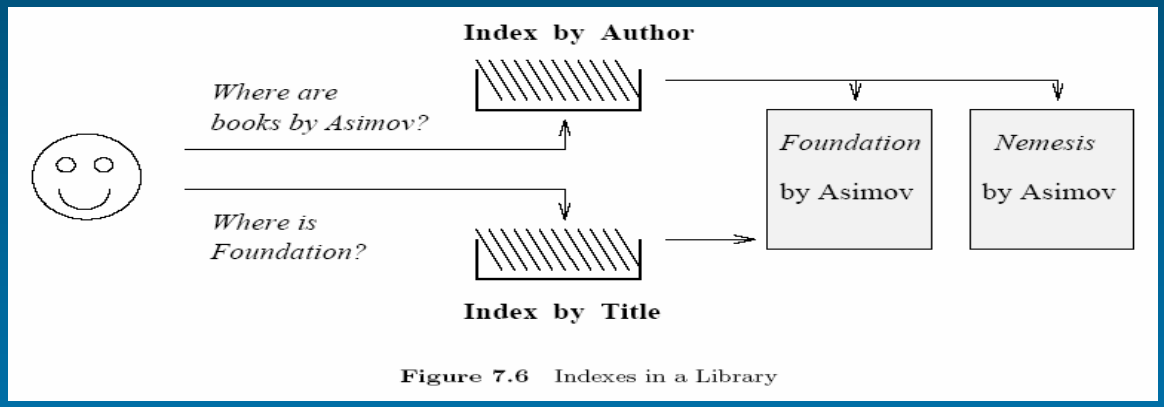


Figure 7.6 Indexes in a Library

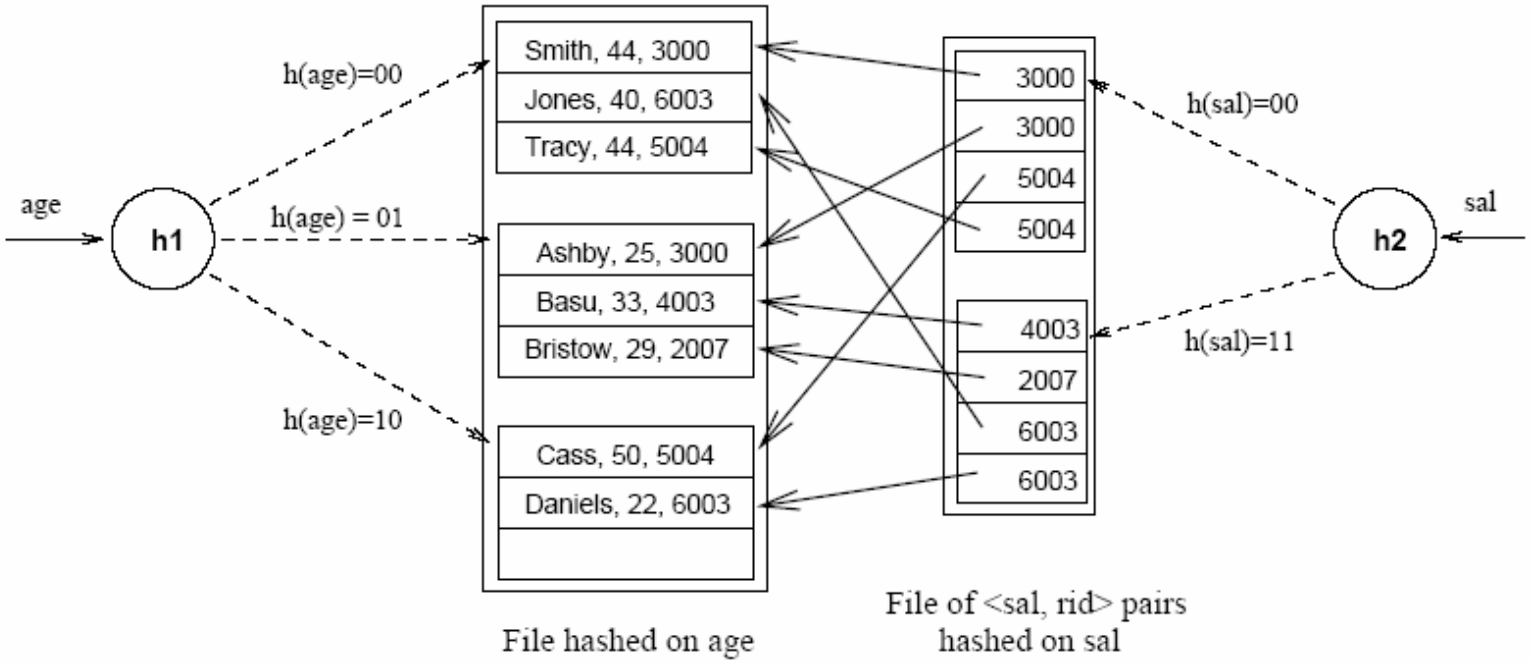
Indeksi - uvod..

- Indeks je datoteka, katere vsebina omogoča **hiter dostop** do zapisov v osnovni datoteki, ki je z indeksom indeksirana
- Pod hitrim dostopom pojmujemo dostop, ki je (občutneje) hitrejši, kot pa ga sicer omogoča organizacija osnovne datoteke
- Indeks si lahko predstavljamo **kot zbirko vpisov** *k**, kjer je *k* iskalni ključ, * pa kazalec(ci) na zapise, ki imajo polja v zapisu enaka ključu indeksa

Indeksi - uvod..

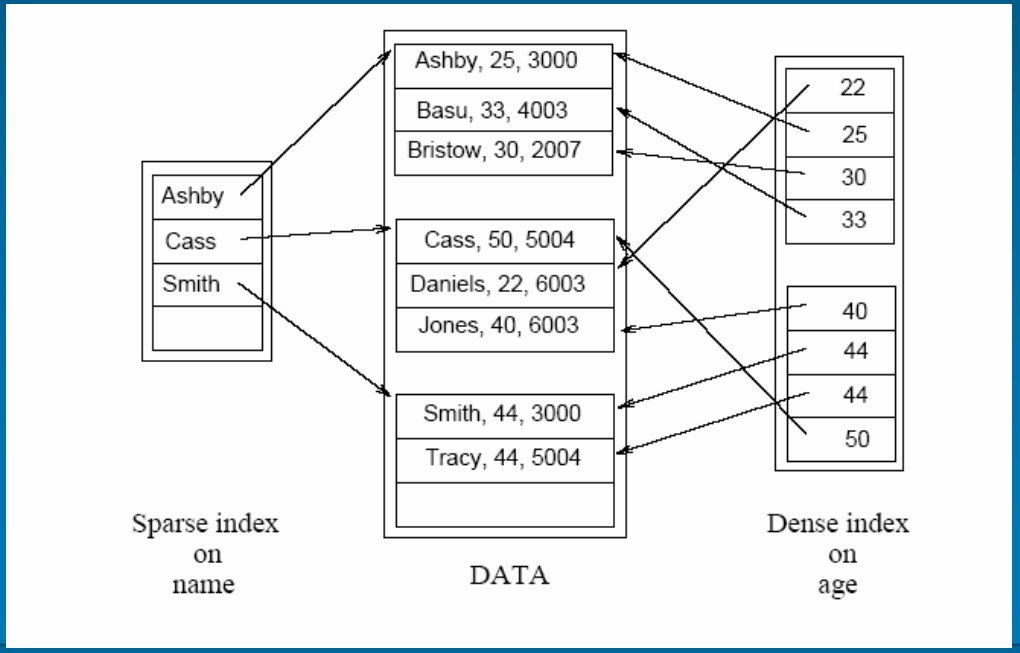
- Pojavita se 2 vprašanji:
  - Kako so podatkovni vpisi v indeksu organizirani, da bodo podpirali hitro in učinkovito iskanje zapisov?
    - Ena rešitev za organizacijo indeksa je uporaba razpršene datoteke (naslednja slika), ki omogoča hitro iskanje zapisov
    - Obstaja tudi več drugih indeksnih struktur, ki omogočajo hitro iskanje in bodo predstavljene v nadaljevanju
  - Kaj natančno je shranjeno v podatkovnem vpisu (Podatkovni vpis *k** nam omogoča dobiti enega ali več zapisov z vrednostjo ključa *k*)
    1. k* je dejanski zapis (z iskalnim ključem k),
    2. podatkovni vpis je par <k, rid>, kjer rid predstavlja *record id* zapisa z vrednostjo indeksnega ključa k,
    3. podatkovni vpis je par <k, rid-list>, kjer je rid-list seznam *record id-jev* tistih zapisov, ki imajo v poljih vrednost enako indeksnemu ključu.

Indeksi - uvod



Vrste indeksov..

- **Gosti indeks**: na vsak zapis osnovne datoteke, kaže kazalec iz indeksa
- **Redki indeks**: kazalci iz indeksa kažejo na skupine zapisov osnovne datoteke



Vrste indeksov..

- **Primarni indeks:** indeksiranje osnovne datoteke je izvedeno po njenem ključu (iskalni ključ in primarni ključ osnovne datoteke sta enaka)
- **Sekundarni indeks:** indeksiranje osnovne datoteke je izvedeno po podatkovnem elementu (polju), ki nastopa v zapisu osnovne datoteke, vendar pa ni ključ

Vrste indeksov..

- **Enonivojsko indeksiranje**: indeksirana je osnovna datoteka – v indeksu poiščemo kazalec na polje ali skupino polj v osnovni datoteki, kjer nato poiščemo iskani zapis
- **Večnivojsko indeksiranje**: indeksirana je osnovna datoteka, indeksiran je indeks na osnovno datoteko, indeksiran je indeks na indeks itd.

Vrste indeksov

- **Statično indeksiranje**: indeks ostaja nespremenjen, čeprav se osnovni datoteki zapisi dodajajo ali iz nje brišejo; ko postane iskanje neučinkovito, se izvede reorganizacija indeksa
- **Dinamično indeksiranje**: indeks se ob dodajanju in brisanju zapisov osnovne datoteke prilagaja vsebini, tako da so iskalne poti do vseh zapisov v osnovni datoteki enako dolge; indeksa ni potrebno nikoli reorganizirati

Primarni in sekundarni indeks

- Primarni indeks ne vsebuje duplikatov
- Sekundarni indeks lahko vsebuje duplikate
- Če vemo, da duplikati ne obstajajo, potem pravimo temu indeksu **unique indeks**
- Za dva podatkovna vpisa pravimo, da sta **duplikata**, če vsebujeta enaki vrednosti za iskalni ključ

Indeksi s kompozitnimi iskalnimi ključi

- Indeksni ključ lahko vsebuje več polj. Takemu ključu pravimo **kompozitni iskalni ključ**
- Slika prikazuje razliko med kompozitnima iskalnima ključema <age,sal> in <sal,age>
- Če imamo indeks z indeksnim ključem, lahko izvajamo poizvedbe, pri katerih iščemo zapise z vrednostmi polj, ki so enake ključu (to imenujemo **equality query**)
- Druga vrsta poizvedbe pa nam omogoča, da je samo ena vrednost v ključu konstantna, druge pa ne. Tako lahko pri poizvedbi dobimo več zapisov (**range query**)

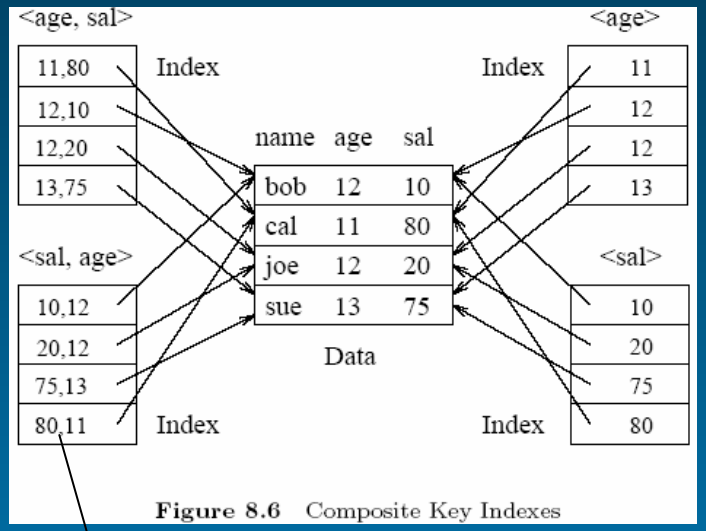


Figure 8.6 Composite Key Indexes

kompozitni indeksni ključ

Specifikacija indeksa v SQL 92

- Standard SQL 92 ne opredeljuje nobenega stavka za kreiranje ali brisanje indeksov. Še več, standard sploh ne zahteva implementacije indeksov
- V praksi je drugače; vsi komercialni SUPB podpirajo eno ali več vrst indeksiranja
- Primer SQL stavka za kreiranje B+ drevesnega indexa:

```
CREATE INDEX IndAgeRating ON Students
       WITH  STRUCTURE = BTREE,
             KEY = (age, gpa)
```

Drevesne strukture za indekse

- ISAM (Indexed Sequential Access Method):
  - je statičen indeks,
  - učinkovit v primerih, ko osnovna datoteka ne spreminja pogosto,
  - ni učinkovit pri datotekah, ki se hitro povečujejo ali krčijo.
- B+ index:
  - je dinamična struktura, ki se zelo dobro prilagaja spremembam v osnovni datoteki podatkov,
  - B+ indeks je najbolj uporabljana vrsta indeksa, saj omogoča hitro iskanje elementov, iskanje intervalov in se učinkovito prilagaja spremembam v osnovni datoteki.

ISAM..

- Za primer vzemimo sortirano datoteko oseb, po atributu “starost”
- Če hočemo najti vse osebe starejše od 30 let, moramo najprej najti prvo (s pomočjo binarnega iskanja), ki je starejša od 30 in od te naprej prebrati preostale zapise datoteke
- Lahko bi izdelali dodatno datoteko, s po enim zapisom za vsako stran v osnovni datoteki (dodatna datoteka je tudi urejena po atributu starost)

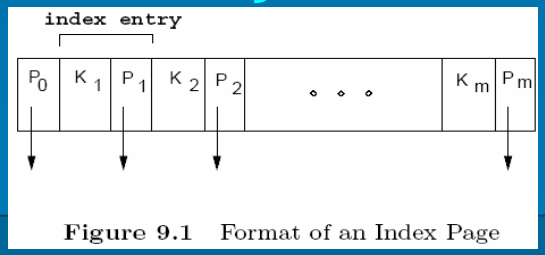


Figure 9.1 Format of an Index Page

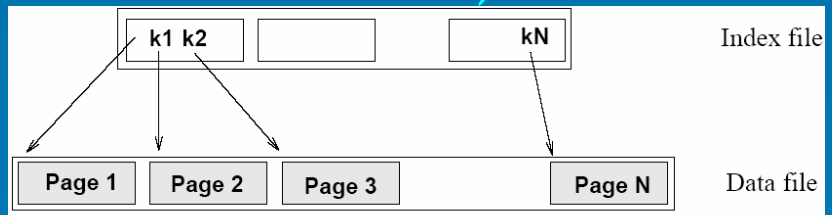


Figure 9.2 One-Level Index Structure

ISAM..

- Format strani v indeksni datoteki prikazuje leva slika. Vsaka stran v indeksu vsebuje en kazalec več, kot pa je ključev na strani. Ključ predstavlja separator vsebine, na katero kažeta levi in desni kazalec
- Binarno iskanje se sedaj izvede nad indeksno datoteko, ki je manjša od osnovne datoteke. Na ta način smo dosegli hitrejše iskanje

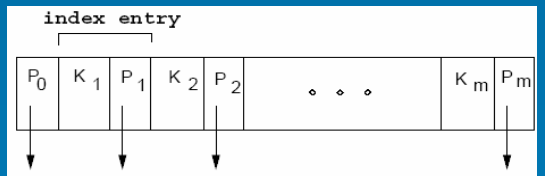


Figure 9.1 Format of an Index Page

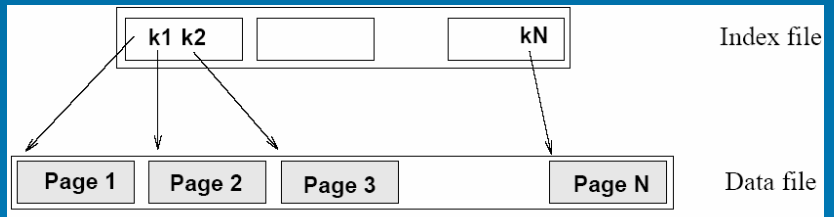
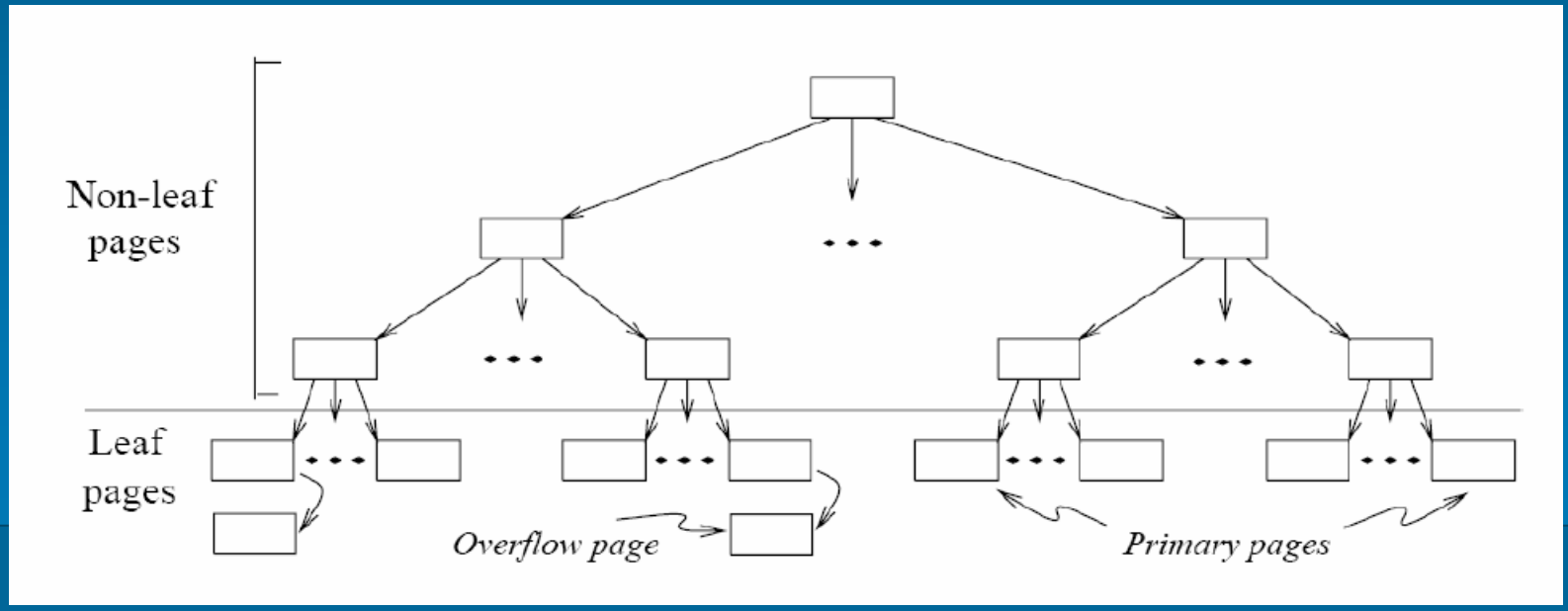


Figure 9.2 One-Level Index Structure

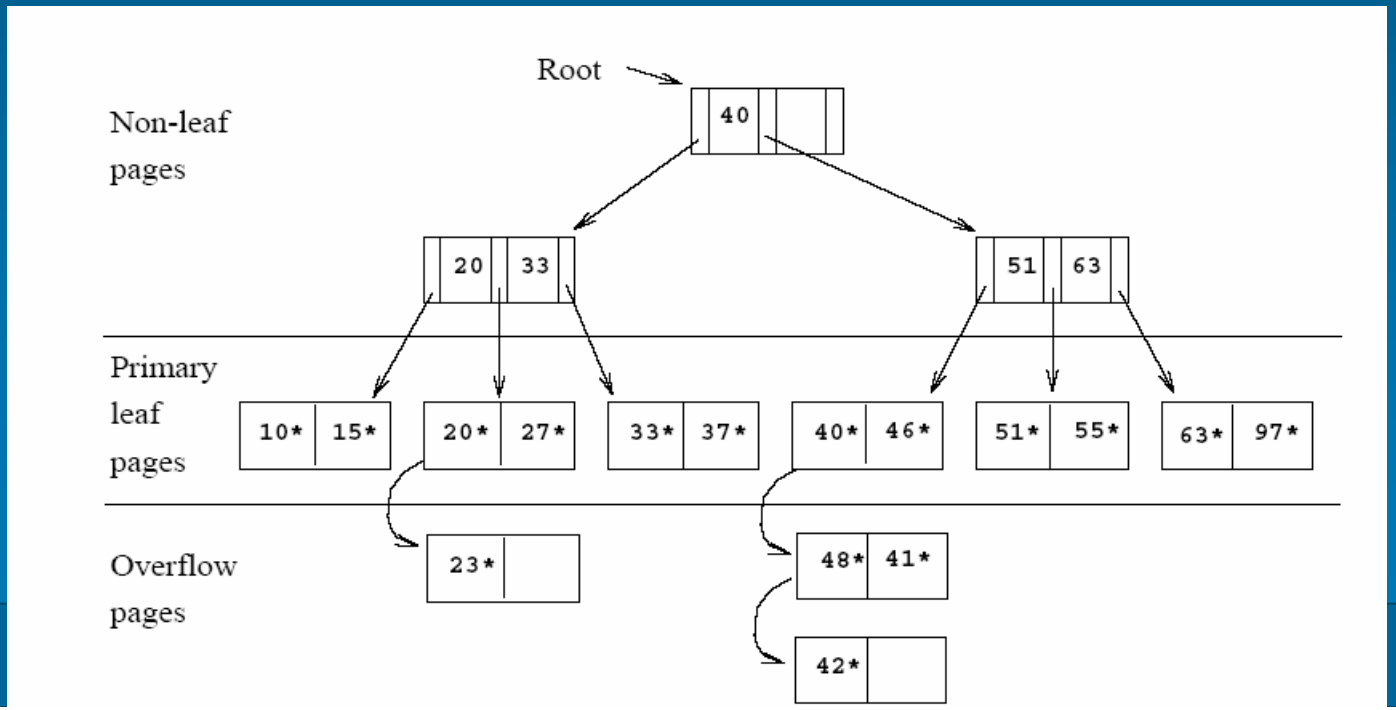
ISAM..

- Velikost indeksne datoteke poraja idejo o ISAM indeksu: Zakaj ne bi pri gradnji indeksa uporabili koncept REKURZIJE, tako da bi velikost vsakega posameznega indeksa znašala samo eno stran?
- Gradnja indeksa nas pripelje do drevesne strukture



ISAM..

- ISAM indeks sestavljajo tri vrste strani, ki vsebujejo:
  - Vozlišča, ki niso listi (**non-leaf pages**)
  - Vozlišča, ki so primarni list in so alocirani sekvenčno (vozlišča, ki so bila listi takoj po kreiranju indeksa) (**primary leaf pages**)
  - Vozlišča, ki so dodana primarnim listom (in so tudi primarni listi) zaradi spreminjanja vsebine tabele in s tem indeksa (**overflow pages**)



ISAM..

- Struktura ISAM indeksa je od njegovega kreiranja dalje popolnoma statična (razen overflow strani)
- Vsako indeksno vozlišče je velikosti ene strani, vsi podatki (pari: podatek, kazalec na podatkovno stran) pa se nahajajo v listih drevesa
- Ko se indeksna datoteka kreira, so vse strani v listih urejene zaporedno in urejene po iskalnem ključu

ISAM..

- Ko se podatkovno datoteko in posledično v listne strani dodaja podatke, je lahko potrebno v indeks dodati nove strani (overflow pages), kajti drevesna struktura indeksa je statična

ISAM..

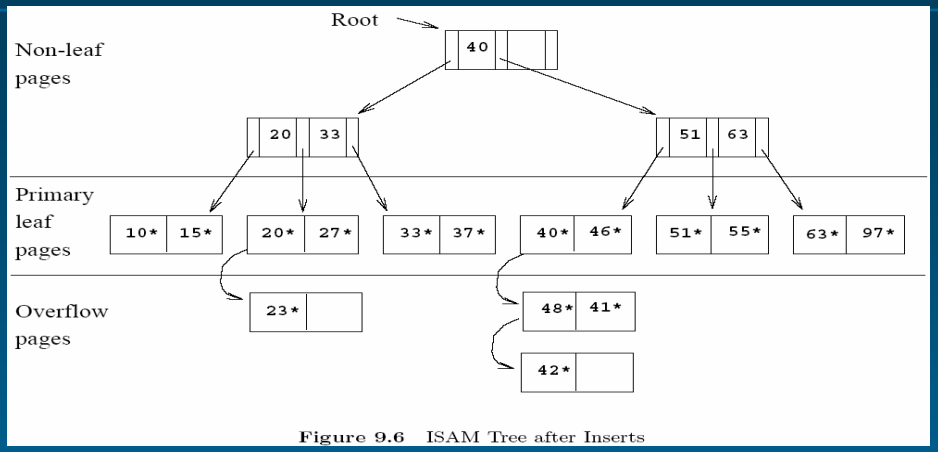


Figure 9.6 ISAM Tree after Inserts

- Ko je enkrat indeks kreiran, brisanja in vstavljanja vplivajo le na spremembe listov
- Dobilo lahko dolge verige overflow strani => počasno delovanje
- REŠITEV: ponovna gradnja indeksa (znebimo se overflow strani)
- ISAM je dober za datoteke, ki se jih malo ažurira. Začetni indeks se lahko zgradi tako, da so strani v listih 20% nezasedene (imamo nekaj maneverskega prostora za vstavljanje)

ISAM..

- Operaciji iskanja in vstavljanja

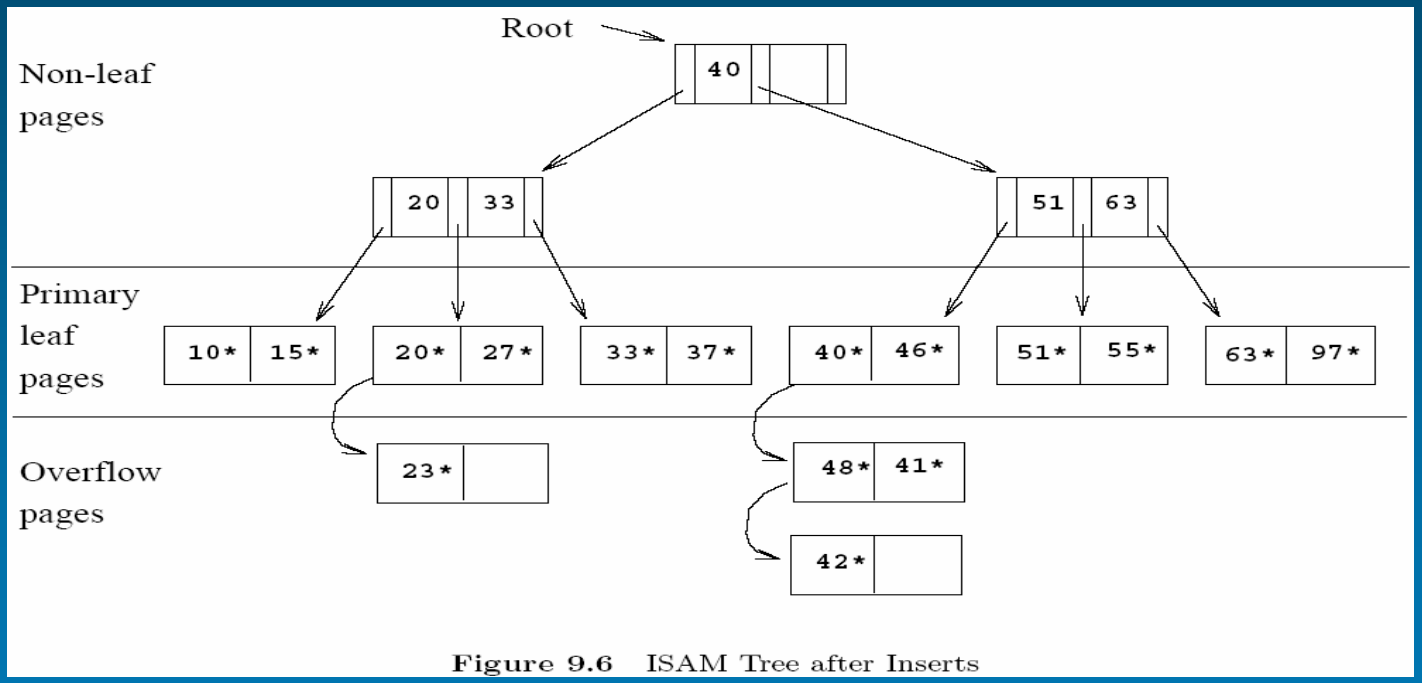


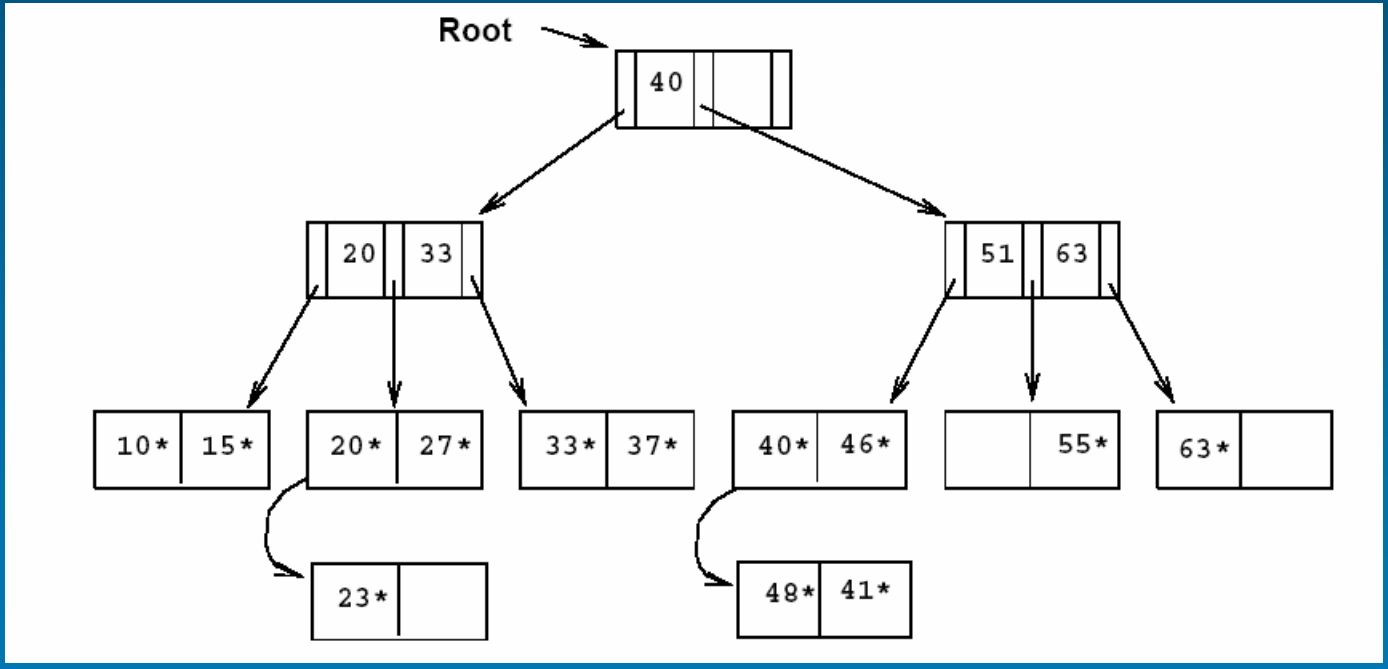
Figure 9.6 ISAM Tree after Inserts

ISAM..

- Operacija brisanja:
  - Če ostane vozlišče na overflow strani prazno, sta vozlišče (in stran) odstranjena
  - Če zbrišemo zapis, ki ima vrednost ključa enako podatku v primarnem listu, potem pustimo stran (kjer je primarni list) nespremenjeno (lahko ostane tudi prazna). Namenjena je bodočim vstavljanjem
  - Če zbrišemo zapis, ki ima ključ enak vrednosti v vozlišču, potem vozlišča, ki ni list, ne spreminjamo. Vozlišča, ki niso listi, so usmerjevalnega značaja za iskanje in ostale operacije (glej primer na naslednji strani)

ISAM..

- Po brisanju vrednosti 51 je ta vrednost ostala v vozlišču, ki ni list. Če bi naknadno iskali vrednost 51, bi šele v primarnem listu ugotovili, da je ni



ISAM

- Kompleksnost iskanja je $\log_F N$, kjer je N število strani s primarnimi listi, F pa število naslednikov indeksnega vozlišča (strani, ki vsebuje vmesna vozlišča, ki niso listi – nonleaf pages), kar je dosti hitreje kot binarno iskanje
- Dobra stran ISAM je, da zaradi statičnosti niso potrebna zaklepanja strani, ki niso listi, ker se vsebina le teh nikoli ne spreminja

B+ indeks..

- B+ indeks je dinamičen. Njegova struktura se dinamično prilagaja spremembam v osnovni datoteki
- B+ drevo predstavlja iskalno strukturo. To je uravnoteženo drevo, katerega vozlišča usmerjajo iskanje, listi pa vsebujejo podatke (ključe)
- Listi v B+ drevesu so povezani v dvosmerni urejen seznam strani

B+ indeks..

- Lastnosti B+ drevesa:
  - Operacije (insert, delete) ohranjajo drevo uravnoteženo,
  - Vozlišča (razen root-a) morajo biti vsaj 50% zasedena,
  - Iskanje določene vrednosti zahteva le pot od root vozlišča do ustreznega listnega vozlišča. Poti do vseh vozlišč so zaradi uravnoteženosti enake in določajo višino drevesa.

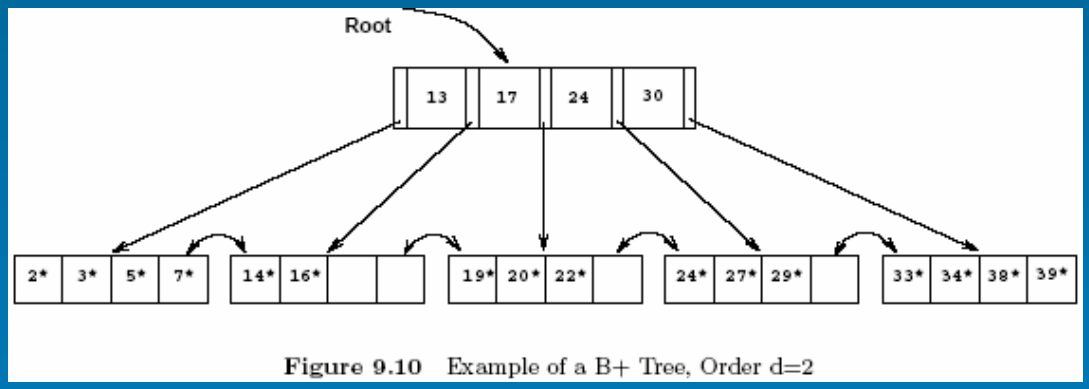


Figure 9.10 Example of a B+ Tree, Order d=2

B+ indeks

- Vsako vozlišče B+ drevesa vsebuje m vpisov; d=<m=<2d
- d je parameter B+ drevesa (red drevesa) in predstavlja kapaciteto vozlišča. Edina izjema je korensko vozlišče, za katerega velja 1=<m=<3d
- Lastnosti B+ dreves si poglejmo skozi uporabo operacij Iskanja, Vstavljanja in Brisanja. Predpostavili bomo, da v osnovni datoteki ni duplikatov. Primer našega B+ indeksa prikazuje slika (d=2):

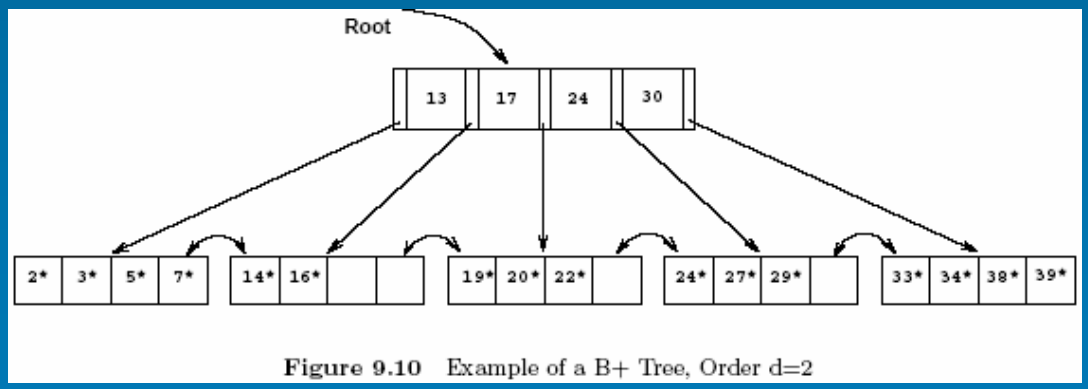


Figure 9.10 Example of a B+ Tree, Order d=2

B+ indeks - Iskanje

- Algoritem išče list, v katerem se nahaja iskalni ključ
- Če iščemo vrednost 5, sledimo levemu kazalcu, ker je 5 < 13
- Pri iskanju 14 ali 15 sledimo 2. kazalcu. Vrednosti 15 v listu ne najde, zato iskanje ne uspe

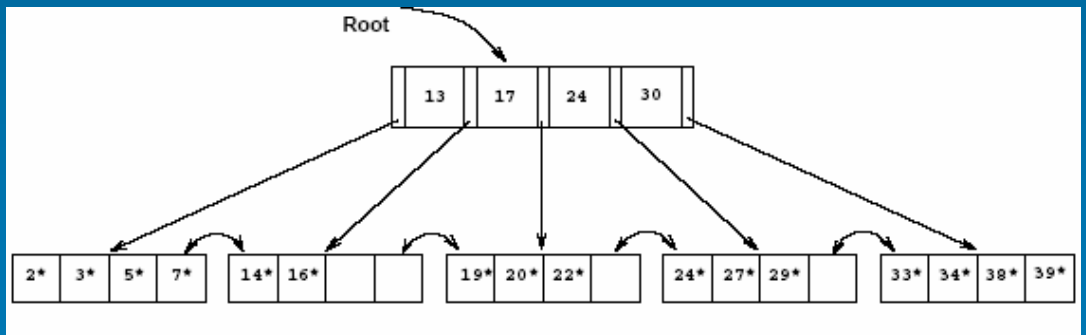


Figure 9.10 Example of a B+ Tree, Order d=2

B+ indeks – Vstavljanje..

- Algoritem za vstavljanje poišče list, v katerega spada nova vrednost in jo vanj vstavi, če to dopušča zasedenost lista
- Lahko je potrebno opraviti distribucijo ali razcepljanje strani

B+ indeks - Vstavljanje..

- Vstavljanje vrednosti 8 z uporabo razcepljanja

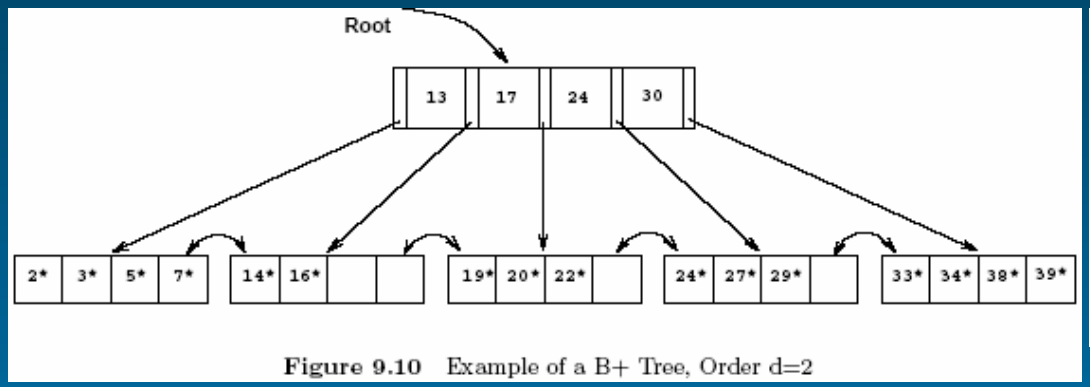
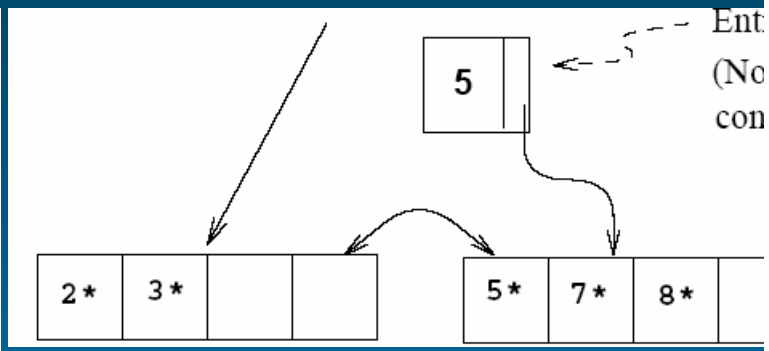
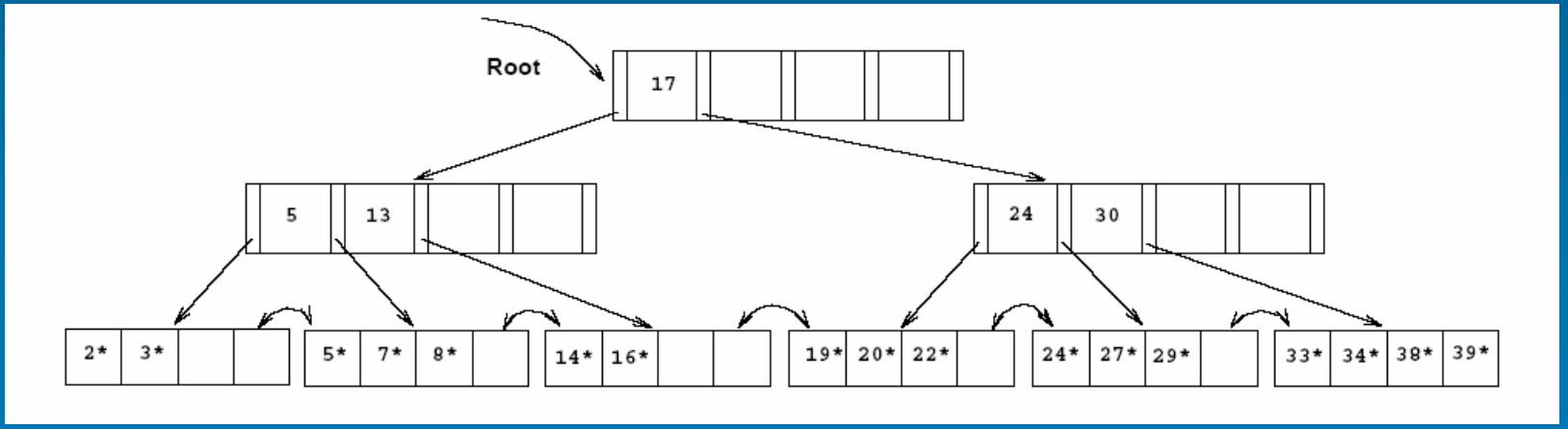


Figure 9.10 Example of a B+ Tree, Order d=2





B+ indeks - Vstavljanje

- Vstavljanje vrednosti 8 z uporabo redistribucije

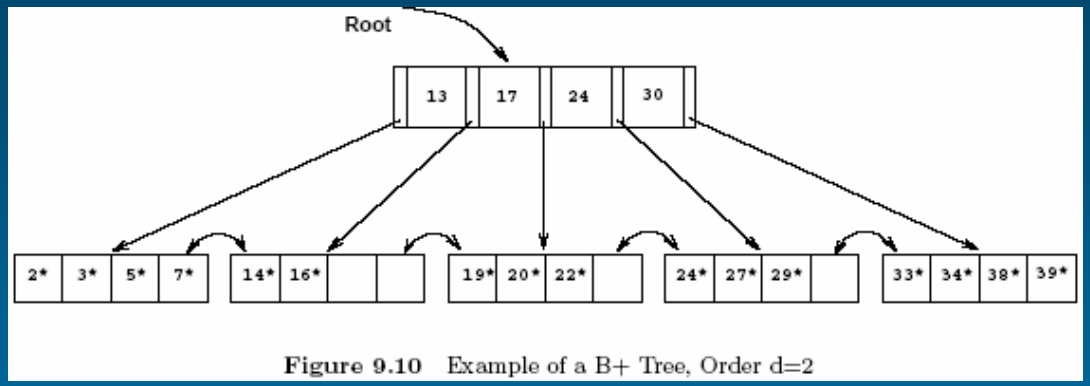
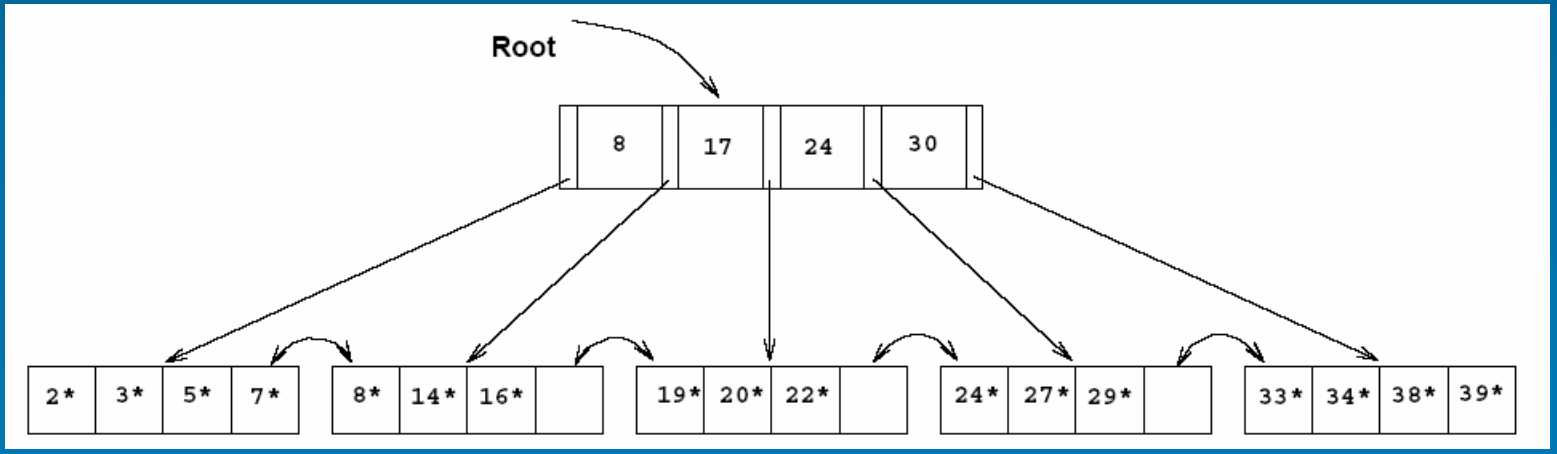


Figure 9.10 Example of a B+ Tree, Order d=2

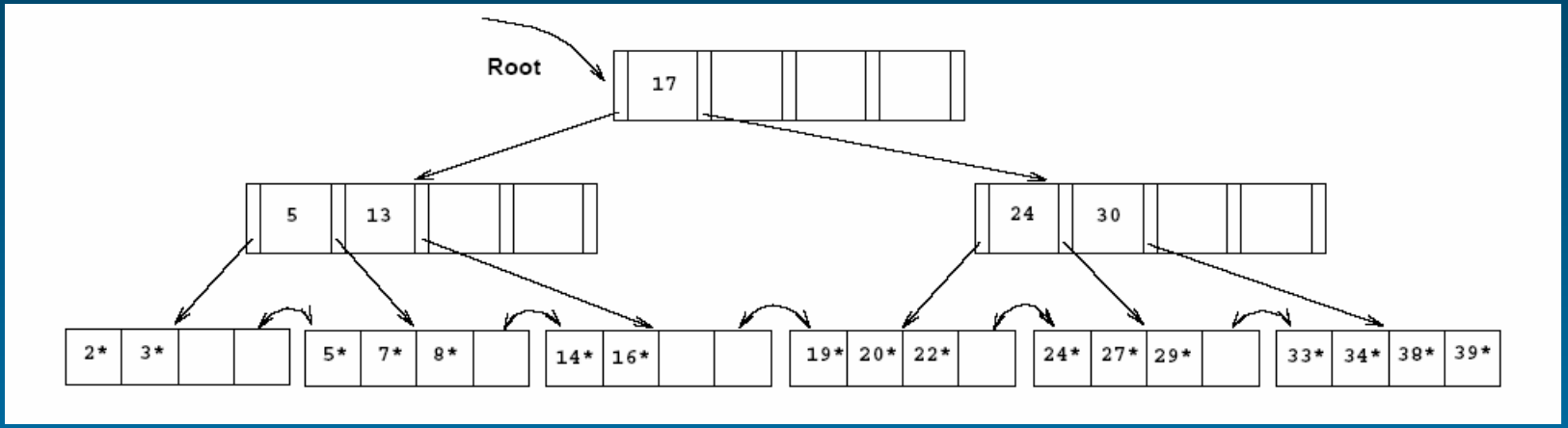


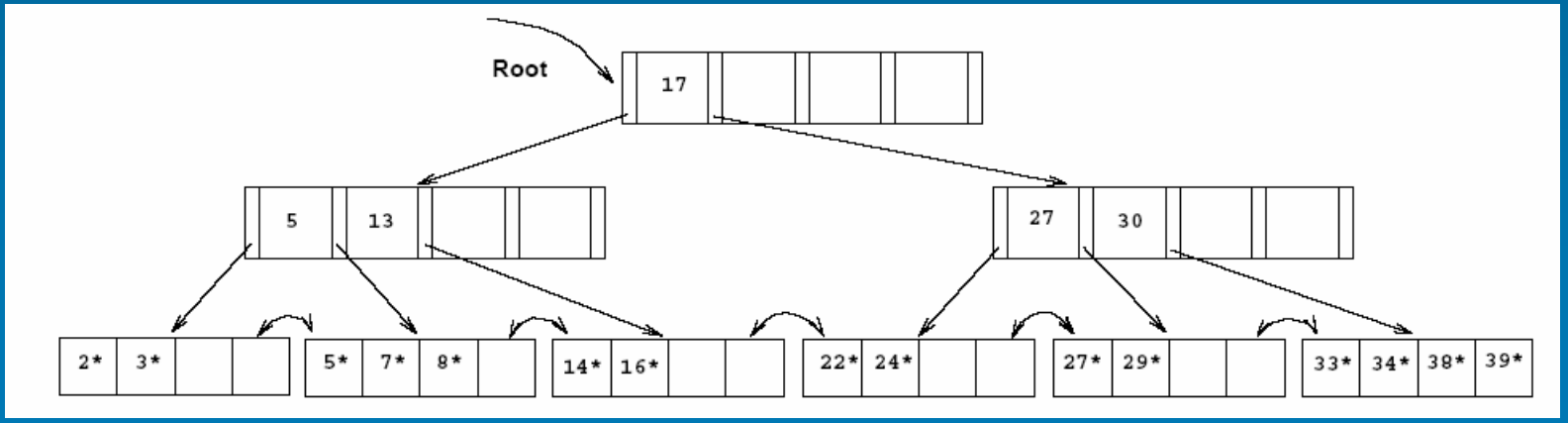
B+ indeks – Brisanje..

- Algoritem za brisanje poišče list, kjer je podatek za brisanje in ga odstrani iz drevesa
- Lahko se zgodi, da se zasedenost vozlišča po brisanju zniža pod dovoljeno mejo
- V tem primeru je potrebno uporabiti redistribucijo ali pa zlivanje dveh vozlišč v eno (v tem primeru posodobiti tudi prednike)

B+ indeks - Brisanje

- Brisanje ključev 19 in 20 z uporabo redistribucije





B+ indeks - Duplikati

- Lahko se poslužujemo overflow strani, kot v ISAM
- Lahko jih vstavljamo “normalno”, potrebno pa je spremeniti algoritem iskanja tako, da bo vedno poiskal “najbolj levi element” z iskano vrednostjo ključa
- Uporaba podatkovni vpis je par <k, rid-list> je najbolj naravna v tem primeru

Poglavje 9

Obnavljanje PB

Povzeto po [4]

Podatkovne nesreče - Uvod..

- SUPB mora med drugim uporabnikom zagotavljati tudi nemoten dostop do celovitih – neokrnjenih podatkov
- Dostop do podatkov in njihovo celovitost lahko ogrozijo dogodki, ki se imenujejo podatkovne nesreče
- Podatkovne nesreče lahko povzročijo nekonsistentnost podatkov med seboj ali s stvarnostjo, ali pa onemogočijo nadaljnji dostop do PB

Podatkovne nesreče - Uvod..

- Postopke, s katerimi se ohranja konsistentnost PB po nesrečah, imenujemo obnavljanje (recovery). Te postopke izvaja SUPB pod nadzorom skrbnika PB
- Obnavljanje obsega:
  - Pripravo podatkov za obnovitev
  - Detekcijo podatkovne nesreče, ki je povzročila nekonsistentno stanje ali nedostopnost podatkov
  - Obnovitev PB v stanje pred nesrečo

Podatkovne nesreče - Uvod..

- Na PB lahko gledamo kot na sistem, ki se sme nahajati le v veljavnih stanjih, v katerih so podatki v PB med seboj konsistentni
- Prehod iz veljavnega stanja v veljavno stanje se izvede z ažuriranjem podatkov: skupino operacij, ki poizvedujejo in spreminjajo podatke

Podatkovne nesreče - Uvod..

- Zaporedje ažuriranj, ki povzroče prehod PB iz enega v drugo veljavno stanje, se imenuje transakcija
- Med izvajanjem transakcije se PB lahko nahaja tudi v neveljavnih stanjih
- V neveljavnem stanju lahko PB tudi obstane, če se med izvajanjem zgodi podatkovna nesreča, ker se v tem primeru transakcija ne konča

Podatkovne nesreče - Uvod..

- Obnavljanje PB mora zagotoviti, da se PB po nesreči obnovi v eno izmed veljavnih stanj:
  - to je lahko zadnje veljavno stanje ali
  - eno izmed prej veljavnih stanj
- V vseh veljavnih stanjih so podatki med seboj sicer konsistentni, vprašanje je, ali so podatki konsistentni s stvarnostjo
- S stvarnostjo so najbolj konsistentni podatki zadnjega veljavnega stanja

Podatkovne nesreče - Uvod..

- Obnavljanje ne obsega samo povrnitev PB v neko veljavno stanje, ampak tudi zagotavljanje podatkov, ki so potrebni za obnavljanje
- Postopki za zagotavljanje obnovitvenih podatkov se morajo izvajati z ažuriranjem PB, v okviru transakcij

Podatkovne nesreče - Uvod

- Posledično je to dodatna obremenitev sistema, kar pomeni slabše performanse
- Čim hitrejšo in čim kakovostnejšo obnovitev PB po nesreči želimo, tem več dodatnih postopkov je potrebno izvajati med transakcijami

Transakcije..

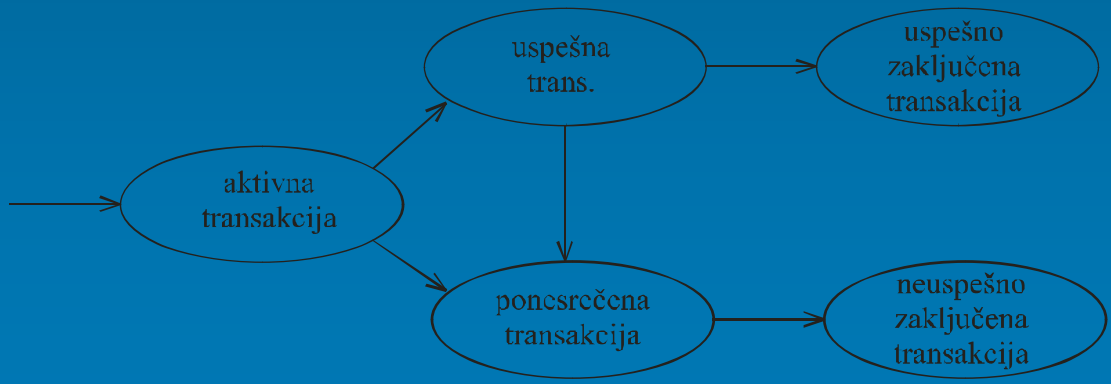
- Transakcija je logični vrstni red ažuriranj in povpraševanj, ki prevede PB iz enega v novo veljavno stanje. Transakcija ohranja konsistentnost podatkov v PB
- Če hočemo ohraniti konsistentno PB, se mora transakcija izvesti v celoti, ali pa sploh ne. Oziroma: izvesti se morajo vsa ažuriranja transakcije ali nobeno

Transakcije..

- Transakcija se prične z operacijo "*Začetek transakcije*", konča pa ali z ukazom "*Pomni*" (COMMIT) ali "*Pozabi*" (CANCEL, ROLLBACK)
- Ukaz COMMIT: sporočilo SUPB, da so se vsa ažuriranja v okviru transakcije uspešno izvedla in da naj se vse spremembe v PB ohranijo
- Ukaz ROLLBACK: navodilo SUPB, naj se vsa v okviru transakcije izvedena ažuriranja razveljavijo

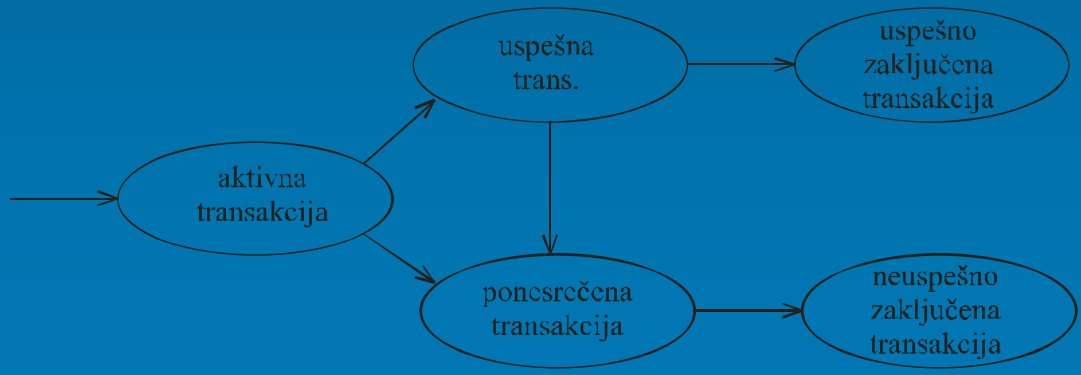
Transakcije..

- Življenski cikel transakcije lahko predstavimo z diagramom prehajanja stanj – transakcija se nahaja v več stanjih
- Transakcija je aktivna od ukaza “Začetek transakcije” do ukaza “Commit” (preide v stanje “uspešna transakcija”)



Transakcije..

- Pri nekaterih vrstah obnavljanja mora SUPB vsa ažuriranja uspešne transakcije šele uveljaviti v PB, ker se dejanska ažuriranja v PB še niso izvedla
- Ko je uveljavljanje ažuriranj izvedeno, postane transakcija uspešno zaključena (stanje “uspešno zaključena transakcija”)



Transakcije

- Če se med uveljavljanjem ažuriranj zgodi podatkovna nesreča, postane transakcija ponesrečena
- Tudi aktivna transakcija se lahko ponesreči in tako postane ponesrečena transakcija
- V takih primerih mora SUPB poskrbeti za razveljavitev vseh morebitnih ažuriranj v PB, nato pa postane transakcija “neuspešno zaključena”
- Če se transakcija ni ponesrečila po lastni krivdi (na primer zaradi neustreznih vhodnih podatkov), jo SUPB vrne v ponovno izvajanje

Vrste podatkovnih nesreč..

- Podatkovne nesreče se med seboj razlikujejo:
  - po vzrokih
  - po tem, kdo odkrije nesrečo
  - po posledicah, ki jih povzroče
- Od vrste podatkovnih nesreč so odvisni nadaljnji postopki obnavljanja
- Vrste podatkovnih nesreč:
  - transakcijske nesreče, ki jih odkrijejo uporabniški programi
  - transakcijske nesreče, ki jih odkrije SUPB
  - sistemske nesreče
  - diskovne nesreče

Vrste podatkovnih nesreč

- 1. in 2. skupina predstavljata nesreče lokalnega značaja – samo posamezna transakcija se ne zaključi uspešno
- 3. in 4. skupina predstavlja nesreče s širšimi posledicami

Transakcijske nesreče..

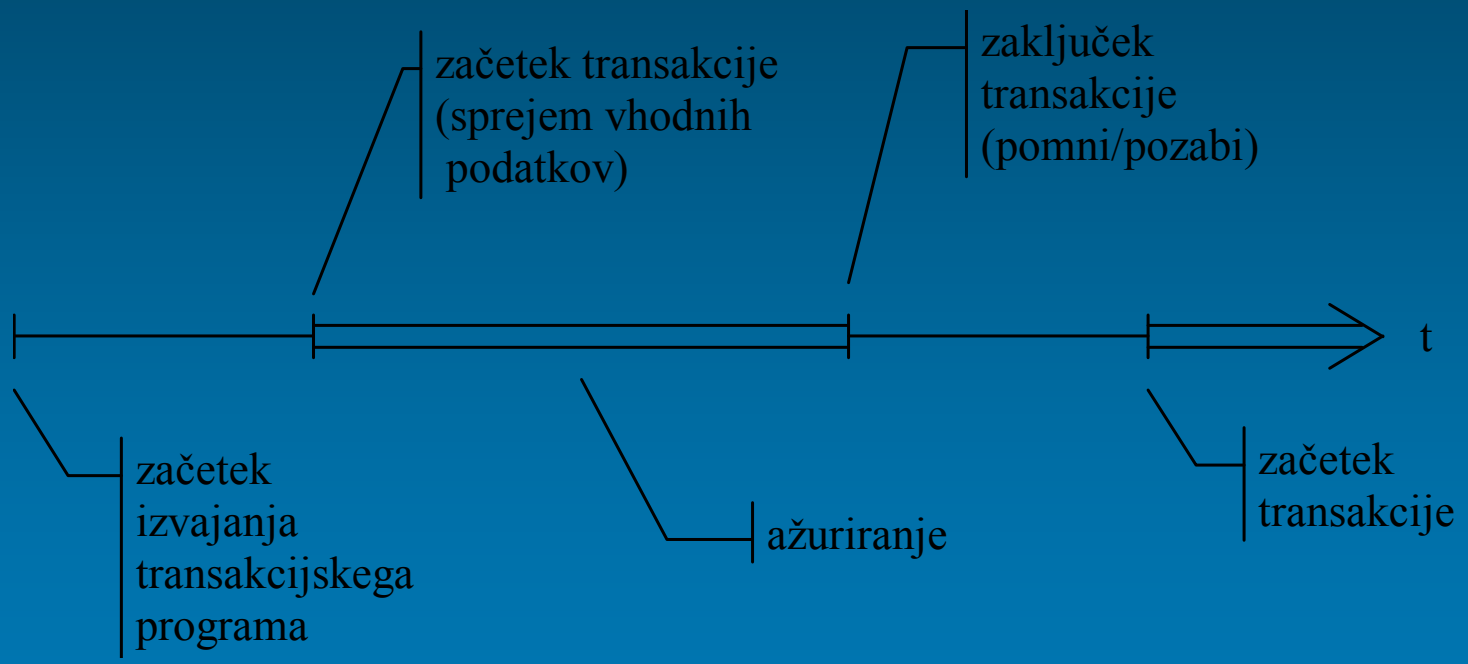
- Transakcijske nesreče, ki jih odkrijejo uporabniški programi:
  - Npr.: Med izvajanjem transakcije se lahko izkaže, da so vhodni podatki v protislovju s podatki v podatkovni bazi, in da transakcije ni mogoče uspešno izvesti. V takem primeru izda transakcijski program ukaz **Rollback** SUPB, katerega naloga je nato, da vsa že izvedena ažuriranja podatkovne baze v okviru transakcije razveljavi
- Pri izvajanju transakcije lahko pride do prekinitve izvajanja transakcijskega programa:
  - Razlog je lahko slabo napisan ali slabo preizkušen transakcijski program

Transakcijske nesreče..

- Transakcijo lahko prekine tudi proces za nadzor sočasnega izvajanja transakcij, če ugotovi, da se neka transakcija ne more sočasno izvajati z drugim, ker bi to ogrozilo konsistentnost PB. V tem primeru gre za transakcijsko nesrečo, ko jo odkrije SUPB.

Transakcijske nesreče

- Izvajanje transakcije:



Sistemske nesreče..

- Za sistemsko nesrečo štejemo izgubo podatkov v notranjem pomnilniku, zaradi prekinitve delovanja le-tega
- Napaka je lahko povzročena s:
  - prekinitvijo napajanja
  - napako pri branju ukaza ali podatka iz notranjega pomnilnika

Sistemske nesreče

- Sistemska nesreča povzroči prekinitev izvajanja trenutno aktivnih transakcij
- Pred nadaljnjo uporabo PB po sistemski nesreči, je potrebno obnoviti PB v zadnje veljavno stanje pred nesrečo in ponoviti izvajanje prekinjenih transakcij

Diskovne nesreče..

- Tudi podatki, ki se nahajajo na sekundarnem pomnilniku (trdi disk), se lahko uničijo ali izgubijo
- Razlogi za okvaro:
  - okvara diskovne površine
  - okvara bralno pisalnih glav
  - okvara krmilnika diska itd.
- Tudi pri teh nesrečah so lahko nekatere transakcije med svojim izvajanjem prekinjene

Diskovne nesreče

- Ko PB spet začne delovati, je potrebno obnoviti PB v zadnje veljavno stanje pred nesrečo in ponoviti izvajanje prekinjenih transakcij

Vrste obnavljanja PB – Uvod..

- Obnavljanje PB temelji na redundantnih podatkih
- Redundantni podatki se generirajo občasno ali sočasno z izvajanjem operacij v okviru transakcij

Vrste obnavljanja PB – Uvod..

- Podatki za obnavljanje:
  - kopija PB
  - vhodni transakcijski podatki, na osnovi katerih se izvajajo transakcije in oznaka transakcijskega programa
  - vrednosti zapisov v PB pred njihovim ažuriranjem (stare vrednosti zapisov) – omogoča razveljavitev ažuriranj neuspešno zaključenih transakcij
  - vrednosti zapisov v PB po njihovem ažuriranju (nove vrednosti zapisov) – omogoča ponovitev ažuriranj uspešno izvedenih transakcij, ki so se izgubila v podatkovni nesreči
- Za obnavljanje lahko uporabimo vse naštete podatke ali le nekatere izmed njih

Vrste obnavljanja PB – Uvod..

- Obnavljanja se razlikujejo glede na:
  - hitrost obnovitve PB
  - pogostost in količino beleženja redundantnih podatkov
  - poslabšanja odzivnih časov, zaradi beleženja redundantnih podatkov
- Čim pogostejše in v čim popolnejšem obsegu se beležijo redundantni podatki, tem **hitrejša** je obnovitev PB

Vrste obnavljanja PB – Uvod

- S tem pa so povezani slabši odzivni časi SUPB in večji stroški zaradi dodatnega pomnilnega prostora
- Odločiti se je potrebno za nek pameten kompromis med odzivnostjo in stroški!!!

Dvojna podatkovna baza..

- V primeru vzdrževanja dvojne podatkovne baze obstaja več možnosti, kaj vse podvojiti:
  - najenostavnejši primer: diskovni krmilnik zapisuje sočasno podatke na 2 fizično ločena diska. Če se ena PB pokvari, se uporablja njena dvojnica
  - za večjo zanesljivost: podvoji se tudi diskovni krmilnik ali kar cel računalniški sistem
- Omenjena zaščita je relativno draga. Omogoča pa hitro obnavljanje PB predvsem po diskovnih in delno sistemskih podatkovnih nesrečah

Dvojna podatkovna baza..

- Ob tem je koristno uporabljati še kakšen drug način beleženja redundantnih podatkov, zaradi primera, ko postaneta nedostopni obe PB. Na primer: izpad elektrike

Dvojna podatkovna baza..

- Preprostejša varianta dvojne PB: kopija, kjer se vsebina PB periodično kopira na magnetni trak. Ali pa se PB delno ali v celoti “exportira” v datoteko na disk, ki je potem vir za obnovitev PB
- Kopija PB omogoča popolno obnovitev PB v primeru diskovnih nesreč, s čimer preide v veljavno stanje, v katerem se je nahajala v trenutku izdelave kopije

Dvojna podatkovna baza

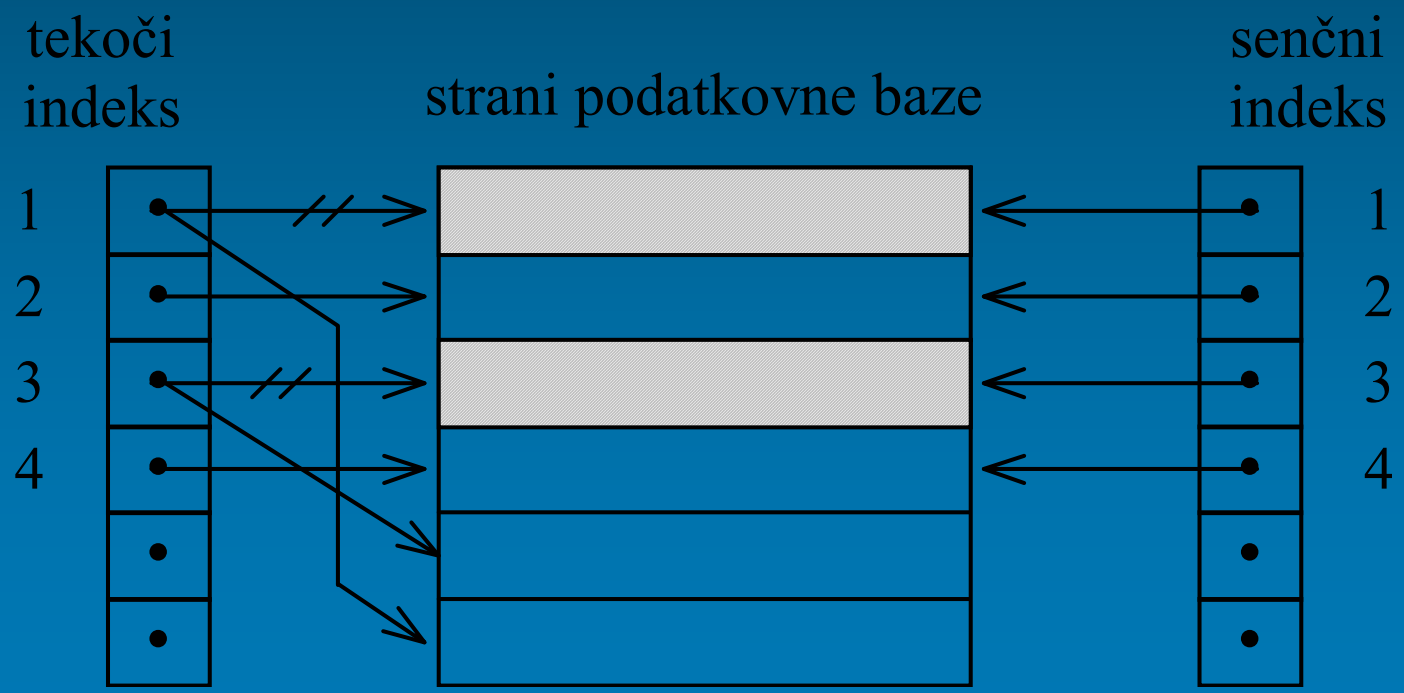
- Med izdelavo kopije naj se transakcije ne bi izvajale, zato da se bo kopija PB zanesljivo nahajala v veljavnem stanju
- PB je možno tudi delno ali inkrementalno kopirati. To kopiranje se lahko izvaja tudi v času, ko je PB v uporabi

Obnavljanje s senčnimi stranmi..

- Obnavljanje s senčnimi stranmi je učinkovito predvsem pri transakcijskih podatkovnih nesrečah, ko je potrebno razveljaviti že izvedena ažuriranja PB
- Princip delovanja:
  - Strani notranjega pomnilnika, ki so bile ažurirane, se ne zapisujejo neposredno v PB, ampak na nezasedene bloke na disku
  - Če se transakcija zaključi uspešno, se omenjeni novi bloki na disku vključijo v PB, namesto starih neažuriranih
- Naslov vsake strani na disku je zapisan v indeksu strani

Obnavljanje s senčnimi stranmi..

- Obstajata dva indeksa strani:
  - **tekoči** indeks strani
  - **senčni** indeks strani



Obnavljanje s senčnimi stranmi..

- Na začetku transakcije sta oba indeksa enaka. Tekoči indeks se nahaja v notranjem pomnilniku, senčni pa na disku. Naslov senčnega indeksa na disku je zapisan v posebni startni datoteki
- Operaciji PreberiBlok(X) in IzpišiBlok(X) uporabljata za dostop do blokov na disku (povpraševanje, ažuriranje) tekoči indeks

Obnavljanje s senčnimi stranmi..

Obnavljanje s senčnimi stranmi..

- Ažurirni ukaz Spremeni(X,x) se izvede v naslednjih korakih:
  - če se stran X s podatkom x še ne nahaja v notranjem pomnilniku, se izvede operacija PreberiBlok(X)
  - zapisu x, ki se nahaja na strani X notranjega pomnilnika, se vrednost ustrezno spremeni
  - če stran X z zapisom x še ni bila ažurirana (na disku), se na disku poišče še nerabljena stran in z njenim naslovom se v tekočem indeksu zamenja dosedanji naslov strani. Sicer ta nerabljena stran že obstaja za stran X

Obnavljanje s senčnimi stranmi..

Obnavljanje s senčnimi stranmi..

- V primeru transakcijske nesreče, ko se izda ukaz Pozabi, se tekoči indeks strani preprosto prepiše s senčnim indeksom in v PB ni sledu o ažuriranjih, ki so se v okviru transakcij izvedla
- Aktivirajo se senčne strani in transakcijski program lahko nadaljuje z izvajanjem naslednje transakcije

Obnavljanje s senčnimi stranmi..

- Ob uspešnem zaključku transakcije (ukaz Pomni), se izvede naslednje:
  1. vse ažurirane strani, ki se še nahajajo v pomnilniku, se izpišejo na disk,
  2. tekoči indeks strani se izpiše na prosto mesto na disk in to postane senčni indeks,
  3. naslov novega senčnega indeksa na disku se vpiše v startno datoteko.

Obnavljanje s senčnimi stranmi

- Če bi se med izvajanjem transakcije ali korakov 1-3 izvedla sistemska nesreča, bi SUPB ob ponovnem zagonu iz startne datoteke prebral naslov senčnega indeksa in na ta način prekinjena transakcija ne bi v PB pustila nobenih sledi
- Obnavljanje s senčnimi stranmi ščiti PB pred transakcijskimi in sistemskimi nesrečami
- Za zaščito pred diskovnimi nesrečami, je potrebno poskrbeti dodatno

Obnavljanje z dnevnikom in kopijo..

- To obnavljanje temelji na izdelavi kopije PB in izdelavi dnevnika:
  - s kopijo lahko PB obnovimo v veljavno stanje, ko se je izdelala kopija
  - v dnevnik se zapisujejo podatki, s katerimi je možno s kopijo obnovljeno PB obnoviti v zadnje veljavno stanje tik pred nesrečo
- Dnevnik vsebuje tudi podatke, s katerimi je možno ponovno izvesti tudi transakcije, ki so bile zaradi nesreče prekinjene

Obnavljanje z dnevnikom in kopijo..

- 2 vrsti obnavljanja PB z dnevnikom glede na čas izvedbe ažuriranja PB:
  - odloženo ažuriranje
  - sprotno ažuriranje
- **Odloženo ažuriranje**: vsa ažuriranja v okviru transakcije se najprej shranijo v dnevnik. Pri uspešnem zaključku transakcije se izvede dejansko ažuriranje PB
- **Sprotno ažuriranje**: Vsa ažuriranja se izvajajo v PB. V dnevnik se vpisujejo le podatki, ki so potrebni za morebitno obnavljanje PB

Obnavljanje z dnevnikom in kopijo

- Performanse:
  - Odloženo ažuriranje je učinkovitejše, če se v povprečju izvede več neuspešnih transakcij, ker v tem primeru ni treba spreminjati PB
  - Sprotno ažuriranje je učinkovitejše, če se v povprečju izvede več uspešnih transakcij

Obnavljanje z dnevnikom in kopijo - Odloženo ažuriranje..

- V dnevnik se beležijo:
  - zapisi, ki vsebujejo podatke, potrebne za izvedbo odloženega ažuriranja PB in
  - podatki za njeno morebitno obnavljanje
- Dnevnik je zaporedna datoteka, v katerem so zapisi urejeni po času njihovega nastanka
- Zapisi se v dnevnik vedno dodajajo na konec datoteke

Obnavljanje z dnevnikom in kopijo - Odloženo ažuriranje..

- Vsak zapis v dnevniku je opremljen:
  - z enolično oznako transakcije - Ti
  - s časom generiranja zapisa - t
- Zapis ob začetku izvajanja transakcije vsebuje:
  - oznako “Začetek”
  - oznako transakcijskega programa Pj
  - za vsak vhodni zapis pa par: (ImeZapisa, VrednostZapisa)

Obnavljanje z dnevnikom in kopijo - Odloženo ažuriranje..

- Zapis ob ukazu za ažuriranje vsebuje:
  - oznako NovaVrednost
  - vrsto operacije: Dodaj, Izbriši, Spremeni
  - par (NaslovZapisa, NovaVrednostZapisa)
- Zapis ob prehodu transakcije v stanje Uspešna transakcija:
  - oznako Pomni
- Zapis ob prehodu transakcije v stanje Neuspešna transakcija:
  - oznako Pozabi

Obnavljanje z dnevnikom in kopijo - Odloženo ažuriranje..

- Ker se sočasno lahko izvaja več transakcij so dnevniški zapisi teh transakcij lahko med seboj pomešani
- Ko se transakcija prične izvajati, SUPB v dnevnik zapiše zapis:

```
Dodaj(D, <Ti, t, Začetek, Pj, (V1, v1),...(Vn, vn)>)
```

Obnavljanje z dnevnikom in kopijo - Odloženo ažuriranje..

- Ob vsakem ažuriranju, ki se pojavi v okviru transakcije, se v dnevnik doda naslednji zapis:

  `Dodaj(D, <Ti, t, NovaVrednost, vrsta-operacije, (X, x)>)`

- Ob neuspešnem zaključku transakcije (ukaz Pozabi) se v dnevnik doda:

  `Dodaj(D, <Ti, t, Pozabi>)`

  in izbriše transakcijo iz liste aktivnih transakcij.

Obnavljanje z dnevnikom in kopijo - Odloženo ažuriranje..

- Ob uspešnem zaključku trans. se v dnevnik doda zapis:

  `Dodaj(D, <Ti, t, Pomni>)`

  poleg tega pa se izvede tudi uveljavitev sprememb v PB

- Uveljavitev sprememb v PB se izvede tako, da se bere dnevniške zapise z oznako NovaVrednost za transakcijo Ti (po času naprej), ter se izvedejo v njih zapisani ukazi operacij nad PB:

  `PoiščiPreberi(D, <Ti, t, NovaVrednost, vrsta-operacije, (X, x)>)`

Obnavljanje z dnevnikom in kopijo - Odloženo ažuriranje

- Ko so izvedene vse operacije za Ti, ki so zapisane v dnevniku, SUPB zbriše Ti iz liste aktivnih transakcij

Obnavljanje z dnevnikom in kopijo - Sprotno ažuriranje..

- V dnevnik se beležijo enake vrednosti, kot pri odloženem ažuriranju, dodatno pa še zapisi z vrednostmi pred ažuriranjem (stare vrednosti)
- Zapisovanje ob začetku izvajanja transakcije, njenem uspešnem ali neuspešnem zaključku, je enako kot pri odloženem ažuriranju

Obnavljanje z dnevnikom in kopijo - Sprotno ažuriranje..

- Spremenjen je postopek pri ukazih za ažuriranje. Ob vsakem ukazu za ažuriranje se v dnevnik najprej doda zapis s staro vrednostjo:

```
Dodaj(D, <Ti, t, StaraVrednost, vrsta-operacije, (X,
  x)>)
```

- Nato se izvede ažuriranje podatkovne baze (spreminjanje, brisanje, dodajanje)
- In nato še vpis v dnevnik:

```
Dodaj(D, <Ti, t NovaVrednost, vrsta-operacije, (X, x)>)
```

Obnavljanje z dnevnikom in kopijo - Sprotno ažuriranje

- Ob uspešnem zaključku transakcije se v dnevnik doda zapis Pomni, transakcija pa se izbriše iz liste aktivnih transakcij
- Pri neuspešnem zaključku se v dnevnik najprej doda zapis Pozabi, nato pa je ažuriranja v bazi potrebno razveljaviti
- Po razveljavitvi se transakcija zbriše iz liste aktivnih transakcij

Obnavljanje z dnevnikom in kopijo - Razveljavitev transakcije..

- Pri sprotnem ažuriranju se vse spremembe zapisov sproti vpisujejo v podatkovno bazo
- V primeru transakcijske ali sistemske nesreče, je za nadaljevanje dela najprej potrebno odstraniti iz PB vsa ažuriranja oz. nastale spremembe, ki so nastale s strani prekinjenih transakcij. Pravimo, da SUPB izvede razveljavitev

Obnavljanje z dnevnikom in kopijo - Razveljavitev transakcije..

- Pri sprotnem ažuriranju se razveljavitev izvede tako, da se iz dnevnika bere stare zapise (StaraVrednost) od zadnjega proti prvemu po padajočem času t:

```
PoiščiPreberi(D, <Ti, t, StaraVrednost, vrsta-operacije,
   (X, x)>)
```

Obnavljanje z dnevnikom in kopijo - Razveljavitev transakcije..

- Za vsak tak prebrani dnevniški zapis se izvede ažuriranje PB:
  - Če je bila operacija vrste Spremeni, se zapisu X priredi njegova stara vrednost: Spremeni (X,x=StaraVrednost)
  - Če je bila operacije vrste Dodaj, se dodani zapis izbriše: Izbriši (X)
  - Če je bila operacija vrste Izbriši, se izbrisani zapis doda nazaj v bazo: Dodaj (X,x=StaraVrednost)

Obnavljanje z dnevnikom in kopijo - Razveljavitev transakcije

- V primeru sistemske nesreče je potrebno razveljaviti ažuriranja večih transakcij
- V dnevniku so njihovi zapisi med seboj pomešani, vendar pa so urejeni po času, kar omogoča izvesti razveljavitev

Obnavljanje z dnevnikom in kopijo - Ponovitev ažuriranj transakcije

- Ažuriranja, ki jih izvede uspešna transakcija, je v primeru sistemskih in diskovnih podatkovnih nesreč treba ponoviti
- Vzroki:
  - ker so se ažuriranja izgubila ali pa
  - ni zanesljivo, da so se vsi datotečni vmesniki z ažuriranimi zapisi tudi izpisali na disk (sistemske nesreče)
- Po izvedeni ponovitvi se v PB zanesljivo nahajajo vsa ažuriranja, ki so jih izvedle uspešne transakcije

Obnavljanje z dnevnikom in kopijo - Obnavljanje po sistemskih in diskovnih nesrečah..

- Pri sistemskih podatkovnih nesrečah se prekine izvajanje trenutno aktivnih transakcij
- Ohrani se PB na disku, izgubijo pa se podatki v notranjem pomnilniku
- Po ponovnem zagonu je PB treba obnoviti, tako da najprej razdelimo transakcije na:
  - **prekinjene transakcije:** v dnevniku obstajajo njihovi zapisi
  - **uspešne transakcije:** tiste, za katere obstaja v dnevniku zapis z oznako “Pomni” in
  - **neuspešne transakcije:** tiste, za katere obstaja v dnevniku zapis z oznako “Pozabi”

Obnavljanje z dnevnikom in kopijo - Obnavljanje po sistemskih in diskovnih nesrečah

- Obnavljanje uspešnih transakcij:
  - Pri njih nismo prepričani, ali so se strani, ki so jih transakcije ažurirale, zanesljivo zapisale na disk
  - Zaradi tega je potrebno njihova ažuriranja ponoviti, tako pri uporabi sprotnega, kot odloženega ažuriranja
- Obnavljanje neuspešnih transakcij:
  - Pri teh transakcijah pri sprotnem ažuriranju nismo prepričani, ali so se njihova ažuriranja tudi že zares razveljavila v PB
  - Take transakcije je zato potrebno ponovno razveljaviti
- Obnavljanje prekinjenih transakcij:
  - Najprej izvedemo njihovo razveljavitev, nato jih vrnemo transakcijskim programom v ponovno izvajanje

Obnavljanje z dnevnikom in kopijo - Kontrolna točka..

- Pri sistemskih nesrečah se pojavi vprašanje: Za koliko časa nazaj je potrebno uspešne transakcije ponoviti in neuspešne razveljaviti?
- Odgovor: Ponoviti oz. razveljaviti je potrebno vse transakcije, katerih zapise najdemo v dnevniku

Obnavljanje z dnevnikom in kopijo - Kontrolna točka..

- PROBLEM: Dnevnik se začne pisati od trenutka, ko je bila izdelana kopija PB. To pomeni, da je v določenih primerih potrebno obnoviti in razveljaviti veliko število transakcij, kar pomeni dolgotrajno obnavljanje!!!
- REŠITEV: Čas obnavljanja skrajšamo z zahtevo po izpisu vseh datotečnih vmesnikov na disk. Tako smo prepričani, da so bile transakcije, ki so bile zaključene pred izpisom vmesnikov, zanesljivo uveljavljene ali razveljavljene v PB na disku
- Temu postopku pravimo Kontrolna točka

Obnavljanje z dnevnikom in kopijo - Kontrolna točka..

- Kontrolna točka se izvede v odvisnosti od števila transakcij na časovno enoto
- Ob sistemski nesreči je potrebno razveljaviti oz. ponoviti le transakcije, ki so bile aktivne v času izdelave kontrolne točke, ali so se pričele izvajati kasneje

Obnavljanje z dnevnikom in kopijo - Kontrolna točka

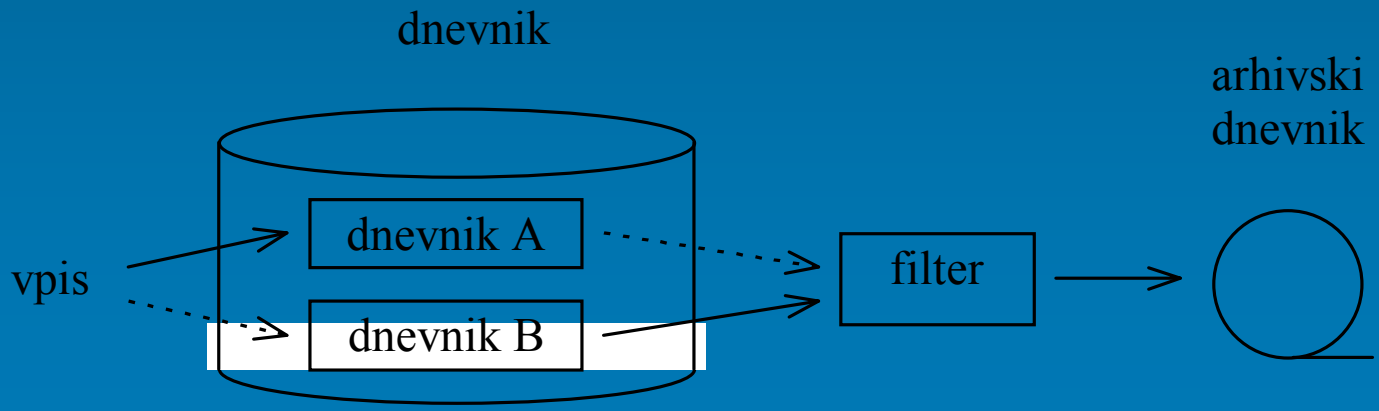
- Pri kontrolni točki se izvedejo naslednje operacije:
  1. prekine se izvajanje novih ukazov transakcijskih programov, dokončajo se vse razveljavitve in uveljavitve transakcij,
  2. izvede se izsiljeni izpis vseh datotečnih vmesnikov dnevnika na disk,
  3. izvede se izsiljeni izpis vseh datotečnih vmesnikov PB na disk,
  4. v dnevnik se doda zapis “Kontrolna točka” in izvede izsiljen izpis datotečnega vmesnika dnevnika na disk,
  5. v startno datoteko PB se zapiše naslov zapisa “Kontrolna točka” v dnevniku,
  6. nadaljuje se izvajanje ukazov transakcijskih programov.

Obnavljanje z dnevnikom in kopijo - Dnevnik..

- Dnevnik in kopija PB predstavljata redundantne podatke, ki omogočajo obnovitev po sistemskih in diskovnih nesrečah
- Kopija in dnevnik se ne shranjujeta v fizično istem zunanjem pomnilniku, kot PB. S tem se zmanjša možnost uničenja vseh podatkov
- Dnevnik se uporablja tudi za tekoče razveljavljanje (sprotno ažuriranje) oz. uveljavljanje (odloženo ažuriranje). To mora biti izvedeno v čimkrajšem času

Obnavljanje z dnevnikom in kopijo - Dnevnik..

- Najpogostejša varianta realizacije dnevnika:
  - dve dnevniški datoteki na disku in ena na magnetnem traku – arhivska
  - Ob izvajanju transakcij se dnevniški zapisi vpisujejo v eno izmed datotek na disku (npr.: v dnevnik A)
  - Ko se dnevnik A napolni približno 90%, se izvrši preklop vpisovanja
  - Vsi dnevniški vpisi, ki se pričnо po preklopu, se vpisujejo v dnevnik B, zapisi transakcij, ki so se začele pred preklopom, pa se še naprej zapisujejo v dnevnik A.



Obnavljanje z dnevnikom in kopijo - Dnevnik

- Ko se zaključijo vse transakcije, ki se vpisujejo v dnevnik A, se izvede kontrolna točka. S tem so se vse transakcije v A zaključile pred kontrolno točko
- Edina nesreča, pri kateri bi še potrebovali dnevniške zapise iz dnevnika A, je diskovna nesreča
- Ker se pri ponovitvi transakcijksih ažuriranj rabijo le zapisi z oznako “NovaVrednost”, se le ti pri arhiviranju dnevnika A prepišejo v arhivski dnevnik. Temu pravimo filtriranje
- Arhivski dnevnik vsebuje le še tiste zapise, ki bi jih utegnili rabiti za obnavljanje, kar dosti zmanjša majhna velikost arhivskega dnevnika
- Ko se napolni dnevnik B, se postopek preklopa in arhiviranja ponovi

Poglavje 10

Sočasni dostop

Povzeto po [4]

Sočasni dostop do PB..

- Sodobni OS omogočajo izvajanje več programov hkrati
- Poleg SUPB se tako lahko izvaja več uporabniških programov, ki dostopajo do PB hkrati
- Transakcije, ki jih izvajajo uporabniški programi se izvajajo prepletajoče, zato tudi SUPB izvaja ukaze prepletajoče
- Prepletajoče izvajanje transakcij pa skriva dve pasti:
  - podatkovna baza lahko zaide v nekonsistentno stanje,
  - rezultati povpraševanj v podatkovni bazi so lahko napačni

Sočasni dostop do PB..

- Prepletajoča izvedba transakcij lahko bazo pusti v nekonsistentnem stanju, čeprav sta obe transakciji pravilno izvedli svoja ažuriranja. Primera:
  - "**izgubljeno ažuriranje**": na osnovi istega prebranega zapisa se izvede ažuriranje istega zapisa s strani dveh transakcij, kar pomeni, da obvelja samo zadnje ažuriranje,
  - **branje "neobstoječega zapisa":** predpostavimo, da neka transakcija prebere zapis, ki ga je ravnokar ažurirala druga transakcija, ki se je takoj zatem ponesrečila in zato razveljavila. Prva transakcija je torej prebrala zapis (oz. njegovo stanje), ki ni nikoli veljavno obstajal v PB!!!

Sočasni dostop do PB..

- Naloga nadzora nad sočasno uporabo PB (*concurrency control*) je:
  - ohraniti podatkovno bazo v konsistentnem stanju,
  - dopustiti čimvečjo sočasnost izvajanja transakcij
- V ta namen se uporabljata dve tehniki:
  - zaseganje zapisov (*locking*),
  - časovno označevanje (*timestamping*)

Sočasni dostop do PB

- Sočasno izvajanje transakcij bomo obravnavali na primeru centralizirane PB, ki jo upravlja en sam SUPB
- Do problemov sočasnega izvajanja transakcij prihaja tudi pri porazdeljenih PB. Ker se tam transakcije zares izvajajo sočasno, so problemi sočasnosti izvajanja transakcij še težje rešljivi. Niso pa nerešljivi!

Sočasno izvajanje transakcij..

- Z vidika nadzora nad sočasnim izvajanjem transakcij so pomembni predvsem tisti ukazi, ki spreminjajo vsebino PB
- Omejili se bomo na naslednje ukaze:
  - Pomni,
  - Pozabi,
  - PoiščiPreberi(X, x),
  - Ažuriraj(X, x) (Ta ukaz zamenjuje oz. pokriva ukaze: Dodaj(X, x), Spremeni(X, x), Izbriši(X)).

Sočasno izvajanje transakcij..

- Zaporedje izvajanja ukazov v okviru (več) transakcij se imenuje razpored (*schedule*)
- Zaporeden razpored: pri sočasnem izvajanju transakcij se najprej izvedejo vsi ukazi ene transakcije, nato pa vsi ukazi druge transakcije (primeri na naslednjih straneh)

Sočasno izvajanje transakcij..

Zamislimo si banko, ki vodi račune svojih komitentov. Komitenti med seboj kupčujejo, zato je potrebno pri vsaki kupčiji izvesti transakcijo - prenos sredstev z računa enega komitenta na račun drugega komitenta. Po izvedbi ene ali večih transakcij mora vsota sredstev na vseh računih skupaj ostati nespremenjena.

Omejimo se na dve transakciji T0 in T1. Izvedeta se lahko po dveh različnih zaporednih razporedih, in po celi vrsti izmeničnih razporedov. Pri tem se lahko posamezna transakcija zaključi uspešno (z ukazom Pomni) ali pa neuspešno (z ukazom Pozabi), pri čemer pa se vsa njena že izvedena ažuriranja v podatkovni bazi razveljavijo. Predpostavili bomo tudi, da se ažuriranja izvajajo kot sprotna ažuriranja.

T0	*T1*
PoiščiPreberi(R1,a) *a := a - 10* *Ažuriraj(R1,a)* *PoiščiPreberi(R2,b)* *b := b + 10* *Ažuriraj(R2,b)* *Pomni*	
	PoiščiPreberi(R2,c) *c := c - 20* *Ažuriraj(R2,c)* *PoiščiPreberi(R3,d)* *d := d + 20* *Ažuriraj(R3,d)* *Pomni*

T0	*T1*
	PoiščiPreberi(R2,c) *c := c - 20* *Ažuriraj(R2,c)* *PoiščiPreberi(R3,d)* *d := d + 20* *Ažuriraj(R3,d)* *Pomni*
PoiščiPreberi(R1,a) *a := a - 10* *Ažuriraj(R1,a)* *PoiščiPreberi(R2,b)* *b := b + 10* *Ažuriraj(R2,b)* *Pomni*	

Sočasno izvajanje transakcij..

- Ukaz transakcije T1 - PoiščiPreberi(R3,d) se ni uspešno izvršil
- Transakcijski program je izdal ukaz Pozabi, SUPB pa je nato transakcijo T1 razveljavil in s tem obnovil podatek R2 v prvotno stanje
- Stanje v podatkovni bazi po izvedenih transakcijah je R1 = 90, R2 = 210 in R3 = 300. Merilo uspešnosti je ohranjanje vsote 600
- Razpored je zaporeden

Zaporedni razpored B2

T0	*T1*
	PoiščiPreberi(R2,c) *c := c - 20* *Ažuriraj(R2,c)* *PoiščiPreberi(R3,d)* *Pozabi*
PoiščiPreberi(R1,a) *a := a - 10* *Ažuriraj(R1,a)* *PoiščiPreberi(R2,b)* *b := b + 10* *Ažuriraj(R2,b)* *Pomni*	

Sočasno izvajanje transakcij..

- Prepletajoč razpored: med ukazi ene transakcije se izvajajo tudi ukazi drugih transakcij.

TA IZMENIČNI RAZPORED JE OK

T0	T1
PoiščiPreberi(R1,a) *a := a - 10* *Ažuriraj(R1,a)*	
	PoiščiPreberi(R2,c) *c := c - 20* *Ažuriraj(R2,c)*
PoiščiPreberi(R2,b) *b := b + 10* *Ažuriraj(R2,b)* *Pomni*	
	PoiščiPreberi(R3,d) *d := d + 20* *Ažuriraj(R3,d)* *Pomni*

BRANJE NEOBSTOJEČEGA ZAPISA

T0	T1
PoiščiPreberi(R1,a) *a := a - 10* *Ažuriraj(R1,a)*	
	PoiščiPreberi(R2,c) *c := c - 20* *Ažuriraj(R2,c)*
PoiščiPreberi(R2,b) *b := b + 10* *Ažuriraj(R2,b)* *Pomni*	
	PoiščiPreberi(R3,d) *Pozabi*

IZGUBLJENO AŽURIRANJE

T0	T1
PoiščiPreberi(R1,a) *a := a - 10* *Ažuriraj(R1,a)* *PoiščiPreberi(R2,b)* *b := b + 10*	
	PoiščiPreberi(R2,c) *c := c - 20* *Ažuriraj(R2,c)*
Ažuriraj(R2,b) *Pomni*	
	PoiščiPreberi(R3,d) *d := d + 20* *Ažuriraj(R3,d)* *Pomni*

Sočasno izvajanje transakcij..

- Transakcije so med seboj neodvisne, če se lahko izvajajo v poljubnem vrstnem redu: zaporedno ali prepletajoče, pri čemer se kakšne izmed transakcij lahko zaključijo tudi neuspešno
- Velja naslednje: Če vsaka izmed transakcij ohranja konsistentnost podatkovne baze, potem jo ohranja tudi vsak njihov zaporedni razpored

Sočasno izvajanje transakcij..

- Nekateri prepletajoči razporedi ne ohranjajo konsistentnosti PB
- Tisti prepletajoči razpored, ki učinkuje na PB enako kot kak izmed zaporednih razporedov, se imenuje zaporedniški razpored
- Velja naslednje: Če vsaka izmed transakcij ohranja konsistentnost podatkovne baze, potem jo ohranja tudi vsak njihov zaporedniški razpored

Sočasno izvajanje transakcij..

- Nadzor nad sočasno uporabo PB mora zagotoviti prepletajoče izvajanje transakcij po enem izmed možnih zaporedniških razporedov
- Pri sočasni uporabi PB se včasih pripeti, da se konsistentnost podatkov ohrani, nekonsistentni pa so rezultati povpraševanja, kar kaže primer na desni
- **RAZLOG**: vsota po vseh računih se je ohranila in je enaka 600, transakcija T2 pa jo je izračunala napačno: v = 610. Razlog je v tem, ker je T2 prebrala podatek R1 pred ažuriranjem, podatek R2 pa po ažuriranju

T0	*T2*
	v := 0
PoiščiPreberi(R1,a) *a := a - 10*	
	PoiščiPreberi(R1,s) *v := v + s*
Ažuriraj(R1,a) *PoiščiPreberi(R2,b)* *b := b + 10* *Ažuriraj(R2,b)* *Pomni*	
	PoiščiPreberi(R2,s) *v := v + s* *PoiščiPreberi(R3,s)* *v := v + s* *Pomni*

Sočasno izvajanje transakcij

- Rezultat povpraševanja transakcije T2 iz primera lahko ocenimo kot približno točen ali pa kot napačen
- Če so zahtevani popolnoma natančni rezultati, mora nadzor nad sočasno uporabo podatkovne baze zagotoviti tudi konsistentnost povpraševanj v podatkovni bazi

Zaseganje zapisov..

- Možna rešitev predstavljenih problemov: možnost, da si transakcijski program pridobi izključno pravico dostopa do zapisov, do katerih transakcija dostopa
- S tem se prepreči vmešavanje sočasnih transakcij v njen postopek ažuriranja

Zaseganje zapisov..

- Pravilo po katerem se mora ravnati SUPB, da zaščiti konsistentnost PB, je poimenovano kot “Piščevo pravilo”:
  - Ko se v okviru transakcij izvaja ažuriranje dela PB, mora biti celotno zaporedje operacij ažuriranja zaščiteno pred vmešavanjem s strani transakcij, ki žele sočasno ažurirati isti del PB

Zaseganje zapisov..

- Pravilo po katerem se mora ravnati SUPB, da zaščiti konsistentnost rezultatov povpraševanja, je poimenovano kot “Bralčevo pravilo”:
  - Ko se v okviru transakcije izvaja le povpraševanje v PB, potem je lahko (ali pa tudi ne) celotno zaporedje operacij povpraševanja zaščiteno pred vmešavanjem s strani transakcij, ki bi žele ažurirati isti del PB
- Ali želi “bralec” zaščito ali ne, je odvisno od tega, ali želi priti do točnih ali samo do približnih rezultatov
- Zaščita pred vmešavanjem sočasnih transakcij se izvede s pomočjo zaseganja zapisov

Zaseganje zapisov..

- Zaradi tega se ukazom za upravljanje s podatki dodajo še ukazi za zaseganje:
  - E (exclusive)-zaseži(X),
  - D (shared)-zaseži(X),
  - Sprosti(X).
- Način delovanja: transakcijski program izda zahtevo po zaseženju. Če je zaseženje mogoče, to SUPB takoj odobri. Če to ni možno, mora transakcija na odobritev počakati

Zaseganje zapisov..

- 2 pravili glede zaseganja podatkov:
  - ekskluzivno zaseženi podatek se ne more še dodatno zaseči, niti ekskluzivno, niti delno,
  - deljeno zaseženi podatek se lahko dodatno deljeno zaseže, ne more pa se zaseči ekskluzivno.

Zaseganje zapisov

- Kompatibilnostna matrika o tem katera zaseženja podatkov so možna, glede na trenutno zaseženje teh podatkov:

Zahteva po zaseženju

Trenutno zaseženje	E	D	-
E	ne	ne	da
D	ne	da	da
-	da	da	da

Zaseganje zapisov - PXC, PSC..

- Pri izvajanju transakcij je potrebno pri zaseganju podatkov upoštevati določena pravila, ki jim pravimo protokol
- Protokol zagotavlja, da se prepletajoče izvajanje transakcij izvaja po enem izmed zaporedniških razporedov
- 2 protokola:
  - PXC (Protocol eXclusive Commit): temelji na ekskluzivnem zaseganju podatkov,
  - PSC (Protocol Shared Commit): temelji na ekskluzivnem in deljenem zaseganju

Zaseganje zapisov - PXC, PSC..

- Pravila, ki jih uvaja PXC protokol:
  - transakcija, ki želi podatek ažurirati, ga mora najprej ekskluzivno zaseči,
  - če zahteva po zaseženju ne more biti takoj odobrena, preide transakcija v stanje čakanja, njeno izvajanje se nadaljuje po odobritvi zaseženja,
  - vsa zaseženja se smejo sprostiti šele po zaključku transakcije (uspešnem ali neuspešnem),
  - transakcija, ki želi le prebrati podatek in ji ni mar za sočasno ažuriranje tega podatka s strani kake druge transakcije, ga sme prebrati ne glede na to, ali je podatek zasežen ali ne

Zaseganje zapisov - PXC, PSC..

- Dodatno vsebuje protokol PSC še naslednje pravilo:
  - Transakcija, ki želi ekskluzivno zaseči podatek, mora imeti pred tem odobreno njegovo deljeno zaseženje
- Razlike med protokoloma so v obsegu dopuščenega sočasnega izvajanja transakcij:
  - PSC dopušča sočasno izvajanje dveh zgolj povpraševalnih transakcij, ki zahtevata zaščito pred ažuriranjem; PXC pa ne,
  - PSC veliko bolj dopušča nastop mrtve zanke (medsebojno blokiranje transakcij)

Zaseganje zapisov - PXC, PSC..

- Da se protokol za zaseganje podatkov lahko izvaja, mora SUPB za podatke voditi evidenco o tem:
  - ali podatek je ali ni zasežen,
  - kako je zasežen,
  - kdo vse ga je zasegel,
  - kdo vse ga želi zaseči in na kakšen način.
- Zato se za vsak podatek vzdržujeta 2 listi:
  - lista odobrenih zaseženj, ki vsebuje pare: (Oznaka transakcije, Vrsta Odobrenega Zaseženja),
  - lista zahtevanih zaseženj, ki vsebuje pare: (Oznaka transakcije, Vrsta Zahtevanega Zaseženja).

Zaseganje zapisov - PXC, PSC

- SUPB mora med izvajanjem transakcij ves čas vzdrževati obe listi
- Do sprememb na listah prihaja:
  - pri sprejetju zahteve po zaseženju,
  - pri odobritvi zaseženja,
  - ob zaključku transkacije

Zaseganje zapisov - Objekti zaseženja..

- Objekti zaseženja so lahko različni:
  - logični,
  - fizični.
- Logični objekti zaseženja:
  - relacije,
  - n-terice v relacijah,...
- Fizični objekti zaseženja:
  - celotna fizična podatkovna baza,
  - tabele,
  - fizični bloki oz. strani,
  - fizični zapisi v tabeli.

Zaseganje zapisov – Objekti zaseženja

- Granulacija objektov zaseženja vpliva na:
  - obseg sočasnoti pri izvajanju transakcij,
  - obseg podatkov o odobrenih in zahtevanih zaseženjih,
  - stopnjo dodatne obremenitve SUPB z izvajanjem nadzora nad zaseženji.
- Ukaza “pomni” in “pozabi” pomenita tudi sprostitev zaseženj podatkov
- Protokol PXC uvaja še ukaz:
  - E-PoišČiPreberi(X,x),
- Protokol PSC pa ukaza:
  - D-PoišČiPreberi(X,x) in
  - E-Ažurirarj(X,x). Najprej zahteva razširitev zaseženja, potem izvede ažuriranje

Zaseganje zapisov – Primeri..

Ker hoče T0 zaseči podatek, ki ga je pred tem zasegla že T1, mora čakati.

T0	T1
E-PoiščiPreberi(R1,a) *a := a - 10* *Ažuriraj(R1,a)*	
	E-PoiščiPreberi(R2,c) *c := c - 20* *Ažuriraj(R2,c)*
E-PoiščiPreberi(R2,b) *(čakanje na odobritev)*	
	E-PoiščiPreberi(R3,d) *d := d + 20* *Ažuriraj(R3,d)* *Pomni*
(zaseženje odobreno) *(ukaz se izvede)* *b := b + 10* *Ažuriraj(R2,b)* *Pomni*	

Tukaj bi se izvedlo branje neobstoječega podatka. Z zaseganjem je razpored zaporedniški.

T0	T1
E-PoiščiPreberi(R1,a) *a := a - 10* *Ažuriraj(R1,a)*	
	E-PoiščiPreberi(R2,c) *c := c - 20* *Ažuriraj(R2,c)*
E-PoiščiPreberi(R2,b) *(čakanje na odobritev)*	
	E-PoiščiPreberi(R3,d) *Pozabi*
(zaseženje odobreno) *(ukaz se izvede)* *b := b + 10* *Ažuriraj(R2,b)* *Pomni*	

Zaseganje zapisov – Primeri..

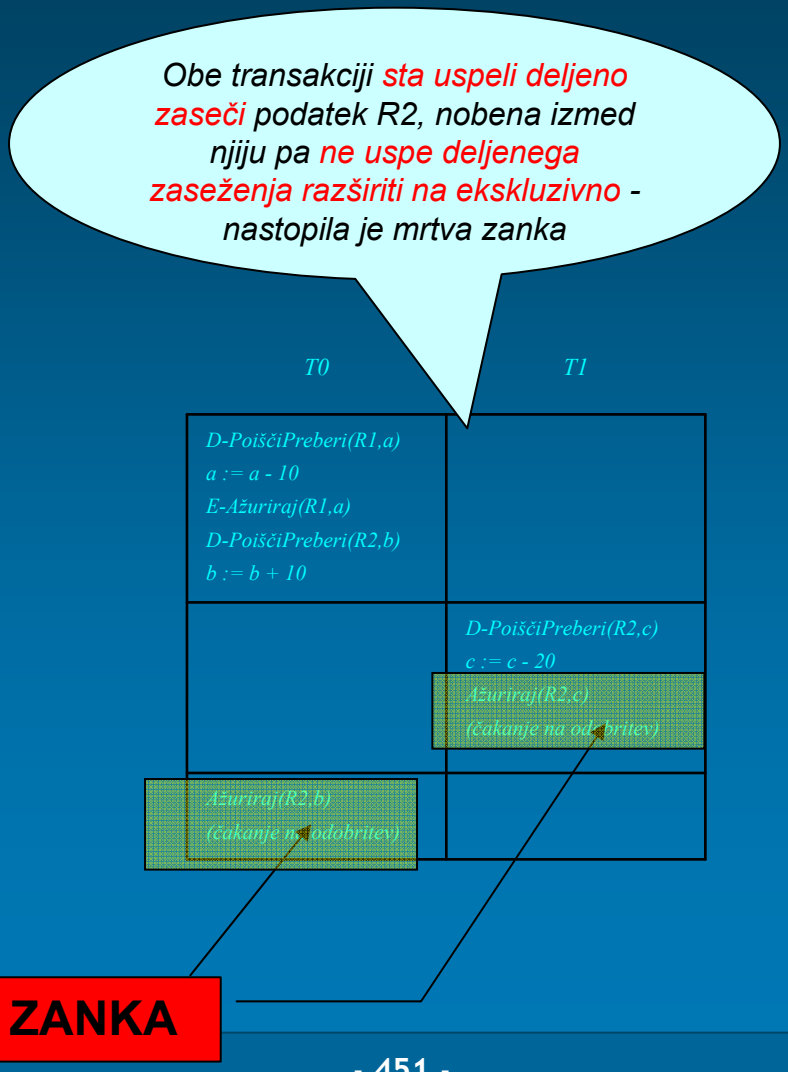
Ažuriranje se ne izgubi

T0	T1
E-PoiščiPreberi(R1,a) *a := a - 10* *Ažuriraj(R1,a)* *E-PoiščiPreberi(R2,b)* *b := b + 10*	
	E-PoiščiPreberi(R2,c) *(čakanje na odobritev)*
Ažuriraj(R2,b) *Pomni*	
	(zaseženje odobreno) *(ukaz se izvede)* *c := c - 20* *Ažuriraj(R2,c)* *E-PoiščiPreberi(R3,d)* *d := d + 20* *Ažuriraj(R3,d)* *Pomni*

IZGUBLJENO AŽURIRANJE

T0	T1
PoiščiPreberi(R1,a) *a := a - 10* *Ažuriraj(R1,a)* *PoiščiPreberi(R2,b)* *b := b + 10*	
	PoiščiPreberi(R2,c) *c := c - 20* *Ažuriraj(R2,c)*
Ažuriraj(R2,b) *Pomni*	
	PoiščiPreberi(R3,d) *d := d + 20* *Ažuriraj(R3,d)* *Pomni*

Zaseganje zapisov – Primeri

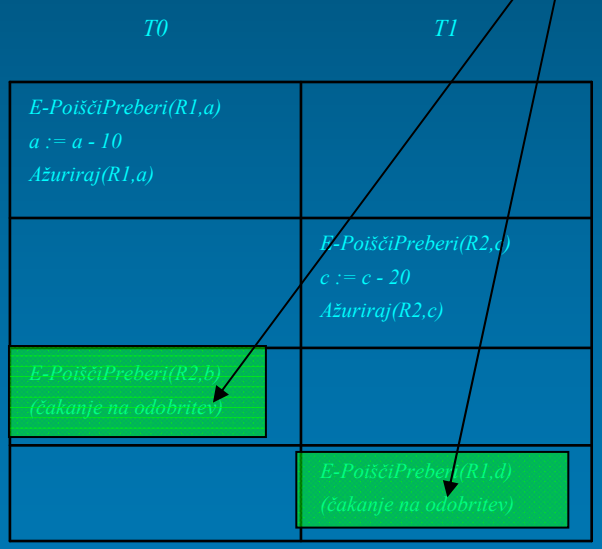


Mrtva zanka – Uvod..

- Pri sočasnem izvajanju transakcij se pojavljata naslednja problema:
  - livelock: Posamezno transakcijo lahko pri odobritvi zaseženja določenega podatka prehitevajo vse ostale transakcije in se s tem njeno čakanje raztegne na nedoločen čas (problem je rešljiv z algoritmi za dodeljevanje zaseženj),
  - mrtva zanka (deadlock): ko dve (ali več transakcij) zaseže vsaka svoj podatek, vsaka od njiju pa želi še podatek, ki ga je že zasegla tekmica. Zaradi tega transakciji čakata druga na drugo, in ker se po protokolu sprostijo zaseženi podatki le ob zaključku transakcije, ni čakanja nikoli konec.

Mrtva zanka – Uvod..

- Primer nastopa mrtve zanke, ki lahko nastopi tudi pri PXC:



Vsaka transakcija hoče zaseči podatek, ki ga je zasegla že druga transakcija.

Mrtva zanka – Uvod

- Problem mrtve zanke se razreši na dva načina:
  - preprečimo, da se mrtva zanka sploh lahko pojavi, kar rešujemo z enim ali več od naslednjih prijemov: urnikom izvajanja transakcij, vnaprejšnja zahteva po zaseženjih zapisov, ureditev objektov zaseganja, odloča transakcijski program in prekinitev in ponovno izvajanje transakcij
  - odpravimo mrtvo zanko potem, ko je ta že nastopila

Preprečevanje nastopa mrtve zanke..

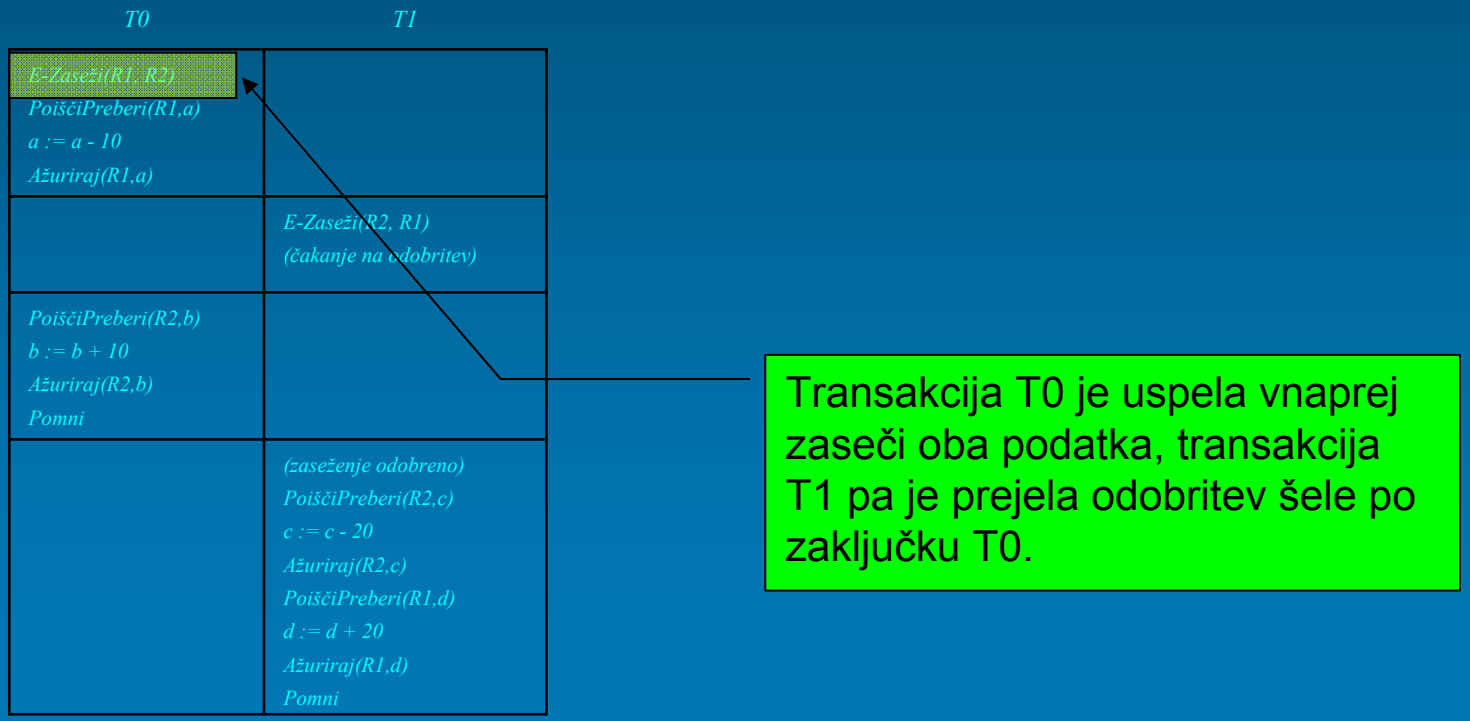
- **Urnik izvajanja transakcij**:
  - Z izvajanjem transakcij po urniku ne dopustimo sočasnega izvajanja takih transakcij oziroma transakcijskih programov, ki bi utegnili imeti konfliktne podatkovne zahteve
  - V tem primeru tudi ni potrebno izvajati zaseganja podatkov. Pravimo, da se na ta način poveča "propustnost" podatkovne baze
  - Ker praviloma ni vnaprej znano katere podatke bo transakcija ažurirala, se možna sočasnost (prepletajoči način) izvajanja transakcij zelo omeji
  - V najneugodnejšem primeru je urnik transakcij ekvivalenten zaseženju celotne podatkovne baze s strani posamezne transakcije

Preprečevanje nastopa mrtve zanke..

- **Vnaprejšnja zahteva po zaseženju podatkov:**
  - Transakcija postavi pred svojim prvim ažuriranjem zahtevo po zaseženjih vseh podatkov, ki jih namerava ažurirati:
    - Če je možno vsa zahtevana zaseženja takoj odobriti, jih tudi SUPB odobri in transakcija se lahko nemoteno izvaja vse do svojega zaključka.
    - Če kakšnega izmed zaseženj ni možno odobriti, se ji ne odobri nobenega zaseženja in transakcija preide v stanje čakanja.
  - Pri taki rešitvi ne sme biti izbor podatka, ki ga bo transakcija ažurirala, pogojen z uspehom ali neuspehom kakšnega predhodnega ažuriranja, zato ker v takem primeru transakcija:
    - bodisi ne more vedeti, katere podatke naj zaseže,
    - bodisi zaseže preventivno bistveno večje število podatkov, kot bi bilo potrebno, zato se zmanjšuje možnost sočasnega izvajanja drugih transakcij. (primer na naslednji strani)

Preprečevanje nastopa mrtve zanke..

- Primer vnaprejšnje zahteve po zaseženju podatkov:

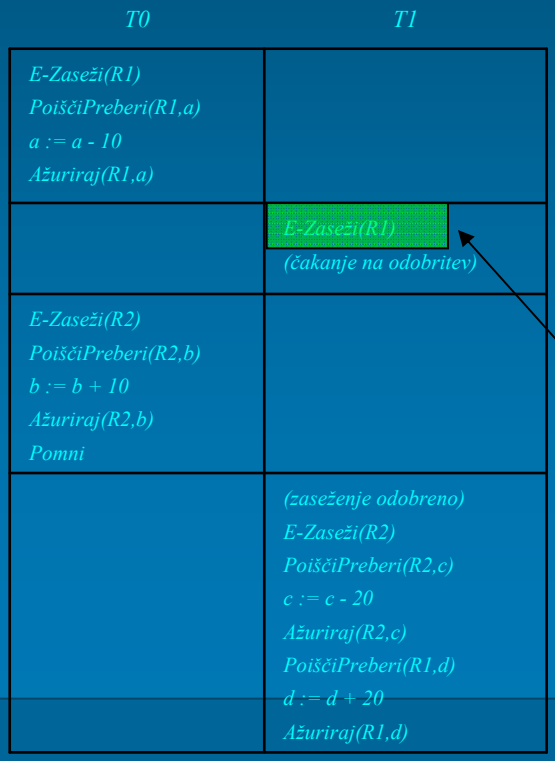


Preprečevanje nastopa mrtve zanke..

- **Ureditev objektov zaseganja**:
  - Objekte zaseganja, ki so lahko zapisi, strani zapisov, n-terice, tabele, se sme zaseči samo po določenem vrstnem redu, ki mora biti znan vnaprej vsem transakcijskim programom
  - Če je objektov zaseganja veliko (npr. zapisi), je določanje njihovega vrstnega reda in tudi nadzor nad pravilnim vrstnim redom zaseganja dokaj zamudno opravilo
  - poleg tega pa je potrebno vnaprej poznati podatke, ki se bodo v okviru transakcije ažurirali. (primer na naslednji strani)

Preprečevanje nastopa mrtve zanke..

- Primer ureditve objektov zaseganja:
  - Predpostavimo, da je potrebno zaseči najprej R1 in šele nato R2



Ker transakcija T1 ni uspela takoj zaseči podatka, je prešla v čakanje. Obe transakciji sta zasegali podatke po enakem vrstnem redu: R1, R2.

Preprečevanje nastopa mrtve zanke..

- Odloča transakcijski program:
  - Če zahteva po odobritvi zaseženja podatka v okviru transakcije ni takoj odobrena, odloči transakcijski program, kako naprej:
    - izvajanje transakcije lahko program takoj prekine in ko jo SUPB razveljavi, prične z njenim ponovnim izvajanjem; lahko pa jo uvrsti na začetek čakalne vrste in prične izvajati transakcijo, ki je naslednja na vrsti;
    - v presledkih lahko izvede nekaj poskusov zaseganja podatka in če ne uspe, prekine izvajanje transakcije.

Preprečevanje nastopa mrtve zanke..

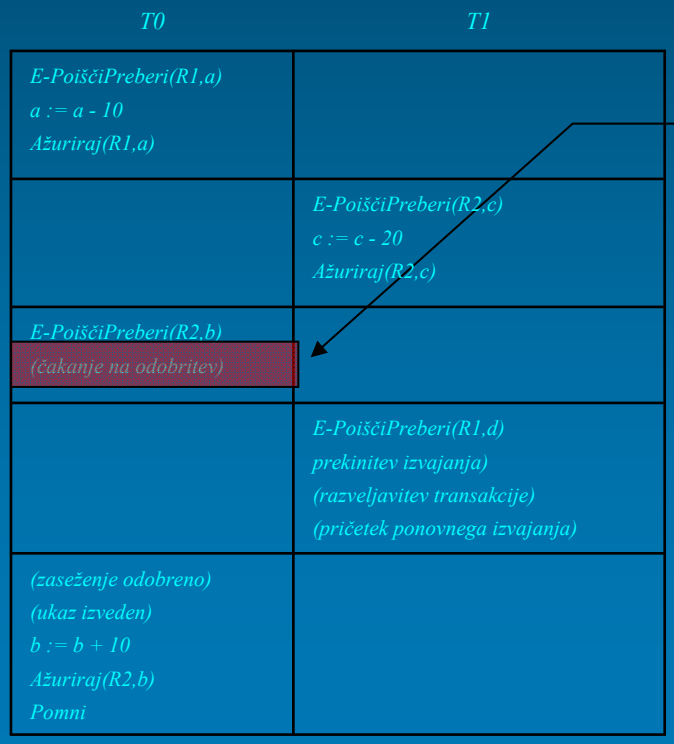
- **Prekinitev in ponovno izvajanje transakcij**:
  - Uporaba protokolov:
    - **Čakaj ali izdihni** (Wait Die) in
    - **Rani ali čakaj** (Wound-Wait).
- Vsaki izmed transakcij pripiše SUPB ob njenem pričetku časovno oznako - njen startni čas. Na ta način je za poljubni dve transakciji možno ugotoviti, katera je "starejša" in katera "mlajša"
- Starejša transakcija je tista, ki se že dlje časa izvaja in je zato njen startni čas manjši

Preprečevanje nastopa mrtve zanke..

- Ko transakcija TA zahteva zaseženje podatka, ki je že zasežen s strani transakcije TB, in se njeni zahtevi zaradi nekompatibilnosti zaseženj ne da takoj ugoditi, se po protokolu “Čakaj ali izdihni” izvede naslednje:
  - če je transakcija TA starejša od TB, preide TA v stanje čakanja na odobritev,
  - če je mlajša, pa se njeno izvajanje prekine, transakcija se razveljavi in posreduje transakcijskemu programu v ponovno izvajanje

Preprečevanje nastopa mrtve zanke..

- Razpored ažuriranj pri uporabi protokola “Čakaj ali izdihni”:



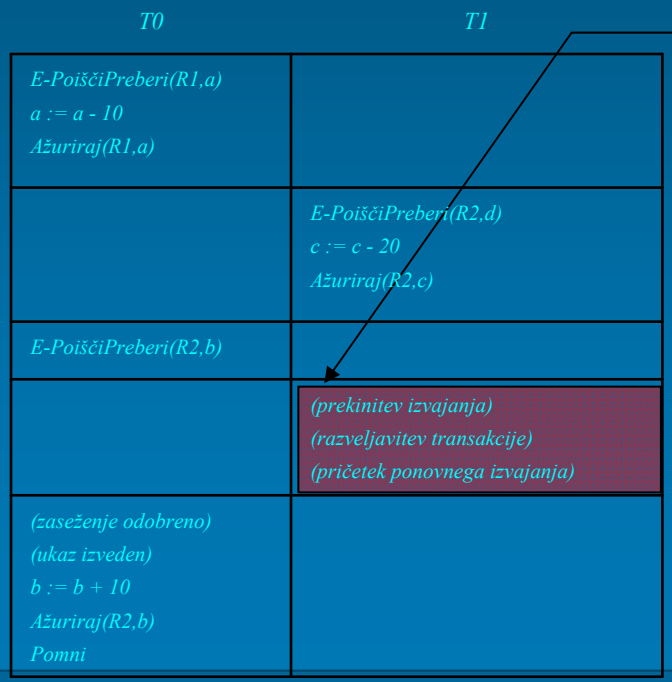
Protokol Čakaj ali izdihni prepreči nastop mrtve zanke. T0 je starejša, zato preide v stanje čakanja na odobritev.

Preprečevanje nastopa mrtve zanke..

- Ko transakcija TA zahteva zaseženje podatka, ki je že zasežen s strani transakcije TB, in se njeni zahtevi zaradi nekompatibilnosti zaseženj ne da takoj ugoditi, se po protokolu “Rani ali čakaj” izvede naslednje:
  - če je transakcija TA starejša od TB, se prekine transakcija TB, razveljavi in vrne v ponovno izvajanje. Po razveljavitvi TB se transakciji TA odobri zaseženje podatka,
  - če je TA mlajša, pa preide v stanje čakanja.

Preprečevanje nastopa mrtve zanke

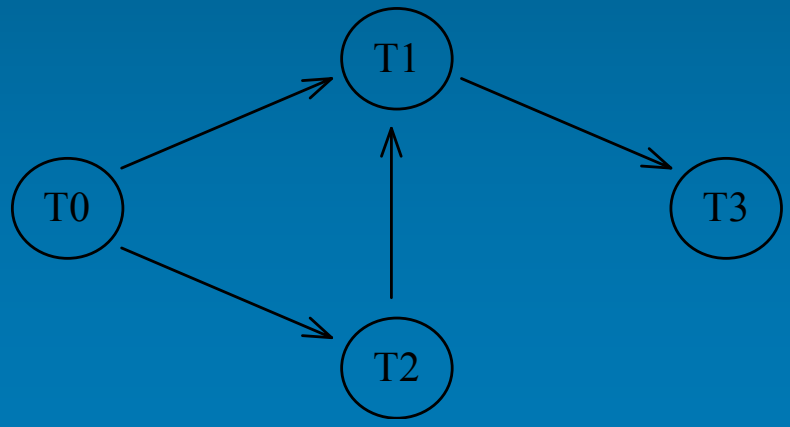
- Razpored ažuriranj pri uporabi protokola “Rani ali čakaj”:



Isti problem kot v prejšnjem primeru se s protokolom Rani ali čakaj izvede na naslednji način. Ker je T0 starejša od T1, se T1 prekine, razveljavi in vrne v ponovno izvajanje.

Odpravljanje mrtve zanke..

- Mrtvo zanko je potrebno najprej odkriti, nato pa pristopimo k njenemu razreševanju
- V ta namen lahko uporabimo čakalni graf
- Primer čakalnega grafa brez cikla:

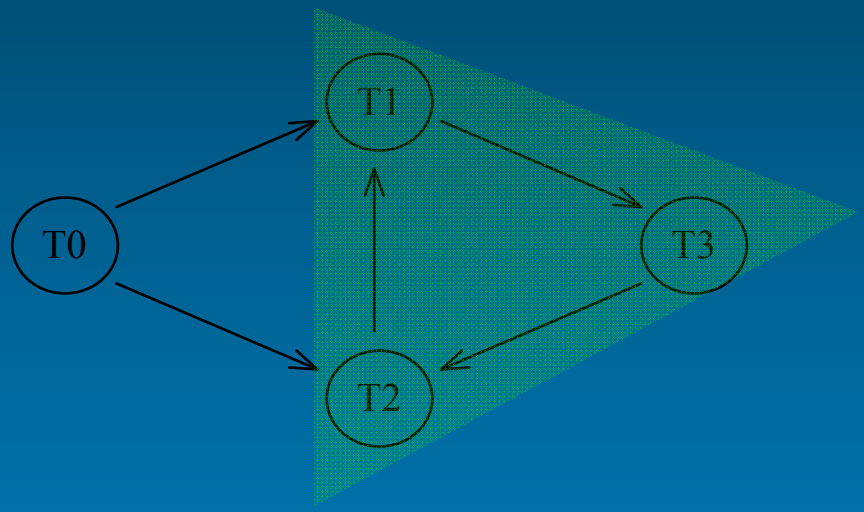


Odpravljanje mrtve zanke..

- **Čakalni graf** je usmerjni graf **G = < V, P >**
- **“V”** predstavlja množico vozlišč, v kateri vsak predstavlja po eno aktivno transakcijo T_i
- “P” predstavlja množico usmerjenih povezav (T_i, T_j), ki predstavljajo čakanje transakcije T_i na odobritev zaseženja podatka, ki ga ima zaseženega transakcija T_j
- Mrtva zanka se v grafu kaže kot cikel. V njem se lahko nahajata dve ali več transakcij, ki druga drugo čakajo na sprostitev zaseženj

Odpravljanje mrtve zanke..

- Primer mrtve zanke, v kateri se nahajajo tri transakcije: T1, T2, T3.



- Za odpravljanje mrtve zanke je zadolžen SUPB, ki ves čas vzdržuje čakalni graf
- Spremembe v grafu nastopijo:
  - ob neodobrenih zahtevah po zaseženjih in
  - ob zaključkih transakcij.

Odpravljanje mrtve zanke..

- Ko SUPB zasledi v grafu cikel, se mora lotiti odpravljanja mrtve zanke
- Odpravi jo tako, da eno izmed transakcij v mrtvi zanki izbere za žrtev in prekine njeno izvajanje
- S tem, ko se ena izmed transakcij v mrtvi zanki prekine, izgine iz čakalnega grafa tudi njeno vozlišče in vse pripadajoče povezave, kar pomeni, da cikla več
- Po razveljavitvi ažuriranj prekinjene transakcije, eni izmed preostalih transakcij odobri zahtevano zaseženje, prekinjena transakcija pa se vrne v ponovno izvajanje

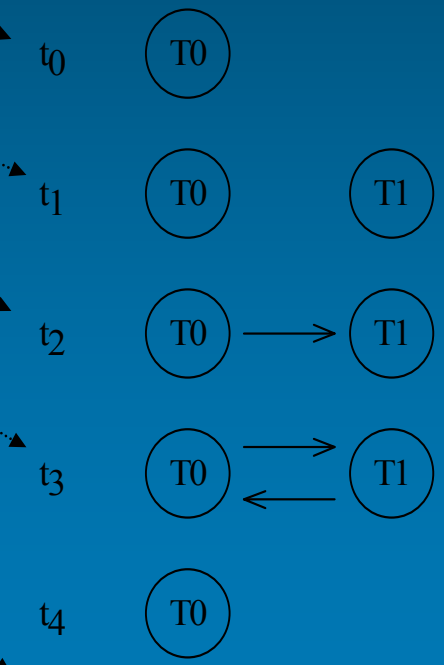
Odpravljanje mrtve zanke..

- SUPB izbere žrtev preko enega od naslednjih kriterijev:
  - čas izvajanja transakcij ("starost") do nastopa mrtve zanke,
  - predvideni čas izvajanja transakcij do njihovega zaključka,
  - število odobrenih zaseženj,
  - število izvedenih ažuriranj.
- Ne glede na izbrani kriterij je potrebno poskrbeti, da ne bo vedno znova razveljavljena ena in ista transakcija

Odpravljanje mrtve zanke

- Razvoj čakalnega grafa:

T0	*čas*	*T1*
E-PoiščiPreberi(R1,a) *a := a - 10* *Ažuriraj(R1,a)*	*t0*	
	t1	*E-PoiščiPreberi(R2,c)* *c := c - 20* *Ažuriraj(R2,c)*
E-PoiščiPreberi(R2,b) *(čakanje na odobritev)*	*t2*	
	t3	*E-PoiščiPreberi(R1,d)* *(čakanje na odobritev)*
	(nastop mrtve zanke)	
		(prekinitev izvajanja) *(razveljavitev)* *(pričetek ponovnega izvajanja)*
(zaseženje odobreno) *(ukaz je izveden)* *b := b + 10* *Ažuriraj(R2,b)* *Pomni*	*t4*	



Časovno označevanje..

- Postopek za nadzor nad sočasnim izvajanjem transakcij, ki ne temelji na zaseganju, zato predstavlja manjšo dodatno obremenitev SUPB
- Postopek je uporaben tudi pri porazdeljenih PB, kjer ne obstaja centraliziranega nadzora nad izvajanjem transakcij
- Pri časovnem označevanju se podatki ne zasegajo, zato tudi ne more priti do mrtve zanke

Časovno označevanje..

- Pri časovnem označevanju se transakcije izvajajo tako, da je njihov zaporedniški razpored ekvivalenten takemu zaporednemu razporedu, po katerem se najprej izvede starejša transakcija in nato mlajša
- Konfliktne oziroma potencialno konfliktne zahteve po branju ali ažuriranju podatkov se razrešujejo s prekinitvijo in ponovnim izvajanjem transakcije

Časovno označevanje..

- Protokol obsega naslednja pravila:
  - Vsaki transakciji Ti priredi SUPB časovno oznako **to(Ti),** ki je enaka njenemu startnemu času, in enolično označuje posamezno transakcijo.
    - V primeru centralizirane podatkovne baze se dve transakciji tako ne moreta pričeti ob istem času,
    - pri porazdeljenih podatkovnih bazah pa se časovna oznaka kombinira še z oznako lokalnega SUPB, kar nato skupaj omogoča enolično identifikacijo.
  - Transakciji, katere izvajanje se po pravilih protokola prekine, se pri ponovnem izvajanju priredi nova časovna oznaka.
  - Pri izvajanju transakcij se mora kot zaščita pred transakcijskimi nesrečami uporabljati *odloženo ažuriranje*. Vse spremembe, ki jih je transakcija izvedla, se v podatkovni bazi uveljavijo šele ob njenem uspešnem zaključku. Tako je preprečena možnost branja neobstoječega podatka (neobstoječega zapisa).

Časovno označevanje..

- Vsakemu podatku P oz. objektu zaseganja P, priredi SUPB dve časovni oznaki:
  - **tp(P)** je enak časovni oznaki to(Ti) najmlajše transakcije doslej, ki je podatek P uspešno prebrala;
  - **ta(P)** je enak časovni oznaki to(Ti) najmlajše transakcije doslej, ki je podatek P uspešno ažurirala.
- Ob zahtevi transakcije Ti po branju podatka P:
  - če je **to(Ti) < ta(P),** se Ti razveljavi in vrne v ponovno izvajanje;
  - v nasprotnem primeru se branje izvede, časovna oznaka povezana z branjem podatka P pa se spremeni v: **tp(P) = max(tp(P), to(Ti)).**
- Ob zahtevi transakcije Ti po ažuriranju podatka P:
  - če je **to(Ti) < tp(P)** ali **to(Ti) < ta(P),** se Ti razveljavi in vrne v ponovno izvajanje;
  - v nasprotnem primeru se ažuriranje izvede, časovna oznaka povezana z ažuriranjem podatka P, pa se spremeni v: **ta(P) = to(Ti).**

Časovno označevanje..

- Protokol s peto točko zagotavlja, da starejša transakcija ne more prebrati podatka, ki ga je pred tem ažurirala mlajša transakcija. S točko šest pa, da starejša transakcija ne more ažurirati podatka, ki ga je pred tem prebrala ali ažurirala mlajša transakcija
- Transakcija TX se lahko ob nekem branju ali ažuriranju nadaljuje le, če je bilo zadnje ažuriranje (ali branje) tega podatka s strani starejše transakcije TY
- Če transakciji podatek le bereta, je zaporedje branj seveda nepomembno

Časovno označevanje

- Transakcije se ob uporabi časovnega označevanja praviloma prekinjajo pogosteje, kot bi bilo potrebno oz. bi bilo ob uporabi prej obravnavanih protokolov
- Ob konfliktni situaciji se vedno prekine delovanje starejše transakcije
- Daljše transakcije postanejo pogosto prekinjene zaradi ažuriranja, ki ga izvedejo mlajše. Protokol daje implicitno prednost krajšim transakcijam pred daljšimi

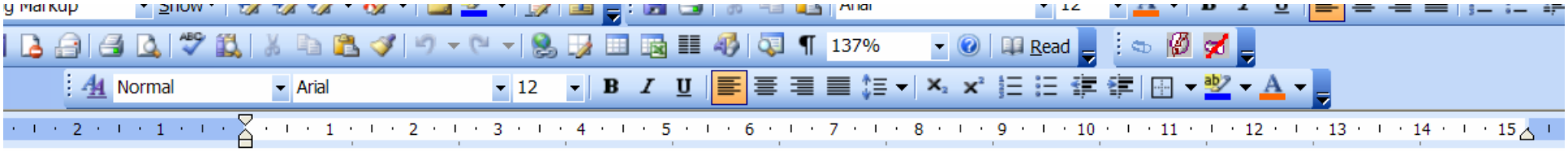
Izkušnje iz prakse

- Sodobni SUPB izvajajo avtomatsko zaseganje in avtomatsko odkrivanje mrtve zanke
- S posebnimi ukazi dosežemo implicitni E-PoiščiPreberi. Oracle: SELECT … FOR UPDATE

Vaja

Ugotovi, ali je izmenični razpored ukazov transakcij T1 in T2 zaporedniški. Če ni, napiši razlog (detaljno argumentiraj!) in napiši zaporedniški vrstni red izvajanja ukazov transakcij T1 in T2 z uporabo protokola PXC. X, Y in Z so naslovi v podatkovni bazi, x, y in z pa lokalne spremenljivke.

t	T1: Odštej	T2: Prištej
1	Začetek	
2	PoišČiPreberi(X,x)	
3		Začetek
4		PoišČiPreberi (Y,y)
5	x=x-10	
6		y=y+10
7	PoišČiPreberi (Y,y)	
8		PoišČiPreberi (Z,z)
9	y=y-10	
10		z=z+10
11	Ažuriraj(X,x)	
12		Ažuriraj(Z,z)
13	Ažuriraj(Y,y)	
14		Ažuriraj(Y,y)
15	Pomni	
16		Pomni



Razpored ukazov transakcij NI zaporedniški. Razlog: izgubljeno ažuriranje iz vrstice 13 transakcije T1, ki se izgubi z ažuriranjem iz vrstice 14 transakcije T2.

Rešitev z uporabo protokola PXC:

t	T1: Odštej	T2: Prištej
1	Začetek	
2	E-PoišČiPreberi(X,x)	
3		Začetek
4		E-PoišČiPreberi (Y,y)
5	x=x-10	
6		y=y+10
7	E-PoišČiPreberi (Y,y) (čakanje na odobritev)	
8		E-PoišČiPreberi (Z,z)
9		z=z+10
10		Ažuriraj(Z,z)
11		Ažuriraj(Y,y)
12		Pomni (vsa zaseženja se sprostijo)
13	(zaseženje je odobreno, ukaz se izvede)	
14	y=y-10	
15	Ažuriraj(X,x)	
16	Ažuriraj(Y,y)	
17	Pomni	