Poglavje 3

Obnavljanje podatkovne baze

- Potreba po obnovljivosti
- Transakcije in obnovljivost
- Komponente SUPB za obvladovanje obnovljivosti
- Tehnike obnovljivosti
- Obnovljivost v porazdeljenih SUPB

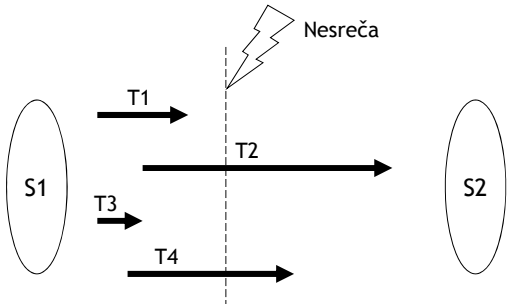
Obnavljanje podatkovne baze

Kaj si bomo pogledali?

- Potreba po obnovljivosti
- Transakcije in obnovljivost
- Komponente SUPB za obvladovanje obnovljivosti
- Tehnike obnovljivosti
- Obnovljivost v porazdeljenih SUPB

Kaj je obnova podatkov po nesreči?

- Proces vzpostavljanja podatkovne baze v zadnje veljavno stanje, ki je veljalo pred nastopom nesreče.



Potreba po obnovljivosti…

- Shranjevanje podatkov se običajno navezuje na štiri različne tipe medijev za shranjevanje podatkov, z naraščajočo stopnjo zanesljivosti:
  - glavni pomnilnik (neobstojni pomnilnik): podatki v njem ne preživijo sistemskih nesreč, je zelo hiter,
  - magnetni disk ("online" obstojni pomnilnik): zaneslivejši in cenejši od glavnega pomnilnika, vendar tudi počasnejši,
  - magnetni trak ("offline" obstojni pomnilnik): še zaneslivejši in cenejši od diska, vendar tudi počasnejši, omogoča samo zaporedni dostop,
  - optični disk: najzaneslivejši od vseh, še cenejši od traku, hitrejši od traku, omogoča neposredni dostop do podatkov.

Potreba po obnovljivosti...

- Obstaja več vrst nesreč, od katerih je potrebno vsako obravnavati na drugačen način.
- Nesreča lahko prizadene podatke tako v glavnem, kot v sekundarnem pomnilniku.
- Vzroki za nesreče so naslednji:
  - **odpoved sistema** (system crash): zaradi napak v strojni ali programski opremi; posledica je izguba podatkov v glavnem pomnilniku,
  - **poškodbe medija**: zaradi trka glave diska ob magnetno površino postane medij neberljiv; posledica so neberljivi deli sekundarnega pomnilnika,
  - **programska napaka v aplikaciji**: zaradi logične napake v programu, ki dostopa do podatkov v PB, pride do napak v eni ali več transakcijah,
  - **neprevidnost**: zaradi nenamernega uničenja podatkov s strani administratorjev ali uporabnikov,
  - **sabotaža** (namerno oviranje dela): zaradi namernega popačenja ali uničenja podatkov, uničenja programske ali strojne opreme.

Potreba po obnovljivosti

- Ne glede na vrsto napake, vedno smo pri nesrečah soočeni z dvema bistvenima problemoma:
  - izguba podatkov v glavnem pomnilniku (vključno s podatki v podatkovnih vmesnikih),
  - izguba podatkov na sekundarnem pomnilniku.
- V nadaljevanju:
  - pregled tehnik za preprečevanje izgub podatkov zaradi prej omenjenih problemov in
  - tehnike za obnavljanje po nesreči.

Transakcije in obnovljivost…

- Transakcija predstavlja osnovno enoto obnovljivosti.
- Za obnovljivost skrbi upravljalec za obnovljivost (recovery manager), ki mora v primeru nesreče zagotavljati dve od štirih lastnosti transakcij (ACID): atomarnost in trajnost.
- Naloga upravitelja obnovljivosti je, da pri obnovitvi PB po nesreči zagotovi:
  - da se vse spremembe, ki so bile v PB izvedene v okviru posamične transakcije uveljavijo **v celoti** ali pa
  - da se ne uveljavi **nobena** sprememba.
- Problem je kompleksen, ker pisanje v PB ne predstavlja atomarne akcije → transakcija lahko izvede COMMIT (uveljavitev sprememb), vendar se spremembe v PB ne zabeležijo, ker enostavno ne "dosežejo" PB (nastop nesreče).

Transakcije in obnovljivost…

- Kako poteka branje in pisanje podatkov - primer:

```
read(staffNo = x, salary)
salary = salary * 1.1
write(staffNo = x, new_salary)
```

- Za implementacijo branja mora SUPB izvesti naslednje korake:
  - poišči naslov bloka na disku, ki vsebuje zapis s ključem x;
  - prenesi blok v podatkovni vmesnik v pomnilniku;
  - kopiraj podatke o plači iz podatkovnega vmesnika v spremenljivko *salary*;
- Pri pisanju pa SUPB izvede naslednje operacije:
  - poišči naslov bloka na disku, ki vsebuje zapis s ključem x;
  - prenesi blok v podatkovni vmesnik v pomnilniku;
  - kopiraj podatke iz spremenljivke *salary* v podatkovni vmesnik;
  - zapiši vsebino podatkovnega vmesnika nazaj na disk;

  ZAKAJ na ta način? Ker se na disk piše ali bere CELOTNI blok.

Transakcije in obnovljivost…

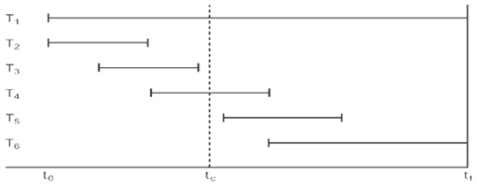
- Podatkovni vmesniki (database buffers) predstavljajo področje v glavnem pomnilniku, v katerega se pri prenašanju podatkov iz/v sekundarnega pomnilnika podatki pišejo ali iz njega berejo.
- Prenos vsebine podatkovnih vmesnikov v sekundarni pomnilnik (trajne spremembe) se pri posodobitvah sproži samo v primeru izvedbe posebnih ukazov:
  - COMMIT ali
  - avtomatično, ko postanejo podatkovni vmesniki polno zasedeni (eksplicitno zapisovanje vsebine podatkovnih vmesnikov v sekundarni pomnilnik označujemo kot prisilno zapisovanje (force-writing)).

Transakcije in obnovljivost…

- Če se nesreča pripeti med pisanjem v podatkovne vmesnike ali med prenosom podatkov iz pod. vmesnikov v sek. pomnilnik, mora upravitelj za obnovljivost ugotoviti status transakcije, ki je izvajala pisanje v času nesreče:
  - če je transakcija izvedla ukaz COMMIT, mora upravitelj za obnovljivost zaradi zagotavljanja lastnosti trajnosti izvesti ponovno izvajanje transakcije (**rollforward** ali **redo**),
  - če transakcija ni izvedla ukaza COMMIT, mora upravitelj za obnovljivost zaradi zagotavljanja lastnosti atomarnosti izvesti razveljavljanje posodobitev, ki jih je do tedaj transakcija izvedla (**rollback** ali **undo**).
- Če je potrebno razveljaviti samo eno transakcijo govorimo o parcialnem razveljavljanju (partial undo). Ta se izvaja tudi pri sočasnem dostopanju do podatkov zaradi uporabe protokolov za nadzor sočasnosti.
- Če je potrebno razveljaviti vse v času nesreče aktivne transakcije, govorimo o globalni razveljavitvi (global undo).

Transakcije in obnovljivost… - PRIMER: uporaba UNDO/REDO

- Transakcije T1 do T6 se izvajajo sočasno, SUPB začne delovati ob t_0, nesreča pa nastopi ob t_f:



- T2 in T3 izvedeta COMMIT in spremembe se uveljavijo v PB.
- T1 in T6 ne izvedeta ukaza COMMIT do trenutka nesreče, zato jih upravitelj za obnavljanje pri ponovnem zagonu razveljavi (UNDO).
- Za T4 in T5 (COMMITed) ni jasno, do katere mere so se njune spremembe uveljavile v PB - ali je bila vsebina podatkovnih vmesnikov zapisana v sekundarni pomnilnik ali ne.
  - ker nimamo na razpolago nobene dodatne informacije o stanju transakcij, je upravitelj za obnavljanje prisiljen ponoviti (REDO) transakcije T4 in T5.

Transakcije in obnovljivost - Upravljanje z medpomnilnikom…

- Pri obnavljanju PB igra pomembno vlogo tudi upravljanje z medpomnilnikom.
- Upravljanje z medpomnilnikom je zadolženo za učinkovito upravljanje s podatkovnimi vmesniki, ki se uporabljajo za prenos podatkov med glavnim in sekundarnim pomnilnikom:
  - branje podatkov iz sek. pomn. v vmesnike dokler niso zapolnjeni in
  - uporaba zamenjevalne strategije, ki določa, kateri vmesniki se zapišejo v PB z namenom sprostitve vmesnikov za branje novih podatkov.
- Poznamo različne zamenjevalne strategije, npr.: FIFO in LRU.
- Upravljalec medpomnilnika ne sme prebrati strani iz diska, če se ta že nahaja v medpomnilniku.

Transakcije in obnovljivost - Upravljanje z medpomnilnikom…

- Pristop za učinkovito uprav.: podatkom za upravljanje z medpomn. se doda dve spremenljivki za vsak podatkovni vmesnik: *pinCount* in *dirty*, ki sta na začetku postavljeni na *pinCount* = 0 in *dirty* = off.
- Ko se pojavi zahteva po strani iz diska, upravitelj medpomnilnika preveri, če se ta že nahaja v medpomnilniku. Če se ne, potem upravitelj izvede naslednje:
  - uporabi zamenjevalno strategijo, s katero izbere podatkovni vmesnik za zamenjavo (replacement buffer) in poveča pinCount zahtevanega vmesnika. (Strategija ne izbere vmesnika, ki ima pinCount>1.)
  - če je dirty bit pod. vmesnika, ki bo zamenjan, postavljen na "on", se stran prepiše na disk.
  - stran se prenese iz diska v okvir, ki je določen za zamenjavo.

Transakcije in obnovljivost - Upravljanje z medpomnilnikom…

- Če se ista stran ponovno zahteva, se spremenljivka pinCount poveča za 1.
- Opomba:
  - pinCount: kolikokrat je bila stran v podatkovnem vmesniku zahtevana, vendar ne sproščena (število trenutnih uporabnikov strani).
- Ko sistem (druge enote SUPB) obvesti upravitelja medpomnilnika, da je stran prenehal uporabljati, se pinCount podatkovnega vmesnika zmanjša za 1. Sistem upravitelju hkrati tudi sporoči, ali je bila stran posodobljena, in setira spremenljivko dirty=on.
- Ko je pinCount=0 se stran lahko zapiše na disk, če je bila posodobljena (dirty=on).

Transakcije in obnovljivost - Upravljanje z medpomnilnikom…

- Z vidika obnavljanja PB je v okviru zapisovanja strani na disk (za katerega je odgovoren upravitelj medpomnilnika) potrebno omeniti naslednje zapisovalne politike:
  - **Steal policy** omogoča, da upravitelj medpomnilnika zapiše vsebino podatkovnega vmesnika na disk preden transakcija izda ukaz COMMIT (preden je njegov pinCount=0). Z drugimi besedami, upravitelj transakciji "ukrade" stran. Alternativna politika: **no-steal**.
  - **Force policy** zagotavlja, da se vse strani, ki jih transakcija posodobi zapišejo na disk, takoj ko transakcija izda ukaz COMMIT. Alternativna politika: **no-force**.

Transakcije in obnovljivost - Upravljanje z medpomnilnikom…

- Z vidika implementacije je najpreprostejša uporaba politik no-steal in force:
  - no-steal: ni potrebno razveljavljati (undo) posodobitev ponesrečenih transakcij v PB, ker se spremembe še niso zapisale na disk,
  - force: ni potrebno ponoviti sprememb, ki so jih povzročile uveljavljene transakcije, ki so se ponesrečile po izdaji ukaza COMMIT, ker se spremembe zapišejo v PB takoj ob COMMIT-u.
- Po drugi strani pa:
  - steal: se izogne potrebi po velikem medpomnilnem prostoru za shranjevanje posodobljenih strani,
  - no-force: ni potrebno izvajati večkratnega zapisovanja posodobljene strani na disk s strani večih transakcij.
- Zaradi omenjenih razlogov večina SUPB-jev implementira steal, no-force politiko!

Komponente SUPB za obvladovanje obnovljivosti

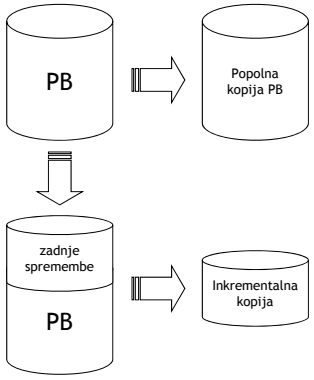
- SUPB naj bi zagotavljal naslednje komponente za obnavljanje PB po nesrečah:
  - *mehanizem za izdelavo varnostnih kopij*, ki periodično kreira kopije PB,
  - *dnevnik*, ki hrani podatke o trenutnem stanju transakcij in spremembah v PB,
  - *mehanizem za izvajanje kontrolnih točk*, ki omogoča da se posodobitve, ki jih izvajajo transakcije v PB, ohranijo (zahteva po izpisu vseh datotečnih vmesnikov na disk),
  - *upravljalec za obnovljivost*, komponenta SUPB, ki omogoča obnoviti podatkovno bazo v zadnje konsistentno stanje, ki je veljalo pred nastopom nesreče.

Mehanizem za izdelavo varnostnih kopij…

- Mehanizem mora omogočati izdelavo varnostnih kopij PB in dnevnika v določenih intervalih, ne da bi pred tem bilo potrebno prekiniti delovanje PB (stop).
- Kopijo PB se uporabi v primeru poškodb PB ali njenega uničenja.
- Varnostna kopija se običajno hrani na magnetnem traku.
- V zadnjem času se že uporabljajo varnostne kopije v oblaku
  - performance,
  - stroškovna učinkovitost.

Mehanizem za izdelavo varnostnih kopij

- Varnostna kopija je lahko:
  - popolna kopija PB ali
  - inkrementalna kopija, ki vsebuje samo spremembe izvedene od zadnje popolne ali inkrementalne kopije PB.



Izdelava klasične varnostne kopije...

- Dobra praksa s področja obnavljanja podatkovnih baz po nesrečah narekuje hrambo varnostnih kopij podatkov, ki so za poslovanje kritični, na ločeni lokaciji, izven prostorov poslovnega sistema.
- Poslovni sistemi za to običajno poskrbijo tako, da:
  - varnostne kopije zapisujejo na magnetne trakove (lahko tudi diske) in jih pošiljajo na neko oddaljeno lokacijo,
  - kar pa je drag in kompleksen postopek,
  - ki zahteva posebno strojno opremo,
  - ustrezno usposobljene kadre in
  - procedure (predpise), ki zagotavljajo, da se varnostne kopije sproti posodabljajo, da so zavarovane in da je podatke iz njih mogoče pridobiti in jih uporabiti v primeru nesreč.

Izdelava klasične varnostne kopije

- Številni sistemi prepuščajo transport in varovanje varnostnih kopij podatkov zunanjim izvajalcem,
- Sami pa še vedno skrbijo za zagotavljanje integritete podatkov v svojih varnostnih kopijah in za prej omenjene procedure → visoki stroški.

Izdelava varnostne kopije v oblaku...

- Kot alternativa klasični izdelavi varnostnih kopij podatkov, se z razvojem oblačnega računalništva, pojavlja vedno več ponudnikov, ki omogočajo izdelavo varnostnih kopij podatkov v oblakih (ang. Cloud backup). (Amazon, Azure, Arhiviraj.si,…)
- Kopija v oblaku predstavlja način izdelave varnostne kopije podatkov, kjer se podatke iz podatkovne baze pošlje preko javnega ali privatnega omrežja na podatkovni strežnik, ki se nahaja na oddaljeni lokaciji – oblaku.

Izdelava varnostne kopije v oblaku...

- Podatkovni strežnik upravlja ponudnik oblačnih storitev.
- Ponudnik stranki zaračunava storitev hranjenja kopije, na osnovi potrebnega diskovnega prostora, pasovne širine ali števila uporabnikov te storitve.
- Sistem za izdelavo varnostnih kopij v oblaku temelji na aplikaciji, ki se nahaja pri uporabniku storitve in se proži s frekvenco (dnevno, tedensko itd.), ki je opredeljena v pogodbi o uporabi storitve.
- Če ima npr. stranka pogodbo za izdelavo dnevnih varnostnih kopij, potem aplikacija zbere, stisne, kriptira in pošlje podatke na podatkovni strežnik ponudnika storitve vsakih 24 ur.

Izdelava varnostne kopije v oblaku...

- Za zmanjševanje uporabljene pasovne širine pri prenosu podatkov se lahko uporabi pristop inkrementalne izdelave varnostne kopije:
  - kjer se intervalno, glede na sklenjeno pogodbo o uporabi oblačne storitve, v oblak prenašajo samo spremembe v originalni podatkovni bazi.
- Ker se podatki prenašajo preko Interneta, je podatkovna propustnost običajno omejena.
- Poleg tega lahko tudi sami ponudniki oblačnih storitev omejujejo propustnost, da preprečijo posamičnim uporabnikom nesorazmerno rabo virov v oblaku.

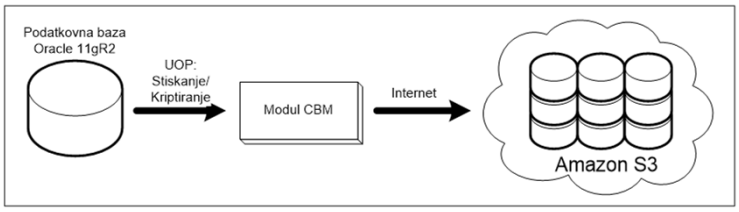
Prednosti varnostne kopije v oblaku...

- Prilagodljivost oblaka našim potrebam glede performans,
- Velika (neomejena) količina razpoložljivega prostora za shranjevanje in
- stroški, ki **se obračunavajo samo glede na dejansko uporabo virov**.
- Poleg tega uporaba oblaka tudi znatno poenostavi lastno informacijsko infrastrukturo, ker ni več potrebe po lastnem upravljanju s hrambo podatkov (npr.: delo z magnetnimi trakovi, pošiljanje magnetnih trakov na ločene lokacije itd.).

Pomanjkljivosti varnostne kopije v oblaku...

- Pomislek glede samega prenosa podatkov v oz. iz oblaka lahko predstavlja omejena pasovna širina Interneta, ki onemogoča hiter prenos velike količine podatkov (izdelava popolne varnostne kopije).
- Ponudnik storitev Amazon omenjeni problem rešuje tako, da ponuja za uvoz ali izvoz podatkov posebni storitvi: premik celotne varnostne kopije v ali iz oblaka in transport s prenosnim trdim diskom.
- Primer: po nesreči nam ponudnik s hitro pošto pošlje celotno varnostno kopijo na prenosnem disku. Na tak način je shranjevanje podatkov v oblaku primerljivo s klasično izdelavo varnostne kopije podatkov.

Primerjava hitrosti - primer



Lokacija pod. baze	Pod. propustnost brez stiskanja	Pod. propustnost z uporabo stiskanja	Čas izdelave celotne varnostne kopije		Čas izdelava inkrement. varnostne kopije	
			Stiskanje		Stiskanje	
			NE	DA	NE	DA
Testni strežnik	10 MB/s	43 MB/s	do 6 ur	od 2 uri	do 1 ure	od 30 min
Navidezni strežnik v EC2	35 MB/s	55 MB/s	do 2 uri	od 1 ure	do 20 min	od 10 min

	Čas izdelave varnostne kopije celotne podatkovne baze		Čas izdelava inkrementalne varnostne kopije	
	Brez stiskanja	S stiskanjem	Brez stiskanja	S stiskanjem
	do 1:15 ure	od 15 min	do 10 min	od 1:42 min
Transport na ločeno lokacijo in skladiščenje	do 3 ure	do 3 ure	do 3 ure	do 3 ure
Skupni čas	do 4:14 ure	od 3:15 ure	do 3:10 ure	od 3:1:4 ure

Ocena stroškov varnostne kopije v oblaku - primer

Storitev S3			
Cena hrambe podatkov (cena prvega 1 TB/mesec znaša 0,14$/GB):			
Količina podatkov	Čas hrambe (dnevi)	Izračun: 0,00452$/GB dan * št. dni * količina podatkov	Cena
celotna varnostna kopija - 250 GB	31 dni	0,00452*31*250	35,00$
1. inkr. varnostna kopija	24 dni	0,00452*24*25	2,71$
2. inkr. varnostna kopija	17 dni	0,00452*17*25	1,92$
3. inkr. varnostna kopija	10 dni	0,00452*10*25	1,13$
Cena prenosa podatkov v oblak:			
1 x 250 GB		1 x 25$	25,00$
3 x 25 GB		3 x 2,5$	7,50$
SKUPAJ:			73,26$

Dnevnik…

- V dnevnik se zapisujejo vse spremembe, ki jih transakcije izvedejo v PB.
- Dnevnik lahko vsebuje naslednje podatke:
  - transakcijske zapise, pri čemer je dnevniški zapis sestavljen iz:
    - identifikator transakcije,
    - tip dnevniškega vpisa (začetek tr., insert, update, detele, abort, commit),
    - identifikator podatka, na katerega se nanaša operacija (operacije: insert, delete, update) v okviru transakcije,
    - predhodna vrednost podatka: vrednost podatka pred ažuriranjem (samo za operacije update in delete),
    - vrednost podatka po ažuriranju (samo za operacije insert in update),
    - podatki potrebni za upravljanje dnevnika: kazalec na prejšnji in naslednji dnevniški zapis, ki pripada določeni transakciji.
  - zapise kontrolnih točk.

Dnevnik…

- Primer segmenta dnevniške datoteke, ki prikazuje 3 sočasne transakcije T1, T2 in T3. Stolpca pPtr in nPtr predstavljata kazalce na predhodni in naslednji dnevniški vpis.

Tid	Time	Operation	Object	Before image	After image	pPtr	nPtr
T1	10:12	START				0	2
T1	10:13	UPDATE	STAFF SL21	(old value)	(new value)	1	8
T2	10:14	START				0	4
T2	10:16	INSERT	STAFF SG37		(new value)	3	5
T2	10:17	DELETE	STAFF SA9	(old value)		4	6
T2	10:17	UPDATE	PROPERTY PG16	(old value)	(new value)	5	9
T3	10:18	START				0	11
T1	10:18	COMMIT				2	0
	10:19	CHECKPOINT	T2, T3				
T2	10:19	COMMIT				6	0
T3	10:20	INSERT	PROPERTY PG4		(new value)	7	12
T3	10:21	COMMIT				11	0

- Zaradi pomembne vloge dnevnika pri obnavljanju podatkov po nesrečah, je ta podvojen ali celo potrojen.
- Včasih je bil dnevnik shranjen na magnetnem traku (zanesljivejši in cenejši).
- Danes se pričakuje, da je SUPB pri manjših nesrečah sposoben hitro obnoviti PB v stanje pred nesrečo. To pa zahteva, da se dnevnik hrani na disku (online medij).

Dnevnik...

- V okoljih kjer se v dnevnike piše velika količina podatkov, te podatke ni mogoče ves čas imeti na razpolago - "online".
- V dnevniku morajo biti na razpolago (online) samo podatki za hitro obnavljanje, za primere manjših nesreč (npr.: razveljavitev transakcije, ki bi lahko povzročila mrtvo zanko).
- Večje napake (npr.: udarec glave diska v magnetno površino) zahtevajo več časa za obnovitev in ponavadi zahtevajo večjo količino podatkov iz dnevnika. V takih primerih, ko obnavljanje PB traja daljši čas, je možno dele dnevnika tudi prenašati iz magnetnih trakov (offline pomnilnik) v "online" pomnilnik (disk).

Dnevnik

- REALIZACIJA DNEVNIKA, ki podpira tudi uporabo "offline" sekundarnega pomnilnika (magnetni trak):
  - na disku se vzdržujeta dve dnevniški datoteki,
  - v datoteko A se vpisujejo podatki, dokler ta ni 70% zasedena,
  - ko kapaciteta datoteke A doseže 70%, se izvrši preklop na dnevniško datoteko B in zapisi novih transakcij se nato vpisujejo v datoteko B,
  - transakcije, ki se do preklopa še niso zaključile, še vedno vpisujejo podatke v datoteko A, dokler se ne zaključijo, nakar se datoteka A zapre,
  - ko se datoteko A zapre se ta prepiše na magnetni trak (offline pomnilnik),
  - ko je datoteka B 70% polna se izvrši preklop na datoteko A,
  - tak način realizacije dnevnika poenostavi obnavljanje PB po nesreči, saj se vpisi določene transakcije nahajajo ali v "online" pomnilniku (disk) ali pa v "offline" pomnilniku (magnetni trak).

Mehanizem za izvajanje kontrolnih točk…

- Podatki iz dnevnika se rabijo za obnovitev PB po nesreči → pri obnavljanju ne vemo, koliko dnevniških vpisov je potrebno prebrati, ne da bi ponavljali transakcije, ki so se v PB že uspešno uveljavile.
- Količino presežnega iskanja in procesiranja zaradi pregledovanja dnevniških vpisov lahko omejimo z uporabo kontrolnih točk.
- KONTROLNA TOČKA: točka sinhronizacije med PB in dnevnikom. Izvrši se zahteva po izpisu vseh podatkovnih 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.

Mehanizem za izvajanje kontrolnih točk…

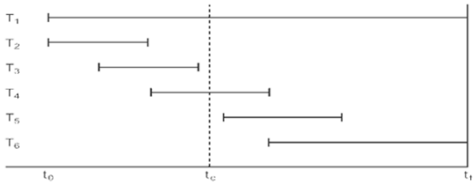
- Kontrolne točke se izvajajo po vnaprej opredeljenem urniku in vključujejo naslednje operacije:
  - zapisovanje vseh dnevniških vpisov iz glavnega pomnilnika v sekundarni pomnilnik (neposredno iz pomnilnika na disk!!!),
  - zapisovanje posodobljenih blokov v podatkovnih vmesnikih v sekundarni pomnilnik,
  - zapisovanje zapisa kontrolne točke v dnevnik. Ta zapis vključuje identifikatorje vseh transakcij, ki so bile v času izvedbe kontrolne točke aktivne.
- Če se transakcije izvajajo zaporedno, potem je pri obnavljanju v dnevniku potrebno poiskati zadnjo transakcijo, ki se je pričela izvajati pred izvedbo zadnje kontrolne točke.

Mehanizem za izvajanje kontrolnih točk...

- Vse transakcije pred prej omenjeno tr. so že bile uveljavljene (commited) in njihove spremembe zapisane v PB v času izvedbe kontrolne točke.
- Zaradi tega je potrebno ponoviti samo transakcijo, ki je bila aktivna v času izvedbe kontrolne točke in vse transakcije, ki so ji sledile in katerih zapisi se nahajajo v dnevniku.
- Če je transakcija aktivna v trenutku nastopa nesreče, jo je potrebno razveljaviti.
- VELJA: Če se transakcije izvajajo sočasno, je potrebno ponoviti vse transakcije, ki so izdale ukaz commit od zadnje kontrolne točke naprej in razveljaviti vse transakcije, ki so bile aktivne v času nastopa nesreče.

Mehanizem za izvajanje kontrolnih točk… - PRIMER: Uporaba UNDO in REDO s kontrolno točko

- Predpostavimo, da se je v času t_c izvedla kontrolna točka:



- Spremembe transakcije T2 in T3 se zapišejo v sekundarni pomnilnik.
- V tem primeru upravitelju za obnavljanje ni potrebno ponoviti (redo) izvajanje transakcij T2 in T3.
- Transakcije T4 in T5 je potrebno ponoviti (redo), ker so ukaz commit izdale po izvedbi kontrolne točke.
- Transakcije T1 in T6 pa je potrebno razveljaviti (undo), ker so bile v času nastopa nesreče aktivne.

Mehanizem za izvajanje kontrolnih točk

- Na splošno velja, da je uporaba kontrolnih točk relativno poceni operacija.
- Ponavadi je mogoče izvesti 3 do 4 kontrolne točke na uro.
- Na ta način je možno zagotoviti, da bo potrebno obnoviti samo podatke iz zadnjih 15-20 minut obratovanja PB.

Tehnike obnovljivosti...

- Vrsta uporabljene procedure za obnavljanje podatkov v PB po nesreči je odvisna od obsega nastale škode. Razlikujemo dva primera:
  - obsežne poškodbe PB:
    - vzrok: npr. diskovna nesreča.
    - posledica nesreče: uničena podatkovna baza.
    - podatke se obnovi z uporabo kopije PB in dnevnika; podatki iz dnevnika služijo za ponovitev (redo) uveljavljenih transakcij.
    - ta način obnavljanja predvideva, da dnevnik ni bil poškodovan; dnevnik naj se torej nahaja na disku, ki je ločen od podatkovnih datotek.

Tehnike obnovljivosti...

- PB ni fizično poškodovana:
  - vzrok: odpoved sistema med izvajanjem transakcij.
  - posledica nesreče: PB preide v neveljavno – nekonsistentno stanje.
  - transakcije, ki so se prekinile je potrebno razveljaviti, ker so postavile PB v nekonsistentno stanje.
  - lahko se tudi zgodi, da je nekatere transakcije potrebno ponoviti, če njihove spremembe niso "dosegle" sekundarnega pomnilnika.
  - v tem primeru za obnavljanje ne potrebujemo kopije PB, ampak zadostujejo predhodne in posodobljene vrednosti podatkov, ki se nahajajo v dnevniških vpisih (glej primer izseka iz dnevnika).
- V nadaljevanju: tehniki za obnavljanje PB po nesrečah, ki ne povzročijo fizične poškodbe PB, ampak privedejo PB v nekonsistentno stanje:
  - odloženo ažuriranje in
  - sprotno ažuriranje.

Obnova podatkov po nesrečah

Kaj si bomo pogledali?

- Tehnike obnovljivosti
- Obnovljivost v porazdeljenih SUPB
- Obnovljivost v SUPB ORACLE

Tehnike obnovljivosti...

- Tehnike obnovljivosti podatkov po nesrečah, ki privedejo PB v nekonsistentno stanje:
  - odloženo ažuriranje,
  - sprotno ažuriranje in
  - uporaba senčnih strani.

- Odloženo in sprotno ažuriranje se ločita po načinu zapisovanja posodobljenih podatkov v PB, obe pa uporabljata dnevnik.
- Senčne strani ne uporabljajo dnevnika.

Te obnovitvene tehnike morajo biti za uporabnika transparentne!
→ uporabniku za obnavljanje ni potrebno eksplicitno skrbeti

Odloženo ažuriranje...

- Pri uporabi protokola za odloženo ažuriranje se posodobljeni podatki ne zapisujejo neposredno v PB.
- Vsa ažuriranja v okviru transakcije se najprej shranijo v dnevnik. Pri uspešnem zaključku transakcije se izvede dejansko ažuriranje PB.
- V primeru nesreče:
  - če se transakcija prekine, v PB ni potrebno razveljaviti nobene spremembe, ker se te nahajajo samo v dnevniku,
  - pred nesrečo uspešno zaključene transakcije je potrebno ponoviti (redo), ker se njihova ažuriranja lahko še niso dejansko zapisala v PB. V tem primeru se uporabi zapise v dnevniku na naslednji način:

Odloženo ažuriranje…

- Uporaba dnevnika za obnavljanje podatkov v primeru odloženega ažuriranja:
  - Ob začetku transakcije se v dnevnik vnese zapis "transaction start".
  - Ob vsaki operaciji zapisovanja se v dnevnik vnese zapis (glej poglavje dnevnik), razen stara vrednost ažuriranega podatka. Ažuriran podatek se dejansko ne vpiše v podatkovni vmesnik ali PB.
  - Ko je transakcija pred tem, da se uveljavi, se v dnevnik vnese zapis "transaction commit", vsi pripadajoči dnevniški zapisi se zapišejo na disk (dnevniško datoteko), nakar sledi njeno uveljavljanje → zapisi iz dnevnika se uporabijo za dejansko posodabljanje podatkov v PB. SUPB nato zbriše transakcijo iz liste aktivnih transakcij.
  - Če se transakcija prekine, se njene dnevniške zapise ne upošteva in zapisovanje v PB se ne izvede.

Odloženo ažuriranje…

- Dnevniški zapisi določene transakcije se zapišejo na disk, preden se njena ažuriranja dejansko uveljavijo.
- Če se sistemska nesreča pripeti med dejanskim ažuriranjem podatkov v PB, ostanejo njeni dnevniški zapisi ohranjeni in uveljavljanje (COMMIT) sprememb se lahko ponovi kasneje.
- Pri obnavljanju je po nesreči v dnevniku potrebno poiskati transakcije, ki so bile v času nesreče aktivne → izvaja se branje dnevniških zapisov v smeri nazaj do zadnje kontrolne točke na naslednji način:

Odloženo ažuriranje…

- Obnavljanje po nesreči poteka na naslednji način:
  - Vsako transakcijo, za katero v dnevniku obstajata zapisa "transaction start" in "transaction commit", je potrebno ponoviti:
    - Uporabijo se dnevniški zapisi transakcije v takem vrstnem redu, kot so bili dodani v dnevnik (dnevniški zapisi vsebujejo nove vrednosti ažuriranih podatkov).
    - Če je bilo zapisovanje ažuriranj transakcije izvedeno že pred nesrečo, pisanje v PB nima nobenega učinka → s ponovnim zapisovanje ažuriranega podatka v PB ne naredimo nobene škode.
    - Metoda zagotavlja, da se v PB posodobi vsak podatek, ki pred nesrečo ni bil pravilno posodobljen.

Odloženo ažuriranje

- Za vsako transakcijo, za katero v dnevniku obstajata zapisa "transaction start" in "transaction abort", se ne izvede ničesar. Ker do dejanskega zapisovanj podatkov v PB sploh ne pride, transakcije ni potrebno razveljavljati (ROLLBACK).

Med obnavljanjem lahko ponovno nastopi nesreča → zapisi iz dnevnika se ponovno uporabijo.

Sprotno ažuriranje…

- Pri uporabi protokola sprotnega ažuriranja transakcija izvaja neposredno spreminjanje podatkov v PB še preden se uspešno zaključi.
- V primeru nesreče je poleg ponavljanja (redo) uspešno zaključenih transakcij, potrebno razveljaviti (rollback) vse transakcije, ki so bile aktivne v času nesreče.
- V primeru sprotnega ažuriranja se za zaščito pred sistemskimi nesrečami uporabi dnevnik na naslednji način:

Sprotno ažuriranje…

- Način uporabe dnevnika pri sprotnem ažuriranju.
  - Ob začetku transakcije se v dnevnik vnese zapis "transaction start".
  - Pri zapisovanju podatka se v dnevnik vnese ustrezen dnevniški zapis.
  - Ko je dnevniški zapis vnesen, se posodobljen podatek zapiše v podatkovni vmesnik (medpomnilnik).
  - Dejansko posodabljanje PB se izvede ob prvem naslednjem prenosu vsebine podatkovnih vmesnikov na disk.
  - Ko transakcija izvede COMMIT, se v dnevnik doda zapis "transaction commit", transakcija pa se izbriše iz liste aktivnih transakcij.

Sprotno ažuriranje…

- POMEMBNO: Pri ažuriranju se morajo zapisi najprej vnesti v dnevnik, šele nato v PB → "write-ahead log protocol".
- V obratnem primeru, če bi se nesreča pojavila pred zapisovanjem zapisa v dnevnik, upravljalec za obnovljivost ne bi mogel razveljaviti oz. uveljaviti operacije, ki jo definira dnevniški vpis.
- Z uporabo write-ahead log protokola lahko upravljalec obnovljivosti enostavno predpostavi naslednje:

  Če v dnevniku ni zapisa "transaction commit" → transakcija je bila v času nesreče aktivna in jo je pri obnavljanju potrebno razveljaviti (undo).

Sprotno ažuriranje…

- Če se transakcija prekine, se dnevnik uporabi za razveljavitev sprememb v PB (stare vrednosti posodobljenih zapisov).
- Transakcija lahko nad določenim podatkom izvede več sprememb → spremembe se razveljavijo v obratnem vrstnem redu.

Sprotno ažuriranje

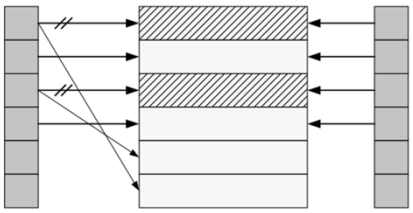
- Obnavljanje po nesreči poteka na naslednji način:
  - Vsako transakcijo, za katero v dnevniku obstajata zapisa "transaction start" in "transaction commit", se ponovi (redo) z uporabo novih vrednosti ažurirarnih podatkov → korak je potreben ker se določena pisanja v PB lahko niso izvedla.
  - Vsako transakcijo, za katero v dnevniku obstaja le zapis "transaction start" (brez "commit"), je potrebno razveljaviti (undo):
    - uporabijo se dnevniški zapisi in sicer stare vrednosti spremenjenih podatkov,
    - spremembe se razveljavljajo v obratnem vrstnem redu kot so bile vnesene v dnevnik.

Primerjava odloženega in sprotnega ažuriranja

- Z vidika uporabnosti:
  - Odloženo ažuriranje je bolj uporabno, če je v povprečju več transakcij prekinjenih (ni potrebno spreminjati PB). Npr.: ko podsistem za zagotavljanje sočasnosti prekinja transakcije.
  - Sprotno ažuriranje je uporabnejše, če se v povprečju pojavi več uspešnih transakcij (ni potrebno veliko popravljati podatkov v PB).

Uporaba senčnih strani…

- Obnavljanje s pomočjo senčnih strani temelji na uporabi dveh "tabel strani" (pristop vzdržuje dve tabeli strani med izvajanjem transakcije):
  - tekoča tabela strani,
  - senčna tabela strani.



Uporaba senčnih strani…

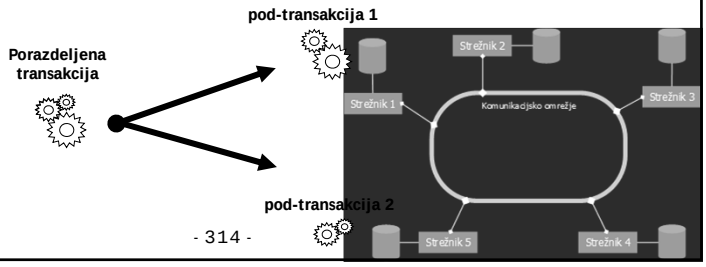
- Princip delovanja:
  - Strani glavnega pomnilnika, ki so bile ažurirane, se ne zapisujejo neposredno v PB, ampak na nezasedene bloke na disku.
  - Če se transakcija zaključi uspešno, se omenjeni novi bloki na disku vključijo v PB, namesto starih - neažuriranih.
  - Na začetku transakcije sta obe tabeli identični. Tekoča tabela strani se nahaja v glavnem pomnilniku, senčna pa na disku.
  - *V primeru nesreče*, ko se izda ukaz Pozabi, se tekoča tabela strani preprosto prepiše s senčno tabelo in v PB ni sledu o ažuriranjih, ki so se v okviru transakcij izvedla.
  - *Po uspešnem zaključku transakcije* se tekoča tabela strani izpiše na prosto mesto na disku in postane senčna tabela strani.

Uporaba senčnih strani

- Prednosti uporabe senčni strani:
  - vzdrževanje dnevnika odpade,
  - obnavljanje podatkov je občutno hitrejše, ker ni potrebno ponavljati ali razveljavljati operacij transakcij (znebimo se "overhead-a" pri delu z dnevnikom).
- Slabosti uporabe senčnih strani:
  - razdrobljenost (fragmentacija) podatkov na disku,
  - potreba po periodičnem "čiščenju" neuporabljenih strani na disku → obnavljanje prostega prostora na disku.

Obnovljivost v porazdeljenih SUPB…

- Porazdeljena podatkovna baza (SUPPB) se sestoji iz logično povezanih PB, ki so fizično porazdeljene na različnih lokacijah in povezane z računalniškim omrežjem.
- V SUPPB se izvajajo porazdeljene transakcije, ki se dekomponirajo v več pod-transakcij, ki se izvajajo na lokalnih SUPB-jih.
- Atomarnost je potrebno zagotoviti na nivoju pod-transakcij in na nivoju celotne porazdeljene transakcije.



Obnovljivost v porazdeljenih SUPB

- Predstavljene tehnike za obnavljanje podatkov zagotavljajo atomarnost transakcij na ravni lokalnih SUPB.
- Atomarnost porazdeljene transakcije je zagotovljena tako da:
  - se vse pod-transakcije uspešno izvedejo (vse izvedejo COMMIT) ali
  - pa se vse prekinejo (ROLLBACK).
- Dva protokola za porazdeljeno obnavljanje: dvo in trifazno potrjevanje (2FC in 3FC).

Dvo-fazno potrjevanje

- Delitev na koordinatorja in strežnike
- Faza glasovanja
  1. Koordinator pošlje poizvedbo za COMMIT (queryToCommit).
  2. Strežniki izvedejo svoje transakcije do točke, ko bi morali sprožiti COMMIT.
  3. Vsak strežnik koordinatorju sporoči, če je bil uspešen oz. neuspešen pri izvajanju svoje transkacije (glasuje ZA ali PROTI)
- Faza COMMIT - uspešna
  1. Če so vsi strežniki glasovali ZA, koordinator vsem pošlje zahtevo za COMMIT
  2. Vsak strežnik zaključi svojo transakcijo in pošlje POTRDITEV, ob tem tudi odklene vse zaklenjene vire.
  3. Koordinator zaključi porazdeljeno transakcijo, ko dobi potrditev vseh strežnikov

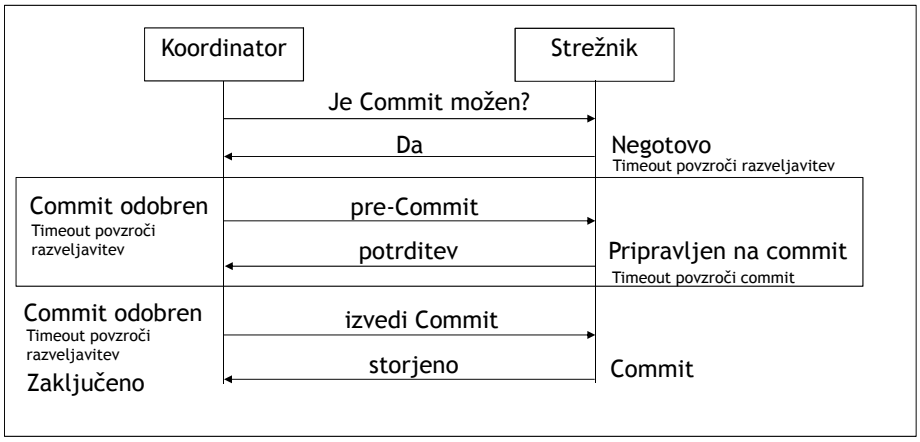
Dvo-fazno potrjevanje

- Faza COMMIT - neuspešna
  1. Če je katerikoli strežnik glasovali PROTI (ali po določenem času vsi strežniki niso odgovorili), koordinator vsem pošlje zahtevo za ROLLBACK
  2. Vsak strežnik razveljavi svojo transakcijo in pošlje POTRDITEV, ob tem tudi odklene vse zaklenjene vire.
  3. Koordinator razveljavi porazdeljeno transakcijo, ko dobi potrditev vseh strežnikov

- Slabost:
  – V primeru, da koordinator odpove, bodo strežniki, ki so glasovali ZA, v nedogled čakali na COMMIT ali ROLLBACK.

Tri-fazno potrjevanje

- Dodaja še fazo pre-COMMIT



Poglavje 4

Administracija podatkovnih baz

- Kaj dela administrator podatkovne baze?
- Delovne naloge administratirja podatkovne baze.
- Parametri zagona podatkovne baze
- Primer prikaza z MS SQL Server podatkovno bazo
- parametri zagona, gruče, distribuirana podatkovna baza

Administracija podatkovnih baz

Kaj si bomo pogledali?

- Delovno mesto administratorja - odgovornosti
- Aktivnosti administratorjev
- Osnovni proces namestitve podatkovne beze SQL Server

Delovno mesto administratorja

- Administrator podatkovne baze je odgovoren za:
  - performanse,
  - celovitost podatkov in
  - varnost v podatkovni bazi.
- Odgovoren je lahko še za:
  - planiranje,
  - razvoj in
  - odkrivanje napak pri delovanju podatkovne baze.

Aktivnosti administratorja PB

- Aktivnosti so odvisne od odgovornosti, ki so administratorju dodeljene
- Administrator:
  - identificira potrebe uporabnikov, nadzoruje dostope do PB in varnostne mehanizme,
  - nadzoruje performanse PB in zagotavlja najoptimalnejše čase izvajanja uporabniških poizvedb,
  - izdeluje konceptualne modele PB,
  - načrtuje organizacijo podatkov in načine uporabniških dostopov do podatkov,
  - prečisti logični podatkovni model, tako da se lahko preslika v ustrezni fizični model,
  - namešča in testira nove različice SUPB,
  - skrbi za upoštevanje standardov ,
  - izdeluje dokumentacijo,
  - uporabnikom dodeljuje pravice za dostop do podatkov,
  - razvija in testira različne scenarije izdelave varnostnih kopij in scenarije obnavljanja PB,
  - zagotavlja, da procedure za izdelavo varnostnih kopij in obnavljanje delujejo,
  - načrtuje potrebni diskovni prostor,
  - tesno sodeluje z vodji projektov, programerji in spletnimi razvijalci,
  - itd.

Namestitev PB – SQL Server 2008

- Osnovni proces namestitve strežnika SQL Server obsega naslednje korake:
- Načrtovanje sistema,
- Nakup strojne in programske opreme,
- Namestitev OS in paketov popravkov,
- Konfiguracija I/O sistema,
- Namestitev SQL Server in paketov popravkov,
- Zagon sistema,
- Konfiguriranje parametrov PB po namestitvi,
- GO!!!

Načrtovanje sistema

- Načrtovanje privarčuje veliko dragocenega časa, zato ga ne preskočite!
- Načrtovati je potrebno več stvari:
  - finance,
  - omrežje,
  - strojno opremo,
  - operacijski sistem in
  - podporo v produkciji.
- Pozorni moramo biti na zahteve glede skalabilnosti in razpoložljivosti podatkovne baze

Poglavje 5

Optimizacija poizvedb

- Zakaj optimizacija poizvedb
- Tehnike optimizacije
- Procesiranje in optimizacija poizvedb
- Beleženje statistike za potrebe optimizacije
- Faze procesiranja poizvedb
- Dinamična in statična optimizacija

Optimizacija poizvedb

Kaj si bomo pogledali?

- Zakaj optimizacija poizvedb
- Tehnike optimizacije
- Procesiranje in optimizacija poizvedb
- Beleženje statistike za potrebe optimizacije
- Faze procesiranja poizvedb
- Dinamična in statična optimizacija

Zakaj optimizacija

- Ko se pojavi relacijski model, časovna zahtevnost poizvedovanja velika kritika.
- Kasneje se razvijejo številni algoritmi za izvajanje kompleksnih poizvedb.
  - Poizvedbo je moč izvesti na številne različne načine.
  - Eden od ciljev optimizacije poizvedb je najti časovno najučinkovitejši način.
- Mrežni in hierarhični ter relacijski sistemi
  - V mrežnih in hierarhičnih sistemih za optimalnost skrbi programer (proceduralno poizvedovanje kot del programa)
  - V relacijskih programer pove samo KAJ in ne KAKO!

Tehnike optimizacije

- Dve osnovni tehniki optimizacije (v praksi gre navadno za kombinacijo):
  - Na osnovi hevrističnih pravil, ki pravilno razvrstijo operacije poizvedbe;
  - Na osnovi primerjave relativnih stroškov različnih strategij in izbiro tiste, ki predstavlja minimalno porabo sredstev.
- V centraliziranih SUPB predstavlja dostop do sekundarnega pomnilnika največji "strošek".

Procesiranje in optimizacija poizvedb

- Procesiranje poizvedbe zajema aktivnosti v zvezi z razčlenjevanjem, preverjanjem, optimizacijo in izvedbo poizvedbe.
  - Cilj: poizvedbo, zapisano v visoko-nivojskem jeziku, tipično SQL, pretvoriti v pravilno in učinkovito strategijo izvedbe, zapisano v nizko-nivojskem jeziku (implementacija relacijske algebre) in izvedba te strategije.
- Optimizacija poizvedbe: izbira najučinkovitejše izvedbene strategije za procesiranje poizvedbe.

Dva pogleda na optimizacijo

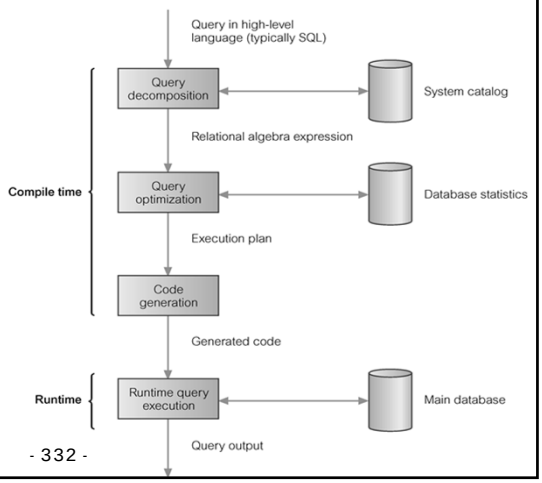
- Z optimizacijo poizvedbe skušamo izbrati transformacijo, ki minimizira porabo sredstev.
- Dva pogleda:
  - A) Želimo doseči minimalni možni čas izvajanja poizvedbe ≈ celotni čas poizvedbe je seštevek časov potrebnih za izvedbo posamezne operacije poizvedbe.
  - B) Želimo čim bolje izkoristiti razpoložljiva sredstva, npr. z velikim številom vzporednih operacij.
- V obeh primerih potrebujemo dodatne podatke, ki nam pomagajo pri optimizaciji.

Beleženje statistike

- Za izvajanje optimizacije potrebno poznati različne podatke – statistika
  - Statistika se nanaša na podatke o relacijah (npr. kardinalnost), atributih (npr. število različnih vrednosti) in indeksih (npr. višina indeksnega drevesa).
  - Vzdrževanje točne in "sveže" statistike je pomembno a zelo potratno… tipičen pristop – osveževanje statistike v nočnih urah.

Faze procesiranja poizvedb

- Procesiranje poizvedb se sestoji iz štirih faz:
  - Dekompozicija (razčlenjevanje in preverjanje);
  - Optimizacija;
  - Generiranje kode;
  - Izvedba.



Dinamična in statična optimizacija…

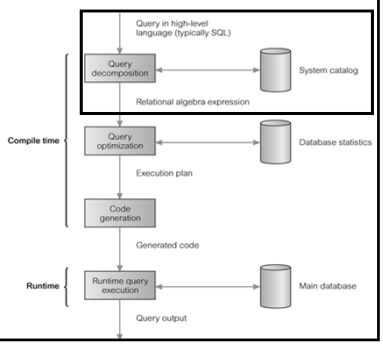
- Dve možnosti, kdaj se izvedejo prve tri faze procesiranja poizvedbe:
  - Dinamično procesiranje: dekompozicija in optimizacija vsakokrat, ko se požene poizvedba.
    - Prednost: uporabi se "sveža" statistika
    - Slabost: časovna potratnost zaradi vsakokratnega razčlenjevanja, preverjanja in optimizacije pred izvedbo. Zaradi prevelike časovne zahtevnosti, se včasih preveri samo nekaj strategij, kar lahko privede do neoptimalne izbire…

Dinamična in statična optimizacija

- Dve možnosti, kdaj se izvedejo prve tri faze procesiranja poizvedbe (nadaljevanje):
  - Statično procesiranje: dekompozicija in optimizacija se za določeno poizvedbo izvede samo enkrat.
    - Prednost: ker se dekompozicija in optimizacija ne izvedejo večkrat, je na razpolago več časa za preverjanje različnih strategij (takrat, ko se optimizacijo izvede).
    - Slabost: stara statistika.
      - Reševanje:
        - » (I) reoptimizacija, ko sistem zazna večje spremembe v statistiki,
        - » (II) optimizacija enkrat za vsako sejo.

Dekompozicija poizvedbe (DP)…

- Dekompozicija – prva faza procesiranja poizvedbe.
  - Cilj dekompozicije:
    - pretvoriti poizvedbo iz visoko-nivojskega jezika v relacijsko algebro.
    - Preveriti sintaktično in semantično pravilnost poizvedbe.
  - Tipični koraki dekompozicije:
    - Analiza
    - Normalizacija
    - Semantična analiza
    - Poenostavitev
    - Reorganizacija poizvedbe



DP – analiza…

- Zajema leksikalno in sintaktično analizo – podobno kot prevajalniki programskih jezikov.
- Dodatno preveri:
  - Če so relacije in atributi poizvedbe zapisani v sistemskem katalogu;
  - Če so operacije poizvedbe primerne glede na tip objekta, nad katerim se izvajajo.
- Primer

SELECT ~~staffNumber~~ ⇢ Ne obstaja v katalogu (staffNo)
FROM Staff
WHERE position ~~> 10;~~ ⇢ Nekompatibilnost operacije in podatkovnega tipa → position je tipa character

DP – analiza…

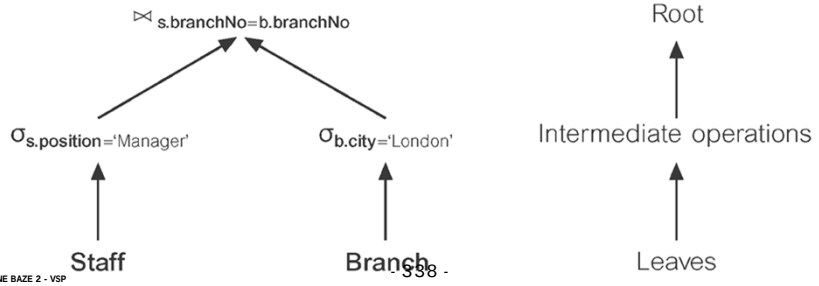
- Po leksikalnem in sintaktičnem preverjanju se poizvedba pretvori v interni zapis, primernejši za nadaljnjo obdelavo.
- Navadno gre za drevo poizvedbe, ki se zgradi v naslednjih korakih:
  - Za vsako osnovno relacijo poizvedbe kreiraj list drevesa.
  - Za vsako vmesno relacijo, ki je rezultat operacij algebre, kreiraj vozlišče drevesa.
  - Rezultat poizvedbe naj bo koren drevesa.
  - Zaporedje operacij za izvedbo naj gre od listov k korenu.

DP – analiza

- Primer:

```
SELECT *
FROM Staff s, Branch b
WHERE s.branchNo = b.branchNo AND
        (s.position = 'Manager' AND b.city = 'London');
```

Drevo relacijske algebre



DP – normalizacija…

- Zajema pretvorbo poizvedbe v normalizirano obliko, ki jo je lažje obdelovati.
  - Gre za pretvorbo predikatnega dela (WHERE).
- S pomočjo transformacijskih pravil se poizvedba pretvori v eno izmed dveh možnih oblik:
  - Konjunktivna normalna oblika: zaporedje konjunkcij, povezanih z operatorjem AND ($\wedge$).
  - Disjunktivna normalna oblika: zaporedje disjunkcij, povezanih z operatorjem OR ($\vee$).

DP – normalizacija

- Konjunktivna normalna oblika
  - Zaporedje konjunkcij, povezanih z operatorjem AND ($\wedge$). Konjunktivna selekcija zajema samo tiste n-terice, ki zadoščajo vsem konjunkcijam.
  - Primer

    (position = 'Manager' $\vee$ salary > 20000) $\wedge$ branchNo = 'B003'
- Disjunktivna normalna oblika
  - Zaporedje disjunkcij, povezanih z operatorjem OR ($\vee$). Dijuktivna selekcija zajema unijo vseh n-teric, ki zadoščajo posameznemu disjunktu.
  - Primer

    (position = 'Manager' $\wedge$ branchNo = 'B003') $\vee$
    (salary > 20000 $\wedge$ branchNo = 'B003')

DP – semantična analiza…

- Namen semantične analize je preprečiti normalizirane poizvedbe, ki so napačno formulirane ali kontradiktorne.
  - Napačna formulacija: posamezne komponente ne prispevajo k ciljnemu rezultatu.
  - Kontradiktornost: predikatu ne zadošča nobena n-terica. Npr. (position = 'Manager' **AND** position = 'Assistant').
- Kontradiktornost sistemi SUPB obravnavajo in rešujejo na različne načine.

DP – semantična analiza…

- Algoritmi za preverjanje semantične pravilnosti obstajajo le za podmnožico poizvedb, ki ne vsebujejo disjunkcij in negacij!
- Za preverjanje poizvedb, ki ne vsebujejo disjunkcij in negacij, lahko uporabimo
  - Graf povezav relacij za preverjanje napačnih formulacij in
  - Graf povezav normaliziranih atributov za preverjanje kontradikcij.

DP – semantična analiza…

- Postopek kreiranja grafa povezav relacij:
  - Kreiraj vozlišče za vsako relacijo ter za rezultat.
  - Kreiraj povezavo med vozlišči, ki predstavljajo relacije v stiku.
  - Kreiraj povezave med vozlišči, ki nastopajo kot izvor v operacijah projekcije.
  - Če graf ni povezan (obstajajo nepovezani deli), je poizvedba napačno formulirana!

DP – semantična analiza…

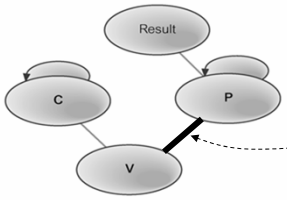
- Postopek kreiranja grafa povezav normaliziranih atributov:
  - Kreiraj vozlišče za vsako referenco na nek atribut ali konstanto 0.
  - Kreiraj usmerjeno povezavo med vozlišči, ki predstavljajo stik.
  - Za vsako selekcijo kreiraj usmerjeno povezavo med vozliščem, ki predstavlja atribut selekcije, in vozliščem, ki predstavlja konstanto 0.
  - Povezave a → b uteži z vrednostjo c, če gre za pogoj neenakosti tipa $a \leq b+c$
  - Povezave 0 → a uteži z vrednostjo -c, če gre za pogoj neenakosti tipa $a \geq c$
  - Če graf vsebuje cikel z negativnim seštevkom uteži, potem je poizvedba kontradiktorna.

Primer preverjanja semantične pravilnosti

- Imamo naslednjo poizvedbo

```
SELECT p.propertyNo, p.street
FROM Client c, Viewing v, PropertyForRent p
WHERE    c.clientNo = v.clientNo AND
         c.maxRent >= 500 AND
         c.prefType = 'Flat' AND
         p.ownerNo = 'CO93';
```

- Ali obstajajo napačne formulacije?



Kaj v poizvedbi manjka?

v.propertyNo = p.propertyNo

Postopek kreiranja grafa povezav normaliziranih atributov:

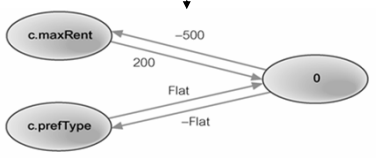
- Kreiraj vozlišče za vsako referenco na nek atribut ali konstanto 0.
- Kreiraj usmerjeno povezavo med vozlišči, ki predstavljajo stik.
- Za vsako selekcijo kreiraj usmerjeno povezavo med vozliščem, ki predstavlja atribut selekcije, in vozliščem, ki predstavlja konstanto 0.
- Povezave a → b uteži z vrednostjo c, če gre za pogoj neenakosti tipa a ≤ b+c
- Povezave 0 → a uteži z vrednostjo -c, če gre za pogoj neenakosti tipa a ≥ c

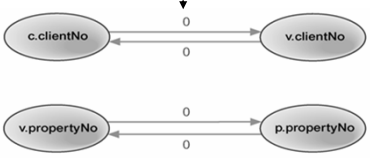
Če graf vsebuje cikel z negativnim seštevkom uteži, potem je poizvedba kontradiktorna.

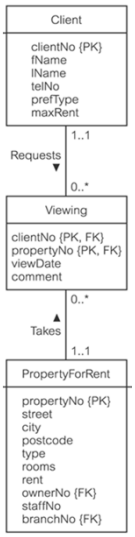
...ilnosti

- Imamo naslednjo poizvedbo

```
SELECT p.propertyNo, p.street
FROM Client c, Viewing v, PropertyForRent p
WHERE    c.maxRent > 500 AND
         c.clientNo = v.clientNo AND
         v.propertyNo = p.propertyNo AND
         c.prefType = 'Flat' AND
         c.maxRent < 200;
```







Postopek kreiranja grafa povezav normaliziranih atributov:

- Kreiraj vozlišče za vsako referenco na nek atribut ali konstanto 0.
- Kreiraj usmerjeno povezavo med vozlišči, ki predstavljajo stik.
- Za vsako selekcijo kreiraj usmerjeno povezavo med vozliščem, ki predstavlja atribut selekcije, in vozliščem, ki predstavlja konstanto 0.
- Povezave a → b uteži z vrednostjo c, če gre za pogoj neenakosti tipa a ≤ b+c
- Povezave 0 → a uteži z vrednostjo -c, če gre za pogoj neenakosti tipa a ≥ c

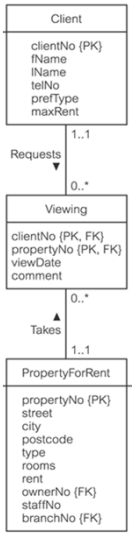
Če graf vsebuje cikel z negativnim seštevkom uteži, potem je poizvedba kontradiktorna.

...vilnosti

- Kaj pa če neenakosti postavimo pravilno?
- Ali je poizvedba še kontradiktorna?

```
SELECT p.propertyNo, p.street
FROM Client c, Viewing v, PropertyForRent p
WHERE    c.maxRent < 500 AND
         c.clientNo = v.clientNo AND
         v.propertyNo = p.propertyNo AND
         c.prefType = 'Flat' AND
         c.maxRent > 200;
```

- Preverimo z grafom povezav norm. atrib.



DP – poenostavitev...

- Zajema
  - detekcijo odvečnih kvalifikatorjev,
  - odpravo ponavljajočih sklopov in
  - transformacijo poizvedbe v semantično ekvivalentno, vendar enostavnejšo in za računanje učinkovitejšo obliko!
- V tej fazi se upoštevajo omejitve dostopa, definicije pogledov ter omejitve skladnosti, kar včasih botruje pojavi redundance.

DP – poenostavitev

- Če uporabnik nima vseh potrebnih pravic, se poizvedba zavrne. Sicer se za transformacijo uporabijo idempotentna pravila booleanove algebre, npr.

$p \wedge (p) \equiv p$	$p \vee (p) \equiv p$
$p \wedge \text{false} \equiv \text{false}$	$p \vee \text{false} \equiv p$
$p \wedge \text{true} \equiv p$	$p \vee \text{true} \equiv \text{true}$
$p \wedge (\neg p) \equiv \text{false}$	$p \vee (\neg p) \equiv \text{true}$
$p \wedge (p \vee q) \equiv p$	$p \vee (p \wedge q) \equiv p$

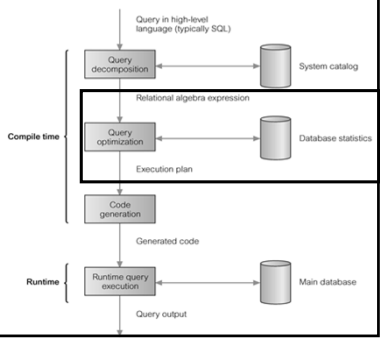
Hevristična optimizacija poizvedb

Kaj si bomo pogledali?

- Pravila transformacije relacijske algebre
- Primer uporabe transformacijskih pravil
- Strategije hevurističnega procesiranja

Hevristična optimizacija poizvedb

- Eden od pristopov optimizacije poizvedb je uporaba hevrističnih pravil.
- S pomočjo hevrističnih pravil lahko nek izraz v relacijski algebri r_1 transformiramo v ekvivalenten izraz r_2, pri čemer je r_2 učinkovitejši.
- Optimizacija je faza, ki sledi fazi dekompozicije.



Pravila transformacije relacijske algebre

- Optimizator pretvarja izraze relacijske algebre s pomočjo transformacijskih pravil.
- Transformacijsko pravilo 1
  - Konjunkcija operacij selekcije je enakovredna kaskadi posameznih operacij selekcije.
    $\sigma_{p \wedge q \wedge r}(R) = \sigma_p(\sigma_q(\sigma_r(R)))$
  - Včasih imenujemo tudi kaskada selekcije
  - Primer:
    $\sigma_{branchNo='B003' \wedge salary>15000}(Staff) = \sigma_{branchNo='B003'}(\sigma_{salary>15000}(Staff))$

Pravila transformacije relacijske algebre

- Transformacijsko pravilo 2
  - Komutativnost operacij selekcije.
    $\sigma_p(\sigma_q(R)) = \sigma_q(\sigma_p(R))$
  - Primer:
    $\sigma_{branchNo='B003'}(\sigma_{salary>15000}(Staff)) = \sigma_{salary>15000}(\sigma_{branchNo='B003'}(Staff))$

Pravila transformacije relacijske algebre

- Transformacijsko pravilo 3
  - Pri zaporedju operacij projekcij je pomembna samo zadnja operacija.
    $\Pi_L\Pi_M \ldots \Pi_N(R) = \Pi_L(R)$

    Operacije se izvajajo od operatorja navzven, zato ΠL (R) zadnja
  - Primer:
    $\Pi_{lName}\Pi_{branchNo,\ lName}(Staff) = \Pi_{lName}(Staff)$

Pravila transformacije relacijske algebre

- Transformacijsko pravilo 4
  - Komutativnost operacij selekcije in projekcije: Če predikat p vključuje le attribute iz operacije projekcije, potem sta operacije selekcije in projekcije komutativni.
    $\Pi_{Ai, \ldots, Am}(\sigma_p(R)) = \sigma_p(\Pi_{Ai, \ldots, Am}(R))$ where $p \in \{A_1, A_2, \ldots, A_m\}$
  - Primer:
    $\Pi_{fName, lName}(\sigma_{lName='Beech'}(Staff)) = \sigma_{lName='Beech'}(\Pi_{fName,lName}(Staff))$

Pravila transformacije relacijske algebre

- Transformacijsko pravilo 5
  - Komutativnost theta stika in kartezijskega produkta
    $R \bowtie_p S = S \bowtie_p R$
    $R \times S = S \times R$
  - Pravilo velja tudi za stik Equijoin in naravni stik
  - Primer:
    $Staff \bowtie_{staff.branchNo=branch.branchNo} Branch = Branch \bowtie_{staff.branchNo=branch.branchNo} Staff$

Pravila transformacije relacijske algebre

- Transformacijsko pravilo 6
  - Komutativnost selekcije in theta stika (ali kartezijskega produkta): Če predikat v operaciji selekcije vključuje le atribute ene relacije iz stika, potem operaciji selekcija in stik (ali kartezijski produkt) komutirata:
    $\sigma_p(R \bowtie_r S) = (\sigma_p(R)) \bowtie_r S$
    $\sigma_p(R \times S) = (\sigma_p(R)) \times S$, kjer $p \in \{A_1, A_2, \ldots, A_n\}$
  - Ali, če je predikat selekcije konjunkcija oblike $(p \wedge q)$, kjer se p nanaša le atribute relacije R, q pa le na atribute relacije S, potem operaciji selekcije in theta stika komutirata:
    $\sigma_{p \wedge q}(R \bowtie_r S) = (\sigma_p(R)) \bowtie_r (\sigma_q(S))$
    $\sigma_{p \wedge q}(R \times S) = (\sigma_p(R)) \times (\sigma_q(S))$

Pravila transformacije relacijske algebre

- Primer za transformacijsko pravilo 6

$$\sigma_{position='Manager' \wedge city='London'}(Staff \bowtie_{Staff.branchNo=Branch.branchNo} Branch) =$$
$$(\sigma_{position='Manager'}(Staff)) \bowtie_{Staff.branchNo=Branch.branchNo} (\sigma_{city='London'}(Branch))$$

Pravila transformacije relacijske algebre

- Transformacijsko pravilo 7
  - Komutativnost projekcije in theta stika (ali kartezijskega produkta): če je seznam atributov projekcije oblike $L = L_1 \cup L_2$, kjer L_1 zajema le atribute relacije R, L_2 pa atribute relacije S, pogoj stika pa atribute L, potem operaciji projekcije in theta stika komutirata.

    $\Pi_{L_1 \cup L_2}(R \bowtie_r S) = (\Pi_{L_1}(R)) \bowtie_r (\Pi_{L_2}(S))$

  - Primer:

    $\Pi_{position,city,branchNo}(Staff \bowtie_{Staff.branchNo=Branch.branchNo} Branch) =$
    $(\Pi_{position,\ branchNo}(Staff)) \bowtie_{Staff.branchNo=Branch.branchNo} ($
    $\Pi_{city,\ branchNo}(Branch))$

Pravila transformacije relacijske algebre

- Transformacijsko pravilo 7 (dodatek):
  - Če pogoj stika vsebuje tudi atribute, ki jih ni v L ($M = M_1 \cup M_2$, kjer M_1 vsebuje le atribute iz R, M_2 pa le atribute iz S), potem osnovno transformacijsko pravilo 7 velja, če vključimo še dodatno projekcijo:

    $\Pi_{L_1 \cup L_2}(R \bowtie_r S) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup M_1}(R)) \bowtie_r (\Pi_{L_2 \cup M_2}(S)))$

    Dodatna projekcija

  - Primer:

    $\Pi_{position,\ city}(Staff \bowtie_{Staff.branchNo=Branch.branchNo} Branch) =$
    $\Pi_{position,\ city}((\Pi_{position,\ branchNo}(Staff)) \bowtie_{Staff.branchNo=Branch.branchNo}$
    $(\Pi_{city,\ branchNo}(Branch)))$

Pravila transformacije relacijske algebre

- Transformacijsko pravilo 8
  - Komutativnost operacij unija in presek.

    $R \cup S = S \cup R$

    $R \cap S = S \cap R$

Pravila transformacije relacijske algebre

- Transformacijsko pravilo 9
  - Komutativnost operacij selekcije in operacij množic (unija, presek in razlika).

    $\sigma_p(R \cup S) = \sigma_p(S) \cup \sigma_p(R)$

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

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

Pravila transformacije relacijske algebre

- Transformacijsko pravilo 10
  - Komutativnost projekcije in unije.
    $\Pi_L(R \cup S) = \Pi_L(S) \cup \Pi_L(R)$

Pravila transformacije relacijske algebre

- Transformacijsko pravilo 11
  - Asociativnost theta stika (in kartezijskega produkta)
    - Kartezijski produkt in naravni stik sta vedno asociativna

      $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

      $(R \times S) \times T = R \times (S \times T)$

    - Če pogoj stika q vsebuje le atribute iz S in T, potem je theta stik asociativen

      $(R \bowtie_p S) \bowtie_{q \wedge r} T = R \bowtie_{p \wedge r} (S \bowtie_q T)$

Pravila transformacije relacijske algebre

- Primer za transformacijsko pravilo 11

  $(Staff \bowtie_{Staff.staffNo=PropertyForRent.staffNo} PropertyForRent) \bowtie_{ownerNo=Owner.ownerNo \wedge staff.lName=Owner.lName} Owner =$

  $Staff \bowtie_{staff.staffNo=PropertyForRent.staffNo \wedge staff.lName=lName} (PropertyForRent \bowtie_{ownerNo} Owner)$

Pravila transformacije relacijske algebre

- Transformacijsko pravilo 12
  - Asociativnost unije in preseka.

    $(R \cup S) \cup T = S \cup (R \cup T)$

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

Primer uporabe transformacijskih pravil...

- Primer poizvedbe:
  - Za stranke, ki iščejo stanovanje, najdi nepremičnine, ki ustrezajo lastnostim, ki so jih stranke podale.

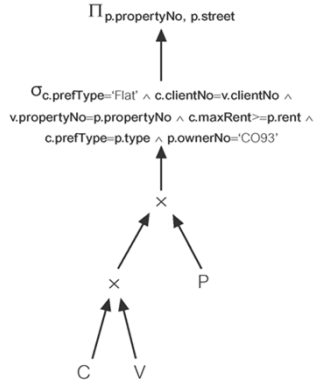
```
SELECT p.propertyNo, p.street
FROM Client c, Viewing v, PropertyForRent p
WHERE  c.prefType = 'Flat' AND
       c.clientNo = v.clientNo AND
       v.propertyNo = p.propertyNo AND
       c.maxRent >= p.rent AND
       c.prefType = p.type AND
       p.ownerNo = 'CO93';
```


Primer uporabe transformacijskih pravil...

- SQL pretvorimo v relacijsko algebro

$$\Pi_{p.propertyNo,\ p.street}(\sigma_{c.prefType='Flat' \wedge c.clientNo = v.clientNo \wedge v.propertyNo = p.propertyNo \wedge c.maxRent \geq p.rent \wedge c.prefType = p.type \wedge p.ownerNo = 'CO93'}((c \times v) \times p))$$

- Poizvedbo lahko prikažemo kot kanonično drevo relacijske algebre:



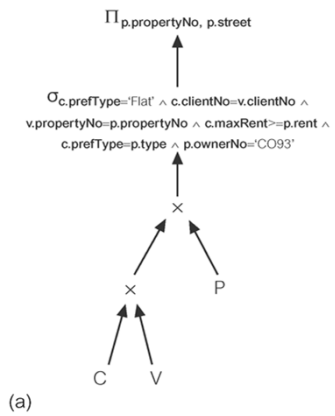
(a)

Primer uporabe transformacijskih pravil...

- Za izboljšanje učinkovitosti izvedbe poizvedbe uporabimo naslednja transformacijska pravila:
  - Prvi korak:
    - Uporabimo pravilo 1, da razbijemo selekcijo na individualne operacije selekcije
    - Uporabimo pravili 2 in 6, da spremenimo vrstni red operacij selekcije ter zamenjamo selekcije in kartezijske produkte.

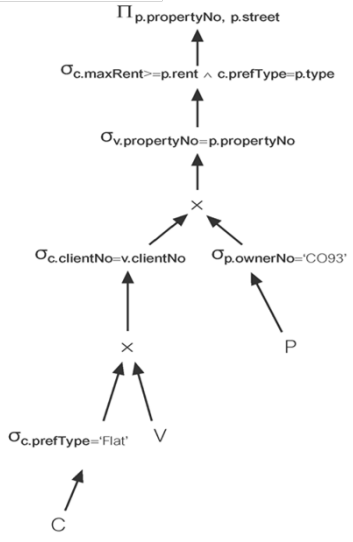
Primer uporabe transformacijskih pravil...

Izhodišče



(a)

Po prvem koraku



(b)

Primer uporabe transformacijskih pravil...

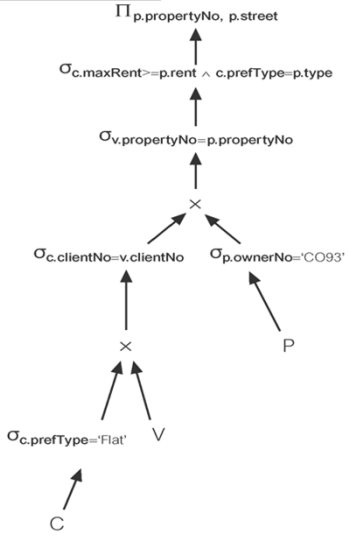
- Drugi korak:
  - Selekcijo z equijoin predikatom in kartezijskim produktom lahko zapišemo kot equijoin stik, kjer velja

    $\sigma_{R.a = S.b}(R \times S) = R \bowtie_{R.a = S.b} S$

  - Transformacijo uporabimo, kjer je možno

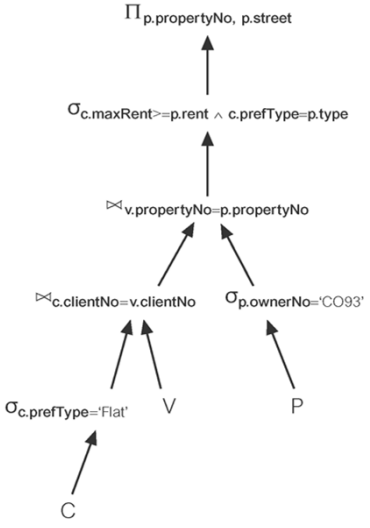
Primer uporabe transformacijskih pravil...

Po prvem koraku



(b)

Po drugem koraku



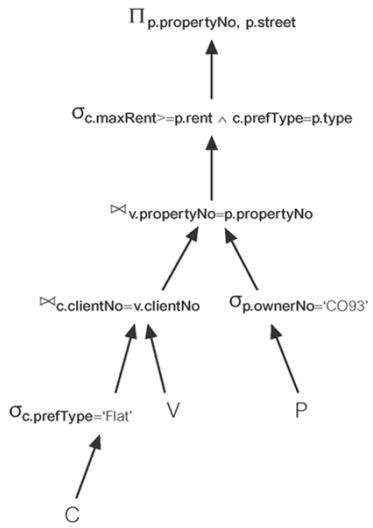
(c)

Primer uporabe transformacijskih pravil...

- Tretji korak:
  - Pravilo 11: da spremenimo vrstni red equijoin stikov, tako da so bolj restriktivni na začetku.

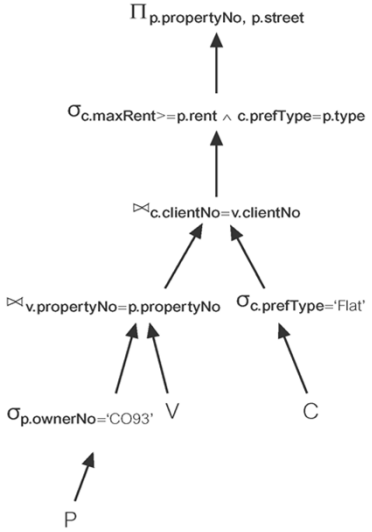
Primer uporabe transformacijskih pravil...

Po drugem koraku



(c)

Po tretjem koraku



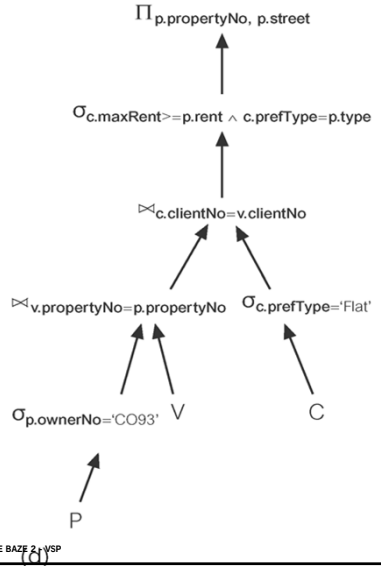
(d)

Primer uporabe transformacijskih pravil...

– Četrti korak:
  - Uporabimo pravili 4 in 7, da projekcije premaknemo pred equijoin stike in naredimo dodatno projekcijo (glej pravilo 7, dodatek).

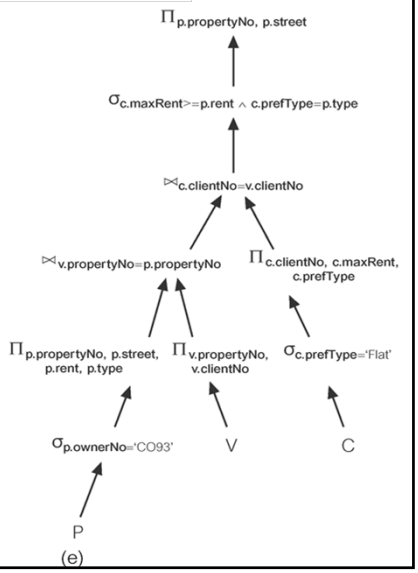
Primer uporabe transformacijskih pravil...

Po tretjem koraku



(d)

Po četrtem koraku



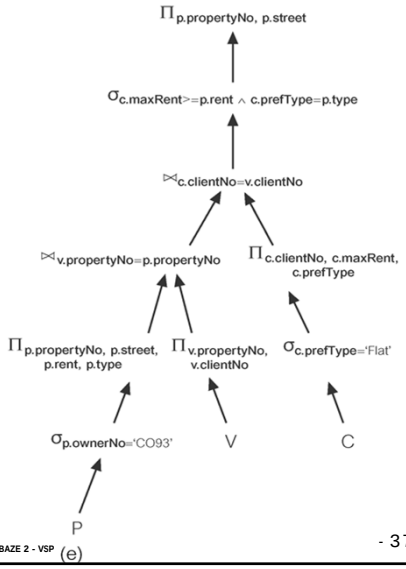
(e)

Primer uporabe transformacijskih pravil…

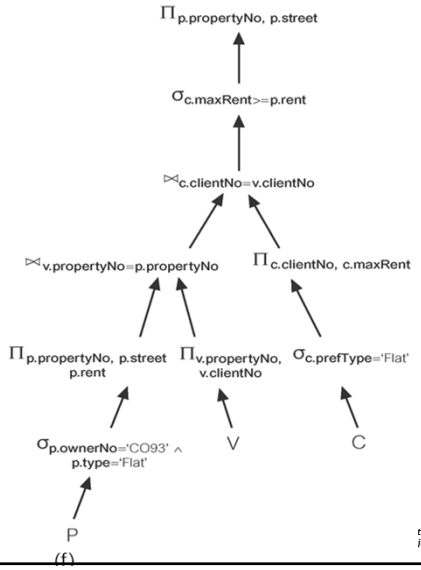
- Za dodatno optimizacijo lahko operacijo selekcije c.prefType = p.type poenostavimo na p.type = 'Flat', saj da je p.type = 'Flat' vemo že iz prvega dela predikata SQL stavka.
  - S takšno substitucijo se selekcija premakne po drevesu navzdol.
  - Končen rezultat je:

Primer uporabe transformacijskih pravil

Po četrtem koraku



Po dodatnem koraku



Strategije hevrističnega procesiranja…

- Strategija 1:
  - Operacije selekcije izvedi čim prej – selekcija zmanjša številnost relacije...
  - Operacije selekcije nad isto relacijo naj bodo skupaj
  - Uporabi transformacijska pravila 2, 4, 6 in 9
- Strategija 2
  - Združi kartezijski produkt z operacijami selekcije, ki predstavljajo obenem stični pogoj.

Strategije hevrističnega procesiranja…

- Strategija 3:
  - Uporabi asociativnost binarnih operacij in ponovno razporedi liste drevesa, tako da bodo najbolj omejevalne selekcije upoštevane na začetku**. Relacije čim bolj zreduciramo** preden uporabimo binarne operacije.
  - Podobno uporabimo asociativnost theta stika, da poizvedbo reorganiziramo tako, da se '**manjši' stiki izvedejo najprej**.

Strategije hevrističnega procesiranja…

- Strategija 4:
  - Z uporabo pravila 3 projekcije uredimo kaskadno, ter uporabimo pravila 4, 7 in 10 v zvezi s komutativnostjo projekcije in binarnih operacij, da projekcijo premaknemo čim bližje listom drevesa.
  - Projekcija zmanjša stopnjo relacij, zato priporočljivo izvesti na samem začetku.
  - Projekcije nad istimi relacijami je dobro obdržati skupaj.

Strategije hevrističnega procesiranja

- Strategija 5:
  - Če se nek izraz pojavi večkrat v eni poizvedbi in rezultat, ki ga ta izraz producira ni prevelik, potem delni rezultat shrani za nadaljnjo uporabo.
  - Smiselno le, če je rezultat dovolj majhen, da ga je moč hraniti v primarnem pomnilniku.
  - Če je potrebno rezultat hraniti v sekundarnem pomnilniku, potem je potrebno izračunati, kaj več stane: branje iz sekundarnega pomnilnika ali ponovni izračun izraza.