

Načrtovanje fizične podatkovne baze

- Fizično načrtovanje PB opredeljuje proces, s katerim izdelamo opis implementacije PB na sekundarnem pomnilnem mediju.
- Odvija se v konkretnem SUPB (npr. MariaDB)
- Opisuje
 - osnovne tabele in poglede,
 - datotečno organizacijo,
 - indekse za doseg učinkovitega dostopa do podatkov,
 - povezane omejitve,
 - varnostne mehanizme.



Metoda načrtovanja fizične PB...

- Sistematični koraki načrtovanja fizične PB:
 - K3 – Pretvori logični model v jezik za ciljni SUPB (tipično SQL)
 - K3.1 – Izdelaj načrt osnovnih relacij
 - K3.2 – Izdelaj načrt predstavitve izpeljanih atributov
 - K3.3 – Izdelaj načrt splošnih omejitev
 - K4 – Izdelaj načrt datotečne organizacije ter indeksov
 - K4.1 – Analiziraj transakcije
 - K4.2 – Izberi datotečno organizacijo
 - K4.3 – Določi indekse
 - K4.4 – Oцени velikost podatkovne baze
 - K5 – Izdelaj načrt uporabniških pogledov
 - K6 – Izdelaj načrt varnostnih mehanizmov
 - K7 – Preveri smiselnost uvedbe nadzorovane redundance podatkov (denormalizacija)

K3 – Pretvorba v jezik za SUPB

- Namen koraka: iz logičnega modela izdelati podatkovno shemo za ciljni SUPB.
- Poznati moramo funkcionalnosti ciljnega SUPB, npr.:
 - Kako izdelati osnovne relacije?
 - Ali ciljni SUPB podpira primarne, tuje in alternativne ključne?
 - Ali podpira obveznost podatkov (NOT NULL)?
 - Ali podpira domene – podatkovne podtipe, objektne tipe, ...
 - Ali podpira pravila omejitve podatkov?
 - Ali podpira prožilce (triggers) in bazne podprograme (stored procedures)?

K3.2 – Predstavitev izpeljanih (izračunanih) atributov...

- Namen: določiti, kako bodo v SUPB predstavljeni izpeljani oz. izračunani atributi.
- Preuči logični podatkovni model in podatkovni slovar; izdelaj seznam izpeljanih atributov.
- Za vsak izpeljani atribut določi:
 - Atribut je shranjen v podatkovni bazi
 - Atribut se vsakokrat posebej izračuna in se ne hrani v podatkovni bazi.

K3.2 – Predstavitev izpeljanih atributov...

- Pri odločitvi, kako predstaviti izpeljane attribute, upoštevaj:
 - “strošek” shranjevanja in vzdrževanja skladnosti izpeljanih atributov z osnovnimi atributi, iz katerih je izpeljan;
 - “strošek” vsakokratnega izračunavanja izpeljanega atributa.
- Izberi stroškovno ugodnejšo rešitev.
- Pomembno: scenariji uporabe!

Primer hranjenja izpeljanega atributa

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

Staff

staffNo	fName	IName	branchNo	noOfProperties
SL21	John	White	B005	0
SG37	Ann	Beech	B003	2
SG14	David	Ford	B003	1
SA9	Mary	Howe	B007	1
SG5	Susan	Brand	B003	0
SL41	Julie	Lee	B005	1

K3.3 – Načrt splošnih omejitev

- Namen: izdelava načrta splošnih omejitev za ciljni SUPB.
- Glede podpore splošnim omejitvam obstajajo velike razlike med SUPB-ji.
- Primer splošne omejitve:

```
CONSTRAINT StaffNotHandlingTooMuch  
CHECK (NOT EXISTS (SELECT staffNo  
                        FROM PropertyForRent  
                        GROUP BY staffNo  
                        HAVING COUNT (*) > 100))
```

Pomen omejitve: nihče od zaposlenih ne sme skrbeti za več kot 100 nepremičnin

K4 – Datotečna organizacija in indeksi

- Namen: izbrati optimalno datotečno organizacijo za shranjevanje osnovnih relacij ter potrebne indekse za doseganje ustrezne učinkovitosti.
- Načrtovalec mora dobro poznati, kakšne strukture in organizacije SUPB podpira ter kako deluje.
- Ključnega pomena so uporabniške zahteve v zvezi z željeno/pričakovano učinkovitostjo transakcij.
- Med SUPB-ji velike razlike.
- Vprašanje: datotečna organizacija (engine) SUPB MySQL-MariaDB?

K4.1 – Analiza transakcij...

- Namen: razumeti namen transakcij, ki bodo tekle na SUPB ter analizirati tiste, ki so najpomembnejše.
- Poskušaj določiti kriterije učinkovitosti:
 - Pogoste transakcije, ki imajo velik vpliv na učinkovitost;
 - Transakcije, ki so kritičnega pomena za poslovanje;
 - Pričakovana obdobja (v dnevu/ tednu), ko bo SUPB najbolj obremenjen (peak load).
- Preveri tudi:
 - Attribute, ki jih transakcije spreminjajo
 - "Iskalne" pogoje v transakcijah...

K4.1 – Analiza transakcij...

- Pogosto ni moč analizirati vseh transakcij. Pregledamo zgolj najpomembnejše.
- Za identifikacijo najpomembnejših transakcij lahko uporabimo:
 - Matriko transakcija/relacija, ki kaže, katere relacije se v transakcijah uporabljajo.
 - Diagram uporabe transakcij, ki kaže, katere transakcije bodo potencialno zelo frekventno izvajane.

K4.1 – Analiza transakcij...

- Možen pristop k obravnavi potencialno problematičnih delov modela:
 - Izdelamo matriko povezav transakcija/relacija,
 - Ugotovimo, katere relacije se najpogosteje uporabljajo v transakcijah,
 - Analiziramo, kateri podatki se uporabljajo v transakcijah, ki te relacije uporabljajo.

Primer matrike transakcija/relacija

Table 17.1 Cross-referencing transactions and relations.

Transaction/ Relation	(A)				(B)				(C)				(D)				(E)				(F)							
	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D				
Branch									X				X												X			
Telephone																												
Staff	X				X				X								X								X			
Manager																												
PrivateOwner	X																											
BusinessOwner	X																											
PropertyForRent	X					X	X	X													X							X
Viewing																												
Client																												
Registration																												
Lease																												
Newspaper																												
Advert																												

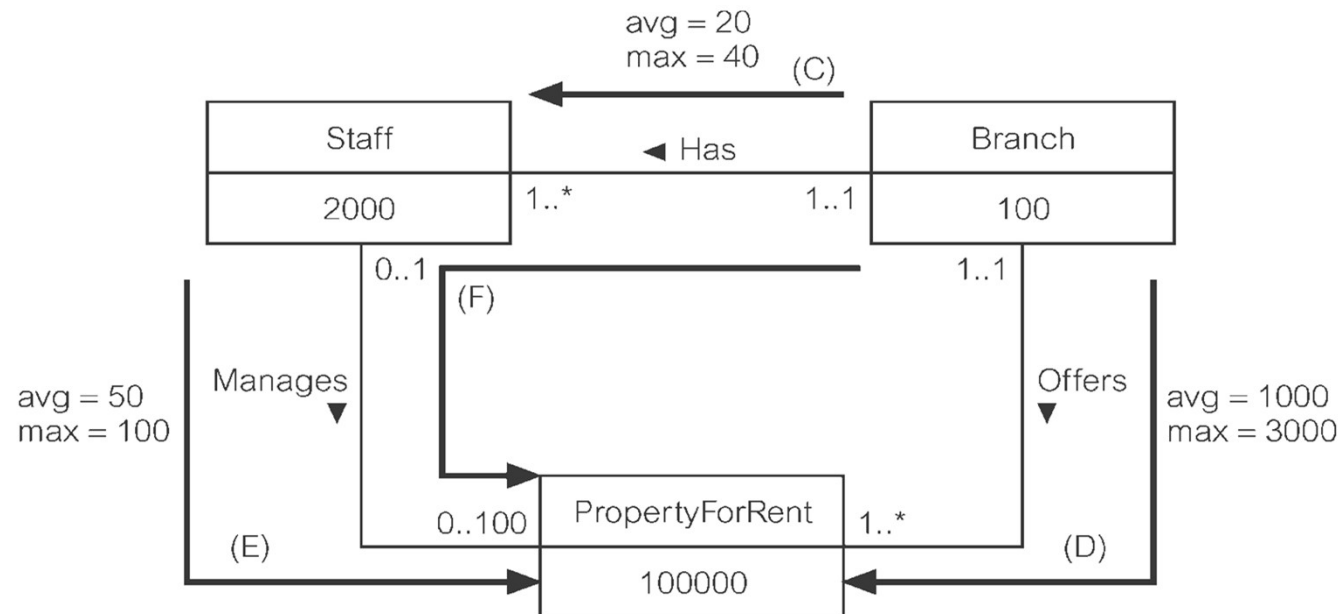
X

Dodatno lahko zapišemo število operacij na časovno enoto

I = Insert; R = Read; U = Update; D = Delete

(F) Identify the total number of properties assigned to each member of staff at a given branch.

Primer diagrama uporabe transakcij



- V specifikaciji zahtev je ocenjeno:
 - 100.000 nepremičnin;
 - 2.000 zaposlenih v 100 agencijah (branch);
 - Vsaka agencija ima povprečno 1.000, maksimalno 3.000 nepremičnin
- Zanima nas pričakovana oz. povprečna (avg) in maksimalna ocena (max) potreb

K4.2 – Izbira datotečne organizacije (ENGINE)

- Namen: izbrati učinkovito datotečno organizacijo za vse osnovne relacije.
- Datotečne organizacije (pogojujejo vrsto indeksa):
 - Kopica (Heap)
 - Hash (samo primerjava enakosti)
 - Metoda indeksiranega zaporednega dostopa (Indexed Sequential Access Method - ISAM) (tudi primerjava intervalov)
 - Drevo B+ (tudi primerjava intervalov)
 - Gručni index (Cluster – shrani podatke urejeno, ponavadi po primarnem ključu).
- Različni SUPB-ji podpirajo različne nabore datotečnih organizacij (MySQL: InnoDB, myISAM, FEDERATED, MEMORY, BLACKHOLE, ...).

K4.3 – Izbira indeksov...

- Namen: ugotoviti, ali lahko z dodatnimi indeksi povečamo učinkovitost sistema: tipično $O(n) \rightarrow O(\log n)$
- Običajen pristop:
 - Zapise na disku pustimo neurejene.
 - Izdelamo primarni indeks (po PK) in toliko sekundarnih indeksov, kolikor je potrebno.

Primarni indeks = indeks po primarnem ključu tabele. Vsak zapis ima svojo enolično vrednost.

Sekundarni indeks = indeks po atributih, ki niso primarni ključ. Vrednosti se lahko ponavljajo.

K4.3 – Izbira indeksov...

- Alternativni pristop (razmeroma redko uporabljen)
 - Zapise **uredimo** glede na primarni ključ ali indeks gruče. V slednjem primeru kot atribut za urejanje izberemo:
 - Atribut, ki se največkrat uporablja za povezovanja ali
 - Atribut, ki se najpogosteje uporablja za dostop do podatkov v relaciji.
 - Če je izbrani atribut za urejanje primarni ključ, potem s tem implementiramo primarni indeks sicer pa indeks gruče.

Indeks gruče = indeks po atributih, ki so obenem tudi atributi, po katerih je urejena tabela, niso pa nujno PK. Iskalni ključ ni nujno unikaten (če ni PK).

- Tabela ima lahko tako primarni indeks kot indeks gruče

K4.3 – Izbira indeksov...

- Sekundarni indeksi so način, kako omogočiti učinkovito iskanje tudi po drugih atributih.
- Pri določanju sekundarnih indeksov upoštevamo:
 - Povečanje učinkovitosti (predvsem pri iskanju po PB)
 - Dodatno delo, ki ga mora sistem opravljati za vzdrževanje indeksov. To vključuje:
 - Dodajanje zapisa v vsak sekundarni indeks, kadarkoli dodamo nek zapis v osnovno relacijo
 - Spreminjanje sekundarnega indeksa vsakokrat, ko se osnovna relacija spremeni
 - Povečanje porabe prostora v sekundarnem pomnilniku
 - Povečanje časovnega obsega za optimizacijo poizvedb zaradi preverjanja vseh sekundarnih indeksov.

K4.3 – Izbira indeksov...

- Nekaj smernic za uporabo sekundarnih indeksov:
 - Ne indeksiraj majhnih tabel (presežek nalaganja in vzdrževanja indeksov).
 - Če tabela ni urejena po primarnem ključu, potem kreiraj indeks na osnovi primarnega ključa (ponavadi avtomatsko).
 - Če je tuji ključ pogosto v uporabi, dodaj sekundarni indeks na tuji ključ.
 - Sekundarni indeks dodaj vsakemu atributu, ki se pogosto uporablja kot iskalni ključ.
 - Sekundarne indekse dodaj atributom, ki nastopajo v pogojih za selekcijo ali stik: ORDER BY; GROUP BY ali v drugih operacijah, ki vključujejo sortiranje (npr. UNION ali DISTINCT).

K4.3 – Izbira indeksov

- Nekaj smernic za uporabo sekundarnih indeksov (nadaljevanje):
 - Dodaj sekundarni indeks atributom, ki nastopajo v vgrajenih funkcijah;
 - Izogibaj se indeksiranju atributov, ki se pogosto spreminjajo.
 - Izogibaj se indeksiranju atributov v tabelah, nad katerimi se bodo pogosto izvajale poizvedbe, ki bodo vključevale večji del zapisov.
 - Izogibaj se indeksiranju atributov, ki so predstavljeni z daljšimi nizi znakov.

Redundanca v indeksih

- Več indeksov nad tabelo
 - hitrejši dostop do podatkov na različne načine
 - poraba časa za vzdrževanje indeksov
 - poraba prostora za hranjenje indeksov („čez palec“ lahko ocenimo velikost indeka \approx velikost tabele)
- Npr. indeksi po naravnih ključih (primarnih, tujih)
 - Sestavljeni ključi iz več atributov
 - Pogosto ponavljajoče se vrednosti
- Možnost kompresije???

Tabela MERITEV

merilnik	datum zajema	trajanje	parameter	podparameter	vrednost
E405	2020-08-04 04:00:00	00:10:00.000	18248	minv	124,40
E405	2020-08-04 04:00:00	00:10:00.000	18248	cmin	0,01
E405	2020-08-04 04:00:00	00:10:00.000	18248	vred	124,40
E405	2020-08-04 04:00:00	00:10:00.000	18248	std	0,40
E405	2020-08-04 04:00:00	00:10:00.000	18249	minv	4,78
E405	2020-08-04 04:00:00	00:10:00.000	18249	cmin	0,01
E405	2020-08-04 04:00:00	00:10:00.000	18249	vred	4,80
E405	2020-08-04 04:00:00	00:10:00.000	18249	std	0,01
E405	2020-08-04 04:00:00	00:10:00.000	18263	minv	49,50
E405	2020-08-04 04:00:00	00:10:00.000	18263	cmin	0,00
E405	2020-08-04 04:00:00	00:10:00.000	18263	vred	49,70
E405	2020-08-04 04:00:00	00:10:00.000	18263	std	0,05
E405	2020-08-04 04:10:00	00:10:00.000	15489	cmax	0,01
E405	2020-08-04 04:10:00	00:10:00.000	15489	minv	17,80
E405	2020-08-04 04:10:00	00:10:00.000	15489	cmin	0,01
E405	2020-08-04 04:10:00	00:10:00.000	15489	vred	17,90
E405	2020-08-04 04:10:00	00:10:00.000	15489	std	0,10
E405	2020-08-04 04:10:00	00:10:00.000	15488	cmax	0,01
E405	2020-08-04 04:10:00	00:10:00.000	15488	minv	26,00
E405	2020-08-04 04:10:00	00:10:00.000	15488	cmin	0,01
E405	2020-08-04 04:10:00	00:10:00.000	15488	vred	26,00
E405	2020-08-04 04:10:00	00:10:00.000	15488	std	0,10
E405	2020-08-04 04:10:00	00:10:00.000	15487	cmax	0,01
E405	2020-08-04 04:10:00	00:10:00.000	15487	minv	33,20
E405	2020-08-04 04:10:00	00:10:00.000	15487	cmin	0,01
E405	2020-08-04 04:10:00	00:10:00.000	15487	vred	33,20
E405	2020-08-04 04:10:00	00:10:00.000	15487	std	0,00
E405	2020-08-04 04:10:00	00:10:00.000	15490	cmax	0,01
E405	2020-08-04 04:10:00	00:10:00.000	15490	minv	973,00
E405	2020-08-04 04:10:00	00:10:00.000	15490	cmin	0,01
E405	2020-08-04 04:10:00	00:10:00.000	15490	vred	974,00
E405	2020-08-04 04:10:00	00:10:00.000	15490	std	0,50
E405	2020-08-04 04:10:00	00:10:00.000	18243	cmax	0,00
E405	2020-08-04 04:10:00	00:10:00.000	18243	minv	7,00
E405	2020-08-04 04:10:00	00:10:00.000	18243	cmin	0,00
E405	2020-08-04 04:10:00	00:10:00.000	18243	vred	7,50
E405	2020-08-04 04:10:00	00:10:00.000	18243	std	0,20

Kompresija indeksov

- Oracle:

```
CONSTRAINT PK_MERITEV
    PRIMARY KEY (MERILNIK, DATUM_ZAJEMA, TRAJANJE, PARAMETER, PODPARAMETER)
    USING INDEX (CREATE UNIQUE INDEX IDX1_MERITEV ON
        MERITEV (MERILNIK, DATUM_ZAJEMA, TRAJANJE, PARAMETER, PODPARAMETER)
        COMPRESS ADVANCED HIGH LOCAL) -- ali LOW
```

- Ogromni prihranki (3GB → 500 MB, HIGH; 1.2 GB, LOW)
- Možna tudi preprosta *prefiksna kompresija*
- Komprimiran datotečni sistem (npr. ZFS, 2-4x prihranek)
- PostgreSQL: indeksi nad stolpčno organiziranimi tabelami (bolje za kompresijo v npr. ZFS), razširitev Citus columnar
- Oracle/PostgreSQL/MariaDB/MySQL podpirajo tudi kompresijo vrstic (ali stolpcev v primeru stolpčne organizacije) **tabele**

K4.4 – Ocena velikosti podatkovne baze

- Namen: oceniti, koliko prostora v sekundarnem pomnilniku zahteva načrtovana podatkovna baza.
- Ocena je odvisna
 - od velikosti in števila zapisov
 - od velikosti, števila in tipa indeksov
 - od ciljnega SUPB
 - ... še od česa?
- Programska podpora: ocena velikosti podatkovne baze s pomočjo orodja PowerDesigner.



Metoda načrtovanja fizične PB...

- **Možni koraki načrtovanja fizične PB:**
 - K3 – Pretvori logični model v jezik za ciljni SUPB
 - K3.1 – Izdelaj načrt osnovnih relacij
 - K3.2 – Izdelaj načrt predstavitve izpeljanih atributov
 - K3.3 – Izdelaj načrt splošnih omejitev
 - K4 – Izdelaj načrt datotečne organizacije ter indeksov
 - K4.1 – Analiziraj transakcije
 - K4.2 – Izberi datotečno organizacijo
 - K4.3 – Določi indekse
 - K4.4 – Oцени velikost podatkovne baze
 - K5 – Izdelaj načrt uporabniških pogledov
 - K6 – Izdelaj načrt varnostnih mehanizmov
 - K7 – Preveri smiselnost uvedbe nadzorovane redundance podatkov (denormalizacija)

K5 – Načrt uporabniških pogledov

- Namen: izdelati načrt uporabniških pogledov, ki so bili opredeljeni v okviru zajema uporabniških zahtev.
- Uporabimo mehanizem pogledov (view).
- Pogled je navidezna relacija, ki fizično ne obstaja v PB, temveč se vsakokratno kreira s pomočjo poizvedbe.
- Poglede lahko modeliramo na nivoju konceptualnega modela kot entitetne tipe
 - PowerDesigner omogoča uporabo posebnega vizualnega elementa

K6 – Načrt varnostnih mehanizmov

- Namen: izdelati načrt dostopno-varnostnih mehanizmov skladno z zahtevami naročnika.
- SUPB-ji tipično podpirajo dve vrsti dostopne varnosti:
 - varnost dostopa in uporabe podatkovne baze (navadno zagotovljeno s pomočjo uporabniških imen in gesel);
 - varnost uporabe podatkov – kdo lahko uporablja določene relacije ter kako.
- Med SUPB-ji so velike razlike v mehanizmih, ki jih imajo na voljo za dosego varnosti.
- GDPR, ZVOP-2 !!!!!

K7 – Uvedba nadzorovane redundance...

- Namen: ugotoviti, ali je smiselno dopustiti določeno mero redundance (denormalizacija) ter tako izboljšati učinkovitost.
- Rezultat normalizacije je načrt, ki je po strukturi konsistenten ter minimalen.
- Včasih normalizirane relacije ne dajo zadovoljive učinkovitosti.
- Razmislimo, ali se zaradi izboljšanja učinkovitosti odpovemo določenim koristim, ki jih prinaša normalizacija.

K7 – Uvedba nadzorovane redundance...

- Upoštevamo tudi:
 - Implementacija denormaliziranih relacij je težja;
 - Z denormalizacijo velikokrat zgubimo na prilagodljivosti modela;
 - Denormalizacija navadno pospeši poizvedbe, vendar upočasni spreminjanje podatkov.

K7 – Uvedba nadzorovane redundance...

▪ Denormalizacija:

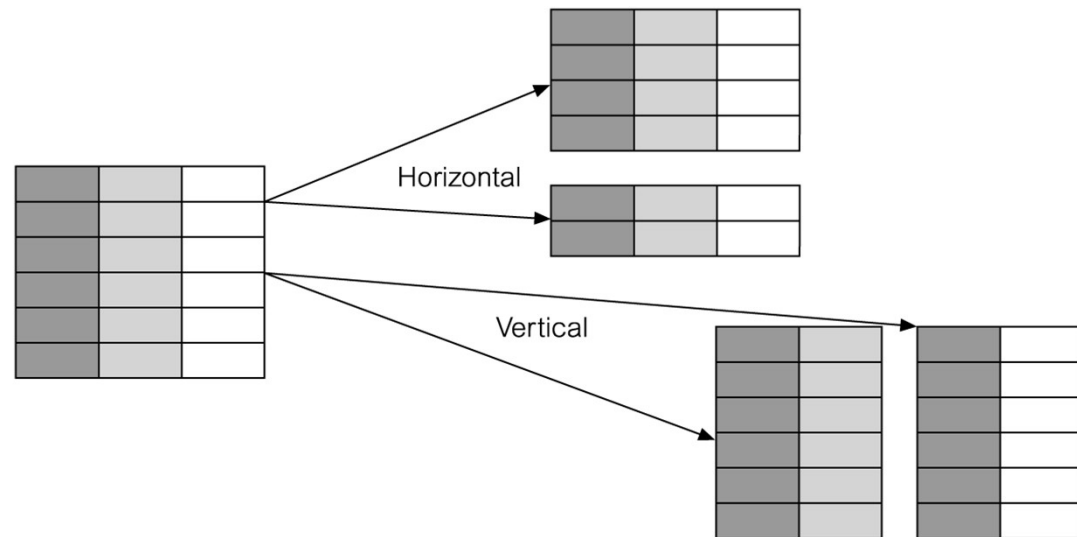
- Denormalizacija se nanaša na dopolnitev relacijske sheme, tako da eni ali več relacij znižamo stopnjo normalne oblike (npr. 3.NO → 2.NO).
- Nanaša se tudi na primere, ko dve normalizirani relaciji združimo v eno, ki je še vedno normalizirana (3. NO, BCNO), vendar zaradi združitve vsebuje več nedefiniranih vrednosti (NULL) oz. večvrednostne odvisnosti (npr. 4.NO → 3.NO).
- Tipični razlogi: stične operacije, hierarhije, izračun atributov
- Nekateri vidiki denormalizacije so podprti tudi v SUPB

K7 – Uvedba nadzorovane redundance...

- Koraki denormalizacije:
 - K7.1 – združevanje 1:1 povezav (če podatke istočasno uporabljamo)
 - K7.2 – Podvajanje neosnovnih atributov v povezavah 1:več za zmanjšanje potrebnih stikov (dodamo attribute, ki izvirajo iz drugih entitetnih tipov oz. relacij)
 - K7.3 – Podvajanje tujih ključev v 1:več povezavah za zmanjšanje potrebnih stikov (dodajanje tranzitivnih povezav).
 - K7.4 – Podvajanje atributov v več:več povezavah za zmanjšanje potrebnih stikov.
 - K7.5 – Uvedba ponavljajočih skupin atributov za zmanjšanje potrebnih stikov.
 - K7.6 – Kreiranje tabel, ki predstavljajo izvleček osnovne tabele.
 - K7.7 – Razbitje (particioniranje) velikih tabel (horizontalno, vertikalno).

Partitioniranje relacij

- Za povečanje učinkovitosti nad tabelami z zelo velikim številom vrstic (ali stolpcev) uporabimo pristop, kjer relacijo razbijemo na manjše dele - particije.
- Poznamo dva načina delitve na particije:
 - Horizontalni in
 - Vertikalni (1:1)



Prednosti razbitja relacije na particije

- Uporaba particioniranja prinaša številne prednosti:
 - Boljša porazdelitev obremenitev (load balancing)
 - Možnost uporabe lokalnih ali globalnega indeksa (particioniranje indeksov)
 - Večja učinkovitosti (manj podatkov za obdelavo, paralelnost izvajanja,...)
 - Boljša razpoložljivost
 - Boljša obnovljivost
 - Več možnosti za zagotavljanje varnosti

Slabosti razbitja relacije na particije

- Particioniranje ima tudi slabosti:
 - Kompleksnost (particije niso vedno transparentne za uporabnike...)
 - Slabša učinkovitost (če je potrebno poizvedovati istočasno po več particijah, je učinkovitost občutno slabša)
 - Podvajanje podatkov (pri vertikalnem particioniranju, ključni potrebni za povezovanje)
- Ročno ali avtomatsko dodajanje novih particij
 - Nadležno ročno kreiranje novih particij (klic procedure ob zunanjem dogodku, npr. prehod v novo leto)
 - Avtomatsko kreiranje ob prihodu nove vrednosti relativno slabo podprto (le Oracle, pa še to precej nerodno)

Horizontalno particioniranje v MariaDB 10+ in MySQL 5.6+

```
CREATE TABLE employees (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  fname VARCHAR(25) NOT NULL,
  lname VARCHAR(25) NOT NULL,
  store_id INT NOT NULL,
  department_id INT NOT NULL
)
-- RANGE
PARTITION BY RANGE(id) (
  PARTITION p0 VALUES LESS THAN (5),
  PARTITION p1 VALUES LESS THAN (10),
  PARTITION p2 VALUES LESS THAN (15),
  PARTITION p3 VALUES LESS THAN MAXVALUE
);
-- HASH
PARTITION BY HASH( func(store_id) )
PARTITIONS 4;
```

- BY LIST (en atribut)
VALUES IN (seznam), DEFAULT
 - NULL gre v DEFAULT
- BY RANGE (en atribut)
VALUES LESS THAN (x), MAXVALUE
 - NULL gre v najmanjšo skupino
- BY KEY(atributi) (avt. zgoščevanje)
 - št. particij: ... PARTITIONS n;
- BY HASH(funkcija(atrib.)) (zgoščevanje)
 - izbor hash funkcije (npr. MD5)
 - št. particij: ... PARTITIONS n;
- SUBPARTITION (gnezdenje)
- BY RANGE/LIST COLUMNS (po več atributih istočasno), poseben primer podparticij

Shard = horizontalna particija

Avtomatsko particioniranje

- Oracle: avtomatsko particioniranje seznamov ali po intervalih (sintaksa zahteva ročno specifikacijo vsaj ene particije)
 - ... `PARTITION BY LIST (atribut) AUTOMATIC`
`(PARTITION part_null VALUES (NULL)); -- Ni nujno NULL`
 - ... `PARTITION BY RANGE(datum) INTERVAL (INTERVAL '1' YEAR)`
`(PARTITION part_null VALUES LESS THAN DATE'2020-01-01');`
- Tradicionalno (PostgreSQL, MariaDB/MySQL)
 - Vseobsegujoča particija, npr.
... `PARTITION vseobsegujoca VALUES LESS THAN MAXVALUE # RANGE`
... `PARTITION vseobsegujoca DEFAULT # LIST`
 - Vnaprej predvideno dodajanje novih particij (npr. po koledarju za vsak mesec)
 - Periodično ali dogodkovno proženje procedure, ki po nekem kriteriju reparticionira vseobsegujočo particijo (npr. nova particija za vsako unikatno vrednost)
 - Samo lokalni indeksi (glede na particije)! Oracle ima lahko tudi globalni indeks.

Implementacija nadzorovane redundance (denormalizacija)

- Včasih zavestno uporabljamo relacije, ki ne ustrezajo najvišjim normalnim oblikam.
- Prve in druge normalne oblike nikoli ne kršimo.
- Višjim normalnim oblikam se včasih odrečemo zaradi dejanskega poznavanja problematike, doseganja boljše učinkovitosti ali ohranjanja omejitev (funkcionalnih odvisnosti).

Denormalizacija in SQL

- Za obravnavo denormaliziranih relacij imamo v SQL več možnosti
 - Če dejanske funkcionalne odvisnosti ne poznamo, vemo pa za njen obstoj
 - preverjanje spoštovanja omejitve (s pomočjo baznih omejitev ali prožilcev)
 - Če dejansko funkcionalno odvisnost poznamo in jo lahko učinkovito izračunamo
 - izračun funkcionalno odvisnih atributov s pogledom ali baznim prožilcev
- Pri vseh naštetih možnosti nam zelo koristijo shranjeni podprogrami (**stored procedures**)

Shranjeni podprogrami v SUPB

- Shranjeni podprogrami: procedure in funkcije, ki jih pogosto potrebujemo
- Najpogosteje so to **poimenovani SQL bloki**, ki jih lahko kličemo s parametri
- Lahko spreminjajo podatke ali vračajo rezultate
- Žal so implementacije **pogosto sistemsko odvisne**

Shranjeni podprogrami v SUPB

- Omogočajo modularno in razširljivo pisanje programov
 - Funkcija: vrne natanko eno vrednost kot rezultat
 - Procedura: ne vrača vrednosti, ali pa jo vrača v izhodnih (OUT) argumentih
- Različni programski jeziki, ne samo SQL
 - Oracle: PL/SQL, Java
 - PostgreSQL:
 - Standardno: **PL/pgSQL**, PL/Tcl, PL/Perl, **PL/Python**
 - Dodatno (razširitve, definiran API): **PL/Java**, PL/PHP, PL/Ruby, **PL/R**, PL/sh (Linux shell), C, C++, Rust, ...

Shranjeni podprogrami v SQL

- Parametri (predvsem v procedurah)
 - vhodni (IN)
 - izhodni(OUT)
 - vhodno-izhodni (IN OUT)
- Pogosto potrebna uporaba postopkovnih dodatkov: spremenljivke, kurzorji, ...
 - ISO/ANSI: SQL/PSM (Persistant Stored Modules).
 - Oracle: PL/SQL, Microsoft: T-SQL (Transact SQL)
- Deklaracija in uporaba podprogramov

```
CREATE PROCEDURE
```

```
    Test (a IN VARCHAR(10)) AS ... ;
```

```
CALL ali EXECUTE Test('abcd');
```

```
DROP PROCEDURE Test;
```

```
Jadralec(jid, ime, rating, starost)
Coln(cid, ime, dolzina, barva)
Rezervacija(jid, cid, dan)
```

Primer izračuna shranjenega atributa

- V tabelo jadralec dodamo število rezervacij za vsakega jadralca.

```
ALTER TABLE jadralec
  ADD stRez INTEGER DEFAULT 0 NOT NULL;
```

- Kako izračunamo vrednost tega atributa?

```
UPDATE jadralec j
  SET stRez =
    (SELECT COUNT(*)
     FROM rezervacija r
     WHERE r.jid = j.jid);
```

Kdaj je vse
to zares
potrebno
izračunati?

Primer procedure (Oracle)

- Inicializiraj število rezervacij na poljubno vrednost (parameter).

```
CREATE PROCEDURE JADR_REZ_INIT
( INIT IN INTEGER DEFAULT 0 ) AS
BEGIN
    UPDATE jadralec j
        SET j.stRez = INIT;
END;

CALL JADR_REZ_INIT(0);
```

```
Jadralec(jid, ime, rating, starost)
Coln(cid, ime, dolzina, barva)
Rezervacija(jid, cid, dan)
```

```
Jadralec(jid, ime, rating, starost)
Coln(cid, ime, dolzina, barva)
Rezervacija(jid, cid, dan)
```

Primer procedure (MySQL)

- Inicializiraj število rezervacij na poljubno vrednost (parameter).

```
DELIMITER //
CREATE PROCEDURE JADR_REZ_INIT
(IN INIT INTEGER)
BEGIN
    UPDATE jadralec j
    SET j.stRez = INIT;
END//
DELIMITER ;
CALL JADR_REZ_INIT(0);
```

```
Jadralec(jid, ime, rating, starost)
Coln(cid, ime, dolzina, barva)
Rezervacija(jid, cid, dan)
```

Primer procedure (Oracle)

```
Jadralec(jid, ime, rating, starost)
Coln(cid, ime, dolzina, barva)
Rezervacija(jid, cid, dan)
```

- Izračunaj dejansko število rezervacij.

```
CREATE PROCEDURE JADR_REZ AS
BEGIN
  UPDATE jadralec j
  SET stRez = ( SELECT COUNT(*)
                FROM rezervacija r
                WHERE r.jid = j.jid );
END;

CALL JADR_REZ();
```

Primer procedure (MySQL)

```
Jadralec(jid, ime, rating, starost)
Coln(cid, ime, dolzina, barva)
Rezervacija(jid, cid, dan)
```

- Izračunaj dejansko število rezervacij.

```
DELIMITER //
```

```
CREATE PROCEDURE JADR_REZ()
```

```
BEGIN
```

```
    UPDATE jadralec j
```

```
        SET stRez = ( SELECT COUNT(*)
```

```
                        FROM rezervacija r
```

```
                        WHERE r.jid = j.jid );
```

```
END//
```

```
CALL JADR_REZ();
```

Primer procedure in funkcije (Oracle)

```
Jadralec(jid, ime, rating, starost)  
Coln(cid, ime, dolzina, barva)  
Rezervacija(jid, cid, dan)
```

- Izračunaj dejansko število rezervacij.

```
CREATE FUNCTION JADR_REZ_FUNC(JJID IN INTEGER) RETURN INTEGER AS  
  x INTEGER; -- Lokalna spremenljivka  
BEGIN  
  SELECT COUNT(*) INTO x  
  FROM rezervacija r  
  WHERE r.jid = jjid;  
  RETURN x;  
END;
```

```
CREATE PROCEDURE JADR_REZ AS  
BEGIN  
  UPDATE jadralec j  
  SET stRez = JADR_REZ_FUNC(j.jid);  
END;
```

Primer procedure in funkcije (MySQL)

```
Jadralec(jid, ime, rating, starost)
Coln(cid, ime, dolzina, barva)
Rezervacija(jid, cid, dan)
```

- Izračunaj dejansko število rezervacij.

```
DELIMITER //
```

```
CREATE FUNCTION JADR_REZ_FUNC (JJID IN INTEGER) RETURNS INTEGER AS
```

```
BEGIN
```

```
    DECLARE x INTEGER;           -- Lokalna spremenljivka
```

```
    SELECT COUNT(*) INTO x
```

```
        FROM rezervacija r
```

```
        WHERE r.jid = jjid;
```

```
    RETURN x;
```

```
END//
```

```
CREATE PROCEDURE JADR_REZ()
```

```
BEGIN
```

```
    UPDATE jadralec j
```

```
        SET stRez = JADR_REZ_FUNC(j.jid);
```

```
END//
```

Bazni prožilci (triggerji)

- Prožilec: sestavljen SQL stavek, podobne oblike kot shranjena procedura, vendar nima argumentov
- Izvede se avtomatsko kot stranski produkt spremembe neke poimenovane tabele
- Ne kličemo ga ročno, ampak ga sproži prožilni dogodek
- Uporaba:
 - preverjanje pravilnosti vnosev in integritetnih omejitev (tudi denormalizacija)
 - opozarjanje na potrebne uporabniške akcije ob spremembah
 - vzdrževanje seznamov sprememb v PB
- ISO standard: stavčni in vrstični prožilci
- Oracle: sestavljeni (COMPOUND) prožilci

Sintaksa prožilcev dogodkov nad tabelami

- ISO standard:

```
CREATE TRIGGER
```

```
BEFORE | AFTER dogodek ON tabela [OR ...]
```

```
[REFERENCING sinonimi za stare ali nove vrednosti]
```

```
[FOR EACH ROW]
```

```
[WHEN (pogoj)] -- pogoj za vrstico (kot WHERE)
```

```
BEGIN
```

```
    -- telo prožilca
```

```
END;
```

- Dogodki: INSERT, UPDATE, DELETE

Posebnost: PostgreSQL

```
CREATE TRIGGER name BEFORE | AFTER | INSTEAD OF event [  
  OR ... ]  
  ON table_name  
  FOR EACH ROW EXECUTE FUNCTION tf();
```

INSTEAD OF: za npr. delo s pogledi, namesto dogodka izvede funkcijo

```
CREATE OR REPLACE FUNCTION tf()  
  RETURNS TRIGGER  
  LANGUAGE plpgsql AS $$  
  BEGIN  
    ...  
  END;  
  $$;
```

\$\$ označuje “Dollar-Quoted String Constant”: večvrstični niz znakov, podobno kot delimiter v MySQL.

Posebni CONSTRAINT triggerji: se podrejajo SET CONSTRAINTS nastavitvam (transakcije; npr. odloženo preverjanje)

Stavčni prožilci

- Stavčni prožilec se izvede le enkrat na stavek, ki spremeni tabelo
- Oracle, PostgreSQL: stavčni prožilci so privzeti.
- MySQL: ne podpira stavčnih prožilcev (samo vrstične).

Primer stavčnega prožilca

```
Jadralec(jid, ime, rating, starost)
Coln(cid, ime, dolzina, barva)
Rezervacija(jid, cid, dan)
```

```
CREATE TRIGGER IzracunajSteviloRezervacij
AFTER INSERT OR UPDATE OR DELETE ON rezervacija
-- za vsak stavek (ne MySQL)
BEGIN          -- PL/SQL blok
  UPDATE jadralec
  SET stRez =
    ( SELECT COUNT(*)
      FROM rezervacija
      WHERE rezervacija.jid = jadralec.jid)
END;
```

Vrstični prožilci

- Vrstični prožilec se izvede za vsako spremenjeno vrstico
- Odvisno od vrste dogodka lahko referenciramo
 - stare vrednosti pred spremembo (OLD): DELETE, UPDATE
 - nove vrednosti po spremembi (NEW): INSERT, UPDATE
 - Oracle: v WHEN sklopu OLD in NEW uporabljamo normalno, znotraj BEGIN/END pa z dvopičjem :OLD, :NEW
 - Oracle: z REFERENCING sklopom lahko OLD in NEW preimenujemo
 - Oracle: zastavice INSERTING, UPDATING, DELETING
- Prednost vrstičnih prožilcev: izvedemo telo prožilcev samo za vrstice, ki so se zares spremenile
- Nerodno: pogosto moramo za vsako vrsto dogodka napisati svoj prožilec (zelo podoben ostalim), pomagajo npr. Oracle zastavice

Primer vrstičnega prožilca (INSERT)

```
Jadralec(jid, ime, rating, starost)
Coln(cid, ime, dolzina, barva)
Rezervacija(jid, cid, dan)
```

```
CREATE TRIGGER IzracunajSteviloRezervacij_I
AFTER INSERT ON rezervacija
REFERENCING NEW AS nova          -- Alias za NEW
FOR EACH ROW  -- za vsako novo vrstico
BEGIN  -- PL/SQL blok
    UPDATE jadralec
    SET stRez = stRez +1
    WHERE jadralec.jid = :nova.jid;
END;
```

Primer vrstičnega prožilca (DELETE)

```
Jadralec(jid, ime, rating, starost)
Coln(cid, ime, dolzina, barva)
Rezervacija(jid, cid, dan)
```

```
CREATE TRIGGER IzracunajSteviloRezervacij_D
AFTER DELETE ON rezervacija
REFERENCING OLD AS stara          -- Alias za OLD
FOR EACH ROW  -- za vsako novo vrstico
BEGIN  -- PL/SQL blok
    UPDATE jadralec
    SET stRez = stRez -1
    WHERE jadralec.jid = :stara.jid;
END;
```

Primer vrstičnega prožilca (UPDATE)

```
Jadralec(jid, ime, rating, starost)
Coln(cid, ime, dolzina, barva)
Rezervacija(jid, cid, dan)
```

```
CREATE TRIGGER IzracunajSteviloRezervacij_U
AFTER UPDATE ON rezervacija
REFERENCING OLD AS stara NEW AS nova
FOR EACH ROW -- za vsako novo vrstico
WHEN (stara.jid != nova.jid)
BEGIN -- PL/SQL blok
    UPDATE jadralec
    SET stRez = stRez + 1
    WHERE jadralec.jid = :nova.jid;
    UPDATE jadralec
    SET stRez = stRez - 1
    WHERE jadralec.jid = :stara.jid;
END;
```

MySQL: shranjene procedure in prožilci

- Spremenimo ločilo za konec stavka (namesto podopičja):
npr. DELIMITER //
- Razlike pri parametrih: IN, OUT, INOUT pred imenom
npr. (IN INIT INTEGER) samo za procedure, ni privzetih vrednosti
- Deklaracija lokalnih spremenljivk znotraj BEGIN/END:
npr. DECLARE x INTEGER;
- Ne uporablja AS, RETURNS namesto RETURN
- Ni stavčnih prožilcev, ni aliasov za OLD in NEW
- Ne uporabljamo dvopičja: OLD namesto :OLD
- Ne pozna WHEN sklopa (lahko pa uporabimo proceduralni IF ... END)
- Samo en dogodek na prožilec (INSERT, UPDATE, DELETE)

MySQL: primer vrstičnega prožilca (INSERT)

```
Jadralec(jid, ime, rating, starost)
Coln(cid, ime, dolzina, barva)
Rezervacija(jid, cid, dan)
```

```
DELIMITER //
CREATE TRIGGER IzracunajSteviloRezervacij_I
AFTER INSERT ON rezervacija
FOR EACH ROW -- za vsako novo vrstico
BEGIN
    UPDATE jadrlec
    SET stRez = stRez + 1
    WHERE jadrlec.jid = NEW.jid;
END//
```

MySQL: primer vrstičnega prožilca (DELETE)

```
Jadralec(jid, ime, rating, starost)
Coln(cid, ime, dolzina, barva)
Rezervacija(jid, cid, dan)
```

```
DELIMITER //
CREATE TRIGGER IzracunajSteviloRezervacij_D
AFTER DELETE ON rezervacija
FOR EACH ROW -- za vsako novo vrstico
BEGIN
    UPDATE jadralec
    SET stRez = stRez -1
    WHERE jadralec.jid = OLD.jid;
END//
```

MySQL: primer vrstičnega prožilca (UPDATE)

```
Jadralec(jid, ime, rating, starost)
Coln(cid, ime, dolzina, barva)
Rezervacija(jid, cid, dan)
```

```
DELIMITER //
CREATE TRIGGER IzracunajSteviloRezervacij_U
AFTER UPDATE ON rezervacija
FOR EACH ROW -- za vsako vrstico
BEGIN
    IF NEW.jid != OLD.jid THEN
        UPDATE jadralec
        SET stRez = stRez + 1
        WHERE jadralec.jid = NEW.jid;
        UPDATE jadralec
        SET stRez = stRez - 1
        WHERE jadralec.jid = OLD.jid;
    END IF;
END//
```

Nadzorovana denormalizacija..

- Normalizirane sheme v nekaterih primerih predstavljajo oviro za učinkovito implementacijo
- Primer:
 - Rezultat(Startna številka, Ime, Priimek, Cas_Prvi_Tek, Cas_Drugi_Tek, Cas_Skupaj)
 - Ta relacija ni v tretji normalni obliki:
Cas_Prvi_Tek, Cas_Drugi_Tek → Cas_Skupaj
 - Ni je potrebno dekomponirati v tretjo normalno obliko, pod pogojem, da kontroliramo ali izračunavamo vsebino tako, da nikoli ne more biti v *Cas_Skupaj* kaj drugega, kot vsota.

Preverjanje denormalizacije z omejitvijo

```
ALTER TABLE Rezultat
ADD CONSTRAINT PreveriSkupniCas
CHECK
    (NOT EXISTS
        (SELECT Startna_stevilka
        FROM Rezultat
        WHERE Cas_Skupaj != Cas_Prvi_Tek + Cas_Drugi_Tek
        ));
```

Vpisujemo vse tri attribute: Cas_Prvi_Tek, Cas_Drugi_Tek, Cas_Skupaj, omejitev pa ob vsaki spremembi tabele kontrolira vse vrstice, ce smo res povsod vnesli vsoto.

Implementacija denormalizacije s prožilcem

```
CREATE TRIGGER IzracunajSkupniCas
AFTER INSERT OR UPDATE ON Rezultat
FOR EACH ROW -- za vsako dodano ali spremenjeno vrstico
BEGIN
    UPDATE Rezultat
        SET Cas_Skupaj = Cas_Prvi_Tek + Cas_Drugi_Tek;
    WHERE :NEW.Startna_stevilka = Rezultat.Startna_stevilka;
END;
```

Vpisujemo samo neodvisna attribute: Cas_Prvi_Tek in Cas_Drugi_Tek, prožilec pa ob vsaki spremembi izračuna vsoto v spremenjeni vrstici.

Izračun atributa preko uporabe pogleda

- Osnovna relacijska shema:
Rezultat (Startna številka, Ime, Priimek, Cas_Prvi_Tek,
Cas_Drugi_Tek)

```
CREATE VIEW SkupniRezultat AS (  
  SELECT Rezultat.*, Cas_Prvi_Tek + Cas_Drugi_Tek AS Cas_Skupaj  
  FROM Rezultat  
);
```

Vpisujemo samo neodvisna attribute: Cas_Prvi_Tek in Cas_Drugi_Tek, pogled pa ob vsaki uporabi izračuna vsote v vseh vrsticah.