Načrtovanje fizične podatkovne baze

- Fizično načrtovanje PB opredeljuje proces, s katerim izdelamo opis implementacije PB na sekundarnem pomnilnem mediju.
- Opisuje
  - osnovne relacije,
  - datotečno organizacijo,
  - indekse za dosego učinkovitega dostopa do podatkov,
  - povezane omejitve in
  - varnostne mehanizme.

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 – Oceni 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 ključe?
  - Ali podpira obveznost podatkov (NOT NULL)?
  - Ali podpira domene?
  - Ali podpira pravila omejitve podatkov?
  - Ali podpira prožilce (triggers) in bazne podprograme (stored procedures)?

K3.2 – Predstavitev izpeljanih atributov...

- Namen: določiti, kako bodo v SUPB predstavljeni izpeljani 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 atribute, upoštevaj:
  - "strošek" shranjevanja in vzdrževanja skladnosti izpeljanih atributov z osnovnimi atributi, iz katerih je izpeljan;
  - "strošek" vsakokratnega izračunavanja izpeljanega atributa.
- Izberi stroškovno ugodnejšo rešitev.

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	lName	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 želeno/pričakovano učinkovitostjo transakcij.
- Med SUPB-ji velike razlike.

- Vprašanje: datotečna organizacija SUPB MySQL?

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:
  - Atribute, 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.

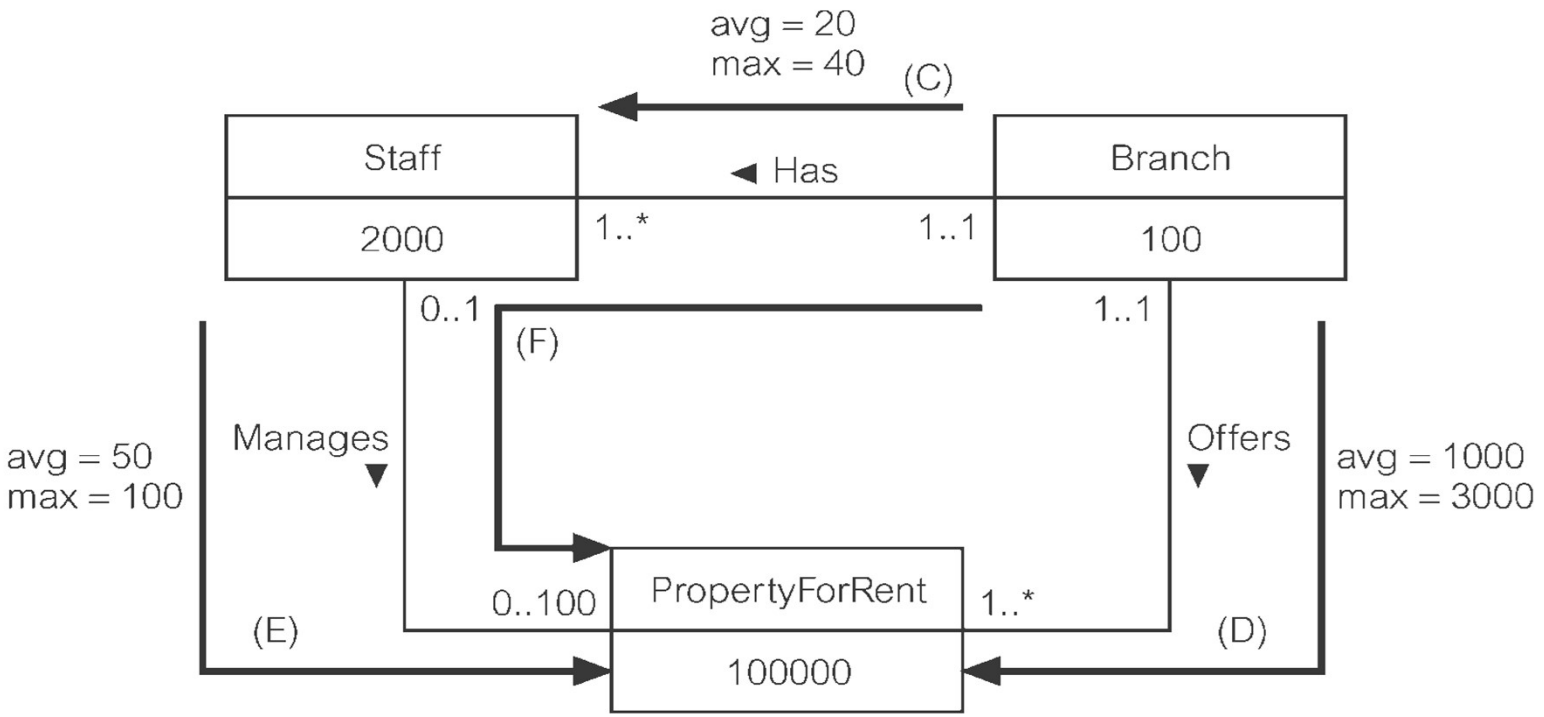
Transaction/ Relation	(A)				(B)				(C)				(D)				(E)				(F)			
	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D
Branch										X				X								X		
Telephone																								
Staff		X				X				X								X				X		
Manager																								
PrivateOwner	X																							
BusinessOwner	X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

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

Dodatno lahko zapišemo število operacij na časovno enoto

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;
- Vsaka agencija ima od 1.000 do 3.000 nepremičnin

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)
  - Gruča (Cluster – shrani podatke urejeno za uporabo B indeksa, pogosto primarni ključ).
- Različni SUPB-ji podpirajo različne nabore datotečnih organizacij.

K4.3 – Izbira indeksov...

- Namen: ugotoviti, ali lahko z dodatnimi indeksi povečamo učinkovitost sistema.
- Možen pristop:
  - Zapise pustimo neurejene.
  - Izdelamo toliko sekundarnih indeksov, kolikor je potrebno.

Sekundarni indeks = indeks po atributu, ki ni obenem tudi atribut, po katerem je urejena relacija

K4.3 – Izbira indeksov...

- Alternativni pristop
  - Zapise uredimo po primarnem ključu oz. po indeksu gruče. V tem 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 sortiranje primarni ključ, potem gre za primarni indeks sicer za indeks gruče.
  - Relacija ima lahko primarni indeks ali indeks gruče

Primarni indeks = indeks po primarnem ključu, po katerem je urejena relacija. Vsak zapis ima svojo vrednost.

Indeks gruče = indeks po atributu/ih, ki je obenem tudi atribut/i, po katerem je urejena relacija, ni pa nujno PK. Iskalni ključ ni nujno unikaten (če ni PK).

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 relacij.
  - Če datoteka ni urejena po primarnem ključu, potem kreiraj indeks na osnovi primarnega ključa.
  - Če je tuji ključ pogosto v uporabi, dodaj sekundarni indeks na tuji ključ.
  - Sekundarni indeks dodaj vsakemu atributu, ki se pogosto uporablja kot sekundarni 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 relacijah, nad katerimi se bodo pogosto izvajale poizvedbe, ki bodo vključevale večji del zapisov.
  - Izogibaj se indeksiranju atributov, ki so predstavljeni z daljšimi stringi.

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 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 – Oceni 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.

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 !!!!!

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 zavoljo 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...

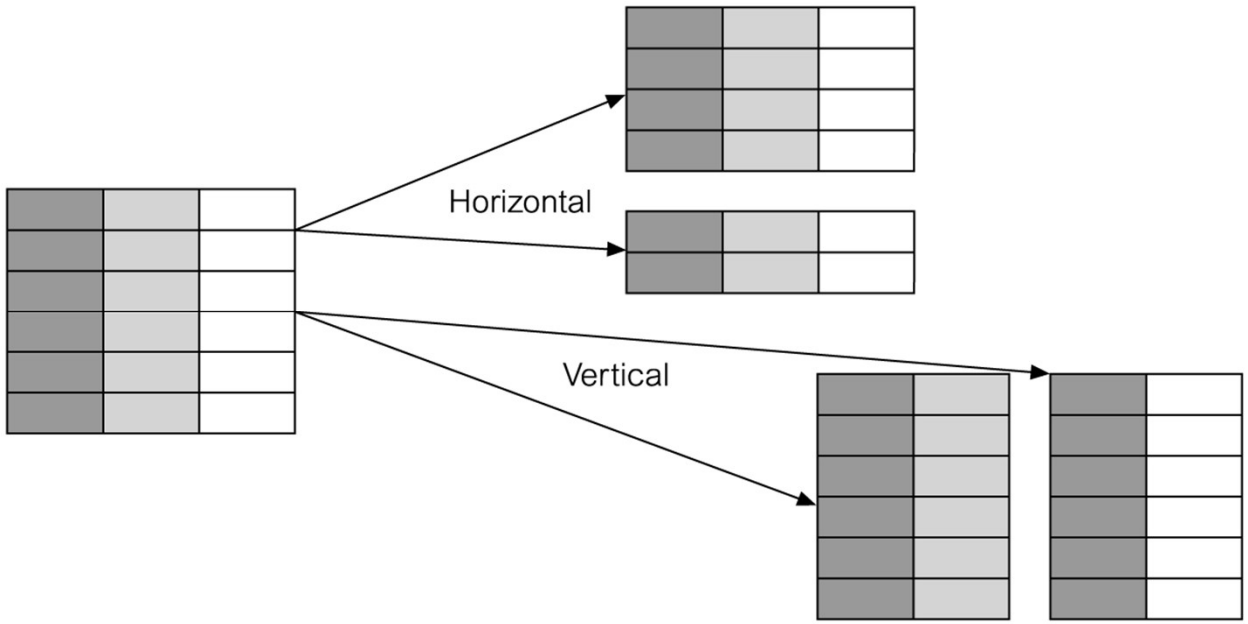
- Denormalizacija:
  - Denormaliacija se nanaša na dopolnitev relacijske sheme, tako da eni ali več relacij znižamo stopnjo normalne oblike (npr. 3.NO → 2.NO).
  - Nanaša se tudi na primere, ko dve normalizirani relaciji združimo v eno, ki je še vedno normalizirana (3. NO, BCNO), vendar zaradi združitve vsebuje več nedefiniranih vrednosti (NULL) oz. večvrednostne odvisnosti (npr. 4.NO → 3.NO).

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 atribute, 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 relacij (horizontalno, vertikalno).

Razbitje relacij

- Za povečanje učinkovitosti nad relacijami z zelo velikim številom n-teric uporabimo pristop, kjer relacijo razbijemo na manjše dele - particije.
- Poznamo dva načina delitve:
  - Horizontalni in
  - Vertikalni.



Prednosti razbitja relacije na particije

- Uporaba particioniranja prinaša številne prednosti:
  - Boljša porazdelitev obremenitev (load balancing)
  - 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 (pri poizvedbah, kjer je potrebno poizvedovati po več particijah, je učinkovitost slabša)
  - Podvajanje podatkov (pri vertikalnem particioniranju)

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(store_id)
   PARTITIONS 4;
```

- RANGE/LIST (po enem atributu)
- RANGE/LIST COLUMNS (po več atributih)
- (LINEAR) HASH ali KEY (uporabniško ali avtomatsko zgoščevanje)
- SUBPARTITION (gnezdenje)

shard=horizontal partition

Nadzorovana 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 SQL

- Shranjeni podprogrami: procedure in funkcije, ki jih pogosto potrebujemo
- Poimenovani SQL bloki, ki jih lahko kličemo s parametri
- Lahko spreminjajo podatke ali vračajo rezultate
- Funkcija: vrne natanko eno vrednost kot rezultat
- Procedura: vrača vrednost v izhodnih argumentih
- Omogočajo modularno in razširljivo pisanje programov
- Žal so implementacije pogosto sistemsko odvisne.

Shranjeni podprogrami

- 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
- Deklaracija in uporaba podprogramov

```
CREATE PROCEDURE
      Test (a IN VARCHAR(10)) AS … ;

CALL ali EXECUTE Test('abcd');

DROP PROCEDURE Test;
```

Primer izračuna shranjenega atributa

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

- 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 stRez = INIT;
END;

CALL JADR_REZ_INIT(0);
```

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

Primer procedure (MySQL)

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

- Inicializiraj število rezervacij na poljubno vrednost (parameter).

```
DELIMITER //
CREATE PROCEDURE JADR_REZ_INIT
(IN INIT INTEGER)
BEGIN
   UPDATE jadralec j
     SET 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)

- 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();
```

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

Primer procedure (MySQL)

- 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();
```

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

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 INTEGER) RETURNS INTEGER
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
- Stavčni in vrstični prožilci.

Sintaksa prožilcev dogodkov nad tabelami

- ISO standard:

```
CREATE TRIGGER
BEFORE | AFTER dogodek ON tabela
[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

Stavčni prožilci

- Stavčni prožilec se izvede le enkrat na stavek, ki spremeni tabelo
- Oracle: 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
- 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).

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 jadralec
   SET stRez = stRez +1
   WHERE jadralec.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 stevilka, Ime, Priimek, Cas_Prvi_Tek, Cas_Drugi_Tek, Cas_Skupaj)
  - Ta relacija ni niti v tretji normalni obliki:
    Cas_Prvi_Tek,Cas_Drugi_Tek $\rightarrow$ 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 atribute: 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 atribute: 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 stevilka, 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 atribute: Cas_Prvi_Tek in Cas_Drugi_Tek, pogled pa ob vsaki uporabi izračuna vsote v vseh vrsticah.