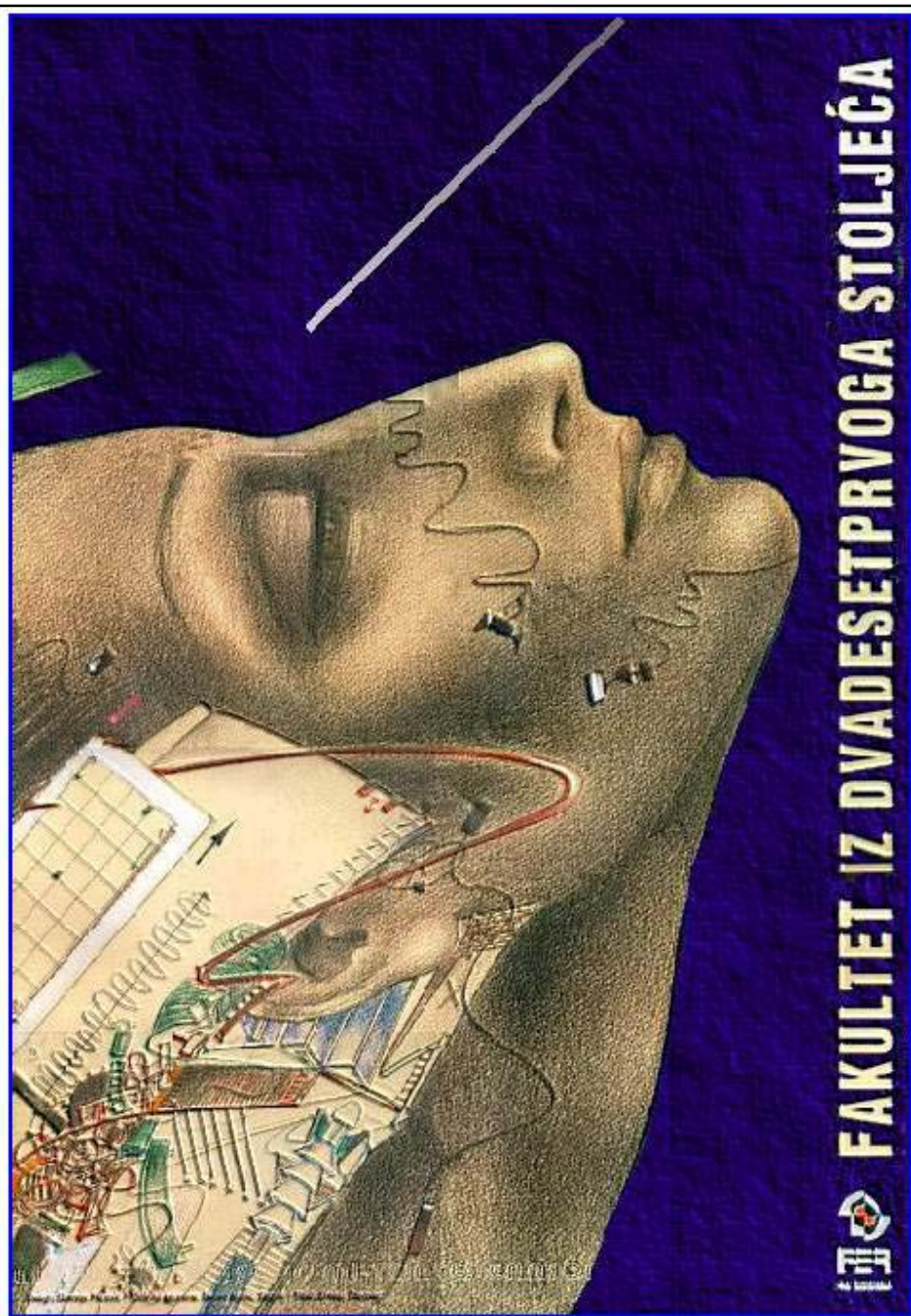


Baze podataka

Predavanja
lipanj 2008.

16. Pohranjene procedure i okidači



Pohranjene procedure

Primjer 1:

	CHAR (13)	CHAR (30)
osoba	jmbg	prez
	2203979622876	Horvat
	1712 12871211	Kolar
	2707986736233	Še5fer
	03AB621.22876	Novak

- smatra se da su ispravne one vrijednosti atributa jmbg u kojima postoji točno 13 znamenaka
- smatra se da su ispravna ona prezimena u kojima ne postoji niti jedna znamenka
- ispisati podatke o osobama s neispravnim jmbg ili prezimenom
- **kad bi barem postojala SQL funkcija CountDigits(nizZnakova)**

```
SELECT * FROM osoba
WHERE CountDigits(jmbg) <> 13
OR CountDigits(prez) > 0;
```

Pohranjene procedure (pohranjene funkcije)

- Pohranjena procedura ili pohranjena funkcija je potprogram koji je pohranjen u rječniku podataka i koji se izvršava u kontekstu sustava za upravljanje bazama podataka
 - može se promatrati kao procedura ili funkcija kojom se proširuje skup SQL funkcija ugrađenih u SUBP
 - procedura je potprogram koji u pozivajući program ne vraća rezultat
 - funkcija je potprogram koji u pozivajući program vraća rezultat

Primjer 1 (nastavak):

- Funkcija koja u zadanom nizu znakova broji koliko ima znakova koji su znamenke (broji znakove iz intervala '0' ... '9'). Pretpostavlja se da duljina zadanog niza znakova ne premašuje 255 bajtova

```
CREATE FUNCTION brojZnamenki (niz CHAR(255))  
    RETURNING SMALLINT AS broj  
    DEFINE brojac, i SMALLINT;  
  
    LET brojac = 0;  
    FOR i = 1 TO CHAR_LENGTH(niz)  
        IF SUBSTRING(niz FROM i FOR 1) BETWEEN '0' AND '9' THEN  
            LET brojac = brojac + 1;  
        END IF;  
    END FOR;  
    RETURN brojac;  
END FUNCTION;  
  
GRANT EXECUTE ON brojZnamenki TO PUBLIC;
```

- funkciju brojZnamenki svaki (sadašnji i budući) korisnik može koristiti na jednak način kao što se koriste standardne SQL funkcije

Primjer 1 (nastavak):

	CHAR (13)	CHAR (30)
osoba	jmbg	prez
	2203979622876	Horvat
	1712 12871211	Kolar
	2707986736233	Še5fer
	03AB621.22876	Novak

- funkcija brojZnamenki se može iskoristiti za ispis onih osoba u čijem jmbg nema točno 13 znamenaka ili u prezimenu postoje znamenke

```
SELECT *, brojZnamenki(jmbg) AS br1, brojZnamenki(prez) AS br2
FROM osoba
WHERE brojZnamenki(jmbg) <> 13 OR brojZnamenki(prez) > 0;
```

jmbg	prez	br1	br2
1712 12871211	Kolar	12	0
2707986736233	Še5fer	13	1
03AB621.22876	Novak	10	0

Primjer 1 (nastavak):

- Pohranjena funkcija se iz interaktivnih alata (npr. Aqua Data Studio) može pozvati na sljedeći način:

```
EXECUTE FUNCTION brojZnamenki('abc123efg456');
```

broj
6

```
CREATE FUNCTION brojZnamenki (niz CHAR(255))  
    RETURNING SMALLINT AS broj  
...
```

Primjer 2:

- Korisnik novak je službenik u banci kojem je potrebno omogućiti obavljanje **isključivo** jedne vrste bankovne transakcije: prebacivanje iznosa s jednog na drugi račun

racun	brRacun	stanje
	1001	1250.15
	1002	-300.00
	1003	10.25

- Zadatak se ne može riješiti dodjelom dozvole za obavljanje operacije UPDATE nad relacijom racun korisniku novak (**zašto?**)

Dozvole za pohranjene procedure/funkcije

- SQL naredbe za dodjeljivanje i ukidanje dozvola za izvršavanje procedura
- `GRANT EXECUTE ON {procName | funName}
TO {PUBLIC | userList | roleList}
[WITH GRANT OPTION]`
- `REVOKE EXECUTE ON {procName | funName}
FROM {PUBLIC | userList | roleList}
[CASCADE | RESTRICT]`

Primjer 2 (nastavak):

```
CREATE PROCEDURE prebaci (saRacunaBr LIKE racun.brRacun
                        , naRacunBr  LIKE racun.brRacun
                        , iznos       LIKE racun.stanje)
-- prenesi zadani iznos
UPDATE racun SET stanje = stanje - iznos
  WHERE brRacun = saRacunaBr;
UPDATE racun SET stanje = stanje + iznos
  WHERE brRacun = naRacunBr;
END PROCEDURE;
GRANT EXECUTE ON prebaci TO novak;
```

Primjer 2 (nastavak):

racun

brRacun	stanje
1001	1250.15
1002	-300.00
1003	10.25

novak

```
UPDATE racun SET stanje = stanje - 60.30  
WHERE brRacun = 1001;
```

[Error] No UPDATE permission

novak

```
EXECUTE PROCEDURE prebaci (1001, 1002, 60.30);
```

racun

brRacun	stanje
1001	1189.85
1002	-239.70
1003	10.25

- Problem: što će se dogoditi ako korisnik pri pozivu procedure kao broj prvog računa zada postojeći, a kao broj drugog računa zada nepostojeći broj računa?

```
EXECUTE PROCEDURE prebaci(1001, 1005, 30.15);
```

Iznimke (*Exceptions*)

- ukoliko SUBP tijekom obavljanja operacije utvrdi da se dogodila pogreška (*error condition*), obavljanje operacije se prekida, a stanje pogreške se signalizira iznimkom (*exception*)

```
SELECT (stanje/(stanje-10.25)) FROM racun;
```

[Error] An attempt was made to divide by zero.

```
SELECT * FROM ispit;
```

[Error] No SELECT permission.

- pogreške koje SUBP nije u stanju prepoznati (jer ih ne smatra pogreškama), mogu se signalizirati naredbom **RAISE EXCEPTION**. Npr, u poboljšanoj proceduri **prebaci** signalizira se pogreška u slučaju kad ne postoji neki od zadanih brojeva računa

```
EXECUTE PROCEDURE prebaci(1001, 1005, 30.15);
```

[Error] Ne postoji drugi račun

Primjer 2 (nastavak):

```
CREATE PROCEDURE prebaci (saRacunaBr LIKE racun.brRacun
                        , naRacunBr  LIKE racun.brRacun
                        , iznos       LIKE racun.stanje)
-- provjeri postoje li zadani brojevi računa
IF (SELECT COUNT(*) FROM racun
    WHERE brRacun = saRacunaBr) = 0 THEN
    RAISE EXCEPTION -746, 0, 'Ne postoji prvi račun';
END IF;
IF (SELECT COUNT(*) FROM racun
    WHERE brRacun = naRacunBr) = 0 THEN
    RAISE EXCEPTION -746, 0, 'Ne postoji drugi račun';
END IF;
-- prenesi zadani iznos
UPDATE racun SET stanje = stanje - iznos
    WHERE brRacun = saRacunaBr;
UPDATE racun SET stanje = stanje + iznos
    WHERE brRacun = naRacunBr;
END PROCEDURE;
GRANT EXECUTE ON prebaci TO novak;
```

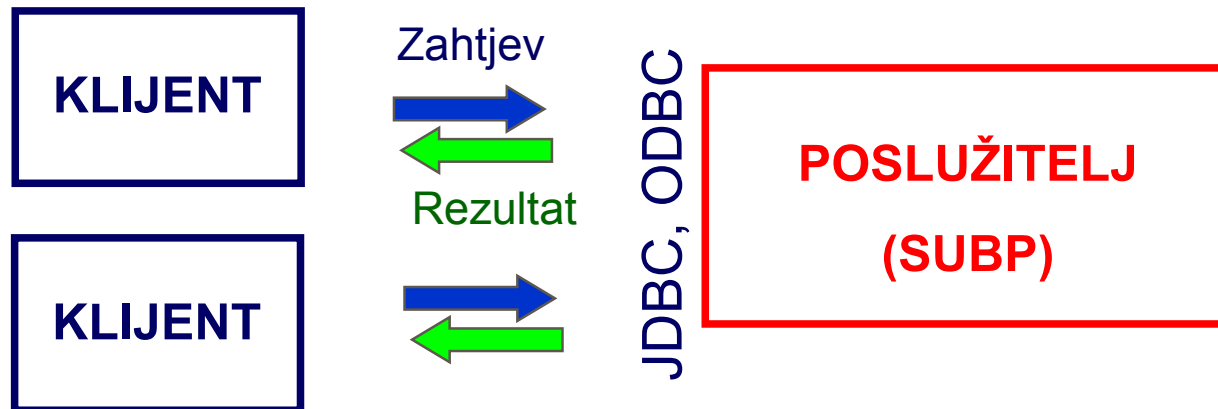
SPL (*Stored Procedure Language*)

- Proizvođači SUBP koriste vlastite inačice jezika za definiranje pohranjenih procedura (standard postoji, ali je rijetko gdje implementiran)
 - IBM Informix: SPL (Stored Procedure Language)
 - Oracle: PL/SQL (Procedural Language/Structured Query Language)
 - Microsoft SQL Server: Transact-SQL
- Navedeni jezici proširuju mogućnosti SQL jezika proceduralnim elementima koji se koriste u strukturiranim jezicima (C, Java, ...). Osim SQL naredbi, pohranjene procedure omogućuju korištenje
 - varijabli
 - naredbi za kontrolu toka programa (*if, for, while, ...*)
 - naredbi za rukovanje iznimkama (*exception handling*)

Prednosti uporabe pohranjenih procedura

- proširenje mogućnosti SQL jezika
- omogućena je zaštita podataka na razini funkcije (a ne samo objekta)
- omogućena je uporaba klijent-poslužitelj arhitekture oslonjene na poslužitelj:
 - postiže se veća učinkovitost SUBP
 - SUBP ne mora ponavljati prevođenje i optimiranje SQL upita
 - postiže se veća produktivnost programera i smanjuje mogućnost pogreške
 - programski kôd potreban za obavljanje nekog postupka koji čini logičku cjelinu implementira se i testira na samo jednom mjestu

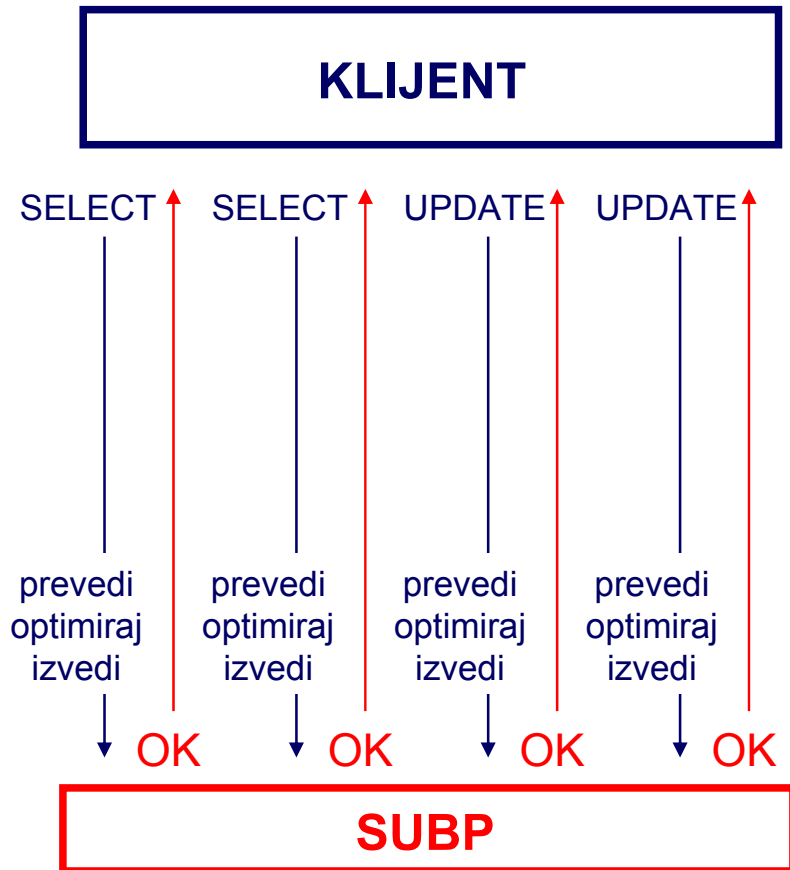
(Klijent-poslužitelj arhitektura)



- sustav obuhvaća dvije komponente
 - klijent i poslužitelj (*client-server*)
- koncept zahtjev-odgovor (*request-response*): klijent postavlja zahtjev, poslužitelj odgovara
- komunikacija između klijenta i poslužitelja se odvija preko dobro definiranih, standardnih programskih sučelja: npr. ODBC (*Open Database Connectivity*), JDBC (*Java Database Connectivity*)

(Klijent-poslužitelj arhitektura - oslonjena na klijenta)

- provjeri postoje li zadani brojevi računa, ako postoje, prebaci iznos s jednog na drugi račun

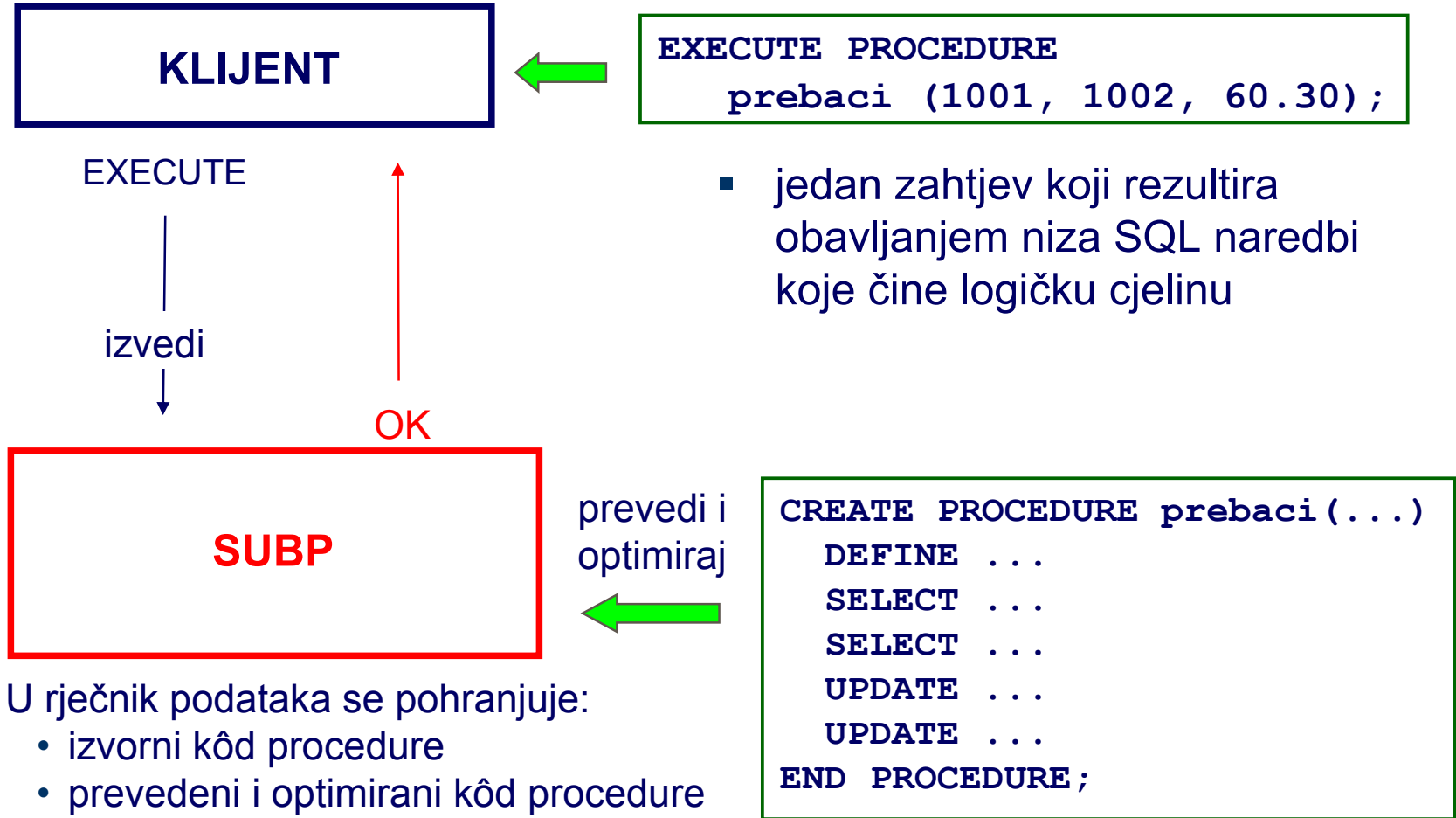


```
SELECT COUNT(*) FROM racun
WHERE brRacun = 1001;
?
SELECT COUNT(*) FROM racun
WHERE brRacun = 1002;
?
UPDATE racun
SET stanje = stanje - 60.30
WHERE brRacun = 1001;
UPDATE racun
SET stanje = stanje + 60.30
WHERE brRacun = 1002;
```

- niz zahtjeva za obavljanje jedne po jedne SQL naredbe

(Klijent-poslužitelj arhitektura - oslonjena na poslužitelj)

- provjeri postoje li zadani brojevi računa, ako postoje, prebaci iznos s jednog na drugi račun



Okidači

Primjer 3:

racun	brRac	sifKlijent	stanje
	1001	98281	216.80
	1002	89734	134.99
	1003	23232	2750.00
	1004	63443	849.50

uplataIsplata	brRac	vrijeme	iznos
	1001	7.8.2007 08:20	15.00
	1002	9.4.2006 12:31	-100.21
	1001	6.5.2007 14:15	452.15
	1004	5.5.2007 16:42	1200.00
	1004	9.9.2005 10:15	-350.50
	1002	7.2.2007 15:01	235.20
	1003	1.4.2005 12:44	2750.00
	1001	1.9.2007 12:19	-250.35

- u relaciju **uplataIsplata** upisuju se promjene na računima
- tijekom godina evidentiran je vrlo veliki broj uplata i isplata
- stanje na određenom računu moglo bi se izračunati zbrajanjem iznosa u relaciji **uplataIsplata**, koji se odnose na dotični račun
- u ovom primjeru, uz svaki račun se redundantno pohranjuje trenutno stanje računa, koje u svakom trenutku mora odgovarati stanju koje bi se dobilo zbrajanjem iznosa u relaciji **uplataIsplata**
- kako osigurati da se pri svakoj relevantnoj promjeni podataka (unos, brisanje, izmjena iznosa) u relaciji **uplataIsplata** izmijeni i odgovarajuće stanje u relaciji **racun**?

Aktivne baze podataka

- konvencionalni SUBP je pasivan
 - operacije nad podacima se izvršavaju isključivo na temelju eksplicitnog zahtjeva korisnika/aplikacije
- aktivni SUBP i aktivne baze podataka
 - aktivni SUBP autonomno reagira na određene događaje (*events*)
 - u aktivnim bazama podataka neke operacije nad podacima se izvršavaju automatski, reakcijom na određeni događaj ili stanje
- željeno ponašanje sustava postiže se definiranjem aktivnih pravila (*active rules*)
- najčešće korištena paradigma za opisivanje aktivnih pravila u današnjim SUBP je događaj-uvjet-akcija (*ECA: Event-Condition-Action*)
 - okidači (*triggers*)

on *event*

if *condition* **then** *action*

- događaj (*event*): ako se dogodi, izračunava se uvjet
 - općenito, događaji mogu biti:
 - unos, izmjena ili brisanje podatka
 - čitanje podatka
 - uspostavljanje SQL-sjednice
 - protok određene količine vremena, dostizanje trenutka u vremenu, ...
- uvjet (*condition*): ako je rezultat izračunavanja uvjeta istina, obavljaju se akcije
 - zadaje se u obliku predikata (slično kao u WHERE dijelu SQL naredbi)
- akcije (*action*): niz operacija, najčešće operacije nad podacima
 - SQL naredbe INSERT, UPDATE, DELETE, poziv procedure, ...

Primjer 3 (nastavak):

- kako osigurati da se pri svakoj relevantnoj promjeni podataka (unos, brisanje, izmjena iznosa) u relaciji **uplataIsplata** izmijeni i odgovarajuće stanje u relaciji **racun**?

racun	brRac	sifKlijent	stanje
	1001	98281	216.80
	1002	89734	134.99

uplataIsplata	brRac	vrijeme	iznos
	1001	7.8.2007 08:20	15.00
	1002	9.4.2006 12:31	-100.21
	1001	6.5.2007 14:15	452.15

- potrebno je utvrditi koji događaji mogu uzrokovati neispravnu vrijednost atributa stanje u relaciji racun, te pod kojim uvjetima treba obaviti koje akcije kako bi se očuvao integritet podataka, npr.
- događaj: obavljanje operacije INSERT nad relacijom uplataIsplata
- uvjet: iznos \neq 0.00
- akcija: pribrojiti vrijednost atributa iznos unesene n-torke u odgovarajuće stanje

Primjer 3 (nastavak):

- događaj: obavljanje operacije INSERT nad relacijom uplataIsplata
- uvjet: iznos \neq 0.00
- akcija: pribrojiti vrijednost atributa iznos unesene n-torke u odgovarajuće stanje

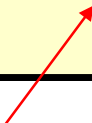
```
CREATE TRIGGER insUplataIsplata
  INSERT ON uplataIsplata
  REFERENCING NEW AS novaUplataIsplata
  FOR EACH ROW
  WHEN (novaUplataIsplata.iznos  $\neq$  0)
    (UPDATE racun SET stanje = stanje + novaUplataIsplata.iznos
     WHERE brRac = novaUplataIsplata.brRac) ;
```

- kad god se obavi naredba INSERT nad relacijom uplataIsplata SUBP obavlja
 - nakon unosa svake n-torke (jednom INSERT naredbom može se unijeti više n-torki) provjerava uvjet `novaUplataIsplata.iznos \neq 0`
 - na sadržaj unesene n-torke može se referencirati koristeći "ime" n-torke koje je zadano pomoću `REFERENCING NEW AS novaUplataIsplata`
 - ako je uvjet zadovoljen (za dotičnu n-torku), obavlja izmjenu stanja u relaciji racun

Primjer 3 (nastavak):

- događaj: brisanje n-torke iz relacije uplataIsplata
- uvjet: iznos \neq 0.00
- akcija: oduzeti vrijednost atributa iznos unesene n-torke od odgovarajućeg stanja

```
CREATE TRIGGER delUplataIsplata
DELETE ON uplataIsplata
REFERENCING OLD AS brisanaUplataIsplata
FOR EACH ROW
WHEN (brisanaUplataIsplata.iznos  $\neq$  0)
  (UPDATE racun SET stanje = stanje - brisanaUplataIsplata.iznos
   WHERE brRac = brisanaUplataIsplata.brRac) ;
```



- ukoliko je potrebno, moguće je navesti više SQL naredbi, međusobno odijeljenih zarezima
- SQL naredbe koje se mogu koristiti za opisivanje akcije:
 - INSERT
 - UPDATE
 - DELETE
 - EXECUTE PROCEDURE

Primjer 3 (nastavak):

- događaj: izmjena vrijednosti atributa iznos u relaciji uplataIsplata
- uvjet: nova vrijednost iznosa \neq stara vrijednost iznosa
- akcija: u odgovarajuće stanje pribrojiti razliku između nove i stare vrijednosti atributa iznos

```
CREATE TRIGGER updIznosUplataIsplata
  UPDATE OF iznos ON uplataIsplata
  REFERENCING OLD AS staraUplataIsplata NEW AS novaUplataIsplata
  FOR EACH ROW
    WHEN (novaUplataIsplata.iznos  $\neq$  staraUplataIsplata.iznos)
      (UPDATE racun SET stanje = stanje +
        novaUplataIsplata.iznos - staraUplataIsplata.iznos
        WHERE brRac = staraUplataIsplata.brRac) ;
```

- **UPDATE OF iznos ON uplataIsplata:** događaj izmjene vrijednosti atributa iznos u relaciji uplataIsplata
- **UPDATE OF a, b, c ON relacija:** događaj izmjene vrijednosti bilo kojeg od atributa a1, a2, a3 u relaciji
- **UPDATE ON relacija:** događaj izmjene vrijednosti bilo kojeg atributa u relaciji

Naredba CREATE TRIGGER

- oblik naredbe za kreiranje okidača propisan je SQL standardom, ali SUBP koriste uglavnom vlastite inačice
- jedna od važnijih mogućnosti koje su na raspolaganju pri definiciji okidača:
 - moguće je specificirati da li se akcije navedene u okidaču obavljaju:
 - po jednom za svaku n-torku na koju je djelovala operacija koja je aktivirala okidač (operacija koja je uzrokovala događaj)
 - FOR EACH ROW
 - samo jednom, nakon što se obavi operacija koja je aktivirala okidač
 - AFTER INSERT, AFTER UPDATE, AFTER DELETE
 - samo jednom, prije nego se obavi operacija koja je aktivirala okidač
 - BEFORE INSERT, BEFORE UPDATE, BEFORE DELETE
- uništavanje okidača: DROP TRIGGER *imeOkidača*

Primjena okidača

- implementacija integritetskih ograničenja
 - okidače treba koristiti onda kada integritetska ograničenja nije moguće opisati na drugi način (PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, ...)
 - obavljanjem korektivne akcije koja bazu podataka dovodi u konzistentno stanje (primjer 3)
 - odbijanjem operacije koja narušava integritetsko ograničenje (primjer 4)
- praćenje rada korisnika (primjer 5)
- sustavi obavještanja (primjer 6)
- itd.

Primjer 4:

- u relaciji ispit osigurati integritetsko ograničenje prema kojem je promjena ocjena dopuštena samo ako se mijenja na nižu ocjenu, npr.
 - dopušteno je ocjenu izvrstan promijeniti u dobar
 - nije dopušteno ocjenu dovoljan promijeniti u vrlo dobar

ispit	matBr	sifPred	datIspr	ocj	sifNast
	100	1001	29.06.2006	3	1111
	100	1001	05.02.2006	1	3333
	101	1002	27.06.2006	2	2222
	102	1001	29.01.2006	1	2222

- očito je da ne postoji korektivna akcija koja bi bazu podataka mogla dovesti u konzistentno stanje nakon što korisnik obavi naredbu:

```
UPDATE ispit SET ocjena = 2
WHERE ocjena = 1;
```

- jedini način na koji se može osigurati navedeno integritetsko ograničenje jest: odbiti izvršavanje takve naredbe

Primjer 4 (nastavak):

```
CREATE PROCEDURE dojavipogreskuUvecanjeOcjene ()
    RAISE EXCEPTION -746, 0, 'Ocjena se ne smije uvećati';
END PROCEDURE;

CREATE TRIGGER updOcjIspit
    UPDATE OF ocj ON ispit
    REFERENCING OLD AS stariIspit NEW AS noviIspit
    FOR EACH ROW
        WHEN (noviIspit.ocj > stariIspit.ocj)
            (EXECUTE PROCEDURE dojavipogreskuUvecanjeOcjene());
```

- što se dešava pri izvršavanju naredbe
- nakon promjene prve n-torke, akcije iz okidača se neće obaviti jer uvjet za obavljanje akcije nije ispunjen
- nakon promjene druge n-torke, aktivirat će se akcija iz okidača
 - poziva se procedura
 - procedura signalizira pogrešku
 - budući da se naredba mora obaviti u cjelosti ili uopće ne, sustav poništava i promjenu prve n-torke, a korisniku prikazuje opis pogreške

```
UPDATE ispit SET ocj = 2
WHERE matBr = 100;
```

Primjer 5:

- pretpostavi li se da je izmjena podataka u relaciji racun naročito osjetljiva operacija - potrebno je pratiti rad korisnika (*audit trail*)

```
CREATE TABLE auditTrailZaRacun (  
    korisnik      CHAR(32)  
    , vrijeme     DATETIME YEAR TO SECOND  
    , brRac1      ...  
    , sifKlijent1 ...  
    , stanje1     ...  
    , brRac2      ...  
    , sifKlijent2 ...  
    , stanje2     ...  
);
```

```
CREATE TRIGGER updRacun  
    UPDATE ON racun  
    REFERENCING OLD AS stari NEW AS novi  
    FOR EACH ROW  
        -- uvjet se može ispustiti  
    (INSERT INTO auditTrailZaRacun VALUES (  
        USER, CURRENT, stari.brRac, stari.sifKlijent, stari.stanje,  
        novi.brRac, novi.sifKlijent, novi.stanje));
```

Primjer 5 (nastavak):

```
...  
FOR EACH ROW  
  (INSERT INTO auditTrailZaRacun VALUES (  
    USER, CURRENT, stari.brRac, stari.sifKlijent, stari.stanje,  
    novi.brRac, novi.sifKlijent, novi.stanje));
```

- što se dešava obavljanjem naredbe

novak

```
UPDATE racun  
  SET stanje = stanje + 10  
  WHERE brRac BETWEEN 1002 AND 1003;
```

racun

brRac	sifKlijent	stanje
1001	98281	216.80
1002	89734	134.99
1003	23232	2750.00
1004	63443	849.50

- osim promjene u relaciji racun, u relaciju auditTrailZaRacun bit će dodane dvije n-torke

auditTrailZaRacun

korisnik	vrijeme	brRac1	sifKlijent1	stanje1	brRac2	sifKlijent2	stanje2
...
novak	2007.02.27 14:13:47	1002	89734	134.99	1002	89734	144.99
novak	2007.02.27 14:13:47	1003	23232	2750.00	1003	23232	2760.00

Primjer 6:

- postoji pohranjena procedura saljiPostu(adresa, tekst)
- u relaciji artikl nalaze se podaci o artiklima na skladištu. Za svaki artikl prati se trenutno stanje (količina) artikla
- kada stanje artikla padne ispod optimalne količine, potrebno je na e-mail adresu djelatnika zaduženog za nabavu tog artikla poslati poruku

artikl	sifArt	stanje	optKol	adresaZaduzenog
	1001	250	150	pero@tvrka.hr
	1002	400	200	joza@tvrka.hr
	1003	450	350	jura@tvrka.hr

```
CREATE TRIGGER updArtikl
UPDATE OF stanje ON artikl
REFERENCING OLD AS stari NEW AS novi
FOR EACH ROW
  WHEN (stari.stanje >= stari.optKol
        AND novi.stanje < stari.optKol)
  (EXECUTE PROCEDURE saljiPostu(stari.adresaZaduzenog
                                , 'Nabavi artikl: ' || stari.sifArt));
```

Primjer 6 (nastavak):

artikl	sifArt	stanje	optKol	adresaZaduzenog
	1001	250	150	pero@tvrka.hr
	1002	400	200	joza@tvrka.hr
	1003	450	350	jura@tvrka.hr

- rezultat obavljanja naredbe

```
UPDATE artikl  
SET stanje = stanje - 150;
```

artikl	sifArt	stanje	optKol	adresaZaduzenog
	1001	100	150	pero@tvrka.hr
	1002	250	200	joza@tvrka.hr
	1003	300	350	jura@tvrka.hr

- + dvije poruke

pero@tvrka.hr: Nabavi artikl: 1001
jura@tvrka.hr: Nabavi artikl: 1003

- ako se nakon toga obavi naredba

```
UPDATE artikl  
SET stanje = stanje - 100;
```

artikl	sifArt	stanje	optKol	adresaZaduzenog
	1001	0	150	pero@tvrka.hr
	1002	150	200	joza@tvrka.hr
	1003	200	350	jura@tvrka.hr

- + poruka

joza@tvrka.hr: Nabavi artikl: 1002

KRATKI PRIRUČNIK ZA SPL

Kreiranje pohranjene procedure

- Procedura se kreira SQL naredbom oblika:

```
CREATE PROCEDURE imeProcedure (eventualni argumenti)
    tijelo procedure
END PROCEDURE;
```

```
CREATE FUNCTION imeFunkcije (eventualni argumenti)
    tijelo funkcije
END FUNCTION;
```

- Eventualne pogreške u sintaksi naredbe sustav će dojaviti za vrijeme obavljanja naredbe (na isti način kao i pogreške za vrijeme obavljanja ostalih SQL naredbi).
- Brisanje (uništavanje) procedure

```
DROP PROCEDURE imeProcedure;
DROP FUNCTION imeFunkcije;
```

- Izmjena procedure: brisanjem starog objekta i definiranjem novog objekta pod istim imenom, npr.

```
DROP PROCEDURE imeProcedure;
CREATE PROCEDURE imeProcedure ...;
```

Struktura pohranjene procedure

```
CREATE PROCEDURE imeProcedure (eventualni argumenti)
    definicija varijabli
    naredba;
    naredba;
    ...
END PROCEDURE;
```

```
CREATE FUNCTION imeProcedure (eventualni argumenti)
    definicija varijabli
    naredba;
    naredba;
    ...
END FUNCTION;
```

- Naredbe procedure završavaju znakom ; (točka-zarez)

Definicija varijabli

- Definicije varijabli se navode na početku procedure. Sadržaj varijable je nedefiniran dok mu se ne pridruži neka vrijednost.
- Tipovi varijabli mogu biti definirani eksplicitno:

```
CREATE PROCEDURE imeProcedure (eventualni argumenti)
  DEFINE ime CHAR(20);
  DEFINE ocjena, brojIzlazaka SMALLINT;
  ...
```

- ili implicitno, prema tipovima atributa u relacijama baze podataka

```
CREATE PROCEDURE imeProcedure (eventualni argumenti)
  DEFINE ime LIKE student.imeStud;
  DEFINE ocjena LIKE ispit.ocjena;
  ...
```

- kad god je moguće, tipove varijabli treba definirati implicitno
 - u slučaju promjene tipa podatka nekog atributa u relaciji, sve što je potrebno obaviti jest ponovo prevesti procedure

Naredba LET

- koristi se za pridruživanje vrijednosti varijablama

```
CREATE PROCEDURE ...  
    DEFINE r, površina DECIMAL(10,5);  
    DEFINE brojIspita SMALLINT;  
    DEFINE sumaOcjena INTEGER;  
    DEFINE prosjek DECIMAL(3,2);  
    DEFINE brojZnam SMALLINT;  
  
    LET r = 10;  
    LET površina = 3.14159 * r * r;  
  
    LET brojIspita = (SELECT COUNT(*) FROM ispit);  
    LET sumaOcjena = (SELECT SUM(ocjena) FROM ispit);  
    LET prosjek = sumaOcjena/brojIspita;  
    LET brojZnam = brojZnamenki('123abc');  
    ...
```

- rezultat obavljanja SELECT naredbe koja vraća jednu jednostavnu vrijednost (skalar) može se koristiti na svim mjestima na kojima se koriste izrazi. SELECT naredba mora biti unutar okruglih zagrada

Naredbe IF, WHILE, FOR

- naredba za jednostranu, dvostranu ili višestranu selekciju
- naredba za realizaciju petlje s ispitivanjem uvjeta na početku
- naredba za realizaciju petlje s unaprijed utvrđenim brojem ponavljanja

```
IF uvjet THEN
    naredbe
ELIF uvjet THEN
    naredbe
ELIF uvjet THEN
    naredbe ...
ELSE
    naredbe
END IF;
```

```
WHILE uvjet
    naredbe
    ...
    EXIT WHILE;
    CONTINUE WHILE;
END WHILE;
```

kao break u jeziku C
kao continue u jeziku C

```
FOR i = m TO n STEP k
    naredbe
    ...
    EXIT FOR;
    CONTINUE FOR;
END FOR;
```

kao break u jeziku C
kao continue u jeziku C

SQL naredbe u pohranjenim procedurama

- U pohranjenim procedurama mogu se koristiti (gotovo) sve do sada prikazane SQL naredbe (izuzetak je npr. DROP DATABASE)

```
...  
DELETE FROM stud  
  WHERE prezStud LIKE 'Z%';  
UPDATE stud SET pbrMjestoStan = 10000  
  WHERE pbrMjestoStan = 41000;  
INSERT INTO mjesto VALUES (31000, 'Osijek');  
...
```

- Rezultat SELECT naredbe može se pohraniti u varijable, npr.

```
DEFINE v_imeStud LIKE student.imeStud;  
DEFINE v_prezStud LIKE student.prezStud;  
...  
SELECT imeStud, prezStud  
  INTO v_imeStud, v_prezStud  
  FROM student  
  WHERE mbrStud = 12345;
```

- broj i tipovi varijabli moraju odgovarati broju i tipovima izraza iz liste za selekciju
- SELECT naredba smije vratiti samo jednu n-torku

Uporaba varijabli u SQL naredbama

- varijable se slobodno mogu koristiti na svim mjestima na kojim se u SQL naredbama koriste izrazi, npr.
 - u izrazima u SELECT listi, u WHERE dijelu SQL naredbe
 - u VALUES listi INSERT naredbe
 - u izrazima u SET dijelu UPDATE naredbe, ...

```
CREATE PROCEDURE ...
  DEFINE iznos, koef DECIMAL (3,2);
  DEFINE datum DATE;
  DEFINE s INTEGER;
  DEFINE n CHAR(20);
  LET koef = (SELECT MAX(koef) FROM nastavnik);
  LET s = 100; LET n = 'Primorsko-goranska';
  LET datum = TODAY - 365*20;
  SELECT AVG(ocjena) * koef INTO iznos FROM ispit
    WHERE datIspit = datum;
  UPDATE stud SET datRodStud = datum
    WHERE datRodStud <> datum;
  INSERT INTO zupanja VALUES(s, n);
  ...
```

Argumenti pohranjene procedure

```
CREATE PROCEDURE imeProcedure (imeArg tip, imeArg tip, ...)
```

- tipovi podataka ulaznih argumenata procedure mogu, kao i varijable, biti definirani eksplicitno ...

```
CREATE FUNCTION površina (sirina INTEGER, visina INTEGER)  
...
```

- ili implicitno ...

```
CREATE PROCEDURE postaviAdresu (p_mbrStud LIKE stud.mbrStud  
                                , p_adresa LIKE stud.adresa)  
...
```

- jednako kao u drugim programskim jezicima, argumenti se u tijelu procedure/funkcije mogu koristiti na jednak način kao i varijable

Rezultati funkcije

- tipovi rezultata koje funkcija vraća moraju se deklarirati

```
CREATE FUNCTION imeFunkcije (imeArg tip, imeArg tip, ...)
  RETURNING INTEGER AS ime, CHAR(20) AS ime, DATE AS ime
  DEFINE ...
  ...
END FUNCTION;
```

- tipovi rezultata mogu se deklarirati jedino eksplicitno (nije moguće koristiti oblik "LIKE atribut" kao pri definiciji argumenata ili varijabli)
- "ime" rezultata se navodi opcionalno: korisno je deklarirati ime rezultata jer se npr. pri pozivu funkcije iz interaktivnog alata rezultat prikazuje zajedno s deklariranim imenom

Povrat rezultata funkcije u pozivajući program

- koristi se naredba RETURN slična naredbi RETURN u ostalim programskim jezicima. RETURN naredba se u tijelu procedure može pojaviti više puta. Naredbom je u pozivajući program moguće vratiti jednu ili više vrijednosti

```
CREATE FUNCTION opsegPovrsina(radijus DECIMAL(10,5))  
    RETURNING DECIMAL(10,5) AS opseg  
    , DECIMAL(10,5) AS povrsina  
    DEFINE o, p DECIMAL(10,5);  
    LET o = 2 * radijus * 3.14159;  
    LET p = radijus * radijus * 3.14159;  
    RETURN o, p;  
END FUNCTION;
```

```
EXECUTE FUNCTION opsegPovrsina(4.5);
```

opseg	povrsina
28.27431	63.61720

Načini poziva procedure (funkcije)

- Iz interaktivnih alata, npr. Aqua Data Studio

```
EXECUTE PROCEDURE prebaci (1001, 1002, 60.30);
```

- procedura ne vraća rezultat (eventualno signalizira pogrešku)

```
EXECUTE FUNCTION opsegPovrsina(4.5);
```

- funkcija vraća rezultat (eventualno signalizira pogrešku)

opseg	povrsina
28.27431	63.61720

Načini poziva procedure (funkcije)

- Iz pohranjene procedure ili funkcije

```
CREATE PROCEDURE x (...)  
  DEFINE brojZnam ...  
  DEFINE opseg, povrsina ...  
  ...  
  -- procedure ne vraćaju rezultat  
  EXECUTE PROCEDURE prebaci (1001, 1002, 60.30);  
  ...  
  -- funkcije koje vraćaju jednu vrijednost  
  LET brojZnam = brojZnamenki('abc123');  
  CALL brojZnamenki('abc123') RETURNING brojZnam;  
  ...  
  -- funkcije koje vraćaju više vrijednosti  
  CALL opsegPovrsina(4.5) RETURNING opseg, povrsina;
```

Načini poziva funkcije

- Korištenje funkcija u SQL naredbama
 - funkcije koje vraćaju točno jednu vrijednost mogu se u SQL naredbama koristiti na svim mjestima na kojima se mogu koristiti ugrađene SQL funkcije

```
SELECT *, brojZnamenki(adresa) AS brojZnam  
FROM osoba  
WHERE brojZnamenki(adresa) > 0;  
  
DELETE FROM osoba  
WHERE brojZnamenki(jmbg) <> 13;
```