BAZE PODATAKA – DEVETI TERMIN VJEŽBI (PL/SQL)

1. Napraviti blok naredbi koji će omogućiti da se na ekranu ispisuju imena svih zaposlenih iz tabele HR.EMPLOYEES koji rade od datuma koji se zadaje u varijabli DATUM, i čiji rukovodilac radi u odjeljenju sa šifrom datom u varijabli SO. Za provjeru rezultata koristiti datum zaposlenja `11-05-1999` i vrijednost 80 za varijablu SO.
2. Korisnik MILOSB napravio je tabelu BPISPIT za koju imate pravo pregleda. Tabela sadrži spisak studenata i bodove koje su oni ostvarili tokom semestra iz predmeta Baze podataka: na prvom kolokvijumu, drugom kolokvijumu, na aktivnostima na času i na završnom ispitu. Napraviti identičnu kopiju ove tabele pod Vašim korisničkim nalogom. U tabeli su unešeni rezultati na svim oblicima ispitivanja.
   a) Napraviti proceduru UNESI sa tri ulazna parametra: IND, POEN i ISPITIVANJE, čijim se pozivanjem omogućava unos bodova za studente iz tabele BPISPIT. Tabela se popunjava tako što procedura za studenta sa datim brojem indeksa IND unosi broj poena POEN na ispitivanju ISPITIVANJE koje može imati vrijednost: K1, K2, A ili I. Takođe, u kolonu UKUPNO koja postoji u tabeli BPISPIT ova procedura za studenta kojem su u datom trenutku unijeti bodovi računa i unosi ukupan broj poena (zbir poena na svim ispitivanjima). Ukoliko za datog studenta, i na datom ispitivanju poeni već postoje, procedura zamjenjuje ranije unijeti rezultat novim brojem poena (na ostalim ispitivanjima ostaju stari poeni).
   b) Napraviti funkciju ISP koja za ulazni argument ima broj indeksa studenta, IND i koja kao izlazni argument vraća string 'IZASAO' ako student ima poene na ispitu (u koloni I), i string 'NIJE IZASAO' ako ovi poeni ne postoje. Pretpostaviti da i 0 poena na ispitu označava da student nije izašao na ispit. Neka funkcija pri pozivu automatski računa ukupan broj poena za datog studenta.
   c) Napraviti proceduru OCIJENI bez ulaznih parametara koja će u koloni OCJENA unijeti odgovarajuću ocjenu (>=50 je ocjena E, >=60 je ocjena D itd.). U slučaju da student nije izašao na ispit, procedura upisuje N kao rezultat ispita. Provjeru da li je student izašao na ispit ili ne procedura vrši pozivanjem funkcije ISP.
3. Napraviti tabelu PREDUZECE koja je po svom sadržaju identična tabeli HR.EMPLOYEES. Napraviti proceduru PROC_PLATE koja omogućava promjenu iznosa plate zaposlenima iz tabele PREDUZECE čiji je rukovodilac zadat sa šifrom SS za iznos od PR procenata. SS i PR su parametri (argumenti) procedure. Procedura treba da ima i treći argument, string STR. Ako korisnik zada vrijednost stringa 'povecaj' plata treba da se poveća, a ako zada vrijednost stringa 'smanji' plata treba da se smanji. Podrazumijevana vrijednost stringa je 'povecaj'.
4. Napisati proceduru PROC_PLATE2 koja obavlja iste funkcije kao i procedura iz prethodnog zadatka, sa tom razlikom da se umjesto šifre rukovodioca zadaje ime i prezime rukovodioca i odjeljenje u kojem zaposleni radi, uz dodatni uslov da rukovodilac mora biti u istom odjeljenju kao i zaposleni. Ime i prezime rukovodioca se zadaje kao jedan string. Kada se ne zada pravilna vrijednost stringa STR, neophodno je obezbijediti da izuzetak bude obrađen, poništavanjem modifikacija i ispisivanjem odgovarajućeg obavještenja na ekranu. Kao primjer možete koristiti rukovodioca koji se zove 'Alexander' i koji radi u odjeljenju sa šifrom 60.
5. Napraviti tabelu CASOVI koja ima četiri kolone: NID, IME, BRP (broj predmeta), BRC (broj časova). Napisati proceduru CASOVI_P koja ima jedan parametar, cio broj P, koja omogućava da, koristeći tabele dostupne u vidu javnih sinonima NASTAVNICI, PREDMETI i OPT popunjavate podacima tabelu CASOVI. Ukoliko se ova procedura pozove sa vrijednošću parametra 1, ona će obrisati prethodni sadržaj tabele CASOVI, i unijeti spisak nastavnika, broj

predmeta na kojima su angažovani, i ukupan broj časova koje drže na Univerzitetu. Ukoliko se pozove sa vrijednošću parametra 0, procedura će umjesto ukupnog broja časova koje nastavnici drže upisati maksimalan broj časova koje drže na nekom predmetu.

6. Kreirati proceduru CASOVI_M koja nema ulaznih argumenata, a koja služi za ažuriranje tabele CASOVI. Ova procedura svim nastavnicima koji imaju 90% opterećenja od maksimalnog opterećenja u tabeli CASOVI povećava broj predmeta za jedan.

PREDLOG RJEŠENJA, UVOD U PL/SQL

1. Da bi riješili ovaj zadatak, uvešćemo neke osnovne pojmove vezane za PL/SQL. PL/SQL je proceduralno proširenje Oracle-SQL-a uvedeno sa ciljem povećanja mogućnosti koje nudi standardni SQL, i koji korisnicima nudi mogućnost pravljenja kompleksnih aplikacija za baze podataka.

Glavna konstrukcija u PL/SQL-u se naziva **blok**. U blokovima se mogu koristiti konstante i promjenljive, dok se u promjenljivima mogu čuvati rezultati upita. Iskazi u PL/SQL blokovima mogu sadržati SQL iskaze, kontrolne strukture (petlje), uslovne iskaze (if-then-else), obradu izuzetaka, kao i pozive na druge SQL iskaze. Blokovima se mogu specificirati procedure i funkcije. Ovakvi blokovi mogu biti grupisani u **pakete**, koji u okviru ovog kursa neće biti detaljnje obrađivani (pogledati u literaturi).

Svaki blok čini imenovanu programsku cjelinu, i bitno je istaći činjenicu da blokovi mogu biti ugniježđeni. Osnovna struktura PL/SQL bloka izgleda:

[<Zaglavlje bloka>]
[**declare**
<Konstante>
<Promjenljive>
<Kurzori>
<Korisnički definisani izuzeci>]
begin
<PL/SQL iskazi>
[**exception**
<kod za hvatanje izuzetaka>]
end;

Zagrade '[]' označavaju sadržaj koji je opcion. Zaglavlje bloka (Block header) specificira da li je blok funkcija, procedura ili paket. U nabrojanim slučajevima blok mora biti imenovan. Neimenovani blokovi nazivaju se *anonymous* blokovi.

Deklaracija varijabli i konstanti, i dodjela vrijednosti

Promjenljive, konstante, kurzori i izuzeci koji će biti korišćeni u datom bloku moraju biti deklarisani u odgovarajućem dijelu bloka namijenjenom za deklaraciju. Objašnjenja vezana za kurzore i izuzetke biće data u kasnijem izlaganju. Deklaracija promjenljivih i konstanti obavlja se na sljedeći način:

<ime promjenljive> [**constant**] <tip podatka> [**not null**] [:= <izraz>];

Napomenimo još jednom da srednje zagrade označavaju opcioni sadržaj. Validni tipovi podataka su svi tipovi koji se koriste u SQL-u, i dodatno, tip podatka **boolean**. Tip podatka

boolean može imati isključivo vrijednosti *true*, *false* i *null*. **Not null** klauzula obezbjeđuje da varijabli ne može biti dodijeljena *null* vrijednost. Dio [:= <izraz>] služi za inicijalizaciju promjenljive (ili konstante). Ukoliko dio [:=<izraz>] nije specificiran, varijabli se podrazumijevano dodjeljuje *null* vrijednost. Klauzula **constant** obezbjeđuje da jedna vrijednost dodijeljena promjenljivoj ne može biti kasnije promijenjena (deklariše se konstanta).

U PL/SQL-u ne mogu se koristiti komande SQL-a vezane za definiciju struktura podataka, kakva je, na primjer, **create table**.

Pridruživanje podataka varijablama može se obaviti na više načina. Jedan od načina je korišćenjem operatora ':=', odnosno, na primjer:

```
declare
varijabla integer := 0;
. . .
begin
varijabla := varijabla + 1;
. . .
```

Drugi način pridruživanja podataka je korišćenjem **select** upita, na način koji slijedi:

select <kolona(e)> **into** <odgovarajuća lista varijabli>
from <tabela(e)> **where** <uslov>;

Od prevashodne važnosti je da prethodni **select** upit vraća najviše jednu vrstu, u protivnom neće biti moguće pridruživanje vrijednosti atributa listi varijabli. Potrebno je napomenuti i to da tipovi podataka varijabli moraju biti u skladu sa tipovima podataka odgovarajućih atributa koji im se pridružuju. Za većinu tipova podataka PL/SQL obavlja automatsku konverziju podataka.

Osim eksplicitnog specificiranja tipa podataka, datoj promjenljivoj ili konstanti moguće je dodijeliti tip podatka kolone iz neke postojeće tabele (tzv. usidrena deklaracija). Na primjer, TABELA.KOLONA%TYPE daje varijabli ili konstanti tip podatka kolone KOLONA iz tabele TABELA. Tada se pridruživanje vrijednosti vrši na sljedeći način (i u ovom slučaju rezultat **select** upita mora biti jedna vrsta):

```
declare
varijabla TABELA.KOLONA1%TYPE;
begin
select vrijednost into varijabla from TABELA where uslov;
. . .
end;
```

Za rješavanje našeg zadatka biće potrebno i uvesti pojam kurzora i petlji.

Kurzori

U PL/SQL-u postoji mogućnost procesiranja rezultata upita na takav način da se u jednom trenutku procesira jedna vrsta. Ovakav mehanizam obezbjeđuju **kurzori**. Kurzor je pokazivač na rezultat upita, koji se koristi za dodjeljivanje vrijednosti atributa (vrijednosti iz kolona rezultata) varijablama. Kurzori se najčešće upotrebljavaju u kombinaciji sa petljama, tako da se

svaka vrsta koja je dodijeljena kurzoru može pojedinačno procesirati. Kurzor se deklariše na sljedeći način:

cursor <naziv kurzora> [(<lista parametara>)] **is** <**select** upit>

Na ovakav način se specificira set vrsta, odnosno rezultat upita, tako da vrste mogu biti procesirane jedna po jedna (*tuple-oriented way*). Naziv kurzora ne smije biti naziv nijedne varijable. Parametar ima formu: <naziv parametra> <tip parametra>. Mogući tipovi parametara su **char**, **varchar2**, **date** i **boolean**, kao i njihovi podtipovi kao što je **integer**. Parametri se koriste za dodjeljivanje vrijednosti koje su date u **select** upitu.

U slučaju da će selektovane vrste biti modifikovane u datom bloku, mora se koristiti klauzula **for update**[(<kolona(e)>)] na kraju deklaracije kurzora. U tom slučaju selektovane vrste biće nedostupne korisniku sve dok se ne odradi transakcija pomoću **commit**. Prije korišćenja, *deklarisani kurzor mora da se otvori,* a nakon procesiranja selektovanih redova *kurzor mora biti zatvoren.*

U vezi sa kurzorom je **select** upit na koji se on odnosi. Prije upotrebe, deklarisani kurzori moraju biti otvoreni na sljedeći način:

open <naziv kurzora> [(<lista parametara>)] ;

Tada se izvršava **select** upit koji je u vezi sa deklarisanim kurzorom, i kurzor referencira prvu vrstu koja je rezultat tog **select** upita. Da bi se pristupilo toj vrsti iz rezultata tog **select** upita, neophodno je koristiti **fetch** komandu:

fetch <naziv kurzora> **into** <lista varijabli>;

gdje će atributi selektovane vrste biti pridruženi odgovarajućoj listi varijabli. Kada se izvrši **fetch** komanda, kurzor referencira na sljedeću vrstu iz rezultata svog **select** upita, itd. Napomenimo da tipovi podataka varijabli moraju korespondirati tipovima podataka atributa koji im se pridružuju.

Nakon iscrpljivanja svih vrsta (to jest kada petlja u kojoj se nalazi kurzor dođe do kraja tabele, to jest rezultata upita), koristi se **close** komanda za zatvaranje kurzora, na sljedeći način:

close <naziv kurzora>

I u slučaju kurzora, ukoliko se postavi tip podatka u obliku TABELA%ROWTYPE, tada se deklariše tip podatka koji se odnosi na sve vrijednosti atributa (kolona) čitave vrste tabele TABELA. Ovakvi *zapisi* često se koriste u kombinaciji sa kurzorima. Pristupanje pojedinim podacima takvog zapisa obavlja se u obliku: TABELA.KOLONA. Primjer ovakve deklaracije sličan je kao u slučaju varijabli:

```
declare
zapis TABELA%ROWTYPE;
begin
select kolona1, kolona2,...,kolonan
into zapis
from TABELA where neki_uslov;
. . .
```

end;

Dakle, deklarisali smo zapis koji je istog tipa kao vrsta tabele TABELA, i u ovu varijablu smjestili smo rezultat upita **select** kolona1, kolona2,...,kolona*n*.

Za rješavanje našeg zadatka neophodno je i uvesti osnovnu sintaksu petlji. **While** petlja definiše se na sljedeći način:

[<< <naziv labele> >>]
while <uslov> **loop**
<sekvenca naredbi>;
end loop [<naziv labele>] ;

While petlja može biti imenovana. Imenovanje petlji korisno je u slučaju ugniježdenih petlji.

Kao i u klasičnom programiranju, pored **while** petlje koja će se izvršavati dok god je određeni uslov ostvaren, i ovdje se može koristiti **for** petlja, kada je poznat broj iteracija (ili je makar implicitno poznat, kao u slučaju kurzora):

[<< <naziv labele > >>]
for <index> **in** [**reverse**] <donja_granica>..<gornja_granica> **loop**
<sekvenca naredbi>
end loop [<naziv labele>] ;

Brojač *index* je implicitno deklarisan, i on važi samo unutar **for** petlje na koju se odnosi. Njemu ne može biti dodijeljena vrijednost, ali se zato može koristiti u izrazima. Ukoliko se koristi ključna riječ **reverse**, iteracije će ići od gornje do donje granice.

Kurzori u kombinaciji sa **for** petljom mogu se koristiti na sljedeći način:

[<<<naziv labele>>>]
for <naziv zapisa > **in** <naziv kurzora>[(<lista parametara>)] **loop**
<sekvenca naredbi>
end loop [<naziv labele>];

Zapis za čuvanje vrste je implicitno deklarisan. Takođe, ovakva petlja implicitno odrađuje **fetch** u svakoj iteraciji, i dodatno, otvara i zatvara kurzor prije,odnosno poslije kraja petlje. Ako nijedna vrsta nije ostala za pridruživanje, petlja se automatski prekida, bez korišćenja **exit**-a.

Petlju **for** je moguće deklarisati i na sljedeći način:

for <naziv zapisa> **in** (<**select** uslov>) **loop**
<sekvenca naredbi>
end loop;

U svakoj iteraciji selektovana je po jedna vrsta iz rezultata **select** upita.

If-then-else, koji služi za uslovnu kontrolu, definiše se kao:

if <uslov> **then** <sekvenca naredbi>
[**elsif**] <uslov> **then** <sekvenca naredbi>

. . .
[**else**] <sekvenca naredbi> **end if**;

Logika je ista kao i u standardnim programskim jezicima.

Osim komandi za definisanje podataka (kao **create table**) sve SQL komande su dozvoljene u PL/SQL-u. Naročito je bitno istaći mogućnost korišćenja naredbi: **delete, update, insert** i **commit.** Bitno je istaći da je **select** naredbu moguće koristiti isključivo za pridruživanje vrijednosti varijablama zajedno sa **into** (osim u slučaju podupita, i za dodjeljivanje kurzorima). Komande **delete** i **update** mogu biti korišćene na takav način da se njihova primjena ograničava na vrstu koja je odabrana naredbom **fetch**. U tom slučaju se dodaje klauzula:

where current of <naziv kurzora>

Svaka petlja može biti bezuslovno (nasilno) prekinuta korišćenjem sljedeće klauzule:

exit [<labela bloka>] [**when** <uslov>]

Korišćenjem ove naredbe bez labele bloka, podrazumijevano se prekida ona petlja koja sadrži naredbu **exit**. Uslov može biti na primjer obično poređenje vrijednosti. Međutim, u većini slučajeva uslov se odnosi na kurzor, koji se nalazi u petlji. Za ovu namjenu može se koristiti KURZOR%NOTFOUND, gdje će biti vraćena vrijednost *true* ukoliko je posljednji **fetch** nije uspio da vrati odgovarajuću vrstu, i *false* ukoliko je uspio. Dakle, može se postaviti u **exit** naredbi da je potrebno bezuslovno prekinuti petlju onda kada više nema vrsta za obradu. Ekvivalentno se može koristiti predikat %FOUND.

Imajući u vidu prethodne napomene, zadatak ćemo riješiti korišćenjem anonimnog bloka naredbi. Predlog rješenja zadatka iz postavke je:

```
declare
      datum varchar2(20):='11-05-1999'/*ovdje unijeti datum*/;
      so number:=80/*ovdje unijeti sifru odjeljenja*/;
      cursor radc is select e.first_name|| ' ' || e.last_name radnik
      from hr.employees e where hire_date>to_date(datum,'dd-mm-yyyy')
      and exists(select * from hr.employees where
      e.manager_id=employee_id and department_id=so);
begin
      for zapis in radc loop
            dbms_output.put_line(zapis.radnik);
      end loop;
end;
```

Varijabla DATUM je tipa **varchar2**. Obratiti pažnju da je prilikom deklaracije tekstualnih varijabli u anonimnom bloku naredbi *uvijek* neophodno zadati dužinu stringa. Druga varijabla SO je numeričkog tipa, pa je, *kao i u slučaju standardnog SQL-a moguće ne zadavati dužinu varijable numeričkog tipa*. U sljedećem redu deklarisan je kurzor RADC, koji je pokazivač na rezultat SQL upita desno od ključne riječi IS. Obratiti pažnju da je u tom SQL upitu moguće koristiti deklarisane varijable. Ovo je jedan od glavnih razloga za neophodnost deklaracije varijabli. Korišćenje ključne riječi EXISTS je u ovom slučaju smisleno, jer u postavci zadatka

nije eksplicitno navedeno ko je rukovodilac na koji se odnosi spisak ili kojem tačno odjeljenju pripada, već je jedino traženo da se vrate one vrste tabele HR.EMPLOYEES kod kojeg zaposleni rade od zadatog datuma, i za koje *postoji* rukovodilac (njihov) koji radi u datom odjeljenju. Ekvivalentan upit, bez korišćenja ključne riječi EXISTS bi bio sljedeći:

```
select e.first_name|| ' ' || e.last_name radnik from hr.employees e
where hire_date>to_date('11.5.1999','dd-mm-yyyy') and
manager_id=(select e.manager_id from hr.employees where
e.manager_id=employee_id and department_id=80)
```

Ovdje je dat direktno SELECT upit koji rješava problematiku na ekvivalentan način. Za vježbu ga modifikujte tako da može biti u sklopu deklaracije kurzora.

Kretanje kroz **for** petlju funkcioniše prema objašnjenju koje slijedi. Na početku petlje kurzor RADC pokazuje na početak rezultata SELECT upita na koji se odnosi. Zatim se u prvoj iteraciji zapisu ZAPIS pridružuje implicitno (**fetch** se odrađuje sam) prvi red tog rezultata. U sljedećoj iteraciji kurzor pokazuje na sljedeći red, pa se, dakle, on pridružuje zapisu. Petlja se prekida kad se iscrpi svaki red rezultata na koji se kurzor odnosi. U okviru petlje koristi se funkcija `dbms_output.put_line()`, koja će rezultat, odnosno u našem slučaju kolonu radnik iz zapisa ZAPIS, kojoj se pristupa korišćenjem operatora '.' (ZAPIS.RADNIK) ispisati na ekranu. Napomenimo da se komentari zapisuju korišćenjem '/*' za otvaranje i '*/' za zatvaranje komentara, što je korišćeno za pojašnjenje varijabli u kodu.

2. Za rješavanje ovog zadatka neophodno je uvesti osnovnu sintaksu procedura. Zbog sličnosti, na ovom mjestu predstavićemo i osnovnu sintaksu funkcija. Procedura je imenovani blok koji ima ulazne argumente i čijim se pozivom sa odgovarajućim argumentima izvršava kod definisan u tijelu procedure. Procedure nemaju izlaznih parametara. Sintaksa procedure je sljedećeg oblika:

create [**or replace**] **procedure** <naziv procedure> [(<lista parametara>)] **is**
<deklaracije>
begin
<sekvenca naredbi>
[**exception**
<obrada izuzetaka>]
end [<naziv procedure>];

Funkcije imaju sintaksu analognog oblika, samo je razlika u tome što one vraćaju vrijednost, odnosno imaju izlazni argument, a procedure ne:

create [**or replace**] **function** <naziv funkcije> [(<lista parametara>)]
return <tip podatka> **is**
<deklaracije>
begin
<sekvenca naredbi>
[**exception**
<obrada izuzetaka>]
end [<naziv procedure>];

Brisanje procedura, odnosno funkcija obavlja se naredbom:

drop procedure <naziv procedure>, odnosno,

drop function <naziv funkcije>

Ključna riječ **declare** ne koristi se u slučaju blokova i procedura, za razliku od upotrebe kod anonimnih blokova. Parametre čine svi validni tipovi podataka, s tom razlikom što se ne stavlja njihova dužina. Dakle, u deklaraciji parametara ne stavlja se npr. **varchar(5)**, nego samo **varchar**. Mogu se koristiti i implicitni tipovi oblika %ROWTYPE i %TYPE.

Parametri se zadaju u opštem obliku:

<naziv parametra> [**IN** | **OUT** | **IN OUT**] <tip podatka> [{ := | DEFAULT} <izraz>]

Značenje opcionih ključnih riječi **IN, OUT** i **IN OUT** istražiti koristeći literaturu. Procedure i funkcije pozivaćemo u okviru anonimnog bloka naredbi.

a) Procedura UNESI može biti napravljena na sljedeći način:

```
create or replace procedure unesi(ind varchar2, poen number,
ispitivanje varchar2) is
      cursor studentc is select * from bpispit where indeks=ind for
      update of k1,k2,a,i,ukupno;
      pom bpispit%rowtype;
begin
      open studentc;
      fetch studentc into pom;
      if upper(ispitivanje)='K1'then
            update bpispit set k1=poen where current of studentc;
      elsif upper(ispitivanje)='K2' then
            update bpispit set k2=poen where current of studentc;
      elsif upper(ispitivanje)='I' then
            update bpispit set i=poen where current of studentc;
      else
            update bpispit set a=poen where current of studentc;
      end if;
update bpispit set ukupno=k1+k2+a+i where current of studentc;
      close studentc;
end;
```

Obratiti pažnju da su parametri procedure zadati tako da u slučaju tipa **varchar2** ne stoji dužina stringa. Ovo je pravilo u slučaju definisanja parametara bilo funkcija bilo procedura. Kurzor STUDENTC na kraju specificiranja upita za koji se vezuje ima ključne riječi FOR UPDATE OF gdje se daje spisak kolona koje će u tabeli za koju se kurzor poziva biti izmijenjene. Ukoliko će se u pozivu procedure ili funkcije vršiti promjene podataka, a pritom se koriste kurzori, *neophodno* je koristiti ove ključne riječi. Kurzor je nakon ključne riječi BEGIN otvoren naredbom OPEN. Zapisu POM, koji sadrži tipove podataka svih kolona tabele BPISPIT, što je deklarisano korišćenjem %ROWTYPE, naredbom FETCH se pridružuje vrijednost na koju pokazuje kurzor. Ovo je neophodno, da bi se prilikom modifikovanja tabele referencirala tačno određena vrsta. Obratiti pažnju da zbog jedinstvenosti broja indeksa svakog studenta kurzor uvijek referencira na rezultat sastavljen od jedne vrste (koja se odnosi na

studenta sa datim brojem indeksa). Zato nema potrebe za petljama. Svaka promjena vrši se za vrstu tabele BPISPIT vezanu za kurzor, pa se koristi WHERE CURRENT OF u kombinaciji sa UPDATE, koji sprječava da promjene u tabeli budu mimo vrste na koju se kruzor odnosi. Da ne stoje ove ključne riječi, modifikovala bi se čitava tabela, dakle, za svakog studenta. Posljednji UPDATE računa ukupan broj poena za datog studenta.

b) Funkcija može biti napravljena na sljedeći način:

```
create or replace function isp(ind varchar2) return varchar2 is
cursor c is select * from bpispit where indeks=ind for update of
ukupno;
ii bpispit%rowtype;
begin
      open c;
      fetch c into ii;
      if ii.i=0 or ii.i is null then
            return 'NIJE IZASAO';
      else
      update bpispit set ukupno=ii.k1+ii.k2+ii.i+ii.a where current of
c; /*staviti prije return!*/
            return 'IZASAO';
      end if;
close c;
   end;
```

Ključna riječ RETURN označava izlazne argumente, i kao i u slučaju ulaznih argumenata, neophodno je zadati njihov tip (bez dužine). Sada provjerimo da li radi kreirana funkcija:

```
begin
      declare
      pom varchar2(10);
      begin
            pom:=isp('1/07');
            dbms_output.put_line(pom);
      end;
end;
```

U promjenljivu POM upisuje se rezultat procedure za studenta sa datim indeksom, i onda se on ispisuje na ekranu.

c) Procedura za davanje ocjena može se realizovati na sljedeći način:

```
create or replace procedure ocijeni is
cursor occ is select indeks, ukupno from  bpispit for update of
ocjena, ukupno;
begin
   for zap in occ loop
      if isp(zap.indeks)='IZASAO' then
            if zap.ukupno>=50 and zap.ukupno<60 then
            update bpispit set ocjena='E' where current of occ;
            elsif zap.ukupno>=60 and zap.ukupno<70 then
            update bpispit set ocjena='D' where current of occ;
            elsif zap.ukupno>=70 and zap.ukupno<80 then
            update bpispit set ocjena='C' where current of occ;
```

```
            elsif zap.ukupno>=80 and zap.ukupno<90 then
            update bpispit set ocjena='B' where current of occ;
            elsif zap.ukupno>=90 then
            update bpispit set ocjena='A' where current of occ;
            else
            update bpispit set ocjena='F' where current of occ;
            end if;
      elsif isp(zap.indeks)='NIJE IZASAO' then
      update bpispit set ocjena='N' where current of occ;
      end if;
   end loop;
end;
```

Procedura nema ulaznih argumenata. Za razliku od nekih programskih jezika, ovdje se ne stavljaju prazne zagrade u slučaju nepostojanja argumenata procedure. Obratiti pažnju da je poziv funkcije identičan kao u bilo kojem drugom programskom jeziku. Provjerite da li je tabela BPISPIT pravilno ažurirana. Ova procedura se jednostavno poziva:

```
begin
      ocijeni;
end;
```

3. Procedura može biti realizovana na sljedeći način:

```
create or replace procedure PROC_PLATE(SS number,PR number,STR
varchar2 default 'povecaj')IS
cursor c1 is select salary from preduzece where manager_id=SS for
update of salary;
plata number;
begin
      open c1;
      loop
            fetch c1 into plata;
            exit when c1%NOTFOUND;
            if upper(STR)='POVECAJ' then
                  update preduzece set salary=plata+plata*PR/100 where
            current of c1;
      elsif upper(STR)='SMANJI' then
            update preduzece set salary=plata-plata*PR/100 where
      current of c1;
      end if;
      end loop;
      close c1;
      commit;
end;
```

Ključna riječ **default** označava podrazumijevanu vrijednost parametra, u slučaju kada se procedura pozove bez tog parametra. Pri rješavanju ovog zadatka korišćena je petlja **loop**, koja se bezuslovno prekida korišćenjem naredbe **exit**. Petlja će biti prekinuta kada pridruživanje varijabli **plata** primjenom **fetch** naredbe bude bezuspješno. To se detektuje predikatom `%NOTFOUND`, koji vraća vrijednost TRUE kada je pridruživanje vrijednosti iz kurzora varijabli

bezuspješno. Kada ovaj predikat ima vrijednost TRUE izvršava se naredba **exit** i prekida se petlja **loop**. Naredba **commit** u Oracle Express-u nije neophodna na ovom mjestu.

4. U ovom zadatku pojavljuje se primjer izuzetka. Stoga će prije izlaganja rješenja biti izloženi osnovni koncepti obrade izuzetaka u PL/SQL-u.

Svaka greška prilikom izvršavanja PL/SQL bloka izaziva izuzetak. Mogu se razlikovati dvije vrste izuzetaka:

1. sistemski definisani izuzeci
2. korisnički definisani izuzeci, koji moraju biti deklarisani u deklaracionom dijelu bloka u kojem se koriste

Sistemski definisani izuzeci se sami podižu u slučaju javljanja odgovarajuće greške. Korisnički definisani izuzeci podižu se korišćenjem:

raise <naziv izuzetka>

U bloku naredbi iza ključne riječi **exeption**, koja se stavlja na kraju bloka, slijede korisnički definisane naredbe za obradu izuzetaka. Implementacija naredbi za obradu izuzetaka radi se na sljedeći način:

when <naziv izuzetka> **then** <sekvenca naredbi>;

Nabrojimo neke od poznatih sistemskih izuzetaka:

Ime izuzetka	Broj (šifra)	Komentar
CURSOR ALREADY OPEN	ORA-06511	You have tried to open a cursor which is already open
INVALID CURSOR	ORA-01001	Invalid cursor operation such as fetching from a closed cursor
NO DATA FOUND	ORA-01403	A select . . . into or fetch statement returned no tuple
TOO MANY ROWS	ORA-01422	A select . . . into statement returned more than one tuple
ZERO DIVIDE	ORA-01476	You have tried to divide a number by 0

Posljednja **when** klauzula u dijelu bloka za implementaciju obrade izuzetaka može sadržati naziv izuzetka **others**, koja se odnosi na podrazumijevani način obrade izuzetka (može se staviti, na primjer, **rollback**).

Obrada izuzetka može se izvršiti tako da se ispiše i odgovarajuće obavještenje. Za ovu namjenu koristi se procedura **raise_application_error**, koja ima dva argumenta: broj greške (iz opsega od -20000 do -20999), i poruku greške, koja može biti maksimalne dužine 2048 karaktera. Ovdje postoji mogućnost korišćenja operatora za konkatenaciju stringova.

Procedura iz zadatka može biti realizovana na sljedeći način:

```
create or replace procedure proc_plate2(ime varchar2,SO number, pr
number, STR varchar2 default 'POVECAJ') is
izuzetak exception;
```

```
cursor c1 is select salary from preduzece p where manager_id =(select
employee_id from preduzece p1 where upper(p1.first_name||'
'||p1.last_name)like upper('%'||ime||'%') and
p.manager_id=p1.employee_id) and department_id=so for update of
salary;
   plata number;
   begin
     open c1;
     loop
          fetch c1 into plata;
          exit when c1%notfound;
        if upper(STR)='POVECAJ' then
          update preduzece set salary=plata+plata*PR/100 where
          current of c1;
        elsif upper(STR)='SMANJI' then
          update preduzece set salary=plata-plata*PR/100 where
          current of c1;
          else raise izuzetak;
          end if;
     end loop;
     close c1;
     commit;
     exception
          when izuzetak then raise_application_error(-20015,'Nijeste
          unijeli pravilan parametar za modifikaciju plate.
          Vrijednosti parametra mogu biti POVECAJ i SMANJI');
          rollback;
     end;
```

Prvo skrenimo pažnju na SQL upit za koji se vezuje kurzor. U pitanju je korelisani upit jer se tabela PREDUZECE vezuje sama sa sobom. U podupitu koji služi za određivanje MANAGER_ID-a potrebno je izvršiti provjeru imena, gdje je u dijelu `upper('%'||ime||'%')` izvršena konkatenacija džokera '%' sa imenom varijable da bi se spriječilo da ime varijable IME bude protumačeno kao string. Džokeri ostavljaju mogućnost da korisnik zada proizvoljan string koji može biti sadržan u imenu i/ili prezimenu rukovodioca. Pošto se korelisani upiti izvršavaju tako što se za spoljašnju fiksiranu vrstu (fiksiranu vrstu tabele sa aliasom P) fiksira i MANAGER_ID, uslov jednakosti `p.manager_id = p1.employee_id` omogućava da se za podupit odredi EMPLOYEE_ID onog zaposlenog iz tabele sa aliasom P1 koji je rukovodilac zaposlenog iz tabele sa aliasom P (za koji smo rekli da je trenutno fiksiran). Da nema dodatnih uslova u podupitu, za svaki red tabele P koji predstavlja zaposlenog koji ima rukovodioca našao bi se po jedan EMPLOYEE_ID tog njegovog rukovodioca. Fiksiranjem imena rukovodioca ova mogućnost se isključuje. Dodatno, uslov za broj odjeljenja u spoljašnjem upitu sužava pretragu spoljašnje tabele sa aliasom P, tako da će pozivom procedure biti modifikovane plate onih zaposlenih koji rade u odjeljenju sa zadatom šifrom, i sa kojima u tom istom odjeljenju radi njihov rukovodilac, sa zadatim imenom.

Dio `izuzetak exception` predstavlja deklaraciju izuzetka. Na mjestu `raise izuzetak` se poziva izuzetak. Na mjestu `exception` iza ključne riječi se vrši obrada izuzetka. Kad se desi izuzetak, odnosno kada se ne zada pravilno parametar STR objaviće se

poruka o greški koja je kao argument-string zadata u funkciji `raise_application_error`. Pogledati objašnjenje koje je ranije dato za broj greške.

Provjerite funkcionalnost navedene procedure za sve slučajeve. Kao primjer možete koristiti rukovodioca koji se zove 'Alexander' i koji radi u odjeljenju sa šifrom 60.

5. Procedura može biti implementirana na sljedeći način:

```
create or replace procedure CASOVI_P(p number) is
cursor sl1 is select n.nid sn,n.ime ime, count(*) brp,
sum(o.cp+o.cv*o.gv+o.cl*o.gl) casovi from nastavnici n, opt o where
o.nid=n.nid group by n.nid,ime;
cursor sl2 is select n.nid sn,n.ime ime,count(*) brp,
max(o.cp+o.cv*o.gv+o.cl*o.gl) casovi from nastavnici n, opt o where
o.nid=n.nid group by n.nid,ime;
begin
   delete from casovi;
   if p=1 then
     for vr1 in sl1 loop
          insert into casovi
          values(vr1.sn,vr1.ime,vr1.brp,vr1.casovi);
     end loop;
   else
     for vr2 in sl2 loop
          insert into casovi
          values(vr2.sn,vr2.ime,vr2.brp,vr2.casovi);
     end loop;
   end if;
end;
```

6. Pomoć: pogledati na koji način se rezultat SELECT-a može smjestiti u varijablu.

Literatura:

[1] Oracle/SQL Tutorial – Michael Gertz, *University of California*, 2000

[2] https://docs.oracle.com/cd/E11882_01/appdev.112/e25519.pdf (Oracle Database PL/SQL Language Reference, *poslednji put pristupano 12. novembra 2014. godine*)